INTEGRATION OF ANALYSIS TECHNIQUES IN SECURITY AND FAULT-TOLERANCE

Gabriele Lenzini

Members of the dissertation committee:

prof. dr. P. H. Hartel	University of Twente, Enschede (promotor)
prof. dr. H. Brinksma	University of Twente, Enschede (promotor)
dr. S. Etalle	University of Twente, Enschede (assistant promotor)
dott. S. Gnesi	Istituto di Scienza e Tecnologie dell'Informazione CNR, Pisa, Italy
prof. dr. R. J. Wieringa	University of Twente, Enschede
prof. dr. W. J. Fokkink	Vrije Universiteit of Amsterdam, Amsterdam
prof. dr. J.-F. Raskin	Université Libre de Brussels, Brussels, Belgium

Distributed and Embedded Systems research group, Department of Computer Science and the Centre for Telematics and Information Technology, P.O. Box 217, 7500 AE Enschede, The Netherlands.

This thesis in included in the Centre for Telematics and Information Technology (CTIT) Ph.D. Series, University of Twente, Enschede, The Netherlands

The work in this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics), Eindhoven University of Technology, Department of Mathematics and Computing Science, Eindhoven, The Netherlands.

This research is partially supported by PAW: Privacy in an Ambient World a TUD/DIES/KUN/TNO-EIB/TNO-FEL collaboration, funded by IOP GenCom under project nr. IGC03001.

This research is partially supported by Istituto di Informatica e Telematica (IIT-CNR) Area della Ricerca CNR, Via G. Moruzzi, 1 – 56124 Pisa (ITALY)

This research is partially supported by the Istituto di Scienza e Tecnologie dell'Informazione (ISTI-CNR) Area della Ricerca CNR, Via G. Moruzzi, 1 – 56124 Pisa (ITALY)

This thesis was edited with Vim and typeset with LaTeX.
Keywords: formal methods, security protocol analysis, fault-tolerance, formal models

Printed by Ipskamp PrintPartners, Enschede, The Netherlands
CTIT Ph. D. Thesis Series Number: 05-70 — ISSN 1381-3617
IPA Dissertation Series Number: 2005-07
ISBN 90-365-2200-5

INTEGRATION OF ANALYSIS TECHNIQUES IN SECURITY AND FAULT-TOLERANCE

DISSERTATION

to obtain the doctor's degree at the University of Twente,
on the authority of the rector magnificus,
prof. dr. W. H. M. Zijm,
on account of the decision of the graduation committee,
to be publicly defended
on Thursday, 30th June 2005, at 16.45

by

Gabriele Lenzini
born on 24 August 1968
in Livorno, Italy

This dissertation is approved by:

Prof. Dr. Pieter H. Hartel (promotor)
Prof. Dr. Ed Brinksma (promotor)
Dr. Sandro Etalle (assistant promotor)

To my beloved parents, Rosanna and Umberto,
to my dear “little” sister Ilaria.

In memory of my grandmother “nonna” Gina.

Abstract

In the last decade security related problems have attracted the attention of many researchers from different areas, especially from the formal methods field. The massive research that grows up around a new field is usually guided by the techniques and experiences specific to who is facing the problem. This happened, for example, in cryptographic protocol analysis.

Once the surge of interest around a specific problem subsides, it is time to look for an integration among the different proposed approaches. In our opinion the quest for integration has a doubly beneficial effect. Firstly, it allows to reuse knowledge, experiences and tools from different areas of research in computer science. Secondly, it allows to understand the intrinsic difficulties standing at the basis of a certain problem, independently of the formal approach initially used to face it. For example, security properties like secrecy and authenticity are nowadays considered equally difficult to prove; this similarity was not clear at the beginning, when the different formalizations for secrecy and authenticity seemed not to be directly comparable.

This thesis focuses on the study of integration of formal methodologies in security protocol analysis and fault-tolerance analysis. The research is developed in two different directions: interdisciplinary and intra-disciplinary. In the former, we look for a beneficial interaction between strategies of analysis in security protocols and fault-tolerance; in the latter, we search for connections among different approaches of analysis within the security area. In the following we summarize the main results of the research.

Interdisciplinary Perspective. In this perspective, we recognize in Model Checking [48], in Partial Model Checking [16], and in Non Interference Analysis [101] those methodologies that can be applied profitably in security protocol analysis and fault-tolerance analysis. This division of research is articulated in the following objectives:

$\surd$ **Objective 1** *To show how model checking can be applied to the validation of both security protocols and fault-tolerant systems.*

Model Checking has been widely used to validate fault-tolerant, safety-critical, systems. We report on an industrial experience of validation in this field. Through another industrial experience of validation of an authentication protocol we show that model checking is a valid strategy for the analysis of security protocols as well. When studying model checking for security protocol analysis, we also develop an original logic-based, on-the-fly, model checker for the analysis of security cryptographic protocols.

$\surd$ **Objective 2** *To show how a precise specification framework allows security protocol engineering and fault-tolerance engineering to share common strategies and tools of analysis. The framework requires the neat separation of the system model from its malicious environment.*

The specification framework is proposed and discussed prevalently in the framework of the CCS [158] process algebra. The common analysis strategies we identify originate in partial model checking and non interference analysis. Such strategies have been proposed and studied prevalently in the verification of security protocols, and they have made it possible to analyze the behavior of a protocol acting in an open, malicious, environment. By partial model checking, the problem of checking a security property, expressed as a μ-calculus logic formula over a CCS protocol model, can be reduced to a validation problem in the μ-calculus. By a non interference approach, many security properties can be formalized and checked by existing tools.

When applying partial model checking to fault-tolerance we identify a subset of the μ-calculus, whose validation problem can be solved in time linear in the size of the formula. We provide examples of safety and liveness properties that can be expressed in the sub-calculus identified. When applying non interference ideas we show how fault-tolerance can be reformulated in the context of the Generalized Non Deducibility on Composition (GNDC) scheme of analysis; we show also how GNDC analysis strategies and existing tools can be used in fault-tolerance.

Intra-disciplinary Perspective. In this outlook we relate the use of different formal models in security. The formal models we consider are: process algebra, automata, and multiset rewriting. The division of our study, conducted in a scenario of analysis of security protocols, is organized in the following objectives:

$\surd$ **Objective 3** *To relate two famous formalisms used in the analysis of security, process algebras and multiset rewriting, in the framework of cryptographic and authentication protocol analysis.*

Actually, with "process algebra" we denote a family of calculi which have been proposed for describing features of distributed and concurrent systems. Here, "multiset rewriting", which has roots in concurrency theory and rewriting logic, denotes a language used to study fundamental issues in authentication protocols. We define special encodings between the two formalisms which preserve a bisimulation-like equivalence, and consequently secrecy and authentication properties.

$\surd$ **Objective 4** *To redefine the GNDC theory in terms of Team Automata.*

Automata-based formalisms have been widely used in the analysis of fault-tolerant systems. Recently, Team Automata have been proposed to specify computer supported cooperative work and concurrent systems, but they still miss an analysis framework. By proposing a GNDC theory in terms of Team Automata, we allow the migration of some of the theory for security analysis from process algebra to the automata world. We show how to apply our framework to study an integrity property over a multicast cryptographic protocol.

Samenvatting

Problemen gerelateerd aan veiligheid (*security*) hebben in het afgelopen decennium de aandacht gekregen van vele onderzoekers uit verschillende gebieden, in het bijzonder uit de formele methoden. De hoeveelheid onderzoek die tot stand komt rondom een nieuw gebied wordt meestal gestuurd door de specifieke technieken en ervaringen van de persoon die het probleem aanpakt.

Zodra den verhoogde interesse rondom een specifiek probleem afneemt, is het tijd om op zoek te gaan naar integratie van de verschillende invalshoeken die zijn voorgesteld. Wij zijn van mening dat de zoektocht naar integratie een dubbel voordelig effect heeft. Ten eerste staat de integratie het hergebruik van kennis, ervaringen en gereedschappen uit verschillende onderzoeksgebieden binnen de informatica toe. Ten tweede helpt de integratie de intrinsieke moeilijkheden te begrijpen die aan de basis staan van een zeker probleem. Zo worden de veiligheidseigenschappen *secrecy* en *authenticity* bijvoorbeeld tegenwoordig beschouwd als even moeilijk te bewijzen; deze gelijkheid was oorspronkelijk niet zo duidelijk, toen de verschillende formalisaties voor *secrecy* en *authenticity* niet direct vergelijkbaar leken.

De nadruk in dit proefschrift ligt op het bestuderen van de integratie van formele methodologieën binnen *fault tolerance* en *security protocol analysis*. Het onderzoek vindt in twee verschillende richtingen plaats: interdisciplinair en intradisciplinair. In de eerstgenoemde richting zoeken we naar een voordelige interactie tussen analysestrategieën in *fault tolerance* en in veiligheid; in de laatstgenoemde richting zoeken we naar connecties tussen de verschillende manieren van analyse binnen het gebied van veiligheid. In wat volgt vatten we de belangrijkste uitkomsten van dit onderzoek samen.

Interdisciplinair Perspectief. In dit perspectief herkennen we in *Model Checking* [48], in *Partial Model Checking* [16] en in *Non-Interference Analysis* [101] de methodologieën die op een voordelige manier kunnen worden toegepast in *fault tolerance* en *security protocol analysis*. Deze verdeling van onderzoek komt in de volgende doelstellingen naar voren:

$\surd$ **Doelstelling 1** *Laten zien hoe* model checking *kan worden toegepast om zowel* fault tolerant *systemen als veiligheidsprotocollen te valideren.*

Model checking wordt vaak gebruikt om *fault-tolerant, safety-critical* systemen te valideren. Wij rapporteren over een industriële ervaring met validatie in dit onderzoeksveld. Middels een andere industriële ervaring met validatie van een authenticatieprotocol laten we zien dat ook *model checking* een valide strategie is voor het analyseren van complexe authenticatieprotocollen. Terwijl we *model checking* voor *security protocol analysis* bestuderen, ontwikkelen we ook een originele,

op logica gebaseerde, *on-the-fly model checker* voor de analyse van cryptografische veiligheidsprotocollen.

$\surd$ **Doelstelling 2** *Laten zien hoe een precies specificatiekader* fault tolerance *en* security protocol engineering *toestaat om gemeenschappelijke strategieën en gereedschappen voor analyse te delen. Het kader vereist een duidelijke afscheiding tussen het systeemmodel en haar boosaardige omgeving.*

Het specificatiekader wordt voornamelijk binnen de context van de procesalgebra CCS [158] voorgesteld en bediscussieerd. De gemeenschappelijke analysestrategieën die wij identificeren, stammen uit *Partial Model Checking* en *Non-Interference analysis*. Zulke strategieën zijn voornamelijk voor de verificatie van veiligheidsprotocollen voorgesteld en bestudeerd; zij maken het bijvoorbeeld mogelijk om het gedrag van een protocol te analyseren dat in een open, boosaardige omgeving handelt. Door middel van *partial model checking* kan het probleem om een veligheidsprotocol, uitgedrukt in de μ-calculus als een logische formule over een CCS protocolmodel, te verifiëren, gereduceerd worden tot een validatieprobleem in de μ-calculus. Middels een *non-interference* aanpak kunnen vele veiligheidseigenschappen door bestaande gereedschappen geformaliseerd en geverifieerd worden.

Bij het toepassen van *partial model checking* op *fault tolerance* identificeren we een deelklasse van de μ-calculus waarvan verificatieprobleem kan worden opgelost in lineaire tijd, afhankelijk van de lengte van de formule. We geven voorbeelden van *safety* en *liveness* eigenschappen die uitgedrukt kunnen worden in de geïdentificeerde subcalculus. Bij het toepassen van *non-interference* ideeën laten we zien hoe *fault tolerance* herformuleerd kan worden in de context van het *Generalized Non Deducibility on Composition (GNDC)* analyseschema; we laten ook zien hoe *GNDC* analysestrategieën en bestaande gereedschappen kunnen worden hergebruikt in *fault tolerance*.

Intradisciplinair Perspectief. In dit perspectief relateren we het gebruik van verschillende formele modellen in veiligheid. De formele modellen die we beschouwen zijn: procesalgebra, automaten en *multiset* herschrijven. Onze studie, uitgevoerd als een scenario voor de analyse van veiligheidsprotocollen, is als volgt georganiseerd:

$\surd$ **Doelstelling 3** *Het relateren van twee standaard formalismen die gebruikt worden voor veiligheidsanalyse, procesalgebra's en* multiset *herschrijven, binnen het kader van cryptografische en authenticatie protocolanalyse.*

Met "procesalgebra" duiden we eigenlijk een familie van calculi aan die zijn voorgesteld om eigenschappen van gedistribueerde en concurrente systemen mee te beschrijven. Hier duidt "*multiset* herschrijven", wat wortelt in de theorie van *concurrency* en herschrijflogica, een taal aan die gebruikt wordt om fundamentele noties in authenticatieprotocollen mee te bestuderen. Wij definiëren speciale coderingen van de twee formalismen die een bisimulatie-achtige equivalentie vertonen, en vervolgens *secrecy*- en *authenticatieeigenschappen*.

$\surd$ **Doelstelling 4** *Het herdefiniëren van de* GNDC *theorie in termen van teamautomaten.*

Formalismen gebaseerd op automaten worden vaak gebruikt voor de analyse van *fault tolerant* systemen. Teamautomaten zijn recentelijk voorgesteld voor de specificatie van noties voor *computer supported cooperative work* en concurrente systemen, maar zij missen nog een analysekader.

Door een *GNDC* theorie in termen van teamautomaten voor te stellen, staan we de migratie van een gedeelte van de theorie voor veiligheidsanalyse uit procesalgebra naar de wereld van automaten toe. We laten zien hoe ons kader kan worden toegepast om een integriteitseigenschap over een *multicast* cryptografisch protocol te bestuderen.

Acknowledgments

This work spreads over a large part of my recent, often messy, life. Remembering all the people that are connected with this work is definitely a hard task. Most of them are friends or colleagues with whom I have been working along these years. With all these people I have shared a significant part of my life; basically, my thoughts, my worries, my enthusiasms, my doubts, and my "time" ... a great amount of TIME.

This thesis would not has been possible (morally and physically) without the constant support of Stefania Gnesi of the Formal Methods && Tools group of the ISTI-CNR, where I have been working for many years; Ed Brinksma, Sandro Etalle, and Pieter Hartel my tireless promoters in the Netherlands; Fabio Martinelli and Anna Vaccarelli of the Security group of the IIT-CNR, where I am currently working.

I would like also to thank Marlous, the DIES group secretary, whose organizational skills have made my (indeed everyone's) life much easier since my first "a.u.b." in the Netherlands[1]. Thanks also to Ruth, my English teacher, and to Jan Cederquist who reviewed a chapter of this thesis. I thank the best amusing co-authors of mine Marinella Petrocchi and Maurice ter Beek. We started for fun around a table with "cappuccino e brioche" and we have finished for formulating all those incomprehensible mathematics about team automata. Marinella and Maurice are also great friends of mine that I must thank in a special manner: "Grazie Mari e Mau per l'affetto sincero che mi avete sempre mostrato".

I also thank Stefano Bistarelli, Iliano Cervesato, and Fabio Martinelli. They are "responsible" for the heavy mathematics in another chapter of this thesis.

I thank Sandro Etalle, my mentor and supporter in the Netherlands. Since the beginning he has listened to all my complaints about living abroad; sometimes, wisely, he has also ignored them. He has taught me a lot of things: "how to work (and live!) in an organized way", "how to write (and live!) with self-confidence", or "how to/not to write a paper". I know it has been hard work for him, and especially reading this part you can understand how easy it is for me to break practically all the good advices that Sandro gave me! "Thanks Sandro, I will remember your teachings!".

I want to say a special word of thanks to Ricardo Corin from Córdoba, Argentina. He has been the link between work and life. At work, we have been spending days talking about process algebras and strange protocols. At coffee time, we have been filling white boards, pieces of paper, or mensa napkins of strange symbols and drawings ... all potential ideas of our future works. To be sincere, we would have never brought to the end anything of those ideas because, I realized, Italians and Argentineans are disorganized exactly in the same way. "We would have never reached

[1] A.U.B., shortcut for "Alstublieft" ("Please" in English, or "S'il Vous Plait" in French, or "Prego" in Italian), is the first word that we learn in this well educated country. It is uttered practically by everyone and written everywhere.

the end" I was saying ... if Jerry den Hartog, the Dutchman had not been there. He put order in our chaos. With Jerry we have managed to build up a team that really has worked as every group should work, in "symphony".

No, sorry, you can't go ... I haven't finished yet[2]. I have to thank the other side of Ricardo, Laura Brandán Briones. Laura is Ricardo's never-missing girlfriend (or, if I am in real delay with this thesis, even his wife[3]). Ricardo-y-Laura have been my family in the Netherlands. In their house I have found warmth, help, support, hope, plenty of dinners, thousands of chats, laughs, a great amount of happiness. There has not been a single day that I have not spent with them. "Ric and Lau, mis valiosos amigos, gracias desde lo más profundo de mi corazón."

I would like to thank my old and dear friends Maurizio Tiengo and Giovanni Cerini, for their loyal friendship: "Grazie Maurizio e Giovi, per la vostra leale amicizia". Thanks to Antonio Ruzzelli and his Italian pizza parties that have always given me great fun, and to Jordan my gentle, sportive, office-mate.

I also want to remember Livorno, its wonderful sea, its wonderful sunsets, its colors, and its fresh air that have cured my spirit many times; my house in via della Pieve where I used to alternate accordeon lessons with the reading of my first book on Dutch life; Fabiano and its marbled mountains. Holland and its polite, positive, and respectful approach to life. And finally, Enschede and Rietmolenstraat 64, my present, multi-ethnic, cosy, busy and handy house.

A sincere thanks to Isabel and to her precious, lovely, sometimes harsh, irreplaceable, and constant encouragement. She has spent plenty of time trying to make me improve my character by inciting me to fight and to resist against the difficulties of the everyday life: "Gracias querida Isabel, por todo lo que haces".

A special, personal, deep thanks has to be devoted to my family: my mother Rosanna Paola, my father Umberto Giovanni Oreste, and my sister Ilaria Francesca. My family have supported me, always, every-way, and everywhere. "Mamma, Babbo e Ilaria grazie di esserci stati sempre, comunque, anche quando, mi rendo conto, non è stato facile starmi vicino".

At this point, I like to imagine that most of you are asking why, why on earth did a guy like me have to come to the Netherlands to complete his long path? I will give an answer, and I will do with the words of a song by Vinicio Capossela, an Italian contemporary poet/musician. The song is entitled *Modí*, the nickname of the most famous painter from my native city Livorno: Amedeo Modigliani.

Basically, this song says that people from Livorno (like me) need to go somewhere else, usually very far, to find what they are looking for. Moreover, to reach what for someone else is simply at distance of grasp, we also need to follow weird and conflicting roads:[4]

[..] per amore tradivi,	*[..] you betrayed for loving,*
per esister morivi,	*you died for living,*
per trovarmi fuggivi fin qua,	*to find me, you run away*
perché Livorno dà gloria	*'cause Livorno gives glory*
soltanto all'esilio	*only to the exile,*
e ai morti la celebrità.	*and celebrity to those who have passed*
(*Modí*, V. Capossela)	*away.*

[2] "*Knowing where and when to stop is also a matter of good manner*", Aristotle in Metaphysics, IV century B.C.

[3] Ricardo and Laura got married on February 12th, 2005

[4] All the translations are unofficial.

Prologue

Summary

Organization

This thesis is composed of different papers I have presented and published during my Ph.D. studies. The thesis is organized into three main parts, and a conclusive part (see Figure 1).

Part I: Formal Validation of Systems: Industrial Test Cases. This part collects my work on formal validation of industrial systems. It reports the technical details of two experiences of formal analysis. The former concerns the verification of a safety-critical fault-tolerant railway system, and the latter focuses on the analysis of security aspects of an authentication protocol used in the OSA/Parlay telecommunication network. Industrial test cases play an important role in this thesis; all the problems that I have developed here were conceived while I was working on the validation of real systems. In other words, the effort of modeling and analyzing a real system makes clear what are the real difficulties, and hence the problems to be solved to have a real impact for validation and verification.

Part II: Analysis Techniques in Security and Fault-Tolerance. This part assembles my work on the development of techniques of validation in security protocol[5] analysis and fault-tolerance analysis. In this chapter the beneficial interaction between the two disciplines emerges. First, we re-design in fault-tolerance analysis terms, techniques of validation that have been originally introduced for security analysis; non-interference and its formalization in terms of process algebras, and module checking [126] through partial model checking [16]. Second, we design and implement a logic-based model checker for security protocol analysis, whereas model checking is a traditional validation technique for the analysis of dependable systems.

Part III: Comparison of Formal Models in Security Protocol Analysis. This part gathers my work on the interaction between different methodologies of modeling and verifying in security protocol analysis. As formal models we chose process algebras (PA), multiset rewriting (MSR), and team automata [194] (TA). Generally speaking, PA denote a family of calculi which have been proposed for describing features of distributed and concurrent systems; MSR roots in concurrency theory and rewriting logic and has been incorporated into a high-level specification language for authentication protocols, the Common Authentication Protocol Specification Language (CAPSL) [64]. TA derive from automata and they have been originally used to model concurrency

[5] Security protocols are sometimes called cryptographic protocols. We will use these two terms interchangeably.

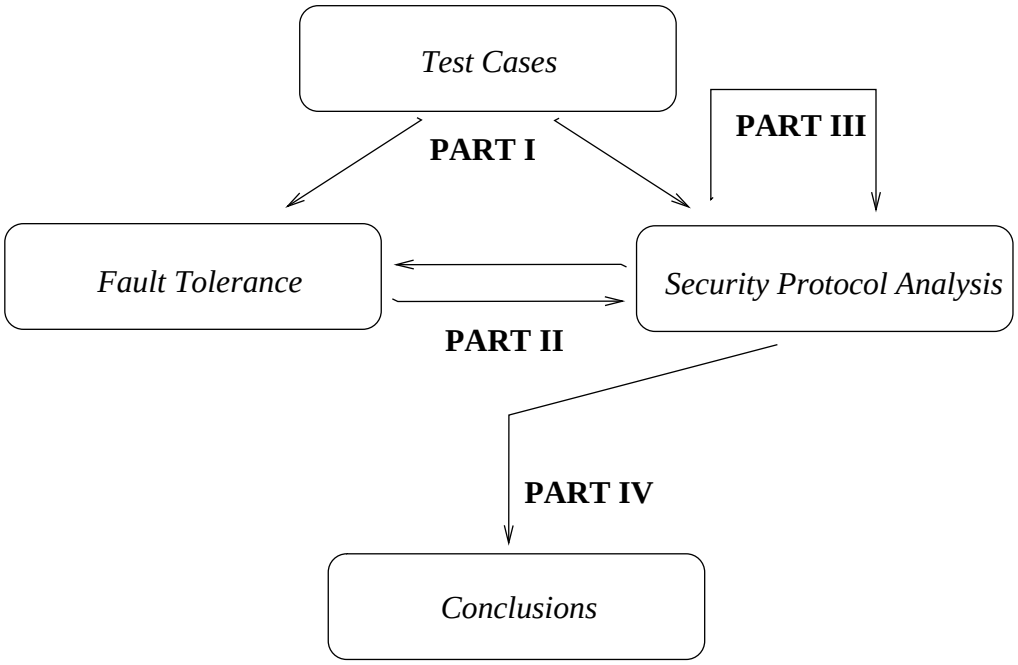


Figure 1: Organization Chart of the Thesis

and computer supported cooperative works. First we compare the expressiveness of PA and MSR in the restricted field of security protocol analysis, and later we show how TA can be furnished with a stable framework for the analysis of cryptographic protocols.

Part IV: Conclusions This part concludes the thesis. We identify general principles that emerge from the present study and from our experience of integration.

Chapter Contents

Each of the main parts is organized into two chapters. We now summarise the contents of each chapter.

Part I. This part contains Chapter 1 and Chapter 2 and it concerns the application of formal methods to fault-tolerance and security protocol analysis. Chapter 1 is based on [97, 96, 98]; it reports a complete validation exercise of a fault-tolerant railway system. Chapter 2 is based on [58, 57]; it relates on an experience of the validation of an authentication protocol that is part of a Telecom web service.

Chapter 1 describes the experiences of formal specification and validation on a railway safety-critical control system in which specific methodologies for the analysis of complex dependable systems (*e.g.,* triple modular redundancy) have been expressed. From the technical document describing the system we built a PROMELA [118] formal model. This requires the design of a formal model that unambiguously describes the system at an appropriate level of abstraction. This means first to examine a whole bunch of documents describing the system requirements and to carefully filter out the information that is not significant,

or interesting, or that lays at a level of detail that is not the one the validation refers to. The model is then analyzed through model checking [48, 49] by using the tool SPIN [118]. Model checking suffers from a well known problem: the size of the state space of the model can grow beyond the limits of the available hardware *i.e.,* exponentially in the number of its components [47]. We find the same problem with our complex model. This implies the use of abstraction and compositional strategies over the formal model, during the analysis phase.

Chapter 2 describes a validation experience that consists in modeling and verifying a real-world authentication protocol. This "Trust and Security Management" protocol, is implemented as a protocol in the Parlay/OSA Application Program Interfaces (APIs) [1]. Parlay/OSA architectures aim to stimulate third parties in developing new services exploiting mobile telecommunications resources, while allowing the network operator to maintain control over its network specially with respect to the quality of service offered and the security usage. The chapter explains in detail how a formal model has been built, starting from the UML [179] specification of the protocol, and how the experiments of verification have been performed. Moreover, it critically comments on the verification results, which point out weaknesses in the authentication procedure, and it suggests a possible solution for strengthening the security of the protocol.

Part II. This part contains Chapter 3 and Chapter 4, and it concerns the integration of techniques of analysis in fault-tolerance and security cryptographic protocol engineering. They show how the two disciplines can benefit of common strategies of analysis: model checking, techniques of analysis of non-interference, and partial model checking. Chapter 3 is made up of the articles [100, 99] and Chapter 4 is based on the papers [94, 95, 134].

Chapter 3 is theoretical. It studies how fault-tolerance analysis can benefit from techniques of analysis developed for the study of security protocols. It uses the CCS process algebra as a formal framework to model the fault-tolerant system and its (potentially malicious) environment as two separate and interacting CCS processes. The environment is able to induce the system to switch to insecure states. In this framework a system enjoys a fault-tolerance property if the systems satisfies the property despite any interaction with the environment. From the point of view of the analysis, this chapter studies the fault-tolerance of a system, with respect to a given property, when the environment is an unspecified component. In this case, the role of environment in fault-tolerance can be compared with that played by the intruder in security protocol analysis. This chapter restates in fault-tolerance two strategies of validation used in security protocols analysis. The first strategy consists in reducing the problem of checking if a property (here a μ-calculus formula) holds in our framework, to a problem of validity in the μ-calculus. We exploit partial model checking in this reduction step, and we show how the validity problem, generally EXPTIME complete, can be solved efficiently in the universal conjunctive subclass of the μ-calculus. The second strategy consists in applying the Generalized Deducibility on Compositions framework (in short, GNDC) [86] to fault-tolerance. GNDC is a uniform scheme for defining and analyzing security properties, and it originates in the field of non-interference for security analysis. This chapter shows how fault-tolerance properties can be uniformly characterized as GNDC properties, and how theoretical results (*e.g.,* compositionality), validation techniques, and tools – well established in the GNDC security analysis – can be exploited for

fault-tolerance.

Chapter 4 proposes a logic-based model checking framework [48] for the verification of security cryptographic protocols. Model checking enjoys a background of good results in dependability and fault-tolerance analysis (*e.g.*, see [19, 38]). On the contrary its use in security cryptographic protocol analysis was quite new when the papers, on which this chapter is based, were initially proposed in 2000. In the proposed model checking framework, protocols are modeled as terms of a process algebra which is inspired by the Abadi and Gordon spi-calculus [6]. Security properties, such as secrecy and authenticity, are formalized using linear time temporal logic.

Part III. This part contains Chapter 5 and Chapter 6 and it concerns the integration of analysis techniques within the field of security protocol analysis. Chapter 5 is based on [26, 24, 25] and Chapter 6 is based on [197, 199, 198].

Chapter 5 develops a comparison between process algebras and multiset rewriting when applied to the analysis of security cryptographic protocols. We compare an instance of process algebra (called PA_P) and an instance of multiset rewriting (called MSR_P) which are expressive when used to describe security protocols. Special *encodings* from one formalism to the other allow secrecy and authenticity properties to be preserved.

Chapter 6 starts from the fact that team automata (TA) are an emerging model for the formalization of cooperative network systems and recently of multicast/broadcast protocols. In fact, TA theory extends the classical I/O automata theory by allowing the definition of different parallel composition operators, that make it possible to formalize complex interactions. This last feature makes TA an interesting model for the analysis of security protocols even though TA lack a well-established analysis framework. The present chapter describes how to model an insecure scenario for cryptographic multicast/broadcast protocols in terms of TA and it proposes also the definition of GNDC theory for TA. Moreover it shows how, once established the GNDC framework in terms of TA, it is possible to reuse part of the analysis theory developed for process algebra in the automata world so that integrity properties can be proved.

Part IV. This part is composed of Chapter 7. It synthesizes general principles representing the conclusive remarks of the thesis.

Publications

To conclude this section we list our refereed conference and workshop publications that constitute the backbone of the parts of this thesis.

Publications in Part I

- GNESI, S., LATELLA, D., LENZINI, G., AMENDOLA, A., ABBANEO, C., AND MARMO, P. An Automatic SPIN Validation of a Safety Critical Railway Control System. In *Proc. of the International Conference on Dependable Systems and Networks (DSN-2000) – formerly FTCS-30 and DCCA-8 – June 25-28, 2000, New York, NY, USA*, pp. 119–124, IEEE Computer Society, 2000.

- GNESI, S., LATELLA, D., LENZINI, G., AMENDOLA, A., ABBANEO, C., AND MARMO, P. A Formal Specification and Validation of a Control System in Presence of Byzantine Errors. In *Proc. of the 6th International Conference Tool and Algorithms for the Construction and Analysis of Systems (TACAS 2000) March 25 - April 2, 2000, Berlin, Germany*, no. 1785 of LNCS, pp. 535–549, Springer-Verlag, 2000.

- CORIN, R., DI CAPRIO, G., ETALLE, S., GNESI, S., LENZINI, G., AND MOISO, C. A Formal Security Analysis of an OSA/Parlay Authentication Interface. In *Proc. of the 7th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'05), June 15-17 , 2005, Athens, Greece* no. 3535 of LNCS, pp. 131–146, Springer-Verlag, 2005.

Publications in Part II

- GNESI, S., LENZINI, G., AND MARTINELLI, F. Logical Characterization and Analysis of Fault Tolerant Systems Through Partial Model Checking. In *Proc. of the ICLP'03 International Workshop on Software Verification and Validation (SVV 2003), December 14, 2003, Mumbai, India* vol. 118 of *ENTCS*, Elsevier, pp. 57–70, 2005.

- GNESI, S., LENZINI, G., AND MARTINELLI, F. Applying Generalized non Deducibility on Compositions (GNDC) Approach in Dependability. In *Proc. of the Formal Methods for Security and Time (MEFISTO Project), August 6, 2003, Pisa, Italy* (2004), vol. 99 of *ENTCS*, pp. 111–126, Elsevier Science, 2003.

- LENZINI G, GNESI, S., LATELLA, D. Spider: a Security Model Checker. In *Proc. of the 1st International Workshop on Formal Aspect of Security and Trust (FAST 2003), Sept. 8-14, 2003, Pisa, Italy* (2003), F. Martinelli and T. Dimitrakos, Eds., Istituto di Informatica e Telematica (IIT-CNR), Pisa, Italy, pp. 163–180. Technical Report ITT-CNR-10/2003

- GNESI, S., LATELLA, D., AND LENZINI, G. A BRUTUS Model Checking for a Dialect of Spi-Calculus (Extended Abstract). In *Proc. of the Workshop on Formal Methods and Computer Security (FMCS 2000), July 15-19, 2000, Chicago IL, USA* (2000), Informal Proceedings

- GNESI, S., LATELLA, D., AND LENZINI, G. A BRUTUS Logic for a Spi-Calculus. In *Proc. of the ACM SIGPLAN Workshop on Issues in the Theory of Security (WITS'02), January 14-15, 2002, Portland, OR, USA* (2002), J. Guttman, Ed., Informal Proceedings.

Publications in Part III

- BISTARELLI, S., CERVESATO, I., LENZINI, G., AND MARTINELLI, F. Relating Multiset Rewriting and Process Algebras for Security Protocol Analysis. R. Gorrieri, Ed, vol. 13, n. 1 of *Journal of Computer Security*, pp. 3-47, IOS Press, 2005.

- BISTARELLI, S., CERVESATO, I., LENZINI, G., AND MARTINELLI, F. Relating Process Algebras and Multiset Rewriting for Immediate Decryption Protocols. In *Proc. of the Second International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security (MMM-ACNS 2003), Sept. 20-24, 2003, St. Petersburg, Russia*, V. A. S. V. I. Gorodetski and L. J. Popyack, Eds., vol. 2776 of *LNCS/LNAI*, pp. 86–99, Springer-Verlag, 2003.

- TER BEEK, M. H., LENZINI, G., AND PETROCCHI, M. Team Automata for Security Analysis of Multicast/Broadcast Communication. In *Presented at the Workshop on Issues in Security and Petri Nets (WISP), June 23, 2003, Eindhoven, The Netherland* (2003), R. G. N. Busi and F. Martinelli, Eds., Informal Proceedings, pp. 57–72.

- TER BEEK, M. H., LENZINI, G., AND PETROCCHI, M. Team Automata for Security (a Survey). In *Proc. of the 2nd International Conference on Security Issues in Coordination Models, Languages, and Systems (SecCo 2004), affiliated to CONCUR 2004, 30th Aug., 2004, London, United Kingdom*, R. Focardi and G. Zavattaro Eds, vol. 128, n. 5 of *ENTCS*, pp. 105–119, Elsevier Science, 2005.

- TER BEEK, M. H., LENZINI, G., AND PETROCCHI, M. Team Automata for Security Analysis. TR-CTIT 04-1, Centre for Telematics and Information Technology (CTIT), Univ. Twente, Enschede, The Netherlands, 2004.

Contents

I Formal Validation of Systems: Industrial Test Cases **1**

1 Validation of Fault-Tolerance Systems: a Test Case **3**
1.1 Introduction . . . 3
1.1.1 Linear Time Logic . . . 5
1.1.2 PROMELA and SPIN . . . 6
1.2 System Description . . . 8
1.3 Formal Models . . . 9
1.3.1 Formalization of Faults . . . 10
1.4 The TMR model . . . 15
1.4.1 Formal Verification of TMR . . . 19
1.4.2 Discussion . . . 21
1.5 The TMR-PCU model . . . 21
1.5.1 Formal Verification on TMR-PCU . . . 25
1.5.2 Discussion . . . 26
1.6 Abstraction and Implementation Strategies . . . 26
1.7 Conclusions . . . 28

2 Validation of Security Protocols: a Test Case **31**
2.1 Introduction . . . 31
2.2 The OSA/Parlay Architecture . . . 33
2.2.1 Trust and Security Management protocol . . . 34
2.3 Formal Security Analysis . . . 35
2.3.1 Modeling Choices . . . 35
2.3.2 Formal Models . . . 37
2.3.3 Formal Analysis and Detected Weakness . . . 38
2.4 Discussion . . . 42
2.5 Conclusions . . . 43

II Analysis Techniques in Security and Fault-Tolerance **45**

3 Techniques of Security Protocol Analysis in Fault-Tolerance **47**
3.1 Related Work . . . 47
3.2 Introduction . . . 48

3.3 CCS Background . . . 50
3.4 Modeling Fault-Tolerant Systems . . . 51
3.4.1 Our Scenario for Fault-Tolerance Analysis . . . 53
3.5 Background on Logic and Properties of Processes . . . 55
3.5.1 Modal μ-calculus . . . 55
3.5.2 Equational μ-calculus . . . 56
3.5.3 Observable μ-calculus Properties over Processes . . . 57
3.5.4 Partial Model Checking . . . 57
3.6 Logic Characterization and Analysis of Fault-Tolerance . . . 58
3.6.1 The Problem . . . 58
3.6.2 Improving the Time Complexity of the Analysis . . . 60
3.6.3 $\forall_\wedge MC$ Formulas in Fault-Tolerance . . . 62
3.6.4 Our Running Example . . . 68
3.7 Background on Observational Properties . . . 70
3.7.1 Observational Equivalences among Processes . . . 70
3.7.2 Information Flow and Non Interference Properties . . . 71
3.8 GNDC Characterization and Analysis of Fault-Tolerant . . . 73
3.8.1 Fault-Tolerance Properties as Instances of GNDC . . . 74
3.8.2 Other Observational Relations in GNDC for Fault-Tolerance . . . 77
3.8.3 Compositional Analysis of Fault-Tolerance . . . 79
3.9 Conclusions . . . 80

4 SPYDER: a Model Checker for Security Protocols 83
4.1 Introduction . . . 83
4.2 The *spy-calculus* . . . 86
4.2.1 Syntax . . . 86
4.2.2 Semantics . . . 88
4.3 A Logic for Security Properties . . . 92
4.4 Typed *spy-calculus* . . . 96
4.4.1 Building Typed Protocols . . . 98
4.5 Towards Finite Model Checking . . . 100
4.5.1 Model Checking the *spy-calculus* . . . 105
4.6 Conclusions . . . 108

III Comparison of Formal Models in Security Protocol Analysis 113

5 Relating Multiset Rewriting and Process Algebra in Security 115
5.1 Introduction . . . 115
5.2 Background . . . 117
5.2.1 First Order Multiset Rewriting . . . 117
5.2.2 Process Algebras . . . 118
5.3 Security Protocols . . . 119
5.3.1 Formalizing Protocols as Multiset Rewriting . . . 119
5.3.2 Protocols as Processes . . . 122
5.4 Encoding Protocol Specifications . . . 125
5.4.1 From MSR_P to PA_P . . . 126

5.4.2 From PA_P to MSR_P . . . 128
5.5 Correspondence Relation between MSR_P and PA_P . . . 131
5.6 Security Analysis . . . 135
5.6.1 Secrecy . . . 135
5.6.2 Authentication . . . 136
5.7 Conclusions . . . 138
5.8 (Appendix) Theorem Proofs . . . 139

6 Security Analysis with Team Automata **151**
6.1 Introduction . . . 151
6.2 Background on Team Automata . . . 152
6.2.1 The Max-ai Team Automata . . . 155
6.2.2 Compositionality in Team Automata . . . 157
6.3 An Insecure Communication with Team Automata . . . 157
6.4 GNDC Security Analysis for Team Automata . . . 159
6.4.1 Reformulating GNDC in Terms of Team Automata . . . 160
6.4.2 Security Analysis Strategies for Team Automata . . . 161
6.5 A Case Study: The EMSS Protocol . . . 163
6.5.1 The EMSS Protocol Modeled by Team Automata . . . 164
6.5.2 Analysis of the EMSS Protocol . . . 167
6.6 Conclusions and Future Work . . . 169

IV Epilogue **171**

7 Conclusions and Future Work **173**
7.1 Conclusions . . . 173
7.2 Future Work . . . 178

Bibliography **179**

List of Figures

1 Organization Chart of the Thesis ii

1.1 The LTL Operators 6
1.2 The Structure of SPIN 7
1.3 The Architecture of Computerized Central Apparatus 9
1.4 The Model of the Safety Nucleus 11
1.5 The Model of the of Safety Nucleus and Peripheral Control Units 12
1.6 Verification Results (TMR) 21
1.7 Verification Results (TMR-PCU) 27
1.8 Abstract and Concrete SPIN Models 29
1.9 Memory versus Time usage in SPIN with CO and CO+MA Options 30

2.1 The OSA/Parlay Architecture 34
2.2 The Message Sequence Chart of the TSM Protocol 36
2.3 "CoProVe" Model used for Checking the Secrecy of KA. 39
2.4 "CoProVe" Model Showing the Weaknesses of the Selection Methods 44

3.1 Flow Diagram of the Fault-Tolerant Battery 54
3.2 Example of Properties of the Equational μ-calculus 57
3.3 The partial evaluation function for $\|_{\mathcal{A}}$. 59
3.4 $\forall_{\wedge} MC$ Formulas and their Relation to Fault-Tolerant Properties. 65
3.5 The Flow Diagram of $Bat^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}}$ 69
3.6 The Minimum Automata weak Bisimilar to $Ene^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}}$ 69

4.1 The SPYDER Environment 85
4.2 Rules Defining the Derivation of Messages 90
4.3 Transition Rules of the SPYDER Labelled Transition Systems. 91
4.4 Syntax of the Logic used within SPYDER 94
4.5 Architecture of SPYDER Prototype 108
4.6 Finite State Model of the NSSK Protocol 109
4.7 Details of the Finite State Model of the NSSK Protocol 110
4.8 Details of the Finite State Model of the NSSK Protocol (continued) 111

5.1 Example of Application of $\lfloor\lceil\_\rceil_I\rfloor_I$ 134
5.2 Corresponding Couples in Relation $\mathcal{R}$ 141

6.1 Transition Space of TA 153
6.2 Example of two Composite Automata $\mathcal{C}_1$ and $\mathcal{C}_2$ 156
6.3 Example of two Different TA over $\{\mathcal{C}_1, \mathcal{C}_2\}$ 156
6.4 Insecure Communication Scenario for Team Automata 159

7.1 The Principles and Statements of Integrations. 174
7.2 Meeting Points between Fault-Tolerance and Security Protocol Analysis 175
7.3 Meeting Points between PA and MSR in Cryptographic Protocol Analysis 176
7.4 Meeting Points between Team Automata and PA in Security Protocol Analysis . . 177

Part I

Formal Validation of Systems: Industrial Test Cases

1

Validation of Fault-Tolerant Systems, a Test Case: Analysis of a Railway, Safety-Critical, Control System

"[...] e allora io quasi quasi prendo il treno e vengo, vengo da te, ma il treno dei desideri, dei miei pensieri all'incontrario va." (A. Celentano, in Azzurro*, V. Pallavicini e P. Conte, 1968)*

"[...] I am going to take the train and come, come to you. But the train of my desires, of my thoughts runs in the opposite direction.

Abstract

This chapter describes an experience of formal validation of a fault-tolerant railway control system. The system is designed for the automatic management of medium-large scale railway networks, and it is currently running at the Italian train station of "Roma Termini".

The validation experience has been conducted in 1998 in the context of an industrial joint project involving three partners: an Italian company working in the field of railway engineering, the Ansaldobreda Segnalamento Ferroviario of Napoli, and two research institutes of the Italian Research Council of Pisa, the Istituto di Elaborazione dell'Informazione and the Centro Nazionale Universitario di Calcolo Elettronico. The project required the development of formal models describing different components of the system: a fault-tolerant exclusion mechanism and a fault-tolerant communication protocol. Moreover, the project demanded the verification of fault-tolerance properties in case of Byzantine and fail silent faults.

This chapter reports on the design of the formal models and on the experiments of formal verification. We use PROMELA as specification formal language, and SPIN as a model checker. The properties of interest are specified as safety or liveness properties by means of PROMELA assertions or linear time logical formulas. To cope with the state-space explosion problem we split the main models in sub-models. Each sub-model is realized in both a concrete and an abstract version. Any abstract sub-model is provably equivalent to the corresponding concrete with respect to an established set of properties, but it contains a lesser degree of parallelism. By an appropriate composition of abstract sub-models in a whole system model, we are able to keep under control the space explosion problem and to complete the verification of most of the demanded properties.

1.1 Introduction

The need for safety in automatic management of modern railways forces the introduction of sophisticated, fault tolerant, computer-based control systems that have an intrinsic degree of complexity [13]. Their validation requires techniques from *formal methods* [111, 205, 34, 54] that are able to overcome the limitations of traditional methodologies, such as testing and simulation.

Generally speaking, formal methods are a set of mathematical approaches that support the rigorous specification, design, and verification of computer systems; for these activities, they provide formal languages, verification techniques and automatic tools. It has been proved that the use of the formal methods in industrial processes helps in reaching a high level of dependability during the design and the development of a software or hardware component [54]. Important factors encourage the application of formal methods in the industrial production. First of all, the interest in discovering as many errors as possible before entering in the production phase; during this stage the cost of correction per error increases enormously [135], whereas in railway applications an error can even cause a disaster. Secondly, many institutions require industries to conform to international standards that strongly suggest formal methods, for example the EU directives EN 50128 CENELEC Railways Applications [175], and the IEC 65108 [119].

Railway control systems are particularly suitable to be analyzed with formal methods. Eisner [73] states that railway systems share important robustness and locality properties that distinguish them from most hardware systems, this peculiarity makes them easily checkable by symbolic model checking [48] and Stålmarck checking [184]. As a matter of fact, in the last decade many railway industries have started pilot projects to evaluate the impact of formal methods on their production costs. Sometimes, industries have even developed their own validation environment such as, for example, the LIVE [14] environment by the Ansaldobreda Segnalamento Ferroviario. The experiments and the results from these pilot projects have stimulated a wide range of scientific production (*e.g.*, see [87, 132, 28, 45, 165, 27, 46]). As a significant example, Groote et al [107] use the micro Common Representative Language (μCRL) [105] to model the vital processor interlocking that runs at the Dutch station of Hoorn-Kersenboogerd; correctness criteria, expressed in a modal logic for μCRL [106], are verified automatically using tools generated with the meta environment ASF+SDF [124]. Lately, Bernardeschi et al [20] show how it is possible to formalize a significant part of a complex railway control system in the CCS process algebras [158], then properties written in the computational tree logic (CTL) [189] are verified with the tool JACK [33].

In this chapter we describe the principal results of a project carried out by the Ansaldobreda Segnalamento Ferroviario (ASF) of Napoli – an Italian company working in the field of railway engineering – and two research institutes of the Italian Research Council (CNR), the Istituto di Elaborazione della Informazione (IEI-CNR) and the Centro Nazionale Universitario di Calcolo Elettronico (CNUCE-CNR) of Pisa[1]. The project has required the validation of a fault tolerant control system in presence of Byzantine [128] and silent faults. First described in [164], the system is designed to behave safely even in case of arbitrary failures of some of its component, and it controls safety-critical components of a railway network.

The industrial partner ASF has suggested the use of the PROMELA [116] specification language and of the SPIN [117, 118] model checker. With SPIN, ASF has previously verified safety properties of different parts of the system [45]. The analysis described in this chapter uses the version 3.2 [117] of SPIN [2]. It was the newest version in 1998, when the work was conducted. Some advanced features were not present at that time, for example the extension of PROMELA and SPIN to the discrete time [32]; this explains why in this chapter we design our own strategies to describe time-related behavior, such as fail silent faults and time-outs.

[1] In September 2003 IEI-CNR and CNUCE-CNR merged into ISTI-CNR, the Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo".

[2] At present, the latest version is the 4.2.2.

1.1.1 Linear Time Logic

In section recalls the (propositional) linear time logic [174] (LTL, for short). SPIN accepts properties expressed as formulas of LTL. We start with the definition of the linear time structure, that formalizes the notion of a time line.

Definition 1.1.1 (Linear Time Structure) *Let AP be a set of atomic propositions. A* linear time structure *is a triple $\mathcal{M} = (S, x, L)$ where S is a set of states, $x : \mathbb{N} \to S$ is an infinite sequence of states (called time line), and $L : S \to 2^{AP}$ is a function that labels each state with the set of atomic proposition true in that state.*

Formulas of LTL are built using the following grammar:

$$\phi \;=\; p \;\;|\;\; \neg\phi \;\;|\;\; \phi \wedge \phi' \;\;|\;\; \mathsf{X}\,\phi \;\;|\;\; \phi\,\mathcal{U}\,\phi'$$

where $p \in AP$, is a propositional symbol. Informally, $\neg\phi$ and $\phi \wedge \phi'$ are the propositional negation and conjunction of formulas, whereas X and $\mathcal{U}$ are the basic temporal operator of LTL. The former is called "next", the latter "until". In LTL we find also the (classic) propositional derived operators:

$$\begin{array}{lcl} \mathtt{ff} & \stackrel{\text{def}}{=} & p \wedge \neg p \\ \mathtt{tt} & \stackrel{\text{def}}{=} & \neg\mathtt{ff} \\ \phi \vee \phi' & \stackrel{\text{def}}{=} & \neg(\neg\phi \wedge \neg\phi') \\ \phi \Rightarrow \phi' & \stackrel{\text{def}}{=} & \neg\phi \vee \phi' \end{array}$$

and the (important) following derived temporal operators:

$$\begin{array}{lcl} \text{"eventually"} \quad \Diamond\,\phi & \stackrel{\text{def}}{=} & \mathtt{tt}\,\mathcal{U}\,\phi \\ \text{"always"} \quad \Box\,\phi & \stackrel{\text{def}}{=} & \neg\Diamond\,\neg\phi \end{array}$$

Formulas of LTL are interpreted over linear time structures $\mathcal{M} = (S, x, L)$. A graphical, intuitive, explanation of the temporal operators is shown in Figure 1.1.

Informally, we say that a formula ϕ holds in a state of a time line we mean that it holds in the time line that starts from that state; we say that a formula holds in the time line $\mathcal{M}$ if it holds in state the first state $x(0)$. The informal semantics of the temporal operators is as follows: $\mathsf{X}\,\phi$ holds in $\mathcal{M}$ if and only if ϕ holds in state $x(1)$; $\Box\,\phi$ holds in $\mathcal{M}$ if and only if ϕ holds in every state of the time line; $\Diamond\,\phi$ holds in $\mathcal{M}$ if and only if there is a future state of where ϕ eventually holds; $\phi\,\mathcal{U}\,\phi'$ holds in $\mathcal{M}$ if ϕ holds in all the states until (possibly included) the state where ϕ' holds.

Formally, the notation $\mathcal{M}, x \models \phi$ means that ϕ is true in the time line x of the structure $\mathcal{M}$. Assuming the notation x^i standing for the suffix $x(i), x(i+1), \ldots$ of the time line x, the satisfiability relation, $\models$, is defined inductively on the structure of the formula ϕ as follows:

$$\begin{array}{lll} \mathcal{M}, x \models p & \text{iff} & p \in L(x(0)) \\ \mathcal{M}, x \models \phi \wedge \phi' & \text{iff} & \mathcal{M}, x \models \phi \text{ and } \mathcal{M}, x \models \phi' \\ \mathcal{M}, x \models \neg\phi & \text{iff} & \mathcal{M}, x \not\models \phi \\ \mathcal{M}, x \models \mathsf{X}\,\phi & \text{iff} & \mathcal{M}, x^1 \models \phi \\ \mathcal{M}, x \models \phi\,\mathcal{U}\,\phi' & \text{iff} & \text{exists } j,\ \mathcal{M}, x^j \models \phi' \text{ and for all } i < j,\ \mathcal{M}, x^i \models \phi \end{array}$$

We explicitly give also the formal semantics of the derived temporal operator:

$$\begin{array}{lll} \mathcal{M}, x \models \Box\,\phi & \text{iff} & \text{for all } i \geq 0,\ \mathcal{M}, x^i \models \phi \\ \mathcal{M}, x \models \Diamond\,\phi & \text{iff} & \text{exists } i \geq 0,\ \mathcal{M}, x^i \models \phi \end{array}$$

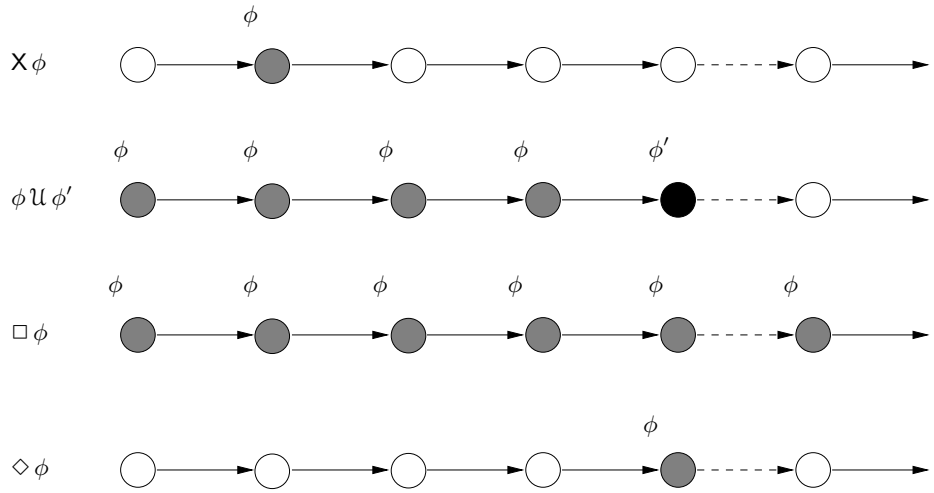


Figure 1.1: Graphical intuition for the semantics of the linear time operators "next", "until", "always", and "eventually". Each sequence represents the time line satisfying the related formula on the left. States are depicted as circles, and arrows connect states in temporal sequence. A formula above a state indicates that the formula holds in the state.

A formula ϕ is said to be satisfiable if and only if there exists a linear time structure $\mathcal{M} = (S, x, L)$, such that $\mathcal{M}, x \models \phi$; in this case we say that $\mathcal{M}$ is a *model* for ϕ. In this chapter, the following problem is also relevant:

Definition 1.1.2 (Model Checking Problem in LTL) *Let $\mathcal{M} = (S, x, L)$ be a linear time structure, and ϕ be a LTL formula. The* model checking problem *consists in answering the following question: "is $\mathcal{M}$ a model for ϕ"? Or equivalently does "$\mathcal{M}, x \models \phi$"?.*

An algorithm solving the model checking problem is called model checker.

1.1.2 PROMELA and SPIN

This section briefly introduces the SPIN model checker [118] and its high-level specification language, PROMELA [116]. We do not enter in any technical detail here: when necessary throughout this chapter, we shall provide brief explanations. For a complete reference on SPIN and PROMELA we suggest the book [118].

SPIN is an efficient tool for the simulation and the verification of PROMELA models. SPIN runs on Unix, Linux, and Windows. Its basic structure is illustrated in Figure 1.2. In simulation mode, SPIN can be used to get a quick impression of the types of behavior that are captured by a model. In verification mode, SPIN checks correctness claims that are generated from logic formulas expressed in LTL. When a claim is not valid over a model, SPIN produces a counter example that shows explicitly how the property may be violated. The counter example can be fed back to the SPIN simulator, so that the trail can be inspected in detail to determine the cause of violation.

At high-level, a system model is specified as a set of PROMELA process templates, that SPIN translates into a set of finite Büchi automata [39]. A global automaton of the system behavior is obtained by the interleaving product of all the automata composing the system. Once a model is built, SPIN is used to generate an optimized, on-the-fly, verification program that can be compiled

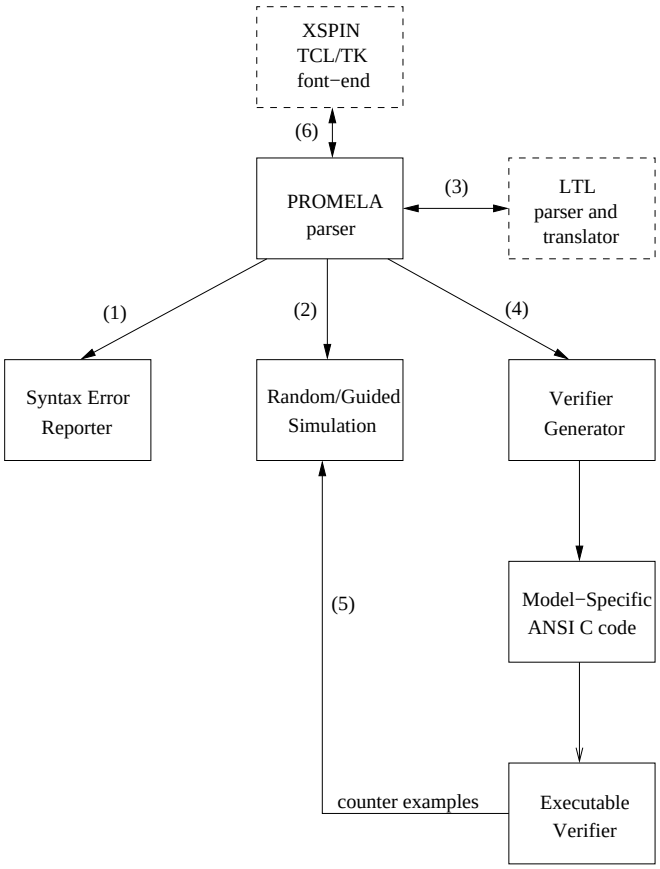


Figure 1.2: The structure of SPIN. A PROMELA, high-level, description of a system is first checked for syntax errors (1). Interactive simulation can be used to gain basic confidence that the model has the intended properties (2). Optionally a PROMELA correctness claim can be generated from a LTL formula (3). Then, SPIN is used to generate a verifier, compiled with possible compile-time choices for optimization in memory usage (4). If the verification fails, SPIN returns a counter example that can be fed back into the simulator (5). A graphical front-end, XSPIN, provides a user-friendly approach to the SPIN environment (6).

and run separately. Different options can be set when compiling a verifier: partial order reduction, memory compression, data compact representation, or other optimization strategies can be exploited in the analysis to deal with state space explosion problem [201], a fundamental problem for any state space methods like model checking. Almost any system has huge number of states, and the size of the structure used to represent a system, called states space, tends to grow exponentially in the number of its processes and variables. This explosion causes a seriuos waste of the computer memory, and in absence of optimization strategies it usually makes many verification fail by out-of-memory.

Significant to this chapter are the optimization options of partial order reduction and memory compression methods. Partial order reduction aims to reduce the number of system states that need to be visited and stored in the state space to solve the model checking problem. This option is enabled by default for all SPIN verification runs. Memory compression methods aim to reduce the amount of memory that is required to store each state of the system. Options, such as `COLLAPSE`

and MA are reserved for this target. Both reduce the memory requirement of an exhaustive search by increasing the run-time requirements. The COLLAPSE feature exploits a hierarchical indexing method to achieve compression. The MA, or minimized automaton, reduces memory by building and updating a minimized finite state recognizer the state descriptor. Technical notes about the algorithms for partial order reduction and memory compressions can be found in [118].

PROMELA is a language with a C-like syntax for the specification of the high-level behavior of concurrent and interacting processes in a distributed system. PROMELA is based on Dijkstra's guarded command language [68] and Hoare's language CSP [115]; PROMELA has nondeterministic control flow structures and primitives for process creation and interprocess communication. Other control flow statements allow the definition of atomic sequences, deterministic steps, and escape sequences. A PROMELA specification consists of one or more process templates, called *proctype*, and at least one process instantiation. Processes are typically instantiated by the built-in "init" process or by any already running process. Processes may terminate or run indefinitely. Instantiated processes communicate via rendez-vous, via asynchronous message passing through buffered channels, or shared memory.

1.2 System Description

The object of our study is the *Computerized Central Apparatus* (ACC) [3] a hardware system specifically designed to manage medium/large railway networks. ACC is a highly programmable and centralized control system deployed in a wider railway signaling system. This latter is a complex and distributed architecture designed to manage a large railway network. Each node of the network controls either a medium-large railway station or a line section with small stations, or a traffic line with a simple interlocking logic. Figure 1.3 depicts the ACC architecture. The ACC is composed of two independent sub-systems dedicated to management and vital functions:

Management functions control auxiliary tasks, such as data recording, diagnostic management and remote control interface. They are run by the ACC sub-system called "RDT" (acronym for Recording, Diagnosis and data Transmission) in Figure 1.3.

Vital functions are generally safety-critical procedures: they control critical machineries such as train movements and the wayside equipment. Vital functions are run by the ACC sub-system called "Vital Section" in Figure 1.3.

The vital section of the ACC is composed of several *Control Posts*, several *Peripheral Control Units* (PCUs), and a *Safety Nucleus* (SN). Control Posts are formed by input/output interfaces and by terminals. From them, a human operator can issue critical commands intended for the PCUs that, in turn, execute them. These commands are critical because their execution affects physical machineries such as railway semaphores, rail points or level crossings. For this reason particular attention is paid to guarantee the safety of the system in case of faults.

The SN, a hardware component, is specifically designed for control and safety purposes. It monitors the state of the system and tries to discover a faulty component, *i.e.*, a PCU or a communication bus. Its architecture is based on a triple modular redundant [191] configuration of computers; for this reason the SN also faces the problem of an internal consensus. The classical solution to this problem (also known as the Byzantine Generals Problem [128, 22]), cannot be implemented in the SN due to hardware constraints; in case of inconsistency, instead of looking for a

[3] "Apparato Centrale a Calcolatore", in Italian.

consensus the SN tries to exclude the faulty component or to force the whole system to shutdown safely.

Remark 1.2.1 We study of the behavior of the SN and its interaction with the PCUs. ■

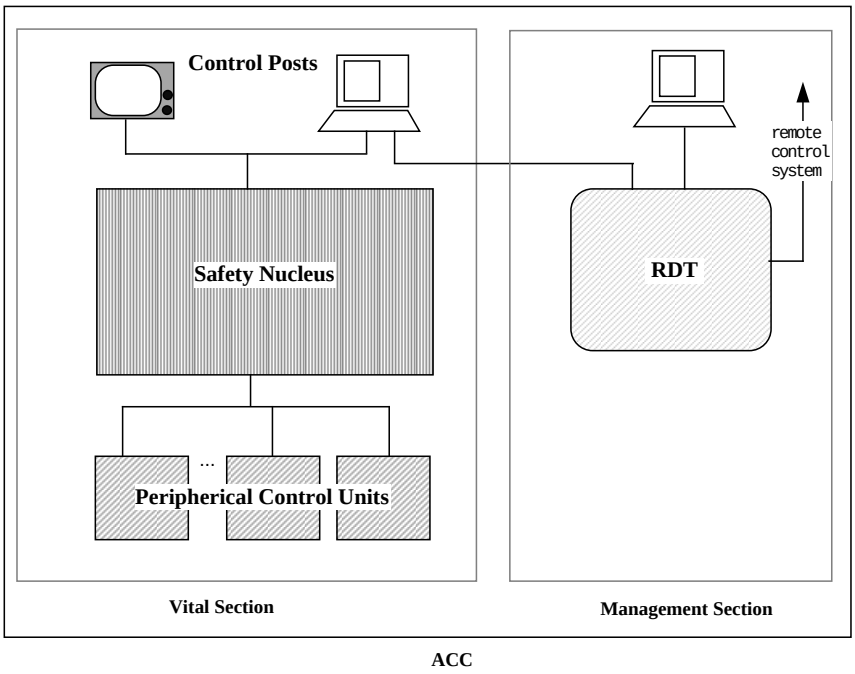


Figure 1.3: The architecture of the Computerized Central Apparatus (ACC). The Management Section controls auxiliary tasks. The Vital Section controls physical devices and train movements. Commands issued from the Control Posts are executed by the Peripheral Control Units. The Safety Nucleus controls and manages the system in case of arbitrary faults in the Peripheral Control Units or in the communication lines.

1.3 Formal Models

This section discusses how we represented time-outs, Byzantine faults and transient faults in PROMELA. It also illustrates the two PROMELA models of the vital section of the ACC, which we call TMR and TMR-PCU; they describe different views of the ACC vital section:

1. TMR describes, in detail, the triple modular redundant architecture of the SN and its exclusion logic mechanism (see also Figure 1.4). We use TMR primarily to verify safety properties of the SN in presence of Byzantine behavior of one of its components. PCUs play only a marginal role here.

2. TMR-PCU describes, in detail, the SN-PCU communication infrastructure (*i.e.*, busses), the relative communication protocol, and the internal PCU architecture (see also Figure 1.5). Here, we model only those parts of the SN that are involved in the communication with the PCUs, *i.e.*, we explicitly avoid modeling the exclusion logic. We use TMR-PCU to verify

liveness properties of the SN-PCU communication protocol and safety properties in case of communication time-outs caused by faults in the busses or in some PCUs.

1.3.1 Formalization of Faults

Before describing the PROMELA models TMR and TMR-PCU, we focus our attention on the formalization of the classes of faults that we consider in our analysis:

- fail silent faults (*i.e.*, faults causing time-outs in communication);
- Byzantine faults.

Both classes of faults are used within the models TMR and TMR-PCU. Byzantine faults are supposed to happen only in some SN modules. In reality Byzantine faults only occur in SN modules, therefore we assume that they do not occur elsewhere. Fail silent faults may originate in any ACC component: SN modules, PCUs, or communication lines. Moreover, we also discuss how to model temporary fail silent faults.

Fail Silent Faults

Fail silent faults cause the system to omit the correct answer [19]. In the case of ACC they cause time-outs in communications. In other words, a fail silent fault becomes visible to the other system components only when a communication event results in a time-out. ACC communications are with time-outs, but since PROMELA does not deal with time[4], we have to abstract from any definition of it in our models. We simulate time-outs with a special *empty message* ϵ, whose presence in a channel must be interpreted as absence of the expected message. The use of the empty message lightly changes the interpretation of send and receive and, consequently, their implementation in our models. A "send" of a message m is now implemented as a non deterministic choice between transmitting either m or ϵ. A "receive" of a message in variable x requires a test $x == \epsilon$ after the reception: in fact, in a 'receive" with time-outs there is the need to discern, depending on the content of the message gotten, if a time-out has expired or not. This latter case happens if and only if the message received is the empty message ϵ.

In PROMELA, where typed and buffered (of lenght N) channels are defined via the declaration `chan <name> = [N] of <type>`, the message ϵ is defined as a reserved constant value for example the integer value 0. This value must not be used in any other communication along the whole formal model. Consequently, a "send" with time-outs is implemented in PROMELA with the following code:

```
/* ************************************* */
/* implementation of a send with time-out  */
/* ************************************* */

//  global definitions
//  (in the environment where the module are defined)

define EMPTY  0 // empty message
```

[4] See the discussion in Section 1.1.2

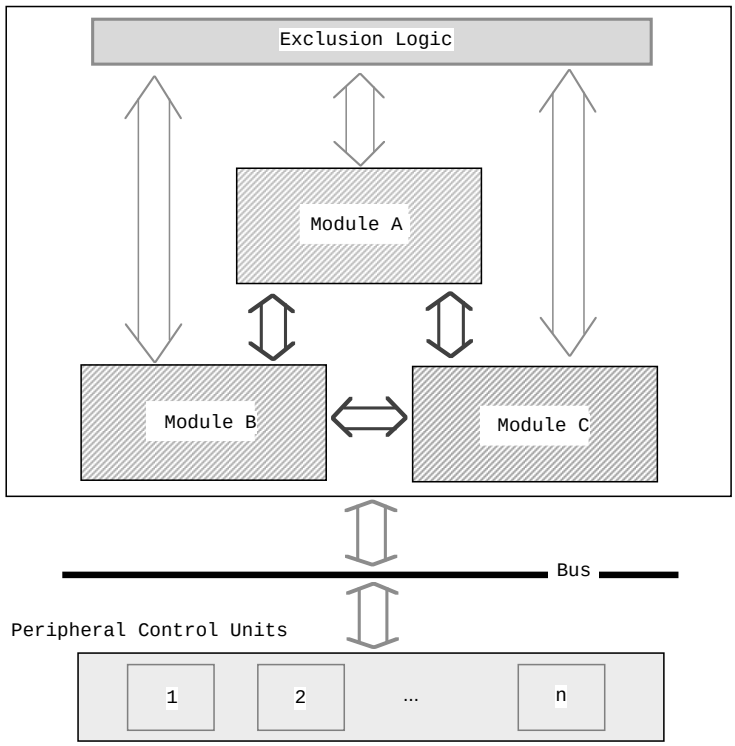


Figure 1.4: The architecture of TMR. The triple modular redundancy of the SN is represented in detail: three (functionally) identical modules and an exclusion logic. Each SN module communicates with the others and with the exclusion logic via a dedicated symmetric link. PCUs are connected with the SN through a shared bus.

```
chan c = [0] of <t>; // (synchronous) channel

[...]

// local definition  (within a module)
<t> msg;             // message (<> 0) of type <t>

[...]

/* implementation of a send with time-out  */
if
:: c!msg    // send the real message
:: c!EMPTY  // send the empty message
fi;
```

In PROMELA, comments are enclosed within `/* */` and the statement

```
if :: <guard1> -> <s1>
   :: <guard2> -> <s2>

  [...]
```

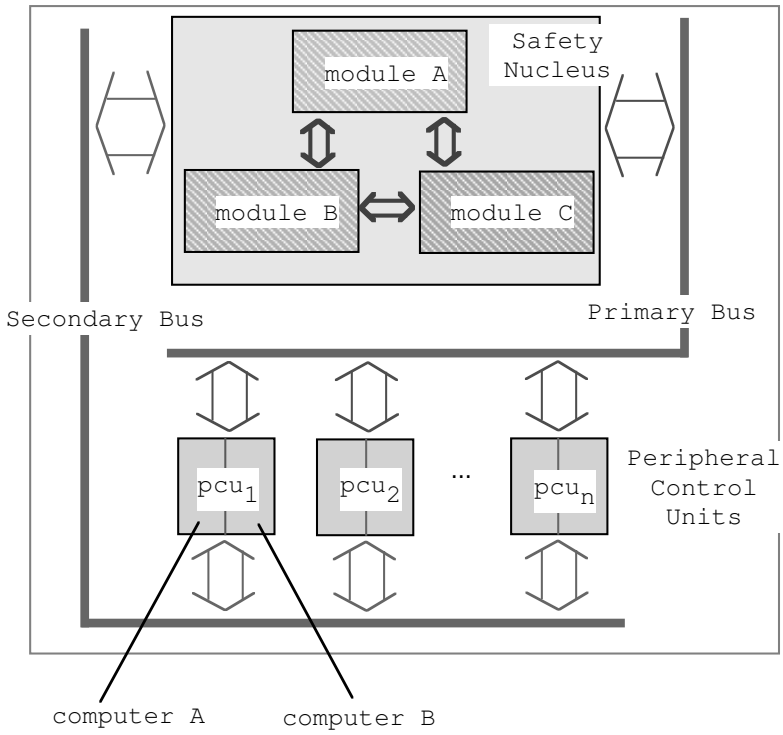


Figure 1.5: The architecture of TMR-PCU. The communication infrastructure between SN and the PCUs is represented in detail. PCUs are linked with the SN modules through a pair of busses, called primary and secondary respectively. Each PCU module is composed of two identical computers in configuration 2 out-of 2.

```
   :: <guardn> -> <sn>
   :: else     -> <sn+1>
fi
```

is a guarded, non deterministic, choice among the statements `s1`, `s2`, ..., `sn`. A statement `s`, is enabled if the corresponding guard, `guard`, is satisfied. When more statements are enabled, one statement is selected non-deterministically. When present, the `else` guard is satisfied if and only if all the other guards are not. The keyword `true`, is a guard that is always enabled; it is usually omitted and `:: true -> <s>` is written as `:: <s>`. The primitive `c!x` is the send command over the channel `c` of the value associated to `x`. In the previous code implementing the send with time-outs, the guards are always satisfied, so the "send" with time-outs is a pure non-deterministic choice between sending the message `msg` or the empty message.

In PROMELA the "receive" with time-outs is coded as follows:

```
/* ************************************** */
/* implementation of a receive with time-out */
/* ************************************** */

// global definitions
define EMPTY  0      // empty message
```

```
chan c = [0] of <t>; // (synchronous) channel

[...]

/* implementation of a receive with time-out  */
c?x ->
     if
     :: (x == EMPTY)  ->  <exception time-outs>
     :: else          ->  <reception>
     fi;
```

In PROMELA, `c?x` is the receive operation from channel `c` of a value that will be stored into the variable `x`. In the previous code, a message is first retrieved from the channel `c` then, if the message proves to be the empty message, a procedure for handling the time-out is called.

Byzantine Faults

We start with a definition of Byzantine behavior due to Lamport *et al* [128]:

Definition 1.3.1 (Byzantine Behavior) *Assuming to have a finite set M of identical modules m whose behavior is specified by a communication algorithm A:*

1. *all loyal modules in M run the same algorithm A, and in particular they correctly send all messages as specified in A;*
2. *a Byzantine module $m' \in M$ runs the same algorithm A as a loyal module but it can arbitrarily fail in executing it. As an effect of failure the Byzantine module may send wrong messages, it may send a message delayed with respect to a synchronization event, or it may send no message at all.*

Definition 1.3.1 focuses on communication events: any Byzantine fault in a module becomes observable only when the faulty unit communicates. As a consequence of this assumption any Byzantine fault is modeled as a communication error; precisely as a communication of a corrupted message or as a delay in the communication, or as a lack of communication.

We model both a delay and a lack of communication with a time-out *i.e.*, with the empty message ϵ. To generate a corrupted message, we define a function $corrupt() : (T - \{\epsilon\}) \longrightarrow T$, where T is a message type. Given a message m, $corrupt(m) \neq m$ indicates that the message m is corrupted.

In PROMELA, an instance of $corrupt()$ is the function $corrupt(n) = -n$. Note that, because for the choice of modeling the empty message with the integer value 0 there is no semantic ambiguity between the concepts of "corrupted message" and "absence of a message".

Byzantine faults, and the way we model them, affect the PROMELA implementation of a send. On the contrary the implementation of a receive does not require any further change with respect to its implementation with time-outs. In PROMELA a Byzantine send appears as follows:

```
/* ************************************* */
/*    implementation of a Byzantine fault    */
/* ************************************* */
```

```
// global definitions
define EMPTY 0 // empty message
chan c = [0] of <t>; // (synchronous) channel

// local (to a module) definitions
<t> msg;             // message

[...]

/* implementation of a send with Byzantine failure */
if
:: true -> c!msg          // send the corrected message
:: true -> c!EMPTY        // time-outs (delay or lack)
:: true -> c!(-msg)       // send a corrupted message
fi;
```

In the code above, a non deterministic choice guides the possibility of sending the correct message or causing a time-out, or sending a wrong message.

Temporary Fail Silent Faults

The behavior of each ACC module (see also Section 1.4 and 1.5) consists of a cyclic execution of a sequence of statements; we call this *execution loop*. ASF is interested in modeling time-outs that are persistent for at least one whole execution loop but not necessarily in all the loops. This interest is motivated by what ASF has observed in the field.

As a solution, we model such faults in the following way: at the beginning of an execution loop, a non-deterministic choice decides if a component, for instance a bus or a module, run in either error-prone or in error-free mode. Running in error-prone mode means that every communication involving the component ends in a time-out. For example, if the component represents a communication bus, every communication through it results in a time-out. A scheme of this solution in PROMELA is as follows:

```
/* initial setting of the state bit */
bit error_free = 1

/* execution loop */
do

 /* change the state bit */
  if
     ::  error_free = !error_free
     ::  skip
  fi;

      do  /* a send */
        :: c!(msg && error_free)
```

```
        :: [...]
      od
do
```

In the code above, the state bit is changed first, then the send occurs. Moreover the statement

```
do
  :: <guard1> --> <s1>
  :: <guard2> --> <s2>

[...]

  :: <guardn> --> <sn>
od
```

is a repetition construct; it cyclically selects, non-deterministically, one enabled statement among the guarded statements `s1`, `s2`, ..., `sn`. We use the repetition construct to implement an execution loop. At the beginning of the execution loop a choice may change the state from error free (`error_free = 1`) to error prone (`error_free = 0`) or vice versa. Later any outgoing message is sent in conjunction with the state bit: a value of 0 has the effect of resetting the outgoing message to the value we use to model the empty message.

1.4 The TMR model

This section describes the TMR model. Its general architecture, drawn in Figure 1.4, consists of:

- three identical *central modules*, called module A, module B and module C. They constitute the triple modular redundant configuration of the SN. They communicate with each other, with the exclusion logic, and with the PCUs.
- a module called *exclusion logic*. It watches the central modules and acts as a voter. Moreover, the exclusion logic is able to exclude one inconsistent central module or to bring the SN to a safe shutdown.
- the PCUs, consisting of n control units (in our study $n = 2$). In the TMR, the behavior of the PCUs is only sketched;
- the set of *communication channels*. Three symmetric channels connecting the three central modules, three symmetric channels between the central modules and the exclusion logic, and a single bus between the central modules and the PCUs.

In the following we explain only the PROMELA model of a central module. This is sufficient to understand what we are going to verify. The complete PROMELA codes, composed of thousands of code lines, is property of ASF. We describe here, with permission, only what is needed to understand this work.

The Central Modules In TMR, modules A, B, and C are designed for the following tasks:

- to collect global information about the whole system state *i.e.*, the local states of the other central modules;
- to elaborate summary information about the system state first, then sending it to the exclusion logic;
- to communicate with the PCUs.

The three central modules communicate with each other via symmetric channels. Each module is connected via another symmetric channels with the exclusion logic, and via a bus with the PCUs. The behavior of a central module is the execution loop, formally described with the following pseudo-code:

```
/* ****************************** */
/*          execution loop        */
/* ****************************** */
loop
1. * <synchronization>
2.   <command elaboration>
3. * <data exchange with the other modules>
4.   <distributed voting>
5. * <communication with exclusion logic>
 /* communication with the 2 PCUs */
       for i = 1 to 2 do
       6.1   if <is my turn> then
       6.2 *   <synchronization>
       6.3 *   <send command to the PCUs[i]>
       6.4 *   <receive acknowledge from the PCUs[i]>
       endfor
endloop
```

In the code above we indicated the communication phases with an "*". We now describe informally each phase.

Synchronization. During this phase, each module exchanges a synchronization message with the other modules. This phase is used to collect information about the state of activity of the other modules. A time-out is interpreted as a sign of non activity. and the module that caused the time-out is excluded by any later communication within the current execution loop. Within the current loop the module that has caused the time-out is excluded from any subsequent communication. The system is expected to run in a configuration of at least 2 out of 3; if a module detects a time-out from both the other two modules it enters in a safe shutdown state;

Command elaboration. During this phase, each module composes a summing up of the local view that the module has about the state of activity of the other two modules;

Data exchange. During this phase, each module sends to, and receives from the other modules the message composed in the previous phase;

Distributed voting. During each phase, each module checks the consistency of its local information (about the system state) with that received from the other two modules, and composes a new message about the result of this test;

Communication with the exclusion logic. During this phase, the result of the test is sent to the exclusion logic, which, after having analyzed all the results, can decide to disconnect the module(s) considered potentially faulty;

Communication with the PCUs. During this phase, a module communicates with the PCUs by running a particular circular protocol. At each loop of this protocol only two modules are enabled to communicate with the PCUs. A distributed procedure within the protocol, assures a cyclic selection of the two modules candidate to the communication. In the TMR, this procedure is extremely simplified.

In the following we report a synthesis of the PROMELA code implementing the synchronization phase for the module C. In the code we have omitted programming details that are not significant at this level of description, for example the statements `atomic` or `d_step` used to reduce unnecessary parallelism in the model.

```
/* ************************************** */
/*           synchronization phase        */
/* ************************************** */

/***      in the global environment       ***/
#define EMPTY 0 // the empty message
#define SYNCH 1 // the synchronization message


/* global_activeC is the global state of module C.  */

global_activeC = 1  // state of activity of C

[..]

/***        module C's local variables       ***/
activeA_C = 1; /* state of A, in C viewpoint     */
sentA_C = 0;   /* flag "sent" (to module A)      */
recvA_C = 0;   /* flag "received" (from module A)*/

activeB_C = 1; /* state of B, in C viewpoint     */
sentB_C = 0;   /* flag "sent" (to module B)      */
recvB_C = 0;   /* flag "received" (from module B)*/

do

/* -------------------- */
/* communication with A */
/* -------------------- */
```

```
/* send (with time-outs) to A */
 :: (!sentA_C) ->
     if
     // send the synch (if A is active)
     :: true -> outA!(SYNCH && activeA_C);
     // time-outs
     :: true -> outA!(EMPTY);
     fi;
     sentA_C = 1;

/* receive (with time-outs) from A */
 :: atomic{(!recvA_C && inA?[synA]) -> inA?synA;}
            recvA_C = 1;

/* set the activity state of A */
      if
      :: synA == SYNCH -> activeA_C = 1;
      // time-outs imply no activity
      :: else -> activeA_C = 0;
      fi;

/* -------------------- */
/* communication with B */
/* -------------------- */

/* Here activeB_C is used instead of activeA_C  */
/* recvB_C instead of recvA_C,                  */
/* sentB_C instead of sendA_C etc.              */

 :: [ ... the same for B ... ]

/* exitloop when done all the two modules */
 :: (sentA_C && sentB_C && recvA_C && recvB_C) -> break;
od;

/* safe shutdown if A and B are not active */

if
:: !activeA_C && !activeB_C ->
    /* goto a part of the code that is recognized as a */
    /* safe shutdown. The module wait to be restarted  */
    goto SHUTDOWN

:: else -> skip
fi;
```

```
SHUTDOWN:
   /* Set the state of activity of C to ''inactive''  */
   /* C has safely shut down.                         */
       global_activeC = 0;
   <wait>
```

Here `activeA_C` and `activeB_C` are C's local variables that indicate the state of activity of module A and module B, respectively; they are set at the beginning of the execution loop, and reset in case a time-out occurs in a communication with module A or module B, respectively.

The PROMELA code implementing the other phases is similar except for the type of messages involved and for some different local computation. As explained in Section 1.3.1, in its Byzantine implementation a module may, in any send action, send a corrupted message. In reference to the previous code, the fragment of PROMELA code that shows the Byzantine implementation of the "communication with module A" is as follows:

```
[...]

 /* communication with A */
 :: (!sentA_C) ->
     if
     /* send the synch (if A is active) */
     :: true -> outA!(SYNCH && activeA_C);
     /* send a corrupted message */
     :: true -> outA!(-SYNCH && activeA_C);
     /* time-outs */
     :: true -> outA!(EMPTY);
     fi;
     sentA_C = 1;

[...]
```

1.4.1 Formal Verification of TMR

This section lists some of the properties we verify for the TMR model and the related results. We postpone the discussion about how to cope with the state explosion problem, till Section 1.6. Properties are expressed as either LTL formulas or PROMELA *assertions*. An assertion in PROMELA is a statement including a boolean expression that is evaluated each time the statement is executed. Assertions are used to express invariant properties over a model.

An informal description of the properties is as follows:

(TMR1) *After a communication phase it is always true that if two modules do not receive any reply from the third module, this latter module will be eventually disconnected by the exclusion logic.*

(TMR2) *After a communication phase, it is always true that if one module does not receive any reply from the other two modules, it will eventually enter a safe shut-down state.*

(TMR3) *After a distributed voting phase, it is always true that if two modules, in reciprocal agreement on the global state knowledge, recognize that a third module is not in agreement with them, this latter module will be disconnected eventually by the exclusion logic.*

All the previous properties can be formalized with LTL formulas with the following common structure:

$$\Box\,(p \Rightarrow \Box\,(q \Rightarrow \Diamond\, r))$$

Here p, q and r are predicates on variables.

As an example let us consider property TMR1 written in the following way: "after a communication phase it is always true that if module A and module B do not receive any reply from C, this latter will be eventually disconnected by the exclusion logic". This formula is expressed with the following formula:

$$\Box\left(\neg(\texttt{activeC\_A}) \Rightarrow \Box\left(\neg(\texttt{activeC\_B}) \Rightarrow \Diamond\,\neg(\texttt{global\_activeC})\right) \wedge \Box\left(\neg(\texttt{activeC\_B}) \Rightarrow \Box\left(\neg(\texttt{activeC\_A}) \Rightarrow \Diamond\,\neg(\texttt{global\_activeC})\right)\right)\right. \tag{1.4.1}$$

Here `activeC_A` is a boolean variable of module A that evaluates to true if and only if A receives a reply from module C; `activeC_B` evaluates to true if and only if B receives a reply from module C; `global_activeC` is a global valuable that evaluates to true if and only if module C is active. Informally formula (1.4.1) evaluates true if and only if when A does not receives a reply from C, and B does not receive a reply from C then eventually C is not active.

(TMR4) *After a communication phase, every module has sent and received a message (or the empty message) from all the other modules.*

Property TMR4 is specified with an assertion placed after each communication phase. For example, this property in case of module C, is:

```
assert{(recvA_C+recvB_C==2) && (sentA_C+sentB_C==2)}
```

Here variables `recvA_C` and `recvB_C` are reset at the beginning of the execution loop, and set after module C has received a message from module A and module B respectively. Similarly, `sentA_C` (`sentB_C`, resp.) is reset at the beginning of the loop, and it is set after any send action towards module A (module B, resp.).

(TMR5) *A module is in safe shut-down state only if the other two have caused a time-out in a previous communication phase.*

Within module C, this property is specified as the following assertion located after the `SHUTDOWN` entry label (see the PROMELA code in Section 1.4):

```
assert{activeA_C + activeB_C == 0}
```

property	*state vector*	*depth*	*RAM*	*result*
TMR1	192	5266	20	success
TMR2	192	5266	22	success
TMR3	196	45273	25	fail
TMR4	68	9763	14	success
TMR5	68	9763	14	success

Figure 1.6: Summary of the verification results on TMR. These results are obtained by running SPIN with MA+CO options selected, and with one Byzantine module. In the first column we have the property name, in the second the number of bytes required to store the state vector, in the third the depth of the search in number of steps, in the fourth the total memory required by the verification in Mbytes, and in the last the result of the verification.

1.4.2 Discussion

Figure 1.6 reports a summary of the results of the verifications. We run the verification in presence of one Byzantine module. We briefly discuss the result concerning property TMR3. The analysis of the counterexample shows that the Byzantine module C causes one of the loyal modules to be disconnected by the exclusion logic. In fact, module C fails in participating in a communication with one module and makes that module believe that module C is not active. Consequently, in the distributed voting the loyal module is found in disagreement, and then disconnected by the exclusion logic. This is a typical disagreement situation due to Byzantine behaviors.

1.5 The TMR-PCU model

The TMR-PCU describes the SN-PCU communication protocol and in more detail the architecture of PCUs. Figure 1.5 depicts a scheme of the TMR-PCU architecture, that is composed of:

- the three identical *central modules*, A, B and C. Here the modules implement an abstraction of the SN, *i.e.*, the part significant for the later analysis;
- the PCUs. They are composed of n control units (in this study n=2), each consisting of two computers, computer A and computer B;
- the *interconnection* channels. Three symmetric channels connecting the three central modules, and two busses, connecting the three modules to the two computers of the PCUs.

With respect to the TMR-PCU model we are interested to verify:

1. *liveness properties* of SN-PCU communication protocol in the absence of a Byzantine module. This protocol is implemented as a distributed algorithm designed to assure a cyclic use of the busses and a cyclic selection of two central modules demanded to send the commands.
2. *safety properties* of SN-PCU communication protocol in case of some hardware faults. In particular we are interested in temporary, fail silent faults in the interconnection busses and in the computers A and B of a PCU.

In our PROMELA code we define a process for each central module and a process for each peripheral unit. We also define a synchronous symmetric channel between the central modules, and four busses between the SN and PCU. The first two busses, model the primary and secondary bus connecting SN and all computers "A" of each PCU; the second two busses model the primary and secondary busses connecting SN and all computers "B" of each PCU. In this way we want to distinguish between computer A and B, avoiding to define different processes for this.

We now briefly describe the protocol run by a central module and the one run by a peripheral unit, giving the PROMELA code where significant. This protocol is a detailed version of the "communication with the exclusion logic" phase in the TMR (see the pseudo-code in Section 1.4)

The protocol run by a central module It consists of several phases, as described by the following pseudo-code:

```
loop
/* communication with the PCUs */
 for i = 1 to 2 do
  6.1  <synchronization>
  6.2 *   <decide the turn>
  6.3 *   (x,j) =  <diagnostic>
  6.4 *    msg  =  <message elaboration>
  6.5   if <is my turn> the
  6.6 *   <send msg to computer[x] of PCU[i], via bus[j]>
  6.7 *  <receive acknowledge>
 endfor
endloop
```

Informally, before communicating with the PCUs a module tries to gather information about the global state of the system. In this case it is the state of activity of the other two modules, the state of the two busses, and the state of the two computers of each PCU. We now describe each phase separately.

Synchronization. This phase is a synthetic version of the synchronization phase of the TMR. During this phase, a module checks the other modules activity state. This information is used in a distributed tournament procedure to decide what module is enabled to send a message to the periphery.

Diagnostic. During each phase, each module summarizes information about the global state, composed of the activity state of the PCU computers and of the busses. This information is used to decide which bus to use, and whether computer A or B will be the recipient of the message to be prepared next.

Message elaboration. Depending on the state of the PCU computers, either the effective peripheral command or a special DIAGNOSTIC message is prepared;

Communication with the PCUs. During this phase, the SN sends its prepared message to the PCUs.

The protocol run by a peripheral unit

In TMR-PCU the PCU model is realized in more detail. It is schematically described by the following pseudo-code

```
loop
1. <decide the state>
/* communication with the safety nucleus */
  parallel_for (i = A to B) and (j = 1 to 2) do
   2.1 <computer[i] receives from a module via bus[j]>
   2.2 <command elaboration>
   2.3 <computer[i] send the ack via bus[j]>
  endfor
endloop
```

We now describe each phase in detail:

Decide the state. During this phase, non-deterministic choice is made to decide on the functional state of the busses and of the computers A and B of the peripheral unit. In case of a state set to "fault" every communication results in a time-out;

Receive a command. During this phase, each computer of each unit waits for a message from one of the busses;

Elaborate the command. During this phase, each computer of each unit evaluates the message received. A diagnostic message does not imply any further action, while effective commands carry information about what action the PCUs have to perform. In our model they are simply stored in a PCU local stack;

Acknowledgment. During this phase, an acknowledgment message is sent back to the all the module of SN.

In the following we give a synthesis of the PROMELA code of the PCU, called CDA1:

```
/* recvA1_CDA1,recvB1_CDA1  */
/* # msg received via bus1, by computer A and B, resp. */
/* initially set to 0 */

/* recvA2_CDA1,recvB2_CDA1: */
/* # msg received via bus1, by computer A and B, resp. */
/* initially set to 0 */


/* stateBUS1, stateBUS2: state of bus1, bus2          */
/* stateA, stateB      : state of computer A,B        */
/* all set to 1, meaning that the state is not faulty */

/* loop */
do
```

```
::
/* decide the state */
 if
   :: stateA = 0 /* fault in the 1st computer  */
   :: stateA = 1 /* the 1st computer is now ok */
   :: stateB = 0 /* fault in the 2nd computer */
   :: stateB = 1 /* the 1st computer is now ok */
   :: stateBUS1 = 0 /* fault in the 1st bus */
   :: stateBUS1 = 1 /* 1st bus is ok */
   :: stateBUS2 = 0 /* fault in the 2nd bus */
   :: stateBUS2 = 1 /* 2nd bus is ok */
 fi;
RECEIVING:skip;
   i = 0;

    /* Promela channels defined */
    /* A1, A2 : computer A-BUS1, A-BUS2  */
    /* B1, B2 : computer B-BUS1, B-BUS2  */
     do

/* ---------------------------- */
/* computer A receives from bus1 */
/* ---------------------------- */

     :: atomic{!DONE && A1in?[PCU1, senderA1, msg] ->
          A1in?PCU1, senderA1, msg;}

        if
/* if it is a diagnostic message */
          :: msg == DIAGNOSTIC -> skip;

/* if it is a command message store it */
          :: else -> msg[i] = msg; i++;
        fi;

/* acknowledgment to all modules */
        A1out!PCU1,A,(stateA && stateBUS1);
        A1out!PCU1,B,(stateA && stateBUS1);
        A1out!PCU1,C,(stateA && stateBUS1);

        recvA1_CDA1++;

/* ---------------------------- */
/* computer A receives from bus2 */
/* ---------------------------- */

    :: [... the same using A2, stateBUS2,
```

```
                recvA2_CDA1, senderA2 and stateA ..]

/* ---------------------------- */
/* computer B receives from bus1 */
/* ---------------------------- */

    :: [... the same using B1, stateBUS1,
                recvB1_CDA1, senderB1 and stateB ..]

/* ---------------------------- */
/* computer B receives from bus2 */
/* ---------------------------- */

    :: [... the same using B1, stateBUS2,
                recvB2_CDA1, senderB2 and stateB ..]

      :: DONE -> break;

     od;
RECEIVED: skip
/* endloop */

od;
```

1.5.1 Formal Verification on TMR-PCU

In this section we list some of the properties verified for the TMR-PCU model, and the most meaningful results. Again we postpone the discussion about how we were able to verify these properties coping the the state explosion problem, till Section 1.6. The interesting properties in this context can be described informally as follows:

(PCU1) *Correctness of the communication protocols, in absence of Byzantine faults.*

The term *correctness* here means correctness of the diagnostic test and of the tournament algorithm run by a module. This property is verified by checking absence of deadlock. We slightly modify the PROMELA code of the PCUs in such a way as to force a peripheral unit to receive messages according to the intended behavior protocol. If the central module does not follow, or fail to follow, the protocol as the PCUs, the systems deadlocks.

(PCU2) *When two or more modules are active, each peripheral unit eventually receives exactly two messages in a single loop.*

(PCU2′) *In presence of Byzantine errors in one module, and when two or more modules are active, each peripheral unit eventually receives exactly two messages in a single loop.*

(PCU3) *When two or more modules are active, each peripheral unit eventually receives exactly two message via different busses in a single loop.*

(PCU4) *When two or more modules are active, each computer of every peripheral unit receives exactly one message in a single loop.*

All the previous properties are formulated with different LTL formulas with the following common structure:

$$\Box\, p \Rightarrow ((\Box \Diamond\, q) \wedge \Box\, q \Rightarrow (\Diamond\, r)))$$

here p, q and r are propositional formulas composed of predicates on variables. For example the property PCU2 (instantiated for the PCU called CDA1), is expressed by the LTL formula:

$$\begin{aligned}
&\Box\,(\texttt{global_activeA} + \texttt{global_activeB} + \texttt{global_activeC} >= 2) \Rightarrow \\
&\quad \big(\Box \Diamond\,(\texttt{CDA1[5]@is_receiving}) \ \wedge\ \Box\,(\texttt{CDA1[5]@is_receiving} \Rightarrow \\
&\quad\ (\Diamond\,(\texttt{CDA1[5]@has_received} \Rightarrow \\
&\quad\ (\texttt{recvA1_CDA1} + \texttt{recvB1_CDA1} + \\
&\quad\ \texttt{recvA2_CDA1} + \texttt{recvB2_CDA1} == 2))))\big)
\end{aligned} \tag{1.5.1}$$

Variables `global_activeA`, `global_activeB`, and `global_activeC` evaluate to 1 if and only if module A, module B, and module C respectively are active. The state labels `CDA1[5]@is_receiving` and `CDA1[5]@has_received` indicate that the CDA with identifier 5 is either running or has completed the communication phase with the central modules. Variables `recvA1_CDA1` and `recvA2_CDA1` (`recvB1_CDA1` and `recvB2_CDA1`, respectively) indicate the number of messages that computer A has received from bus 1 and from bus 2 (computer B has received from bus 1 and from bus 2, respectively). Informally formula 1.5.1 says that when al least two out of three central modules are active, the PCU called CDA1 is infinitely often in its receiving state and, whenever eventually the communication phase with central modules terminates, it has received exactly two messages.

1.5.2 Discussion

Figure 1.7 reports a summary of the results of the verifications, run in the presence of one Byzantine module. We briefly discuss the results for property PCU2′. We want to prove safety properties of the tournament algorithm in the hypothetic situation of a persistent Byzantine module. We prove that Byzantine behavior in the communication with the periphery phase makes the tournament algorithm fail. Analyzing the counter-example, we notice that three modules (and not two) send a message to the PCUs. With this result we underline the critical role of safety logic: if it fails to disconnect a Byzantine module before the tournament, this algorithm fails as well.

1.6 Abstraction and Implementation Strategies

The complexity of the ACC model forces us to introduce modularity techniques to cope with the state explosion problem. We proceed as follows:

1. by physically separating, in the PROMELA model, each phase in the ACC behavior, with the intention to use them as building blocks. In other words, we plan to develop the phases in separate files, to be included in main file representing the whole ACC model;

property	*state vector*	*depth*	*RAM*	*output*
PCU1	352	44047	60	success
PCU2	284	25465	23	success
PCU2′	284	1295	33	fail
PCU3	284	25465	23	success
PCU4	288	449467	33	success

Figure 1.7: Summary of the verification results on TMR-PCU. These results are obtained by running SPIN with MA+CO options selected, and without Byzantine modules (with the only exception of PCU2'). In the first column we have the property's name, in the second the dimensions of the state vector (in byte), in the third the depth of the search (in number of steps), in the fourth the total memory needed to the terminate the verification (in Mbytes), and in the last the result of the verification.

2. by modeling each building block representing a *communication* phase, both in a *correct* and in a *Byzantine* version;

3. by modeling each building block representing a correct or a Byzantine communication both in a *concrete* and in an *abstract* version.

In the Byzantine (versus the correct) version we implement the Byzantine version of the send. In this way we: (a) can control the state space growth of the whole model by incrementally inserting Byzantine phases, which introduce more non-determinism than the corresponding correct phase; (b) can test the robustness of the system in the presence of some particular Byzantine phases and not in the presence of a widely distributed, less realistic, Byzantine behavior.

In the concrete (versus the abstract) version, we model communication with the maximum parallelism: that is what happens in the real system. On the contrary, in the abstract version we impose a total order the communication events. For example, module A sends and receives first from B and then from C; module B receives and sends first to A and then sends and receives from C; finally module C first receives from A and from B first, and then sends to A and to B. By building a modular model we obtain an acceptable degree of scalability. In this case, scalability refers to abstract versus the concrete implementations and with respect to certain properties decided in accordance with ASF. We prove invariant properties both in concrete and abstract versions. These properties express fundamental invariants on the communication phases among internal modules composing the ACC. These properties can be informally described as follows:

(P1) before starting a communication phase, at least two out of three modules are active;

(P2) after a communication phase, each module has sent a message to all the other active modules;

(P3) after a communication phase, from all the other active modules, a module has either received a message, or detected a time-out.

(P4) after a communication phase, if a module has detected a time-out while receiving from all other active modules, it will go in a safe shutdown state.

The properties, expressed as PROMELA assertions, were shown to be satisfied by using SPIN, on both the concrete and abstract models. This was a sufficient condition (we agree with ASF) for

not loosing information when substituting, in the model, a concrete phase with the corresponding abstract version.

The correct and the concrete versions, with respect to the Byzantine and the abstract implementations, have different impacts on the state space. The correct version has less non-determinism than the Byzantine version; the abstract version eliminates all non-determinism in the communication events. By an appropriate composition of different versions in the whole system model, we obtain a large set of models at different abstraction levels (see Figure 1.8): we checked safety properties introduced in Section 1.4.1 and Section 1.5.1 by varying the number of the Byzantine phases inserted in the model. In addition, whenever the state dimension started to become problematic for our computational resources, we preferred the abstract over the concrete implementation of some, or all, the phases. In this way we executed a wide set of verification runs. For example, the results reported in Figure 1.6 have been performed by considering a module with one Byzantine phase in its abstract implementation, whereas those in Figure 1.7 have been run with all the communication phases in their abstract versions.

1.7 Conclusions

The project described in this chapter consists in verifying safety properties of a model of a safety-critical control system in presence of Byzantine behavior of one of its components.

In the context of the project that motivates this valitation work, we report that some of errors we have found fulfill the expectation of ASF; some other confirme what ASF has discovered with traditional techniques (*i.e.*, code inspection, testing). Moreover, the great flexibility and high expandability of formal models has helped us during almost all the steps of the project, when we have been able to enriched our models, with respect to the initial requirements, at a very low time and resources cost.

On the basis of this project an assessment of the application of the tool we used to support formal specification and verification process has been made. For what concerns the language PROMELA, we already underlined its suitability and expressive power in describing this type of distributed system. The only disadvantage we have found was the absence of any automatic management of termination of processes, that obliged us to model ad hoc time-outs as an active communication with heavy repercussions on the size of the state space. In fact, we need to formalize a shutdown as an active behavior; a shutdown module does nothing but participates in all the communication phases by sending empty messages to cause time-outs.

Regarding the tool SPIN the most important fact to be underlined is related to strategies dealing with the state explosion problem. In particular, the use of a minimized automaton encoding technique (`MA`) combined with the state compression option (`COLLAPSE`) turns out to be useful in helping with out-of-memory problems, but at the cost of a long execution time. Most verifications, due to the large state space size required the use of both optimization strategies.

As an example, Figure 1.9 contains representative data, concerning a verification on a 256 Mbyte RAM Pentium II - Linux Suse 5.3 - for a system model whose complete description required 348 bytes per state; in the figure memory and time resources have been compared by using, respectively, the `COLLAPSE` (for which we ran out-of-memory, with the longest depth-first search path containing 15125 transitions from the initial state) and the `COLLAPSE` + `MA` options (for which we have successfully terminated the verification, with longest depth-first search of 15916).

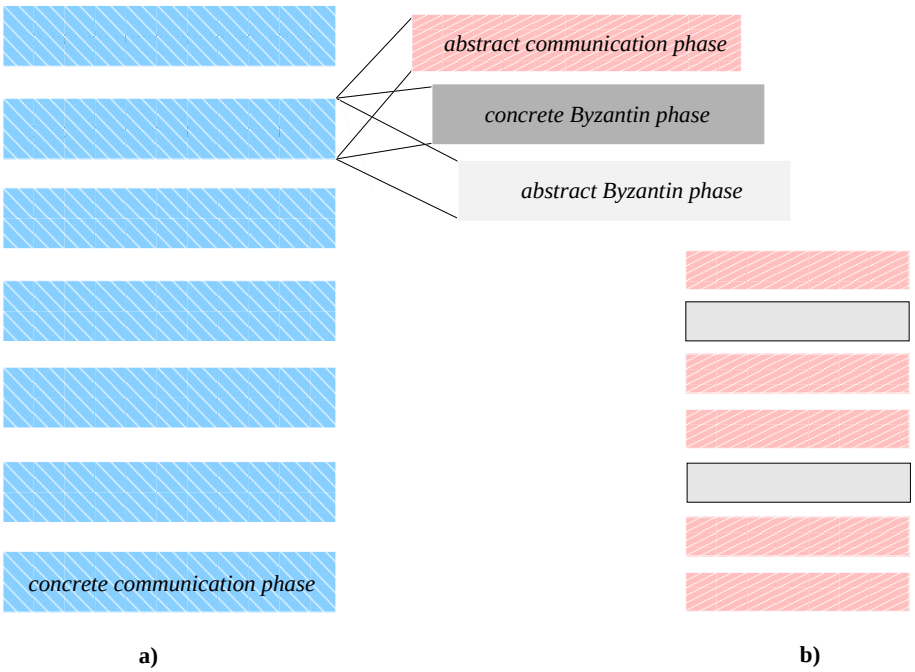


Figure 1.8: a) The framework in which we develop abstract/concrete and Byzantine/correct model. b) One of the model we used in TMR-PCU verification.

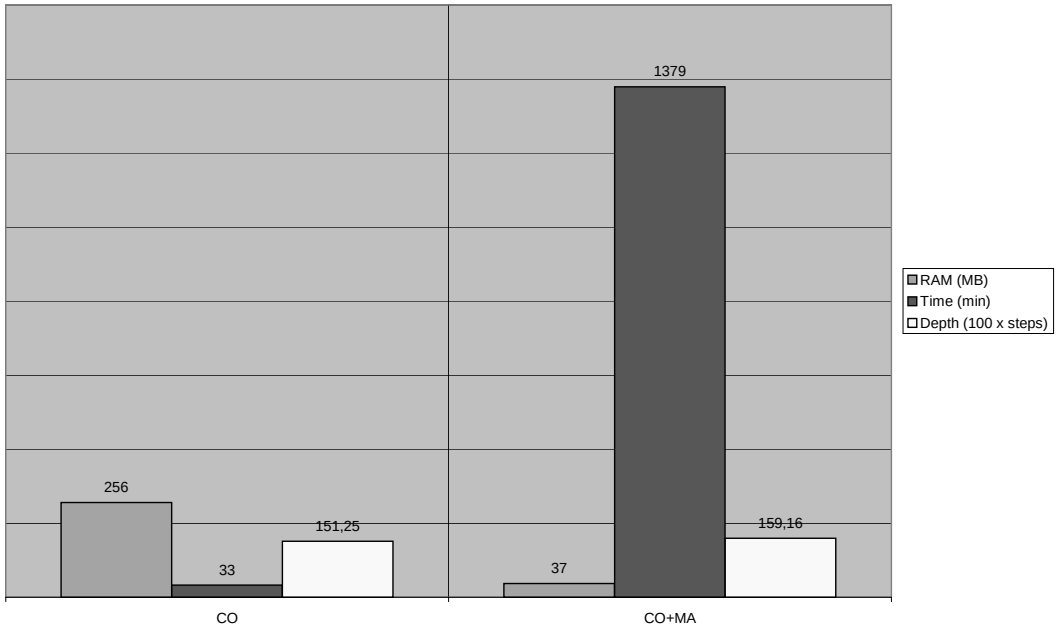


Figure 1.9: A representative example of memory versus time usage in our SPIN experiments, with CO and with CO+MA optimization strategies. On the left we report the RAM (in Mbytes) and the time (in minutes) and the depth (in hundreds of steps) reached in a verification that ran out-of-memory. In this verification only the CO option is used when compiling the model checker. On the right, we report the same data for the same verification with the CO+MA compiler option enabled. In this case, a significant reduction of memory makes the verification end without running out of memory, at the cost of a considerable increase of the running time.

2

Validation of Security Protocols, a Test Case: Security Analysis of the OSA/Parlay Framework

"Al solito, quelli dei telefoni tiravano a praticare l'assurdo. Dicevano, per esempio: il numero da lei chiamato è inesistente ... Ma come si permettevano un'affermazione accussí? Tutti i nummari che uno arrinisciva a pinsari erano esistenti. Se veniva a fagliare un nummaro, uno solo nell'ordine infinito dei nummari, tutto il mondo sarebbe precipitato nel caos. Se ne rendevano conto quelli dei telefoni, sì o no?" (Salvo Montalbano in La pazienza del Ragno, *A. Camilleri, 2003)*

"As usual those of the phone company were talking nonsense. They say, for instance: the number you're calling doesn't exist ... How could they dare to make such a statement? All the numbers one can think of must exist. If only one number was missing, one of them in the infinite sequence of numbers, the whole world would fall into chaos. Aren't those of the phone company aware of this?"

Abstract

This chapter reports on an experience in analyzing the security of the Trust and Security Management (TSM) protocol, an authentication procedure within the OSA/Parlay Application Program Interfaces (APIs) of the Open Service Access and Parlay Group. The experience has been conducted jointly by research institutes, experienced in security, and an industry expert in telecommunication networking. OSA/Parlay APIs are designed to enable the creation of telecommunication applications outside the traditional telecommunication network space and business model. Network operators consider the OSA/Parlay architecture promising in stimulating the development of web-service applications by third party providers that are not necessarily expert in telecommunications. The TSM protocol is executed by the gateways to OSA/Parlay networks; its role is to authenticate the client applications that try to access the interfaces of some object representing an offered network capability. For this reason potential security flaws in the TSM authentication strategy can lead to unauthorized use of network with evident damages to the operator and to the quality of service. This chapter reports on a rigorous formal analysis of the TSM specification originally given in UML; it reports on the design activity of the formal model, the tool-aided verification performed, and the security flaws discovered. This will allow us to discuss how the security of the TSM protocol can be generally improved.

2.1 Introduction

OSA/Parlay [1] Application Program Interfaces (APIs) [108] are designed for an easy interaction between traditional IT applications and telecommunication networks. OSA/Parlay APIs are abstract

[1] See `http://www.parlay.org`

building blocks of network capabilities that developers, not necessarily expert in telecommunications but perhaps with more expertise in the enterprise market, can quickly comprehend and use to generate new applications. Concisely, OSA/Parlay APIs proposes an attractive framework where programmers can develop innovative resources or design new services.

An example of such a service is the retrieval and purchase of goods via a mobile phone. The service could be provided by a third party provider, different from the mobile operator. In this case, the provider could develop the service by assembling components that control network capabilities and functions, *e.g.*, sending/receiving a SMS. These components are furnished by the telecoms operator in particular their APIs. For example, the sending/receiving of a SMS could be realized in the following SOAP body that, in XML notation where namespace and encoding descriptors are omitted, appears as follows:

```
<sendSMS>
 <dest_address>
   tel:1234567
 </dest_address>
 <send_address>
   tel:0123456
 </send_address>
 <message>
   Could you please reserve
   two seats for 9 o'clock?
 </message>
</sendSMS>
```

OSA/Parlay APIs can also be used in the development of new web-based services. The Parlay community has designed particular APIs, called Parlay X APIs, based on web service principles and oriented to the Internet community.

When network resources are broadly accessible, it becomes crucial to define and enforce appropriate access rules between the entities that offer network capabilities and the service suppliers, so that an operator can maintain full control over the usage of her resources and on the quality of service. For instance, it is important that the use of services is guided by a set of rules defining the supply conditions and the reciprocal obligations between the client and the network operator. Service Level Agreements (SLAs) are commonly used to formalize a detailed description of all the aspect of the deal. To avoid that unauthorized entities can sign an agreement and use the network illegally, on-line authentication checks are of primary importance.

Authentication, in a distributed setting is usually achieved by the use of cryptographic protocols. Experience teaches that such protocols need to be carefully checked, before being fielded. Formal methods have been profitably applied in the verification of many security or authentication protocols (e.g., [5, 53, 104, 138, 152, 177, 182]), and nowadays developers have access to libraries of reliable protocols for different security goals. For example the Secure Socket Layer (SSL) by Netscape, is widely used to ensure authenticity and secrecy in Internet transactions. Unfortunately, the use of reliable, plugged-in, protocols is not sufficient to ensure security, just like the use of reliable cryptography is not sufficient to ensure secrecy in a communication. In this ambit formal methods help to validate the correct use of security procedures.

In this chapter we discuss a validation experience whose aim is to analyze formally the authentication mechanism in the Trust and Security Management protocol in OSA/Parlay APIs [1]. As a result of the analysis we propose an improvement concerning its security. This protocol is

designed to protect telecommunication capabilities from unauthorized access and it implements an authentication procedure. TSM is specified in the Unified Modeling Language (UML) [171], where its composing messages, its interfaces towards the client and the server, and the methods implementing security-critical procedures, are described at different levels of abstraction. The formal validation experiment, conducted within a joint project between research Institutes and Telecom Italia Lab, has revealed some security flaws of the authentication mechanism. From the analysis of the traces showing the attacks, we were able to suggest possible solutions to fix the security weaknesses discovered, and to state a general principle of prudent engineering (in the style of [8]) for improving the security in web-service applications.

2.2 The OSA/Parlay Architecture

The OSA/Parlay architecture enables service application developers to make use of network functionality through an open standardized interface. OSA/Parlay APIs [1] provide an abstract and coherent view of heterogeneous network capabilities, and they allow a developer to interface its applications via distributed processing mechanisms. The OSA/Parlay architecture, shown in Figure 2.1, consists of:

- a set of *Client Applications* accessing the network resources;
- a set of *Service Interfaces*, or Service Capability Features (SCFs), that represent interfaces for controlling the network capabilities provided by network resources (*e.g.*, controlling the routing of voice calls, sending/receiving SMSs, locating a terminal, etc.);
- a *Framework*, that provides a modular and "controlled" access to the SCFs.
- *Network Resources*, in the telecommunication network, implementing the network capabilities.

A *Parlay Gateway* includes the framework functions and the Service Capability Services (SCSs) *i.e.*, the modules implementing the SCFs: it is a logical entity that can be implemented in a distributed way across several systems. Since the target applications could be deployed in an administrative domain different from the one of the Parlay Gateway, the secure and controlled access to the SCFs is a predominant aspect for the Parlay architecture. To get the references of the required SCFs, an application must interact several times with the framework interfaces. For example, the application must carry out an authentication phase before selecting the SCFs required, as described in Section 2.2.1. In this phase the framework verifies whether the application is authorized to use the SCFs, according to a subscription profile. Finally, an agreement is digitally signed, and the framework gives to the application the references to the required SCFs (*e.g.*, as CORBA interface reference). These references are valid only for a single session of the application. When the framework has to return an SCF reference to an application, it contacts the SCS which implements it, by passing all the configuration parameters *e.g.*, the Service Level Agreement conditions, stored in the subscription profile of the application. The SCS creates a new instance of the SCF, configured with the received parameters, and returns its reference to the framework. Each time the application invokes a method on the SCF instance, the SCS executes it by taking into account the configuration parameters received at instantiation time.

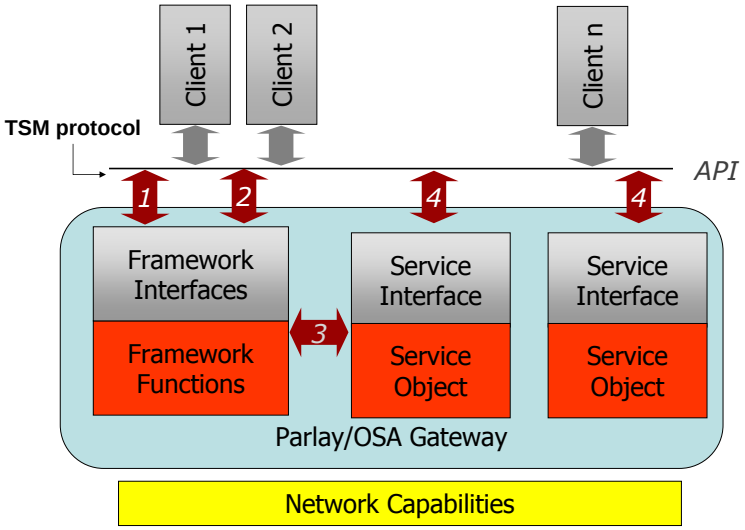


Figure 2.1: The OSA/Parlay Architecture. The Trust and Security Management protocol runs between the Framework Interfaces and the Clients.

2.2.1 Trust and Security Management protocol

One of the critical steps for guaranteeing controlled access to the SCFs is the authentication phase between the gateway and the application. It is supported by the protocol implemented by the Trust and Security Management (TSM) API. We focuses on the analysis of the properties of this security protocol, whose behavior is summarized by the message sequence chart in Figure 2.2. The main steps of the protocol are:

- Initiate Authentication: the client invokes the method "`initiateAuthenticationWithVersion`" on the framework's *public* interface (*e.g.*, an URL) to initiate the authentication process. Both the client and the framework provide a reference to their own access interfaces.

- Select Authentication Mechanism: the client invokes the method "`selectAuthenticationMechanism`" on the framework authentication interface, to negotiate which hash function will be used in the authentication steps.

- The client and the framework authenticate each other. The framework could authenticate the client before (or after) the client authenticates the framework, or the two authentication processes could be interleaved. However, the client shall respond immediately to any challenge issued by the framework, as the framework might not respond to any challenge issued by the client until the framework has successfully authenticated the client. Each authentication step is performed following a one-way Challenge Handshake Authentication Protocol (CHAP) [133] *i.e.*, by issuing a challenge in the "challenge" method, and checking if the partner returns the correct response. An invocation of the method "`authentication succeeded`" signals the success of the challenge.

- Request an access session: when authenticated by the framework, the client is permitted to invoke "`requestAccess`" to start an access session. The client provides a reference to its own Access interface, and the framework returns a reference to Access interface, unique for this client.
- The access interface is used to negotiate the signing algorithm to be used in the session and to obtain the references to other framework interfaces (we will call them, *service framework interfaces*), such as service discovery and service agreement management.

Having obtained the reference to a service framework interface the TSM finishes. Note that the references to the interfaces must remain secret: if an intruder got hold of them, it would be able to (abusively) access the services. For this reason our analysis will mainly concentrate on the secrecy of these references. In fact, after the TSM ends, the client selects the required SCFs by invoking the method "`selectService`" on the service agreement management interface. The client obtains a service token, which can be signed as part of the service agreement by the client and the framework, through the "`signServiceAgreement`" and the "`signAppService Agreement`" methods. Generally the service token has a limited lifetime: if the lifetime of the service token expires, a method receiving the service token will return an error code. If the sign service agreement phase succeeds, the framework returns to the client a reference to the selected SCF, personalized with the client configuration parameters.

2.3 Formal Security Analysis

This section explains in detail the formal analysis of the security of the TSM protocol that we have done. To carry out the verification phase we used CoProVe [59] a constraint-based system for the verification of cryptographic protocols [2]. CoProVe has been developed at the University of Twente (NL); it is an improved version of the system designed by Millen and Shmatikov [156]. CoProVe is based on the strand spaces model [77]; it enjoys an efficient implementation, a monotonic behavior which allows to detect flaws associated to partial runs, and an expressive syntax in which a principal may also perform explicit checks for deciding whether to continue or not with the execution. All these features make CoProVe quite efficient in practice. The intruder model is that of Dolev-Yao [69], where the malicious entity is identified with the communication infrastructure. Protocols are written in Prolog-lake style, and properties are expressed as reachability predicates. In case a security flaw is discovered, CoProVe can show one or all the traces showing the attack.

2.3.1 Modeling Choices

One of the challenges in applying tools of automatic analysis to industrial architectures lies in translating the (usually less formal) specification into a rigorous formal model. In our experience, translating a complex system design into a formal protocol specification involves many non-trivial steps: software technology concepts such as method invocation and object interfaces have to be "encoded" into an algebraic protocol specification. This encoding phase also forces the engineer to reason about the security implication of using these constructs.

The OSA/Parlay framework APIs specification consists of many pages of UML specification; at this level of abstraction it is difficult to have a good overview of its security mechanisms. In the APIs specification, for instance, there is no explicit transmission of messages: the exchange

[2] Freely accessible via the web at `http://wwwes.cs.utwente.nl/24cqet/coprove.html`

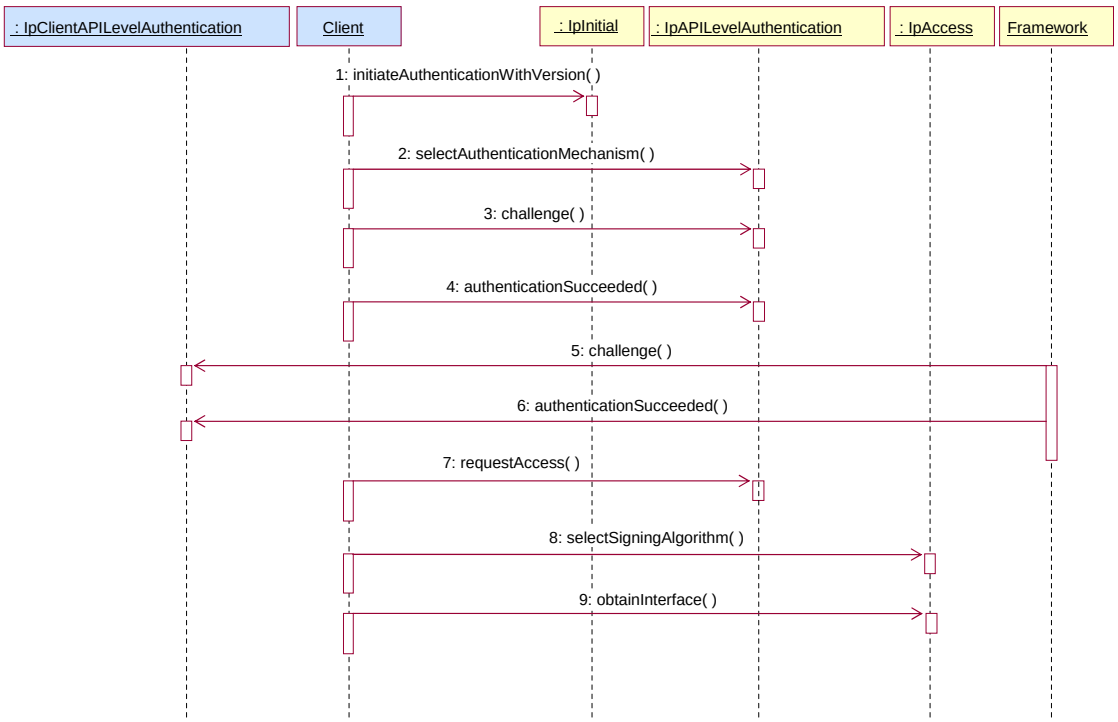


Figure 2.2: Message sequence chart describing the steps of the TSM protocol [1]

of one (sometimes even more) messages happens exclusively by the mechanism "invocation of a method over an object interface". Moreover, different levels of abstraction are mixed: for example, the same mechanism of "method invocation" is used both to describe, in one step, the whole set of critical steps of the CHAP handshake and the single message starting of the protocol. More critically, "method invocation" does not specify the confidentiality of the input/output parameters involved. Innocent acknowledgment messages are treated in the same way as references to confidential object interfaces.

The application of clear modeling choices encourages the design of a formal model without the previous ambiguities. In translating the TSM specification in a model we define and apply the following modeling choices.

Modelling Choice 1 *A reference to a (new) private interface, F, is modeled by a (new) shared encryption key, KF.*

Choice 1 reflects the fact that an intruder who does not know the private interface reference cannot infer anything from any method invocation over that interface. This simple, but essential observation will make our security analysis straightforward, as we explain in Section 2.3.

Modelling Choice 2 *Calling a method, with parameter M, over a private interface F is modeled as sending the message $\{M\}_{KF}$* i.e., *, M encrypted with KF. Dually, getting the result is translated as receiving a message encrypted with the same KF;*

In Choice 2 we treat a reference to an object interface as a communication port; consequently calling a method equates transmitting a message through that port. Moreover, we model the transmission of a message through F, as the transit of a message encrypted with the key KF. In other words, calling a method over an interface is modeled as a communication encrypted with the interface key. This choice reminds of an observation by Abadi and Gordon [7], who suggest the use of cryptographic keys to model mobility. Our situation is indeed much simpler: the only form of "mobility" we have, is the dynamic creation of a "channel", *i.e.*, an interface reference.

2.3.2 Formal Models

We apply Choices 1 and 2 to design the TSM formal *abstract* model written in the usual representation of cryptographic protocols. The obtained model is as follows:

* initiate *		* request access *	
step 1.	$C \longrightarrow F : C, KC$	step 8.	$C \longrightarrow F : \{req\}_{KF}$
step 2.	$F \longrightarrow C : KF$	step 9.	$F \longrightarrow C : \{KA/fail\}_{KF}$
* select authentication methods *		* select signing methods *	
step 3.	$C \longrightarrow F : \{[h, h', h'']\}_{KF}$	step 10.	$C \longrightarrow F : \{[s, s', s'']\}_{KA}$
step 4.	$F \longrightarrow C : \{h\}_{KF}$	step 11.	$F \longrightarrow C : \{s\}_{KA}$
* challenge *		* request for service interface *	
step 5.	$F \longrightarrow C : \{F, N\}_{KC}$	step 12.	$C \longrightarrow F : \{req'\}_{KA}$
step 6.	$C \longrightarrow F : \{C, h(N, SCF)\}_{KC}$	step 13.	$F \longrightarrow C : \{KS/fail\}_{KA}$
step 7.	$F \longrightarrow C : \{ok/fail\}_{KC}$		

In this abstract model, C represents a client and F the framework, while $C \longrightarrow F : M$ denotes C sending message M to F. With $\{M\}_K$ we indicate the plain-text M encrypted with a key K, while $h(M)$ denotes the result of applying a hash function h to M. In step 1 the client initiates the protocol over the public interface of the framework, by providing its name and a reference to its interface, KC. In step 2 the framework replies by sending a reference, KF, to its own interface.

Remark 2.3.1 It may seem odd that despite modelling choice we transmit references to interfaces (represented as keys) in clear. The expectation here is that the challenge response protocol of steps 5-7 would avoid intrusion anyway. ■

In steps 3 and 4 the client asks the framework to choose an authentication method among h, h' and h''. In steps 5 and 6 the actual CHAP protocol is carried out, using the hash function selected in step 4. Here, SCF represents a shared secret between C and F, required by CHAP [133]. Indeed the UML specification did not provide the details about the CHAP implementation; here we use the version of CHAP where the client and the framework already share the secret SCF. In steps 8 and 9 the client asks for an interface where to invoke the request access for a service. In

steps 10 and 11 the framework chooses the interface. Finally in steps 12 and 13 the client sends a request for a service and receives back the reference to the relative framework interface.

The abstract model has been translated into the language required by CoProVe. The result of this translation is a *concrete* formal model; in addition, we encode (in the language of CoProVe) the security properties that we want to check. In Figure 2.3 we report one of the concrete models we used for checking whether KA remains secret or not.

The specification in Figure 2.3 involves three principals: one client (`c`), one framework (`f`) and eavesdropping agent (`sec`). Each role is specified by a sequence of send or receive actions that mimic exactly the steps of the abstract model. Symbol "`+`" is used to denote symmetric encryption using shared keys. Formal parameters (*e.g.*, in the client role `C,F,Kc,Kf,N,Req,Ka,Scf` are used to denote all the objects used in the role specification. In a scenario these parameters are instantiated with actual constants representing real objects (*i.e.*, `c,f,_,kf,n,_,ka,scf`). Here "`_`" is used when no instantiation is required, that is when a free variable is involved. The intruder is assumed to know only the client and framework names plus its own name "`e`". Verification of secrecy consists in asking if there is a trace leading the eavesdropper to know a secret.

2.3.3 Formal Analysis and Detected Weakness

The analysis performed on the model of TSM protocol, pointed out weaknesses in the security mechanism. In the following we will describe the flaws discovered as a commented list of items. Where significant, we show the output produced by CoProVe and we interpret the output.

Flaw 1. An intruder can impersonate a client and start an authentication challenge with the framework.

An intruder can obtain the reference to the interface used by the client to start the authentication challenge (key `kf`). This happens, unsurprisingly, because the reference `kf` is transmitted in clear, as the following trace of CoProVe confirms:

```
1.  [c,send([c,kc])]
1'. [f,recv([c,kc])]
2.  [c,recv(_h325)]
2'. [f,send(kf)]
```

Each row represents a communication action. For example, `c,send[c,kc]` represents the action "send" that "`c`" executes with message "`[c,kc]`"; `c,recv(_h325)` represents the results of a "receive" where the client "`c`" receives the name (in this case generated by the intruder) "`_h325`". The sequence of actions reveal the attack. It can be visualized in the conventional notation of security protocol (where, we also write `_h325` as KE, the intruder key, because this is its understood meaning.):

$$\begin{array}{llll} 1. & C \longrightarrow I(F) : C, KC & 2. & I(F) \longrightarrow C : KE \\ 1'. & I(C) \longrightarrow F : C, KC & 2'. & F \longrightarrow I(C) : KF \end{array}$$

This run comprises two parallel runs of the protocol, in which the intruder plays, respectively, the role of the client against the framework ($I(C)$ in *steps* $1'$ and $2'$) and the framework against the client ($I(F)$ in steps 1 and 2).

```
% Initiator role specification
client(C,F,Kc,Kf,N,Req,Ka,Scf,[
   send([C,Kc]),
   recv(Kf),
   recv([F,N]+Kc),
   send([C,sha([N,Scf])]+Kc),
   send(Req+Kf),
   recv(Ka+Kf)]).

% Responder role specification
framewk(C,F,Kc,Kf,N,Req,Ka,Scf,[
   recv([C,Kc]),
   send(Kf),
   send([F,N]+Kc),
   recv([C,sha([N,Scf])]+Kc),
   recv(Req+Kf),
   send(Ka+Kf)]).

% Secrecy check
%(it is a singleton role)
secrecy(N, [ recv(N) ] ).
```

```
% scenario specification
% pairs [name, Name]
% [label for the role; actual role]
scenario
([[c,Client1],
  [f,Framew1],
  [sec,Secr1]]):-

client(c,f,kc,_,_,req,_,scf,Client1),
framewk(c,f,_,kf,n,_,ka,scf,Framew1),
secrecy(ka, Secr1).

% The initial intruder knowledge
initial_intruder_knowledge([c,f,e]).

% specify which roles we want
% to force to finish
%(only sec in this example)
has_to_finish([sec]).
```

Figure 2.3: The "CoProVe" specification (in two columns) used to check the secrecy of KA. To reduce the search space here we implemented only steps 1-2, 5-6 and 8-9. In other words we assumed: (a) a fixed hashing function h; (b) that the framework does not reply (instead of replying "false") if the client answer wrongly to the CHAP challenge.

This flaws is not serious in itself (provided the authentication procedure is able to detect an intruder and close the communication), but it becomes serious when combined with the next weaknesses in the security; by knowing `kf` an intruder is able to grab other confidential information.

Flaw 2. An intruder can impersonate a client, authenticate itself to the framework and obtain the reference to the interface used to request access to a service (key `ka`).

This is a serious flaw that compromises the main goal of the protocol itself. Informally, a malicious application can pass the authentication phase instead of an honest client, and it can obtain a reference to the interface used to request a service (key `ka`). The study of the output of CoProVe shows the existence of an "oracle" attack, where the intruder uses the client to get the right answer to the challenge:

```
1.  [c,send([c,kc])]
1'. [f,recv([c,kc])]
2.  [c,recv(_h325)]
2'. [f,send(kf)]
5'. [f,send([f,n] + kc)]
5.  [c,recv([f,n] + kc)]
6.  [c,send([c,sha([n,scf])] + kc)]
6'. [f,recv([c,sha([n,scf])] + kc)]
8.  [c,send(req + _h325)]
```

```
9.  [c,recv(_h391 + _h325)]
8'. [f,recv(req + kf)]
9'. [f,send(ka + kf)]
    [sec,recv(ka)]
```

Using the standard informal notation for describing protocols, the above trace is read as follows:

1.	$C \longrightarrow I(F) : C, KC$	6.	$C \longrightarrow I(F) : \{C, h(N, SCF)\}_{KC}$
1'.	$I(C) \longrightarrow F : C, KC$	6'	$I(C) \longrightarrow F : \{C, h(N, SCF)\}_{KC}$
2.	$I(F) \longrightarrow C : KE$	8.	$C \longrightarrow I(F) : \{req\}_{KE}$
2'.	$F \longrightarrow I(C) : KF$	9.	$I(F) \longrightarrow C : \{fail\}_{KE}$
5'.	$F \longrightarrow I(C) : \{F, N\}_{KC}$	8'.	$I(C) \longrightarrow F : \{req\}_{KF}$
5.	$I(F) \longrightarrow C : \{F, N\}_{KC}$	9'.	$F \longrightarrow I(C) : \{KA\}_{KF}$

This run comprises two parallel runs of the protocol, in which the intruder plays, respectively, the role of the framework against the client and the role of the client against the framework.

Searching among the set of attacks returned by CoProVe, we find also the following, straightforward, man-in-the-middle, attack:

```
1.  [c,send([c,kc])]
1'. [f,recv([c,kc])]
2'. [f,send(kf)]
2.  [c,recv(kf)]
5'. [f,send([f,n] + kc)]
5.  [c,recv([f,n] + kc)]
6.  [c,send([c,sha([n,scf])] + kc)]
6'. [f,recv([c,sha([n,scf])] + kc)]
8.  [c,send(req + kf)]
8'. [f,recv(req + kf)]
9'. [f,send(ka + kf)]
9.  [c,recv(_h325)]
    [sec,recv(ka)]
```

This trace shows that the intruder can eavesdrop first the key `kf`, passed in clear, and then steal the message `ka+kf`. At this point key `ka` can be obtained by a simple decryption. This attack is obviously straightforward at this point of the analysis, but it became clear as soon as we applied Choice 1.

Flaw 3. An intruder can impersonate a client, authenticate itself to the framework, send a request for a service and obtain the reference to a service framework interface (key `ks`).

This is also a serious flaw that compromises the main goal of the protocol. An intruder can obtain the reference to a service framework interface (key `ks`). It is easy to understand, that this is possible, for example, as a consequence of flaw 1 and 2: once an intruder has authenticated itself instead of the client, it can easily obtain the reference.

Further checks with CoProVe, show that the intruder can even retrieve this reference with a man-in-the-middle attack *e.g.*, by listening to the communication between the client and the framework and stealing the reference when it is passed in clear. In our model this attack can be

explained as follows: the intruder intercepts, by eavesdropping, the message $\{KS\}_{KA}$ and it decrypts it. This is possible because the encryption key KF is passed in clear and, by eavesdropping, the intruder can easily obtain $\{KA\}_{KF}$, and hence KA (see Flaw 2).

Flaw 4. An intruder can force the framework to use an authentication mechanism of her choice.

This flaw has been discovered using the specification in Figure 2.4, with two instances of the framework. When a client offers a list of authentication methods, the first instance selects the first method at the head of a list (here consisting of only two items), whereas the second instance chooses the second. In this way we model different choices made by the framework.

The attack is shown by the following CoProVe trace; an intruder can force the framework to select a particular authentication mechanism, by the use of a replay attack.

```
a.1.  [c,send([c,kc])]
a.1'. [f,recv([c,kc])]
a.2.  [c,recv(_h320)]
a.2'. [f,send(kf)]
a.3.  [c,send([a1,a2] + _h320)]
a.3'. [f,recv([a1,a2] + kf)]
a.4'. [f,send([a1,a1] + kf)]
a.4.  [c,recv([a1,a1] + _h320,)]
a.5'. [f,send([f,n] + kc)]
a.5.  [c,recv([f,n] + kc)
a.6.  [c,send([c,sha([n,scf])] + kc)
a.6'. [f,recv([c,sha([n,scf])] + kc)
a.8.  [c,send(req + _h320)]
a.9.  [c,recv(req + _h320)]
a.8'. [f,recv(_h404 + kf)]
a.9'. [f,send(ka + kf)]
b.1'. [f,recv([c,_h487])]
b.2'. [f,send(kf2)]
b.3'. [f,recv([a1,a1] + kf2)]
b.4'. [f,send([a1,a1] + kf2)]
b.8'. [f,recv(_h488 + kf2)]
b.9'. [f,send(ka2 + kf2)],
      [sec,recv(ka2)]
```

The attack can be represented in the following abstract steps:

a.1	$C \longrightarrow I(F1) : C, KC$	*a.4'*	$F1 \longrightarrow I(C) : \{[h1]\}_{KF}$
a.1'	$I(C) \longrightarrow F1 : C, KC$		$[\ldots]$
a.2	$I(F) \longrightarrow C : KE$	b.1'	$I(C) \longrightarrow F2 : C, KE$
a.2'	$F1 \longrightarrow I(C) : KF$	b.2'	$F2 \longrightarrow I(C) : KF2$
a.3	$C \longrightarrow I(F1) : \{[h1, h2]\}_{KE}$	b.3'	$I(C) \longrightarrow F2 : \{[h1, h1]\}_{KF2}$
a.3'	$I(C) \longrightarrow F1 : \{[h1, h2]\}_{KF}$	b.4'	$F2 \longrightarrow I(C) : \{[h1]\}_{KF2}$

In the trace the intruder acts as a man-in-the-middle in a communication between the client and the first instance of the framework $F1$ and it learns what method the framework is able to use

(sequences $a.i$). In the second run, the intruder acts as a client, and it offers to the second instance of the framework $F2$ the choice that the framework is able to accept (sequences $b.i$). The structure of the attack is such that it can be applied also for forcing the selection of a signing methods *i.e.*, steps 10 and 11 of the abstract model.

2.4 Discussion

The analysis performed so far shows some weaknesses of the protocol, and gives also useful indications on how to improve the robustness of the protocol. This section discusses the weaknesses here presented, and suggests possible solutions to increase the overall security. We start with some preliminary considerations.

The security weak is because some references to interfaces are passed in the clear. This is because the role of those references has been misunderstood, or under-evaluated, or more probably not recognized in the UML, high-level, object specification. A rigorous, synthetic, formal specification and precise modeling choices help in giving each object its right role. In our case we were able to identify in the role of some references to object interface the same role that session keys have. This observation can be quoted as a principle:

> Independently of their high-level representation, data that directly or indirectly gives access to a secret, must be thought of (hence, modeled) as encryption keys.

This principle plays a role also in fixing the protocol. In fact, the common practice in protocol engineering [8] suggests the use of (other) session keys to protect the confidentiality of sensitive information, which in the case of TSM are the references to interfaces.

According to this model, session keys are indeed missing completely from the present implementation[3], while their use could prevent the intruder from gaining a reference to an interface (as shown, by a man-in-the-middle attack). Note that unfortunately it is not sufficient to establish a session key during the challenge phase. In this case, Flaw 2 remains intact, as confirmed by CoProVe. This implies that the structure of the protocol needs to be globally reviewed. An additional point of discussion concerns the correct use of a CHAP-based authentication. From the OSA/Parlay documentation [1] we read that security can be ensured if the "challenge" is frequently invoked by the framework to authenticate the client that, in turn, must reply "immediately":

> However, *the client shall respond immediately to any challenge issued by the framework*, as the framework might not respond to any challenge issued by the client until the framework has successfully authenticated the client" ([1], page 19)

Our analysis proves that not only the intruder can act as a client with respect to the framework, but also that it can passively observe, as man-in-the-middle, the framework and a client authenticating each other as many times as they want, and then steal the references to the service framework interfaces when they are transmitted in clear. At this point the intruder can substitute itself for the client.

Flaw 4 is different in nature, and it teaches that particular care must be paid to the choice of the encryption algorithms or digital signature procedures offered by the framework: for example, the intruder can force the system to use the encryption algorithm that is easier to crack.

[3] Do not confuse them with the session keys that appear in the abstract model. Those are part of the model and represent private references to interfaces.

2.5 Conclusions

This chapter discusses an industrial experience of formal analysis applied to the security aspects of the OSA/Parlay Trust and Security Management protocol. The protocol is devised to authenticate the clients before giving them access to the network services. Our experience confirms that formal methods are an invaluable tool that can discover serious security flaws that may be overlooked otherwise. This is true in two respects. First, the use of a formal model, where only the relevant security features are expressed, helps in pointing out what are the critical parts for security. In an informal description, on the other hand, this information is usually dispersed and difficult to gather. Second, the use of an automatic tool allows us to identify dangerous man-in-the middle attacks, which are notoriously difficult to detect in high-level specifications.

From this experience, conducted within a joint project between industry and research institutes, we state a general principle for security in web-services: it is essential to identify clearly the security role of each object involved in service specification. It is vital especially for those objects that abstractly represent encryption keys. This principle helps in simplifying the security analysis. With the application of this principle we discover serious weaknesses more easily, and we are able to discuss how the security of the TSM protocol can be generally improved.

```
% Initiator role specification
client(C,F,Kc,Kf,N,Req,Scf,
       Ka,A1,A2,A,[
   recv([C,F]),
   send([C,Kc]),
   recv(Kf),
   send([A1,A2]+Kf),
   recv([A,A]+Kf),
   recv([F,N]+Kc),
   send([C,sha([N,Scf])]+Kc)
   send(Req+Kf),
   recv(Ka+Kf)]).

% Responder role specification
framewk(C,F,Kc,Kf,N,Req,Scf,
        Ka,A1,A2,[
   recv([C,Kc]),
   send(Kf),
   recv([A1,A2]+Kf),
   send([A1,A1]+Kf),
   send([F,N]+Kc),
   recv([C,sha([N,Scf])]+Kc),
   recv(Req+Kf),
   send(Ka+Kf)]).

framewk2(C,F,Kc,Kf,Req,
           Ka,A1,A2,[
   recv([C,Kc]),
   send(Kf),
   recv([A1,A2]+Kf),
   send([A2,A2]+Kf),
   recv(Req+Kf),
   send(Ka+Kf)]).
```

```
% secrecy check (singleton role)
secrecy(N, [ recv(N) ] ).

% Scenario
scenario([
    [c,Client1],
    [f,Framew1],
    [f,Framew2],
    [sec,Secr1]
          ]) :-
client(c,f,kc,_,_,req,scf,_,a1,
        a2,_,Client1),

framewk(c,f,_,kf,n,_,scf,ka,_,_,
        Framew1),

framewk2(c,f,_,kf2,n2,_,ka2,_,_,
        Framew2),

secrecy(ka2, Secr1).

% Set up the intruder knowledge
initial_intruder_knowledge([c,f,e]).

% specify which roles we want
% to force to finish
% (only sec in this example)
has_to_finish([sec]).
```

Figure 2.4: The “CoProVe” code used to discover flaw 4 (in two columns). The model of the framework includes the “select authentication method” phases of the abstract model and implements steps 1–9 of the abstract model. Step 7 is omitted, *i.e.*, the framework does not reply (instead of sending “fail”) in case of failure of the challenge phase. The second instance of the framework models only steps 1–4 and steps 8–9, *i.e.*, those steps strictly necessary to discover the attack.

Part II

Analysis Techniques in Security and Fault-Tolerance

3

Techniques of Security Protocol Analysis in Fault-Tolerance

Fenesta cu sta nova gelosia tutta lucente de centrella d'oro, tu m'annascunne Nennella bella mia, lassamela verè sinnò me moro. (R. Murolo in La Nova Gelosia*, Serenata Napoletana del '700)*

Damn this new jalousie, shiny and with a golden chain, you hide beloved Nennella from my sight, please let me see her, or I will die.

Abstract

This chapter shows how fault-tolerance analysis can benefit from techniques of analysis developed for the study of security protocols. We use the CCS process algebra as a formal framework. We model the fault-tolerant system and its environment as two separate and interacting CCS processes, $P_{\mathcal{F}}^{\#}$ and F respectively. In $P_{\mathcal{F}}^{\#}$, we describe the system's failing behavior and its fault-recovering procedures. Faults are represented by reserved actions from a finite set $\mathcal{F}$. In F, we model the fault assumptions, that is the assumptions over the modalities of occurrence of faults; moreover, F is able to trigger fault actions in $P_{\mathcal{F}}^{\#}$ by interacting through the set of actions in $\mathcal{F}$. In the CCS, this framework has to the general form $(P_{\mathcal{F}}^{\#} \parallel F) \setminus \mathcal{F}$. From the point of view of the analysis, we study the fault-tolerance of $P_{\mathcal{F}}^{\#}$ with respect to a given property, when F is an unspecified component; in this case, the analysis can be made independent from any particular fault assumption, and the role of F can be compared with that played by the intruder in security protocol analysis. We restate in fault-tolerance two strategies of validation used in security protocols analysis. The first strategy consists in reducing the problem of checking if a property (here a μ-calculus formula) holds in our framework, to a problem of validity in the μ-calculus. We exploit partial model checking in this reduction step, and we show how the validity problem, generally EXPTIME complete, can be solved efficiently in the universal conjunctive subclass of the μ-calculus. Through examples, we show that this subclass is sufficiently expressive to model many important fault-tolerance properties. The second strategy consists in characterizing the fault-tolerance properties (here "fault tolerance", "fail stop", "fail safe", and "fail silent") in the Generalized Non Deducibility on Compositions, a scheme that has been profitably applied in the definition and in the analysis of many security protocol properties. Thus, we can reuse in fault-tolerance the techniques for validating non-interference from which the Generalized Non Deducibility on Compositions originates. We also argue about the availability of effective methodologies of analysis, and about the possibility of applying compositional techniques.

3.1 Related Work

Some preliminary ideas about a relationship between security and fault tolerance analysis can be found in [204, 187, 185, 153, 151, 180, 122, 89].

An informal and introductory comparison between properties in dependability and security is presented in [121]. In a seminal paper [204], Weber shows that the concept of non-interference [101] used in security, captures the intuitive notions of "fault-tolerance" and of "graceful degradation"; informally, they can be read as "the occurrence of faults does not interfere (or weakly interferes) with the visible behavior of the system" [204]. Weber suggests to validate a system under different sets of fault scenarios, and he supposes that the likelihood of these scenarios is determined by an environment interacting with the system.

We anticipate that in this chapter we develop these ideas further. First, we model a fault-tolerant system and its environment in the formal framework of the Calculus of Communicating Systems [158]; then we characterize fault-tolerance in terms of logical and non-interference problems. In our framework, we are able to propose different strategies of analysis, which were missing in [204].

In [153] Meadows proposes a classification of security properties inspired by the taxonomy used in fault-tolerance. She also argues that security analysis can be improved by incorporating techniques typical of dependability. Four years later, following a complementary trend, Meadows and McLean claim that the use of emerging results in security analysis can enrich the fault prevention and fault removal strategies [151].

Rushby [180], observes analogies between non-interferences approaches in security and in safety analysis mainly regarding the technique in system design called "partitioning". Foley [88, 89] uses CSP [115] to define the "integrity" property as a predicate over traces. Integrity is a common property of dependability and computer security; Foley shows how his characterization classifies integrity as a non-interference property.

A formal characterization of safety properties such as non-interference, non deducibility, and casuality and their role in fault intrusion tolerance is discussed by Stavridou and Dutertre in [187]. They affirm that, even though the pessimistic worst-case assumptions used in security are too strong when applied to fault-tolerance, non-interference provides a useful framework for specifying and verifying safety, reliability and availability [187]. They also point out the need for verification techniques of non-interference, especially those addressing compositionality.

In [185], Simpson, Woodcock and Davies, uses CSP to formalize "fail safe", "fail soft", and "fault-tolerance" as properties of non-interference. These properties are expressed by a weak version of a relation, called protection, defined for classes of events: in a process P, E is protected from F if availability of E actions in any trace of P is unaffected by the occurrence of events from F. A particular process, $Run(F)$ makes events from F always available, and properties over a system are defined by assuming the system to run concurrently with $Run(F)$. They also use the CSP model checker [176] as a verification tool.

3.2 Introduction

In this chapter, we apply to fault-tolerance analysis two strategies used to define and to analyze computer and protocol security properties. The first strategy, studied Section 3.6, requires a fault-tolerance property to be expressed by a μ-calculus formula. It consists in reducing the problem of checking if a property holds in our framework, to a problem of validity in the μ-calculus. We exploit partial model checking in this reduction step, and we show how the validity problem, generally EXPTIME complete, can be solved efficiently in the universal conjunctive subclass of the μ-calculus. The second strategy, studied in Section 3.8, consists in characterizing the fault-tolerance properties "fault-tolerance", "fail stop", "fail safe", and "fail silent" in the Generalized

Non Deducibility on Compositions (in short, GNDC) [86]. GNDC is a scheme that has been profitably applied in the definition and in the analysis of many security protocol properties. The analysis of fault-tolerance properties within the GNDC can benefit from techniques and tools for the verification of information flow and non-interference properties [82], from which GNDC originate. Moreover, the uniform framework of GNDC helps in proving similarities between security properties and fault-tolerance properties; we show, for example, that fault-tolerance is exactly the BNDC [81] security property. Potentially this is also a first step towards a formal and uniform taxonomy of fault-tolerance properties.

As a common modeling framework, our approach requires that a system, its failing behavior, and its fault-recovering procedures, are formally specified as finite state terms in a process algebra. Here, we use the Calculus of Communicating System (CCS) [158], but our framework is completely general and it can be easily rephrased in other process algebras, for instance the CSP [115] or the π-calculus [161]. The validation framework we propose falls into the *open system* paradigm: a system acts within an unspecified environment which is able to trigger actions in the system. We will call such an environment *faulty environment*. Usually, the presence of an environment causing any action of the system's interface has two unpleasant effects: the first is the well-known state space explosion [201], the second is that unrealistic situations may arise during the analysis [93]. As a solution we verify a system P in a well-characterized class $\mathcal{E}_{\mathcal{F}}$ of (faulty) environments. Each faulty environment $F \in \mathcal{E}_{\mathcal{F}}$ acts as a fault-injector that interacts with the system only through a specified finite set $\mathcal{F}$ of fault actions. Differently from [185, 200], we treat F as an unspecified component of the system. In this way, we check the reliability of a system model with respect to any potential occurrences of faults. In CCS, our framework can be summarized as $(P \parallel F) \setminus \mathcal{F}$, where P is the model of our fault-tolerant candidate system, F is an unspecified term in $\mathcal{E}_{\mathcal{F}}$, and $\mathcal{F}$ is the finite set of fault actions.

In the first part of this chapter we formalize fault-tolerance in a logical formalism, here a variant of the μ-calculus [35]. By partial model checking [16], the fault tolerance analysis problem is reduced to a validity problem in the μ-calculus. Intuitively, the idea is as follows: proving that $\forall F \in \mathcal{E}_{\mathcal{F}}, (P \parallel F)\setminus\mathcal{F}$ satisfies a fault-tolerance property ϕ, is equivalent to prove that the modified formula $\phi /\!\!/_{\mathcal{F}} P$ is valid in $\mathcal{E}_{\mathcal{F}}$, where $/\!\!/_{\mathcal{F}}$ is the partial evaluation for the parallel composition and restriction operators. The modified formula $\phi /\!\!/_{\mathcal{F}} P$ characterizes exactly the scenarios of faults the system is resilient to. Moreover, by considering the characteristic formulas ϕ of a set of possible fault scenarios, checking if P is fault-tolerant with respect to those scenarios is equivalent to check the validity of $\phi \Rightarrow \varphi /\!\!/_P$. logical characterization of fault-tolerance is given, several analysis techniques may be adopted. Some of them lead to efficient analysis of certain properties: we identify a class of μ-calculus formulas whose validity checking can be performed in linear-time in the dimension of $\phi /\!\!/_{\mathcal{F}} P$.

In the second part we study the application of Generalized Non Deducibility on Compositions (GNDC) in fault-tolerance analysis. GNDC, first presented in [86], is a framework where a family of security properties has been uniformly expressed and verified [86, 85]. GNDC has roots in non-interference analysis, and it has not been applied to fault tolerance so far. In our framework a GNDC property has the form:

$$P \text{ satisfies } GNDC^{\alpha}_{\lhd} \qquad \text{iff} \qquad \forall F \in \mathcal{E}_{\mathcal{F}} : (P \parallel F) \setminus \mathcal{F} \lhd \alpha(P)$$

Generally speaking this means that a system P enjoys $GNDC^{\alpha}_{\lhd}$ if and only if P shows (with respect to a process relation $\lhd$) the same behavior as $\alpha(P)$. This must be true even if P is composed, by the parallel operator $\parallel$, with any environment F chosen from $\mathcal{E}_{\mathcal{F}}$. Here, $\mathcal{E}_{\mathcal{F}}$ represents

the set of all environments which interact with P through actions $\mathcal{F}$. GNDC is parametric in $\triangleleft$, a relation among processes representing the notion of "observation", and in α a function between CCS terms. Given P, $\alpha(P)$ describes the expected (correct) behavior of P.

In the uniform scheme of the GNDC we express and compare the fault-tolerance properties "fault-tolerance", "fail stop", "fail safe", and "fail silent". This comparison describes a preliminary step towards a formal classification of dependable properties, on the basis of the work by Focardi *et al* [85] who have compiled a classification of security properties. Finally, we show how some of the theoretical results of GNDC originally stated for security analysis (*e.g.*, compositionality in proving a GNDC property) can be reformulated and reused in the analysis of fault-tolerance.

The chapter is organized as follows: Section 3.3 summarises the basic theory of CCS. Section 3.4 explains the uniform scheme that we use to model a fault-tolerant (candidate) system and its environment. Section 3.5 recalls the (equational) μ-calculus modal logic for process analysis. Section 3.6 describes our characterization of fault-tolerance in the μ-calculus logic framework. Moreover, it explains our solution methods based on partial model checking and on an efficient methodology to check the validity in a subclass of the μ-calculus. Section 3.7 summarises the definitions of process behavioral equivalences we use in the rest of the chapter. Section 3.8 describes our characterization of fault-tolerance in the GNDC scheme, and underlines our solution methods in this framework. Section 3.9 concludes the chapter.

3.3 CCS Background

This section summarises the basic notions and definitions of the Calculus of Communicating System (CCS) [158], the calculus used through the chapter.

CCS assumes a set $Act = \mathcal{L} \cup \overline{\mathcal{L}}$ of (observable) *communication actions*. Names from $\mathcal{L}$ model the emission of a signal; overlined names from $\overline{\mathcal{L}}$ (called co-names) represent the reception of a signal. The purpose of putting a line, called complementation, over a names is to show that the corresponding action can synchronize with its complemented partner. Complementation follows the rule that $\overline{\overline{a}} = a$, for any communication action $a \in Act$. A special symbol, τ, is used to model any (unobservable) *internal action*; hence the full set of possible actions is $Act_\tau = Act \cup \{\tau\}$. We let a, b range over Act_τ. The following grammar specifies the syntax of the language defining all the CCS processes:

$$P, Q \quad ::= \quad \mathbf{0} \;\mid\; a.P \;\mid\; P + Q \;\mid\; P \parallel Q \;\mid\; P \setminus \mathcal{A} \;\mid\; P[f] \;\mid\; A$$

Informally, $\mathbf{0}$ is the process that does not perform any action. $a.P$ is the process ready to perform action a, then it behaves as P. Process $P + Q$ can choose non-deterministically to behave either as P or as Q. $\parallel$ is the operator of parallel composition: in $P \parallel Q$, P and Q may evolve concurrently or communicate via complementary communication actions. In $P \setminus \mathcal{A}$, where $\mathcal{A} \subseteq \mathcal{L}$, actions $a \in \mathcal{A} \cup \overline{\mathcal{A}}$ are prevented from happening; they are possible only in a communication internal to P. $P[f]$ is the process obtained from P by changing each $a \in Act_\tau$ into $f(a)$; the relabeling function f must be such that $f(\tau) = \tau$. A is a process identifier. We assume that every process identifier A has a defining equation $A \stackrel{\text{def}}{=} P$.

The operational semantics of CCS is given in the form of labelled transition systems $(\mathcal{E}, Act_\tau, \xrightarrow{a})$, where states $\mathcal{E}$ are CCS terms, actions Act_τ are CCS actions, and the transition relation $\xrightarrow{a} \subseteq \mathcal{E} \times Act_\tau \times \mathcal{E}$ is defined by structural induction as the least relation generated by the following set of inference rules:

$$\frac{}{a.P \xrightarrow{a} P} \qquad \frac{P \xrightarrow{a} P'}{P + Q \xrightarrow{a} P'} \qquad \frac{Q \xrightarrow{a} Q'}{P + Q \xrightarrow{a} Q'}$$

$$\frac{P \xrightarrow{a} P'}{P \parallel Q \xrightarrow{a} P' \parallel Q} \qquad \frac{Q \xrightarrow{a} Q'}{P \parallel Q \xrightarrow{a} P \parallel Q'} \qquad \frac{P \xrightarrow{\overline{a}} P', Q \xrightarrow{a} Q',\ a \neq \tau}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}$$

$$\frac{P \xrightarrow{a} P'}{P[f] \xrightarrow{f(a)} P'[f]} \qquad \frac{P \xrightarrow{a} P',\ a \notin \mathcal{A} \cup \overline{\mathcal{A}}}{P \setminus \mathcal{A} \xrightarrow{a} P' \setminus \mathcal{A}} \qquad \frac{P \xrightarrow{a} P', A \stackrel{\text{def}}{=} P}{A \xrightarrow{a} P'}$$

The transition relation $\xrightarrow{a}$ defines the usual concept of derivation in one step: $P \xrightarrow{a} P'$ means that process P evolves in one step into process P' by executing action $a \in Act_\tau$. We write $P \xrightarrow{a}$ to underline that P can perform an action a and evolve in some process. The transitive and reflexive closure of $\bigcup_{a \in Act_\tau} \xrightarrow{a}$ is written $\longrightarrow^*$.

Definition 3.3.1 *Given a CCS process P, the set $Der(P) = \{P' | P \longrightarrow^* P'\}$, is the set of its* derivatives. *A CCS process P is* finite state *if $Der(P)$ is finite.*

Definition 3.3.2 *Let $Sort(P)$ (called the* sort *of P) be the set of names of actions that syntactically appear in the process P, and let $\mathcal{F}$ be a finite set of actions. The set, $\mathcal{E}_\mathcal{F}$,* of processes whose sort is in $\mathcal{F} \cup \{\tau\}$, *is so defined:*

$$\mathcal{E}_\mathcal{F} \stackrel{def}{=} \{F : Sort(F) \subseteq \mathcal{F} \cup \{\tau\}\}$$

3.4 Modeling Fault-Tolerant Systems

Using process algebras it is possible to provide a uniform framework for specifying fault-tolerant systems. In [19, 21] CCS/Meije is used to specify a fault-tolerant system, its failing behavior, its recovery strategies, and the fault assumptions. Fault assumptions define if a fault is, for instance, temporary, permanent, or Byzantine.

We follow a similar modeling approach, but differently from [19, 21] we do not include any specific fault assumption in the system specification. Instead, we develop a neat separation between the system and its environment that acts as a fault-injector. We call such a fault-injector environment *faulty environment*. This choice has important conceptual implications:

- the specification of the system must describe the behaviors of the system in reaction to faults, but not the fault assumptions;
- all the fault assumptions are part of the faulty environment.

These are our technical ideas to encode fault-tolerance analysis as the analysis of an open system. We are interested in evaluating the system behavior in a general and unspecified faulty environment; we describe our strategies of analysis in Section 3.6 and Section 3.8. In the following, when talking about formal specifications of fault-tolerant systems, we understand the following definitions:

A system is a finite state CCS process, P, describing the behavior of the system through the execution of actions. Generally, P is a parallel composition of sub-processes, each modeling sub-components of the system communicating with each other.

A failing system is a finite state CCS process, $P_{\mathcal{F}}$, obtained by extending the process P with the possibility of executing fault actions from a set $\mathcal{F}$. In $P_{\mathcal{F}}$ we specify also the failure modes *i.e.*, the behavior of the system induced by the occurrence of the faults.

A fault-tolerant (candidate) system is a finite state CCS process, $P^{\#}_{\mathcal{F}}$, obtained by adding to $P_{\mathcal{F}}$ those processes modeling some error-recovery mechanism in accordance with some fault-tolerant design strategy (*e.g.*, modular redundancy, voting). In CCS, $P^{\#}_{\mathcal{F}}$ has the form $(P^{(1)}_{\mathcal{F}} \parallel \cdots \parallel P^{(n)}_{\mathcal{F}} \parallel Q) \setminus \mathcal{A}$ where:

- $P^{(1)}_{\mathcal{F}} \ldots P^{(n)}_{\mathcal{F}}$ are n copies of $P_{\mathcal{F}}$ in a parallel composition.
- Q is a process that represents an additional error detection module, for instance, a voter. The detail description of this process usually depends on the particular fault-tolerance strategy we are describing in the system specification.
- $\mathcal{A} = \{a_1, \cdots, a_n\}$, $\mathcal{A} \cap \mathcal{F} = \emptyset$ is the set of names over which $P^{(1)}_{\mathcal{F}} \ldots P^{(n)}_{\mathcal{F}}$ and Q communicate.

Occurrences of faults are induced by a faulty environment F, that causes faults to happen. It interacts with $P^{\#}_{\mathcal{F}}$ only through actions in $\mathcal{F}$.

The previous definitions suggest a uniform characterization of $P^{\#}_{\mathcal{F}}$: a fault-tolerant (candidate) system P can be obtained by applying, to P, a function, β, from processes to processes. In the following, we abstract from any particular β in our modifier $(\_)^{\#}_{\mathcal{F}}$. So $P^{\#}_{\mathcal{F}}$ is the CCS specification of the fault-tolerant version of P obtained by applying some fault-tolerance technique. As a unique constraint, at model level, the set $\mathcal{F}$ must remain disjoint from any other set of actions and must be accessible to the environment.

Example 3.4.1 We show the CCS specification of a simple fault-tolerant battery, Bat. The battery returns one unit of energy when it receives a request message. Actions $\mathtt{get}$ and $\overline{\mathtt{ret}}$ model the request signal and the unit of energy, respectively. The CCS process describing Bat is as follows:

$$Bat \stackrel{\text{def}}{=} \mathtt{get}.\overline{\mathtt{ret}}.Bat$$

In its failing version, $Bat_{\{\mathtt{f}_0,\mathtt{f}_1\}}$, the battery may crash after it receives a request. As an effect, it may produce either a valid energy unit (action $\mathtt{ret}_1$) or an invalid burst of energy (action $\mathtt{ret}_0$). We assume two different possible faults: the former (action $\mathtt{f}_0$) certainly causes the battery to fail; the latter (action $\mathtt{f}_1$) causes the battery to switch in a failing state where either a valid or an invalid energy unit may be produced non-deterministically. A silent action, τ, models some internal behavior that appears before the module switches into its failing state as an effect of a $\mathtt{f}_1$ fault. The CCS specification of $Bat_{\{\mathtt{f}_0,\mathtt{f}_1\}}$ is as follows:

$$\begin{aligned} Bat_{\{\mathtt{f}_0,\mathtt{f}_1\}} \stackrel{\text{def}}{=} \; & \mathtt{get}.(\overline{\mathtt{ret}}_1.Bat_{\{\mathtt{f}_0,\mathtt{f}_1\}} \\ & \quad + \mathtt{f}_0.\overline{\mathtt{ret}}_0.Bat_{\{\mathtt{f}_0,\mathtt{f}_1\}} + \mathtt{f}_1.\tau.Bat') \\ Bat' \stackrel{\text{def}}{=} \; & \overline{\mathtt{ret}}_1.Bat_{\{\mathtt{f}_0,\mathtt{f}_1\}} + \overline{\mathtt{ret}}_0.Bat_{\{\mathtt{f}_0,\mathtt{f}_1\}} \end{aligned}$$

Starting from $Bat_{\{\mathtt{f}_0,\mathtt{f}_1\}}$ we design the fault-tolerant version of the battery. It is composed of two redundant instances of the battery and of the two additional modules: Spl, a splitter, and Con, a voter. We now give the CCS processes describing all of these components.

$$Bat^{(i)} \stackrel{\text{def}}{=} Bat[\mathtt{get}_i/\mathtt{get}, \mathtt{ret}_{i,1}/\mathtt{ret}]$$

$$Bat^{(i)}_{\{\mathtt{f}_0,\mathtt{f}_1\}} \stackrel{\text{def}}{=} Bat_{\{\mathtt{f}_0,\mathtt{f}_1\}}[\mathtt{get}_i/\mathtt{get}, \mathtt{ret}_{i,0}/\mathtt{ret}_0, \mathtt{ret}_{i,1}/\mathtt{ret}_1]$$

The two indexed instances of the battery Bat and $Bat_{\{\mathtt{f}_0,\mathtt{f}_1\}}$, are specified by processes $Bat^{(i)}$ and $Bat^{(i)}_{\{\mathtt{f}_0,\mathtt{f}_1\}}$ respectively, for $i = 1, 2$. Action $\mathtt{get}_1$ (respectively, $\mathtt{get}_2$) represents the request that the splitter directs to the first (respectively, the second) instance of the battery. Actions $\mathtt{ret}_{1,1}$ and $\mathtt{ret}_{1,0}$ (respectively, $\mathtt{ret}_{2,1}$ and $\mathtt{ret}_{2,0}$) represent the outputs of the first (respectively, the second) battery in case of a valid or an invalid production of energy.

$$Spl \stackrel{\text{def}}{=} \mathtt{get}.\overline{\mathtt{get}}_1.\overline{\mathtt{get}}_2.\mathtt{ack}.Spl$$

The CCS process specifying the splitter, Spl, delivers the energy request to each of the two redundant modules. For sake of simplicity, our splitter forwards a request of energy in a precise order. Moreover, Spl cannot accept a new energy request until it receives a synchronization signal from the controller (action $\mathtt{ack}$).

$$Con \stackrel{\text{def}}{=} \mathtt{ret}_{1,0}.Con' + \mathtt{ret}_{1,1}.\overline{\mathtt{ret}}.Con''$$

$$Con' \stackrel{\text{def}}{=} \mathtt{ret}_{2,0}.Con''' + \mathtt{ret}_{2,1}.\overline{\mathtt{ret}}.Con'''$$

$$Con'' \stackrel{\text{def}}{=} \mathtt{ret}_{2,0}.Con''' + \mathtt{ret}_{2,1}.Con'''$$

$$Con''' \stackrel{\text{def}}{=} \overline{\mathtt{ack}}.Con$$

The controller can collect the energy units from the two batteries. If a valid unit is returned, the controller shows it to the environment (action $\overline{\mathtt{ret}}$). It also absorbs an eventual over production of energy. After the controller has received a signal from both the batteries, it sends to the splitter the synchronization message $\overline{\mathtt{ack}}$; if both batteries fail in producing their unit of energy the controller only sends the message $\overline{\mathtt{ack}}$, and the splitter is ready to receive a new energy request. We can now build two different fault-tolerant (candidate) specifications:

$$Bat^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}} \stackrel{\text{def}}{=} (Spl \parallel Bat^{(1)} \parallel Bat^{(2)}_{\{\mathtt{f}_0,\mathtt{f}_1\}} \parallel Con) \setminus \mathcal{A} \tag{3.4.1}$$

$$Battery^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}} \stackrel{\text{def}}{=} (Spl \parallel Bat^{(1)}_{\{\mathtt{f}_0,\mathtt{f}_1\}} \parallel Bat^{(2)}_{\{\mathtt{f}_0,\mathtt{f}_1\}} \parallel Con) \setminus \mathcal{A}$$

where $\mathcal{A} = \{\mathtt{get}_1, \mathtt{get}_2, \mathtt{ret}_{1,0}, \mathtt{ret}_{1,1}, \mathtt{ret}_{2,0}, \mathtt{ret}_{2,1}, \mathtt{ack}\}$. ■

$Bat^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}}$, contains one potentially failing battery (see also Figure 3.1); $Battery^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}}$, uses two failing batteries. We will use $Bat^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}}$ and $Battery^{\#}\{\mathtt{f}_0, \mathtt{f}_1\}$ as test cases throughout the chapter.

3.4.1 Our Scenario for Fault-Tolerance Analysis

In this section, we introduce in our general scenario for the analysis of fault-tolerance, and we give a formalization of it in CCS. As introduced in Section 3.2, we propose to study a fault-tolerant (candidate) system in a generic and unspecified faulty environment acting as a fault-injector. The

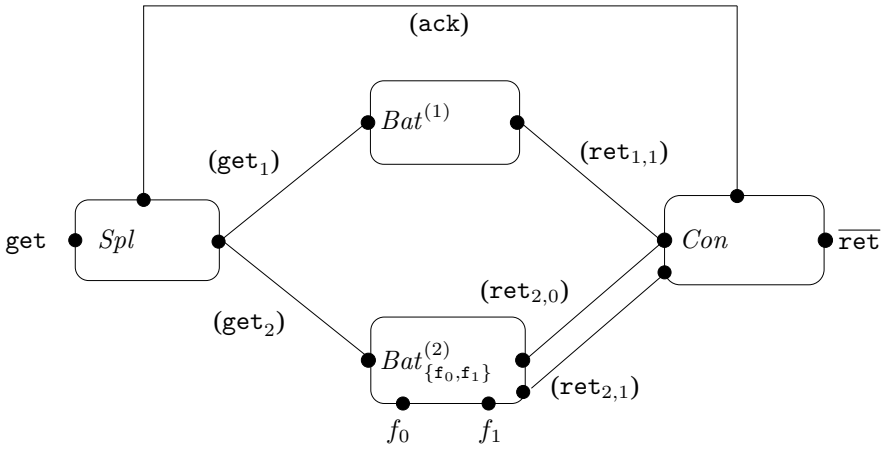


Figure 3.1: The flow diagram of the (candidate) fault-tolerant version of the battery, $Bat^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}}$. Restricted actions are within brackets.

faulty environment is able to interact with the systems through a finite and defined set of fault actions, in fact triggering the occurrence of faults in the systems. In CCS this scenario is so defined:

$$\forall F \in \mathcal{E}_{\mathcal{F}}, \qquad (P^{\#}_{\mathcal{F}} \parallel F) \setminus \mathcal{F} \tag{3.4.2}$$

In scenario (3.4.2), process F is the faulty environment, that interacts with $P^{\#}_{\mathcal{F}}$ through the finite set of actions $\mathcal{F}$. Moreover, F is an unspecified component ranging over $\mathcal{E}_{\mathcal{F}}$, the set of possible CCS processes whose sort is in $\mathcal{F} \cup \{\tau\}$. Set $\mathcal{E}_{\mathcal{F}}$ is the class of all possible faulty environment and it represents our unique fault assumption model.

Remark 3.4.2 $P^{\#}_{\mathcal{F}}$ we do not include other fault actions than those triggered by F. Therefore, $\mathcal{F}$ is *exactly* the set of names over which $P^{\#}_{\mathcal{F}}$ and F interact. ■

Remark 3.4.3 Fault actions are restricted. This implies that $P_{\mathcal{F}}$ and F have to synchronize on $\mathcal{F}$. At the abstraction level of our scenario of analysis, faults are then considered internal (*i.e.*, not observable) actions of the failing systems: only the (probably faulty) behavior of a system is really observable. ■

In practice, a system is either resilient to faults or the presence of faults is highlighted by its subsequent behavior. Roughly speaking, in our framework "fault-tolerance" means that faults cannot interfere with the normal observable behavior of the system.

Example 3.4.4 The scenario for analyzing the fault-tolerant (candidate) battery $Bat^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}}$ is as follows:

$$\forall F \in \mathcal{E}_{\{\mathtt{f}_0,\mathtt{f}_1\}}, \ (Bat^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}} \parallel F) \setminus \{\mathtt{f}_0, \mathtt{f}_1\}$$

■

3.5 Background on Logic and Properties of Processes

In this section we summarize the technical background required to understand the logic characterization of fault-tolerance we will develop in Section 3.6. We summarise basic notions of the modal μ-calculus and of the equational μ-calculus in Section 3.5.1 and Section 3.5.2 respectively, we digest the use of the μ-calculus for observational properties over processes in Section 3.5.3, and we synthetize the basic of partial model checking in Section 3.5.4.

3.5.1 Modal μ-calculus

The modal μ-calculus [35] is a modal logic with fix-point operators. It is used in computer science to express temporal properties of distributed systems, such as non-terminating behaviors, safety and liveness properties [127]. Formulas of the μ-calculus are generated by the following grammar:

$$\phi := \mathtt{tt} \mid \mathtt{ff} \mid X \mid \phi \wedge \phi' \mid \phi \vee \phi' \mid \langle a \rangle \phi \mid [a]\phi \mid \mu X.\phi \mid \nu X.\phi$$

Here a ranges over the action set Act_τ and X ranges over a set of variables $\mathcal{V}$. The fix-point operators are ν (greatest fix-point) and μ (least fix-point). The semantics, $\|\phi\|_\rho$, of a μ-calculus formula ϕ is defined over labelled transition systems. Let $\mathcal{M} = (\mathcal{Q}, Q_0, Act_\tau, \xrightarrow{a})$ be a labelled transition system, and ρ an environment function that associates a subset of $\mathcal{Q}$ to the free variables in ϕ. As a notation $\rho[x/X]$ is the environment ρ where x is associated with X. If we let σ range over $\{\mu, \nu\}$ then $\|\phi\|_\rho$ is the set of states of $\mathcal{M}$ defined by the following equations:

$$\|X\|_\rho = \rho(X), \quad \|\mathtt{tt}\|_\rho = \mathcal{Q}, \quad \|\mathtt{ff}\|_\rho = \emptyset$$
$$\|\phi_1 \wedge \phi_2\|_\rho = \|\phi_1\|_\rho \cap \|\phi_2\|_\rho, \quad \|\phi_1 \vee \phi_2\|_\rho = \|\phi_1\|_\rho \cup \|\phi_2\|_\rho$$
$$\|[a]\phi\|_\rho = \{Q | \forall Q' : Q \xrightarrow{a} Q' \text{ implies } Q' \in \|\phi\|_\rho\}$$
$$\|\langle a \rangle \phi\|_\rho = \{Q | \exists Q' : Q \xrightarrow{a} Q' \text{ and } Q' \in \|\phi\|_\rho\}$$
$$\|\sigma X.\phi\|_\rho = \sigma f \text{ where } f(x) \stackrel{\text{def}}{=} \|\phi\|_{\rho[x/X]}$$

A labelled transition system $\mathcal{M}$ satisfies a μ-calculus formula ϕ, written $\mathcal{M} \models_\rho \phi$, if $Q_0 \in \|\phi\|_\rho$. We remove the subscript ρ when it is clear from the context or when ϕ does not contain free variables, *i.e.,* when ϕ is a closed formula.

Remark 3.5.1 The modal μ-calculus here presented does not contain a negation operator $\neg$. However for any formula ϕ there is a formula ϕ^c, called the complement of ϕ, which expresses the negation of ϕ. A formula ϕ^c is obtained by substituting for every operator in ϕ its dual according to the following inductive rules:

$$\begin{array}{lcl} \mathtt{tt}^c & = & \mathtt{ff} \\ (\langle K \rangle \phi)^c & = & [K]\phi^c \\ (\phi \wedge \phi')^c & = & \phi^c \vee \phi'^c \\ (\mu Z.\phi)^c & = & \nu Z.\phi^c \\ Z^c & = & Z \end{array} \qquad \begin{array}{lcl} \mathtt{ff}^c & = & \mathtt{tt} \\ ([K]\phi)^c & = & \langle K \rangle \phi^c \\ (\phi \vee \phi')^c & = & \phi^c \wedge \phi'^c \\ (\nu Z.\phi)^c & = & \mu Z.\phi^c \end{array}$$

■

The modal μ-calculus subsumes [63, 23] several other state-based logics such as PDL, CTL, and CTL*, and action-based logics such as ECTL*, ACTL and ACTL*. Moreover, the μ-calculus

enjoys the finite model property, *i.e.*, if a closed formula ϕ is satisfiable then there exists a finite model for ϕ [192]. A finitary axiomatization has been proposed by Walukievicz [202].

Many properties can be express in the μ-calculus (see [190, 37]). For example, $\nu X.\langle - \rangle \mathtt{tt} \wedge [-]X$ expresses "deadlock freedom", and $\mu X.\langle - \rangle \mathtt{tt} \wedge [-a]X$ expresses "action a must eventually occur". In writing properties, here and in the rest of the chapter, we use the shortcut notations $[K]\phi$ and $\langle K \rangle \phi$ where K is a set of actions in Act_τ: $[K]\phi$ is a macro for $\bigwedge_{a \in K}[a]\phi$ and $\langle K \rangle \phi$ for $\bigvee_{a \in K}\langle a \rangle \phi$. Moreover $-K$ is an abbreviation for $Act_\tau - K$. The abbreviation $-a$ stands for $-\{a\}$; $\langle Act_\tau \rangle \phi$ and $[Act_\tau]\phi$ are synthetically written as $\langle - \rangle \phi$ and $[-]\phi$, respectively.

3.5.2 Equational μ-calculus

The equational μ-calculus [125, 16, 23, 17] is an equivalent variant of the μ-calculus. By the use of standard techniques [15, 145, 146], a μ-calculus formula ϕ can be transformed, in linear-time in ϕ, into an equivalent equational μ-calculus formula and vice-versa. For this reason the μ-calculus and the equational μ-calculus can be use interchangeably in all the results we show in this chapter. Bhat and Cleaveland proposed translations from CTL, CTL* and ECTL* into equational μ-calculus [23].

The equational μ-calculus is based on fix-point equations that substitute the recursion operators. Let X be a variable ranging over a set $\mathcal{V}$ of variables, then a least (greatest) fix-point equation is $X =_\mu \phi$ ($X =_\nu \phi$), where ϕ is an *assertion*, that is a modal formula without recursion operators. The syntax of assertions (ϕ) and of lists of equations (φ) is defined by the following grammar:

$$\begin{array}{lll} \textit{assertion} & \phi := & \mathtt{tt} \mid \mathtt{ff} \mid X \mid \phi \wedge \phi' \mid \phi \vee \phi' \mid \langle a \rangle \phi \mid [a]\phi \\ \textit{equations list} & \varphi ::= & (X =_\nu \phi)\ \varphi \mid (X =_\mu \phi)\ \varphi \mid \epsilon \end{array}$$

It is assumed that variables appear only once on the left-hand sides of an equations list φ : the set of these variables is denoted as $Defs(\varphi)$. An equations list φ is closed if every variable that appears in the assertions of the list is in $Defs(\varphi)$. Figure 3.2 gives example of properties in the equational μ-calculus.

The semantics of the equational μ-calculus is defined over labelled transition systems. As a notation, $\sqcup$ represents the union of disjoint environments, and [] denotes the empty environment. Letting σ be in $\{\mu, \nu\}$, $\sigma U.f(U)$ represents the σ fix-point of the function f in one variable U. Let be $\mathcal{M} = (\mathcal{Q}, Q_0, Act_\tau, \xrightarrow{a})$ a labelled transition system and ρ an environment function that assigns a subset of $\mathcal{Q}$ to the free variables in ϕ. The semantics, $\|\varphi\|'_\rho$, of an equation list φ is an environment which assigns subsets of states of $\mathcal{Q}$ systems to variables in $Defs(\varphi)$. Formally, $\|\varphi\|'_\rho$ is defined by the following equations:

$$\begin{array}{rcl} \|\epsilon\|'_\rho & = & [] \\ \|(X =_\sigma \phi)\varphi\|'_\rho & = & \|\varphi\|'_{(\rho \sqcup [U' \setminus X])} \sqcup [U'/X] \end{array}$$

where $U' = \sigma U.\|\phi\|_{(\rho \sqcup [U/X] \sqcup \rho'(U))}$, and $\rho'(U) = \|\varphi\|'_{(\rho \sqcup [U/X])}$. The interpretation, $\|\phi\|_\rho$, of an assertion ϕ is defined as for the μ-calculus.

Informally $\|(X =_\sigma \phi)\varphi\|'_\rho$ says that the solution to $(X =_\sigma \phi)\varphi$ is the σ fixed point solution U' of $\|\phi\|_\rho$ where the solution to the rest of the list of equations φ is used as environment. A labelled transition system $\mathcal{M}$ satisfies an equation list φ, written $\mathcal{M} \models_\rho \varphi \downarrow X$, if $Q_0 \in \|\varphi\|'_\rho(X)$, where X is the first variable in the list φ. We omit ρ out when it is evident from the context or when φ is closed.

$X =_\nu [-]X \wedge \langle - \rangle \mathtt{tt}$	Absence of deadlock
$X =_\nu [-]X \wedge \langle \mathtt{a} \rangle \mathtt{tt}$	On all-paths an action a occurs
$Y =_\mu [-]Y \vee \langle \mathtt{a} \rangle \mathtt{ff}$	There is path on which eventually no a occurs
$\begin{cases} X =_\nu Y \\ Y =_\mu [-]Y \vee \langle \mathtt{a} \rangle \mathtt{X} \end{cases}$	Along all paths a occurs infinitely often

Figure 3.2: Examples of properties in the equational μ-calculus [17], and their informal meaning.

3.5.3 Observable μ-calculus Properties over Processes

In case an external observer cannot see τ actions, a natural way of analyzing a process is abstracting these actions from the behavior of the processes, while preserving the branching structure. This is the viewpoint that we consider in our framework. Let us consider the following labelled transition relation $\stackrel{a}{\Longrightarrow}$, between CCS terms

$$\begin{array}{lll} P \stackrel{\tau}{\Longrightarrow} P' & \text{if} & P \stackrel{\tau}{\longrightarrow}^* P' \\ P \stackrel{a}{\Longrightarrow} P' & \text{if} & P \stackrel{\tau}{\longrightarrow}^* \stackrel{a}{\longrightarrow} \stackrel{\tau}{\longrightarrow}^* P', \quad a \in Act \end{array}$$

Transitions $\stackrel{a}{\Longrightarrow}$ are called *observable* transitions, or *weak* transitions in contrast to transitions $\stackrel{a}{\longrightarrow}$ which are qualified as *strong*.

Modal μ-calculus formulas are usually too strong with respect to what an external observer can see. In other words they can distinguish processes that are indistinguishable if we would consider only observable transitions. In order to make properties compatible with the notion of external observation, modalities of the μ-calculus must be interpreted in terms of observable transitions, that is by using the transition $\stackrel{a}{\Longrightarrow}$. In that case, $\langle\langle \_ \rangle\rangle$ and $[[\_]]$ are used instead of $\langle \_ \rangle$ and $[\_]$, respectively. The interpretation of formula $\langle\langle a \rangle\rangle \phi$, for example, is like $\langle a \rangle \phi$ where the weak transition relation is used instead of the strong one:

$$\|\langle\langle a \rangle\rangle \phi\|'_\rho = \{Q | \; \exists \, Q' : Q \stackrel{a}{\Longrightarrow} Q' \text{ and } Q' \in \|\phi\|'_\rho\}$$

Weak modalities can be also defined in terms of the corresponding strong modalities [189]; for example $\langle\langle a \rangle\rangle \stackrel{\text{def}}{=} \mu Z.\langle \tau \rangle Z \wedge \langle a \rangle \mu Z.\phi \wedge \langle \tau \rangle Z$. The sub-logic of the μ-calculus obtained restricting the modalities to the subset $\{\langle\langle \; \rangle\rangle, [[\;]], \langle\langle K \rangle\rangle, [[K]]\}$ (with $\tau \notin K$) is called the *observational μ-calculus* [189].

3.5.4 Partial Model Checking

Partial model checking [16, 17] is a technique that relies upon compositional methods for proving properties of concurrent system. It has been introduced first by Andersen, who used the equational μ-calculus for technical convenience [16]. Indeed, the μ-calculus and the equational μ-calculus can be used interchangeably in this context, as noticed in Section 3.5.2.

Reformulated in the CCS, the intuitive idea underlying partial evaluation is the following: proving that $(P \parallel Q) \setminus \mathcal{A}$ satisfies an equational μ-calculus formula ϕ is equivalent to proving that Q satisfies a modified formula $\phi /\!\!/_{\mathcal{A}} P$, where $/\!\!/_{\mathcal{A}} P$ is the partial evaluation function for the operators of parallel composition and restriction.

In the following we use $P \parallel_{\mathcal{A}} Q$ as an abbreviation for $(P \parallel Q) \setminus \mathcal{A}$. In Figure 3.3 we give the definition of $/\!\!/_{\mathcal{A}}$, the partial evaluation function for the CCS operator $\parallel_{\mathcal{A}}$ where $\mathcal{A} \subseteq Act$. Andersen proves the following lemma [16]:

Lemma 3.5.2 *Given a process $P \parallel_{\mathcal{A}} Q$ (where P is finite-state) and an equational specification $\varphi \downarrow X$ we have:*

$$P \parallel_{\mathcal{A}} Q \models (\varphi \downarrow X) \qquad \textit{iff} \qquad Q \models (\varphi \downarrow X) /\!\!/_{\mathcal{A}} P$$

Remark 3.5.3 Andersen [17] proves that the size of $(\varphi \downarrow X) /\!\!/_{\mathcal{A}} P$ is exponentially larger that $(\varphi \downarrow X)$ in the worst case. Andersen also proposes heuristics that make $(\varphi \downarrow X) /\!\!/_{\mathcal{A}} P$ smaller while maintaining logic equivalence. Quoting Andersen "the strategies are generally valid but might or might not succeed in decreasing the size of the assertion" [17]. ■

3.6 Logic Characterization and Analysis of Fault-Tolerance

In this section we explain our first framework for the analysis of fault-tolerance. From Section 3.4, we recall that our proposal consists in viewing fault-tolerance analysis as the analysis of an open system; a fault-tolerant system is studied when acting in a faulty environment. Here, we characterize the problem of the analysis of fault-tolerance, with respect to a property, as a validation problem in the (equational) μ-calculus. Within this framework we reformulate, in fault-tolerance terms, a technique of validation studied in security protocol analysis [146, 148]. The technique is based on partial model checking. Moreover, we study an efficient solution [122] for checking the validity of a subclass of the μ-calculus. In the rest of the chapter, we do not make any distinction between μ-calculus and equational μ-calculus since these logics are equivalent and a formula in one logic can be transformed in an equivalent formula in the other logic in linear-time (see Section 3.5.2). Without lost of generality, we will refer only to the μ-calculus.

3.6.1 The Problem

Let us consider a system model P, its fault-tolerant (candidate) version $P^{\#}_{\mathcal{F}}$, and a μ-calculus formula ϕ expressing a desirable property of a system even in presence of faults. We are interested in understanding under which fault assumptions $P^{\#}_{\mathcal{F}}$ satisfies ϕ. This set can be formalized as follows:

$$\mathfrak{F}_{\phi}^{(P^{\#}_{\mathcal{F}} \parallel_{\mathcal{F}} F)} = \{F \in \mathcal{E}_{\mathcal{F}} : (P^{\#}_{\mathcal{F}} \parallel_{\mathcal{F}} F) \models \phi\} \tag{3.6.1}$$

Set $\mathfrak{F}_{\phi}^{(P^{\#}_{\mathcal{F}} \parallel_{\mathcal{F}} F)}$ characterizes the fault-tolerant capability of $P^{\#}_{\mathcal{F}}$ as the set of faulty environments that make $P^{\#}_{\mathcal{F}}$ preserve ϕ. If this set coincides with the class of all faulty environment $\mathcal{E}_{\mathcal{F}}$, that is if $\mathfrak{F}_{\phi}^{(P^{\#}_{\mathcal{F}} \parallel_{\mathcal{F}} F_{\mathcal{F}})} = \mathcal{E}_{\mathcal{F}}$, then it means that no faulty environment is able to force $P^{\#}_{\mathcal{F}}$ not to satisfy ϕ. This observation leads to a first logic characterization of fault-tolerance, as in the following definition:

Definition 3.6.1 (Logic Characterisation of Fault-Tolerance I) *A process $P^{\#}_{\mathcal{F}}$ is* fault-tolerant *with respect to the logical property ϕ if and only if*

$$\mathfrak{F}_{\phi}^{(P^{\#}_{\mathcal{F}} \parallel_{\mathcal{F}} F_{\mathcal{F}})} = \mathcal{E}_{\mathcal{F}}.$$

Supposing $\mathcal{M} = \{\mathcal{Q}, Q_0, Act_\tau, \rightarrow\}$ be a finite state LTS, where $\mathcal{Q} = \{Q_0, \ldots, Q_n\}$:

$$
\begin{array}{lcl}
(\varphi \downarrow X) /\!\!/_{\mathcal{A}} \mathcal{M} & = & (\varphi /\!\!/_{\mathcal{A}} \mathcal{M}) \downarrow X_{Q_0}, \\
\epsilon /\!\!/_{\mathcal{A}} \mathcal{M} & = & \epsilon \\
(X =_\sigma \phi)\varphi /\!\!/_{\mathcal{A}} \mathcal{M} & = & \left\{ \begin{array}{l} (X_{Q_1} =_\sigma \phi /\!\!/_{\mathcal{A}} Q_1) \\ \ldots \\ (X_{Q_n} =_\sigma \phi /\!\!/_{\mathcal{A}} Q_n) \\ \varphi /\!\!/_{\mathcal{A}} \mathcal{M} \end{array} \right. \\
X /\!\!/_{\mathcal{A}} Q & = & X_Q \\
\langle a \rangle \phi /\!\!/_{\mathcal{A}} Q & = & \langle a \rangle (\phi /\!\!/_{\mathcal{A}} Q) \vee \bigvee_{Q \xrightarrow{a} Q'} \phi /\!\!/_{\mathcal{A}} Q', \quad \text{if } a \neq \tau \wedge a \notin \mathcal{A} \cup \overline{\mathcal{A}} \\
\langle a \rangle \phi /\!\!/_{\mathcal{A}} Q & = & \mathtt{ff}, \quad \text{if } a \in \mathcal{A} \cup \overline{\mathcal{A}} \\
\langle \tau \rangle \phi /\!\!/_{\mathcal{A}} Q & = & \langle \tau \rangle (\phi /\!\!/_{\mathcal{A}} Q) \vee \bigvee_{Q \xrightarrow{\tau} Q'} \phi /\!\!/_{\mathcal{A}} Q' \vee \bigvee_{\left\{ \begin{array}{l} Q \xrightarrow{a} Q' \\ a \in \mathcal{A} \cup \overline{\mathcal{A}} \end{array} \right.} \langle \overline{a} \rangle (\phi /\!\!/_{\mathcal{A}} Q') \\
[a] \phi /\!\!/_{\mathcal{A}} Q & = & [a] (\phi /\!\!/_{\mathcal{A}} Q) \wedge \bigwedge_{Q \xrightarrow{a} Q'} \phi /\!\!/_{\mathcal{A}} Q', \quad \text{if } a \neq \tau \wedge a \notin \mathcal{A} \cup \overline{\mathcal{A}} \\
[a] \phi /\!\!/_{\mathcal{A}} Q & = & \mathtt{tt}, \quad \text{if } a \in \mathcal{A} \cup \overline{\mathcal{A}} \\
[\tau] \phi /\!\!/_{\mathcal{A}} Q & = & [\tau] (\phi /\!\!/_{\mathcal{A}} Q) \wedge \bigwedge_{Q \xrightarrow{\tau} Q'} \phi /\!\!/_{\mathcal{A}} Q' \wedge \bigwedge_{\left\{ \begin{array}{l} Q \xrightarrow{a} Q' \\ a \in \mathcal{A} \cup \overline{\mathcal{A}} \end{array} \right.} [\overline{a}] (\phi /\!\!/_{\mathcal{A}} Q') \\
(\phi_1 \wedge \phi_2) /\!\!/_{\mathcal{A}} Q & = & (\phi_1 /\!\!/_{\mathcal{A}} Q) \wedge (\phi_2 /\!\!/_{\mathcal{A}} Q), \\
(\phi_1 \vee \phi_2) /\!\!/_{\mathcal{A}} Q & = & (\phi_1 /\!\!/_{\mathcal{A}} Q) \vee (\phi_2 /\!\!/_{\mathcal{A}} Q) \\
\mathtt{tt} /\!\!/_{\mathcal{A}} Q & = & \mathtt{tt} \\
\mathtt{ff} /\!\!/_{\mathcal{A}} Q & = & \mathtt{ff}
\end{array}
$$

Figure 3.3: The partial evaluation function for $\|_{\mathcal{A}}$.

To check whether a model $P^{\#}_{\mathcal{F}}$ satisfies Definition 3.6.1 we have to solve the following problem:

$$\forall F \in \mathcal{E}_{\mathcal{F}}, \qquad (P^{\#}_{\mathcal{F}} \parallel F) \setminus \mathcal{F} \models \phi \tag{3.6.2}$$

Solving problem (3.6.2), that is checking the (candidate) system again any faulty environment is, in principle, useful. As an example let us consider $P = \overline{\mathtt{a}}.0 + \mathtt{f}.\mathtt{f}.\overline{\mathtt{b}}.0$. In P, two consecutive occurrences of a fault make the (good) action $\overline{\mathtt{b}}$ occur: here we think about one fault cancelling the effect of the other. Checking the fault-tolerance of P in a particular environment, for example as $F = \overline{\mathtt{f}}.F$ leads to the conclusion that, in $(P \parallel F) \setminus \{\mathtt{f}\}$, action $\overline{\mathtt{b}}$ eventually happens. Checking against an unspecified environment we can figure out, for example, that $F = \overline{\mathtt{f}}.0$ makes this property false in P.

From the point of view of the analysis Definition 3.6.1 is not practical. It requires to perform model checking against all environments. By exploiting partial model checking techniques we can provide a more suitable definition of the set that characterize the fault-tolerant capability of $P^{\#}_{\mathcal{F}}$, as follows:

$$\mathfrak{F}^{F}_{(\phi /\!\!/_{\mathcal{F}} P^{\#}_{\mathcal{F}})} = \{F \in \mathcal{E}_{\mathcal{F}} : F \models \phi /\!\!/_{\mathcal{F}} P^{\#}_{\mathcal{F}}\} \tag{3.6.3}$$

Set $\mathfrak{F}^{F}_{(\phi /\!\!/_{\mathcal{F}} P^{\#}_{\mathcal{F}})}$ characterizes the fault-tolerance of $P^{\#}_{\mathcal{F}}$ as the set of models of the formula $\phi /\!\!/_{\mathcal{F}} P^{\#}_{\mathcal{F}}$. If this set coincides with the set, $\mathcal{E}_{\mathcal{F}}$, of all possible (faulty environment) models, this means that $\phi /\!\!/_{\mathcal{F}} P^{\#}_{\mathcal{F}}$ is valid in $\mathcal{E}_{\mathcal{F}}$. Characterization (3.6.1) and (3.6.3) are equivalent as stated in the following proposition:

Proposition 3.6.2 $\mathfrak{F}^{(P^{\#}_{\mathcal{F}} \|_{\mathcal{F}} F)}_{\phi} = \mathfrak{F}^{F}_{(\phi /\!\!/_{\mathcal{F}} P^{\#}_{\mathcal{F}})}$

Proof. The thesis follows directly from Lemma 3.5.2. ∎

The characterization in (3.6.3) is easier to manage since it corresponds to a common representation of sets, and permits to define the analysis of a fault-tolerant process, with respect to a property ϕ, as a validity checking problem in the μ-calculus. It brings to the following alternative logic characterization of fault-tolerance:

Definition 3.6.3 (Logic Characterisation of Fault-Tolerance II) *A process $P^{\#}_{\mathcal{F}}$ is* fault-tolerant *with respect to the logical property ϕ if and only if $\phi /\!\!/_{\mathcal{F}} P^{\#}_{\mathcal{F}}$ is a valid formula in $\mathcal{E}_{\mathcal{F}}$.*

We prove that:

Proposition 3.6.4 *A process $P^{\#}_{\mathcal{F}}$ satisfies Definition 3.6.1 if and only if it satisfies Definition 3.6.3.*

Proof. From Proposition 3.6.2. ∎

Definition 3.6.3 and Proposition 3.6.4 state that for checking if a model $P^{\#}_{\mathcal{F}}$ satisfies this definition of fault tolerance, with respect to a property ϕ, we have to solve the following validation problem:

$$\forall F \in \mathcal{E}_{\mathcal{F}}, \qquad F \models \phi', \quad \text{where } \phi' = \phi /\!\!/_{\mathcal{F}} P^{\#}_{\mathcal{F}} \tag{3.6.4}$$

In the next section we study efficient solutions to this problem, with respect to time complexity.

3.6.2 Improving the Time Complexity of the Analysis

The validity (satisfiability) problem for the (equational) μ-calculus, such as (3.6.4), is generally EXPTIME complete [75, 193]. Better performances are reached on particular subclasses of the μ-calculus. For example, the satisfiability problem in the disjunctive subclass of the μ-calculus is linear time in the size of the formula [203]. It follows that the validity problem for those formulas whose complement falls in the disjuntive subclass of the μ-calculus (*i.e.*, the conjunctive subclass of μ-calculus) can be solved in linear time.

Our framework use the partial model checking to reduce the fault-tolerance checking problem to a validity problem in the μ-calculus. To obtain an efficient strategy of analysis, we have to look for a subclass of μ-calculus that is closed under the partial evaluation function of the partial model checking and whose complemented class fall in the disjuntive subclass of the μ-calculus. In this section we define the *universal conjunctive* subclass of the μ-calculus (we write in short, $\forall_{\wedge} MC$), and we prove that

- $\forall_{\wedge} MC$ is closed under the partial evaluation function $(\_) /\!\!/_{\mathcal{F}} P$;

- the class of the complemented $\forall_\wedge MC$ formulas is strictly included in the disjunctive μ-calculus.

Remark 3.6.5 We use the two previous results to prove that (3.6.4) is solvable in time linear in the size of ϕ'. ■

Remark 3.6.6 The size of the formula obtained after the partial model checking procedure is polynomial in the size of the process and the formula, hence it can be, in the worst case, exponentially longer than the original formula. Thus the effectiveness of our solution methods depends also on the success of the heuristics that Andersen proposes to make $\phi /\!\!/_{\mathcal{F}} P^{\#}_{\mathcal{F}}$ smaller while maintaining logic equivalence. ■

In the next sections we summarize the definitions and results about the disjunctive subclass of the μ-calculus; then we introduce the *universal conjunctive* subclass of the μ-calculus and we show how our validation problem can be solved efficiently in this class.

Disjunctive μ-calculus

From [120] we reproduce the definition of the disjunctive μ-calculus. Formulas in this subclass of the μ-calculus, called disjunctive formulas, are interesting because their satisfiability problem can be solved in time linear in the size of the formula [120]. Their definition depends on the definition of the following special class of formulas:

Definition 3.6.7 (Special Conjuntive Formulas) *A conjunction $\alpha_1 \wedge \ldots \wedge \alpha_n$ is* special *if and only if every α_i is either a literal or a formula of the form*[1] *$(a \rightarrow \Phi)$ and for every action a there is at most one conjunct of the form $(a \rightarrow \Phi)$ among $\alpha_1, \ldots, \alpha_n$.*

Definition 3.6.8 (Disjunctive μ-calculus formulas) *The set of* disjunctive *μ-calculus formulas is the smallest set $\mathcal{D}$ defined by the following clauses:*

- *every literal is a disjunctive formula,*
- *if $\alpha, \beta \in \mathcal{D}$ then $\alpha \vee \beta \in \mathcal{D}$. Moreover if X occurs only positively in α, and does not occur in the context $X \wedge \gamma$, for some γ, then $\mu X.\alpha$, $\nu X.\alpha \in \mathcal{D}$.*
- *$(a \rightarrow \Phi) \in \mathcal{D}$ if $\Phi \subseteq \mathcal{D}$*
- *special conjunctive formulas are disjunctive formulas*

The following theorems hold [120]:

Theorem 3.6.9 *For every μ-calculus formula there exists an equivalent disjunctive μ-calculus formula.*

Theorem 3.6.10 *Satisfiability checking for a disjunctive μ-calculus formula can be done in linear time in the size of the formula.*

Remark 3.6.11 Theorems 3.6.9 and 3.6.10 lead to the conclusion that the transformation from μ-calculus to disjunctive μ-calculus introduces, in the worst case, an exponential blow up. We will avoid this problem by expressing our formulas directly in a subclass of the disjunctive μ-calculus. ■

[1] The μ-calculus construct $a \rightarrow \Phi$, where a is an action and Φ a finite set of μ-calculus formulas, is an abbreviation for $\bigwedge\{[a]\phi : \phi \in \Phi\} \wedge [a] \bigvee \Phi$ [203]. $\langle a \rangle \phi$ is equivalent to $a \rightarrow \{\phi, \mathtt{tt}\}$, while $[a]\phi$ is equivalent to $(a \rightarrow \mathtt{ff}) \wedge (a \rightarrow \{\phi\})$ so any μ-calculus formula can be written using this new construct.

$\forall_\wedge MC$, the Universal Conjunctive μ-calculus

We now identify and define the $\forall_\wedge MC$; then we prove our main results, namely:

- $\forall_\wedge MC$ is closed under the partial evaluation function $(\_)/\!\!/_{\mathcal{F}} P$ (Lemma 3.6.13);
- the class of complemented $\forall_\wedge MC$ formulas is strictly included in the disjunctive μ-calculus (Lemma 3.6.14).

Definition 3.6.12 (Universal Conjuntive Formulas) *The set, $\forall_\wedge MC$, of universal conjunctive μ-calculus formulas is the largest subset of μ-calculus formulas that can be written without either the $\vee$ operator or the $\langle\_\rangle$ modality. The formulas of the $\forall_\wedge MC$ are generated by the following grammar:*

$$\phi ::= \mathtt{ff} \mid \mathtt{tt} \mid Z \mid \phi \wedge \phi' \mid [K]\phi \mid \mu Z.\phi \mid \nu Z.\phi$$

Lemma 3.6.13 *$\forall_\wedge MC$ is closed under the partial model checking function $/\!\!/_{\mathcal{F}} P$.*

Proof. By definition of $(\_)/\!\!/_{\mathcal{F}} P$ (see Figure 3.3). For all $\mathcal{F}$ and all CCS process P, $(\_)/\!\!/_{\mathcal{F}} P$ preserves $\wedge$ and $[\_]$, while the transformation of $[\_]$ introduces only $\wedge$. ■

Lemma 3.6.14 *If $\phi \in \forall_\wedge MC$, the complement formula of ϕ, ϕ^c, is disjunctive in the sense of Definition 3.6.8.*

Proof. By structural induction over ϕ. If ϕ is a literal, or if $\phi = \phi_1 \wedge \phi_2$ the lemma holds trivially. If $\phi = [K]\phi_1$, then $\phi^c = \langle K\rangle {\phi_1}^c$. In this case the lemma holds since $\langle K\rangle {\phi_1}^c$ can be written as $\langle K\rangle \rightarrow \{{\phi_1}^c, \mathtt{tt}\}$, and ${\phi_1}^c$ is disjunctive by the induction hypothesis. If $\phi = \mu X.\phi_1(X)$ then $\phi^c = \nu X.{\phi_1}^c(X)$ and ${\phi_1}^c$ is disjunctive by the induction hypothesis. The case $\phi = \nu X.\phi_1(X)$ is treated similarly. This concludes the proof. ■

The previous results are the foundation of our solution method for (3.6.4), synthesized by the following theorem:

Theorem 3.6.15 *If $\phi' \in \forall_\wedge MC$ then (3.6.4) can be solved in time linear in the size of ϕ'.*

Proof. Problem (3.6.4) requires the validity check of ϕ'. The $\forall_\wedge MC$ formula ϕ' is valid if and only if the complement formula ${\phi'}^c$ is not satisfiable. ${\phi'}^c$ can be obtained in linear time from ϕ'. By Lemma 3.6.14 ${\phi'}^c$ is disjuntive, and by Theorem 3.6.10 the satisfiability of ${\phi'}^c$' is solvable in linear time in its size. This concludes the proof. ■

Corollary 3.6.16 *If $\phi \in \forall_\wedge MC$ then (3.6.2) is answered in time linear in the size of $\phi/\!\!/_{\mathcal{F}} P$.*

Proof. By partial model checking, $\forall F \in \mathcal{E}_{\mathcal{F}}$, $(P^{\#}_{\mathcal{F}} \parallel F) \setminus \mathcal{F} \models \phi$ if and only if $\forall F \in \mathcal{E}_{\mathcal{F}}$, $F \models \phi/\!\!/_{\mathcal{F}} P^{\#}_{\mathcal{F}}$. By Lemma 3.6.13 $\phi/\!\!/_{\mathcal{F}} P^{\#}_{\mathcal{F}}$ is in the $\forall_\wedge MC$, and the Corollary follows immediately from Theorem 3.6.15. ■

3.6.3 $\forall_\wedge MC$ Formulas in Fault-Tolerance

This section discusses the role of $\forall_\wedge MC$ in fault-tolerance. We start with a general discussion followed by a list of concrete properties. Those properties, taken from the literature, are divided into two sublists: the former contains properties that fall in $\forall_\wedge MC$, the latter contains those that do not fall in $\forall_\wedge MC$.

Suitability of $\forall_\wedge MC$ in Fault-Tolerance: a Discussion

Accordingly to [19], fault-tolerance properties can be divided informally into the following categories:

- *fault-tolerance* if the system delivers a correct answer despite faults;
- *fail-silence* if system failures can only be omission failures *i.e.,* it gives either a correct answer or no answer;
- *fail-stop* if, in case of faults, the system terminates;
- *fail-safe* if the system, in case of faults, enters into a state in which no catastrophic event occur.

Depending on the particular system some of the previous properties are formalized as *safety* or *liveness* properties; others are formalized as a combination of them, *i.e.,* they are neither safety nor liveness properties. Safety and liveness properties where first described by Lamport in [127], who studied linear-time properties of reactive systems. He suggested that the intuitive meaning of safety is "nothing bad happens in the lifetime of the system", while the meaning of liveness is "something good eventually happens" [127].

In the following we give an intuition to understand what kind of properties are expressible with $\forall_\wedge MC$. We need to refer to a formal definition of safety and liveness. In the rest of the chapter we will use the following linear-time semantic definitions of safety and liveness [10, 166, 143], that correspond to the branching-time definitions of universally safety and universally liveness properties respectively [144].

Definition 3.6.17 (Semantic Characterization of Safety and Liveness) *Let P be a property expressed as a sequence of events. P is a:*

safety property, *if and only if every infinite sequence of events that does not satisfy this property contains a finite prefix such that no infinite sequences obtained by adding an infinite suffix to this finite prefix satisfies this property;*

liveness property, *if and only if for every finite sequence we can find an infinite suffix, so that the resulting infinite sequence satisfies the property.*

Remark 3.6.18 Alene and Schneider proved in [10] that any property can be classified as a safety or a liveness property, or an intersection of them. ■

"Fail silence", "fail stop" and "fail safe" enjoy a common structure [19]. In the μ-calculus, the previous properties are expressible by formulas whose external form is $\nu X.[\mathcal{F}]\phi \wedge [-]X$, where ϕ expresses respectively a behavior without faults, a stop, or a safe behavior: if $\phi \in \forall_\wedge MC$, formulas in this form fit $\forall_\wedge MC$. Properties expressing "fault-tolerance" are more general and do not have a common form. In this case to understand if they fit $\forall_\wedge MC$ we have to study each formula separately. As a general consideration we note that safety properties of the form $\nu X.\phi \wedge [-]X$, expressing "no bad state is ever reached" (here ϕ^c, the complement of ϕ, holds in the bad state) will fit $\forall_\wedge MC$ only if ϕ fits. On the other hand, safety properties of the form $\nu X.[K]\mathtt{ff} \wedge [-]X$, expressing "no bad action in K ever happens" do fit $\forall_\wedge MC$.

Liveness properties are a bit more tricky. For example, the general formulation of a liveness property that involves a condition over a state *i.e.*, expressing that "a state satisfying ϕ is eventually reached" is $\mu X.\phi \vee (\langle - \rangle \mathtt{tt} \wedge [-]X)$ [190]. This form does not fit $\forall_\wedge MC$ because of the $\langle - \rangle$ and the $\wedge$.

The liveness property expressing "an action a eventually occurs" requires a discussion. Its μ-calculus formulation, $\mu X.\langle - \rangle \mathtt{tt} \wedge [-a]X$ [37], does not fit $\forall_\wedge MC$. The subformula $\langle - \rangle \mathtt{tt}$ is required to avoid that the formula is trivially satisfied because the system deadlocks. If we assume to check deadlock freedom separately, this conjunct can be removed and the weaker formulation, $\mu X.[-a]X$, fits $\forall_\wedge MC$.

Other categories of formulas, which are neither safety nor liveness properties, (*e.g.*, some cyclic properties) have to be considered one by one. We anticipate that witnesses of these formulas fall in $\forall_\wedge MC$. We are now ready to give concrete examples of each category of formula in the next section. This preliminary discussion also suggests how the $\forall_\wedge MC$ formulas are related to the set of fault-tolerance properties as illustrated in Figure 3.4.

Examples of $\forall_\wedge MC$ Formulas in Fault-Tolerance

This section provides examples of fault-tolerance properties that do fit $\forall_\wedge MC$ (positive examples) and that do not fit $\forall_\wedge MC$ (negative examples). providing examples of (categories of) formulas in

Positive Example

Our examples consist of a list of $\forall_\wedge MC$ properties taken from [188, 37, 190]. For all the properties, we underline their use in fault-tolerance.

Safety Properties

Nothing Bad (positive example [190], page 128–130). Let ϕ^c be a property that holds in a bad state. The general form expressing that "in case of some fault in $\mathcal{F}$ the bad state is never reached" is expressible as follows:

$$NothingBad(\phi) \stackrel{\text{def}}{=} \nu Z.[\mathcal{F}]\phi \wedge [-]Z$$

This class of safety formulas is in $\forall_\wedge MC$ only if ϕ^c is in $\forall_\wedge MC$. A necessary condition for this to happen is, for example, that ϕ^c does not contain conjunctions.

Never ([37] page 42, [190] page 128). Let K be the set of transitions indicating a bad behavior. Formula

$$Never(K) \stackrel{\text{def}}{=} \nu Z.[K]\mathtt{ff} \wedge [-]Z$$

expresses the safety property "no transition in K ever happens"[2]. It can be used to express the following fail safe property "in case of a fault in $\mathcal{F}$, no bad transition ever happens":

$$\nu X.[\mathcal{F}]Never(K) \wedge [-]X$$

[2]Another way of interpreting $Never(K)$ is that the bad state $\langle K \rangle \mathtt{tt}$ is ever reached.

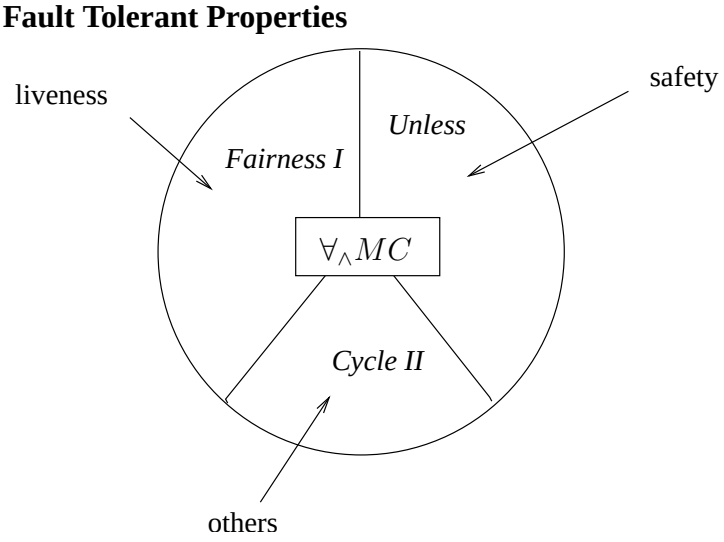


Figure 3.4: Fault-tolerance properties that are expressible as $\forall_\wedge MC$ formulas. $\forall_\wedge MC$ is able to express many safety property and some liveness property. See text for more details.

Deadlock or Termination ([188], page 7).

$$NoObservableAction \stackrel{\text{def}}{=} [[-]]\texttt{ff}$$

The previous formula expresses that the system is incapable of performing any observable action, that is it either deadlocks or stops. This property can be used to express the fail stop property "in case of faults $\mathcal{F}$, the system deadlocks or terminates":

$$\nu X.[\mathcal{F}]NoObservableAction \wedge [-]X$$

Unless ([190], page 43).

$$Unless(K, J) \stackrel{\text{def}}{=} \mu Z.[-(K \cup J)]\texttt{ff} \wedge [-J]Z$$

This formula expresses "in any run actions from K happen unless a J action occurs". This property where an action in J may not eventually occur, implements the weak version of the until modality [35]. It can be used to express fault-tolerance, or as a building block to express fail safe as follows:

$$\nu X.[\mathcal{F}]\,Unless(K, J) \wedge [-]X$$

Cycle I ([37] page 42) Properties expressing cycles are in $\forall_\wedge MC$. The simplest example is the following safety property:

$$Cycle(a, b) \stackrel{\text{def}}{=} \nu Z.[b]\texttt{ff} \wedge [-a, b]Z \wedge [a](\nu Y.[a]\texttt{ff} \wedge [-a, b]Y \wedge [b]Z)$$

expressing "two actions a and b occur in alternation". A sequence (also infinite) of other actions is allowed to interleave between an a and the next b. It expresses fault-tolerance, or it can be used to express a fail safe behavior as follows:

$$\nu X.[\mathcal{F}]Cycle(a,b) \wedge [-]X$$

Formulas expressing cycles with more than two actions also fall in $\forall_\wedge MC$ [190].

Liveness Properties

Fairness I ([37], page 110). For an agent with sort $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$

$$Fairness(b;a) \stackrel{\text{def}}{=} \nu Z.\mu Y.\nu W.[[b]]Z \wedge [[a]]Y \wedge [[c,\epsilon]]W$$

expresses the liveness property "no infinite sequence can be performed containing infinitely many occurrences of a, but no occurrences of b". It represents a fair behavior in case of some fault, for example as in the following fail safe property:

$$\nu X.[\mathcal{F}]Fairness(b;a) \wedge [-]X$$

Fairness II ([190], page 130). Another $\forall_\wedge MC$ fairness property is:

$$Fairness'(a;b,c) \stackrel{\text{def}}{=} \nu Z.(\mu Y_1.[b](\nu Y_2.[c](\nu Y_3.Y_1 \wedge [-a]Y_3) \wedge [-a]Y_2) \wedge [-]Z$$

Informally, it says "in any run, if b and c happen infinitely often, than so does a". This formula can be used to express the property saying and that "in any run, fairness holds in case of some fault in $\mathcal{F}$ occur":

$$\nu X.[\mathcal{F}]Fairness'(a;b,c) \wedge [-]X \tag{3.6.5}$$

Finitely Often ([190], page 132). Another liveness property that falls in $\forall_\wedge MC$ is:

$$FinOft(a) \stackrel{\text{def}}{=} \mu Z.\mu Y.[a]Z \wedge [-a]Y$$

expressing "in each run, a can only happen finitely often". Used as an invariant, it expresses fault-tolerance. It can be also used to express either a fail silent or a fail safe behavior, as follows:

$$\nu X.[\mathcal{F}]FinOft(a) \wedge [-]X$$

Other Properties

Some properties that are neither safety nor liveness properties fall in $\forall_\wedge MC$ also. For example, the following variants of the cyclic property $Cycle(a,b)$:

Cycle II ([190], page 131-132)

$$Cycle'(a,b) \stackrel{\text{def}}{=} \mu Y.[b]\mathtt{ff} \wedge [a](\mu Z.[a]\mathtt{ff} \wedge [b]Y \wedge [-b]Z) \wedge [-b]Y$$

This property expresses the property "two actions a and b do occur in alternation" with the constraint that no intervening actions are allowed to continue forever without the next a or b happening. The variant using a greatest fixed point

$$Cycle''(a,b) \stackrel{\text{def}}{=} \nu Y.[b]\mathtt{ff} \wedge [a](\mu Z.[a]\mathtt{ff} \wedge [b]Y \wedge [-b]Z) \wedge [-b]Y$$

also falls in $\forall_\wedge MC$. It expresses the same property as $Cycle'(a,b)$ does, with the constraint "other actions can intervene forever between an a and the next b, but whenever an a happens b must eventually happen".

Negative Examples

This section lists examples of formulas that do not fit $\forall_\wedge MC$.

Nothing Bad (negative example [190], page 128–130] Let ϕ^c be a condition that holds in a bad state. The general form expressing the safety property that nothing bad happens is:

$$NothingBad(\phi) \stackrel{\text{def}}{=} \nu X.[\mathcal{F}]\phi \wedge [-]X$$

If $\phi \notin \forall_\wedge MC$, this family of properties fall outside in $\forall_\wedge MC$; a sufficient condition for this to happen is when ϕ^c contains conjunctions.

Deadlock Freedom ([37], page 109).

$$DeadlockFree \stackrel{\text{def}}{=} \mu X.\langle\langle -\epsilon \rangle\rangle \mathtt{tt} \wedge [[-]]X$$

Deadlock freedom cannot be expressed in $\forall_\wedge MC$, because we cannot avoid the $\langle\langle \_ \rangle\rangle$ moda lity. Deadlock freedom is a particular instance of the formulas expressing the "eventually" modality.

Eventually ([37], page 43). In this category we find, for example, the formula saying "an action a eventually happens":

$$Eventually(a) \stackrel{\text{def}}{=} \mu X.\langle -\rangle \mathtt{tt} \wedge [-a]X$$

If we want to exclude that the property holds because the system deadlocks (see the discussion in section 3.6.3), this property does not fit $\forall_\wedge MC$. Also the generalization

$$Eventually(\phi) \stackrel{\text{def}}{=} \mu X.\phi \vee (\langle -\rangle \mathtt{tt} \wedge [-a]X)$$

expressing "a closed formula ϕ eventually holds" does not fall in $\forall_\wedge MC$ either. As a consequence, the $\forall_\wedge MC$ fragment cannot express any formula containing the "eventually" modality, for instance "always eventually".

The last negative example is "strong until":

Strong Until ([37], page 43):

$$StrongUntil(\phi, \varphi) \stackrel{\text{def}}{=} \mu X.\phi \vee (\varphi \wedge \langle - \rangle \mathtt{tt} \wedge [-]X)$$

where X does not occur in ϕ. Informally strong until says "ϕ holds until φ becomes true". It also requires that φ becomes eventually true. Strong until needs the modality $\langle\rangle$ and, consequently, does not fall in $\forall_{\wedge} MC$.

3.6.4 Our Running Example

This section shows a CCS model of a simple fault-tolerant system, and shows how we check a fault-tolerance property in the framework of the analysis proposed so far.

Let us consider a different version of our fault-tolerant battery specified in Example 3.4.1 (see also Figure 3.5). In this more sophisticated version the controller module also acts as a failure detector: if both batteries do not produce a valid burst of energy, it returns the message $\overline{\mathtt{fail}}$. The CCS model is as follows:

$$\begin{aligned}
Det &\stackrel{\text{def}}{=} \mathtt{ret}_{1,0}.Det' + \mathtt{ret}_{1,1}.\overline{\mathtt{ret}}.Det'' \\
Det' &\stackrel{\text{def}}{=} \mathtt{ret}_{2,0}.\overline{\mathtt{fail}}.Det''' + \mathtt{ret}_{2,1}.\overline{\mathtt{ret}}.Det''' \\
Det'' &\stackrel{\text{def}}{=} \mathtt{ret}_{2,0}.Det''' + \mathtt{ret}_{2,1}.Det''' \\
Det''' &\stackrel{\text{def}}{=} \overline{\mathtt{ack}}.Det
\end{aligned}$$

We now build two new fault-tolerant batteries we call $Ene^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}}$ and $Energiz^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}}$ respectively. In the former (see Figure 3.5) we include one faulty battery, and in the second two faulty batteries:

$$\begin{aligned}
Ene^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}} &\stackrel{\text{def}}{=} (Spl \parallel Ene^{(1)} \parallel Ene^{(2)}_{\{\mathtt{f}_0,\mathtt{f}_1\}} \parallel Det) \setminus \mathcal{A} \\
Energiz^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}} &\stackrel{\text{def}}{=} (Spl \parallel Ene^{(1)}_{\{\mathtt{f}_0,\mathtt{f}_1\}} \parallel Ene^{(2)}_{\{\mathtt{f}_0,\mathtt{f}_1\}} \parallel Det) \setminus \mathcal{A} \\
&\text{where } \mathcal{A} = \{\mathtt{get}_1, \mathtt{get}_2, \mathtt{ret}_{1,0}, \mathtt{ret}_{1,1}, \mathtt{ret}_{2,0}, \mathtt{ret}_{2,1}, \mathtt{ack}\}.
\end{aligned}$$

Let us now consider the following $\forall_{\wedge} MC$ formula expressing the safety property "in any run, action `fail` never occurs"

$$\phi \stackrel{\text{def}}{=} \nu X.([\overline{\mathtt{fail}}]\mathtt{ff} \wedge [-]X) \tag{3.6.6}$$

Equivalently, we can consider its equational version $X =_{\nu} ([\overline{\mathtt{fail}}]\mathtt{ff} \wedge [-]X)$. We want to prove that property (3.6.6) holds on $Ene^{\#}\{\mathtt{f}_0, \mathtt{f}_1\}$ even in case of faults. The scenario of analysis is:

$$\forall F_{\{\mathtt{f}_0,\mathtt{f}_1\}} \in \mathcal{E}_{\{\mathtt{f}_0,\mathtt{f}_1\}}, \qquad (Ene^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}} \parallel F_{\{\mathtt{f}_0,\mathtt{f}_1\}}) \setminus \{\mathtt{f}_0, \mathtt{f}_1\} \models \phi \tag{3.6.7}$$

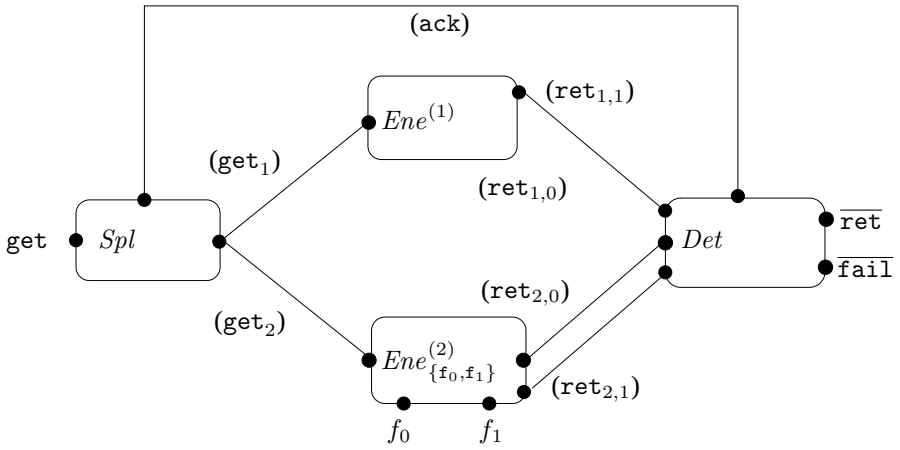


Figure 3.5: The flow diagram of the (candidate) fault-tolerant version of the battery, $Ene^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}}$. Actions in brackets are internal actions.

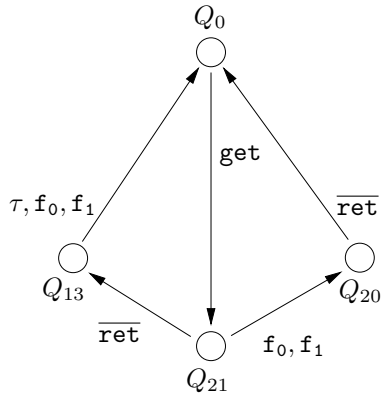


Figure 3.6: The minimum automata weak-bisimilar to $Ene^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}}$.

Here, to keep the size of the model small, and the analysis tractable, we use the minimum process weak bisimilar to $Ene^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}}$, reported in Figure 3.6. The use of a weakly bisimilar process here is justified by the fact that we are analyzing fault-tolerance at the abstraction level of an external observer. This means also that properties are intended in their observational-based interpretation; the equivalence of the analysis is so preserved. The partial evaluation, $\phi /\!\!/_{\{\mathtt{f}_0,\mathtt{f}_1\}} Ene^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}}$ produces the following formula:

$$\begin{array}{rlll} \phi /\!\!/_{\{\mathtt{f}_0,\mathtt{f}_1\}} Ene^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}} = & (X_0 & =_\nu [\overline{\mathtt{fail}}]\mathtt{ff} \wedge ([-\mathcal{F}]X_0 \wedge X_1) \\ & (X_1 & =_\nu [\overline{\mathtt{fail}}]\mathtt{ff} \wedge ([-\mathcal{F}]X_1 \wedge X_2 \wedge [\overline{\mathtt{f}_0}, \overline{\mathtt{f}_1}]X_3) \\ & (X_2 & =_\nu [\overline{\mathtt{fail}}]\mathtt{ff} \wedge ([-\mathcal{F}]X_2 \wedge [\overline{\mathtt{f}_0}, \overline{\mathtt{f}_1}]X_3)) \\ & (X_3 & =_\nu [\overline{\mathtt{fail}}]\mathtt{ff} \wedge ([-\mathcal{F}]X_3 \wedge X_0) \end{array} \tag{3.6.8}$$

where $\mathcal{F} = \{\overline{\mathtt{f}_0}, \overline{\mathtt{f}_1}, \mathtt{f}_1, \mathtt{f}_0\}$.

If we want to answer to the question "for every environment does the energizer satisfy the formula ϕ" we have to check the validity of formula (3.6.8). With the theory we described, this can be done in linear time. Let us observe that the result in [120] (*i.e.*, that the satisfiability problem

for μ-calculus disjunctive formulas can be solved in linear time) is not still directly applicable. For using it, we need to answer (3.6.7) by checking the satisfiability for the complement of formula $\phi /\!\!/_{\{\mathtt{f}_0,\mathtt{f}_1\}} Ene^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}}$, which proves to be exactly disjunctive in the sense of Definition 3.6.8.

3.7 Background on Observational Properties

This section opens the second part of the chapter, where we characterize fault-tolerance in terms of GNDC. First, we summarize the basic background to understand the GNDC characterization of fault-tolerance. In Section 3.7.1 we recalls the notion of observational equivalences among CCS processes. In Section 3.7.2 we summarise the definitions of NDC, BNDC, SNNI, BSNNI, and SBSNNI; these definitions formalize in CCS basic non-interference properties.

Later, in Section 3.8 we present our characterization of fault-tolerance in the GNDC, and we show how to re-use, in fault-tolerance strategies analysis techniques proper of the non-interference.

3.7.1 Observational Equivalences among Processes

Properties over a system model can be expressed also by comparing its behavior with that of another model. As we have done in Section 3.5.3, we consider the case in which an external observer cannot see τ actions; hence, actions τ are abstracted from our definition of process behavior. We use *trace equivalence* and *weak bisimulation* [159, 83] as binary relations to compare the behavior of two processes. These relations represent two different notions of observational equivalence among processes. Informally, the former states that the sets of traces of two trace equivalent processes appear the same to an external observer; the latter affirms that two weakly bisimilar processes share also the branching structure of their labelled transition systems.

Since we focus on observable actions, both previous relations implicitly refer to the observable transition $\stackrel{a}{\Longrightarrow}$, defined in Section 3.5.3. We now recall the formal definition of trace equivalence and weak bisimulation from [83]:

Definition 3.7.1 *Let $\tilde{a} = a_1 \dots a_n \in Act^*$ be a sequence of actions. We write $P \stackrel{\tilde{a}}{\Longrightarrow} P'$ if and only if there exist $P_1, \dots, P_n \in \mathcal{E}$ such that $P \stackrel{a_1}{\Longrightarrow} P_1 \stackrel{a_2}{\Longrightarrow} \dots \stackrel{a_n}{\Longrightarrow} P_n$. Let $\mathcal{T}(P) = \{\tilde{a} \in Act^* : \exists P', P \stackrel{\tilde{a}}{\Longrightarrow} P'\}$ be the set of traces associated to a process P. We have that a CCS process Q can execute all the traces of a CCS process P (written $P \leq_{trace} Q$) if and only if $\mathcal{T}(P) \subseteq \mathcal{T}(Q)$. Two processes P and Q are said to be* trace equivalent *(written $P \approx_t Q$) if and only if $P \leq_{trace} Q$ and $Q \leq_{trace} P$.*

Remark 3.7.2 Relation $\approx_t$ can be defined also as $\leq_{trace} \cap (\leq_{trace})^{-1}$. ■

The general notion of bisimulation [158] consists of a mutual step-by-step simulation: given two processes P and Q when P executes a certain action moving to P' then Q must be able to simulate the single step by executing the same action and moving to a term Q' which is again bisimilar to P', and vice-versa. A weak bisimulation is a bisimulation which does not care about internal τ actions.

We write $P \stackrel{\widehat{a}}{\Longrightarrow} P'$ for $P \stackrel{a}{\Longrightarrow} P'$ if $a \in Act$ and for $P(\stackrel{\tau}{\longrightarrow})^* P'$ if $a = \tau$. Note that $P \stackrel{\widehat{\tau}}{\Longrightarrow} P'$ means that P evolves in P' with zero or more τ.

Definition 3.7.3 *A relation $\mathcal{R}$ on $\mathcal{E} \times \mathcal{E}$ is a* weak bisimulation *if for each $(P, Q) \in \mathcal{R}$ and for each $a \in Act_\tau$:*

$$\text{if } P \xrightarrow{a} P' \quad \text{then} \quad \exists Q' : Q \stackrel{\widehat{a}}{\Longrightarrow} Q' \text{ and } (P', Q') \in \mathcal{R}$$
$$\text{if } Q \xrightarrow{a} Q' \quad \text{then} \quad \exists P' : P \stackrel{\widehat{a}}{\Longrightarrow} P' \text{ and } (P', Q') \in \mathcal{R}$$

Two processes P and Q are weakly bisimilar *(written $P \approx Q$) if a weak bisimulation relation $\mathcal{R}$ exists such that $(P, Q) \in \mathcal{R}$.*

Remark 3.7.4 $P \approx Q$ implies $P \approx_t Q$ [83]. ■

Weak bisimulation is able to detect most of the safety properties we use in this chapter, such as fail safe, fail silent, fail stop, and fault-tolerance. Some liveness properties, such as deadlock freedom, can be caught too; instead, the use of weak bisimulation does not allow to distinguish between deadlocks and livelocks [37].

3.7.2 Information Flow and Non Interference Properties

Information flow properties have been introduced to study and control flow of information among different entities. Many information flow properties have been uniformly formalized in a CCS-like process algebraic setting [80, 81]. The common intuition behind these properties is strictly related to the classic notion of non-interference [101], which aims to control the information flow between two levels of user, low and high. Basically, non-interference says "no low level user is able to deduct anything about the activity of a high level user". Non-interference properties have been also restated in terms of network security [85], where high users represent network intruders, and low level users model cryptographic protocols.

Among the many formalizations of non-interference properties we are interested in the *Non Deducibility on Compositions* (NDC, for short), expressed in CCS as follows [3]:

$$P \in NDC \qquad \text{iff} \qquad \forall X \in \mathcal{E}_{\mathcal{H}} : (P \parallel X) \setminus \mathcal{H} \approx_t P \setminus \mathcal{H} \tag{3.7.1}$$

In (3.7.1) $\mathcal{E}_{\mathcal{H}}$, where $\mathcal{H} \subset Act$ is the set of all processes whose sort is the set of high actions. The NDC is defined in terms of a trace equivalence. The version of NDC that uses weak bisimulation, instead of trace equivalence, is called bisimulation-based NDC (in short, BNDC). Properties NDC and BNDC (we write (B)NDC when we do not want to distinguish between them) can be read as "no high level activity can change the low level observational behavior". In fact, in (3.7.1) $P \setminus \mathcal{H}$ exhibits only the low level behavior of P, while $(P \parallel X) \setminus \mathcal{H}$ is the low level behavior of $P \parallel X$.

Remark 3.7.5 From the informal reading of (B)NDC we can foresee its reading in fault-tolerance: "no faulty environment (high level activity) can change the fault-tolerant system (low level) behavior". ■

A serious obstacle to the verification of (B)NDC is the universal quantification over all the possible $X \in \mathcal{E}_{\mathcal{H}}$. In [83] two possible solutions are suggested and studied:

[3] Indeed, in [80] NDC has been originally formalized in SPA (Security Process Algebra) which is basically CCS where the set of actions are partitioned into the sets $\mathcal{H}$ and L of high and low actions.

(Solution A) to define a most powerful enemy (with respect to a behavioral equivalence relation) in such a way that the universal quantification over all possible enemies can be removed in favor of a single check against the most powerful enemy.

(Solution B) to prove other properties over P, stronger than (B)NDC, that do not require any quantification.

Solution A, is based on the following definition and proposition.

Definition 3.7.6 *A relation over processes, $\lhd$, is a* precongruence *with respect to the CCS operators $\|$ and $\setminus$ if the following property holds, for all $P, Q, X \in \mathcal{E}$ and $\mathcal{A} \subseteq Act$:*

$$P \lhd Q \text{ implies } (P \parallel X) \setminus \mathcal{A} \lhd (Q \parallel X) \setminus \mathcal{A}$$

The following results holds for precongruences [86, 85]:

Proposition 3.7.7 *$\leq_{trace}$ is a precongruence with respect to the CCS operators $\|$ and $\setminus$.*

Proposition 3.7.8 *Let be $\lhd$ a precongruence with respect to $\|$ and $\setminus$. If there exists two processes $Top \in \mathcal{E}_{\mathcal{H}}$ such that for every process $X \in \mathcal{E}_{\mathcal{H}}$ we have $X \lhd Top$, then*

$$P \in NDC_{\lhd} \qquad \textit{iff} \qquad (P \parallel Top) \setminus \mathcal{H} \lhd P \setminus \mathcal{H}$$

In Proposition 3.7.8 we have used a generalized version of the NDC, where a generic precongruence $\lhd$ is used instead of the trace equivalence $\approx_t$. Proposition 3.7.8 implies also the following corollary about the congruence relation induced by a precongruence $\lhd$ [85]:

Corollary 3.7.9 *Let $\lhd$ be a precongruence with respect to $\|$ and $\setminus$, and let $\sim \stackrel{def}{=} \lhd \cap (\lhd)^{-1}$. If there exists two processes $Bot, Top \in \mathcal{E}_{\mathcal{H}}$ such that for every process $X \in \mathcal{E}_{\mathcal{H}}$ we have $Bot \lhd X \lhd Top$, then*

$$P \in NDC_{\sim} \qquad \textit{iff} \qquad (P \parallel Bot) \setminus \mathcal{H} \sim (P \parallel Top) \setminus \mathcal{H} \sim P \setminus \mathcal{H}$$

In summary, solution A is based on the existence of a most powerful process Top. If we can find a process Top such that $\forall X \in \mathcal{E}_{\mathcal{H}}, X \lhd Top$, then checking NDC against Top is necessary and sufficient for checking NDC against all X in $\mathcal{E}_{\mathcal{H}}$: the quantification is removed in favor of single check against the (albeit huge) process Top.

Solution B exploits the following SNNI (acronym for, Strong Non-deterministic Non Interference) property:

$$P \in SNNI \qquad \text{iff} \qquad P \setminus \mathcal{H} \approx_t P/\mathcal{H} \tag{3.7.2}$$

Here $/$ is the CCS hide operator [83]: $P/\mathcal{H}$ is the process P where all actions in $H \cup \overline{\mathcal{H}}$ are replaced by a τ action. SNNI is defined in terms of a trace equivalence; the version using weak bisimulation is called Bisimulation-based SNNI (in short, BSNNI).

Remark 3.7.10 SNNI and BSNNI can be checked by exploiting only local conditions. No universal quantification over $\mathcal{E}_{\mathcal{H}}$ is required. ■

Another property of interest is the following SBSNNI [83] (Strong BSNNI):

$$P \in SBSNNI \qquad \text{iff} \qquad \forall P' \in Der(P) : P' \in BSNNI \tag{3.7.3}$$

Informally P enjoys SBSNNI if any P' in the derivative set of P enjoys BSNNI. Note that if P is finite state, the $Der(P)$ is finite as well, and SBSNNI can be checked by performing a finite number of BSNNI checks. Moreover, SBSNNI enjoys compositionality with respect to the CCS $\|$ and $\setminus$ operators:

Proposition 3.7.11 $P, Q \in SBSNNI$ *implies* $(P \parallel Q) \setminus \mathcal{H} \in SBSNNI$

Finally, the following proposition holds [83]:

Proposition 3.7.12 *The following relations between NDC, BNDC, SNNI, and SBSNNI hold:*

- $SNNI = NDC$
- $SBSNNI \subset BNDC$

SNNI is a sufficient and necessary condition for NDC. We can check $P \in SNNI$ instead of $P \in NDC$, and checking SNNI requires a test only involving local information in P.

SBSNNI is a sufficient condition for BNDC. We can check $P \in SBSNNI$ to understand if $P \in BNDC$. SBSNNI is easily verifiable if P is finite state: it requires to check BSNNI – check that requires only local information – over the finite set of derivatives of P. Moreover, the SBSNNI is compositional: we can reduce the combinatorial explosion due to the parallel composition by checking it directly on P subsystems.

3.8 GNDC Characterization and Analysis of Fault-Tolerant

This section introduces GNDC basic ideas and its application in fault tolerance analysis. From Section 3.2 we know that a GNDC property has the following general form:

$$P \in GNDC^{\alpha}_{\triangleleft} \qquad \text{iff} \qquad \forall X \in \mathcal{E}_{\mathcal{H}} : (P \parallel X) \setminus \mathcal{H} \triangleleft \alpha(P)$$

This scheme is general enough to capture a wide class of security property definitions. For example, more specific security properties such as the BNDC and the NDC, can be subsumed as GNDC properties [85]. We instantiate fault-tolerance in GNDC, in three steps.

The first step requires to specify what P and $\mathcal{E}_{\mathcal{H}}$ are in this context: the former, is the process $P^{\#}_{\mathcal{F}}$ obtained by following the uniform modeling framework described in Section 3.3; the latter, that is $\mathcal{E}_{\mathcal{H}}$, is the set of all faulty environment, $\mathcal{E}_{\mathcal{F}}$. We recall from Definition 3.3.2 that $\mathcal{E}_{\mathcal{F}} \stackrel{\text{def}}{=} \{X \mid Sort(X) \subseteq \mathcal{F} \cup \{\tau\}\}$. Then the general GNDC scheme we propose for fault-tolerance, is:

Definition 3.8.1 (GNDC Characterization of Fault-Tolerance)

$$P^{\#}_{\mathcal{F}} \in GNDC^{\alpha}_{\triangleleft} \qquad \textit{iff} \qquad \forall F \in \mathcal{E}_{\mathcal{F}} : (P^{\#}_{\mathcal{F}} \parallel F) \setminus \mathcal{F} \triangleleft \alpha(P^{\#}_{\mathcal{F}}) \tag{3.8.1}$$

Remark 3.8.2 The separation between the system model and the environment we made in Section 3.4.1, allows us to leave F unspecified and to let it range over $\mathcal{E}_{\mathcal{F}}$. ■

The second step requires $\alpha(P^{\#}_{\mathcal{F}})$ to be some basic property of fault-tolerance. In Section 3.8.1 we will see how to express properties like fail safe, fail silent, fail stop, and fault-tolerance over our test cases models.

The third and last step for the instantiation of GNDC in fault tolerance, concerns understanding what families of equivalences $\triangleleft$ are suitable for the analysis of such properties. This will be discussed in Section 3.6.2, where we write P instead of $P^{\#}_{\mathcal{F}}$ when no explicit reference to the specification framework of Section 3.3 is required.

3.8.1 Fault-Tolerance Properties as Instances of GNDC

In this section we show how to express fail stop, fail silent, fail safe and fault-tolerance in the GNDC scheme. Generally speaking these properties are modeled via a modified version $\alpha(P^{\#}_{\mathcal{F}})$, of $P^{\#}_{\mathcal{F}}$, representing the expected behavior with respect to the property under examination. In the following we treat each property separately, and we express them over our running example. The definitions of fail stop, fail silent, fail safe and fault-tolerance are taken from [19].

Fail Stop. A model of a system $P^{\#}_{\mathcal{F}}$ is expected to be *fail stop* if, in case of faults, it switches into a stop state.

In this case the model exhibiting a fail stop behavior, *e.g.*, the process $\alpha_{stop}(P^{\#}_{\mathcal{F}})$, is built using the following ideas. Fault actions in $P^{\#}_{\mathcal{F}}$ are abstracted away and replaced by silent actions; then, expected behavior of the system is either able to manage the fault without showing faulty behavior, or to stop.

Example 3.8.3 Let us consider the fault-tolerant model $Battery^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}}$ introduced in Example 3.4.1. A fail stop behavior model is, for example, the process $\alpha_{stop}(Battery^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}})$, written α_{stop} for short:

$$\alpha_{stop} \stackrel{\text{def}}{=} \mathtt{get}.(\tau.\overline{\mathtt{ret}}.\alpha_{stop} + \tau.0) \tag{3.8.2}$$

In (3.8.2) we model the idea that after having received a request of energy (action $\mathtt{get}$) in case of any fault (here hidden and represented by the silent action τ) our fault-tolerant battery is either able to satisfy the request and produce a valid unit of energy (action $\overline{\mathtt{ret}}$), or it stops by behaving as the stuck process 0.

Let us assume the battery is in an environment that always injects all the possible faults, that is $\mathcal{F}^{*}_{\{\mathtt{f}_0,\mathtt{f}_1\}} \stackrel{\text{def}}{=} \overline{\mathtt{f}}_0.\mathcal{F}^{*}_{\{\mathtt{f}_0,\mathtt{f}_1\}} + \overline{\mathtt{f}}_1.\mathcal{F}^{*}_{\{\mathtt{f}_0,\mathtt{f}_1\}}$. By using the tool CWB [55], for example, we can prove that

$$(Battery^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}} \parallel \mathcal{F}^{*}_{\{\mathtt{f}_0,\mathtt{f}_1\}}) \setminus \{\mathtt{f}_0,\mathtt{f}_1\} \not\approx \alpha_{stop} \tag{3.8.3}$$

From (3.8.3) we can conclude that $Battery^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}} \notin GNDC^{\alpha_{stop}}_{\approx}$. ∎

Fail Safe. A model of a system $P^{\#}_{\mathcal{F}}$ is expected to be *fail safe* if in case of faults it switches to a safe state.

In this case the model depicting fail safe behavior *e.g.*, the process $\alpha_{safe}(P^{\#}_{\mathcal{F}})$ can be built starting from $P^{\#}_{\mathcal{F}}$ following a procedure similar to the previous case. Faults are abstracted away and represented by silent actions. Next the system is either able to manage the faults without showing faulty behavior, or it shows a behavior that is considered safe, *e.g.*, any fault is detected.

The criteria describing a safe behavior are not clearly definable at this level of abstraction; consequently the formulation of this class of properties is too general in GNDC. In our opinion this is due to the fact that the word *fail safe* says nothing about what behavior is considered safe for the system. In fact, this is the feedback we expect to get from using GNDC: formulating properties in GNDC helps us to understand the degree of formality in the definition of a property itself.

Example 3.8.4 Let us consider the CCS model $Energiz^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}}$ introduced in Section 3.6.4. A possible fail safe behavior model, $\alpha_{safe}(Energiz^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}})$ (in short, α_{safe}), is:

$$\alpha_{safe} \stackrel{\text{def}}{=} \mathtt{get}.(\tau.\overline{\mathtt{ret}}.\alpha_{safe} + \tau.\overline{\mathtt{fail}}.) \tag{3.8.4}$$

In (3.8.4) we model as safe behavior the fact that the detector flags that no battery has produced valid energy with the external action $\overline{\mathtt{fail}}$. So after having received a request of energy (action $\mathtt{get}$) and after any fault occurrence, our model is able either to manage the fault and produce a valid unit of energy (action $\overline{\mathtt{ret}}$) or it signals that a failure happened (action $\overline{\mathtt{fail}}$). Again using the CWB we can verify that:

$$(Energiz^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}} \parallel \mathcal{F}^{*}_{\{\mathtt{f}_0,\mathtt{f}_1\}}) \setminus \{\mathtt{f}_0,\mathtt{f}_1\} \approx \alpha_{safe} \tag{3.8.5}$$

Here $\mathcal{F}^{*}_{\{\mathtt{f}_0,\mathtt{f}_1\}}$ is the faulty environment representing our fault assumptions. From (3.8.5) we conclude that our fault-tolerant model satisfies the fail safe property under the assumption that faults happen as expressed in $\mathcal{F}^{*}_{\{\mathtt{f}_0,\mathtt{f}_1\}}$.

In Section 3.8.2 we will discuss when, for suitable process relation $\lhd$, $(Energiz^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}} \parallel \mathcal{F}^{*}_{\{\mathtt{f}_0,\mathtt{f}_1\}}) \setminus \{\mathtt{f}_0,\mathtt{f}_1\} \lhd \alpha_{safe}$ is a sufficient condition for concluding that $Bat^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}} \in GNDC^{\alpha_{safe}}_{\lhd}$. ■

Fail Silent. A model of a system $P^{\#}_{\mathcal{F}}$ is expected to be fail silent if a fault is ignored.

In this case $\alpha(P^{\#}_{\mathcal{F}})$ can be built starting from $P^{\#}_{\mathcal{F}}$ following the idea that it is able to manage its faults without showing failure. Again occurrences of faults are abstracted away and represented by silent actions.

Example 3.8.5 Let us consider the model $Battery^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}}$ introduced in Example 3.4.1. A model of fail silent behavior, $\alpha_{silent}(Battery^{\#}_{\{\mathtt{f}_0,\mathtt{f}_1\}})$ (in short, α_{silent}), is:

$$\alpha_{silent} \stackrel{\text{def}}{=} \mathtt{get}.(\tau.\overline{\mathtt{ret}}.\alpha_{silent} + \tau.\alpha_{silent}) \tag{3.8.6}$$

In (3.8.6) fail silent behavior is intended as the ability of the system of neither stopping nor showing unpredictable behavior. In case of fault, it becomes ready again to receive a new request for energy. So after having received a request for energy (action $\mathtt{get}$), our model is able either to

manage the fault and produce a valid unit of energy (action $\overline{\texttt{ret}}$) or it is ready to receive a new energy request. Again using the CWB we can verify that:

$$(Battery^{\#}_{\{\texttt{f}_0,\texttt{f}_1\}} \parallel \mathcal{F}^{*}_{\{\texttt{f}_0,\texttt{f}_1\}}) \setminus \{\texttt{f}_0,\texttt{f}_1\} \quad \approx \quad \alpha_{silent} \tag{3.8.7}$$

Formula (3.8.7) implies that our fault-tolerant model satisfies the fail silent property under the assumption that faults happen as expressed in $\mathcal{F}^{*}_{\{\texttt{f}_0,\texttt{f}_1\}}$. For some other class of equivalences $\triangleleft$, this also implies that $Battery^{\#}_{\{\texttt{f}_0,\texttt{f}_1\}} \in GNDC^{\alpha_{silent}}_{\triangleleft}$, as explained in Section 3.8.2. ■

Fault-Tolerance. A model of a system $P^{\#}_{\mathcal{F}}$ is expected to be fault-tolerant if its behavior is observationally equal to the behavior of a module that does not fail at all. In this case then $\alpha_{ft}(P^{\#}_{\mathcal{F}}) = P^{\#}_{\mathcal{F}} \setminus \mathcal{F}$, that is fault-tolerant systems that can never execute any fault action.

Example 3.8.6 In this last example we consider two different versions of fault tolerant models: the first $Bat^{\#}_{\{\texttt{f}_0,\texttt{f}_1\}}$ is the fault tolerant candidate model introduced in Example 3.4.1. The second model is the modified version $Ene^{\#}_{\{\texttt{f}_0,\texttt{f}_1\}}$ introduced in Example 3.8.4. Fault tolerant behavior for $Bat^{\#}_{\{\texttt{f}_0,\texttt{f}_1\}}$ and $Ene^{\#}_{\{\texttt{f}_0,\texttt{f}_1\}}$ is formally defined by the following CCS processes:

$$\alpha_{ft} \stackrel{\text{def}}{=} \alpha_{ft}(Bat^{\#}_{\{\texttt{f}_0,\texttt{f}_1\}}) \quad = \quad Bat^{\#}_{\{\texttt{f}_0,\texttt{f}_1\}} \setminus \{\texttt{f}_0,\texttt{f}_1\} \tag{3.8.8}$$

$$\alpha'_{ft} \stackrel{\text{def}}{=} \alpha_{ft}(Ene^{\#}_{\{\texttt{f}_0,\texttt{f}_1\}}) \quad = \quad Ene^{\#}_{\{\texttt{f}_0,\texttt{f}_1\}} \setminus \{\texttt{f}_0,\texttt{f}_1\} \tag{3.8.9}$$

In both (3.8.8) and (3.8.9) the expected fault-tolerant behavior is the same behavior as resp. models $Bat^{\#}_{\{\texttt{f}_0,\texttt{f}_1\}}$ and $Ene^{\#}_{\{\texttt{f}_0,\texttt{f}_1\}}$ where the fault actions are indeed not allowed to happen. By using the CWB we verify that:

$$(Bat^{\#}_{\{\texttt{f}_0,\texttt{f}_1\}} \parallel \mathcal{F}^{*}_{\{\texttt{f}_0,\texttt{f}_1\}}) \setminus \{\texttt{f}_0,\texttt{f}_1\} \quad \approx \quad \alpha_{ft} \tag{3.8.10}$$

$$(Ene^{\#}_{\{\texttt{f}_0,\texttt{f}_1\}} \parallel \mathcal{F}^{*}_{\{\texttt{f}_0,\texttt{f}_1\}}) \setminus \{\texttt{f}_0,\texttt{f}_1\} \quad \approx \quad \alpha'_{ft} \tag{3.8.11}$$

■

Remark 3.8.7 Observe that the GNDC instance where $\triangleleft$ is $\approx$, and where $\alpha(P^{\#}_{\mathcal{F}})$ is $P^{\#}_{\mathcal{F}} \setminus \mathcal{F}$ (*i.e.*, what it will be using our "fault-tolerance" property called α_{ft} in Section 3.8.1), is BNDC re-stated in our framework. ■

In fact

$$P^{\#}_{\mathcal{F}} \in GNDC^{\alpha_{ft}(P^{\#}_{\mathcal{F}})}_{\approx} \qquad \text{iff} \qquad \forall F \in \mathcal{E}_{\mathcal{F}} : (P^{\#}_{\mathcal{F}} \parallel F) \setminus \mathcal{F} \approx P^{\#}_{\mathcal{F}} \setminus \mathcal{F} \tag{3.8.12}$$

As a final observation we note that BNDC is not compositional with respect to parallel composition (see [146]), that is from $P, P' \in BNDC$ it cannot be deduced that $P \parallel P' \in BNDC$.

Anyway there are bisimulation based-equivalences that are compositional and imply $BNDC$, so that they can be used to prove a sufficient condition for fault-tolerance, and the formulation of fault-tolerance given in (3.8.12) results are attractive from this point of view. An example of these properties is SBSNNI introduced in Section 3.7.1.

3.8.2 Other Observational Relations in GNDC for Fault-Tolerance

In the previous section we have used weak bisimulation when formalizing the instances of fault-tolerance properties in the GNDC; weak bisimulation is useful to detect most of the properties defined so far. However in practical situations we expect that many systems will be fault-tolerant under weaker conditions. As long as the system response is "good enough", it may not be a problem if the existence of faults can be deduced.

For example, the definition of fault-tolerance given in (3.8.12) is too strong and prevents the observer to deduce that *any* faults have occurred.

If we exclude deadlock detection, for all the other safety properties defined in this chapter the ability to distinguish the branching structures is not required. In fact safety properties do not depend on the (branching) path leading to a fault.

This allows us to resort to a weaker form of observational equivalence such as *trace equivalence* and *simulation*. This has also, within GNDC theory, a positive effect on compositionality and on avoiding the universal quantification of fault injectors over the faulty environment. In the following we write $P \parallel_{\mathcal{F}} Q$ as an abbreviation for $(P \parallel Q) \setminus \mathcal{F}$, and we refer to a generic $\alpha(\_)$ function. Obviously the results will also hold for all $\alpha(\_)$'s considered so far.

Most General (Faulty) Environment

The possibility of avoiding universal quantifier in expression (3.8.1) is based on the theory of precongruences whose results we introduced in Section 3.7.8. These results can be restated in terms of the GNDC, also:

Proposition 3.8.8 ([86]) *Let $\lhd$ a precongruence with respect to $\parallel_{\mathcal{F}}$. If there exists a process $Top \in \mathcal{E}_{\mathcal{F}}$ such that for every process $X \in \mathcal{E}_{\mathcal{F}}$ we have $X \lhd Top$, then:*

$$P \in GNDC^{\alpha}_{\lhd} \qquad \textit{iff} \qquad (P \parallel_{\mathcal{F}} Top) \lhd \alpha(P)$$

In particular, if the hypothesis of the proposition above holds then it is sufficient to check that $\alpha(P)$ is satisfied when P is composed with the *most general environment*, Top. In our fault-tolerance analysis context it would permit to make only one single check, in order to prove that a fault-tolerance property holds in every fault scenario. We have also the following corollary for the congruence induced by $\lhd$:

Corollary 3.8.9 ([86]) *Let $\lhd$ to be a precongruence with respect to $\parallel_{\mathcal{F}}$ and let $\bowtie$ be defined as $\lhd \cap \lhd^{-1}$. If there exist two processes $Bot, Top \in \mathcal{E}_{\mathcal{F}}$ such that for every process $X \in \mathcal{E}_{\mathcal{F}}$ we have $Bot \lhd X \lhd Top$ then*

$$P \in GNDC^{\alpha}_{\bowtie} \qquad \textit{iff} \qquad (P \parallel_{\mathcal{F}} Bot) \bowtie (P \parallel_{\mathcal{F}} Top) \bowtie \alpha(P)$$

We show that whenever we are interested in properties based on the notion of trace equivalence, Proposition 3.7.8 and Corollary 3.8.9 hold.

In [86, 85] is reported the following proposition stating that $\leq_{trace}$ is a precongruence with respect to CCS operators.

Proposition 3.8.10 *$\leq_{trace}$ is a precongruence with respect to $\parallel_{\mathcal{F}}$*

In addition we can prove the existence of the most general (failing) environment, and provide its description. Let us consider the following process:

$$Top^{\mathcal{F}} = \sum_{f \in \mathcal{F}} \mathtt{f}.Top^{\mathcal{F}} + \overline{\mathtt{f}}.Top^{\mathcal{F}} \tag{3.8.13}$$

It is straightforward to demonstrate that:

Proposition 3.8.11 *If $X \in \mathcal{E}_{\mathcal{F}}$ then $X \leq_{trace} Top^{\mathcal{F}}$.*

Proof. We prove that $\mathcal{R} \stackrel{\text{def}}{=} \{(X', Top^{\mathcal{F}}) \mid X \in Der(X') \cap \mathcal{E}_{\mathcal{F}}\}$ is a (weak) simulation [158] (see also Definition 3.8.12) containing the pair $(X, Top^{\mathcal{F}})$. As the simulation preorder is finer that the trace preorder the thesis follows. There are three possible cases:

- $X' \stackrel{\mathtt{f}}{\longrightarrow} X''$, with $X'' \in \mathcal{E}_{\mathcal{F}}$, and $Top^{\mathcal{F}} \stackrel{\mathtt{f}}{\longrightarrow} Top^{\mathcal{F}}$ is derivable; hence, $(X'', Top^{\mathcal{F}}) \in \mathcal{R}$.
- $X' \stackrel{\overline{\mathtt{f}}}{\longrightarrow} X''$, with $X'' \in \mathcal{E}_{\mathcal{F}}$, and $Top^{\mathcal{F}} \stackrel{\overline{\mathtt{f}}}{\longrightarrow} Top^{\mathcal{F}}$ is derivable; hence, $(X'', Top^{\mathcal{F}}) \in \mathcal{R}$.
- $X' \stackrel{\tau}{\longrightarrow} X''$, with $X'' \in \mathcal{E}_{\mathcal{F}}$, and $Top^{\mathcal{F}} \stackrel{\widehat{\tau}}{\Longrightarrow} Top^{\mathcal{F}}$ is derivable; hence, $(X'', Top^{\mathcal{F}}) \in \mathcal{R}$.

■

So we have proved that there exists a most general environment with respect to $\leq_{trace}$. A similar conclusion can be obtained when the following *simulation* relation is considered:

Definition 3.8.12 ([160]) *Let $\mathcal{S}$ a binary relation on $\mathcal{E} \times \mathcal{E}$. Then $\mathcal{S}$ is said to be a* simulation *if for each $(P, Q) \in \mathcal{S}$ and for each $a \in Act_{\tau}$, if $P \stackrel{a}{\longrightarrow} P'$ then there exists Q' such that $Q \stackrel{\widehat{a}}{\Longrightarrow} Q'$ and $(P', Q') \in \mathcal{S}$.*

We write $Q \leq_{sim} P$ if there exists a simulation $\mathcal{S}$ such that $(P, Q) \in \mathcal{S}$. It is easy to prove that $\leq_{sim}$ is a precongruence with respect to CCS operators and that it admits the same most general environment in (3.8.13).

Proposition 3.8.13

(1) $\leq_{sim}$ is a precongruence with respect to $\|_{\mathcal{F}}$

(2) if $X \in \mathcal{E}_{\mathcal{F}}$ then $X \leq_{sim} Top^{\mathcal{F}}$.

Proof of case 1. Let consider the following $\mathcal{R} \stackrel{\text{def}}{=} \{(P \|_{\mathcal{F}} X, Q \|_{\mathcal{F}} X) \mid P, Q, X \in \mathcal{E}, P \leq_{sim} Q\}$. We show that is a simulation relation. The only interesting case is that involving τ within a communication: let assume that $P \|_{\mathcal{F}} X \stackrel{\tau}{\longrightarrow} P' \|_{\mathcal{F}} X'$, because $P \stackrel{a}{\longrightarrow} P'$ and $X \stackrel{\overline{a}}{\longrightarrow} X'$ with $a \in Act$. Because of $P \leq_{sim} Q$, we have that $Q \stackrel{\widehat{a}}{\Longrightarrow} Q'$, and the transition $Q \|_{\mathcal{F}} X \stackrel{\widehat{\tau}}{\Longrightarrow} Q' \|_{\mathcal{F}} X'$ is derivable; moreover $P' \leq_{trace} Q'$ and hence $(P' \|_{\mathcal{F}} X', Q' \|_{\mathcal{F}} X') \in \mathcal{R}$. The other (simpler) cases are listed as follows:

- $P \|_{\mathcal{F}} X \stackrel{a}{\longrightarrow} P' \|_{\mathcal{F}} X$. This happens for $a \notin \mathcal{F} \cup \overline{\mathcal{F}}$, if $P \stackrel{a}{\longrightarrow} P'$ and $X \stackrel{\overline{a}}{\not\longrightarrow}$. Because of $P \leq_{sim} Q$ then $Q \stackrel{\widehat{a}}{\Longrightarrow} Q'$ and so $Q \|_{\mathcal{F}} X \stackrel{\widehat{a}}{\Longrightarrow} Q' \|_{\mathcal{F}} X$ is derivable. Moreover, $P' \leq_{trace} Q'$ and hence $(P' \|_{\mathcal{F}} X, Q' \|_{\mathcal{F}} X) \in \mathcal{R}$.

- $P \parallel_{\mathcal{F}} X \xrightarrow{a} P \parallel_{\mathcal{F}} X'$. This happens for $a \notin \mathcal{F} \cup \overline{\mathcal{F}}$, if $X \xrightarrow{a} X$ and $P \not\xrightarrow{\overline{a}}$. Then $Q \parallel_{\mathcal{F}} X \xrightarrow{a} Q \parallel_{\mathcal{F}} X'$ is derivable, and $(P \parallel_{\mathcal{F}} X', Q \parallel_{\mathcal{F}} X') \in \mathcal{R}$.

■

Proof of case 2. Directly from the proof of Proposition 3.8.11. ■

As a conclusion, when $\leq_{trace}$ and $\leq_{sim}$ are used as process relations, the check that P satisfies GNDC properties can be carried out only against the "most general (faulty) environment".

3.8.3 Compositional Analysis of Fault-Tolerance

This section illustrates that, when $\leq_{trace}$ and $\leq_{sim}$ are used as process preorders in our analysis scheme, compositional proof rules for establishing that a system enjoys GNDC can be applied. Compositionality is a desirable property in verification to infer a global fault-tolerance exploiting local fault-tolerance results. Let us show it with a simple example, obtained with the following processes:

$$\begin{aligned} Torch &\stackrel{\text{def}}{=} \overline{\texttt{get}}(\texttt{ret}.\overline{\texttt{flash}}.\mathbf{0} + \texttt{fail}.\overline{\texttt{no\_flash}}.\mathbf{0}) \\ S &\stackrel{\text{def}}{=} (Torch \parallel Ene_{\{\texttt{f}_0,\texttt{f}_1\}}{}^{\#}) \setminus \{\texttt{get}, \texttt{ret}\} \end{aligned}$$

This example represents the behavior of a flashing torch $Torch$ using the fault-tolerant energizer of Example 3.8.6. The energizer is expected to furnish one unit of energy, even in case of fault. The flashing torch $Torch$ emits a $\overline{\texttt{flash}}$ action whenever it receives exactly one unit of energy, $\overline{\texttt{no\_flash}}$ otherwise. What an observer watching the system S, obtained by composing the torch and the energizer, expects is to see only $\overline{\texttt{flash}}$ actions. (Recall that the system $Ene_{\{\texttt{f}_0,\texttt{f}_1\}}{}^{\#}$ provides only $\overline{\texttt{ret}}$.) This safety property can be formalized as:

$$S \in GNDC_{\leq_{sim}}^{\alpha(S)} \quad \text{iff} \quad \forall F_f \in \mathcal{E}_{\{\texttt{f}_0,\texttt{f}_1\}} : S \parallel_{\{\texttt{f}_0,\texttt{f}_1\}} F_{\{\texttt{f}_0,\texttt{f}_1\}} \leq_{sim} \alpha(S) \stackrel{\text{def}}{=} \overline{\texttt{flash}}.\mathbf{0}$$

Here the $\leq_{sim}$ relation has been used. In this case the expected behavior (given through $\alpha(S)$) is that one unit of energy is furnished (and so one $\overline{\texttt{flash}}$ is observed). It is easy to convince us that the given specification of the system enjoys our safety property. Let us now consider a system S^n obtained by composing (in parallel) n instances of the system S and a similar safety property, on the S^n, that reflects the question "only at most n flashes are observed". In our scheme this is equivalent to prove that:

$$\begin{aligned} S^n \in GNDC_{\leq_{sim}}^{\alpha(S)^n} \quad & \text{iff} \\ & \forall F_{\{\texttt{f}_0,\texttt{f}_1\}} \in \mathcal{E}_{\{\texttt{f}_0,\texttt{f}_1\}} : S^n \parallel_{\{\texttt{f}_0,\texttt{f}_1\}} F_{\{\texttt{f}_0,\texttt{f}_1\}} \leq_{sim} \alpha(S)^n \stackrel{\text{def}}{=} \underbrace{\alpha(S) \parallel \ldots \parallel \alpha(S)}_{n} \end{aligned}$$

Compositionality would made the previous statement true, for any fixed n, without the need of any additional check. In the following we prove that it is really the case when $\leq_{trace}$ or $\leq_{sim}$ are used. The following results hold:

Proposition 3.8.14 *Let P_1 and P_2 be two processes such that $P_i \in GNDC_{\leq_{trace}}^{\alpha(P_i)}$ for $i = 1, 2$. Then*

- $P_1 \parallel P_2 \in GNDC^{\alpha(P_1)\|\alpha(P_2)}_{\leq_{trace}}$
- $P_1 \parallel P_2 \in GNDC^{\alpha(P_1)\|\alpha(P_2)}_{\leq_{sim}}$

Proof. Exploiting the existence of the most general environment Top, and the fact that $\leq_{trace}$ (resp.$\leq_{sim}$) is a precongruence. ■

Remark 3.8.15 We affirm that global fault-tolerance can be deduced from local fault-tolerance in Section 3.8.3. Here, by local fault-tolerance we mean the property enjoyed by the formal specification of sub-systems which are required to be fault-tolerant on their own. By global fault-tolerance we mean the property enjoyed by the specification of a system which is obtained by the composition of such sub-systems *without* the adjoint of any other global modules, such as a voter. Obviously we do not expect compositionality to hold in such cases. ■

3.9 Conclusions

The general contribution of this chapter is that the theory and tools of security and security protocol analysis can be profitably applied in fault-tolerant analysis. We start showing how a fault-tolerant (candidate) system may be formalized using CCS. The formal specification is built following a uniform modeling scheme requiring both the failing behavior (with respect to fault occurrences) and fault-recovering procedures to be specified. Faults are represented by specific actions in the system model, that a fault injector environment is able to activate.

This general framework has two main advantages. Firstly, it makes a logical characterization of fault-tolerance possible: the fault tolerant verification problem, with respect to a given property, is formulated as a module checking problem [126], *i.e.,* as the verification problem of an open system acting in an *unspecified* fault injecting environment. Secondly, it allows the formalization of some fault-tolerance properties within the GNDC framework. The consequence of our logical characterization of fault-tolerance is that, by partial model checking, the fault-tolerant verification problem may be expressed as a validity problem in the μ-calculus. In this way, general validation tools and proof techniques can be exploited. For a more efficient (and tailored) analysis we propose, for example, the use of universal and conjunctive μ-calculus formulas whose validity problem is solvable in time linear to the size of the formula (obtained after the partial model checking step). A consequence of the characterization of fault-tolerance in the GNDC scheme, is that we benefit from various theoretical results and analysis techniques from security analysis, where GNDC has been introduced. Specifically, when either a trace relation or a simulation relation are used, GNDC theory assures that efficient analysis procedures exist: fault-tolerance benefits both of a static characterization of its properties, and of compositionality proofs. Another advantage, is the possibility of comparing different fault-tolerance properties within GNDC, as is already done for security properties [83, 81, 85]. Potentially, this is a preliminary step towards a formal and uniform taxonomy of fault tolerant and security properties. For example, we show that the fault-tolerance property is formalized as the instance of GNDC known as BNDC. This means that fault-tolerance is precisely characterized as a non-interference property [204]. An immediate consequence is that available tools for checking BNDC [146] can be used to check fault tolerance.

Fail safe and fail silent do not enjoy such a precise classification in terms of GNDC properties known in security, although they are expressible in our running examples. From this formulation effort it emerges that fail safe is a category of properties parametric in the notion of safe behavior, and that its informal definition is too general to be unambiguously expressed in GNDC. This

suggests that only a better classification of fail safeness can lead to a more precise formulation. Similarly we managed to characterize completely fail stop in one of our examples, but here also a general characterization is still missing.

4

SPYDER: a Model Checker for Security Protocols

"The system does not communicate with the outside, so it cannot be influenced remotely. The computer system is secure" (Head engineer John Arnold in Jurassic Park, *Crichton, 1991)*

Abstract

This chapter presents a model checking environment for security protocols. A protocol is described as a term of a process algebra consisting of the parallel composition of a finite number of, communicating and finitely behaviored, processes. Each process represents an instance of a protocol role. The intruder is implicitly modeled in the semantics of the calculus as an environment controlling all the communication events.

Security properties are written as formulas of a linear-time temporal logic. The model checker runs a depth first search algorithm that tests the satisfiability of a formula over all the traces, generated on-the fly, from a typed version of protocol.

4.1 Introduction

Past experience has shown how formal methods can be successfully applied to the analysis of security protocols (*e.g.*, see [40, 172, 77, 3, 102]). For example, secrecy, integrity and authenticity properties [5] can be verified over protocol specifications written in the spi-calculus [7], a process algebra derived from the π-calculus [162] enriched with operators to encrypt and decrypt messages, via symbolic trace analysis [71, 12, 29, 79, 30] or type checking over typed versions on the calculus [3, 102].

Here we propose a *logic-based model-checking* [48] approach to the verification of security protocols. Protocols are expressed over a typed version of a spi-calculus dialect, which we call the *spy-calculus*. Properties are written as logic formulas whose satisfiability is checked over temporal models (*i.e.*, labelled transition systems) which constitute the operational semantics of the *spy-calculus*.

Model-checking applied to process algebras close to the spi-calculus was new when the paper [134](on which this chapter is based) appeared, although its use in protocol security analysis was already known (*e.g.*, see [155, 136, 163, 182, 53, 92]).

The originality of the approach will be explained using the illustration of Figure 4.1. Let us suppose to have a finite *spy-calculus* term Z representing a finite number of runs of a protocol involving a finite number of roles, and a security property f to be checked on it. The semantics of Z, a labelled transition systems LTS, is generally infinite-state, because of the infinite number of messages that each role, running the protocol, can receive from a potential intruder. In fact the

intruder, following the Dolev-Yao [69] model, is potentially able to compose messages of infinite length and to deliver them to honest participants in order to subvert the protocol goal.

To reduce the number of messages received by the honest agents to a finite number, our model checker, SPYDER, checks the satisfiability of f over a typed version of the protocol $\mathcal{C}_{\mathbb{S}_Z}(Z)$. The use of types allows us to filter out, during the execution of input actions, those messages from the intruder that do not match the required type. The resulting model of the protocol ($LTS_\flat$ in Figure) is proved to be finite-state in this case. Moreover in SPYDER, types are provided through the definition of a "typing" function $\mathcal{C}_{\mathbb{S}_Z}$ introduced at run-time. In this way typed versions are not fixed a priori and a user can obtain different typed versions of same untyped protocol model, explore different partitions of the whole state space, and consequently increase the confidence that the results of the analysis hold over the infinite model.

An additional element of flexibility comes from the use of a logic as a language to express security properties. The logic used by SPYDER has been shown to be sufficiently expressive to model a large class of properties for example secrecy, authenticity but also some weak form of privacy, anonymity, and non-repudiation (see [51, 52, 53] for details).

The proposed approach has both advantages and disadvantages. We have already noticed that having a logic introduces flexibility in expressing security properties, especially the non-standard ones such as weak forms of privacy or anonymity. In other model checking approaches, such as for example Casper (which uses the model-checker FDR) [139, 70] the formalization of properties with the exception of secrecy and integrity does not seem an easy task. The same can be said about the tools NRL [155] and Murϕ [163].

On the other hand we cannot cope with infinite-state models, that is with an unbounded network. Some model checkers, for example NRL [155], can. Others, can deduce results over unbounded networks by analyzing bounded versions; for example work on data independence analysis and CSP [36] has shown that FDR [2] (so also Casper [139]) can be used to infer security results over protocols managing infinite nounces, keys *etc.* from an analysis performed over a protocol that uses only a finite set of them. Our approach can help in reaching similar but weaker, results: by the definition of different typing transformations (*e.g.,* $\mathcal{C}_{\mathbb{S}_Z}, \ldots, \mathcal{C}'''_{\mathbb{S}_Z}$ in Figure) and by analyzing the related finite-state models the confidence that the same results hold over the infinite model can be increased. In fact, as a theoretical result we prove that an attack *i.e.,* a trace that does not satisfy a formula f over a finite model of a typed version of a protocol ($\not\models f$ checked over $LTS_\flat$ in Figure) always implies the presence of the same attack over the model of the corresponding untyped version of the protocol ($\not\models f$ checked over LTS in Figure). This means that our framework is sound. Generally the existence of an attack over the untyped version of a protocol does not imply the same attack over a *specific* typed version, but we prove that a transformation (*e.g.,* $\mathcal{C}'_{\mathbb{S}_Z}$ in figure) always exists (even though we will not be able to build it without knowing the attack) such that the resulting typed version ($\mathcal{C}'_{\mathbb{S}_Z}(Z)$ in Figure) shows the same flaw. This implies that our framework is also complete.

The rest of the chapter is organized as follows. Section 4.2 and Section 4.3 present the syntax and semantics of our calculus and the logic used to specify properties. Section 4.4 introduces its typed version which admits a finite-state labelled transition semantics. Section 4.4.1 and Section 4.5 define the class of functions that are used to obtain typed protocols starting from a generic specification. Moreover in those sections the main results of this chapter are proved. Section 4.5.1 formally describes the model checker algorithm whose correctness is based on the theory previously developed. Section 4.6 concludes. A running example is used throughout the chapter to illustrate the main ideas of the approach.

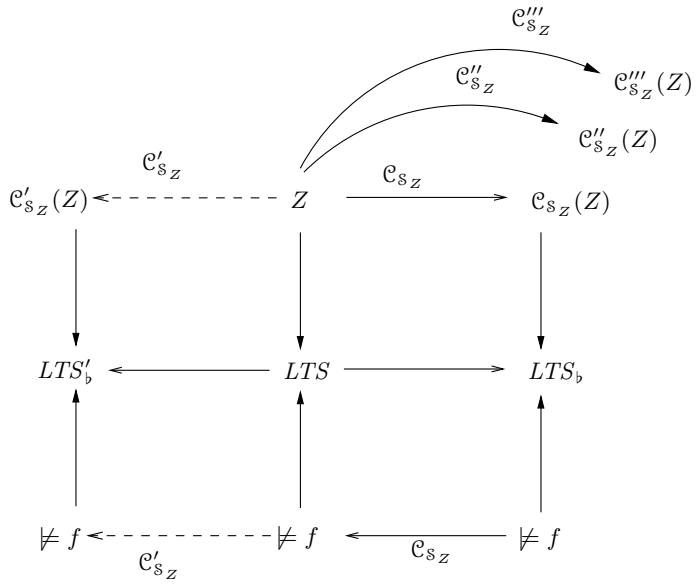


Figure 4.1: The SPYDER environment. A protocol is described as a *spy-calculus* term Z, while a property is specified as a logic formula f. Usually the semantic model of Z is an infinite state labelled transition system LTS. A finite-state model $LTS_\flat$ can be obtained from a typed specification $\mathcal{C}_{\mathcal{S}_Z}(Z)$ of the protocol, obtained from Z and via a typing functions $\mathcal{C}_{\mathcal{S}_Z}$. Different typing functions $\mathcal{C}''_{\mathcal{S}_Z}, \mathcal{C}'''_{\mathcal{S}_Z}$ can be introduced at run-time to obtain different finite models, each representing a particular partition of the whole infinite state space. If $LTS_\flat$ is not a model for f (*e.g.*, if $LTS_\flat \not\models f$), neither is LTS. If LTS is not a model for f, there exists a $\mathcal{C}'_{\mathcal{S}_Z}$ such that the corresponding $LTS'_\flat$ is not a model for f either.

4.2 The *spy-calculus*

This section introduces the *spy-calculus*. It is a process algebra whose syntax is inspired by the spi-calculus of Abadi and Gordon in [7].

4.2.1 Syntax

In the language we assume an infinite set of constants $\mathcal{N}$ (names), an infinite set of variables $\mathcal{V}$ and two binary functions, $\{\_\}\_$ standing for encryption and $\langle \_, \_ \rangle$ standing for pairing, respectively. Moreover, we assume a finite set of $\mathcal{A}$ of labels a, and a finite set $\mathcal{I}$ of integer identifiers i.

The set $\mathcal{M}$ of *messages* is defined as the collection containing at least $\mathcal{N}$ and such that $M, M' \in \mathcal{M}$ implies both $\langle M, M' \rangle \in \mathcal{M}$ and $\{M\}_{M'} \in \mathcal{M}$. Similarly the set $\mathcal{T}$ of *terms* is the collection that contains at least $\mathcal{N} \cup \mathcal{V}$ and such that $T, T' \in \mathcal{T}$ implies both $\langle T, T' \rangle \in \mathcal{T}$ and $\{T\}_{T'} \in \mathcal{T}$. Formally the *spy-calculus* syntax is defined by the following grammar:

$$
\begin{array}{lrcl}
\textit{protocols} & Z & ::= & Z \| Z' \mid (\backslash N)Z \mid (i, P) \\
\textit{roles} & P & ::= & 0 \mid a(x).P \mid \overline{a}(T).P \mid \underline{a}(T).P \mid P + P' \mid (\nu N)P \mid [x \,\mathtt{is}\, T]P
\end{array}
$$

A *protocol* Z is the parallel composition of role instances (i, P). $(\backslash N)$ restricts names in N. A restricted name is initially private *i.e.,* unknown to the intruder. In $(\backslash N)Z$, N is bound in Z.

Each *role instance*, or agent, is composed of an identifier i and by a process (role) P. In turn, a *role* P is either:

1. 0, the process that does nothing;
2. $a(x).P$, the *input* process ready to receive a message which will be bound to the variable x. A label a is used to distinguish among different input actions;
3. $\overline{a}(T).P$, the *output* process ready to send a term T, classified as action a;
4. $\underline{a}(T).P$, the *assert* process ready to perform an *assertion* of the term T. Differently from outputs and inputs, assertions are not communication actions. Assertions are used for verification purposes only and they act as control flags in the execution of the protocols. Assertions were presented first by Woo and Lam in [206] as begin-events and end-events for specifying protocol authenticity properties;
5. $P + P'$, the non-deterministic choice between processes P and P';
6. $(\nu N)P$, the process that generates a new local name N, then used within P;
7. $[x \,\mathtt{is}\, T]P$, the *match* process that requires the term T to unify with the contents of a variable x (possibly binding free variables in T) in order to proceed as P.

In process $(\nu N)P$ the name N is bound in P. In input process $a(x).P$, the variable x is bound in P, while in $[x \,\mathtt{is}\, T]P$ all the free variables in T are bound in P. We say that a variable, or a name, is free if it is not bound.

We assume each role P is always introduced by a defining equation $p\langle \boldsymbol{x} \rangle \stackrel{def}{=} P$ where $\boldsymbol{x}$ is the tuple of all the free variables x in P. In that case $p\langle \boldsymbol{N} \rangle$ where $\boldsymbol{N}$ is a tuple of names, is the same as $P[\boldsymbol{N}/\boldsymbol{x}]$ *i.e.,* the closed process where each free variable x in $\boldsymbol{x}$ is replaced by the corresponding name N in $\boldsymbol{N}$.

Before describing an example specification of a protocol in the *spy-calculus*, let us discuss some of its characteristics. A protocol consists of a number of *finite* and *sequential* processes. The use of finite processes avoids problems due to recursive processes, but it suffices to describe the behavior of a role running a security protocol (*e.g.*, see [43]). In fact, the specification of a role is usually a finite sequence of actions.

Some security protocols show their vulnerability only when running over repeated sessions. So we need to describe more runs of a protocol specification. In the *spy-calculus*, as usual in security protocol analysis (*e.g.*, see [182, 71, 29]), it is possible to model a principal running a finite number of sessions of the protocol by specifying, for example, k instances, $(i_1, P), \ldots, (i_k, P)$, of the same role P. We cannot model an unbounded replication of processes, usually written as $!P$, because this would lead to infinite runs of a protocol.

In any case limiting the number of runs to be finite is a widely accepted strategy for the model checking approach to security (*e.g.*, see [56]), where a finite-state analysis is required. This is the case in most of the famous model checkers Murϕ [163], Brutus [53] or FDR [136], even if some of them limit the number of steps not at syntax level but only during the analysis. By combining a limited number of runs and the fact that agents are finite-behaviored, we implicitly are assuming only a finite number of nounces, keys, process names *etc.*. to be involved in a protocol. Again, this constraint is required in order to have finite-state analysis, although the analysis remains NP-Complete [114] even under these strict conditions. In [178, 36] Roscoe *et al* show how, by the application of data independence techniques, it is possible to reduce the problem of proving the security of a model where an infinite supply of different nounces, keys *etc.* is required, to a finite check where only a finite number of them are indeed involved. These techniques, ad-hoc proved within the CSP theory, would merit more attention but restating them within our framework is beyond the scope of this chapter.

To conclude this discussion, we observe that even the use of finite runs and a finite number of nounces, keys, etc. is not sufficient to obtain finite-state models, for an intruder may still generate data of an infinite length. We will postpone the discussion about how to cope with this last kind of infiniteness, till Section 4.5.

Example 4.2.1 As an example of a specification in *spy-calculus*, let us consider the following key-exchange protocol, a simplified (in the sense that it focuses on the agreement of a new session key) version of Needham-Schroeder Shared Key protocol [167] (NSSK). The protocol requires three roles: two principals, A (the initiator) and B (the responder), and a trusted server S. Following the common informal notation, NSSK is described as follows:

1. $A \rightarrow S : A, B$
2. $S \rightarrow A : \{K_{AB}\}_{K_{AS}}, \{A, K_{AB}\}_{K_{BS}}$
3. $A \rightarrow B : \{A, K_{AB}\}_{K_{BS}}$

Informally, in the protocol, A initiates the communication, by sending a message $\langle A, B\rangle$ to the trusted server S (step 1). With this message A asks S for a session key that A can use to secure communication with B. As a reply, S prepares a message composed of two parts. Both parts contain the new session key K_{AB}, created by S (step 2): the first part, $\{K_{AB}\}_{K_{AS}}$, is encrypted with the key K_{AS} that S (already) shares with A (this means that only A should be able to decrypt the message). The second part, $\{A, K_{AB}\}_{K_{BS}}$, is reserved for B, and it is encrypted with the key

K_{BS} that S shares with B. Instead of sending it directly to B, S relies on A to deliver the message to B in step 3. One session of the NSSK protocol is encoded in the *spy-calculus* as follows:

$$\begin{aligned} & (\backslash K_{as})(\backslash K_{bs})(\backslash A)(\backslash B)(\backslash S) \\ NSSK \stackrel{def}{=} \; & (1, p_A\langle A, B, S, K_{as}\rangle) \| (2, p_B\langle A, B, S, K_{bs}\rangle) \\ & \| (3, p_S\langle A, B, K_{as}, K_{bs}\rangle) \end{aligned}$$

where

$$\begin{aligned} p_A\langle a, b, s, k_{as}\rangle &\stackrel{def}{=} \overline{c_{as}}(\langle a,\, b\rangle).c_{as}(x).[x \texttt{ is } \langle\{x_1\}_{k_{as}}, x_2\rangle].\overline{c_{ab}}(x_2).0 \\ p_B\langle a, b, s, k_{bs}\rangle &\stackrel{def}{=} c_{ab}(y).[y \texttt{ is } \{\langle a,\, y_1\rangle\}_{k_{bs}}].0 \\ p_S\langle a, b, k_{as}, k_{bs}\rangle &\stackrel{def}{=} (\nu K_{ab})c_{as}(z).[z \texttt{ is } \langle a,\, b\rangle].\overline{c_{as}}(\langle\{K_{ab}\}_{k_{as}}, \{\langle a, K_{ab}\rangle\}_{k_{bs}}\rangle).0 \end{aligned}$$

The *spy-calculus* specification of the NSSK protocol shows three agent instances, one for each role in the protocol. Instance $(1, p_A\langle A, B, S, K_{as}\rangle)$ models one session of the process p_A. This is the initiator A, while $(2, p_B\langle A, B, S, K_{bs}\rangle)$ and $(3, p_S\langle A, B, K_{as}, K_{bs}\rangle)$ model respectively the responder B and the trusted server S.

Process A first sends the message $\langle a,\, b\rangle$ to S, then A receives a reply in variable x. Then x is analyzed: if x is indeed a pair whose first part is encrypted with the shared key K_{as}, then the second part of x is stored in variable x_2. Then A sends x_2 to B. Otherwise the system gets stuck.

Three different labels c_{ab}, c_{as} and c_{bs} are used to distinguish input/output actions. In this example keys K_{as} and K_{bs} and the names of participants A, B and S are restricted, that is hidden from the initial knowledge of the intruder. ■

4.2.2 Semantics

Differently from the usual approach taken in process algebras for security (*e.g.,* see [5, 181]), in our calculus, the intruder is not explicitly described as an additional process. Instead, the strongest possible intruder, *i.e.,* the Dolev-Yao [69] *intruder* Ω, will appear in the semantics as an *environment* having complete control of any communications. A similar approach has been taken in [31] where an environment-sensitive semantics is defined.

Assuming such an embedded intruder implies that, instead of symmetric and synchronous communication via shared channels, our calculus uses asynchronous communications via a *unique* and anonymous channel, the intruder/environment (intruder from now on). Moreover, having a unique public channel implies that whenever a process performs an output, the message will be given to the intruder, while during an input the message will be retrieved from those in the intruder's knowledge (see later for a formal definition) *i.e.,* from those that can be composed by using the messages already known to the intruder.

An important question to ask at this point is: can the fact of modeling the intruder as an embedded component cause a loss of power or flexibility? For example in this way we cannot add new intruder capabilities. We can observe that, firstly, a precise set of capabilities (*e.g.,* those used here, see Figure 4.2) has been identified in the literature and proved to be sufficient to catch the security flaws we will investigate here (*e.g.,* see [85]); secondly, the semantics of the input only depends on intruder capabilities, and it is not difficult to extend our calculus in order to be

parametric with respect to this set. The only constraint we have is that the question "M belongs to the intruder knowledge", where M is a message, must be decidable (see also later).

In the following we will give all the definitions required by the *spy-calculus* formal semantics. We start with the definition of *knowledge*:

Definition 4.2.2 (Knowledge) *Let $W \subseteq \mathcal{M}$ be a finite set of messages. The* knowledge *of W, written $KS(W)$, is the set $W \cup \{M : W \vdash^* M\}$, where $\vdash$ is the derivation symbol defining the intruder capabilities in managing messages as given in Figure 4.2.*

Here we require that the proof system is such that the question $M \in KS(W)$ is decidable and derivable. This is indeed the case for the proof system of Table 4.2 (see [79] for the proof of decidability). This assumption is needed if we want our model checking algorithm to be effective.

The operational semantics of the *spy-calculus* is given in terms of a labelled transition system (LTS, in short). States of the LTS are pairs $(\mathcal{G} : Z)$, of *global states* and *protocols*. A global state consists of the *local state* of each agent plus the state of the intruder. In turn, a local state contains the name p of the role played by the agent, the set W of messages it has received so far, and a mapping σ from variables to values. Formally:

Definition 4.2.3 (Global State) *Let there be n process instances involved in a protocol. A* global state $\mathcal{G}$ *is a sequence of $n+1$ local states, where:*

- $\mathcal{G}(0)$ *is the local state of the intruder Ω;*
- $\mathcal{G}(i)$ *is the local state of process instance identified by i, for $i = 1, \ldots, n$.*

We write $Glob$ to indicate the set of all global states.

Definition 4.2.4 (Local State) *A* local state l, *is a triple (p, W, σ) where $p \in \mathcal{N}$ is a name, $W \subseteq \mathcal{M}$ is a set of messages and $\sigma : \mathcal{V} \longrightarrow \mathcal{M}$ is a function from variables to messages.*

Before going on, we give some technical details about the substitution σ, used in the definition of a local state. With the symbol $\perp$ we indicate the function σ undefined everywhere. We say that σ' is approximated by σ, written $\sigma' \sqsupseteq \sigma$ whenever the function σ' coincides with σ in every value of the domain where σ is defined. Moreover we write $\sigma(x) = \sigma'(x)$ if both functions are undefined at x or coincide in x. $\widehat{\sigma}$ is the extension of σ to message terms. The test $\widehat{\sigma}(T) = \widehat{\sigma}(S)$ evaluates to true if and only if both functions return the same ground message, false otherwise.

Giving a global state $\mathcal{G}$, we indicate by p_i, W_i, and σ_i respectively, the items of the local state $\mathcal{G}(i)$, for $i = 0, \ldots, n$. When convenient W_Ω will be used instead of W_0. It is worth to underline that in the operational semantics each W_i and σ_i has a monotone growth rate along the protocol execution. This implies that a variable is bound by the first (leftmost) input prefix or matching operator in which it appears.

The transition rules of the LTS describe how a state $(\mathcal{G} : Z)$ evolves as a consequence of an *action* α. An action is identified by the agent identifier, i (the agent that is performing the action), by the action label a, and (if the action is a visible action) by the messages M involved in the action. Formally:

Definition 4.2.5 (Action) *An* action α *is either: (a) $i.a\langle M\rangle$ or $i.\overline{a}\langle M\rangle$ or $i.\underline{a}\langle M\rangle$ meaning that instance i has executed, respectively an input, output or assertion, labelled a over the message M; or (b) τ, the internal action.*

Expanding rules

$$\frac{m \in W}{W \vdash m}\,\mathbf{E}_{\in} \qquad \frac{W \vdash m \;\; W \vdash k}{W \vdash \{m\}_k}\,\mathbf{E}_{\{\}} \qquad \frac{W \vdash m \;\; W \vdash m'}{W \vdash \langle m, m' \rangle}\,\mathbf{E}_{<>}$$

Shrinking rules

$$\frac{W \vdash \{m\}_k \;\; W \vdash k}{W \vdash m}\,\mathbf{S}_{\{\}} \qquad \frac{W \vdash \langle m, m' \rangle}{W \vdash m}\,\mathbf{S}_{<>} \qquad \frac{W \vdash \langle m, m' \rangle}{W \vdash m'}\,\mathbf{S}'_{<>}$$

Figure 4.2: Inference rules defining the derivation symbol $\vdash$, for the intruder knowledge. W is a set of initial messages. Among the expanding rules the first says that whatever is in W is derivable. The second, defines the rule for symmetric encryption and the third, defines the rule for pair composition. The first shrinking rule defines decryption, whereas the last two define pair projection.

We write Act_τ to indicate the set of all actions. The following definition gives the operational semantics:

Definition 4.2.6 (Operational Semantics)
Let $Z = (\backslash N_1) \ldots (\backslash N_k)(1, P_1) \| \ldots \| (n, P_n)$, be a protocol. The associated labelled transition system is a tuple $(\mathcal{Q}_Z, \langle \mathcal{G}_0, Z \rangle, Act, \mathcal{R}_Z)$, where:

(a) $\mathcal{Q}_Z \subseteq Glob \times \mathbb{Z}$ is the set of states;

(b) $\langle \mathcal{G}_0, Z \rangle$ is the initial state so defined:

- *$\mathcal{G}_0(0) = (\Omega, W_\Omega, \bot)$, is the initial local state of the intruder, where Ω is the name of the intruder and W_Ω is the initial set of messages known to the intruder, composed of all the free names in Z.*
- *$\mathcal{G}_0(i) = (p_i, W_i, \bot)$ for all $i \in \mathcal{I}$, is the initial local state of the role instance i. More precisely, p_i is the name of the process P, if $p_i \stackrel{def}{=} P$, W_i is the set of messages p_i knows, consisting of all free names in P_i.*

(c) Act_τ is the set of actions;

(d) the transition relation $\mathcal{R}_Z \subseteq \mathcal{Q}_Z \times Act \times \mathcal{Q}_Z$ is the least transition relation defined by the rules in Figure 4.3. Whenever $(q, \alpha, q') \in \mathcal{R}_Z$ we will write $q \xrightarrow{\alpha} q'$.

We now explain the rules in Figure 4.3 in an informal way. Rule $\mathbf{r}_{\equiv 1}$ and rule $\mathbf{r}_{+1}$ (and the symmetric $\mathbf{r}_{\equiv 2}$ and $\mathbf{r}_{+2}$) define the usual transitions in case of respectively parallel composition and non deterministic choice. Rule $\mathbf{r}_{=}$ defines the transition in case of a match: we require that the local binding function σ_i can be extended in such a way that x equals T when σ_i is applied. In the premises of the rule $\mathbf{r}_{=}$ the condition $\mathcal{P}(\widehat{\sigma}(T))$ informally means that $\widehat{\sigma}(T)$ must not contain variables as decryption keys. We add this condition only to avoid unfair specifications where encryption could be broken simply by using pattern matching. The use of $\mathcal{P}(\_)$ is indeed a syntactic constraint, whose presence does not interfere with our analysis. Formally $\mathcal{P}(\_)$ is defined as follows:

$$\frac{(\mathcal{G} : Z') \xrightarrow{\alpha} (\mathcal{G}' : Z'')}{(\mathcal{G} : Z \| Z') \xrightarrow{\alpha} (\mathcal{G}' : Z \| Z'')}\ \mathbf{r}_{\equiv 1} \qquad \frac{(\mathcal{G} : Z \| (i, P)) \xrightarrow{\alpha} (\mathcal{G}' : Z \| (i, P'))}{(\mathcal{G} : Z \| (i, P + P'')) \xrightarrow{\alpha} (\mathcal{G}' : Z \| (i, P'))}\ \mathbf{r}_{+1}$$

$$\frac{\sigma_i(x) \neq \bot,\ \mathcal{P}(\widehat{\sigma}_i(T)),\ \exists \sigma' \sqsupseteq \sigma_i : \widehat{\sigma}'(T) = \sigma_i(x)}{(\mathcal{G} : Z \| (i, [x\ \mathtt{is}\ T]P)) \xrightarrow{\tau} (\mathcal{G}[\sigma'/\sigma_i] : Z \| (i, P))}\ \mathbf{r}_{=}$$

$$\frac{\widehat{\sigma}_i(T) = M}{(\mathcal{G} : Z \| (i, \overline{a}(T).P)) \xrightarrow{i.\overline{a}\langle M \rangle} (\mathcal{G}[W_\Omega \cup \{M\}/W_\Omega] : Z \| (i, P))}\ \mathbf{r}_{!}$$

$$\frac{\widehat{\sigma}_i(T) = M}{(\mathcal{G} : Z \| (i, \underline{a}(T).P)) \xrightarrow{i.\underline{a}\langle M \rangle} (\mathcal{G} : Z \| (i, P))}\ \mathbf{r}_{!!}$$

$$\frac{\exists \sigma' \sqsupseteq \sigma_i : \sigma'(x) = M \in KS(W_\Omega)}{(\mathcal{G} : Z \| (i, a(x).P)) \xrightarrow{i.a\langle M \rangle} (\mathcal{G}[W_i \cup \{M\}/W_i][\sigma'/\sigma_i] : Z \| (i, P))}\ \mathbf{r}_{?}$$

$$\frac{N' \notin c(P) \cup c(Z)}{(\mathcal{G} : Z \| (i, (\nu N)P)) \xrightarrow{\tau} (\mathcal{G}[W_i \cup \{N'\}/W_i'] : Z \| (i, P[N'/N]))}\ \mathbf{r}_{\nu}$$

$$\frac{}{(\mathcal{G} : Z \| (\backslash N) Z') \xrightarrow{\tau} (\mathcal{G} : Z \| Z')}\ \mathbf{r}_{\backslash}$$

Figure 4.3: Labelled Transition Systems rules. Rules $\mathbf{r}_{\equiv 2}$ and $\mathbf{r}_{+2}$, which are the symmetric versus of $\mathbf{r}_{\equiv 1}$ and $\mathbf{r}_{+1}$, have been omitted. See the text for an informal explanation.

$$\mathcal{P}(T) = \begin{cases} \texttt{tt} & \text{if } T = M \in \mathcal{M} \\ \texttt{tt} & \text{if } T = x \in \mathcal{V} \\ \mathcal{P}(T') & \text{if } T = \{T'\}_M \\ \mathcal{P}(T_1) \wedge \mathcal{P}(T_2) & \text{if } T = \langle T_1, T_2 \rangle \\ \texttt{ff} & \text{otherwise .} \end{cases}$$

Rule $\mathbf{r}_!$ is for the output transition. We require that the term T evaluates to a message M, and then the transition leads to a state where the message M is added to W_Ω, the intruder local state. Rule $\mathbf{r}_{!!}$ is similar, but because an assertion is not a communication action, no message is put into the local state of the intruder. It only leaves a trace as a label in the transition, whose presence can be tested during the analysis. Rule $\mathbf{r}_?$ defines the input transition. If the local binding function σ_i can be extended in such a way that x equals some message M in the intruder knowledge (*i.e.*, $M \in KS(\Omega)$), that message is retrieved and then added to the local state of the agent instance. Note that even if all the premises are decidable, the number of messages in $KS(W_\Omega)$ is generally infinite. This produces infinite branching *in absence of limiting strategies*. We will describe one such strategy in Section 4.4.1. Rule $\mathbf{r}_\nu$ describes the creation of new names. Here by $c(P)$ we mean the set of constants that appear syntactically in P. Rule $\mathbf{r}_\backslash$ describes the restriction operator. This operator has no effect on the transition itself, but it modifies the definition of the initial knowledge of the intruder. Restricted names are initially hidden from the intruder.

4.3 A Logic for Security Properties

This section shows how a linear time temporal logic can be used to express security properties. Different logics for security can be found in literature (*e.g.*, see [40, 90]), but the logic we propose here is inspired by the linear time temporal logic first introduced in [51]. It has been used to express a large set of properties, from secrecy, authenticity, general correspondence properties, weak forms of anonymity, privacy and non-repudiation (*e.g.*, see [50, 52, 53]). Informally, its terms talk about messages, roles identifiers, role names and local (with respect to a role instance) message terms. Its propositions can express facts about actions that happened, tests about what messages are known to a role (or to the intruder), and equality tests over message terms. Formulas are either propositional formulas or the modal formula "eventually" in its past interpretation. By using this modality it is possible to express properties about temporal correspondences among events in protocol runs.

Formally, the logic shares with the *spy-calculus* the same set of names ($\mathcal{N}$), of variables ($\mathcal{V}$), of integer identifiers ($\mathcal{I}$), and of labels ($\mathcal{A}$). Moreover messages (set $\mathcal{M}$), terms (set $\mathcal{T}$) are defined as in *spy-calculus*. We now explain in detail the syntax of the logic, also presented in Figure 4.4. We let i range over $\mathcal{I}$, T over $\mathcal{T}$, M over $\mathcal{M}$, and λ over $\mathcal{A} \cup \overline{\mathcal{A}} \cup \{\tau\}$.

A message term ς is (a) **name**(i), the name of the agent whose identifier is i, (b) $i.T$ the message term T interpreted in the local state of the agent whose identifier is i or (c) a ground message $M \in \mathcal{M}$.

An atomic proposition ρ is (a) **Knows**(i, ς), a predicate on the presence, in the local state of the agent identified by i of ς; (b) **Acts**$(i.\lambda(\varsigma))$ (respectively **Acts**(τ)) a predicate on the fact that a visible action *i.e.*, a send, receive or assertion, over ς has been performed by the agent identified by i (respectively a silent action has been performed). (c) $\varsigma = \varsigma'$, is an equality test over terms.

A formula f is any propositional logic formula, or the modal formula $\Diamond_P f$, where the symbol $\Diamond_P$ is the modal operator *eventually* in its past interpretation.

The derived operators $\vee$ (propositional or) and $\supset$ (propositional implication) can be derived as usual from $\wedge$ and $\neg$. Moreover as a syntactic sugar we also use the formulas $\exists s.f(s)$, where s is a variable, defined as $\vee_{\{i\in\mathcal{I}\}} f[s/i]$ and $\forall s.f(s)$ defined as $\wedge_{\{i\in\mathcal{I}\}} f[s/i]$.

Example 4.3.1 In this example we show how to express security properties using the logic presented here. Let us start with the following authenticity property over the NSSK protocol formalized in Example 4.2.1:

> "*When role p_B finishes the protocol thinking that the responder is the role p_A, role p_A has at least started the same protocol thinking that the initiator is p_B.*"

The property can be logically expressed with the following formula:

$$f \stackrel{def}{=} \mathbf{Acts}(2.c_{ab}(2.y)) \supset \Diamond_P(\mathbf{Acts}(1.\overline{c_{ab}}(1.x_2))) \wedge (2.y = 1.x_2) \tag{4.3.1}$$

Informally this says that whenever the agent with identification number 2 (the one who is playing the role of p_B) receives a message in y (as a consequence of an input action labelled c_{ab}) then the agent identified by 1 (running the role p_A) has previously sent a message (in x_2) to p_B through an action labelled c_{ab}. Additionally the two messages are required to be equal.

The use of identification numbers in the formula makes it not so readable. A clearer way of writing (4.3.1) is the following:

$$\begin{aligned} f\prime \stackrel{def}{=}\ & \forall b.\, \exists a.\, \mathbf{name}(b) = p_B \ \wedge\ \mathbf{name}(a) = p_A \ \wedge \\ & \mathbf{Acts}(b.c_{ab}(b.y)) \supset \Diamond_P(\mathbf{Acts}(a.\overline{c_{ab}}(a.x_2))) \wedge (b.y = a.x_2) \end{aligned} \tag{4.3.2}$$

Here $\forall$ and $\exists$ are used as syntactic sugar for a finite sequence (over the set of identifiers $\{1, 2, 3\}$) of respectively $\wedge$ and $\vee$. Similar abbreviations are necessary whenever we need to talk about a role independently of the fact that more instances of it are modeled in the *spy-calculus* formulation of a protocol. We will use such style of expressing formulas in the rest of the paper. Formula (4.3.2) informally says that whenever an agent (playing the role of p_B) receives a message in y (as a consequence of an input action labelled c_{ab}) then there exists an agent (running the role p_A) that has previously sent a message (identified by $a.x_2$) to p_B through an action labelled c_{ab}. Additionally (4.3.2) requires the two messages to be equal.

Another example of a property for our model of the NSSK protocol is the secrecy property requiring that the intruder never learns the secret key K_{AB}. It can be expressed by the formula:

$$f_\Omega \stackrel{def}{=} \neg\mathbf{Knows}(x, K_{AB}) \tag{4.3.3}$$

Informally (4.3.3) expresses the fact that in any state the intruder must not be able to compose the secret key K_{AB}, starting from the messages he has eavesdropped in the communication channel. Moreover, if we wanted to express the fact that the secret key K_{AB} is known only to the appropriate roles we could use the following:

$f ::= \rho \mid \neg f \mid f_1 \wedge f_2 \mid \Diamond_P f$		formulas
$\rho ::=$ **Knows**(i, ς)		
$\mid$ **Acts**$(i.\lambda(\varsigma)) \mid$ **Acts**(τ)	$\lambda \in \mathcal{A} \cup \overline{\mathcal{A}} \cup \underline{\mathcal{A}}$	
$\mid (\varsigma = \varsigma')$	$i \in \mathcal{I}$	atomic propositions
$\varsigma ::=$ **name**$(i) \mid i.T \mid M$	$T \in \mathcal{T}, M \in \mathcal{M}$	terms

Figure 4.4: Logic Syntax (see also Section 4.2 for definitions of $\mathcal{I}$, $\mathcal{A}$, $\mathcal{T}$, $\mathcal{V}$).

$$f_H \stackrel{def}{=} \forall x.\mathbf{Knows}(x, K_{AB}) \supset ((\mathbf{name}(x) = p_S) \vee (\mathbf{name}(x) = p_A) \vee (\mathbf{name}(x) = p_B)) \tag{4.3.4}$$

This formula says that for all instantiations of x, if x knows the secret key K_{AB} then x is the instantiation of either the role p_S, p_A or p_B. Here we observe that if we wanted to express the fact that roles p_A and p_B know the secret key at the end of the protocol, we can use a slightly different implementation with assertions. For example role p_A can assert to have finished the protocol by the use of $\underline{end}(b)$, as follows:

$$p_A\langle a, b, s, k_{as}\rangle \stackrel{def}{=} \overline{c_{as}}(\langle a,\ b\rangle).c_{as}(x).[x \texttt{ is } \langle\{x_1\}_{k_{as}}, x_2\rangle].\overline{c_{ab}}(x_2).\underline{end}(b).0$$

The modification to process modeling the role of B is similar, and we omit it. Then the property that each honest role knows the key at the end of the protocol can be expressed as $f_A \wedge f_B$ where:

$$f_A \stackrel{def}{=} \forall x.(\mathbf{name}(x) = p_A \wedge \mathbf{Acts}(x.\underline{end}(B))) \supset \mathbf{Knows}(x, K_{AB})$$
$$f_B \stackrel{def}{=} \forall x.(\mathbf{name}(x) = p_B \wedge \mathbf{Acts}(x.\underline{end}(A))) \supset \mathbf{Knows}(x, K_{AB})$$

Additional examples of properties expressed in this logic, can be found in [50, 52, 53] ■

We now explain how the logic is interpreted over the labelled transition systems, models of *spy-calculus* protocols. Message terms are interpreted over a global state, while the interpretation of a formula is defined over traces obtained from the LTS. A trace is the temporal structure over which the satisfiability of formulas is checked. Formally, a *trace* is a finite sequence $\pi = q_0 \cdot \alpha_1 \cdot q_1 \cdot \ldots \cdot \alpha_n \cdot q_n$, where q_0 is the initial state of the LTS and for all i, $q_i \stackrel{\alpha_{i+1}}{\rightarrow} q_{i+1}$ is a transition from q_i.

We now define the interpretation function for each syntactic category of the logic. We will start from message terms.

Definition 4.3.2 (Message Term Interpretation) *Given a global state $\mathcal{G}$, and a term T, the* term interpretation*, is the function $\mathbb{M} : Glob \rightarrow \mathcal{T} \rightarrow \mathcal{M} \cup \{\perp\}$ given below:*

$\mathbb{M}(\mathcal{G})(M)$	$=$	M,	*where* $M \in \mathcal{M}$
$\mathbb{M}(\mathcal{G})(j.T)$	$=$	$\widehat{\sigma}(T)$,	*where* $\mathcal{G}(j) = (\_, \_, \sigma)$
$\mathbb{M}(\mathcal{G})(\mathbf{name}(j))$	$=$	p,	*where* $\mathcal{G}(j) = (p, \_, \_)$
$\perp$			*otherwise*

Informally, **name**(j) is the message which represents, in $\mathcal{G}$, the name of the process instance whose identifier is j; $j.T$ is the message obtained by instantiating all the variables appearing in T, using the set of bindings that the process identified by j has in $\mathcal{G}$.

Atomic propositions ρ (respectively, formulas f) are interpreted over a trace π. We write $q_i \models \rho$ (respectively, $q_i \models f$) when they are satisfied over a state $q_i = (\mathcal{G}_i : Z_i)$ of π as follows:

Definition 4.3.3 (Atomic Proposition Interpretation) *Given a trace $\pi = q_0 \cdot \alpha_1 \cdot q_1 \cdot \ldots \cdot \alpha_n \cdot q_n$, we have, for $1 \leq i \leq n$:*

$$\begin{array}{lll} q_i \models \textbf{Knows}(j, \varsigma) & \textit{iff} & \mathbb{M}(\mathcal{G}_i)(\varsigma) \in KS(W_j), \quad \textit{where } \mathcal{G}_i(j) = (\_, W_j, \_) \\ q_i \models \textbf{Acts}(j.\lambda(\varsigma)) & \textit{iff} & \alpha_i = j.\lambda(M), \quad \textit{where } M = \mathbb{M}(\mathcal{G}_i)(\varsigma) \\ q_i \models \varsigma = \varsigma' & \textit{iff} & \mathbb{M}(\mathcal{G}_i)(\varsigma) = \mathbb{M}(\mathcal{G}_i)(\varsigma') \end{array}$$

If we assume that the evaluation of the message term ς in q_i returns the message M, informally **Knows**(j, M) is satisfied in q_i if the role instance identified by j can derive M from message set W_j. In turn, **Acts**$(j.\lambda(M))$ is satisfied in a state q_i, if $q_{i-1} \stackrel{j.\lambda(M)}{\rightarrow} q_i$ is a transition of the trace π. Finally, $\varsigma = \varsigma'$ is true if the interpretation terms coincide in q_i.

Definition 4.3.4 (Formulae Interpretation) *Given a formula f and a trace $\pi = q_0 \cdot \alpha_1 \cdot \ldots \cdot \alpha_n \cdot q_n$, we have that for $1 \leq i \leq n$:*

$$\begin{array}{lll} q_i \models \rho & \textit{iff} & q_i \models \rho \\ q_i \models \neg f & \textit{iff} & q_i \not\models f \\ q_i \models f_1 \wedge f_2 & \textit{iff} & q_i \models f_1 \textit{ and } q_i \models f_2 \\ q_i \models \Diamond_P f & \textit{iff} & \textit{there exists } j, 0 \leq j \leq i \textit{ such that } q_j \models f \end{array}$$

Informally the interpretation of ρ, $\neg f$ and $f_1 \wedge f_2$ do not differ from the common interpretation of propositional formulas. Instead $\Diamond_P f$ is satisfied in q_i if it is satisfied in some previous state of the trace.

The obvious extension of satisfiability over π is $\pi \models f$ iff $q_i \models f, \forall i : 0 \leq i \leq \text{length}(\pi)$. Finally we say that a protocol Z satisfies a formula f, written $Z \models f$, if f is satisfied over all the traces of the LTS model of Z.

Example 4.3.5 In this example we show how the interpretation of formulas works. Let us assume we have the protocol below, where a role p_A sends an encrypted message to itself via an insecure channel. Formally:

$$\begin{array}{rcl} Z_s & = & (\backslash K)(1, p_A(K)) \\ p_A(k) & = & \overline{c}(\{A\}_k).c(x).\underline{ok}(A).0 \end{array}$$

In the specification we chose to hide the key K used by the agent from the initial knowledge of the intruder, and we suppose that A signals the end of the protocol with an assertion labelled $\underline{ok}()$. The property expressing the fact that, when the protocol finishes, the message received is the same of the one sent, can be expressed as follows:

$$f = \textbf{Acts}(1.\underline{ok}(A)) \supset (1.x = \{A\}_K)$$

Let us now consider one of the traces coming from the labelled transition system model of Z_s, that is $\pi = q_0 \cdot \alpha_1 \cdot q_1 \cdot \alpha_2 \cdot q_2 \cdot \alpha_3 \cdot q_3$ where:

$$\begin{aligned}
q_0 &= [(\Omega, \{A\}, \bot)], [(p_A, \{A, K_A\}, \bot)] \\
\alpha_1 &= 1.\overline{c}(\{A\}_K) \\
q_1 &= [(\Omega, \{A, \{A\}_K\}, \bot)], [(p_A, \{A, K_A\}, \bot)] \\
\alpha_2 &= 1.c(\{A\}_K) \\
q_2 &= [(\Omega, \{A, \{A\}_K\}, \bot)], [(p_A, \{A, K_A\}, [x \leftarrow \{A\}_K])] \\
\alpha_3 &= 1.\underline{ok}(A) \\
q_3 &= q_2
\end{aligned}$$

It is easy to verify that $\pi \models f$. In fact, for $i \in \{0, 1, 2\}$, $q_i \not\models \mathbf{Acts}(1.\underline{ok}(A))$ which makes the implication true, while (a) $q_3 \models \mathbf{Acts}(1.\underline{ok}(A))$ and (b) $q_3 \models (1.x = \{A\}_K)$, which also makes the implication true. (a) is true because $q_2 \stackrel{1.\underline{ok}(A)}{\rightarrow} q_3$ is a transition in π, and (b) is true because x, interpreted over the local state of 1 in q_3, is equal to $\{A\}_K$. It is also the case that $Z \not\models f$ because there exists a trace π' such that $\pi' \not\models f$, for example $\pi' = q'_0 \cdot \alpha'_1 \cdot q'_1 \cdot \alpha'_2 \cdot q'_2 \cdot \alpha'_3 \cdot q'_3$ where:

$$\begin{aligned}
q'_0 &= q_0 \\
\alpha'_1 &= \alpha_1 \\
q'_1 &= q_1 \\
\alpha'_2 &= 1.c(\{A\}_A) \\
q'_2 &= [(\Omega, \{A, \{A\}_K\}, \bot)], [(p_A, \{A, K_A\}, [x \leftarrow \{A\}_A])] \\
\alpha'_3 &= 1.\underline{ok}(A) \\
q'_3 &= q'_2
\end{aligned}$$

In this trace the intruder intercepts the message $\{A\}_K$ and replaces it by $\{A\}_A$. This is sufficient to conclude $Z \not\models f$.

■

4.4 Typed *spy-calculus*

In Section 4.2 it has been pointed out that the use of $KS(W_\Omega)$ generally creates an infinite number of input transitions but, to perform model checking, we need a finite-state labelled transition system. This section studies the possibility of using *type information* (mainly in input actions) to select only those messages of a certain type in the intruder's knowledge. In this way the intruder has only a finite way of composing messages that fulfill a role's input action and, consequently, the corresponding transition system has a finite number of transitions. This suffices to obtain finite models considering that we deal with finite processes and finite protocol runs. It is worth to underline that the use of types is here oriented only to obtain finite-state models, and not to use type checking by way of protocol analysis.

In the following we define: (a) what a type is; (b) a partial relation between types (sub-typing) and (c) the typed version of our calculus, where variables are adorned with types.

Definition 4.4.1 (Type) *A* type t*, is either: (1) a* basic type `proc`, `key`, `nounce` *or* `atom`*, (2) a* pair type $\langle\!|\ t, t\prime\ |\!\rangle$*, (3) a* crypto type $\{\!|\ t\ |\!\}_{t\prime}$*, a finite* union of types $t \oplus t\prime$*. By* $\mathbb{T}$ *we mean the set of all possibles types.*

Types are partially ordered, with the following ordering relation:

Definition 4.4.2 *Let* $t, t' \in \mathbb{T}$*. We say that* t *is a subtype of* t'*written as* $t \leq t'$*, iff*

$$\begin{cases} t = t' \text{ or } \big(t' = t'_1 \oplus t'_2 \text{ and } (t \leq t'_1 \text{ or } t \leq t'_2)\big) \\ t = \{\!|\ t_1\ |\!\}_{t_2} \text{ and } t' = \{\!|\ t'_1\ |\!\}_{t'_2} \text{ and } t_1 \leq t'_1 \text{ and } t_2 \leq t'_2 \\ t = \langle\!|\ t_1, t_2\ |\!\rangle \text{ and } t' = \langle\!|\ t'_1, t'_2\ |\!\rangle \text{ and } t_1 \leq t'_1 \text{ and } t_2 \leq t'_2 \end{cases}$$

Types are used in a slightly extended version of the *spy-calculus*, called the *spyD-calculus*, where names and variables are decorated with their type. The *spyD-calculus* syntax differs from *spy-calculus* in the following points:

- the set of names $\mathcal{N}$ is partitioned among $\mathcal{N}_a$, $\mathcal{N}_p$, $\mathcal{N}_k$ and $\mathcal{N}_n$ of respectively, atomic messages, process names, keys, and nounces/timestamps.
- variables are written as *typed variables* $(x : t)$, where x is a variable and t is a type. The set $\tilde{\mathcal{T}}$ of *typed terms* $\tilde{T}$, is then built, over the signature $\Sigma \cup (\mathcal{V} \times \mathbb{T})$. Protocols $\tilde{Z}$ and roles $\tilde{P}$ remain almost unchanged but typed terms are used instead of terms.

The syntax of *spyD-calculus* is expressed by the following grammar:

$$\begin{array}{lrcl} \textit{protocols} & \tilde{Z} & ::= & \tilde{Z} \| \tilde{Z}' \mid (\backslash N)\tilde{Z} \mid (i, \tilde{P}) \\ \textit{roles} & \tilde{P} & ::= & 0 \mid a(x:t).\tilde{P} \mid \overline{a}(\tilde{T}).\tilde{P} \mid \underline{a}(\tilde{T}).\tilde{P} \mid \\ & & & \tilde{P} + \tilde{P}' \mid (\nu N)\tilde{P} \mid [(x:t)\ \mathtt{is}\ \tilde{T}]\tilde{P} \end{array}$$

We assume that only *well typed* protocols are possible, meaning that all instances of a variable bound by the same, leftmost, input primitive or match must have the same type.

Starting from basic types, explicitly assigned to names and variables, a *top level type* $\lfloor \tilde{T} \rfloor$ for a typed term $\tilde{T}$ can be deduced easily by structural induction over terms[1]. So for example the top level type of the decorated term $\{(x : \mathtt{proc})\}_{(k:\mathtt{key})}$ is $\{\!|\ \mathtt{proc}\ |\!\}_{\mathtt{key}}$. In the following we say that a message term $\tilde{T}$ has type t if $\lfloor \tilde{T} \rfloor = t$, and we write $|t|$ to indicate the number of basic types appearing in t.

Using types we can define a bounded version of the intruder knowledge.

Definition 4.4.3 (Bounded Knowledge) *Let* $W \subset \mathcal{M}$ *be a finite set of messages, and* t *a type. Then* $KS^{(t)}$ *called the* t-knowledge *is the following set of messages:*

$$KS^{(t)}(W) = \{M \in KS(W) : \lfloor M \rfloor \leq t\}$$

About bounded knowledge the following results hold:

Lemma 4.4.4 *Let* W *be a finite set of messages. Then for every* $t \in \mathbb{T}$*,* $KS^{(t)}(W)$ *is a finite set.*

[1] With a little abuse of notation, the same function symbol $\lfloor \_ \rfloor$ is used both for the function returning a type given a message M, and for the function returning a type given a typed term $\tilde{T}$.

Proof. The proof follows from the fact that $|t|$ is finite, and the number of messages whose type is a subtype of t is finite. ■

Example 4.4.5 Assume $W = \{A, K\}$, and $t = \{\!|\ \mathtt{proc}, \mathtt{key} \oplus \mathtt{proc}\ |\!\}$. Then $KS^{(t)}(W) = \{\langle A,\ A\rangle, \langle A,\ K\rangle\}$. ■

Lemma 4.4.6 *Let W be finite. Given a message M, the question $M \in KS^{(t)}(W)$ is decidable.*

Proof. We can easily compute the $\lfloor M \rfloor$ in linear time in the size of M. If $\lfloor M \rfloor \not\leq t$ then $M \notin KS^{(t)}(W)$. Otherwise $\lfloor M \rfloor = t$ and we know from [53] that $M \in KS(W)$ is decidable. ■

We are now ready to restate the operational semantics of *spyD-calculus*, making use of the bounded knowledge $KS^{(t)}(W)$ in input rule. The *spyD-calculus* semantics is based on labelled transition systems, which we call LTS$_\flat$. The definition is almost the same of that of LTS (see Definition 4.2.6) with the exception of the transition rule $\mathbf{r}_?$, which is re-defined in the rule $\tilde{\mathbf{r}}_?$:

$$\frac{\exists \sigma' \sqsupseteq \sigma_i : \sigma'(x) = M \in KS^{(t)}(W_\Omega)}{(\mathcal{G} : \tilde{Z} \Vdash (i, a(x:t).\tilde{P})) \overset{i.a\langle M\rangle}{\rightarrow_\flat} (\mathcal{G}[W_i \cup \{M\}/W_i][\sigma'/\sigma_i] : \tilde{Z} \Vdash (i, \tilde{P}))}\ \tilde{\mathbf{r}}_?$$

In $\tilde{\mathbf{r}}_?$ the bounded knowledge $KS^{(t)}$ is used instead of KS. We write $(\mathcal{G} : \tilde{Z}) \overset{\alpha}{\rightarrow_\flat} (\mathcal{G}' : \tilde{Z}')$ to say that a typed protocol $\tilde{Z}$, and the global state $\mathcal{G}$, change as a consequence of action α. It follows that:

Theorem 4.4.7 *The LTS$_\flat$ is finite-state.*

Proof. Having only finite processes and a finite number of role instances, the source of infinite behavior is due to the input transition rule $\tilde{\mathbf{r}}_?$. But from Lemma 4.4.4 it follows that there can only be a finite number of transitions for each *spyD-calculus* protocol. ■

Logic formulas (see Section 4.3) may be interpreted over the traces $\pi_\flat = \rho_0 \cdot \alpha_1 \cdots \alpha_n \cdot \rho_n$, originating from LTS$_\flat$, exactly in the same way as they are interpreted over traces π coming from LTS. In fact, the satisfiability relation (see Section 4.3) depends on the component $\mathcal{G}$ of states $\rho = (\mathcal{G} : \tilde{Z})$, whose definition has not changed.

4.4.1 Building Typed Protocols

In this section we explain how to obtain, in a semi-automated way, typed *spyD-calculus* protocols starting from a *spy-calculus* specification. We recall from Section 4.1 that our target is to build a typed version at run-time starting from a single *spy-calculus* version.

To avoid some technicalities, we require that protocols are written in the following *normal form*: variables used in the specification of different roles are distinct, and within a single role variables bound by different binders (*e.g.*, input or match operator) are different. All the examples described in this chapter are in normal form.

The basic way of obtaining a typed version of a protocol specification is through a *typing function*:

Definition 4.4.8 (Typing Transformation) *A* typing transformation *is a partial function $\mathcal{S} : \mathcal{V} \rightarrow \mathcal{V} \times \mathbb{T}$, such that, for all x in the domain of $\mathcal{S}$, $\mathcal{S}(x) = x : t$ where t is a type.*

Based on a transformation $\mathcal{S}$ we can define a mapping $\mathcal{C}_{\mathcal{S}}$, that is a structural extension of $\mathcal{S}$ from variables to message terms, processes and protocols. Although different symbols should be used for those extensions, for convenience we will used the same (overloaded) symbol $\mathcal{C}_{\mathcal{S}}$ for all of them. The general idea is that, given a protocol Z, $\mathcal{C}_{\mathcal{S}}(Z)$ is a typed version of Z with finite-state semantics. To be precise, more than general transformations we are interested in transformations whose domain is the set of variables of a given protocol. Formally:

Definition 4.4.9 (Transformation with respect to a Protocol)
Let Z be a protocol specification in normal form. A transformation with respect to Z *is a transformation $\mathcal{S}_Z$ whose domain is exactly the set of variables appearing in Z.*

We can observe that given a protocol Z and a protocol transformation $\mathcal{S}_Z$, the set of bound and free names of Z and $\mathcal{C}_{\mathcal{S}_Z}(Z)$ is the same. In fact a protocol transformation acts only over variables, while names are left untouched. The same can be observed for bound and free names of processes P, involved in Z, and processes $\mathcal{C}_{\mathcal{S}_Z}(P)$ involved in $\mathcal{C}_{\mathcal{S}_Z}(Z)$.

Example 4.4.10 This example shows a typing transformation. Referring to Example 4.2.1 let us define the following protocol transformation function, over the $NSSK$:

$$\mathcal{S}_{NSSK}(v) \stackrel{def}{=} \begin{cases} x : \langle\!\langle\, \{\!|\, \mathtt{key} \,|\}_{\mathtt{key}}, \{\!|\, \langle\!\langle\, \mathtt{proc}, \mathtt{key} \,\rangle\!\rangle \,|\}_{\mathtt{key}} \,\rangle\!\rangle, & \text{if } v = x \\ x_1 : \mathtt{key}, & \text{if } v = x_1 \\ x_2 : \{\!|\, \langle\!\langle\, \mathtt{proc}, \mathtt{key} \,\rangle\!\rangle \,|\}_{\mathtt{key}}, & \text{if } v = x_2 \\ y : \{\!|\, \langle\!\langle\, \mathtt{proc}, \mathtt{key} \,\rangle\!\rangle \,|\}_{\mathtt{key}}, & \text{if } v = y \\ y_1 : \mathtt{key}, & \text{if } v = y_1 \\ z : \langle\!\langle\, \mathtt{proc}, \mathtt{proc} \,\rangle\!\rangle, & \text{if } v = z \\ \bot, & \text{else} \end{cases}$$

The transformation is built following the intuition that messages received in variable y from role p_B indeed are messages of type $\{\!|\, \langle\!\langle\, \mathtt{proc}, \mathtt{key} \,\rangle\!\rangle \,|\}_{\mathtt{key}}$. The typed version of the protocol is then the following:

$$\begin{array}{rcl} \mathcal{C}_{\mathcal{S}_{NSSK}}(NSSK) & \stackrel{def}{=} & (\backslash K_{as})(\backslash K_{bs})(\backslash A)(\backslash B)(\backslash S) \\ & & (1, p_A\langle A, B, S, K_{as}\rangle) \,|\!|\!|\, (2, p_B\langle A, B, S, K_{bs}\rangle) \\ & & |\!|\!|\, (3, p_S\langle A, B, K_{as}, K_{bs}\rangle) \\ p_A\langle a, b, s, k_{as}\rangle & \stackrel{def}{=} & \overline{c_{as}}(\langle a,\, b\rangle).c_{as}(x : \langle\!\langle\, \{\!|\, \mathtt{key} \,|\}_{\mathtt{key}}, \{\!|\, \langle\!\langle\, \mathtt{proc}, \mathtt{key} \,\rangle\!\rangle \,|\}_{\mathtt{key}} \,\rangle\!\rangle) \\ & & [x : \langle\!\langle\, \{\!|\, \mathtt{key} \,|\}_{\mathtt{key}}, \{\!|\, \langle\!\langle\, \mathtt{proc}, \mathtt{key} \,\rangle\!\rangle \,|\}_{\mathtt{key}} \,\rangle\!\rangle = \\ & & \langle \{x_1 : \mathtt{key}\}_{k_{as}}, x_2 : \{\!|\, \langle\!\langle\, \mathtt{proc}, \mathtt{key} \,\rangle\!\rangle \,|\}_{\mathtt{key}} \rangle]. \\ & & \overline{c_{ab}}(x_2 : \{\!|\, \langle\!\langle\, \mathtt{proc}, \mathtt{key} \,\rangle\!\rangle \,|\}_{\mathtt{key}}).0 \\ p_B\langle a, b, s, k_{bs}\rangle & \stackrel{def}{=} & c_{ab}(y : \{\!|\, \langle\!\langle\, \mathtt{proc}, \mathtt{key} \,\rangle\!\rangle \,|\}_{\mathtt{key}}) \\ & & [y : \{\!|\, \langle\!\langle\, \mathtt{proc}, \mathtt{key} \,\rangle\!\rangle \,|\}_{\mathtt{key}} \ \mathtt{is}\ \{\langle a,\, y_1 : \mathtt{key}\rangle\}_{K_{bs}}].0 \\ p_S\langle a, b, k_{as}, k_{bs}\rangle & \stackrel{def}{=} & c_{as}(z : \langle\!\langle\, \mathtt{proc}, \mathtt{proc} \,\rangle\!\rangle)[z : \langle\!\langle\, \mathtt{proc}, \mathtt{proc} \,\rangle\!\rangle \ \mathtt{is}\ \langle a,\, b\rangle]. \\ & & (\nu K_{ab})\overline{c_{as}}(\langle \{K_{ab}\}_{k_{as}}, \{\langle a, K_{ab}\rangle\}_{k_{bs}} \rangle).0 \end{array}$$

■

4.5 Towards Finite Model Checking

After having shown how typed *spyD-calculus* protocols can be obtained from (untyped) *spy-calculus* specifications via the use of transformation functions, we are interested in investigating the relationship between LTS and $\text{LTS}_\flat$. We would like to be sure that an attack over a *spyD-calculus* protocol implies the presence of the same attack over the corresponding *spy-calculus* protocol, and vice-versa.

This section formally shows that there exists a trace inclusion relation between LTS and $\text{LTS}_\flat$: given a transformation $\mathcal{S}_Z$, the set of traces from LTS always includes the set of traces from $\text{LTS}_\flat$. While this is not a surprise, a more interesting result is that given any trace of LTS, a transformation can always be defined to yield the same trace in $\text{LTS}_\flat$. This is mainly due to the fact that a variable can be tagged with a union of types, so our scheme works also in presence of type flaw attacks.

Example 4.5.1 To show how this is possible let us resort to an example. The following standard version of the seven-message Needham-Schroeder Public-Key Protocol:

$$
\begin{array}{lll}
1. & A & \rightarrow S : B \\
2. & S & \rightarrow A : \{PK_B, B\}_{PK_S} \\
3. & A & \rightarrow B : \{N_A, A\}_{PK_B} \\
4. & B & \rightarrow S : A \\
5. & S & \rightarrow B : \{PK_A, A\}_{PK_S} \\
6. & B & \rightarrow A : \{N_A, N_B, B\}_{PK_A} \\
7. & A & \rightarrow B : \{N_B\}_{PK_B}
\end{array}
$$

In [113] Heather *et al* show an interesting type flaw attack on this protocol, in turn derived from [154], when two runs (below) of the protocol are considered (we labelled with α the steps of the first run and with β those of the second runs):

$$
\begin{array}{lll}
\alpha_3. & I_A & \rightarrow B : \{N_I, A\}_{PK_B} \\
\alpha_4. & B & \rightarrow S : A \\
\alpha_5. & S & \rightarrow B : \{PK_A, A\}_{PK_S} \\
\alpha_6. & B & \rightarrow I_A : \{N_I, N_B, B\}_{PK_A} \\
\beta_3. & I_{(N_B,B)} & \rightarrow A : \{N_I, (N_B, B)\}_{PK_A} \\
\beta_4. & B & \rightarrow I_S : (N_B, B) \\
\alpha_7. & I_A & \rightarrow B : \{N_B\}_{PK_B}
\end{array}
$$

An intruder, playing the role of A in the first run, receives the message $\{N_I, N_B, B\}_{PK_A}$ from B in step 6, which uses this message subsequently in step 3 of the second run. In this second run A is playing the role of B, and so A interprets the message as the start of a new protocol. Consequently A takes the field (N_B, B) as an agent name, and when A tries, in step 4, to request (N_B, B)'s public key (by sending the (N_B, B) identity to the server) this message is intercepted

again and used by the intruder to end correctly the first run of the protocol. We can model the protocol in *spy-calculus* as:

$$(\backslash K_s)(\backslash K_a)(\backslash K_b)(\backslash A)(\backslash B)(\backslash S)$$
$$(1, p_A\langle A, B, S, K_a\rangle) \| (2, p_A\langle B, A, S, K_b\rangle) \|$$
$$(3, p_B\langle A, B, S, K_b\rangle) \| (4, p_B\langle B, A, S, K_a\rangle) \|$$
$$(5, p_S\langle A, B, K_a, K_b, K_s\rangle) \| (6, p_S\langle A, B, K_a, K_b, K_s\rangle)$$

Here we have two instantiations of each role. In particular the role of p_A (respectively of p_B) is played once by A (respectively by B) and once by B (respectively by A). The first steps of role p_B are the following:

$$p_B\langle a, b, s, k_b\rangle \stackrel{def}{=} \overline{c}(x).[x \texttt{ is } \{\langle x_n,\, x_a\rangle\}_{k_b}].\overline{c}(x_a).P'$$

Here P' represents the continuation of the process. To catch the type flaw we have to find a typing that allows the variable x to match with both messages used in the attack. One such a possible type transformation for x is:

$$\mathcal{S}_{NSSK}(x) = x : \{\!|\, \langle\!|\, \texttt{nounce}, \texttt{proc} \oplus \langle\!|\, \texttt{nounce}, \texttt{proc} \,|\!\rangle \,|\!\rangle \,|\!\}_{\texttt{key}}$$

■

Unfortunately it seems not easy to find such a typing transformation without knowing the attack first. Even in the formal proof (see later) our result is not constructive and we have no general method to construct this abstraction. We conjecture that a static analysis of the message flow along a protocol specification may help in defining significant transformations, but we have not yet investigated in this direction. We point out this issue as an area of future work.

To arrive at the main results of this section, we start with some definitions, introducing basic equivalence relations among global states and traces of the relative transition systems.

Definition 4.5.2 *Given a transformation function $\mathcal{S}$, let $q = (\mathcal{G} : Z)$ be a state of the LTS, and $\rho = (\mathcal{G}' : \tilde{Z}')$ a state of the LTS$_\flat$. We say that they are* equal up to $\mathcal{S}$ *and we write $q =_{\mathcal{S}} \rho$, iff*

- $\mathcal{G} = \mathcal{G}'$;
- $\mathcal{C}_{\mathcal{S}}(Z) = \tilde{Z}'$ *where the symbol $=$ is interpreted as syntactic equality.*

Definition 4.5.3 *Given a transformation function $\mathcal{S}$, let $\pi = q_0 \cdot \alpha_1 \cdot q_1 \cdot \ldots \cdot \alpha_n \cdot q_n$, and $\pi_\flat = \rho_0 \cdot \alpha'_1 \cdot \rho_1 \cdot \ldots \cdot \alpha'_n \cdot \rho_n$ be two traces. We say that they are* equal up to $\mathcal{S}$ *and we write $\pi =_{\mathcal{S}} \pi_\flat$ iff for all k*

- $\alpha_k = \alpha'_k$;
- $q_k =_{\mathcal{S}} \rho_k$.

In the following we prove the main lemmas of this section. One states that given a trace in the LTS$_\flat$ model of $\mathcal{C}_{\mathcal{S}_Z}(Z)$, there always exists a corresponding trace (with respect to $=_{\mathcal{S}}$) in the LTS model of Z. The second lemma proves that given a trace in the LTS model of Z, there always exists a typing transformation $\mathcal{C}_{\mathcal{S}_Z}(\_)$ such that a corresponding trace (with respect to $=_{\mathcal{S}}$) exists in the LTS$_\flat$ model of $\mathcal{C}_{\mathcal{S}_Z}(Z)$.

Lemma 4.5.4 *Suppose that Z is a protocol specification in normal form, and $\mathcal{S}_Z$ a protocol transformation with respect to Z. Let Π_Z be the set of traces from the LTS, model of Z, and let $\Pi_{\mathcal{C}_{\mathcal{S}_Z}(Z)}$ be the set of traces of the LTS$_\flat$, model of $\mathcal{C}_{\mathcal{S}_Z}(Z)$. Then for each trace $\pi_\flat \in \Pi_{\mathcal{C}_{\mathcal{S}_Z}(Z)}$ there exists a trace $\pi \in \Pi_Z$ such that $\pi =_{\mathcal{S}_Z} \pi_\flat$.*

Proof. We proceed by induction on the length of the trace $\pi_\flat = \rho_0 \cdot \alpha_1 \cdot \rho_1 \cdot \ldots \cdot \alpha_k \cdot \rho_k$. For any $n \geq 0$, we show that a trace π exists whose prefix of length n (written $\pi[n]$) is equal up to $\mathcal{S}_Z$ to the prefix of length n, of $\pi_\flat$ (written $\pi_\flat[n]$).

[**base case**: $n = 0$]. In this case $\pi_\flat[0] = (\mathcal{G}_0 : \mathcal{C}_{\mathcal{S}_Z}(Z))$. Whatever trace $\pi \in \Pi_Z$ we chose, $\pi[0] =_{\mathcal{S}_Z} \pi_\flat[0]$ holds. In fact, $\pi[0] = (\mathcal{G}_0 : Z)$ and $\pi[0] =_{\mathcal{S}_Z} \pi_\flat[0]$ immediately follows from the definition of "$=_{\mathcal{S}_Z}$".

[**inductive step**: $n > 0$]. Let us assume that the theorem holds for n. Then there exists a trace $\pi \in \Pi_Z$ such that $\pi[n] =_{\mathcal{S}_Z} \pi_\flat[n]$. We prove that we can extend the trace $\pi[n]$ so that $\pi[n+1]$ is a trace of Π_Z and $\pi[n+1] =_{\mathcal{S}_Z} \pi_\flat[n+1]$.

Let $\pi_\flat[n+1]$ be $\pi_\flat[n] \cdot \alpha_{n+1} \cdot \rho_{n+1}$. We distinguish the cases of α_{n+1}.

- $[\alpha_{n+1} = i.a\langle M\rangle, \text{ for some } i]$. This action arises only through the following transition (*i.e.*, by rule $\tilde{\mathbf{r}}_?$)

$$\overbrace{(\mathcal{G}_n : \tilde{Z} \| (i, a(x:t).\tilde{P}))}^{\rho_n} \xrightarrow{i.a\langle M\rangle}_\flat \overbrace{(\underbrace{\mathcal{G}_n[W_i \cup \{M\}/W_i][\sigma'/\sigma_i]}_{\mathcal{G}_{n+1}} : \tilde{Z} \| (i, \tilde{P}))}^{\rho_{n+1}}$$

  Here $\sigma' : \sigma' \sqsupseteq \sigma_i$ and $\sigma'(x) = M \in KS^{(t)}(W_\Omega)$. By the induction hypothesis there exists a trace $\pi \in \Pi_Z$ such that $\pi[n] =_{\mathcal{S}_Z} \pi_\flat[n]$. In particular $\pi[n] = q_0 \cdot \alpha_1 \cdot \ldots \cdot \alpha_n \cdot q_n$, and $q_n =_{\mathcal{S}_Z} \rho_n$. If we assume that $q_n = (\mathcal{G}', Z')$ this means that (a) $\mathcal{G}' = \mathcal{G}_n$ and (b) $\mathcal{C}_{\mathcal{S}_Z}(Z') = \tilde{Z} \| (i, a(x:t).\tilde{P})$. We observe that $M \in KS(W_\Omega)$, because $KS^{(t)}(W_\Omega) \subseteq KS(W_\Omega)$, and this implies that the transition

$$\overbrace{(\mathcal{G}_n : Z \| (i, a(x).P))}^{q_n} \xrightarrow{i.a\langle M\rangle} \overbrace{(\underbrace{\mathcal{G}_n[W_i \cup \{M\}/W_i][\sigma'/\sigma_i]}_{\mathcal{G}_{n+1}} : Z \| (i, P))}^{q_{n+1}}$$

  is possible. It is easy to check that $q_{n+1} =_{\mathcal{S}_Z} \rho_{n+1}$, and this suffices to conclude that $\pi[n+1] =_{\mathcal{S}_Z} \pi_\flat[n+1]$.

- $[\alpha_{n+1} = i.\overline{a}\langle M\rangle, \text{ for some } i]$. This action arises only through the following transition (*i.e.*, rule $\tilde{\mathbf{r}}_!$):

$$\overbrace{(\mathcal{G}_n : \tilde{Z} \| (i, \overline{a}(\tilde{T}).\tilde{P}))}^{\rho_n} \xrightarrow{i.\overline{a}\langle M\rangle} \overbrace{(\underbrace{\mathcal{G}_n[W_\Omega \cup \{M\}/W_\Omega]}_{\mathcal{G}_{n+1}} : \tilde{Z} \| (i, \tilde{P}))}^{\rho_{n+1}}$$

  Here $\widehat{\sigma}_i(\tilde{T}) = M$. By the induction hypothesis there exists a trace $\pi \in \Pi_Z$ such that $\pi[n] =_{\mathcal{S}_Z} \pi_\flat[n]$. In particular $\pi[n] = q_0 \cdot \alpha_1 \cdot \ldots \cdot \alpha_n \cdot q_n$, and $q_n =_{\mathcal{S}_Z} \rho_n$. If we assume that $q_0 = (\mathcal{G}, q)$ this means that: (a) $\mathcal{G}' = \mathcal{G}_n$, (b) $\mathcal{C}_{\mathcal{S}_Z}(P') = (i, \overline{a}(\tilde{T}).\tilde{P})$. Moreover we have $\widehat{\sigma}_i(\tilde{T}) = \widehat{\sigma}_i(T) = M$. This means that the transition

$$\overbrace{(\mathcal{G}_n : Z \| (i, \overline{a}(T).P))}^{q_n} \stackrel{i.\overline{a}\langle M\rangle}{\rightarrow} \overbrace{(\underbrace{\mathcal{G}_n[W_\Omega \cup \{M\}/W_\Omega]}_{\mathcal{G}_{n+1}} : Z \| (i, P))}^{q_{n+1}}$$

is possible. It is easy to check that $q_{n+1} =_{\mathcal{S}_Z} \rho_{n+1}$, and this suffices to conclude that $\pi[n+1] =_{\mathcal{S}_Z} \pi_\flat[n+1]$.

- $[\alpha_{n+1} = \tau]$. The only interesting case (we omit the other cases that are obvious) is when the action is due to the following transition (*i.e.,* rule $\mathbf{r}^{\sim}_{=}$):

$$\overbrace{(\mathcal{G}_n : \tilde{Z} \| (i, [(x:t)\ \mathtt{is}\ \tilde{T}]\tilde{P}))}^{\rho_n} \stackrel{\tau}{\rightarrow} \overbrace{(\underbrace{\mathcal{G}_n[\sigma'/\sigma_i]}_{\mathcal{G}_{n+1}} : \tilde{Z} \| \tilde{P})}^{\rho_{n+1}}$$

Here $\mathcal{P}(\widehat{\sigma}_i(\tilde{T}))$ and $\exists \sigma' \sqsupseteq \sigma_i : \widehat{\sigma}'(\tilde{T}) = \sigma_i(x)$. Let $\pi[n]$ be $q_0 \cdot \alpha_1 \cdot \ldots \cdot \alpha_n \cdot q_n$, and consider the state $q_n = (\mathcal{G}' : Z')$. By the induction hypothesis we know that $q_n =_{\mathcal{S}_Z} \rho_n$ and as a consequence we have that: (a) $\mathcal{G}' = \mathcal{G}_n$, (b) $\mathcal{C}_{\mathcal{S}_Z}(P') = (i, [(x:t)\ \mathtt{is}\ \tilde{T}].\tilde{P})$, and (c) $\widehat{\sigma}_i(\tilde{T}) = \widehat{\sigma}_i(T) = M$. This means that the transition

$$\overbrace{(\mathcal{G}_n : Z \| (i, [x\ \mathtt{is}\ T]P))}^{\rho_n} \stackrel{\tau}{\rightarrow} \overbrace{(\underbrace{\mathcal{G}_n[\sigma/\sigma_i]}_{\mathcal{G}_{n+1}} : Z \| P)}^{\rho_{n+1}}$$

is possible. It is easy to check that $q_{n+1} =_{\mathcal{S}_Z} \rho_{n+1}$, and this suffices to conclude that $\pi[n+1] =_{\mathcal{S}_Z} \pi_\flat[n+1]$.

- $[\alpha_{n+1} = i.\underline{a}\langle M\rangle, \text{ for some } i]$. Similar to the case $\alpha_{n+1} = i.\overline{a}\langle M\rangle$.

■

Lemma 4.5.5 *Suppose that Z is a protocol in normal form, and Π_Z be the set of traces from the LTS model of Z. Then for each trace $\pi \in \Pi_Z$ there exists a transformation S such that in the set of traces $\Pi_{\mathcal{C}_{\mathcal{S}_Z}(Z)}$ of the LTS$_\flat$ model of $\mathcal{C}_{\mathcal{S}_Z}(Z)$ there is a trace $\pi_\flat$ such that $\pi_\flat =_{\mathcal{S}_Z} \pi$.*

Proof. Let π be $q_0 \cdot \alpha_1 \cdots \alpha_n \cdot q_n$. First of all let us define $\mathcal{S}_Z$. The idea is that the type of a variable x is the union of types of the messages that are bound to x along the trace. Because our protocols are in normal form, for each role instance, each variable is bound only by one operator (*i.e.,* input or match). This means that, within a role, only one message is bound to each variable. Globally more messages can be bound to a variable x but only in different role instances. Let us construct $\mathcal{S}_Z$ in the following way:

Remark 4.5.6 Assume $q = (\mathcal{G}', Z')$ is a state of a trace π from the LTS model of a protocol Z, and let $\mathcal{S}_Z$ be the function returned by the algorithm 1. We have that M is the message bound to a variable x (*i.e.,* if $M = \sigma_i(x)$, where $\mathcal{G}(i) = (p_i, \sigma_i, W_i)$ for some i) then

$$\lfloor M \rfloor \leq \lfloor \mathcal{S}_Z(x) \rfloor$$

■

Algorithm 1 function *BuildTypingFunction*(π). Given a trace π returns a typing function $\mathcal{S}(Z)$.

1: $\mathcal{S}_Z \leftarrow \perp$;
2: **for all** $i : \alpha_i \in \pi$ **do**
3: **let** σ_{i-1} the binding function in q_{i-1}
4: **let** σ_i the binding function in q_i
5: **let** $V_{i-1} = \{x : \sigma_{i-1}(x) = \perp\}$ {the set of vars not bound in q_{i-1}}
6: **let** $V_i = \{x : \sigma_i(x) \neq \perp\}$ {the set of vars bound to some message in state q_i}
7: **let** $V = V_i \cap V_{i-1}$ {exactly the set of vars assigned in consequence of the action α_i}
8: **for all** $x \in V$ **do**
9: **if** $\mathcal{S}_Z(x) = \perp$ **then**
10: $\mathcal{S}_Z(x) \leftarrow x : \lfloor \sigma_i(x) \rfloor$
11: **else**
12: $\mathcal{S}_Z(x) \leftarrow \mathcal{S}_Z(x) \oplus \lfloor \sigma_i(x) \rfloor$ {the type of x is the type of each message received in x}
13: **end if**
14: **end for**
15: **end for**

At this point we can prove that there exists a trace $\pi_\flat \in \mathcal{C}_{\mathcal{S}_Z}(Z)$ such that $\pi_\flat =_{\mathcal{S}_Z} \pi$. The proof is by induction over the length of π. We show that for any $n \geq 0$ we can find a trace $\pi_\flat$ such that $\pi[n] = \pi_\flat[n]$

[**base case**: $n = 0$]. In this case $\pi[0] = (\mathcal{G}_0 : Z)$. Whatever trace $\pi_\flat \in \Pi_{\mathcal{C}_{\mathcal{S}_Z}(Z)}$ we chose $\pi_\flat[0] =_{\mathcal{S}_Z} \pi[0]$ holds. In fact, $\pi[0] = (\mathcal{G}_0 : Z)$ and for definition $\pi[0] =_{\mathcal{S}_Z} \pi_\flat[0]$ trivially holds for definition of "$=_{\mathcal{S}_Z}$".

[**inductive step**: $n > 0$]. Suppose that the theorem holds for n, we will prove the theorem for $n+1$. Let $\pi[n+1]$ be $\pi[n] \cdot \alpha_{n+1} \cdot q_{n+1}$. We will distinguish by cases over α_{n+1}.

- $[\alpha_{n+1} = i.a\langle M\rangle, \text{ for some } i]$. This action arises only through the following transition (*i.e.*, by rule $\mathbf{r}_?$):

$$\overbrace{(\mathcal{G}_n : Z \| (i, a(x).P))}^{q_n} \xrightarrow{i.a\langle M\rangle} \overbrace{(\underbrace{\mathcal{G}[W_i \cup \{M\}/W_i][\sigma'/\sigma_i]}_{\mathcal{G}_{n+1}} : Z \| (i, P))}^{q_{n+1}}$$

  where $\sigma' : \sigma' \sqsupseteq \sigma_i$ and $\sigma'(x) = M \in KS(W_\Omega)$. By the induction hypothesis there exists a trace $\pi_\flat \in \Pi_{\mathcal{C}_{\mathcal{S}_Z}(Z)}$ such that $\pi_\flat[n] =_{\mathcal{S}_Z} \pi[n]$. In particular $\pi_\flat[n] = \rho_0 \cdot \alpha_1 \cdot \ldots \cdot \alpha_n \cdot \rho_n$, and $\rho_n = (\mathcal{G}_n : \mathcal{C}_{\mathcal{S}_Z}(Z) \| (i, a(x : \mathcal{S}_Z(x)).\mathcal{C}_{\mathcal{S}_Z}(P)))$. In addition (see remark 4.5.6) we have that $\lfloor M \rfloor \leq \lfloor \mathcal{S}_Z(x) \rfloor$. This means that $M \in KS^{(t)}(W_\Omega)$, where $t = \lfloor \mathcal{S}_Z(x) \rfloor$, and the transition

$$\overbrace{(\mathcal{G}_n : \mathcal{C}_{\mathcal{S}_Z}(Z) \| (i, a(x : \mathcal{S}_Z(x)).\mathcal{C}_{\mathcal{S}_Z}(P)))}^{\rho_n} \xrightarrow{i.a\langle M\rangle}_\flat \overbrace{(\underbrace{\mathcal{G}_n[W_i \cup \{M\}/W_i][\sigma'/\sigma_i]}_{\mathcal{G}_{n+1}} : \mathcal{C}_{\mathcal{S}_Z}(Z) \| \mathcal{C}_{\mathcal{S}_Z}(P))}^{\rho_{n+1}}$$

  is possible. It is easy to check that $q_{n+1} =_{\mathcal{S}_Z} \rho_{n+1}$.

- $[\alpha_{n+1} = i.\overline{a}\langle M\rangle, i.\underline{a}\langle M\rangle($ for some $i),$ and $\alpha_{n+1} = \tau]$. These cases can be proved as in Lemma 4.5.4.

Now we analyze the impact of the protocol transformation on the satisfiability of formulas. In particular given a protocol specification Z, a protocol transformation $\mathcal{S}_Z$, and a formula f to be checked, we want to be sure that if f can be checked over $\mathcal{C}_{\mathcal{S}_Z}(Z)$. The answer is given by the following theorems.

Theorem 4.5.7 *Given a protocol Z, a transformation $\mathcal{S}_Z$ over Z, and a logic formula f. Let Π_Z be the set of traces of the LTS model of Z, and let $\Pi_{\mathcal{C}_{\mathcal{S}_Z}(Z)}$ be the set of traces of the LTS$_\flat$ model of $\mathcal{C}_{\mathcal{S}_Z}(Z)$, and let $\pi \in \Pi_Z$ be and $\pi_\flat \in \Pi_{\mathcal{C}_{\mathcal{S}_Z}(Z)}$ such that $\pi =_{\mathcal{S}_Z} \pi_\flat$. Then $\pi \models f$ iff $\pi_\flat \models \mathcal{C}_{\mathcal{S}_Z}(f)$.*

Proof. When $\pi =_{\mathcal{S}_Z} \pi_\flat$, π and $\pi_\flat$ coincide over the global state; the result follows because $\models$ is defined over the global states. ■

Theorem 4.5.8 *Given a protocol Z and a protocol transformation $\mathcal{S}_Z$ over Z. Suppose that f is a logic formula. Then $\mathcal{C}_{\mathcal{S}_Z}(Z) \not\models f$ implies $Z \not\models f$*

Proof. $\mathcal{C}_{\mathcal{S}_Z}(Z) \not\models f$ means that there exists a trace $\pi_\flat = \rho_0 \cdot \alpha_1 \cdot \ldots \cdot \alpha_n \cdot \rho_n$, of LTS$_\flat$ such that $\pi_\flat \not\models f$. By Lemma 4.5.4 there must exist a trace π of LTS such that $\pi_\flat =_{\mathcal{S}_Z} \pi$. By Theorem 4.5.7 we have that $\pi_\flat \not\models f$. ■

Theorem 4.5.9 *Given a protocol Z and a formula f such that $Z \not\models f$. Then there exists a protocol transformation $\mathcal{S}_Z$ such that $\mathcal{C}_{\mathcal{S}_Z}(Z) \not\models f$.*

Proof. $Z \not\models f$ means that there exists a trace $\pi = \rho_0 \cdot \alpha_1 \cdot \ldots \cdot \alpha_n \cdot \rho_n$, of Π_Z such that $\pi \not\models f$. By Lemma 4.5.5 there exists a protocol transformation $\mathcal{S}_Z$ and a trace π of $\Pi_{\mathcal{C}_{\mathcal{S}_Z}(Z)}$ such that $\pi_\flat =_{\mathcal{S}_Z} \pi$, and by Theorem 4.5.7 we have that $\pi_\flat \not\models f$. ■

Theorem 4.5.9 is possible because variables may assume, via $\mathcal{S}_Z$, potentially any type. This makes our type system too general to be used in a static type checking framework.

4.5.1 Model Checking the *spy-calculus*

This section presents SPYDER, the model checking framework we intend to use for verifying security. The kernel procedure of SPYDER is described by Algorithm 2, a simple procedure which visits a finite labelled transition system in a depth first search mode. As parameters the algorithm requires a closed protocol Z in normal form, a protocol transformation $\mathcal{S}_Z$, and a formula f to be checked. Informally Algorithm 2 uses two stacks: (a) a stack $\mathcal{S}_\Lambda$ containing transitions that implement the depth first traversal of the transition system; (b) a stack $\mathcal{S}_\Pi$ for storing prefixes of traces. In particular during the depth first traversal of transitions, prefixes $\rho_0 \cdot \alpha_1 \cdots \alpha_i \cdot \rho_i$ are built and the satisfiability of the formula f is checked over the state q_i. The procedure stops with a counterexample if f is discovered to be unsatisfied in some ρ_i.

Algorithm 2 has time complexity $O(|V| \cdot (|f| + |W_{max}| \cdot |M_{max}|))$ where $|V|$ is the number of states of the transition system, $|f|$ is the length of formula f, W_{max} is the greatest W_Ω and M_{max} is the biggest message used within the protocol. It is worth to underline that using the protocol transformation $\mathcal{S}_Z$ limits the number of traces of the model to a finite number, even if it remains exponential (when an exponential number of messages is involved in input actions). This means

that our model checker runs in exponential time with respect to the length of messages involved in the protocol.

We observe that the depth of the tree representing the LTS is linear in the number of steps made by the agent. Therefore by using a on-the-fly generation strategy a depth first search requires linear space. SPYDER has been implemented in Ocaml[2], and its main modules are represented in Figure 4.5. The first module lexically checks a *spy-calculus* specification and renames variables to obtain a normal form of the protocol. This specification is then parsed and an internal representation of the protocol is built. The user provides also a logic formula and a typing function that are given as input to the model checker module. The execution finishes with success or with a trace showing why the formula is not satisfied.

Example 4.5.10 We continue example 4.4.10 showing how a formula is checked over a typed version of the protocol:

$$
\begin{array}{rcl}
\mathcal{C}_{\mathcal{S}_{NSSK}}(NSSK) & \stackrel{def}{=} & (\backslash K_{as})(\backslash K_{bs})(\backslash A)(\backslash B)(\backslash S) \\
& & (1, p_A\langle A, B, S, K_{as}\rangle) \| (2, p_B\langle A, B, S, K_{bs}\rangle) \\
& & \|(3, p_S\langle A, B, K_{as}, K_{bs}\rangle) \\
p_A\langle a, b, s, k_{as}\rangle & \stackrel{def}{=} & \overline{c_{as}}(\langle a,\, b\rangle).c_{as}(x : \{\!|\, \langle\!|\, \{\!|\, \mathtt{key} \,|\!\}_{\mathtt{key}}, \{\!|\, \langle\!|\, \mathtt{proc}, \mathtt{key} \,|\!\rangle \,|\!\}_{\mathtt{key}} \,|\!\rangle) \\
& & [x : \{\!|\, \langle\!|\, \{\!|\, \mathtt{key} \,|\!\}_{\mathtt{key}}, \{\!|\, \langle\!|\, \mathtt{proc}, \mathtt{key} \,|\!\rangle \,|\!\}_{\mathtt{key}} \,|\!\rangle = \\
& & \langle \{x_1 : \mathtt{key}\}_{k_{as}}, x_2 : \{\!|\, \langle\!|\, \mathtt{proc}, \mathtt{key} \,|\!\rangle \,|\!\}_{\mathtt{key}}\rangle]. \\
& & \overline{c_{ab}}(x_2 : \{\!|\, \langle\!|\, \mathtt{proc}, \mathtt{key} \,|\!\rangle \,|\!\}_{\mathtt{key}}).0 \\
p_B\langle a, b, s, k_{bs}\rangle & \stackrel{def}{=} & c_{ab}(y : \{\!|\, \langle\!|\, \mathtt{proc}, \mathtt{key} \,|\!\rangle \,|\!\}_{\mathtt{key}}) \\
& & [y : \{\!|\, \langle\!|\, \mathtt{proc}, \mathtt{key} \,|\!\rangle \,|\!\}_{\mathtt{key}}\ \mathtt{is}\ \{\langle a,\, y_1 : \mathtt{key}\rangle\}_{K_{bs}}].0 \\
p_S\langle a, b, k_{as}, k_{bs}\rangle & \stackrel{def}{=} & c_{as}(z : \langle\!|\, \mathtt{proc}, \mathtt{proc} \,|\!\rangle)[z : \langle\!|\, \mathtt{proc}, \mathtt{proc} \,|\!\rangle\ \mathtt{is}\ \langle a,\, b\rangle]. \\
& & (\nu K_{ab})\overline{c_{as}}(\langle \{K_{ab}\}_{k_{as}}, \{\langle a, K_{ab}\rangle\}_{k_{bs}}\rangle).0
\end{array}
$$

Figure 4.6 shows the transition system model of $\mathcal{C}_{\mathcal{S}_Z}(KE)$, while in Figure 4.7 we report a table containing detailed information about the most significant states of the transition system (the ones boxed in Figure 4.6). In the first column, state names q are reported, in the second the arrays representing the global states $\mathcal{G}$ (each element of the array lays in a different row), and finally in the third column the fragment of the calculus Z representing the protocol evolution. We want to stress that the transition system is now finite state.

The formula given in example 4.4.10 (also reported at the end of the paragraph) and expressing an authenticity property can be checked over the finite labelled transition system. It is easy to verify that the formula is not satisfied. For example, in the trace $q_0 \cdot \overline{\lambda_1} \ldots \overline{\lambda_7} q_7$ process B receives a message before A sends it, proving that the intruder has maliciously assumed A's identity.

$$
\begin{array}{rcll}
f & \stackrel{def}{=} & \forall b.\, \exists a.\, \mathbf{name}(b) = B \ \wedge\ \mathbf{name}(a) = A \ \wedge & (4.5.1) \\
& & \mathbf{Acts}(c_{ab}) b\{\langle A,\, x_2\rangle\}_{K_{BS}} \supset \Diamond_P \mathbf{Acts}(\overline{c_{ab}}) a y_1 \ \wedge \\
& & a.y_1 = b.\{\langle A,\, x_2\rangle\}_{K_{BS}}
\end{array}
$$

[2]Ocaml is available on line at the web site `http:\\caml.inria.fr`

Algorithm 2 function *ModelChecking* (Z is a closed protocol in normal form, f is formula and $\mathcal{S}_Z$ is a transformation w.r.t. Z)

1: $\tilde{Z} \leftarrow \mathcal{C}_{\mathcal{S}_Z}(Z)$ {Get a typed protocol, using $\mathcal{S}_Z$}
2: $\rho_0 = (\mathcal{G}_0 : \tilde{Z})$ {Set the initial state}
3: $\mathcal{S}_\Lambda \leftarrow \emptyset$ {an empty stack for containing actions}
4: $\mathcal{S}_\Pi \leftarrow \emptyset$ {an empty stack containing actions and states (i.e., prefix of traces)}
5: $\texttt{push}(\mathcal{S}_\Pi, \epsilon \cdot \rho_0)$
6: **repeat**
7: $\_ \cdot \rho \leftarrow \texttt{head}(\mathcal{S}_\Pi)$ {retrieve the element on the top of the stack (only the state is significant here)}
8: **if** not $mark(\rho)$ **then** {$mark(\rho)$ equal true means that ρ has been not visited yet}
9: $mark(\rho) \leftarrow \texttt{tt}$ {mark ρ as "visited"}
10: **if** $(\rho : \mathcal{S}_\Pi) \not\models f$ **then** {if f is not satisfied over ρ along trace $\mathcal{S}_\Pi$}
11: **return** $\texttt{ff}, \mathcal{S}_\Pi$ {failure and return a counterexample}
12: **else** {f is satisfied}
13: $\Lambda \leftarrow \{\alpha : \alpha$ is an enabled transition from state $\rho\}$
14: **if** $\Lambda = \emptyset$ **then** {no transition is possible}
15: $\texttt{pop}(\mathcal{S}_\Pi)$ {delete head element $\alpha \cdot \rho$ from trace stack (i.e., backtrack)}
16: **else**
17: $\forall \alpha \in \Lambda$, $\texttt{push}(\mathcal{S}_\Lambda, \alpha)$ {push the enabled actions into $\mathcal{S}_\Lambda$}
18: **end if**
19: **end if**
20: **end if**
21: **if** $\mathcal{S}_\Lambda \neq \emptyset$ **then** {if some transition remains}
22: $\alpha \leftarrow \texttt{pop}(\mathcal{S}_\Lambda)$ {retrieve next transition (i.e., depth first search)}
23: **let** $\rho' : \rho \xrightarrow{\alpha} \rho'$ **in** $\texttt{push}(\mathcal{S}_\Pi, \alpha \cdot \rho')$ {extend the trace adding the suffix $\alpha \cdot \rho'$}
24: $\rho \leftarrow \rho'$
25: **else**
26: $\texttt{pop}(\mathcal{S}_\Pi)$ {backtrack}
27: **end if**
28: **until** $\mathcal{S}_\Pi = \emptyset$ {all states have been visited}
29: **return** tt {success}

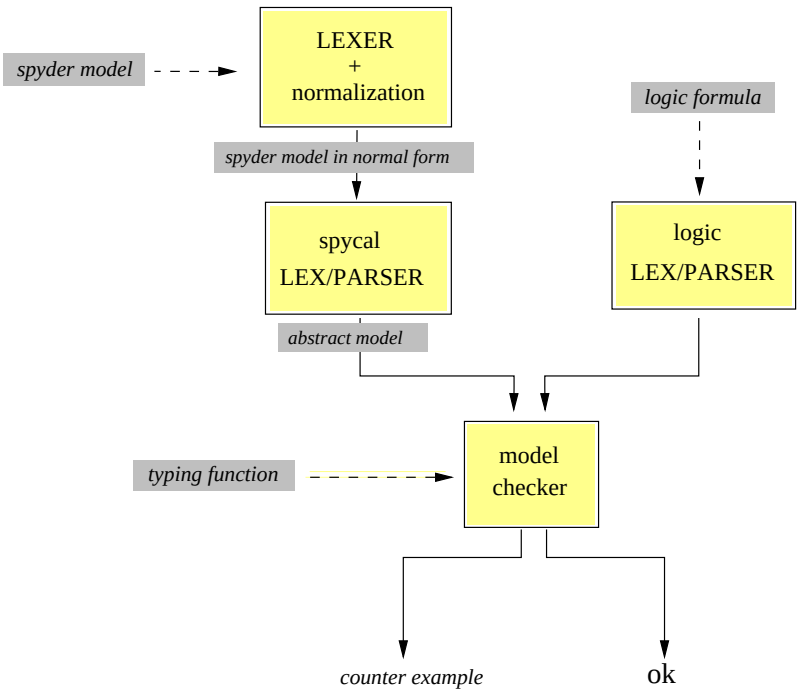


Figure 4.5: The architecture of the SPYDER prototype implemented in Ocaml

■

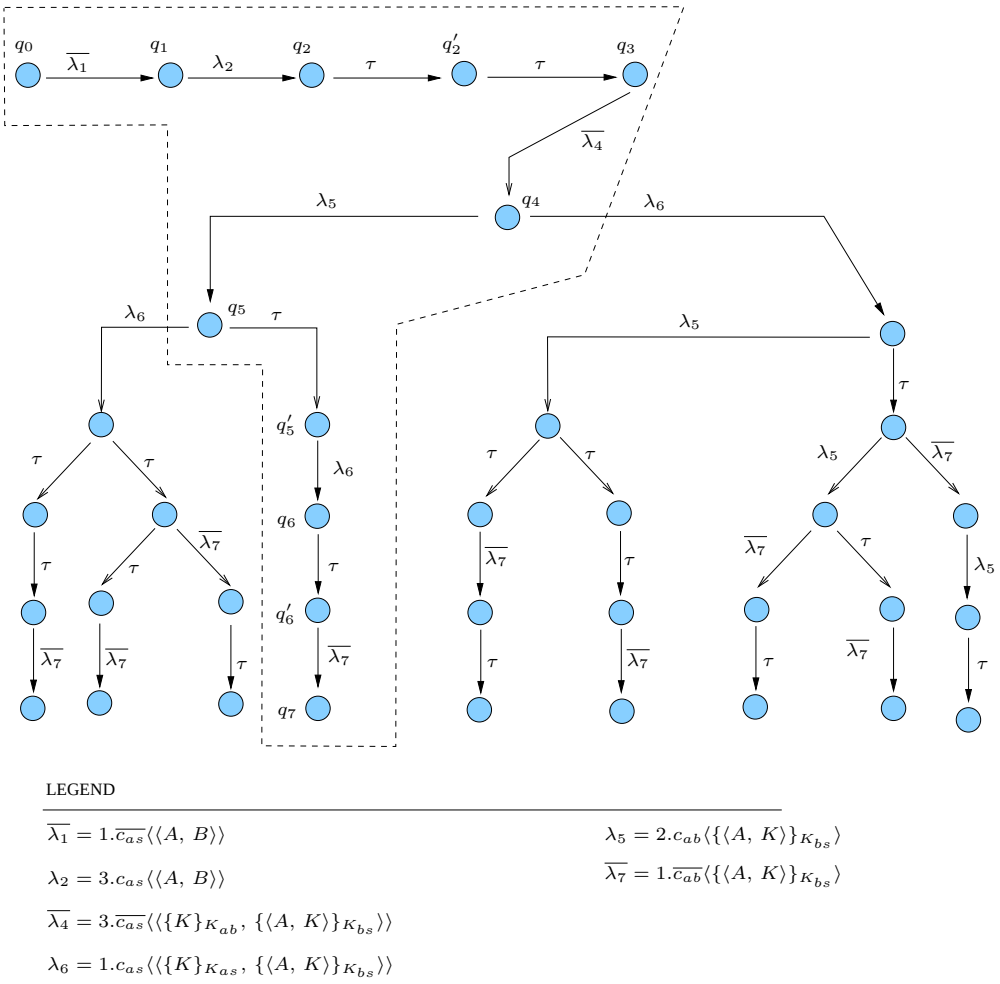
4.6 Conclusions

This chapter presents SPYDER, a model checking environment for a typed spi-calculus dialect. A protocol is specified as a term of a formal calculus called the *spy-calculus* describing a parallel composition of a finite number of process instances, each representing a finite-behaviored role running the protocol. More runs of a protocol can be described by instantiating more copies of each agent. The *spy-calculus* has an operational semantics based on labelled transition systems, where the intruder is described in the Dolev-Yao style. Security properties can be expressed in a linear time temporal logic.

Here types are used to obtain finite-state labelled transition systems. Types are assigned in a flexible manner, by user-defined transformation functions $\mathcal{C}_{\mathcal{S}_Z}$; these are applied to a given protocol specification Z to obtain a typed version in $\mathcal{C}_{\mathcal{S}_Z}(Z)$, before running the model checker.

A transformation $\mathcal{C}_{\mathcal{S}_Z}$ is a "view" that a user can introduce to select a finite partition of the state space. Different transformations can be used to select different portions of the state space, increasing the confidence of the analysis.

As theoretical result we have proved that given a formula f, an attack (that is a trace over which f is not satisfied) over a transformed protocol $\mathcal{C}_{\mathcal{S}_Z}(Z)$ always implies the existence of the same attack over the original protocol Z. Obviously finding an attack via protocol transformations is only a sound method, *i.e.*, if an attack exists over a protocol Z we have no guarantee that the attack can be found over a specific transformed protocol. Anyway, we prove that a transformation $\mathcal{C}'_{\mathcal{S}_Z}$ that preserves the attack always exists. This mean that our transformations are general enough to maintain type flaws.



LEGEND

$\overline{\lambda_1} = 1.\overline{c_{as}}\langle\langle A,\ B\rangle\rangle$

$\lambda_2 = 3.c_{as}\langle\langle A,\ B\rangle\rangle$

$\overline{\lambda_4} = 3.\overline{c_{as}}\langle\langle\{K\}_{K_{ab}},\ \{\langle A,\ K\rangle\}_{K_{bs}}\rangle\rangle$

$\lambda_6 = 1.c_{as}\langle\langle\{K\}_{K_{as}},\ \{\langle A,\ K\rangle\}_{K_{bs}}\rangle\rangle$

$\lambda_5 = 2.c_{ab}\langle\{\langle A,\ K\rangle\}_{K_{bs}}\rangle$

$\overline{\lambda_7} = 1.\overline{c_{ab}}\langle\{\langle A,\ K\rangle\}_{K_{bs}}\rangle$

Figure 4.6: The finite state labelled transition system which models $\mathcal{C}_{\mathcal{S}_{NSSK}}(NSSK)$.

q	$\mathcal{G}$	$\tilde{Z}$
q_0	$\mathcal{G}_0 = \begin{array}{l}(\Omega, \emptyset, \bot)\\ (p_A, W_1, \bot)\\ (p_B, W_2, \bot)\\ (p_S, W_3, \bot)\end{array}$	$\tilde{Z}_0 = \begin{array}{l}(\backslash K_{as})(\backslash K_{bs})(\backslash A)(\backslash B)(\backslash S)\\ (1, p_A\langle A, B, S, K_{as}\rangle)\\ \Vert(2, p_B\langle A, B, S, K_{bs}\rangle)\\ \Vert(3, p_S\langle A, B, K_{as}, K_{bs}\rangle)\end{array}$
q_1	$\mathcal{G}_1 = \begin{array}{l}(\Omega, \overbrace{\{\langle A,\ B\rangle\}}^{W_\Omega}, \bot)\\ (p_A, W_1, \bot)\\ (p_B, W_2, \bot)\\ (p_S, W_3, \bot)\end{array}$	$\tilde{Z}_1 = \begin{array}{l}(1, c_{as}(x)[x \texttt{ is } \langle x_1,\ x_2\rangle]\overline{c_{ab}}(x_2).0)\\ \Vert(2, p_B\langle A, B, S, K_{bs}\rangle)\\ \Vert(3, p_S\langle A, B, K_{as}, K_{bs}\rangle)\end{array}$
q_2	$\mathcal{G}_2 = \begin{array}{l}(\Omega, \langle A,\ B\rangle, \bot)\\ (p_A, W_1, \bot)\\ (p_B, W_2, \bot)\\ (p_S, \underbrace{W_3 \cup \{\langle A,\ B\rangle\}}_{W_3'},\\ \underbrace{[z \leftarrow \langle A,\ B\rangle]}_{\sigma_3'})\end{array}$	$\tilde{Z}_2 = \begin{array}{l}(1, c_{as}(x).[x \texttt{ is } \langle x_1,\ x_2\rangle]\overline{c_{ab}}(x_2).0)\\ \Vert(2, p_B\langle A, B, S, K_{bs}\rangle)\\ \Vert\ \begin{array}{l}(3,\\ [z \texttt{ is } \langle A,\ B\rangle].(\nu K_{ab})\\ \overline{c_{as}}(\langle\{K_{ab}\}_{K_{as}},\\ \{\langle A, K_{ab}\rangle\}_{K_{bs}}\rangle).0)\end{array}\end{array}$
q_2'	$\mathcal{G}_2' = \mathcal{G}_2$	$\tilde{Z}_2' = \begin{array}{l}(1, c_{as}(x)[x \texttt{ is } \langle x_1,\ x_2\rangle]\overline{c_{ab}}(x_2).0)\\ \Vert(2, p_B\langle A, B, S, K_{bs}\rangle)\\ \Vert\ \begin{array}{l}(3,\\ (\nu K_{ab})\\ \overline{c_{as}}(\langle\{K_{ab}\}_{K_{as}},\\ \{\langle A, K_{ab}\rangle\}_{K_{bs}}\rangle).0)\end{array}\end{array}$
q_3	$\mathcal{G}_3 = \begin{array}{l}(\Omega, W_\Omega, \bot)\\ (p_A, W_1, \bot)\\ (p_B, W_2, \bot)\\ (p_S, \underbrace{W_3' \cup \{K\}}_{W_3''}, \sigma_3')\end{array}$	$\tilde{Z}_3 = \begin{array}{l}(1, c_{as}(x)[x \texttt{ is } \langle x_1,\ x_2\rangle]\overline{c_{ab}}(x_2).0)\\ \Vert(2, p_B\langle A, B, S, K_{bs}\rangle)\\ \Vert(3, \overline{c_{as}}(\langle\{K\}_{k_{as}}, \{\langle A, K\rangle\}_{k_{bs}}\rangle).0)\\ \text{where } K \text{ is a new name}\end{array}$
q_4	$\mathcal{G}_4 = \begin{array}{l}(\Omega, \overbrace{\begin{array}{l}W_\Omega \cup \{\langle\{K\}_{K_{as}},\\ \{\langle A, K\rangle\}_{K_{bs}}\rangle\}\end{array}}^{W_\Omega'}, \bot)\\ (p_A, W_1, \bot)\\ (p_B, W_2, \bot)\\ (p_S, W_3'', \sigma_3')\end{array}$	$\tilde{Z}_4 = \begin{array}{l}(1, c_{as}(x)[x \texttt{ is } \langle x_1,\ x_2\rangle]\overline{c_{ab}}(x_2).0)\\ \Vert(2, c_{ab}(y).[y \texttt{ is } \{\langle A,\ y_1\rangle\}_{K_{bs}}].0)\\ \Vert(3, 0)\end{array}$

Figure 4.7: Details of the LTS of the protocol $\mathcal{C}_{\mathcal{S}_{NSSK}}(NSSK)$ (part A). In the first column we list the name of the states, in the second and third column their contents *i.e.*, the global states and suffixes of the *spy-calculus* protocol specification (where we omitted types for sake of saving space). In the initial state $W_1 = \{A, B, S, K_{as}\}$, $W_2 = \{A, B, S, K_{bs}\}$ and $W_3 = \{A, B, S, K_{as}, K_{bs}\}$

q	$\mathcal{G}$	$\tilde{Z}$
q_5	$\mathcal{G}_5 = (\Omega, W'_\Omega, \bot)$ $(p_A, W_1, \bot)$ $(p_B, \overbrace{W_2 \cup \{\{\langle A,\, K\rangle\}_{K_{bs}}\}}^{W'_2}, \underbrace{[y \leftarrow \{\langle A,\, K\rangle\}_{K_{bs}}]}_{\sigma_2})$ (p_S, W''_3, σ'_3)	$\tilde{Z}_5 = (1, c_{as}(x).$ $[x \texttt{ is } \langle x_1,\, x_2\rangle]$ $\overline{c_{ab}}(x_2).0)$ $\|(2, \{\langle A,\, K\rangle\}_{K_{bs}} \texttt{is}$ $\{\langle a,\, y_1\rangle\}_{K_{bs}}.0)$ $\|(3, 0)$
q'_5	$\mathcal{G}'_5 = (\Omega, W'_\Omega, \bot)$ $(p_A, W_1, \bot)$ $(p_B, W'_2, \underbrace{\sigma_2 \cup [y_1 \leftarrow K]}_{\sigma'_2})$ (p_S, W''_3, σ'_3)	$\tilde{Z}'_5 = (1, c_{as}(x).$ $[x \texttt{ is } \langle x_1,\, x_2\rangle]$ $\overline{c_{ab}}(x_2).0)$ $\|(2, 0)$ $\|(3, 0)$
q_6	$\mathcal{G}_6 = (\Omega, W'_\Omega, \bot)$ $(p_A,$ $\underbrace{\overbrace{W_1 \cup \langle\{K\}_{K_{as}}, \{\langle A, K\rangle\}_{K_{bs}}\rangle}^{W'_1},}_{\sigma_1}$ $\underbrace{[x \leftarrow \langle\{K\}_{K_{as}}, \{\langle A, K\rangle\}_{K_{bs}}\rangle]}_{\sigma_1})$ (p_B, W'_2, σ'_2) (p_S, W''_3, σ'_3)	$\tilde{Z}_6 = (1, [x \texttt{ is } \langle x_1,\, x_2\rangle]\overline{c_{ab}}(x_2).0)$ $\|(2, 0)$ $\|(3, 0)$
q'_6	$\mathcal{G}'_6 = (\Omega, W'_\Omega, \bot)$ $(A, W'_1, \underbrace{\sigma_1 \cup [x_1 \leftarrow K,\ x_2 \leftarrow \{\langle A,\, K\rangle\}_{K_{bs}}]}_{\sigma'_1})$ (B, W'_2, σ'_2) (S, W''_3, σ'_3)	$\tilde{Z}'_6 = (1, \overline{c_{ab}}(x_2).0)$ $\|(2, 0)$ $\|(3, 0)$
q_7	$\mathcal{G}_7 = (\Omega, W'_\Omega, \bot)$ (p_A, W'_1, σ'_1) (p_B, W'_2, σ'_2) (p_S, W''_3, σ'_3)	$\tilde{Z}_7 = (1, 0)$ $\|(2, 0)$ $\|(3, 0)$

Figure 4.8: Details of the LTS of the protocol $\mathcal{C}_{\mathcal{S}_{NSSK}}(NSSK)$ (continued).

In our framework we do not have any automatic method to find a typing transformation catching an attack without knowing the attack first. The formal proof of completeness, where this transformation is proved to exist, is not constructive and it does not help in this direction. At the present version of the tool finding the right typing transformation depends on the experience of the engineer that performs the verification. We conjecture that useful hints in defining a significant typing transformation can emerge from a static analysis of the message flow along a protocol specification. This conjecture is supported by those results that show that it is possible to use type checking to check secrecy and authenticity (*e.g.*, see [3, 102]). Type checking for security protocol is not commonly used, and the large availability of dynamic verifier confirms this impression. We claim that static type checking can be profitably integrated in our tool with dynamic analysis: a static (even partial) check can be used as front-end to built a set of typing transformations then used in the dynamic model checker module. We have not investigated this solution yet, and we point out it as a future research.

Our SPYDER implementation runs in exponential time in the size of the longest message involved in the protocol. This matches the expected theoretical computational complexity, so it is the best we can expect. We think that the SPYDER performance can be significantly improved by the use of partial order reduction techniques that may help in reducing even more the size of protocol models (this is evident, for example, just looking at the model in Figure 4.6). In fact, the use of partial order reduction has been applied with success in the model checker BRUTUS [52] whose logic (and relative semantical models) have inspired the one we have used here. We leave as a future work the porting of such techniques to SPYDER.

Part III

Comparison of Formal Models in Security Protocol Analysis

5

Relating Multiset Rewriting and Process Algebras for Security Protocol Analysis

"Quando leggemmo il disïato riso esser basciato da cotanto amante, questi, che mai da me non fia diviso, la bocca mi basciò tutto tremante. Galeotto fu'l libro e chi lo scrisse: quel giorno più non vi leggemmo avante." (Francesca in La Divina Commedia – Inferno Canto V, *Dante Alighieri)*

"When we had read how the desired smile was kissed by one who was so true a lover, this one, who never shall be parted from me, while all his body trembled, kissed my mouth. A Gallehault indeed, that book and he who wrote it, too; that day we read no more."

Abstract

When analysing security protocols, different specification languages support very different reasoning methodologies whose results are not directly or easily comparable. Therefore, establishing clear mappings among different frameworks is highly desirable, as it permits various methodologies to cooperate by interpreting theoretical and practical results of one system into another. In this chapter, we examine the relationship between two general verification frameworks: multiset rewriting (MSR) and a process algebra (PA) inspired by CCS and the π-calculus. Although defining a simple and general bijection between MSR and PA appears difficult, we show that the sub-languages needed to specify cryptographic protocols admit an effective translation that is not only trace-preserving, but also induces a correspondence relation between the two languages. In particular, the correspondence sketched in this chapter permits transferring several important trace-based properties such as secrecy and many forms of authentication.

5.1 Introduction

In the last decade, security-related problems have attracted the attention of many researchers from several different communities, especially formal methods (*e.g.*, [4, 6, 40, 50, 43, 66, 79, 84, 82, 102, 150, 182, 77]). These researchers have often let their investigation be guided by the techniques and experiences specific to their own areas of knowledge. This massive interest has determined a plethora of results that often are not directly comparable or integrable with one another. In the last few years, attempts have been made to unify frameworks for specifying security properties often expressed in different ways [86], and to study the relationships between different models for representing security protocols [44].

In this chapter, we relate a transition-based and a form of process-based models for the description and the analysis of a large class of security protocols. We choose the multiset-rewriting for-

malism MSR as a representative of the former, and synthesize salient features of popular process algebras in a system that we call PA as an abstraction of the latter.

MSR, with its roots in concurrency theory and rewriting logic, has proved to be a language suitable for studying foundational issues in security protocols [43]. It is also playing a practical role through the closely related CIL intermediate language [66] of the CASPL security protocol analysis system [65], in particular since translators from several tools to CIL have recently been developed. Ties between between MSR and strand spaces [76], a popular specification language for crypto-protocols, were analyzed by Cervesato *et al* in [44].

Process algebra encompasses a family of well-known formal frameworks proposed to describe features of distributed and concurrent systems. Here we use a PA that borrows concepts from different calculi, specifically CCS [158] and the π-calculus [158]. We expect our results to be applicable to other (value passing) process algebras used for security protocol analysis, *e.g.*, the spi-calculus [5] or CSP [181]. Indeed, when applied to security protocol analysis, most such languages rely only on a well-identified subset of primitives, that we have isolated in the language considered here.

We relate MSR and PA by defining *encodings* from one formalism to the other. Moreover we propose a *correspondence relation* between MSR and PA protocol models, preserved by our encodings, that is sufficient to transfer several useful trace-based properties such as secrecy and many forms of authentication. Informally, this relation says that an MSR configuration and a PA process correspond if and only if the messages stored on the network and the messages known by the intruder are the same, step by step, in the two models.

Consequences of the results in this chapter are:

- First, our encodings establish a relationship between the *specification methodologies* underlying MSR and PA. MSR is a representation paradigm based on transitions between explicit states, as found, for example, in the vast majority of tools for security protocol analysis [43, 50, 65, 70, 150, 172, 181]. The approach underlying PA and the languages behind it, *e.g.*, [5, 29, 82, 102, 77], represents concurrent systems, with security protocols as a particular instance, as independent threads of computation communicating by message passing. While specifications are obviously related, moving between paradigms is an error-prone process unless guided by formal encodings.

- Second, the relationship we developed helps at relating verification results obtainable in each model, in particular as far as secrecy and authentication are concerned. Systems *à la* MSR overwhelmingly embrace a verification methodology based on some form of trace exploration: model-checking [50, 65, 70, 181], theorem proving [172], or a combination [150]. The situation is more complex in process-algebraic languages, in which the analysis can be based on traces [29, 82, 186], but also on process equivalence [5], type-checking [102] and other forms of symbolic reasoning [110]. While we do not study how these last three forms of analysis map to in the MSR world, we believe that the present study opens the door to such an investigation. Authentication and secrecy are quintessential trace-based safety properties (they are expressed in terms of intruder knowledge and messages passed onto the network and our encodings preserve this information). Therefore relating trace-based results in MSR and PA is valuable, in particular as these languages rely on different notions of traces, and sometimes make different uses of them, *e.g.*, [82].

- Finally, by bridging PA and MSR, we implicitly define a correspondence between PA and other languages for security analysis. MSR has already been related to other formalisms,

such as strand spaces [76] in a setting with an interleaving semantics (a worthy investigation as remarked in [62]), while work on linear logic and MSR appears in [157].

5.2 Background

In this section, we recall the syntax and formal semantics of multiset rewriting (MSR) and we define the language, PA, that we will use as a representative of process algebras. Before doing so, we present our notation for tuples, as both MSR and PA rely on these objects. A *tuple* is defined by the following grammar:

$$\boldsymbol{t} ::= \epsilon \mid t; \boldsymbol{t}$$

A tuple $\boldsymbol{t}$ is a sequence of items. We use the semicolon (";") as the tuple constructor: it is associative but not commutative. We write ϵ for the empty tuple, which acts as the left and right identity of ";". We write $t \in \boldsymbol{t}$ to indicate that item t is present in tuple $\boldsymbol{t}$, and use the notation $\boldsymbol{t}' \sqsubseteq \boldsymbol{t}$ to indicate that $\boldsymbol{t}'$ is a subsequence of $\boldsymbol{t}$, *i.e.*, that $\boldsymbol{t}'$ can be obtained by deleting zero or more symbols from $\boldsymbol{t}$. Finally, given tuples $\boldsymbol{t}$ and $\boldsymbol{t}'$ with $\boldsymbol{t}' \sqsubseteq \boldsymbol{t}$, we write $\boldsymbol{t} - \boldsymbol{t}'$ for the tuple obtained by filtering out all items $t' \in \boldsymbol{t}'$ from $\boldsymbol{t}$, while preserving the order of the remaining elements of the latter.

5.2.1 First Order Multiset Rewriting

The language of first-order MSR is defined by the following grammar:

Elements	$\tilde{a}$	$::=$	$\cdot \mid a(\boldsymbol{t}), \tilde{a}$
Rewriting Rules	r	$::=$	$\tilde{a}(\boldsymbol{x}) \to \exists \boldsymbol{n}.\tilde{b}(\boldsymbol{x}; \boldsymbol{n})$
Rule sets	$\tilde{r}$	$::=$	$\cdot \mid r, \tilde{r}$

Multiset elements are chosen as atomic formulas $a(\boldsymbol{t})$, where $\boldsymbol{t}$ is a tuple of terms over some first-order signature Σ. We write $\tilde{a}(\boldsymbol{x})$ to emphasize that variables, drawn from $\boldsymbol{x}$, appear in a multiset $\tilde{a}$. Similarly we write t (respectively $\boldsymbol{t}$) as $t(\boldsymbol{x})$ (respectively $\boldsymbol{t}(\boldsymbol{x})$), to underline that varibles $\boldsymbol{x}$ appear in a term t (respectively in the tuple of terms $\boldsymbol{t}$). Instead, we write $\underline{t}$ (respectively $\underline{\boldsymbol{t}}$) to emphasize, when required, that a term t is (respectively all the term in $\boldsymbol{t}$ are) ground, *i.e.*, variable-free.

In the sequel, the comma "," will denote multiset union and will implicitly be considered commutative and associative, while "$\cdot$", the empty multiset, will act as a neutral element; we will omit it when convenient. The operational semantics of MSR is expressed by the following two judgments:

Single rule application	$\tilde{r} : \tilde{a} \longrightarrow \tilde{b}$
Iterated rule application	$\tilde{r} : \tilde{a} \longrightarrow^* \tilde{b}$

The multisets $\tilde{a}$ and $\tilde{b}$ are called *states* and are always ground formulas. The arrow represents a transition. These judgments are defined as follows:

$$\frac{}{(\tilde{r}, \tilde{a}(\boldsymbol{x}) \to \exists \boldsymbol{n}.\tilde{b}(\boldsymbol{x}; \boldsymbol{n})) : (\tilde{c}, \tilde{a}[\underline{\boldsymbol{t}}/\boldsymbol{x}]) \longrightarrow (\tilde{c}, \tilde{b}[\underline{\boldsymbol{t}}/\boldsymbol{x}, \underline{\boldsymbol{k}}/\boldsymbol{n}])}\ \mathbf{msr_0}$$

$$\frac{}{\tilde{r} : \tilde{a} \longrightarrow^* \tilde{a}}\ \mathbf{msr_*} \qquad \frac{\tilde{r} : \tilde{a} \longrightarrow \tilde{b} \qquad \tilde{r} : \tilde{b} \longrightarrow^* \tilde{c}}{\tilde{r} : \tilde{a} \longrightarrow^* \tilde{c}}\ \mathbf{msr_1}$$

The first inference shows how a rewrite rule $r = \tilde{a}(\boldsymbol{x}) \to \exists \boldsymbol{n}.\tilde{b}(\boldsymbol{x}; \boldsymbol{n})$ is used to transform a state into a successor state: it identifies a ground instance $\tilde{a}(\underline{\boldsymbol{t}})$ of its antecedent and replaces it

with the ground instance $\tilde{b}(\underline{\boldsymbol{t}}; \underline{\boldsymbol{k}})$ of its consequent, where $\underline{\boldsymbol{k}}$ are fresh constants. Here $[\underline{\boldsymbol{t}}/\boldsymbol{x}]$ denotes the substitution (also written θ) replacing every occurrence of a variable x among $\boldsymbol{x}$ with the corresponding term t in $\boldsymbol{t}$. These rules implement a non-deterministic but sequential computation model. This means that in general several rules are applicable at any step but only one rule, chosen non-deterministically among them, is applied at each step. Concurrency is captured as the permutability of (some) rule applications. The remaining rules define $\longrightarrow^*$ as the reflexive and transitive closure of $\longrightarrow$.

5.2.2 Process Algebras

Process algebraic specifications of security protocols are generally limited to the parallel composition of a number of processes describing the sequence of actions performed by each agent. With this in mind, we forsake the full treatment of a traditional process algebra, such as the π-calculus, in favor of a more specific language, PA, that includes the features commonly used for describing cryptographic protocols. In particular, we lay out PA on two levels: *sequential processes* describe the sequence of atomic actions (input, output, name generation, etc.) performed by an individual agent and *parallel processes* bundle them into a multi-agent specifications. Sequential processes are synchronous, although a systematic use of buffer processes will prevent the possibility of blocking on an output action. For convenience, we will rely on polyadic communication channels.

With these premises, the language of PA is defined by the following grammar:

$$\begin{array}{lrcl} \textit{Parallel processes} & Q & ::= & 0 \mid Q \parallel P \mid Q \parallel !P \\ \textit{Sequential processes} & P & ::= & 0 \mid \overline{a}(\boldsymbol{t}).P \mid a(\boldsymbol{x}).P \mid [\boldsymbol{x} = \boldsymbol{t}]\, P \mid \nu x.P \end{array}$$

Parallel processes are defined as a parallel composition of – possibly replicated – sequential processes. These, in turn, are a sequence of communication actions (input or output), pattern matching and constant generation. An output process $\overline{a}(\boldsymbol{t}).P$ is ready to send a tuple of terms $\boldsymbol{t}$, each built over a signature Σ, along the polyadic channel named a. An input process $a(\boldsymbol{x}).P$ is ready to receive a tuple of (ground) messages, each in the corresponding variable $x \in \boldsymbol{x}$. The process $[\boldsymbol{x} = \boldsymbol{t}]\, P$ is a parallel pattern matching construct which forces any instantiation of $\boldsymbol{x}$ to match the pattern $\boldsymbol{t}$, possibly binding previously unbound variables in the latter. Finally, the creation of a new object in P (as in the π-calculus [162]) is written as $\nu x.P$ (we will sometimes abbreviate $\nu x_1 \ldots \nu x_n.P$ as $\nu \boldsymbol{x}.P$). The binders of our language are $\nu \boldsymbol{x}$, $a(\boldsymbol{x})$ which bind each x in $\boldsymbol{x}$, and $[\boldsymbol{x} = \boldsymbol{t}]$ which binds any first occurrence of a variable in $\boldsymbol{t}$. This induces the usual definition of free and bound variables in a term or process.

The operational semantics of PA is given by the following judgments:

$$\begin{array}{ll} \textit{Single interaction} & Q \Rightarrow Q' \\ \textit{Iterated interaction} & Q \Rightarrow^* Q' \end{array}$$

They are defined as follows:

$$\frac{}{(Q \parallel \overline{a}(\underline{\boldsymbol{t}}).P \parallel a(\boldsymbol{x}).P') \Rightarrow (Q \parallel P \parallel P'[\underline{\boldsymbol{t}}/\boldsymbol{x}])}\ \mathbf{pa_0}$$

$$\frac{\underline{\boldsymbol{t}} = \boldsymbol{t}'[\theta]}{(Q \parallel [\underline{\boldsymbol{t}} = \boldsymbol{t}']\, P) \Rightarrow (Q \parallel P[\theta])}\ \mathbf{pa_{[]}} \qquad \frac{\underline{k} \notin c(Q) \cup c(P)}{(Q \parallel \nu x.P) \Rightarrow (Q \parallel P[\underline{k}/x])}\ \mathbf{pa_\nu}$$

$$\frac{P \equiv P' \quad P' \Rightarrow Q' \quad Q' \equiv Q}{P \Rightarrow Q}\ \mathbf{pa}_{\equiv} \qquad \frac{}{Q \Rightarrow^* Q}\ \mathbf{pa}_{*} \qquad \frac{Q \Rightarrow Q'' \quad Q'' \Rightarrow^* Q'}{Q \Rightarrow^* Q'}\ \mathbf{pa_1}$$

The first inference (*reaction*) shows how two sequential processes, respectively one ready to perform an output of a tuple $\underline{\boldsymbol{t}}$ of ground terms, and one ready to perform an input over $\boldsymbol{x}$ react by applying the instantiating substitution $[\underline{\boldsymbol{t}}/\boldsymbol{x}]$ to P'. The second inference rule (*matching*) says that there must exist a substitution θ that matches terms $\boldsymbol{t'}$ with ground terms $\underline{\boldsymbol{t}}$, for $[\underline{\boldsymbol{t}} = \boldsymbol{t'}]\, P$ to evolve into $P[\theta]$. The third rule defines the semantics of νx as an instantiation with a fresh constant *i.e.*, a name which differs form those appearing in all the process terms (here $c(P)$ denotes the set of constant in P). The next rule allows interactions to happen modulo structural equivalence $\equiv$, that in our case contains the usual monoidal equalities of parallel processes with respect to $\|$ and 0, the unfolding of replication (*i.e.*, $!P \equiv !P \parallel P$), and the equation $[\boldsymbol{t} = \boldsymbol{t'}]\, P \equiv [\boldsymbol{t}^* = \boldsymbol{t'}^*]\, P$ which filter out identities in tuple's matching, *i.e.*, where $\boldsymbol{t}^*$ and $\boldsymbol{t'}^*$ are obtained from $\boldsymbol{t}$ and $\boldsymbol{t'}$ by removing all identical items in corresponding positions in a patter matching over tuples.

Finally, the last two inferences define $\Rightarrow^*$ as the reflexive and transitive closure of $\Rightarrow$.

5.3 Security Protocols

A cryptographic protocol is a collection of distributed programs supporting communication between participating agents and aimed at achieving predetermined security outcomes such as secrecy or authentication. The agents communicating in a protocol are called *principals*, while the individual programs they execute as part of the protocol are called *roles*. Communication happens through a public *network* and is therefore accessible to anyone, unless protected through cryptography.

Both transition- and process-based languages have been widely used for the specification of cryptographic protocols (see for example [4, 6, 50, 43, 66, 79, 84, 82, 102, 150, 182, 77]). In this section, we define MSR_P and PA_P, two security-oriented instances of MSR and PA respectively, and describe how they can be used to specify security protocols.

Narrowing our investigation to a specific domain allows us to compare directly these restricted versions of PA and MSR. Moreover by restricting our analysis to cryptographic protocols, we are able to obtain stronger correspondence results than what seems achievable in a general comparison between PA and MSR[25].

The two specifications will rely on a common first-order signature Σ_P that includes at least concatenation ($\langle \_, \_ \rangle$) and encryption ($\{\_\}_\_$). In both formalisms, terms in Σ_P stand for messages. Predicate symbols are interpreted as such in MSR_P, and as channel names in PA_P. Variables will also be allowed in rules and processes.

5.3.1 Formalizing Protocols as Multiset Rewriting

MSR_P relies on the following predicate symbols [44]:

Network Messages ($\tilde{N}$): are the predicates used to model the network, where $N(t)$ means that the term t is stored in the network.

Role States ($\tilde{A}$): are the predicates used to model roles. Assuming a set of *role identifiers* R, the family of *role state predicates* $\{A_{\rho_i}(\boldsymbol{t}) : i = 0 \ldots l_\rho\}$, is intended to hold in the internal state, $\boldsymbol{t}$, of a principal in role $\rho \in R$ during the sequence of protocol steps $i = 0 \ldots l_\rho$. The behavior of each role ρ is described through a finite number of rules, indexed from 0 to l_ρ.

Intruder ($\tilde{I}$): are the predicates used to model the intruder I, where $I(t)$, means that the intruder knows the message t.

Persistent Predicates ($\tilde{\pi}$): are ground predicates holding data that does not change during the unfolding of the protocol (*e.g.,* $\mathtt{Kp}(K; K')$ indicates that K and K' form a pair of public/private keys). Rules use these predicates in a read-only manner to access the value of persistent data.

A security protocol is expressed in MSR_P as a set of rewrite rules $\tilde{r}$ of a specific format called a *security protocol theory*. Given roles R, it can be partitioned as $\tilde{r} = \cup_{\rho \in R}(\tilde{r}_\rho), \tilde{r}_I$, where $\tilde{r}_\rho$ and $\tilde{r}_I$ describe the behavior of a role $\rho \in R$ and of the intruder I. For each role ρ, the rules in $\tilde{r}_\rho$ consist of:

- one *initial rule*

$$\textit{instantiation} \quad r_{\rho_0}: \ \tilde{\pi}(\boldsymbol{x}) \ \rightarrow \ \exists \boldsymbol{n}.A_{\rho_0}(\boldsymbol{x}; \boldsymbol{n}), \tilde{\pi}(\boldsymbol{x})$$

- zero or more ($i = 1 \ldots l_\rho$) *message exchange rules*:

$$\begin{array}{lrcl}
\textit{send} & r_{\rho_i}: & A_{\rho_{i-1}}(\boldsymbol{x}) & \rightarrow \ A_{\rho_i}(\boldsymbol{x}), N(t(\boldsymbol{x})) \\
\textit{receive} & r_{\rho_i}: & A_{\rho_{i-1}}(\boldsymbol{x}), N(y) & \rightarrow \ A_{\rho_i}(\boldsymbol{x}; y) \\
\textit{analysis} & r_{\rho_i}: & A_{\rho_{i-1}}(\boldsymbol{t}(\boldsymbol{x})) & \rightarrow \ A_{\rho_i}(\boldsymbol{x})
\end{array}$$

The first rule (*instantiation*) describes the instantiation step of a protocol role. All the new names required in a role ρ are generated during instantiation, and similarly all the variables $\boldsymbol{x}$ referring to permanent data $\tilde{\pi}(\underline{\boldsymbol{t}})$ are bound to ground permanent terms in that rule. The second rule (*send*) describes an action of sending a message $\underline{t}$ composed by using (all or a subset of) the ground terms in the role's state. The third rule (*receive*) describes a receive operation, where a message $\underline{t}$ stored in the net is retrieved, bound to variable y and then stored into the internal state of the role. The last rule (*analysis*) simulates the action of a role when it analyses (*e.g.,* decrypts or splits) previously received messages.

This fairly explicit formulation of MSR rules will simplify our comparison with PA_P. Equivalent, but more succinct, formulations can be found in [43, 42].

Rules in $\tilde{r}_I$ are the standard rules describing the intruder in the style of Dolev-Yao [69], whose capabilities consist in intercepting, analyzing, synthesizing and constructing messages, with the ability to access some permanent data. Formally:

$$\begin{array}{lrcl}
r_{I_1}: & \pi(x) & \rightarrow & I(x), \pi(x) \\
r_{I_2}: & \cdot & \rightarrow & \exists n.I(n) \\
r_{I_3}: & N(x) & \rightarrow & I(x) \\
r_{I_4}: & I(x) & \rightarrow & N(x), I(x) \\
r_{I_5}: & I(\langle x_1, x_2 \rangle) & \rightarrow & I(x_1), I(x_2), I(\langle x_1, x_2 \rangle) \\
r_{I_6}: & I(x_1), I(x_2) & \rightarrow & I(\langle x_1, x_2 \rangle), I(x_1), I(x_2) \\
r_{I_7}: & I(\{x\}_k), I(k), \mathtt{Kp}(k; k') & \rightarrow & I(x), \mathtt{Kp}(k; k'), I(\{x\}_k), I(k) \\
r_{I_8}: & I(x), I(k) & \rightarrow & I(\{x\}_k), I(x), I(k) \\
r_{I_9}: & I(x) & \rightarrow & \cdot
\end{array}$$

where x, x_i's and k are variables. Informally, the first rule allows the intruder to access (*i.e.,* get knowledge of) persistent data. In the second, rule the intruder creates a new ground datum. In

the third, a message stored in the network is intercepted, while in the fourth a known message is injected into the network channel. The remaining rules describe the intruder capabilities for managing the messages it knows: more precisely its ability to decompose pairs, to compose pairs, to decrypt a message (if the relative decryption key is known), and to create encrypted messages. Finally, the last one describes the capability of the intruder in deleting messages (*i.e.,* forgetting knowledge).

In MSR_P, a state is a multiset of the form $\tilde{s} = (\tilde{N}, \tilde{A}, \tilde{I}, \tilde{\pi})$, where the components collect ground facts of the form $N(t)$, $A_{\rho_i}(\boldsymbol{t})$, $I(t)$, and $\pi(\boldsymbol{t})$ respectively. An *initial state* $\tilde{s}_0 = (\tilde{I}_0, \tilde{\pi})$ contains only the initial intruder knowledge ($\tilde{I}_0$) and persistent predicates ($\tilde{\pi}$). Note that $\tilde{\pi}$ remains the same in every state. A pair $(\tilde{r} : \tilde{s})$ consisting of a protocol theory $\tilde{r}$ and a state $\tilde{s}$ is called a *configuration*. The initial configuration is $(\tilde{r} : \tilde{s}_0)$.

Example 5.3.1 We make these definitions more concrete by showing the MSR_P representation of the classical Needham-Schroeder Public Key ($NSPK$) protocol [167]. In the common informal notation, it is written as follows:

$$\begin{array}{lcl} 1.\ A & \longrightarrow & B : \{A, N_A\}_{K_B} \\ 2.\ B & \longrightarrow & A : \{N_A, N_B\}_{K_A} \\ 3.\ A & \longrightarrow & B : \{N_B\}_{K_B} \end{array} \tag{5.3.1}$$

The abstract principal A and the role it executes are called the *initiator* since it originates the first message. Dually, B is the *responder*. This first message, $\{A, N_A\}_{K_B}$, consists of A's name and a freshly generated random value N_A (a nonce), and is encrypted using B's public key K_B. Upon successfully decrypting this message (using private key K_B^{-1}), B replies with the second message, $\{N_A, N_B\}_{K_A}$, where N_B is a second nonce, generated by B. Upon successfully processing this message, A sends the final message $\{N_B\}_{K_B}$ which shall be interpreted by B.

Here, A and B perform distinct although related sequences of actions: A generates N_A, sends $\{A, N_A\}_{K_B}$, waits for a message from B and verifies that it matches the format $\{N_A, N_B\}_{K_A}$, and finally sends the third message, $\{N_B\}_{K_B}$. This sequence of actions constitute A's role. B's role is similar. Both MSR_P and PA_P give a role-centric representation of a protocol.

The MSR_P specification of the $NSPK$ protocol consists of the rule-set $\mathcal{R}_{\mathrm{NSPK}}$ which we partition as $(\mathcal{R}_A, \mathcal{R}_B, \tilde{r}_I)$. $\mathcal{R}_A$ and $\mathcal{R}_B$ implement the roles of the initiator (A) and the responder (B) respectively, while $\tilde{r}_I$ describes the actions of a potential attacker, and have been fixed earlier in the discussion.

First some abbreviations. We define

$$\tilde{\pi}(x; y; k_x; k'_x, k_y) \;=\; \mathtt{Pr}(x), \mathtt{PrK}(x; k'_x), \mathtt{PbK}(y; k_y), \mathtt{Kp}(k_x; k'_x)$$

Here, persistent predicate $\mathtt{Pr}(z)$ indicates that z is the name of a principal; the predicate $\mathtt{PbK}(z; k_z)$ defines k_z to be the public key of principal x; the predicate $\mathtt{PrK}(z; k'_z)$ says that k'_z is z's private key; finally, $\mathtt{Kp}(k_z; k'_z)$ relates a public key k_z and the corresponding private key k'_z. Two 5-tuples of variables $(a; b; k_a; k'_a; k_b)$ and $(b; a; k_b; k'_b; k_a)$ will occur repeatedly in this example; therefore we shall abbreviate them as $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively.

Then, the following rules describe A's role:

$$\mathcal{R}_A \left\{ \begin{array}{ll} r_{A_0} : \tilde{\pi}(\boldsymbol{A}) & \rightarrow \exists n_a.\tilde{\pi}(\boldsymbol{A}), A_0(\boldsymbol{A}; n_a) \\ r_{A_1} : \ A_0(\boldsymbol{A}; n_a) & \rightarrow N(\{a, n_a\}_{k_b}), A_1(\boldsymbol{A}; n_a) \\ r_{A_2} : \ A_1(\boldsymbol{A}; n_a), N(m) & \rightarrow A_2(\boldsymbol{A}; n_a; m) \\ r_{A_3} : \ A_2(\boldsymbol{A}; n_a; \{n_a, n_b\}_{k_a}) & \rightarrow A_3(\boldsymbol{A}; n_a; n_b) \\ r_{A_4} : \ A_3(\boldsymbol{A}; n_a; n_b) & \rightarrow N(\{n_b\}_{k_b}), A_4(\boldsymbol{A}; n_a; n_b) \end{array} \right.$$

The first rule r_{A_0} in $\mathcal{R}_A$ is the instantiation rule of this role, and takes care of generating the initiator's nonce, n_a and collecting the persistent information used in the role. Rules r_{A_1} and r_{A_4} are send rules corresponding to the message transmission step 1 and 3 in protocol (5.3.1). Rules r_{A_2} and r_{A_3} realize the initiator's actions in the second step of $NSPK$, namely the reception of a message m from b and the verification that it matches the expected pattern $\{n_a, n_b\}_{k_a}$. Reception and analysis are described as separated steps accordingly to the MSR_P syntax.

The responder's role is similarly specified by the following MSR_P rule set:

$$\mathcal{R}_B \left\{ \begin{array}{ll} r_{B_0} : \ \tilde{\pi}(\boldsymbol{B}) & \rightarrow \exists n_b.\tilde{\pi}(\boldsymbol{B}), B_0(\boldsymbol{B}; n_b) \\ r_{B_1} : \ B_0(\boldsymbol{B}; n_b), N(m) & \rightarrow B_1(\boldsymbol{B}; n_b; m) \\ r_{B_2} : \ B_1(\boldsymbol{B}; n_b; \{a, n_a\}_{k_b}), & \rightarrow B_2(\boldsymbol{B}; n_b; n_a) \\ r_{B_3} : \ B_2(\boldsymbol{B}; n_b; n_a) & \rightarrow N(\{n_a, n_b\}_{k_a}), B_3(\boldsymbol{B}; n_b; n_a) \\ r_{B_4} : \ B_3(\boldsymbol{B}; n_b; n_a), N(m') & \rightarrow B_4(\boldsymbol{B}; n_b; n_a; m') \\ r_{B_5} : \ B_4(\boldsymbol{B}; n_b; n_a; \{n_b\}_{k_b}) & \rightarrow B_5(\boldsymbol{B}; n_b; n_a) \end{array} \right.$$

Again, the instantiation rule r_{B_0} instantiate all the variables $\boldsymbol{B}$ to ground terms. Rules r_{B_1}, r_{B_4} model the receiving steps 1 and 3 in protocol (5.3.1), while r_{B_3} is the rule corresponding the sending step 2. Finally rules r_{B_2}, r_{B_5} describe the analysis steps performed by the role.

Finally, we define the state portion of the initial configuration (*i.e.*, the initial state) to consist of:

$$\underbrace{\underbrace{\underbrace{I(E), I(K_E), I(K'_E)}_{\tilde{I}}}_{\tilde{A}}}_{\tilde{N}}, \underbrace{\begin{array}{l} \tilde{\pi}(A; B; K_A; K'_A; K_B), \\ \tilde{\pi}(B; A; K_B; K'_B; K_A), \\ \tilde{\pi}(B; E; K_B; K'_B; K_E), \\ \tilde{\pi}(E; A; K_E; K'_E; K_A), \end{array}}_{\tilde{\pi}}$$

where A, B, E, are specific principals (a and b above were variables), with E acting as the attacker. For each of them, the pseudo-functions $K_{\_}$ and $K'_{\_}$ denote their public and private key, respectively.

In this initial state, the intruder knowledge consists of its name E and its public/private key pair K_E, K'_E. The persistent data $\tilde{\pi}$ defines the attributes (name, public and private key) of each of these principals, in particular of the intruder E who may participate in the protocol as an honest player if he wishes. This is useful, for example, when testing some authenticity property.

■

5.3.2 Protocols as Processes

A security protocol may be described in a fragment of PA where:

- Every communication happens through the net (here $P_{!net}$ is the process that manages the net as a public channel where protocol roles send and receive messages).

- There is an intruder, with some initial knowledge, able to intercept and forge messages passing through the net (here $Q_{!I}$, with initial knowledge Q_{I_0}).
- Each principal starts the protocol in a certain role ρ.

Formally a security protocol, involving a collection of roles $\{\rho\}$, is expressed in PA_P as a "*security protocol process* Q", defined as the parallel composition of five components: $P_{!net} \parallel \prod_\rho P_{!\rho} \parallel Q_{!I} \parallel Q_{!\pi} \parallel Q_{I_0}$ where $\prod \mathcal{P}$ denotes the parallel composition of all the processes in $\mathcal{P}$. More precisely:

$P_{!net} = !N_i(x).\overline{N_o}(x).0$ This process describes the behavior of the network as a buffer that copies messages from channel N_i (input to the net) to N_o (output from the net), implementing an asynchronous form of message transmission on top of a synchronous calculus.

$P_{!\rho}$ Each of these replicated sequential processes capture the actions that constitute a role, in the sense defined for MSR_P. These processes have the form

$$P_{!\rho} = !\tilde{\pi}(\boldsymbol{x}).\nu \boldsymbol{n}.P_\rho$$

Here P_ρ is a sequential process that performs input and output only on the network channels, and that analyses the received messages. Variables x and n are free in P_ρ.

Notice that pattern matching is sufficient for "extracting" a piece of information when Σ_P is used, but more general mechanisms could be considered (as in Crypto-CCS for example [86]).We have used $\tilde{\pi}(\boldsymbol{x}).P$ as a shortcut for $\pi_1(\boldsymbol{x}_1)\ldots\pi_k(\boldsymbol{x}_k).P$, where $\boldsymbol{x}_i \sqsubseteq \boldsymbol{x}$. Formally,

$$P_\rho \;::=\; 0 \;\mid\; N_o(y).P_\rho \;\mid\; \overline{N_i}(t).P_\rho \;\mid\; [\boldsymbol{x}' = \boldsymbol{t}(\boldsymbol{x})]\, P_\rho$$

$Q_{!I} = !P_{I_1} \parallel \ldots \parallel !P_{I_9} \parallel !P_{I_{10}}$ This is the specification of the intruder model in a Dolev-Yao style. The dedicated channel I holds the information the intruder operates on (it can be either initial, intercepted, or forged). Each P_{I_i}, for $i = 1, \ldots, 9$ describes one capability of the intruder. The additional process $P_{I_{10}}$ has no meaning in term of intruder capability but technically it behaves as a "garbage" collector of messages in the intruder knowledge. Processes P_{I_i} are defined as follows:

$$\begin{array}{lcl}
P_{I_1} &=& \pi(x).\overline{I}(x).0 \\
P_{I_2} &=& \nu n.\overline{I}(n).0 \\
P_{I_3} &=& N_o(x).\overline{I}(x).0 \\
P_{I_4} &=& I(x).\overline{I}(x).\overline{N_i}(x).0 \\
P_{I_5} &=& I(x).\overline{I}(x).[x = \langle x_1,\, x_2\rangle].\overline{I}(x_1).\overline{I}(x_2).0 \\
P_{I_6} &=& I(x_1).\overline{I}(x_1).I(x_2).\overline{I}(x_2).\overline{I}(\langle x_1,\, x_2\rangle).0 \\
P_{I_7} &=& \mathtt{Kp}(w).I(y).\overline{I}(y).[w = \langle y,\, y'\rangle].I(x).\overline{I}(x).[x = \{z\}_{y'}].\overline{I}(z).0 \\
P_{I_8} &=& I(x).\overline{I}(x).I(k).\overline{I}(k).\overline{I}(\{x\}_k).0 \\
P_{I_9} &=& I(x).0 \\
P_{I_{10}} &=& I(x).\overline{I}(x).0
\end{array}$$

Processes P_{I_1} through P_{I_9} perform the same actions as the MSR_P intruder rules with the same index in Section 5.3.1. For example, P_{I_5} retrieves an object x previously memorized as $I(x)$, splits it into the pair (x_1, x_2), and then stores a copy of each of the terms x, x_1 and

x_2: this is exactly what r_{I_5} achieved. Channel I is used to store the intruder's knowledge in a distributed way. Process $P_{I_{10}}$ ensures that writing on I is never blocking, even in our synchronous calculus. In particular, it allows expressing every term t known to the intruder as the singleton process $\overline{I}(\underline{t}).0$, since it can rewrite a trailing sequence of outputs $\overline{I}(\underline{t}).\overline{I}(\underline{t}').0$ into $\overline{I}(\underline{t}).0 \parallel \overline{I}(\underline{t}').0$.

$Q_{!\pi} = \prod !\overline{\pi}(\underline{t}).0$ This process represents what we called "persistent information" in the case of MSR_P. We can assume the same predicate (here channel) names with the same meaning. This information is made available to client processes on each channel π (*e.g.*, `Kp`). It is assumed that no other process performs an output on π.

$Q_{I_0} = \prod \overline{I}(\underline{t}).0$ for terms $\underline{t}$. Q_{I_0} represents the initial knowledge of the intruder.

In PA_P, an *initial state* is a process $(P_{!net} \parallel \prod_\rho !P_\rho \parallel Q_{!I} \parallel Q_{!\pi} \parallel Q_{I_0})$. Subsequent states are obtained by applying the execution rules of PA defined in Section 5.2.2.

Example 5.3.2 In order to gain a better understanding of the PA_P specification methodology, we will now express the *NSPK* protocol (5.3.1) in this language. The PA_P specification of *NSPK* protocol will consist of the following processes:

$$Q_{NSPK} \quad = \quad P_{!net} \parallel P_{!A} \parallel P_{!B} \parallel Q_{!I} \parallel Q_{!\pi} \parallel Q_{I_0}$$

Here $P_{!net}$ and $Q_{!I}$ have already been defined. As with MSR_P, we rely on the abbreviations $\boldsymbol{A} = (a; b; k_a; k'_a; k_b)$ and $\boldsymbol{B} = (b; a; k_b; k'_b; k_a)$ for the given tuples of variables. The other processes are as follows:

$$P_{!A} \quad = \quad !\tilde{\pi}(\boldsymbol{A}).\ \nu n_a.\ \overline{N_i}(\{a, n_a\}_{k_b}).\ N_o(m).\ [m = \{n_a, n_b\}_{k_a}]\ .\ \overline{N_i}(\{n_b\}_{k_b}).\ 0$$

where $\tilde{\pi}(\boldsymbol{A})$ is an abbreviation for the prefix

$$\mathtt{Pr}(a).\mathtt{PrK}(a; k'_a).\mathtt{PbK}(b; k_b).\mathtt{Kp}(k_a; k'_a)$$

First, process $P_{!A}$ receives, through channels $\tilde{\pi}$, the instantiating constants of the initiator role. Then it sends the encrypted message $\{a, n_a\}_{k_b}$ on the net, where n_a is a fresh name and k_b the responder's public key. Then, $P_{!A}$ receives a message m that it tries to interpret as $\{n_a, n_b\}_{k_a}$ by decryption using the private key k_a, and by splitting the results as the pair (n_a, n_b). If this step succeeds the message $\{n_b\}_{k_b}$ is sent back to the net.

The process $P_{!B}$ representing the responder of *NSPK* is similarly defined as follows:

$$\begin{aligned} P_{!B} \quad = \quad & !\tilde{\pi}(\boldsymbol{B}).\ \nu n_b.\ N_o(m).[m = \{a, n_a\}_{k_b}]\ . \\ & \overline{N_i}(\{n_a, n_b\}_{k_a}).\ N_o(m').\ [m' = \{n_b\}_{k_b}]\ .\ 0 \end{aligned}$$

The initial knowledge of the intruder is:

$$Q_{I_0} = \overline{I}(E).0 \parallel \overline{I}(K_E).0 \parallel \overline{I}(K'_E).0$$

i.e., the intruder knows its name and its private/public key pairs. Finally the processes modeling the persistent information are the following:

$$\begin{aligned} Q_{!\pi} \quad = \quad & Q_{\tilde{\pi}(A;B;K_A;K'_A;K_B)} \parallel Q_{\tilde{\pi}(B;A;K_B;K'_B;K_A)} \parallel \\ & Q_{\tilde{\pi}(B;E;K_B;K'_B;K_E)} \parallel Q_{\tilde{\pi}(E;A;K_E;K'_E;K_A)} \end{aligned}$$

where $Q_{\tilde{\pi}(x;y;k_x;k'_x;k_y)}$ is the parallel composition of simple replicated processes that output each object in $\tilde{\pi}(x; y; k_x; k'_x; k_y)$ on channels $\tilde{\pi}$, *i.e.*, :

$$!\overline{\mathtt{Pr}}(x).0 \parallel !\overline{\mathtt{PrK}}(x; k'_x).0 \parallel !\overline{\mathtt{PbK}}(y; k_y).0 \parallel !\overline{\mathtt{Kp}}(k_x; k'_x).0 \,.$$

Here finishes the example showing how to write a security protocol in our subset of PA. ■

5.4 Encoding Protocol Specifications

This section describes two encodings: one from MSR_P to PA_P and the other from PA_P to MSR_P. As we define these encodings, we assume a common underlying signature Σ_P. In particular, the predicate symbols and terms in MSR_P find their counterpart in channel names and messages in PA_P, respectively.

The first mapping, from MSR_P to PA_P, is based on the observation that role state predicates force MSR_P rules to be applied sequentially within a role (this is not true for general MSR theories). Minor technicalities are involved in dealing with the presence of multiple instances of a same role (they are addressed through replicated processes).

At its core, the inverse encoding, from PA_P to MSR_P, maps sequential agents to a set of MSR_P rules corresponding to roles: we generate appropriate role state predicates in correspondence of the intermediate stages of each sequential process. The replication operator is not directly involved in this mapping as it finds its counterpart in the way rewriting rules are applied. The transformation of the intruder, whose behavior is fixed a priori, is treated off-line in both directions.

Before proceeding we introduce some simplifying assumptions and a preliminary observation. Without loss of generality, we assume that the rewrite rules of an MSR_P theory are written in the following form: variables occurring in two occurrences of a role state predicate $A_{\rho_i}(\boldsymbol{x})$, one in the antecedent and one in the consequent of two consecutive rules, have the same name. Moreover, in the antecedent $A_{\rho_i}(\boldsymbol{t}(\boldsymbol{x}))$ of an analysis rule, we require that all the variables introduced by $\boldsymbol{t}(\boldsymbol{x})$ be distinct from the variables $\boldsymbol{x}'$ in the consequent $A_{\rho_i}(\boldsymbol{x}')$ of the preceding rule. These assumptions, purely syntactical, simplify situations in the proofs without invalidating our analysis. Example 5.3.1 implements them.

We begin by characterizing the structure of a generic PA_P *state* reachable from an initial specification (see Sec. 5.3.2) as the parallel composition of precisely identified processes. We have the following proposition:

Proposition 5.4.1 *Let Q be a PA_P initial state. If Q is such that $Q_0 \Rightarrow^* Q$ then Q can be written as:*

$$Q \equiv \overbrace{(P_{!net} \parallel \prod_{\rho} P_{!\rho} \parallel Q_{!I} \parallel Q_{!\pi})}^{Q_!} \parallel (Q_{net} \parallel \prod_{\rho} P_\rho \parallel Q_I \parallel Q_{rem})$$

where:

$$\begin{array}{lcl}
Q_{net} & ::= & 0 \;\mid\; \prod \overline{N_o}(t).0 \\
P_\rho & ::= & 0 \;\mid\; N_o(\boldsymbol{x}).P_\rho \;\mid\; \overline{N_i}(\underline{\boldsymbol{t}}).P_\rho \;\mid\; [\underline{\boldsymbol{t}} = \boldsymbol{t}']\, P_\rho \\
Q_I & ::= & \textit{suffix of } P_{I_j}\textit{, for all } j \\
Q_{rem} & ::= & 0 \;\mid\; N_o(x).\overline{N_i}(x).0 \;\mid\; \tilde{\pi}(\boldsymbol{x}).\nu\boldsymbol{n}.P_\rho \;\mid\; \nu\boldsymbol{n}.P_\rho \;\mid\; \prod \overline{\pi}(\underline{t}).0
\end{array}$$

Proof. By induction over the number of transition steps. As the base of the induction let us observe that a PA_P initial state Q_0 is exactly the process $Q_! \parallel Q_{I_0}$ (where $Q_! = P_{!net} \parallel \prod_\rho P_{!\rho} \parallel Q_{!I} \parallel Q_{!\pi}$), and that $Q_0 \Rightarrow^* Q_0$. Then, let be Q such that $Q_0 \Rightarrow^* Q' \Rightarrow Q$. For inductive hypothesis Q' may be written as a process of form $Q_! \parallel (Q_{net} \parallel \prod_\rho P_\rho \parallel Q_I \parallel Q_{rem})$, and it is easy to check that, each transition Q from Q' can be written as well as a process of form $Q_! \parallel (Q'_{net} \parallel \prod_\rho P'_\rho \parallel Q'_I \parallel Q'_{rem})$. ∎

5.4.1 From MSR_P to PA_P

This section defines the transformation $\lceil \_ \rceil$ that, given an MSR_P configuration $(\tilde{r} : \tilde{s})$ with $\tilde{r} = (\cup_\rho(\tilde{r}_\rho), \tilde{r}_I)$ and $\tilde{s} = (\tilde{N}, \tilde{A}, \tilde{I}, \tilde{\pi})$ returns a PA_P state $Q_! \parallel Q_{net} \parallel \prod_\rho P_\rho \parallel Q_I$ (with $Q_! = (P_{!net} \parallel \prod_\rho P_{!\rho} \parallel Q_{!I} \parallel Q_{!\pi})$).

More precisely $\lceil \_ \rceil$ is a tuple of encodings $\lceil \_ \rceil^{R_\rho}$, $\lceil \_ \rceil^{R_I}$, $\lceil \_ \rceil^N$ $\lceil \_ \rceil^{A_\rho}$, $\lceil \_ \rceil^I$, $\lceil \_ \rceil^\pi$, each operating on a different component of the MSR_P configuration, as depicted in the following scheme:

$$\lceil (\cup_\rho(\tilde{r}_\rho) \cup \tilde{r}_I : \tilde{N}, \tilde{A}, \tilde{I}, \tilde{\pi}) \rceil = \overbrace{(!P_{net} \parallel \underbrace{\prod_\rho P_{!\rho}}_{\lceil \cup_\rho(\tilde{r}_\rho) \rceil^{R_\rho}} \parallel \underbrace{Q_{!I}}_{\lceil \tilde{r}_I \rceil^{R_I}} \parallel \underbrace{Q_{!\pi}}_{\lceil \tilde{\pi} \rceil^{\pi}})}^{Q_!} \parallel (\underbrace{Q_{net}}_{\lceil \tilde{N} \rceil^N} \parallel \underbrace{\prod_\rho P_\rho}_{\lceil \tilde{A} \rceil^{A_\rho}} \parallel \underbrace{Q_I}_{\lceil \tilde{I} \rceil^I})$$

This definition is interpreted as follows:

- P_{net} is fixed a priori (see Section 5.3.2);
- $\prod_\rho P_{!\rho}$ and $Q_{!I}$, result from the transformation of respectively $\cup_\rho(\tilde{r}_\rho)$ and $\tilde{r}_I$;
- $Q_{!\pi}$ results from the transformation of $\tilde{\pi}$, and
- Q_{net}, $\prod_\rho P_\rho$, and Q_I result from transformation of, respectively $\tilde{N}$, $\tilde{A}$ and $\tilde{I}$.

Intuitively, the transformations $\lceil \cup_\rho(\tilde{r}_\rho) \rceil^{R_\rho}$ and $\lceil \tilde{r}_I \rceil^{R_I}$ return the parallel composition of replicated (*i.e.,* preceeded by a !) processes modeling the sequence of actions of each role and of the intruder, respectively. The replication operator makes these processes always available for instantiation as the MSR rules are. The intruder process is fixed a priori and its transformation is obvious. The transformation of $\tilde{r}_\rho$, *e.g.,* the rules of role ρ, is more interesting: it results in a sequential process P_ρ, whose send, receive or match sub-processes are obtained, respectively from send, receive and analysis rules in $\tilde{r}_\rho$ (see also Example 5.4.2). Particular attention is reserved for the translation of the first instantiation rule r_{ρ_0}.

The next transformations act on predicates $\tilde{N}$, $\tilde{A}$ and $\tilde{I}$ in the MSR_P state, and return the parallel composition of sequential processes. More precisely, all the predicates $N(t)$ in $\tilde{N}$ are transformed into singleton output processes $\overline{N_o}(t).0$ representing the availability of the ground datum t on the net. Similarly predicates $I(t)$ in $\tilde{I}$ are transformed into output processes $\overline{I}(t).0$ representing the intruder knows the datum t. Finally the transformation of each predicates $A_{\rho_i}(\boldsymbol{t})$, in $\tilde{A}$ returns the suffix of the process P_ρ that model the remaining role rules $r_{\rho_{i+1}}, \ldots, r_{\rho_{l_\rho}}$. Variable in P_ρ are partially instantiated depending on terms in $\boldsymbol{t}$.

The acquisition of permanent facts and the creation of new variables $\boldsymbol{x}$ are mapped, respectively to a sequence of input actions from processes $Q_{!\pi}$, and actions νx for each x in $\boldsymbol{x}$. In turn $Q_{!\pi}$ is the parallel composition of replicated output processes $\overline{\pi}(\boldsymbol{t}).0$, each obtained from a permanent predicates $\pi(\boldsymbol{t})$ in $\tilde{\pi}$. Their task is to make permanent fact always available to be received.

Whenever unambiguous, we will omit the identifying subscript from the encoding functions $\lceil\_\rceil^{R_\rho}$, $\lceil\_\rceil^{R_I}$, $\lceil\_\rceil^{N}$ $\lceil\_\rceil^{A_\rho}$, $\lceil\_\rceil^{I}$, or $\lceil\_\rceil^{\pi}$, simplifying them to $\lceil\_\rceil$.

$\lceil\_\rceil^{R_\rho}$. In transforming processes $P_{!\rho}$, for each role ρ, a subroutine function $\lceil\_\rceil^{\#}_{(\boldsymbol{x})}$ is called by the top level transformation $\lceil\_\rceil$. $\lceil\_\rceil^{\#}_{(\boldsymbol{x})}$ ranges over the set of role rules $\cup_\rho(\tilde{r}_\rho)$, and takes a tuple $\boldsymbol{x}$ of variables as parameter. This parameter, initially the empty tuple ϵ, collects variables used along the rewriting rule, and uses them opportunely in the building process. We define it on the structure of the role rule $r_{\rho_i} \in \tilde{r}_\rho$ involved. Formally for $i = 0$:

$$\lceil r_{\rho_0} \rceil = \tilde{\pi}(\boldsymbol{x}).\nu\boldsymbol{n}.\lceil r_{\rho_1} \rceil^{\#}_{(\boldsymbol{x};\boldsymbol{n})} \qquad \text{if } r_{\rho_0} : \tilde{\pi}(\boldsymbol{x}) \rightarrow \exists \boldsymbol{n}.A_{\rho_0}(\boldsymbol{x};\boldsymbol{n}), \tilde{\pi}(\boldsymbol{x})$$

A role generation rule is mapped onto a process which first receives, in sequence, permanent terms via the channels π in $\tilde{\pi}$ and then generates all the new names $\boldsymbol{n}$ used in this role.

For $0 < i \leq l_\rho - 1$:

$$\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x})} = \begin{cases} \overline{N_i}(t(\boldsymbol{x})).\lceil r_{\rho_{i+2}} \rceil^{\#}_{(\boldsymbol{x})} & , \text{ if } r_{\rho_{i+1}} = A_{\rho_i}(\boldsymbol{x}) \rightarrow A_{\rho_{i+1}}(\boldsymbol{x}), N(t(\boldsymbol{x})) \\ N_o(y)\lceil r_{\rho_{i+2}} \rceil^{\#}_{(\boldsymbol{x};y)} & , \text{ if } r_{\rho_{i+1}} = A_{\rho_i}(\boldsymbol{x}), N(y) \rightarrow A_{\rho_{i+1}}(\boldsymbol{x};y) \\ [\boldsymbol{x} = \boldsymbol{t}(\boldsymbol{x'})]\ \lceil r_{\rho_{i+2}} \rceil^{\#}_{(\boldsymbol{x'})}, & \text{if } r_{\rho_{i+1}} = A_{\rho_i}(\boldsymbol{t}(\boldsymbol{x'})), \rightarrow A_{\rho_{i+1}}(\boldsymbol{x'}) \end{cases}$$

The transformation of a send or a receive rewriting rule is straightforward. The translation of an analysis rewriting rule is less obvious: the matching $[\boldsymbol{x} = \boldsymbol{t}(\boldsymbol{x'})]$ is intended to simulate the matching that — in the semantics of MSR — happens between the terms in consequent, $A_{\rho_i}(\boldsymbol{x})$, of rule r_{ρ_i} and the terms in the antecedent $A_{\rho_i}(\boldsymbol{t}(\boldsymbol{x'}))$ of (actual) rule $r_{\rho_{i+1}}$. Finally and with a little abuse of notation, we set $\lceil r_{\rho_{l_\rho+1}} \rceil^{\#}_{(\boldsymbol{x})} = 0$.

The final process defining the role ρ behavior is the following: $P_\rho \stackrel{def}{=} \lceil r_{\rho_0} \rceil$

$\lceil\_\rceil^{R_I}$. The intruder is handled by simply mapping $\tilde{r}_I$ to $Q_{!I}$. More precisely, we define the transformation function $\lceil\_\rceil$ that relates the intruder rewriting rule r_{I_j} with the sequential agents P_{I_j} defined in Section 5.3.2. Moreover the transformation produces the additional process $!P_{I_{10}}$.

At this point the transformation is complete as soon as the state $\tilde{s} = (\tilde{N}, \tilde{A}, \tilde{I}, \tilde{\pi})$ is treated.

$\lceil\_\rceil^{A_\rho}$. For each $A_{\rho_i}(\boldsymbol{t}) \in \tilde{A}$, we define $P_{A_{\rho_i}(\boldsymbol{t})} = \lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x})}[\boldsymbol{t}/\boldsymbol{x}]$, where $\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\_)}$ was defined above and $\boldsymbol{x}$ are the variables appearing as argument of the *consequent* predicate $A_{\rho_i}(\boldsymbol{x})$ in r_{ρ_i}.

$\lceil\_\rceil^{N}$, $\lceil\_\rceil^{I}$, $\lceil\_\rceil^{\pi}$. The multiset $\tilde{N}$ guides the definition of Q_{net}, that is $Q_{net} \stackrel{def}{=} \prod_{N(t)\in\tilde{N}} \overline{N}(t).0$. Similarly, $Q_I \stackrel{def}{=} \prod_{I(t)\in\tilde{I}} \overline{I}(t).0$, and $Q_{!\pi} \stackrel{def}{=} \prod_{\pi(t)\in\tilde{\pi}} !\overline{\pi}(t).0$. Formally:

$$\lceil\cdot\rceil = 0 \qquad \lceil N(t), \tilde{N}\rceil = \overline{N_o}(t).0 \parallel \lceil\tilde{N}\rceil$$

$$\lceil\cdot\rceil = 0 \qquad \lceil I(t), \tilde{I}\rceil = \overline{I}(t).0 \parallel \lceil\tilde{I}\rceil$$

$$\lceil\cdot\rceil = 0 \qquad \lceil \pi(t), \tilde{\pi}\rceil = !\overline{\pi}(t).0 \parallel \lceil\tilde{\pi}\rceil$$

Example 5.4.2 (Translation of $NSPK$ from MSR_P to PA_P) We now provide an example on how $\lceil\_\rceil$ works. We apply it to the MSR_P specification of $NSPK$ given in Section 5.3.1.

$$
\begin{array}{lll}
\lceil \overbrace{r_{A_0}}^{\tilde{\pi}(\boldsymbol{A})\to\exists n_a.\tilde{\pi}(\boldsymbol{A}),A_0(\boldsymbol{A};n_a)} \rceil & = & !\tilde{\pi}(\boldsymbol{A}).\nu n_a.\lceil r_{A_1}\rceil^{\#}_{(\boldsymbol{A};n_a)} \\
\lceil \overbrace{r_{A_1}}^{A_0(\boldsymbol{A};n_a)\to N(\{a,n_a\}_{k_b}),A_1(\boldsymbol{A};n_a)} \rceil^{\#}_{(\boldsymbol{A};n_a)} & = & \overline{N_i}(\{a,n_a\}_{k_b}).\lceil r_{A_2}\rceil^{\#}_{(\boldsymbol{A};n_a)} \\
\lceil \overbrace{r_{A_2}}^{A_1(\boldsymbol{A};n_a),N(m)\to A_2(\boldsymbol{A};n_a;m)} \rceil^{\#}_{(\boldsymbol{A};n_a)} & = & N_o(m).\lceil r_{A_3}\rceil^{\#}_{(\boldsymbol{A};n_a;m)} \\
\lceil \overbrace{r_{A_3}}^{A_2(\boldsymbol{A};n_a;\{n_a,n_b\}_{k_a})\to A_3(\boldsymbol{A};n_a;n_b)} \rceil^{\#}_{(\boldsymbol{A};n_a;m)} & = & [(\boldsymbol{A};n_a;m)=(\boldsymbol{A};n_a;\{n_a,n_b\}_{k_a})]\,. \\
 & & \lceil r_{A_4}\rceil^{\#}_{(\boldsymbol{A};n_a;n_b)} \\
\lceil \overbrace{r_{A_4}}^{A_3(\boldsymbol{A};n_a;n_b)\to N(\{n_b\}_{k_b}),A_4(\boldsymbol{A};n_a;n_b)} \rceil^{\#}_{(\boldsymbol{A};n_a;n_b)} & = & \overline{N_i}(\{n_b\}_{k_b}).\lceil.\rceil^{\#}_{(\boldsymbol{A};n_a;n_b)} \\
 & & \\
\lceil.\rceil^{\#}_{(\boldsymbol{A};n_a;n_b)} & = & 0
\end{array}
$$

In summary:

$$
\begin{array}{lll}
\lceil\mathcal{R}_A\rceil & = & !\tilde{\pi}(\boldsymbol{A}).\nu n_a.\overline{N_i}(\{a,n_a\}_{k_b}).N_o(m). \\
 & & [\boldsymbol{A};n_a;m=\boldsymbol{A};n_a;\{n_a,n_b\}_{k_a}]\,.\overline{N_i}(\{n_b\}_{k_b}).0
\end{array}
$$

which can be simplified into

$$
\begin{array}{lll}
\lceil\mathcal{R}_A\rceil & = & !\tilde{\pi}(\boldsymbol{A}).\nu n_a.\overline{N_i}(\{a,n_a\}_{k_b}).N_o(m). \\
 & & [m=\{n_a,n_b\}_{k_a}]\,.\overline{N_i}(\{n_b\}_{k_b}).0
\end{array}
$$

by means of the structural equivalence, which removes items in corresponding positions in pattern matching over tuples. This process is exactly the same provided in Section 5.3.2.

Similarly (omitting the details) it is easy to check that:

$$
\begin{array}{lll}
\lceil\mathcal{R}_B\rceil & = & !\tilde{\pi}(\boldsymbol{B}).\nu n_b.N_o(m). \\
 & & [\boldsymbol{B};n_b;m=\boldsymbol{B};n_b;\{a,n_a\}_{k_b}]\,.\overline{N_i}(\{n_a,n_b\}_{k_a}). \\
 & & N_o(m).[\boldsymbol{B};n_b;n_a;m'=\boldsymbol{B};n_b;n_a;\{n_b\}_{k_b}]\,.0
\end{array}
$$

■

5.4.2 From PA_P to MSR_P

This section defines the transformation $\lfloor\_\rfloor$ that given a PA_P state returns a configuration in MSR_P. Indeed $\lfloor\_\rfloor$ consists of encodings

$\lfloor\_\rfloor_{!\rho}$, $\lfloor\_\rfloor_{!I}$, $\lfloor\_\rfloor_{net}$, $\lfloor\_\rfloor_{\rho}$ $\lfloor\_\rfloor_{I}$ and $\lfloor\_\rfloor_{\pi}$,

each operating on different sub-processes of the PA_P state. The following schema describes the overall encoding pictorially (processes involved in any transformation are boxed):

$$
\lfloor(P_{!net} \parallel \overbrace{\boxed{\textstyle\prod_\rho P_{!\rho} \parallel Q_{!I} \parallel Q_{!\pi}}}^{Q_!})\rfloor \parallel (\boxed{Q_{net} \parallel \textstyle\prod_\rho P_\rho \parallel Q_I} \parallel Q_{rem}) =
$$

$$(\underbrace{\cup_\rho(\tilde{r}_\rho)}_{\lfloor\prod_\rho P_{!\rho}\rfloor_{!\rho}} \cup \underbrace{\tilde{r}_I}_{\lfloor Q_{!I}\rfloor_{!I}} : \underbrace{\tilde{N}}_{\lfloor Q_{net}\rfloor_{net}} , \underbrace{\tilde{A}}_{\lfloor\prod_\rho P_\rho\rfloor_\rho} \underbrace{\tilde{I}}_{\lfloor Q_I\rfloor_I} \underbrace{\tilde{\pi}}_{\lfloor Q_{!\pi}\rfloor_\pi} ,)$$

Note that the following processes are not involved in any transformation:

- $P_{!net}$, since it implements a form of buffering that is unnecessary in MSR;
- Q_{rem}, since it represents partial computations (see Proposition 5.4.1). As we will see later, they will not have any significant MSR_P counterpart.

Intuitively $\lfloor\prod_\rho P_{!\rho}\rfloor_{!\rho}$ analyzes each (un-banged) sequential processes P_ρ in $\prod_\rho P_{!\rho}$ and for each ρ returns the multiset of the rule corresponding to P_ρ's sequential steps. Input, output and analysis sub-process in P_ρ are mapped into receive, send, and analysis rewriting rules for role ρ, respectively. Prefixes $\nu\boldsymbol{x}$ and input sequences $\tilde{\pi}(\boldsymbol{x})$ are turned into an instantiation rule. Technicalities are needed for the management of variables and of the predicate indexes in building rules r_{ρ_i}'s. Two parameters, the step number and the variables, are passed along the transformation. Similar devices support the transformation of each processes P_ρ in $\prod_\rho P_\rho$. They represent partial execution of the protocol by role ρ, their analysis produces the state predicates $A_{\rho_i}(\boldsymbol{t})$, for suitable i and $\boldsymbol{t}$.

The transformation of $Q_{!I}$ and $Q_{!\pi}$ are straightforward: the former maps directly to the intruder rewriting rules of MSR_P, while in the latter each $!\overline{\pi}(\boldsymbol{t}).0$ in $Q_{!\pi}$ is mapped to the persistent predicates $\pi(\boldsymbol{t})$. The same can be said about processes Q_{net}: each sequential process $\overline{N_o}(t).0$ is mapped into a predicate $N(t)$ in the MSR_P state.

The transformation of the processes in Q_I is more complex. Indeed, we need to distinguish between processes that represent immediately available intruder knowledge (*e.g.*, $\overline{I}(t).0$) from processes that do not (*e.g.*, $N_o(x).\overline{I}(x).0$). The former are transformed in corresponding intruder predicates $I(t)$, while the latter are generally discarded. Generally speaking $\lfloor\_\rfloor$ is not injective, and similar situations can happen while transforming processes into MSR_P states. Said differently, PA_P steps are finer grained then the MSR_P steps, and as a consequence some processes do not represent proper MSR objects (for example processes in Q_{rem}) and they have to be ignored, while others represent MSR_P objects even when they are only partially completed (for example processes $\overline{I}(\underline{t}).P'_I$) and their translation can be anticipated (see also Figure 5.1 or later for details).

In the following, with a little abuse of notation, we drop the subscript from the transformations, $\lfloor\_\rfloor_{!\rho}$, $\lfloor\_\rfloor_{!I}$, $\lfloor\_\rfloor_{net}$, $\lfloor\_\rfloor_\rho$, $\lfloor\_\rfloor_I$ and $\lfloor\_\rfloor_\pi$, when no ambiguity arises, writing them instead as $\lfloor\_\rfloor$. We now describe each transformation in detail.

$\lfloor\_\rfloor_{!\rho}$. The basic translation involves the transformation function $\lfloor\_\rfloor^{\#}_{(i;\boldsymbol{x})}$ for the $P_{!\rho}$'s (called as a subroutine by the top level transformation $\lfloor\_\rfloor$) which, given a sequential agent representing a role ρ, returns the multiset of rules $\tilde{r}_\rho$. Here i is a non-negative integer. Formally:

$$\begin{aligned}
\lfloor\tilde{\pi}(\boldsymbol{x}).\nu\boldsymbol{n}.P'_\rho\rfloor &= \{\tilde{\pi}(\boldsymbol{x}) \to \exists\boldsymbol{n}.A_{\rho_0}(\boldsymbol{n};\boldsymbol{x})\} \ \cup\ \lfloor P'_\rho\rfloor^{\#}_{(1:(\boldsymbol{x};\boldsymbol{n}))}\\
\lfloor N_o(y).P'_\rho\rfloor^{\#}_{(i:\boldsymbol{x})} &= \{A_{\rho_{i-1}}(\boldsymbol{x}), N(y) \to A_{\rho_i}(\boldsymbol{x};y)\} \ \cup\ \lfloor P'_\rho\rfloor^{\#}_{(i+1:(\boldsymbol{x};y))}\\
\lfloor\overline{N_i}(t).P'_\rho\rfloor^{\#}_{(i:\boldsymbol{x})} &= \{A_{\rho_{i-1}}(\boldsymbol{x}) \to A_{\rho_i}(\boldsymbol{x}), N(t)\} \ \cup\ \lfloor P'_\rho\rfloor^{\#}_{(i+1:\boldsymbol{x})}\\
\lfloor[\boldsymbol{x}' = \boldsymbol{t}(\boldsymbol{x}'')]\ .P'_\rho\rfloor^{\#}_{(i:\boldsymbol{x})} &= \{A_{\rho_{i-1}}(\boldsymbol{x}[\boldsymbol{t}(\boldsymbol{x}'')/\boldsymbol{x}']) \to A_{\rho_i}(\boldsymbol{x}[(\boldsymbol{x}''\text{-}\ \boldsymbol{x})/\boldsymbol{x}']), N(t)\}\\
&\qquad \cup\ \lfloor P'_\rho\rfloor^{\#}_{(i+1:(\boldsymbol{x}[(\boldsymbol{x}''-\boldsymbol{x})/\boldsymbol{x}']))}\\
\lfloor 0\rfloor^{\#}_{(i;\boldsymbol{x})} &= \cdot
\end{aligned}$$

The transformation of a send, of a receive and of a new process are quite obvious and require no additional comment. The translation of a match process $[\boldsymbol{x}' = \boldsymbol{t}(\boldsymbol{x}'')]\ .P'_\rho$, whose aim is to analyze some previously received message, yields an analysis rewrite rule. It would be straightforward if all variables of the role were matched each time (possibly redundantly) as these variables could be used to build the corresponding role predicate. Instead, only a subset of all variables appears during matching (the variables that are being analyzed), while the corresponding role predicate needs all of them. We reconstruct them be carrying a parameter which stores the tuple of all the variables used so far by the role. With this as a template, we can construct the right tuples in the rule antecedent and in the rule consequent.

$\lfloor\_\rfloor_{!I}$. The intruder process $Q_{!I}$ is mapped directly to the MSR_P intruder rules $\tilde{r}_I$, with each $!P_{I_j}$ associated with r_{I_j}. Process $!P_{I_{10}}$ is dropped.

$\lfloor\_\rfloor_{net}$. Each occurrence of a process $\overline{N_o}(t).0$ in Q_{net} is mapped to a state element $N(t)$.

$\lfloor\_\rfloor_{\rho}$. Let P_ρ be an instantiated suffix (in $\prod_\rho P_\rho$) of a role specification $P_!\rho$, and let $\theta = [\boldsymbol{x}/\boldsymbol{t}]$ be the witnessing substitution. If P_ρ starts with either a persistent input $\pi(\boldsymbol{x})$ or the ν operator, we set $\lfloor P_\rho \rfloor = \cdot$. Otherwise, let i be the index at which P_ρ occurs in $P_!\rho$ as for the above definition. Then $\lfloor P_\rho \rfloor = A_{\rho_i}(\boldsymbol{t})$.

$\lfloor\_\rfloor_{I}$. Each object in Q_I (that, we recall, contains all the prefixes of P_{I_j} processes), is translated using the function $\lfloor\_\rfloor_I$, defined below:

$$\lfloor 0 \rfloor_I = \lfloor \overline{N_o}(\underline{t}).0 \rfloor_I = \lfloor \nu n.P_I \rfloor_I = \lfloor I(x).P_I \rfloor_I = \lfloor \pi(x).P_I \rfloor_I = \cdot$$

$$\begin{array}{lcl} \lfloor \overline{I}(\underline{t}).P_I \rfloor_I & = & I(\underline{t}), \lfloor P_I \rfloor_I \\ \lfloor [\underline{\boldsymbol{t}} = \boldsymbol{t}(\boldsymbol{x})]\ .P_I \rfloor_I & = & \begin{cases} \lfloor P_I[\theta] \rfloor_I & \text{if } \boldsymbol{t}(\boldsymbol{x})[\theta] = \underline{\boldsymbol{t}} \\ \cdot & \text{otherwise} \end{cases} \end{array}$$

$\lfloor\_\rfloor_{\pi}$. Each process $!\pi(\boldsymbol{x})$ in $P_{!\pi}$, or $\pi(\boldsymbol{x})$ in P_π is translated into the state object $\pi(\boldsymbol{x})$.

The intuition underlying the definition of $\lfloor\_\rfloor_I$ is to collect all the ground output events of a partially executed intruder processes (*i.e.,* processes that are suffixes of some P_{I_j}, but that do have not the form $\overline{I}(t).0$)[1] as process $P_{I_{10}}$ has the potential of turning them into the canonical form $\overline{I}(t).0$. In this way, we map any such intruder suffix into an MSR_P state where this knowledge is already present. In particular, each object $\overline{I}(t).0$ (respectively the $\overline{I}(t).\overline{I}(t).0$) in Q_I is rendered as the state element $I(t)$ (respectively pair of elements $I(t), I(t)$), and that the un-banged processes P_{I_j} are mapped into the empty multiset. Note that $\lfloor\_\rfloor_I$ is not injective.

$P_{!net}$ and Q_{rem} disappear (*i.e.,* they are mapped onto the empty multiset).

Example 5.4.3 (Translation of $NSPK$ from PA_P to MSR_P) We now provide an example on how $\lfloor\_\rfloor$ works, by applying it to the PA_P specification of $NSPK$ given in Section 5.3.2. Let us start

[1] From now on let us call them all *intruder partial suffixes*.

by considering the process P_A:

$$P_A = \tilde{\pi}(\boldsymbol{A}).\nu n_a.\overbrace{\overline{N_i}(\{a,n_a\}_{k_b}).\underbrace{N_o(m).\overbrace{[m=\{n_a,n_b\}_{k_a}].\underbrace{\overline{N_i}(\{n_b\}_{k_b}).0}_{P'''_A}}^{P''_A}}_{P'_A}}^{P'_A}$$

we have:

$$\begin{array}{lcl}
\lfloor P_A \rfloor & = & \tilde{\pi}(\boldsymbol{A}) \rightarrow \exists n_a.\tilde{\pi}(\boldsymbol{A}), A_0(\boldsymbol{A};n_a) \\
 & & \cup \lfloor P'_A \rfloor^{\#}_{(1:(\boldsymbol{A};n_a))} \\
\lfloor P'_A \rfloor^{\#}_{(1:(\boldsymbol{A};n_a))} & = & A_0(\boldsymbol{A};n_a) \rightarrow N(\{A,n_a\}_{k_b}), A_1(\boldsymbol{A};n_a) \\
 & & \cup \lfloor P''_A \rfloor^{\#}_{(2:(\boldsymbol{A};n_a))} \\
\lfloor P''_A \rfloor^{\#}_{(2:(\boldsymbol{A};n_a))} & = & A_1(\boldsymbol{A};n_a), N(m) \rightarrow A_2(\boldsymbol{A};n_a;m) \\
 & & \cup \lfloor P'''_A \rfloor^{\#}_{(3:(\boldsymbol{A};n_a;m))} \\
\lfloor P'''_A \rfloor^{\#}_{(3:(\boldsymbol{A};n_a;m))} & = & A_2(\boldsymbol{A};N_A;\{n_a,n_b\}_{k_a}) \rightarrow A_3(\boldsymbol{A};n_a;n_b) \\
 & & \cup \lfloor P'''_A \rfloor^{\#}_{(4:(\boldsymbol{A};n_a;n_b))} \\
\lfloor P''''_A \rfloor^{\#}_{(4:(\boldsymbol{A};n_a;n_b))} & = & A_3(\boldsymbol{A};n_a;n_b) \rightarrow N(\{n_b\}_{k_b}), A_4(\boldsymbol{A};n_a;n_b) \\
 & & \cup \lfloor 0 \rfloor^{\#}_{(5:(\boldsymbol{A};n_a;n_b))} \\
\lfloor 0 \rfloor^{\#}_{(5:(\boldsymbol{A};n_a;n_b))} & = & .
\end{array}$$

In summary:

$$\lfloor P_A \rfloor = \begin{cases}
\tilde{\pi}(\boldsymbol{A}) & \rightarrow \exists n_a.\tilde{\pi}(\boldsymbol{A}), A_0(\boldsymbol{A};n_a) \\
A_0(\boldsymbol{A};n_a) & \rightarrow N(\{a,n_a\}_{k_b}), A_1(\boldsymbol{A};n_a) \\
A_1(\boldsymbol{A};n_a), N(m) & \rightarrow A_2(\boldsymbol{A};n_a;m) \\
A_2(\boldsymbol{A};n_a;\{n_a,n_b\}_{k_a}) & \rightarrow A_3(\boldsymbol{A};n_a;n_b) \\
A_3(\boldsymbol{A};n_a;n_b) & \rightarrow N(\{n_b\}_{k_b}), A_4(\boldsymbol{A};n_a;n_b)
\end{cases}$$

Similarly (omitting details):

$$\lfloor P_B \rfloor = \begin{cases}
\tilde{\pi}(\boldsymbol{B}) & \rightarrow \exists n_b.\tilde{\pi}(\boldsymbol{B}), B_0(\boldsymbol{B};n_b) \\
B_0(\boldsymbol{B};n_b), N(m) & \rightarrow B_1(\boldsymbol{B};n_b;m) \\
B_1(\boldsymbol{B};n_b;\{a,n_a\}_{k_b}), & \rightarrow B_2(\boldsymbol{B};n_b;n_a) \\
B_2(\boldsymbol{B};n_b;n_a) & \rightarrow N(\{n_a,n_b\}_{k_a}), B_3(\boldsymbol{B};n_b;n_a) \\
B_3(\boldsymbol{B};n_b;n_a), N(m') & \rightarrow B_4(\boldsymbol{B};n_b;n_a;m') \\
B_4(\boldsymbol{B};n_b;n_a;\{n_b\}_{k_b}) & \rightarrow B_5(\boldsymbol{B};n_b;n_a)
\end{cases}$$

■

5.5 Correspondence Relation between MSR_P and PA_P

This section introduces a correspondence relation between MSR_P configurations and PA_P states, such that two corresponding computations are characterized by *identical network messages and intruder knowledge, step by step*. This will allow us to prove that the translations presented in

this chapter are reachability-preserving in a very strong sense. Indeed, we show that our encodings transform a configuration (respectively a state) into a state (respectively configuration) that correspond to each other in our relation, and this implies that our encodings can preserve secrecy and authenticity properties while going from MSR to PA and vice versa (this is further discussed in Section 5.6). In the following we formalize the notion of observation and transition step with respect to the intruder and the network in the MSR and PA frameworks.

Our notion of observation is concerned with only those messages representing terms in the net and the intruder knowledge. They are given by the predicates $N(t)$ and $I(t)$ in an MSR_P configuration. Formally we have:

Definition 5.5.1 *Given a multiset of ground atoms $\tilde{s}$ and a predicate name $a \in \{N, I\}$, we define the* projection *of $\tilde{s}$ along a as the set $Prj_a(\tilde{s}) = \{t : a(t) \in \tilde{s}\}$. If $C = (\tilde{r}; \tilde{s})$ is a configuration, we set $Prj_a(\tilde{C}) = Prj_a(\tilde{s})$.*

Collecting the network messages and the intruder knowledge of a PA_P state P is trickier because of the particular form of the processes representing that the intruder and the network (see Section 5.3). More precisely, these terms appear in output actions (over channels N_o or I) that will be surely executed by either Q_I or Q_{net}. Indeed, Q_I and Q_{net} outputs (on those channels) are always realizable, because processes $P_{I_{10}}$ and $P_{!net}$ can always accept them as input. In order to collect those messages we introduce the notation $Q \xrightarrow{\alpha}$ to indicate that α is the set of output actions that process Q (intended to be Q_I or Q_{net}) is able to execute in later steps of execution. Formally:

Definition 5.5.2 *Given a process Q, the judgment $Q \xrightarrow{\alpha}$ is defined by the following rules:*

$$\frac{}{0 \xrightarrow{\emptyset}} \qquad \frac{}{a(\boldsymbol{x}).P \xrightarrow{\emptyset}} \qquad \frac{P \xrightarrow{\alpha}}{\overline{a}(\boldsymbol{t}).P \xrightarrow{\{\overline{a}(\boldsymbol{t})\}\cup\alpha}} \qquad \frac{}{\nu n.P \xrightarrow{\emptyset}} \qquad \frac{Q' \xrightarrow{\alpha} \quad Q \equiv Q'}{Q \xrightarrow{\alpha}}$$

$$\frac{Q \xrightarrow{\alpha} \qquad P \xrightarrow{\alpha'}}{(Q \parallel P) \xrightarrow{\alpha\cup\alpha'}} \qquad \frac{P[\theta] \xrightarrow{\alpha} \qquad \underline{\boldsymbol{t}}' = \boldsymbol{t}[\theta]}{[\underline{\boldsymbol{t}}' = \boldsymbol{t}]\,.P \xrightarrow{\alpha}} \qquad \frac{\nexists\theta : \underline{\boldsymbol{t}}' = \boldsymbol{t}[\theta]}{[\underline{\boldsymbol{t}}' = \boldsymbol{t}]\,.P \xrightarrow{\emptyset}}$$

In the following we write $\overline{a}(\boldsymbol{t}) \in Q$ if $\overline{a}(\boldsymbol{t}) \in \alpha$ where $\alpha : Q \xrightarrow{\alpha}$.

Definition 5.5.3 *Let a be a channel label in $\{N_o, I\}$, we define the* observations *of process Q along a as the set $Obs_a(Q) = \{\boldsymbol{t} : \overline{a}(\boldsymbol{t}) \in Q\}$.*

Using Definitions 5.5.1 and 5.5.3, we make precise what we intend for an MSR_P configuration and a PA_P state to be corresponding.

Definition 5.5.4 *Given an MSR_P configuration C and a PA_P state Q. We say that C and Q are* corresponding, *written $C \bowtie Q$, if and only if the following conditions hold:*

1. $Prj_N(C) = Obs_{N_o}(Q)$
2. $Prj_I(C) = Obs_I(Q)$

Informally $C \bowtie Q$ means that the messages that are stored in the net and the intruder knowledge are the same in configuration C and state Q.

The interaction between our notions of observation and our encodings is captured in the following proposition:

Proposition 5.5.5 *Let C be an MSR_P configuration, and Q be a PA_P state. Then:*

$$\lfloor \lceil C \rceil \rfloor = C; \tag{5.5.1}$$

$$\lceil \lfloor Q \rfloor \rceil = Q' \text{ where } Q' \text{ is such that } \lfloor Q' \rfloor \bowtie Q, \quad Obs_{N_o}(Q') = Obs_{N_o}(Q) \text{ and } Obs_I(Q') = Obs_I(Q). \tag{5.5.2}$$

Proof. The critical point here is when the non injective $\lfloor \_ \rfloor$ function is applied. More precisely, $\lfloor \_ \rfloor$ shows its non-injectivity when dealing with:

(a) *intruder partial suffixes i.e.,* suffixes of some P_{I_j} that do not have the form $\overline{I}(t).0$;

(b) not-yet-instantiated process roles, *i.e.*, un-banged processes in P_ρ starting with π or ν.

In proving (5.5.1), we observe that starting from an MSR_P configuration C, process $\lceil C \rceil$ contain neither intruder partial suffixes nor not-yet-instantiated role processes. As a consequence by applying again $\lfloor \_ \rfloor$, an easy induction yields C back.

More difficult is the proof of (5.5.2). Here Q may contain some process that is an intruder partial suffix, or a not-yet-instantiated process role. In this case different Q, Q', may converge, via $\lfloor \_ \rfloor$, to the same set of predicates $\tilde{\pi}, \tilde{I}$. However not-yet-instantiated process roles do not affect the $\bowtie$ relation, because only communication over π or $\mathbf{pa}_\nu$ transitions are possible from them. Then all the remaining difficulties are hidden in intruder partial suffixes. In Figure 5.1, we have depicted one of these situation, involving where partial suffixes of P_{I_5} and P_{I_6}. Now we can observe that:

- because of the way we have defined $Obs_I(\_)$ and from the fact that $\lfloor Q \rfloor_I = \lfloor Q' \rfloor_I = \ldots = \tilde{I}$, we have that $Obs_I(Q) = Obs_I(Q') = \ldots$, *i.e.,* all the P_I's are equivalent with respect to the following relation

$$\mathcal{O}(Q_1, Q_2) \stackrel{def}{=} Obs_I(Q_1) = Obs_I(Q_2)$$

  From now on let us consider a witness $[Q]$ of the quotient class $Q_I/\mathcal{O}$.

- $Prj_I(\lfloor Q' \rfloor_I) = Obs_I(Q')$ for all $Q' \in [Q']$, because $\lfloor \_ \rfloor_I$ is build exactly to maintain the intruder knowledge.

Now when applying $\lceil \lfloor Q \rfloor_I \rceil_I$ back for some $Q' \in [Q]$, by definition of $\lceil \_ \rceil_I$, we obtain exactly that $Q^\# \in [Q]$ that contain no partial suffixes of P_{I_j}. Again Figure 5.1 may help visualize the intuition. Analogous considerations (indeed simpler) can be provided when predicates $\tilde{N}$ and processes in P_{net} are involved. ∎

Moreover we have that an MSR_P configuration always corresponds to its encoding in PA_P:

Lemma 5.5.6 *Let C be an MSR_P configuration. Then $C \bowtie \lceil C \rceil$.*

Proof. Observe that $\lceil \tilde{N} \rceil = \prod_{N(t) \in \tilde{N}} \overline{N_o}(t).0$, that $\lceil \tilde{I} \rceil = \prod_{I(t) \in \tilde{I}} \overline{I}(t).0$, and that no other multiset in C generates any $\overline{N_o}(t).0$ or $\overline{I}(t).0$, via $\lceil \_ \rceil$. Then it easily follows that:

$$Prj_N(C) = Obs_{N_o}(\lceil C \rceil)$$
$$Prj_I(C) = Obs_I(\lceil C \rceil)$$

∎

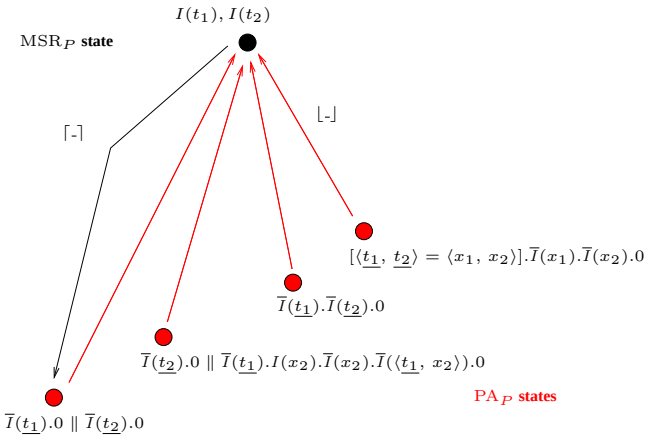


Figure 5.1: An example of a possible scenario when applying the translations $\lfloor\lceil\_\rceil_I\rfloor_I$

The dual result holds as well, *i.e.*, every PA_P state always corresponds to its MSR_P encoding:

Lemma 5.5.7 *Let be Q a PA_P state. Then $\lfloor Q\rfloor \bowtie Q$.*

Proof. The proof follows considering similar argument of Lemma 5.5.6. ∎

On the basis of these concepts, we can now define a relation between MSR_P configurations and PA_P states, a form of weak bisimulation we call *correspondence*, such that if in MSR_P is possible to perform an action (by applying a rule) that will lead to a new configuration, then in PA_P is possible to follow some transitions that will lead in a corresponding state, and vice versa.

Definition 5.5.8 *Let $\mathcal{C}$ and $\mathcal{Q}$ be the set of all MSR_P configurations and PA_P states, respectively. We call* correspondence *the largest relation $\sim \subseteq \mathcal{C} \times \mathcal{Q}$ satisfying the following conditions: for all $(\tilde{r} : \tilde{s}) \sim Q$*

1. $(\tilde{r} : \tilde{s}) \bowtie Q$;
2. *if $\tilde{r} : \tilde{s} \longrightarrow \tilde{s}'$, then $Q \Rightarrow^* Q'$ and $(\tilde{r} : \tilde{s}') \sim Q'$;*
3. *if $Q \Rightarrow Q'$, then $\tilde{r} : \tilde{s} \longrightarrow^* \tilde{s}'$ and $(\tilde{r} : \tilde{s}') \sim Q'$.*

We say $(\tilde{r} : \tilde{s})$ and Q are correspondent *is there exists a correspondence $\sim$ such that $(\tilde{r} : \tilde{s}) \sim Q$.*

The following theorems affirm that there is a correspondence between security protocol specifications written in MSR_P and PA_P when related via the encodings here presented.

Theorem 5.5.9 *Given an MSR_P security protocol theory C. Then $C \sim \lceil C\rceil$.*

Proof. See Appendix 5.8 ∎

Theorem 5.5.10 *Given an PA_P security protocol process Q. Then $\lfloor Q\rfloor \sim Q$.*

Proof. See Appendix 5.8 ∎

This means that any MSR_P step can be faithfully simulated by zero or more steps in PA_P through the mediation of the encoding $\lceil\_\rceil$, and vice-versa, the reverse translation $\lfloor\_\rfloor$ will map steps in PA_P into corresponding steps in MSR_P.

We conclude by observing that our encodings and Theorem 5.5.9 and 5.5.10 allow us to reason about security properties in one of either frameworks and transfer the results to the other.

5.6 Security Analysis

This section shows how our encodings preserve some security properties from one: formalisms to the other: in particular those security properties whose definitions can be expressed in terms of predicates over the intruder knowledge or the set of messages on the networks, specifically *secrecy* and *authenticity*.

5.6.1 Secrecy

A secrecy property requires that a certain message, say M, cannot be discovered by an intruder during any possible interactions with protocol participants. Generally speaking the discovery of a secrecy flaw can be performed by looking for traces where the intruder acquires knowledge of the secret. If no such trace exists, then secrecy is preserved.

In MSR_P, the formal definition of such a secrecy violation is straightforwards in our context by using the $Prj_I(\_)$ function:

Definition 5.6.1 (Secrecy violation in MSR_P) *Let be C be an MSR_P configuration of a protocol, and M be a ground message. We say that C* does not preserve the secrecy *of M if and only if*

$$\exists C'.\ C \longrightarrow^* C' \ \textit{and}\ M \in Prj_I(C')$$

Definition 5.6.1 can often be verified quite efficiently using modern model checking and theorem proving techniques [172, 43].

A secrecy flaw is defined similarly in PA_P:

Definition 5.6.2 (Secrecy violation in PA_P) *Let Q be a PA_P model of a protocol, and M be a ground message. We say that Q* does not preserve the secrecy of *M if and only if*

$$\exists Q',\ Q \Rightarrow^* Q',\ \textit{and}\ M \in Obs_I(Q')$$

Again, Definition 5.6.2 can be efficiently verified by one of the existing strategies for checking secrecy violation or secrecy preservation developed for process algebras, *e.g.*, using reachability analysis techniques [79, 29].

The main fact here is that, independently from the checking strategy chosen, our correspondence relation preserves secrecy. Indeed, the intruder knowledge in two corresponding models, an MSR_P configuration and a PA_P state respectively, is the same step by step. So whenever there is a computation that leads the intruder to discover a secret M in the MSR_P model, there shall be a computation in the PA_P model where the intruder is able to capture the same message. Then, by producing corresponding models, our encodings are able to map secrecy properties from MSR_P to PA_P and vice versa. In fact:

Proposition 5.6.3 *Let be C an MSR_P configuration and M a ground message. Then*

$$M \in Prj_I(C) \textit{ iff } M \in Obs_I(\lceil C \rceil)$$

Proof. Straightforward by Theorem 5.5.9. ∎

and

Proposition 5.6.4 *Let be Q a PA_P state and M a ground message. Then*

$$M \in Obs_I(Q) \text{ iff } M \in Prj_I(\lfloor Q \rfloor)$$

Proof. Straightforward by Theorem 5.5.10. ∎

The obvious conclusion is that secrecy is preserved by our encodings.

Theorem 5.6.5 *Let be C an MSR_P model of a protocol (*i.e., *an initial MSR_P configuration). Then for any message M, a secrecy violation (w.r.t M) happens in C if and only if a secrecy violation (w.r.t. M) happens in $\lceil C \rceil$.*

Proof. Straightforward by Theorem 5.5.9 and Proposition 5.6.3. ∎

Theorem 5.6.6 *Let be Let be Q a PA_P model of a protocol (*i.e., *an initial PA_P state). Then for any message M, a secrecy violation (with respect to M) happens in Q if and only if a secrecy violation (with respect to M) happens in $\lfloor Q \rfloor$.*

Proof. Straightforward by Theorem 5.5.10 and Proposition 5.6.4. ∎

5.6.2 Authentication

The treatment of authentication properties is a bit more delicate. There are several notions of authentication. One of the most popular techniques was introduced by Woo and Lam [206]: roles are annotated with unforgeable control actions called *assertions* that describe the state of the protocol execution from the point of view of the principal executing it: for example the initiator may use $\mathsf{begin}(L)$ to assert that the protocol has started, while the responder may assert $\mathsf{end}(L)$ when it reaches its last event. The label L uniquely identifies relevant parameters of this session (the principals involved, their role, nonces, etc.).

Generally speaking, if a protocol guarantees authentication, then in every run each $\mathsf{end}(L)$ event matches a distinct $\mathsf{begin}(L)$ event preceeding it, even in the presence of an attacker. If this is the case, we know that the initiator and the responder have a compatible view of the world. If we abstract a run as the sequence of assertions issued by all parties, this is equivalent [147] to checking that in each run the number of $\mathsf{end}(L)$ never exceeds the number of $\mathsf{begin}(L)$, for the same L.

Definition 5.6.7 *A protocol P satisfies* authenticity *if and only if for every run of the protocol and for every L, the number of $\mathsf{end}(L)$ events never exceeds the number of $\mathsf{begin}(L)$ events.*

We show how this mechanism works for detecting Lowe's attack on the $NSPK$ protocol [138]. Consider that when one user A starts to run the protocol as initiator apparently with a responder B, it sends a control message $\mathsf{begin}(\langle A, B\rangle)$. When one user B running the role of responder finishes a protocol apparently with an initiator A running the role of initiator then it sends the message $\mathsf{end}(\langle A, B\rangle)$. Ideally, if we assume that these messages are never removed from the net, the number of messages of the form $\mathsf{begin}(\langle A, B\rangle)$ must be greater than the number of messages of the form $\mathsf{end}(\langle A, B\rangle)$ at any point of any computation.

The attack is given by the following sequence of actions. We only need three users: A, B and E such that A initiates a run with a dishonest principal E who reroute it as a run with B. We write $E(A)$ to denote the intruder impersonating the agent A:

$$\begin{array}{lll} A & \longrightarrow E & : \{N_A, A\}_{K_E} \\ E(A) & \longrightarrow B & : \{N_A, A\}_{K_B} \\ B & \longrightarrow E(A) & : \{N_A, N_B\}_{K_A} \\ E & \longrightarrow A & : \{N_A, N_B\}_{K_A} \\ A & \longrightarrow E & : \{N_B\}_{K_E} \\ E(A) & \longrightarrow B & : \{N_B\}_{K_B} \end{array}$$

Principal A starts a run of the protocol with the dishonest agent E, who decrypts the transmitted values and repackages them as if they were intended for principal B. Agent B, believing he is responding to A, sends the message $\{N_A, N_B\}_{K_A}$ to E, who simply forwards it to A. This principal replies to E with the last message $\{N_A, N_B\}_{K_A}$, that E repackages for B as earlier. In the end, A correctly believes she has authenticated E, but B incorrectly assumes he has authenticated A while he was talking to E only. Woo and Lam's method reveals this failure of authentication: if we start the initiator role with the assertion $\mathsf{begin}(\langle A,\, B\rangle)$ and conclude the responder role with $\mathsf{end}(\langle A,\, B\rangle)$, we extract from the above run the trace $\{\mathsf{begin}(\langle A,\, E\rangle), \mathsf{end}(\langle A,\, B\rangle)\}$, which violates Definition 5.6.7. While this method may seem rather simple it has been shown very useful for detecting attacks on security protocols (e.g., see [137]).

A possible solution to include authenticity in our framework comes from the observation that it is possible to encode begin-end assertions through particular control messages in such a way that the observational power of our correspondence relation is enough. Since our correspondence relation "observes" only the status of the net and of the intruder knowledge, this implies that we have to find a way to record the begin-end events in either the intruder knowledge or in the network. Moreover because our notion of observation concerns sets we must face the problem of losing the number of repetitions of events in sets. Both problem can be easily solved (*e.g.*, see [147]).

The latter one, for example can be solved by introducing in each control message information that makes it unique *e.g.*, a timestamp. This information is then filtered out when used to check related begin-end events.

To solve the former problem we will develop a different strategy that consists in sending begin-end assertions over a *private network*, we call N^P. The goal of this private network is only to collect control messages for sake of verification. Moreover we assume assertions be coded as control messages $\langle \mathsf{begin},\, L\rangle$, $\langle \mathsf{end},\, L\rangle$, where the label L carries sufficient information for uniquely identify the session. Moreover we assume that L carries timestamp information that make them unique in different run of the protocol.

In MSR_P to model such a network we need a new predicate N^P. A role may assert something by sending a control message over N^P. This can be done, for example, by using the send rewriting rule. This requires a new class of *assertion rules*, similar to send rules:

$$\textit{assertion rule} \qquad A_{\rho_{i-1}}(\boldsymbol{x}) \rightarrow A_{\rho_i}(\boldsymbol{x}), N^P(\langle a,\, L(\boldsymbol{x})\rangle)$$

where $a \in \{\mathsf{begin}, \mathsf{end}\}$.

In PA_P the private network N^P is modeled by the process $!N_i^P(x).\overline{N_o^P}(x).0$, while a process's assertion is modeled by sending a message, of form either $\langle \mathsf{begin},\, L(\boldsymbol{x})\rangle$ or $\langle \mathsf{end},\, L(\boldsymbol{x})\rangle$, towards the channel N_i^P. We deal with authentication by slightly modifying our encodings to take into account the new symbols N^P. The correspondence relation needs to be modified too. We handle N^P by simply mirroring the treatment of N.

We can now define our instances of Definition 5.6.7 as in the following.

Definition 5.6.8 (Authenticity violation in MSR_P) *Let be C be an MSR_P model of a protocol (*i.e.*, an initial configuration). We say that C* violates authenticity *if and only if for some L, $\exists C'$, $C \longrightarrow^* C'$, such that in $Prj_{N^P}(C')$ the number of $\langle \mathsf{end}, L\rangle$ is greater of the number of $\langle \mathsf{begin}, L\rangle$.*

If it is the case will write $C \not\models \{\mathsf{end}(L) \hookrightarrow \mathsf{begin}(L)\}$.

Definition 5.6.9 (Authenticity violation in PA_P) *Let be Q be a PA_P model of a protocol. We say that Q* violates authenticity *if and only if for some L, $\exists Q'$, $Q \Rightarrow^* Q'$ such that in $Obs_{N^P}(Q')$ the number of $\langle \mathsf{end}, L\rangle$ if greater of the number of $\langle \mathsf{begin}, L\rangle$.*

If it is the case will write $Q \not\models \{\mathsf{end}(L) \hookrightarrow \mathsf{begin}(L)\}$.

All the results stated in Section 5.5, remain valid. Precisely because the messages stored in the network in two correspondent models, respectively an MSR_P and a PA_P, are the same step by step if there is a computation that leads to a authenticity flaw in the MSR_P model, there would be another computation in the PA_P model where the same flaw is shown, and vice versa. Then our encodings, mapping models into correspondent models, are able to map authenticity properties from MSR to PA and vice versa. The previous results can be formalized into the following propositions

Proposition 5.6.10 *Let be C an MSR_P model of a protocol and L a ground control message. Then $C \not\models \{\mathsf{end}(L) \hookrightarrow \mathsf{begin}(L)\}$ iff $\lceil C \rceil \not\models \{\mathsf{end}(L) \hookrightarrow \mathsf{begin}(L)\}$.*

Proof. Straightforward by Theorem 5.5.9. ∎

Proposition 5.6.11 *Let be Q a PA_P model of a protocol and L a ground control message. Then $Q \not\models \{\mathsf{end}(L) \hookrightarrow \mathsf{begin}(L)\}$ iff $\lfloor Q \rfloor \not\models \{\mathsf{end}(L) \hookrightarrow \mathsf{begin}(L)\}$.*

Proof. Straightforward by Theorem 5.5.10. ∎

The obvious conclusion is that authenticity is reserved by our encodings.

Theorem 5.6.12 *Let be Let be C an MSR_P model of a protocol. Then C preserves authenticity if and only if $\lceil C \rceil$ does.*

Proof. Straightforward by Theorem 5.5.9 and Proposition 5.6.10. ∎

Theorem 5.6.13 *Let be Let be Q a PA_P model of a protocol. Then Q preserves authenticity if and only if $\lfloor Q \rfloor$ does.*

Proof. Straightforward by Theorem 5.5.10 and Proposition 5.6.11. ∎

5.7 Conclusions

This chapter shows how multiset rewriting theories (MSR) and process algebras (PA) used to describe security protocols are related. We show how to define semantics preserving transformations between MSR and PA describing protocols. The correspondence relation we used, is based on which messages appear on the network and on which messages the intruder knows.

5.8 (Appendix) Theorem Proofs

This appendix provides a proof for Theorem 5.5.9 and a proof for Theorem 5.5.10.

We begin this section by reminding that a MSR_P state is a multiset of form $\tilde{s} = (\tilde{N}, \tilde{A}, \tilde{I}, \tilde{\pi})$, where the components collect ground facts $N(t)$, $A_{\rho_i}(\boldsymbol{t})$, $I(t)$ and $\pi(\boldsymbol{t})$ respectively, while a PA_P state is a process (see Proposition 5.4.1)

$$\overbrace{(P_{!net} \parallel \prod_{\rho} P_{!\rho} \parallel Q_{!I} \parallel Q_{!\pi})}^{Q_!} \parallel (Q_{net} \parallel \prod_{\rho} P_\rho \parallel Q_I \parallel Q_{rem})$$

where:

$$\begin{array}{lcl} Q_{net} & ::= & 0 \mid \prod \overline{N_o}(t).0 \\ P_\rho & ::= & 0 \mid N_o(\boldsymbol{x}).P_\rho \;\mid\; \overline{N_i}(\underline{\boldsymbol{t}}).P_\rho \;\mid\; [\underline{\boldsymbol{t}} = \boldsymbol{t'}]\, P_\rho \\ Q_I & ::= & \text{suffixes of } P_{I_j}, \text{ for all } j \\ Q_{rem} & ::= & 0 \;\mid\; N_o(x).\overline{N_i}(x).0 \;\mid\; \tilde{\pi}(\boldsymbol{x}).\nu\boldsymbol{n}.P_\rho \;\mid\; \nu\boldsymbol{n}.P_\rho \mid \;\mid\; \prod \overline{\pi}(\underline{t}).0 \end{array}$$

Moreover in the following we will use implicitly the following proposition:

Proposition 5.8.1 $\lfloor !P \parallel P \parallel Q \rfloor = \lfloor !P \parallel Q \rfloor$

Proof. It is based on the fact that $\lfloor \_ \rfloor$ maps processes P, coming from any transition $!P \Rightarrow P \parallel !P$, into the empty multiset. Formally:$\lfloor !P \parallel P \parallel Q \rfloor = \lfloor !P \rfloor, \lfloor P \rfloor, \lfloor Q \rfloor = \lfloor !P \rfloor, \cdot, \lfloor Q \rfloor = \lfloor !P \rfloor, \lfloor Q \rfloor = \lfloor !P \parallel Q \rfloor$ ∎

We now prove the following main theorem:

Main Theorem (Reminder) 1 *Given an* MSR_P *security protocol theory* C. *Then* $C \sim \lceil C \rceil$.

Proof. The proof consists in showing that

$$\mathcal{R} = \{(C, \lceil C \rceil) : C_0 \longrightarrow^* C\} \cup \{(C, Q) : C_0 \longrightarrow^* C, \lfloor Q \rfloor = C\}$$

is a correspondence $\sim$. Because of Lemma 5.5.6 and Lemma 5.5.7 it is sufficient to show that for all $(C, Q) \in \mathcal{R}$:

(I) $C \longrightarrow C'$ implies $Q \Rightarrow^* Q'$ and $(C', Q') \in \mathcal{R}$
(II) $Q \Rightarrow Q'$ implies $C \longrightarrow^* C'$ and $(C', Q') \in \mathcal{R}$.

Precisely $(C', Q') \in \mathcal{R}$ means that either $\lfloor Q' \rfloor = C'$ or $Q' = \lceil C' \rceil$.

Before explaining the technical steps of the proof, let us focus on the following question. What are the $(C', Q') \in \mathcal{R}$ that are reachable via a MSR_P or PA_P transition from $(C, Q) \in \mathcal{R}$? In other words, given a transition $C \longrightarrow C'$ (resp., $Q \Rightarrow Q'$) what transitions $\lceil C \rceil \Rightarrow^* Q'$ or $Q \Rightarrow^* Q'$ where $\lfloor Q \rfloor = C$ (resp., $\lfloor Q \rfloor \longrightarrow^* C'$ or $C \longrightarrow^* C'$ where $\lceil C \rceil = Q$) satisfy condition (**I**) (resp., condition **(II)**) above?

Let us first focus on **(I)** and on Figure 5.2. and suppose that a MSR_P transition $C \longrightarrow C'$ occurs. Via $\lceil \_ \rceil$ the only possibile PA_P transition is $\lceil C \rceil \Rightarrow^* \lceil C' \rceil$ (*e.g.,* states Q and Q' and the relative $Q \Rightarrow^* Q'$ transition in Figure 5.2). Instead via $\lfloor \_ \rfloor$, more transitions $Q \Rightarrow^* Q'$ are

possible; precisely all those such that $\lfloor Q \rfloor = C$ and $\lfloor Q' \rfloor = C'$ (*e.g.*, processes Q''', Q and Q'' in Figure 5.2 and transitions $Q''' \Rightarrow^* Q'$, $Q \Rightarrow^* Q'$ and $Q'' \Rightarrow^* Q'$).

Let now focus on **(II)** and on Figure 5.2 again. Let suppose a PA_P transition $Q \Rightarrow Q'$ occurs. Here it may be that the only couple (C', Q') corresponding in $\mathcal{R}$, via either $\lfloor \_ \rfloor$ or $\lceil \_ \rceil$, to (C, Q) is such that $C = C'$. This happens when transition $Q \Rightarrow Q'$ is not able to simulate any complete MSR_P step (*e.g.*, as the transition $Q \Rightarrow Q''$ and its correspondent $C \longrightarrow^* C$, in Figure 5.2).

Proof of Part (I). The scheme which guides the proof of this part, is the following:

$$\textbf{(I)} \quad C \longrightarrow C' \text{ implies } \begin{array}{l} (a)\ \lceil C \rceil \Rightarrow^* Q' \text{ and } (C', Q') \in \mathcal{R} \\ (b)\ \forall Q : \lfloor Q \rfloor = C, Q \Rightarrow^* Q' \text{ and } (C', Q') \in \mathcal{R} \end{array} \tag{5.8.1}$$

In the following we will itemize each sub-case with $(I.a)$, $(I.a')$, etc., or $(I.b)$ $(I.b')$, etc., depending on it is respectively the first, second, etc., sub-case of branches (a) or (b) of (5.8.1); moreover let us observe that, because $\lfloor \lceil C \rceil \rfloor = C$ (see Lemma 5.5.5)

$$\{(C, \lceil C \rceil) : C_0 \longrightarrow^* C\} \cap \{(C, Q) : C_0 \longrightarrow^* C, \lfloor Q \rfloor = C\} \neq \emptyset$$

As a consequence some sub-cases of (b) will coincide with some sub-case of (a). Precisely those that do really differ, are those involving pairs $(\lfloor Q \rfloor, Q)$ such that $Q \neq \lceil C \rceil$; to avoid repetitions we will treat in $(I.b)$ only those cases that differ from cases in $(I.a)$.

Let be C' such that $C \longrightarrow C'$. It must have happened as a consequence of an application of either a rewriting rule r_{ρ_0}, r_{ρ_i} send or r_{ρ_i} receive or r_{ρ_i} analysis for $i > 0, \ldots, l_\rho$ or finally an intruder rule r_{I_j} for $j = 0, \ldots, 9$. We will treat each rule separately. We also remind that for each rule we will list different sub-cases $(I.a)$ and $(I.b)$.

- **(instantiation rule)** $r_{\rho_0} = \tilde{\pi}(\boldsymbol{x}) \rightarrow \exists \boldsymbol{n}.A_{\rho_0}(\boldsymbol{n}, \boldsymbol{x}), \tilde{\pi}(\boldsymbol{x})$

In this case transition $C \longrightarrow C'$ can be specifically rewritten as:

$$\begin{array}{rcl} C & = & \tilde{\pi}(\underline{\boldsymbol{k}}), C'' \\ & \rightarrow & A_{\rho_0}[\overbrace{\underline{\boldsymbol{k}}/\boldsymbol{x}; \underline{\boldsymbol{m}}/\boldsymbol{n}}^{\theta}], \tilde{\pi}(\underline{\boldsymbol{k}}), C'' \\ & = & \underbrace{A_{\rho_0}(\underline{\boldsymbol{k}}; \underline{\boldsymbol{m}}), C}_{C'} \end{array}$$

where, we remind, $\tilde{\pi}(\underline{\boldsymbol{k}})$ is an abbreviation for $\pi(\underline{\boldsymbol{k}}_1), \cdots, \pi(\underline{\boldsymbol{k}}_r)$ where $\underline{\boldsymbol{k}}_i$ for all i, are all ground tuples of terms.

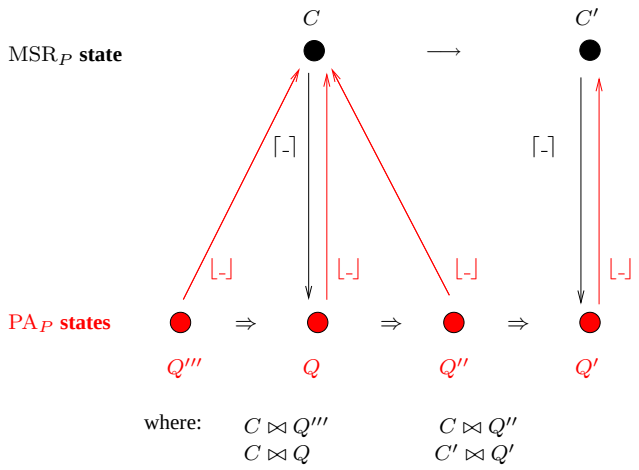


Figure 5.2: A possible scenario involving corresponding couples (C, Q) and (C', Q') in $\mathcal{R}$, when it occurs either a transition $C \longrightarrow C'$ or a transition $Q \Rightarrow Q'$.

$\star$ Case $(I.a)$: $(C, Q) = (C, \lceil C \rceil)$. We have:

$$
\begin{array}{rcll}
\lceil C \rceil & = & \overbrace{!\overline{\tilde{\pi}}(\underline{\boldsymbol{k}}).0}^{\lceil \tilde{\pi}(\underline{\boldsymbol{k}}) \rceil} \parallel \underbrace{!\tilde{\pi}(\boldsymbol{x}).\nu \boldsymbol{n}.\ \overbrace{P_\rho}^{\lceil r_{\rho_1} \rceil^{\#}_{(\boldsymbol{x};\boldsymbol{n})}} \parallel Q''}_{\lceil C'' \rceil} & \text{[def. of } \lceil \_ \rceil] \\
& \equiv & \overline{\tilde{\pi}}(\underline{\boldsymbol{k}}).0 \parallel \tilde{\pi}(\boldsymbol{x}).\nu \boldsymbol{n}.P_\rho \parallel \underbrace{!\overline{\tilde{\pi}}(\underline{\boldsymbol{k}}).0 \parallel !\tilde{\pi}(\boldsymbol{x}).\nu \boldsymbol{n}.P_\rho \parallel Q''}_{\lceil C \rceil} & \\
& \Rightarrow^* & \underbrace{0 \parallel P_\rho[\theta] \parallel \lceil C \rceil}_{Q'} & [pa_0, pa_{\equiv}, pa_\nu] \\
& = & 0 \parallel \lceil r_{\rho_1} \rceil^{\#}_{(\boldsymbol{x};\boldsymbol{n})}[\theta] \parallel \lceil C \rceil & \\
& = & 0 \parallel \lceil A_{\rho_0}(\underline{\boldsymbol{k}}; \underline{\boldsymbol{m}}) \rceil \parallel \lceil C \rceil & \text{[def. of } \lceil A_{\rho_i}(\boldsymbol{t}) \rceil] \\
& \equiv & \lceil A_{\rho_0}(\underline{\boldsymbol{k}}; \underline{\boldsymbol{m}}) \rceil \parallel \lceil C \rceil & \\
& = & \lceil C' \rceil &
\end{array}
$$

$\star$ Case $(I.b)$: $(C, Q) = (\lfloor Q \rfloor, Q)$. We need to identify those Q's such that $\lfloor Q \rfloor = C = \tilde{\pi}(\underline{\boldsymbol{k}}), C''$. The only different case, w.r.t. $(I.a)$, (indeed a family of cases) happen when

$$
Q = \left(\prod_{i=m}^{r} \overline{\pi}(\underline{k}_i).0\right) \parallel \pi_m(x_m).\cdots.\pi_r(x_r).\nu \boldsymbol{n}.P_\rho[\theta'] \parallel \lceil C \rceil
$$

where $\theta' = [\underline{k}_1/x_1, \cdots, \underline{k}_{m-1}/x_{m-1}]$. In words, Q is a partially instantiated role that has already started receiving its permanent terms, but not all. It is worth to underline that both $\prod_{i=m,\ldots,r} \overline{\pi}(\underline{k}_i).0$ and $\pi_m(x_m).\cdots.\pi_r(x_r).\nu \boldsymbol{n}.P_\rho[\theta']$ are mapped by $\lfloor \_ \rfloor$ into the empty multi-

set; as a consequence $\lfloor Q \rfloor = C$. Let now observe that:

$$\begin{array}{rcll} Q & = & \begin{array}{l}\left(\prod_{i=m,\ldots,r} \overline{\pi}(\underline{k}_i).0\right) \parallel \\ \pi_m(x_m).\cdots.\pi_r(x_r).\nu\boldsymbol{n}.P_\rho[\theta'] \\ \parallel \lceil C \rceil\end{array} & \\ & \Rightarrow^* & 0 \parallel P_\rho[\theta] \parallel \lceil C \rceil & [pa_0 \text{ and } pa_\nu \text{ with } \underline{\boldsymbol{m}} \text{ as new names}] \\ & \equiv & \underbrace{P_\rho[\theta] \parallel \lceil C \rceil}_{Q'} & \end{array}$$

and it easy to check that $\lceil C' \rceil = Q'$.

- **(send rule)** $r_{\rho_i} = A_{\rho_{i-1}}(\boldsymbol{x}) \rightarrow A_{\rho_i}(\boldsymbol{x}), N(t(\boldsymbol{x}))$

  In this case transition $C \longrightarrow C'$ can be specifically rewritten as:

$$C = A_{\rho_{i-1}}(\underline{\boldsymbol{x}[\theta]}), C'' \rightarrow \underbrace{A_{\rho_i}(\underline{\boldsymbol{x}[\theta]}), N(\underline{t[\theta]}), C''}_{C'} \tag{5.8.2}$$

where θ is the substitution that allows the rule r_{ρ_i} to be applied. The only significative situation happens as a sub-case of statement (a) of (5.8.1).
$\star$ Case $(I.a)$: $(C, Q) = (C, \lceil C \rceil)$. We have:

$$\begin{array}{rcll} \lceil C \rceil & = & \lceil r_{\rho_i} \rceil^{\#}_{(\boldsymbol{x})}[\theta] \parallel \lceil C'' \rceil & [\text{def. of } \lceil A_{\rho_{i-1}}(\underline{\boldsymbol{x}[\theta]}) \rceil] \\ & = & \overline{N_i}(t[\theta]).\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x})}[\theta] \parallel \lceil C'' \rceil & [\text{unfolding } \lceil r_{\rho_i} \rceil^{\#}_{(\boldsymbol{x})}[\theta]] \\ & = & \underbrace{\begin{array}{l}\overline{N_i}(\underline{t[\theta]}).\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x})}[\theta] \parallel \\ !N_i(x).\overline{N_o}(x).0 \parallel \lceil C''' \rceil\end{array}}_{\lceil C'' \rceil} & [\text{def. of } P_{!net} \text{ in } \lceil C'' \rceil] \\ & \equiv & \underbrace{\begin{array}{l}\overline{N_i}(\underline{t[\theta]}).\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x})}[\theta] \\ \parallel N_i(x).\overline{N_o}(x).0 \parallel \\ !N_i(x).\overline{N_o}(x).0 \parallel \lceil C''' \rceil\end{array}}_{\lceil C'' \rceil} & \\ & \Rightarrow & \underbrace{\overbrace{\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x})}[\theta]}^{\lceil A_{\rho_i}(\underline{\boldsymbol{x}[\theta]}) \rceil} \parallel \overbrace{\overline{N_o}(\underline{t[\theta]}).0}^{\lceil N(\underline{t[\theta]}) \rceil} \parallel \lceil C'' \rceil}_{Q'} & [\text{def. of } pa_0] \\ & = & \lceil C' \rceil & \end{array}$$

- **(receive rule)** $r_{\rho_i} = A_{\rho_{i-1}}(\boldsymbol{x}), N(y) \longrightarrow A_{\rho_i}(\boldsymbol{x}; y)$

  In this case transition $C \longrightarrow C'$ can be specifically rewritten as:

$$C = A_{\rho_{i-1}}(\underline{\boldsymbol{x}[\theta]}), N(\underline{t}), C'' \longrightarrow \underbrace{A_{\rho_i}(\underline{\boldsymbol{x}[\theta]}; y[\underline{t}/y]), C''}_{C'} \tag{5.8.3}$$

where θ is the substitution that allows the rule r_{ρ_i} to be applied. Again the only significative case happens as a sub-case of class (a) in statement (5.8.1).
$\star$ Case $(I.a)$: $(C, Q) = (C, \lceil C \rceil)$. We have:

$$\begin{array}{rcll} \lceil C \rceil & = & \lceil r_{\rho_i} \rceil^{\#}_{(\boldsymbol{x})}[\theta] \parallel \overline{N_o}(\underline{t}).0 \parallel \lceil C'' \rceil & \text{[def. of } \lceil A_{\rho_{i-1}}(\underline{\boldsymbol{x}[\theta]}) \rceil\text{]} \\ & = & N_o(y).\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x};y)}[\theta] \parallel \overline{N_o}(\underline{t}).0 \parallel \lceil C'' \rceil & \text{[espanding } \lceil r_{\rho_i} \rceil^{\#}_{(\boldsymbol{x})}[\theta]\text{]} \\ & \Rightarrow & \underbrace{\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x};y)}[\theta][\underline{t}/y] \parallel 0 \parallel \lceil C'' \rceil}_{Q'} & [pa_0] \\ & = & \lceil C' \rceil & \end{array}$$

- **(analysis rule)** $r_{\rho_i} = A_{\rho_{i-1}}(\boldsymbol{t}(\boldsymbol{x})) \longrightarrow A_{\rho_i}(\boldsymbol{x})$.

  In this case transition $C \longrightarrow C'$ can be specifically rewritten as:

$$C = A_{\rho_{i-1}}(\underline{\boldsymbol{t}(\boldsymbol{x})[\theta']}), C'' \longrightarrow \underbrace{A_{\rho_i}(\underline{\boldsymbol{x}[\theta']}), C''}_{C'} \tag{5.8.4}$$

Again the only interesting scenario comes from sub-case (a) of (5.8.1). While analyzing this case let us:

- rewrite the ground term $\underline{\boldsymbol{t}(\boldsymbol{x})[\theta']}$ as $\underline{\boldsymbol{k}}$;
- assume that the consequent predicate of rule $r_{\rho_{i-1}}$ is $A_{\rho_{i-1}}(\boldsymbol{x'})$, *i.e.*, rule $r_{\rho_{i-1}} = \ldots \longrightarrow A_{\rho_{i-1}}(\boldsymbol{x'})$.
- assume θ be the unifier such that $\boldsymbol{x'}[\theta] = \underline{\boldsymbol{k}}$, that is the substitution that unifies the predicate $A_{\rho_{i-1}}(\boldsymbol{x'})$ with the ground predicate $A_{\rho_{i-1}}(\underline{\boldsymbol{k}})$ in the MSR_P state C.

$\star$ Case $(I.a)$: $(C, Q) = (C, \lceil C \rceil)$. We have:

$$\begin{array}{rcll} \lceil C \rceil & = & \lceil r_{\rho_i} \rceil^{\#}_{(\boldsymbol{x'})}[\theta] \parallel \lceil C'' \rceil & \\ & = & [\overbrace{\boldsymbol{x'}[\theta]}^{\underline{\boldsymbol{k}}} = \boldsymbol{t}(\boldsymbol{x})[\theta]].\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x})}[\theta] \parallel \lceil C'' \rceil & \text{[def. of } \lceil r_{\rho_i} \rceil^{\#}_{(\boldsymbol{x'})}\text{]} \\ & \Rightarrow & \lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x})}[\theta][\theta''] \parallel \lceil C'' \rceil & [pa_{[]}\text{, and } \boldsymbol{t}(\boldsymbol{x})[\theta][\theta''] = \underline{\boldsymbol{k}}] \\ & = & \underbrace{\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x})}[\theta'] \parallel \lceil C'' \rceil}_{Q'} & \text{[(see text below)]} \\ & = & \lceil C' \rceil & \end{array}$$

Note that here, θ' can be used instead of $\theta\theta''$ because θ' and $\theta\theta''$ coincide on $\boldsymbol{x}$, that in turn are all the variables appearing in $\lceil r_{\rho_{i+1}} \rceil$.

- **(intruder rules)** r_{I_j}, for $j = 0, \ldots, 9$.

Let us consider just a significative rule, for example rule $r_{I_6} = I(x_1), I(x_2) \rightarrow I(\langle x_1, x_2 \rangle)$, $I(x_1), I(x_2)$. The proofs for the other intruder's rules are similar. In this case transition $C \longrightarrow C'$ can be specifically rewritten as:

$$C = I(\underline{t_1}), I(\underline{t_2}), C'' \longrightarrow \underbrace{I(\langle \underline{t_1},\, \underline{t_2}\rangle), I(\underline{t_1}), I(\underline{t_2}), C''}_{C'}. \tag{5.8.5}$$

$\star$ Case $(I.a)$: $(C, Q) = (C, \lceil C \rceil)$. Then we have:

$$\begin{array}{rcll}
\lceil C \rceil & = & \overbrace{\overline{I}(\underline{t_1}).0}^{\lceil I(\underline{t_1})\rceil^I} \parallel \overbrace{\overline{I}(\underline{t_2}).0}^{\lceil I(\underline{t_2})\rceil^I} \parallel \lceil C'' \rceil & \text{[def. of } \lceil _ \rceil] \\
& = & \overline{I}(\underline{t_1}).0 \parallel \overline{I}(\underline{t_2}).0 \parallel \underbrace{Q_{!I} \parallel Q''}_{\lceil C'' \rceil} & \text{[expanding } PA_P \text{ state]} \\
& \equiv & \begin{array}{l} \overline{I}(\underline{t_1}).0 \parallel \overline{I}(\underline{t_2}).0 \\ \parallel I(x_1).\overline{I}(x_1).I(x_2).\overline{I}(x_2).\overline{I}(\langle x_1,\, x_2\rangle).0 \\ \parallel I(x).\overline{I}(x).0 \parallel I(x).\overline{I}(x).0 \\ \parallel \lceil C'' \rceil \end{array} & \text{[expanding } Q_{!I} \text{ (} P_{I_6} \text{ and } P_{I_{10}}\text{)]} \\
& \Rightarrow^* & \underbrace{0 \parallel \overbrace{\overline{I}(\langle \underline{t_1},\, \underline{t_2}\rangle).0)}^{\lceil I(\langle \underline{t_1}, \underline{t_2}\rangle)\rceil^I} \parallel \overbrace{\overline{I}(\underline{t_1}).0}^{\lceil I(\underline{t_1})\rceil^I} \parallel \overbrace{\overline{I}(\underline{t_2}).0}^{\lceil I(\underline{t_2})\rceil^I} \parallel \lceil C'' \rceil}_{Q'} & [pa_0] \\
& = & \lceil C' \rceil &
\end{array}$$

Let now start analyzing the case $(C, Q) = (\lfloor Q \rfloor, Q)$. We need to identify those Q's such that $\lfloor Q \rfloor = I(\underline{t_1}), I(\underline{t_2}), C''$. In fact, more different Q's (precisely different Q_I) exist, for the non injective $\lfloor \_ \rfloor_I$ is now involved in the translation (see also Figure 5.1). In addition, we remind, the only really significative (w.r.t. case $(I.a)$) situations are those ones where Q's are such that $Q \neq \lceil C \rceil$

$\star$ Case $(I.b')$: a first case happens when Q contains both the process $\overline{I}(\underline{t_2}).0$ and the proper suffix of P_{I_6}, $\overline{I}(\underline{t_1}).I(x_2).\overline{I}(x_2).\overline{I}(\langle \underline{t_1},\, x_2\rangle).0$.

$$\begin{array}{rcll}
Q & = & \overline{I}(\underline{t_2}).0 \parallel \overline{I}(\underline{t_1}).I(x_2).\overline{I}(x_2).\overline{I}(\langle \underline{t_1},\, x_2\rangle).0 \parallel \lceil C'' \rceil & \\
& \equiv & \begin{array}{l} \overline{I}(\underline{t_2}).0 \\ \parallel \overline{I}(\underline{t_1}).I(x_2).\overline{I}(x_2).\overline{I}(\langle \underline{t_1},\, x_2\rangle).0 \\ \parallel I(x).\overline{I}(x).0 \parallel I(x).\overline{I}(x).0 \\ \parallel \lceil C'' \rceil \end{array} & \text{[espanding } Q_{!I}\text{]} \\
& \Rightarrow^* & \underbrace{0 \parallel \overline{I}(\langle \underline{t_1},\, \underline{t_2}\rangle).0 \parallel \overline{I}(\underline{t_1}).0 \parallel \overline{I}(\underline{t_2}).0 \parallel \lceil C'' \rceil}_{Q'} & [pa_0]
\end{array}$$

and it is easy to verify that $\lfloor Q' \rfloor = C'$.

$\star$ Case $(I.b'')$: a second case happens when Q is $[\langle \underline{t_1},\, \underline{t_2}\rangle = \langle x_1,\, x_2\rangle].\overline{I}(x_1).\overline{I}(x_2).0 \parallel \lceil C'' \rceil$. In words Q contains a proper suffix of process P_{I_5}, standing for the intruder that has already acquired the message $\langle \underline{t_1},\, \underline{t_2}\rangle$, but that has not yet performed the output in which it splits it. We remind that in this case $\lfloor \_ \rfloor_I$ translates the process as it would have already performed the outputs, obtaining the predicates $I(\underline{t_1})$, $I(\underline{t_2})$. Then we have:

$$
\begin{array}{rcll}
Q & = & [\langle \underline{t_1},\, \underline{t_2}\rangle = \langle x_1,\, x_2\rangle].\overline{I}(x_1).\overline{I}(x_2).0 \parallel \lceil C''\rceil & \\
 & \equiv & [\langle \underline{t_1},\, \underline{t_2}\rangle = \langle x_1,\, x_2\rangle].\overline{I}(x_1).\overline{I}(x_2).0 & \\
 & & \parallel I(x).\overline{I}(x).0 \parallel \lceil C''\rceil & \text{[from } !P_{I_{10}}; \\
 & & & pa_!] \\
 & \Rightarrow & \overline{I}(\underline{t_1}).\overline{I}(\underline{t_2}).0 \parallel I(x).\overline{I}(x).0 \parallel \lceil C''\rceil & [pa_{[]}] \\
 & \Rightarrow & \overline{I}(\underline{t_2}).0 \parallel \overline{I}(\underline{t_1}).0 \parallel Q'' & [pa_0] \\
 & \Rightarrow^* & \underbrace{0 \parallel \overline{I}(\langle \underline{t_1},\, \underline{t_2}\rangle).0 \parallel \overline{I}(\underline{t_1}).0 \parallel \overline{I}(\underline{t_2}).0 \parallel Q''}_{Q'} & \text{[see Case } (I.a)]
\end{array}
$$

and it is easy to verify that $\lfloor Q' \rfloor = C'$.
$\star$ Case $(I.b''')$: the last case is when $Q = \overline{I}(\underline{t_1}).\overline{I}(\underline{t_2}).0 \parallel \lceil C''\rceil$, where again a suffix of P_{I_5} is involved. This case is simply a sub-case of the previous one.

Here ends the proof of **(I)**, where we have shown that for every $(C, Q) \in \mathcal{R}$ $C \longrightarrow C'$ implies $Q \Rightarrow^* Q'$, and $(C', Q') \in \mathcal{R}$.

Proof of Part (II). The scheme which guides the proof of this part is the following:

$$\textbf{(II)} \quad \forall (C, Q) \in \mathcal{R}, \qquad Q \Rightarrow Q' \quad \text{implies} \quad C \longrightarrow^* C' \text{ and } (C', Q') \in \mathcal{R}$$

Because, we remind, $\mathcal{R} = \{(C, \lceil C\rceil) : C_0 \longrightarrow^* C\} \cup \{(C, Q) : C_0 \longrightarrow^* C, \lfloor Q \rfloor = C\}$, the previous statement can be specifically restated as:

$$
\begin{array}{l}
\forall (C, Q) \in \mathcal{R}, \\
(a)\ \lceil C\rceil \Rightarrow Q' \text{ implies } C \longrightarrow^* C' \text{ and } (C', Q') \in \mathcal{R} \\
(b)\ \forall Q : \lfloor Q \rfloor = C, Q \Rightarrow Q' \text{ implies } C \longrightarrow^* C' \text{ and } (C', Q') \in \mathcal{R}
\end{array}
\tag{5.8.6}
$$

where $(C', Q') \in \mathcal{R}$ means that either $\lfloor Q' \rfloor = C'$ or $Q' = \lceil C'\rceil$. In the following we treat a list of cases. Each case corresponds to a possible $\Rightarrow$ transition. Again we will itemize each sub-case with $(II.a)$, $(II.a')$, etc., or $(II.b)$ $(II.b')$, etc., depending on it is respectively the first, second, etc., sub-case of branches (a) or (b) of (5.8.6).

• **(pa_0: *i.e.*, communication transition)**

Reasoning about pa_0, we must distinguish among the name of the channel a involved in the reaction *i.e.*, $a = N_i, N_o, \pi, I$. Let us discuss each case separately.

($a = N_i$) Here we treat with transitions that involve channel N_i.

$\star$ Case $(II.a)$: $(C, Q) = (C, \lceil C\rceil)$.

This case may happens when $C = A_{\rho_{i-1}}(\underline{\boldsymbol{x}[\theta]}), C''$ and $r_{\rho_i} : A_{\rho_{i-1}}(\boldsymbol{x}) \longrightarrow A_{\rho_{i-1}}(\boldsymbol{x}), N(t(\boldsymbol{x}))$.

In this case transition $\lceil C \rceil \Rightarrow Q'$ can be specifically rewritten as:

$$\begin{array}{rcl} \lceil C \rceil & = & \overbrace{\overline{N_i}(\underline{t(\boldsymbol{x})[\theta]}).\quad \overbrace{P_\rho[\theta]}^{\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x})}[\theta]}}^{A_{\rho_{i-1}}(\underline{\boldsymbol{x}[\theta]})} \quad \parallel \lceil C'' \rceil \\ & \equiv & \overline{N_i}(\underline{t(\boldsymbol{x})[\theta]}).P_\rho[\theta] \parallel N_i(x).\overline{N}(x).0 \parallel \lceil C'' \rceil \quad \text{[expanding PA}_P\text{ state]} \\ & \Rightarrow & \underbrace{P_\rho[\theta] \parallel \overline{N}(\underline{t(\boldsymbol{x})[\theta]}).0 \parallel \lceil C'' \rceil}_{Q'} \end{array}$$

Then we have:

$$C = A_{\rho_{i-1}}(\underline{\boldsymbol{x}[\theta]}), C'' \longrightarrow \underbrace{N(\underline{t(\boldsymbol{x})[\theta]}), C''}_{C'} \quad [r_{\rho_{i+1}}]$$

and it is easy to check that $\lceil C' \rceil = Q'$.

$\star$ Case $(II.b)$: $(C, Q) = (\lfloor Q \rfloor, Q)$. The only different case in this sub-part happens when $\lfloor Q \rfloor = I(\underline{t}), \lfloor Q'' \rfloor$. We observe that a Q producing such a MSR_P state is the following:

$$Q = \overline{I}(\underline{t}).0 \parallel \overline{N_i}(\underline{t'}).0 \parallel N_i(x).\overline{N_o}(x).0 \parallel Q''$$

where $\overline{N_i}(\underline{t}).0$ is an intruder partial suffix of $P_{I_4} = I(x).\overline{N_i}(x).0$. We remind that $\overline{N_i}(\underline{t}).0$ and $N_i(x).\overline{N_o}(x).0$ are mapped, by $\lfloor \_ \rfloor$, onto the empty multiset.

Let observe that transition $Q \Rightarrow Q'$ can be specifically rewritten as:

$$\begin{array}{rcl} Q & = & \overline{I}(\underline{t}).0 \parallel \overline{N_i}(\underline{t}).0 \parallel N_i(x).\overline{N_o}(x).0 \parallel Q'' \\ & \Rightarrow & \underbrace{\overline{I}(\underline{t}).0 \parallel 0 \parallel \overline{N_o}(\underline{t}).0 \parallel Q''}_{Q'} \end{array}$$

Then we have:

$$\lfloor Q \rfloor = I(\underline{t}), \lfloor Q'' \rfloor \longrightarrow \underbrace{I(\underline{t}), N(\underline{t}), \lfloor Q'' \rfloor}_{C'} \quad [\text{by } r_{I_4}]$$

and it is easy to check that $C' = \lfloor Q' \rfloor$.

($a = N_o$) Here we treat with transitions that involve channel N_o.

$\star$ Case $(II.a)$: $(C, Q) = (C, \lceil C \rceil)$. This case happens when $C = N(\underline{t}), A_{\rho_{i-1}}(\underline{\boldsymbol{x}[\theta]}), C''$ and $r_{\rho_i} : A_{\rho_{i-1}}(\boldsymbol{x}), N(y) \longrightarrow A_{\rho_{i-1}}(\boldsymbol{x}; y)$. In this case transition $\lceil C \rceil \Rightarrow Q'$ can be specifically rewritten as:

$$\begin{array}{rcl} \lceil C \rceil & = & \overline{N_o}(\underline{t}).0 \parallel N_o(y).\quad \overbrace{P_\rho[\theta]}^{\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x})}[\theta]} \quad \parallel \lceil C'' \rceil \\ & \Rightarrow & \underbrace{0 \parallel P_\rho[\theta][\underline{t}/y] \parallel \lceil C'' \rceil}_{Q'} \end{array}$$

Then we have:

$$C = N(\underline{t}), A_{\rho_{i-1}}(\underline{\boldsymbol{x}[\theta]}), C'' \longrightarrow \underbrace{A_{\rho_{i-1}}(\boldsymbol{x}; y)[\theta][\underline{t}/y], C''}_{C'} \quad \text{[by } r_{\rho_i}\text{]}$$

and it is easy to check that $C' = \lfloor Q' \rfloor$.

$\star$ Case $(II.a')$. Another case of this class happen when $C = N(\underline{t}), C''$ and $r_{I_3} = N(x) \longrightarrow I(x)$. Let observe that transition $\lceil C \rceil \Rightarrow Q'$ can be specifically rewritten as:

$$\begin{array}{rcl} \lceil C \rceil & = & \overline{N_o}(\underline{t}).0 \parallel \lceil C'' \rceil \\ & \equiv & \overline{N_o}(\underline{t}).0 \parallel N_o(x).\overline{I}(x).0 \parallel \lceil C'' \rceil \quad \text{[expanding } P_{!I}\text{]} \\ & \Rightarrow & \underbrace{\overline{I}(\underline{t}).0 \parallel \lceil C'' \rceil}_{Q'} \end{array}$$

Then we have:

$$C = N(\underline{t}), C'' \longrightarrow \underbrace{I(\underline{t}), C''}_{C'} \quad \text{[by } r_{I_3}\text{]}$$

and it is easy to check that $C' = \lfloor Q' \rfloor$.

($a = \pi$) Here we will treat with transitions that involve channel π's.

$\star$ Case $(II.a)$: $(C, Q) = (C, \lceil C \rceil)$. The only interesting scenario in this sub-case happens when in C no role predicates, w.r.t. a role ρ are yet produced and when $r_{\rho_0} = \tilde{\pi}(\boldsymbol{t}(\boldsymbol{x})) \longrightarrow \exists \boldsymbol{n}.A_{\rho_0}(\boldsymbol{x}; \boldsymbol{n})$. Let observe that transition $\lceil C \rceil \Rightarrow Q'$ can be specifically rewritten as:

$$\begin{array}{rcl} \lceil C \rceil & = & P_{!\rho} \parallel Q_{!\pi} \parallel \lceil C'' \rceil \\ & \equiv & \overbrace{\tilde{\pi}(\boldsymbol{t})}^{\pi_1(x_1).\cdots.\pi_k(x_k))} .\nu\boldsymbol{n}. \overbrace{P_\rho}^{\lceil r_{\rho_1} \rceil^{\#}_{(\boldsymbol{x};\boldsymbol{n})}} \parallel !\overline{\pi_1}(\underline{t}).0 \parallel \lceil C \rceil \quad \text{[by expanding } Q_{!\pi}\text{, } P_{!\rho}\text{]} \\ & \Rightarrow & \underbrace{\pi_2(x_2).\cdots.\pi_k(x_k).\nu\boldsymbol{n}.P_\rho[\underline{\boldsymbol{t}_0}/x_1] \parallel \lceil C \rceil}_{Q'} \end{array}$$

At this point, by observing that process $\pi_2(\boldsymbol{t}_2).\cdots.\pi_k(\boldsymbol{t}_k).\nu\boldsymbol{n}.P_\rho[\underline{t_0}/t_1].$ is indeed one that is considered garbage by the $\lfloor \_ \rfloor$ (*i.e.,* it is mapped into the empty multiset) it is easy to check that $\lfloor Q' \rfloor = C$, and we conclude observing that $C \longrightarrow^* C$ is a possible transition [2].

$\star$ Case $(II.a')$. Another sub-case happens when intruder is involved. Specifically when $\lceil C \rceil = Q_{!\pi} \parallel Q_{!I} \parallel \lceil C'' \rceil$ and transition $\lceil C \rceil \Rightarrow Q'$ may be istantiated as:

$$\begin{array}{rcl} \lceil C \rceil & = & Q_{!\pi} \parallel Q_{!I} \parallel \lceil C'' \rceil \\ & \equiv & \overline{\pi}(\underline{t}).0 \parallel \pi(x).\overline{I}(x).0 \parallel \lceil C'' \rceil \\ & \Rightarrow & \underbrace{0 \parallel \overline{I}(\underline{t}).0 \parallel \lceil C'' \rceil}_{Q'} \end{array}$$

[2] Note that the particular case where $\lceil C \rceil = \nu n.P_\rho$ *i.e.,* is part of the case pa_ν.

Then we have:

$$C = \pi(\underline{t}), C'' \longrightarrow \underbrace{\pi(\underline{t}), I(\underline{t}), \lceil C'' \rceil}_{C'} \quad [\text{by } r_{I_1}]$$

and it is easy to check that $C' = \lfloor Q' \rfloor$.

$\star$ Case $(II.b)$: $(C, Q) = (\lfloor Q \rfloor, Q)$. The only interesting cases in this side, arise by considering those Q's such that $\lfloor Q \rfloor = C$, for some $C : C_0 \longrightarrow^* C$. In fact, if C contains no role predicates, w.r.t. a role ρ, every Q containing only partial instantiations of that role (*i.e.*, processes starting with a π or ν that are suffix of P_ρ) is such that $\lfloor Q \rfloor = C$. Treating this class of case as a one general case, the transition $Q \Rightarrow Q'$ can be written as:

$$\begin{array}{rcl} Q & = & \pi_j(x_j).\cdots.\pi_k(x_k).\nu\boldsymbol{n}.P_\rho \parallel \overline{\pi_j}(\underline{t}).0 \parallel \lfloor C'' \rfloor \qquad [j > 1] \\ & \Rightarrow & \underbrace{\pi_{j+1}(x_{j+1}).\cdots.\pi_k(x_k).\nu\boldsymbol{n}.P_\rho[\underline{t}/x_j].0 \parallel \lfloor C'' \rfloor}_{Q'} \end{array}$$

Note that despite this transition, $\lfloor Q' \rfloor = C$ still hold. In fact partial instantiated (role) processes are mapped onto the empty multiset. Then we conclude observing that $C \longrightarrow^* C$ is a possible transition.

(a = I) Here we treat with transitions that involve channel I. When the intruder channel I is involved, many different situations involving the intruder arise. Here we will treat just some of the most significative ones *i.e.*, those involving the states in Figure 5.1. The others can be analyzed in a similar way.

$\star$ Case $(II.a)$: $(C, Q) = (C, \lceil C \rceil)$. A sub-case of this class happens when $C = I(\underline{t_1}), I(\underline{t_2}), C''$. We start observing that transition $\lceil C \rceil \Rightarrow Q'$ can be written as:

$$\begin{array}{rcll} \lceil C \rceil & = & \overline{I}(\underline{t_1}).0 \parallel \overline{I}(\underline{t_2}).0 \parallel \lceil C'' \rceil & \\ & \equiv & \begin{array}{l} \overline{I}(\underline{t_1}).0 \parallel \overline{I}(\underline{t_2}).0 \parallel \\ I(x_1).\overline{I}(x_1).I(x_2).\overline{I}(x_2).\overline{I}(\langle x_1, x_2 \rangle).0 \\ \parallel \lceil C'' \rceil \end{array} & [\text{expanding PA}_P \text{ state}] \\ & \Rightarrow & \underbrace{\begin{array}{l} 0 \parallel \overline{I}(\underline{t_2}).0 \parallel \\ \overline{I}(\underline{t_1}).I(x_2).\overline{I}(x_2).\overline{I}(\langle \underline{t_1}, x_2 \rangle).0 \\ \parallel \lceil C'' \rceil \end{array}}_{Q'} & [\text{expanding PA}_P \text{ state}] \end{array}$$

Note that despite this transition, $\lfloor Q' \rfloor = C$ still holds. In fact partial instantiated (role) processes are mapped onto the empty multiset. Then we conclude observing that $C \longrightarrow^* C$ is a possible transition.

No more interesting cases fall in this class. On the contrary, many cases arise when considering situation in class (b) *i.e.*, those Q such that $\lfloor Q \rfloor = C = I(\underline{t_1}), I(\underline{t_2}), C''$.

$\star$ Cases $(II.b), (II.b'), (II.b'')$: $(C, Q) = (\lfloor Q \rfloor, Q')$. Let us consider the following proce sses (see also Figure 5.1)

$$Q_1 = \overline{I}(\underline{t_1}).0 \parallel \overline{I}(\underline{t_1}).I(x_2).\overline{I}(x_2).\overline{I}(\langle \underline{t_1},\, x_2\rangle).0 \parallel \lceil C''\rceil$$
$$Q_2 = \overline{I}(\underline{t_1}).\overline{I}(\underline{t_2}).0 \parallel \lceil C''\rceil$$
$$Q_3 = [\langle \underline{t_1},\, \underline{t_2}\rangle = \langle x_1,\, x_2\rangle]\, \overline{I}(x_1).I(x_2).0 \parallel \lceil C''\rceil$$

each translated into C via $\lfloor\_\rfloor$ (specifically via $\lfloor\_\rfloor_I$). Let us observe that for any $Q_i' : Q_i \Rightarrow Q_i'$ then $\lfloor Q_i'\rfloor = C$, for $i = 1, 2, 3$. Then we conclude observing that $C \longrightarrow^* C$ is a possible corresponding transition.

$\star$ Case $(I.b)'''$: A last interesting situation happens when:

$$Q = \overline{I}(\underline{t_2}).0 \parallel \overline{I}(\underline{t_1}).0 \parallel I(x_2).\overline{I}(x_2).\overline{I}(\langle \underline{t_1},\, x_2\rangle).0 \parallel \lceil C''\rceil$$

In this case we observe that:

$$\begin{array}{rcl} Q & = & \overline{I}(\underline{t_2}).0 \parallel \overline{I}(\underline{t_1}).0 \parallel I(x_2).\overline{I}(x_2).\overline{I}(\langle \underline{t_1},\, x_2\rangle).0 \parallel \lceil C''\rceil \\ & \Rightarrow & \underbrace{0 \parallel \overline{I}(\underline{t_1}).0 \parallel \overline{I}(\underline{t_2}).\overline{I}(\langle \underline{t_1},\, \underline{t_2}\rangle).0 \parallel \lceil C''\rceil}_{Q'} \end{array}$$

Then we have:

$$\lfloor Q\rfloor = I(\underline{t_2}), I(\underline{t_1}), C'' \longrightarrow \underbrace{I(\underline{t_2}), I(\underline{t_1}), I(\langle \underline{t_1},\, \underline{t_2}\rangle), C''}_{C'} \quad [\text{by } r_{I_6}]$$

and it is easy to check that $\lfloor Q'\rfloor = C'$.

• $\boxed{pa_\nu}$ ***i.e.*, new name generation)**

The only possible transition pa_ν happens when analyzing cases in (b) *i.e.*, when $(C, Q) = (Q, \lfloor Q\rfloor)$. In fact no process obtained from $\lceil\_\rceil$ can perform a pa_ν transition as first step.
$\star$ Case $(II.b)$: $(C, Q) = (\lfloor Q\rfloor, Q')$. The first easy scenario is the following:

$$\begin{array}{rcl} Q & = & \overbrace{\nu \boldsymbol{n}}^{\nu n_1.\cdots.\nu n_h}.P_\rho \parallel \lceil C\rceil \\ & \Rightarrow & \underbrace{\nu n_2.\cdots \nu n_h.P_\rho[\underline{m}/n_1] \parallel \lceil C\rceil}_{Q'} \end{array}$$

In this case, being $\nu n_2.\cdots \nu n_h.P_\rho[\underline{m}/n_1]$ one of the processes left out by encoding $\lfloor\_\rfloor$, we have that $\lfloor Q'\rfloor = \lfloor Q\rfloor = C$, and we conclude observing that $C \longrightarrow^* C$ is a possible transition.
$\star$ Case $(II.b')$: the second, more interesting, scenario happens when :

$$\begin{array}{rcll} Q & = & \nu n_h.P_\rho[\theta] \parallel \lceil C\rceil & \text{[where } \theta \text{ are the substitutions applyed so far]} \\ & \Rightarrow & \underbrace{P_\rho \overbrace{[\theta][\underline{m}/n_1]}^{\theta'} \parallel \lceil C\rceil}_{Q'} & \end{array}$$

Then we have

$$\lfloor Q \rfloor = \overbrace{\tilde{\pi}(\underline{\boldsymbol{t}}), C''}^{\lceil C \rceil} \longrightarrow \underbrace{A_{\rho_0}(\boldsymbol{x};\boldsymbol{n})[\theta'], \lceil C \rceil}_{C'} \quad [\text{by } r_{\rho_0}]$$

and it is easy to check that $\lceil C' \rceil = Q'$.

• $\boxed{pa_{[]}}$ **(*i.e.*, matching)**

The only interesting case happens when $C = A_{\rho_{i-1}}(\underline{\boldsymbol{x}'[\theta]}), C''$ and $r_{\rho_i} = A_{\rho_{i-1}}(\boldsymbol{t}(\boldsymbol{x})) \longrightarrow A_{\rho_i}(\boldsymbol{x})$. Let start observing that in this case transition $\lceil C \rceil \Rightarrow Q'$ can be written as:

$$\begin{array}{rcl} \lceil C \rceil & = & \overbrace{[\underbrace{\underline{\boldsymbol{x}'[\theta]}}_{\lceil A_{\rho_{i-1}}(\underline{\boldsymbol{x}'[\theta]}) \rceil} = \boldsymbol{t}(\boldsymbol{x})]}^{\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x})}[\theta]}. \quad \overbrace{P_\rho[\theta]}^{\lceil r_{\rho_{i+1}} \rceil^{\#}_{(\boldsymbol{x})}[\theta][\theta'] = \lceil A_{\rho_i}(\underline{\boldsymbol{x}[\theta][\theta']}) \rceil} \quad \| \lfloor C'' \rfloor \\ \Rightarrow & & \overbrace{P_\rho[\theta][\theta']} \qquad\qquad \| \lfloor C'' \rfloor \quad [\text{where } \theta' : \underline{\boldsymbol{x}'[\theta]} = \boldsymbol{t}(\boldsymbol{x})[\theta']] \end{array}$$

Then we have:

$$C = A_{\rho_{i-1}}(\underline{\boldsymbol{x}'[\theta]}), C'' \longrightarrow \underbrace{A_{\rho_i}(\underline{\boldsymbol{x}[\theta][\theta']}), C''}_{C'}$$

and it is easy to check that $\lceil C' \rceil = Q'$.

• $\boxed{pa_{\equiv}}$ **(*i.e.*, structural equivalence)**

The proof in case of $pa_{\equiv}$ transitions, follows easily from the previous transition cases by induction.

Here ends proof of **(II)**, where we have shown that for every $(C, Q) \in \mathcal{R}$ $Q \Rightarrow Q'$ implies $C \longrightarrow^* C'$, and $(C', Q') \in \mathcal{R}$. ■

Main Theorem (Reminder) 2 *Given an* PA_P *security protocol theory* Q*. Then* $\lfloor Q \rfloor \sim Q$*.*

Proof. Similar to the proof of Theorem 5.5.9, by defining the relation $\mathcal{R}' = \{(\lfloor Q \rfloor, Q) : Q_0 \Rightarrow^* Q\}$ and showing that it is a correspondence relation $\sim$. ■

6

Security Analysis with Team Automata

"Molti ci gabbano" (Leonardo da Vinci in Aforismi, Novelle e Profezie, *L. da Vinci)*

"Many people deceive us"

Abstract

In this chapter we develop a framework based on team automata that can be used for formal security analysis. To this aim, we first define an insecure communication scenario for team automata, which is general enough to encompass various communication protocols. Then, we reformulate the Generalized Non-Deducibility on Compositions schema, originally introduced in the context of process algebras, in terms of team automata. Based on the resulting framework, we subsequently develop a compositional analysis strategy that can be used for the verification of security properties for a variety of communication protocols. We apply the framework in practise, by showing that integrity is guaranteed for a particular instance of the Efficient Multi-chained Stream Signature protocol.

6.1 Introduction

Recent years have seen an increasing interest in the use of automata-based formalisms for the specification and verification of security properties in communication protocols [109, 129, 140, 169, 170]. We continue this line of research by showing how team automata — an extension of Input/Output (I/O) automata [142] — can be used for security analysis.

Team automata offer a flexible formal model which allows one to specify the components of a reactive, distributed system and – separately – to describe their interactions. Originally introduced in the context of Computer Supported Cooperative Work for formalizing the conceptual and architectural levels of groupware systems [18, 74, 123], team automata have proved their usefulness also in the context of computer security. In [195] various access control strategies have been specified and analyzed by means of team automata. An effort was made in [72] to use team automata to model and analyze a privacy property of a protocol by Cachin *et al.* [41] for securing mobile agents in a hostile environment.

In this chapter we develop a general framework for security analysis with team automata. To this aim, we first define an insecure communication scenario for team automata, based on the addition of a so-called most general intruder to a team automaton model of a secure communication protocol. Then, we reformulate the GNDC schema in terms of team automata and subsequently describe a compositional analysis strategy for insecure scenario, which can be used for verifying security properties. Finally, we apply this framework to show that a particular instance of the Efficient Multi-chained Stream Signature (EMSS) protocol [173] achieves integrity. The aim of

this case study is not to provide new insights into the EMSS protocol, but rather to show the effectiveness of our approach for a well-known stream signature protocol, thus facilitating an easy comparison for those familiar with other approaches.

Our approach is not unique. In [140], an experiment involving the combination of simple shared-key communication with the Diffie-Hellman key distribution protocol [67] is modelled and proved correct using I/O automata. As noted by the author herself, a limitation of I/O automata approach is the fact that the protocol allows only purely passive eavesdroppers to listen in on the communication. This choice simplifies the formulation of compositional results, as an eavesdropper cannot change the course of communication, *e.g.,* by conducting a communication in which it pretends to be an honest participant. The I/O automata approach does provide compositional reasoning techniques.

Another related approach can be found in [169, 170], where interactive state machines — another extension of I/O automata — are introduced and applied to security analysis. In particular, interacting state machines are used to model and analyze the classic Needham-Schroeder public-key authentication protocol in the corrected version by Lowe [138]. An advantage of this approach is the fact that it allows one to automatize the verification, and to prove theorem-like properties, using the theorem prover Isabelle/HOL [168]. What is missing are solid techniques for compositional reasoning over more complex communication protocols.

This chapter is organized as follows. In Section 6.2 we define team automata, after which we describe an insecure communication scenario for team automata in Section 6.3. In Section 6.4 we reformulate the GNDC schema in terms of team automata and enrich the insecure scenario with a compositional analysis strategy. We subsequently apply this in Section 6.5 by verifying integrity in a case study, in which team automata specify an instance of the EMSS protocol. Finally, the chapter is concluded by a summary of our main results and some directions for future work.

6.2 Background on Team Automata

A team automaton consists of a number of component automata — which are ordinary automata without final states in which actions are divided into input, output, and internal actions — combined in a coordinated way so that they can perform shared actions. Internal actions have strictly local visibility and cannot be used for communicating with other component automata, while input and output actions together form the external actions that are observable by other components and that are used for the communication between components. During each communication step the components within a team may simultaneously participate in one instantaneous action, *i.e.,* synchronize on this action, or remain idle. Component automata can thus be combined in a loose or more tight fashion depending on the actions on which to synchronize and when. Team automata can in turn be used as components in a higher-level team automaton.

Technically, team automata are an extension of I/O automata. However, whereas I/O automata are required to be input enabled, *i.e.,* in each state it must be possible to execute every input action, such a restriction does not hold for component (and team) automata. Moreover, the composition of a set of component automata need not result in a unique team automaton, but can be a whole range of team automata—distinguishable only by their synchronizations. I/O automata, on the other hand, are uniquely defined by their constituents. Finally, I/O automata do not allow output actions to be synchronized, whereas team automata do.

The main feature distinguishing team automata from other models in the literature is the freedom they offer by allowing one to *choose* the synchronizations when composing a team from a

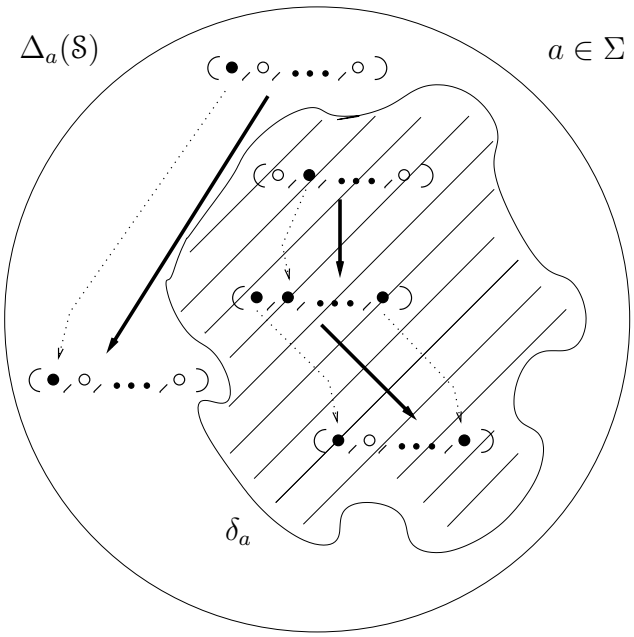


Figure 6.1: Transition space of TA. Here $a \in \Sigma$ is an action name. Each tuple of circles represents a state of the team automaton, whereas each individual circle represents a state of a component automaton. Black circles are the states that participate in a-transitions (here represented as dotted lines). Any transition in the set, $\Delta_a(\mathcal{S})$, of all the a-transitions is a potential a-transition of the team automata (here represented as solid lines). The definition of a transition relation δ_a allows only a selection of a-transitions to be part of the resulting team automata (the cross-hatched area) .

set of component automata. Most automata-based models, on the contrary, use a single method of composition, resulting in composite automata that are uniquely defined by their constituents. This holds for all the above mentioned automata-based models, and — in disguise — in several non-automata-based models, like CSP and statecharts [112].

We briefly introduce the notation and terminology used throughout this chapter; then we recall some definitions and results concerning team automata from [18, 196].

The (Cartesian) product of sets V_i, with $i \in \{1, \ldots, n\}$, is denoted by $\prod_{i\in\{1,\ldots,n\}} V_i$. In addition to the prefix notation, we also use the infix notation $V_1 \times \cdots \times V_n$. For $j \in \{1, \ldots, n\}$, $\text{proj}_j : \prod_{i\in\{1,\ldots,n\}} V_i \to V_j$ is defined by $\text{proj}_j((a_1, \ldots, a_n)) = a_j$. The power-set of a set V is denoted by 2^V. Let Σ and Γ be sets of symbols, $\Gamma \subseteq \Sigma$. The morphism $\text{pres}_{\Sigma,\Gamma} : \Sigma^* \to \Gamma^*$, defined by $\text{pres}_{\Sigma,\Gamma}(a) = a$ if $a \in \Gamma$ and $\text{pres}_{\Sigma,\Gamma}(a) = \lambda$ (the empty string) otherwise, preserves the symbols from Γ and erases all other symbols. In the following we discard Σ when no confusion can arise, and we use of the trivial extension of pres_Γ to sets of sequences.

Let $f : A \to A'$ and $g : B \to B'$ be functions. Then $f \times g : A \times B \to A' \times B'$ is defined as $(f \times g)(a, b) = (f(a), g(b))$. We use $f^{[2]}$ as shorthand for $f \times f$.

Definition 6.2.1 *An* automaton *is a 4-tuple $\mathcal{A} = (Q, \Sigma, \delta, I)$, with a set Q of* states, *a set Σ of* actions, *$Q \cap \Sigma = \varnothing$, a set $\delta \subseteq Q \times \Sigma \times Q$ of* transitions, *and a set $I \subseteq Q$ of* initial states. *The set $\mathbf{C}_\mathcal{A}$ of* computations *of $\mathcal{A}$ consists of all the sequences $\alpha = q_0 a_1 q_1 \cdots a_n q_n$, where $n \geq 0$ and*

$q_0 \in I$, *and for all* $i \in \{1,\ldots,n\}$: $q_i \in Q$, $a_i \in \Sigma$, *and* $(q_{i-1}, a_i, q_i) \in \delta$. *The* Γ-behavior $\mathbf{B}^{\Gamma}_{\mathcal{A}}$ *of* $\mathcal{A}$, *with* $\Gamma \subseteq \Sigma$, *is defined by* $\mathbf{B}^{\Gamma}_{\mathcal{A}} = \mathit{pres}_{\Gamma}(\mathbf{C}_{\mathcal{A}})$.

The Σ-behavior of $\mathcal{A}$ is also called the *behavior* of $\mathcal{A}$, in which case Σ may be omitted. Finally, note that behavioral inclusion defines a preorder relation on automata.

As said before, team automata are composed of component automata, which are automata distinguishing *input*, *output*, and *internal* actions.

Definition 6.2.2 *A* component automaton *is a construct* $\mathcal{C} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$, *with an* underlying automaton $(Q, \Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}, \delta, I)$ *and pairwise disjoint sets* Σ_{inp} *of* input, Σ_{out} *of* output, *and* Σ_{int} *of* internal actions.

The set Σ denotes the set $\Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}$ of *actions* of the component automaton $\mathcal{C}$ and Σ_{ext} denotes its set $\Sigma_{inp} \cup \Sigma_{out}$ of *external* actions. In the sequel we let $\mathcal{S} = \{\mathcal{C}_i \mid i \in \{1,\ldots,n\}\}$ be an arbitrary but fixed set of component automata specified by $\mathcal{C}_i = (Q_i, (\Sigma_{i,inp}, \Sigma_{i,out}, \Sigma_{i,int}), \delta_i, I_i)$, with set $\Sigma_i = \Sigma_{i,inp} \cup \Sigma_{i,out} \cup \Sigma_{i,int}$ of actions and set $\Sigma_{i,ext} = \Sigma_{i,inp} \cup \Sigma_{i,out}$ of external actions.

When composing team automata the various internal actions of the components automata must be kept private, *i.e.*, be uniquely associated to one component automaton. This is obtained by requiring that $\Sigma_{i,int} \cap \bigcup_{j \in (\{1,\ldots,n\}-\{i\})} \Sigma_j = \varnothing$, for all $i \in \{1,\ldots,n\}$, *i.e.*, no internal action of any component from $\mathcal{S}$ may appear as an action in any of the other components constituting $\mathcal{S}$. If this is the case, then $\mathcal{S}$ is called a *composable system* and in the sequel we assume that $\mathcal{S}$ is a composable system. We speak of a team automaton over $\mathcal{S}$ if its components are exactly the automata in $\mathcal{S}$.

The state space of a team automaton is the product of the state spaces of the components (in $\mathcal{S}$). The internal actions of the components are the internal actions of the team automaton. Each action which is output for one or more of the components is an output action of the team. In particular, an action that is an output action of one component and also an input action of another component, is considered an output action of the team automaton. The input actions of the team that do not occur at all as output action of any of the components in $\mathcal{S}$, are the input actions of the team. The reason for this construction is the following. When relating an input action a of a component to an output action a of another component, the input may be thought of as being caused by the output. On the other hand, the output action remains observable as output. Finally, the transitions of a team automaton over $\mathcal{S}$ are based on, but not fixed by, the transition of transition of the components constituting $\mathcal{S}$. They are chosen by allowing certain *synchronizations* on actions, while excluding others. To define a TA, we need to synchronize various component automata. The following definition allows us to define the "maximal synchronization" setting. Let $a \in \Sigma$, in the following the set δ_a, called a-*transitions* of $\mathcal{A}$, is defined as $\delta_a = \{(q, q') \mid (q, a, q') \in \delta\}$.

Definition 6.2.3 *Let* $\mathcal{S}$ *a set of component automata, and* $a \in \bigcup_{i \in \{1,\ldots,n\}} \Sigma_i$. *The set* $\Delta_a(\mathcal{S})$ *of* synchronizations *of* a *is defined as*

$$\begin{aligned}\Delta_a(\mathcal{S}) \;=\; \{(q,q') \in \prod_{i\in\{1,\ldots,n\}} Q_i \times \prod_{i\in\{1,\ldots,n\}} Q_i \mid (\exists\, j \in \{1,\ldots,n\} :\ & \mathit{proj}_j{}^{[2]}(q,q') \in \delta_{j,a}) \\ \text{and } (\forall\, i \in \{1,\ldots,n\} :\big(& \mathit{proj}_i{}^{[2]}(q,q') \in \delta_{i,a}) \text{ or } (\mathit{proj}_i(q) = \mathit{proj}_i(q'))\big)\}.\end{aligned}$$

The set $\Delta_a(\mathcal{S})$ thus contains all possible combinations of a-transitions of the components in $\mathcal{S}$, with all non-participating components remaining idle. It is explicitly required that at least one component is non-idle. Figure 6.1 gives an idea of the transition space $\Delta_a(\mathcal{S})$ of a team automaton

over $\mathcal{S}$. When defining a team automaton over $\mathcal{S}$, a specific subset of $\Delta_a(\mathcal{S})$ must be chosen for each action a. This specifies the synchronization between the components constituting the team.

Definition 6.2.4 *Let $\mathcal{S} = \{(Q_i, \Sigma_{i,inp}, \Sigma_{i,out}, \Sigma_{i,int}, \delta_i, I_i) \mid i \in \{1, \ldots, n\}\}$ be a set of component automata. A* team automaton $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ *is is a construct over $\mathcal{S}$ with:*

$$
\begin{aligned}
Q &= \prod_{i \in \{1,\ldots,n\}} Q_i, \\
\Sigma_{inp} &= (\bigcup_{i \in \{1,\ldots,n\}} \Sigma_{i,inp}) - \Sigma_{out}, \\
\Sigma_{out} &= \bigcup_{i \in \{1,\ldots,n\}} \Sigma_{i,out}, \\
\Sigma_{int} &= \bigcup_{i \in \{1,\ldots,n\}} \Sigma_{i,int}, \\
\delta &\subseteq Q \times \Sigma \times Q
\end{aligned}
$$

Here $I = \prod_{i \in \{1,\ldots,n\}} I_i$, δ is such that $\delta_a = \{(q, q') \mid (q, a, q') \in \delta\} \subseteq \Delta_a(\mathcal{S})$, for all $a \in \Sigma = \Sigma_{inp} \cup \Sigma_{out} \cup \Sigma_{int}$, and $\delta_a = \{(q, q') \mid (q, a, q') \in \delta\} = \Delta_a(\mathcal{S})$, for all $a \in \Sigma_{int}$.

All team automata over a given composable system have the same set of states, the same alphabet of actions — including the distribution over input, output, and internal actions — and the same set of initial states. They only differ in the choice of the transition relation δ and only as far as external actions are concerned: for each external action a we have the freedom to choose δ_a. This implies that $\mathcal{S}$, even if it is a composable system, does not uniquely define a team automaton. Each choice of synchronizations thus defines a team automaton. It is important to observe that *every team automaton is again a component automaton*, which in turn can be used as a component in an *hierarchically* composed team.

It can be useful to *hide* certain external actions of a team automaton before composing this team with other teams to avoid synchronizations on these actions (on a higher level of the composition).

Definition 6.2.5 *Let $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ be a team automaton and let $\Gamma \subseteq \Sigma_{ext}$. Then hide$_\Gamma(\mathcal{T}) = (Q, (\Sigma_{inp} - \Gamma, \Sigma_{out} - \Gamma, \Sigma_{int} \cup \Gamma), \delta, I)$.*

In hide$_\Gamma(\mathcal{T})$, the external actions in Γ have thus become unobservable to other automata by regarding them as internal actions.

6.2.1 The Max-ai Team Automata

In the sequel, we make use of a team automaton of a specific type, called *max-ai team automaton*[1]. Informally, the max-ai team automaton over a composable system $\mathcal{S}$ is the unique team automaton in which any execution of an action a sees the participation of all components having a in their set of actions. Before we can define max-ai automata, we first need to define the following relation $\mathcal{R}_a^{ai}(\mathcal{S})$:

[1] Here "*ai*" stands for action indispensable.

Definition 6.2.6 *Let $a \in \bigcup_{i\in\{1,\ldots,n\}} \Sigma_i$. The set* is-ai *for a in $\mathcal{S}$, denoted by $\mathcal{R}_a^{ai}(\mathcal{S})$, is defined as* $\mathcal{R}_a^{ai}(\mathcal{S}) = \{(q,q') \in \Delta_a(\mathcal{S}) \mid \forall\, i \in \{1,\ldots,n\} : [a \in \Sigma_i \Rightarrow \mathit{proj}_i{}^{[2]}(q,q') \in \delta_{i,a}]\}$.

The set $\mathcal{R}_a^{ai}(\mathcal{S})$ thus contains *all and only* those a-transitions from $\Delta_a(\mathcal{S})$ in which every component automaton with a as an action participates. Hence the max-ai team automaton over $\mathcal{S}$ is the unique team automaton in which any execution of a sees the participation of all components having a in their set of actions.

Definition 6.2.7 $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ *is* the max-ai team automaton *over* $\mathcal{S}$, *denoted by* $|||\,\mathcal{S}$, *if* $\delta_a = \mathcal{R}_a^{ai}(\mathcal{S})$, *for all* $a \in \Sigma$.

Figure 6.2 shows two component automata $\mathcal{C}_1$ and $\mathcal{C}_2$. Figure 6.3 shows two of the several team automata that can be built by starting from those component automata. We enforce maximal synchronization in $\mathcal{T}^{ai} = |||\,\{\mathcal{C}_1, \mathcal{C}_2\}$: any execution of action a and action b sees the participation of both components whenever possible. $\mathcal{T}^{free}$ is the team automaton over $\{\mathcal{C}_1, \mathcal{C}_2\}$ in which any execution of action a and action b sees the participation of only one component.

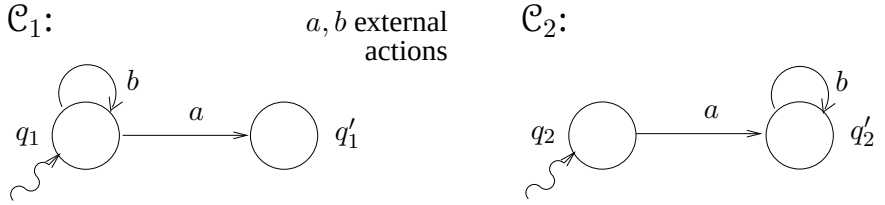


Figure 6.2: Example of two composite automata, $\mathcal{C}_1$ and $\mathcal{C}_2$. Here q_1, q_1', q_2, and q_2' are states, solid lines are transitions, and a and b are external actions. A curved arrow points the initial state of each automaton.

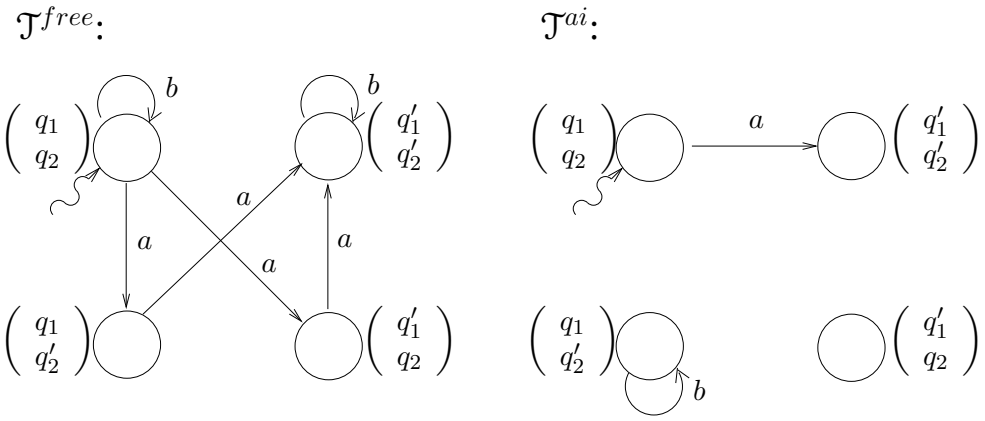


Figure 6.3: Example of two different TA over $\{\mathcal{C}_1, \mathcal{C}_2\}$. $\mathcal{T}^{ai}$ is the max-ai team automaton. $\mathcal{T}^{free}$ is the team automaton whose transition relation selects those transitions of the team involving only one single component.

The Γ-*behavior* of a team automaton $\mathcal{T}$, denoted as $\mathbf{B}_{\mathcal{T}}^{\Gamma}$, is defined as usual in automata theory (see Definition 6.2.1). In particular, $\mathbf{B}_{\mathcal{T}}^{\Gamma} = \mathrm{pres}_{\Gamma}(\mathbf{C}_{\mathcal{T}})$, with set $\mathbf{C}_{\mathcal{T}}$ of computations of $\mathcal{T}$ consisting of all the sequences $\alpha = q_0 a_1 q_1 \ldots a_n q_n$, where $n \geq 0$ and q_0 is an initial state, q_i, are states,

a_i are actions and (q_{i-1}, a_i, q_i) are transitions. When $\Gamma = \Sigma_{out}$, then $\mathbf{B}_{\mathcal{T}}^{\Sigma_{out}}$ is the output behavior of $\mathcal{T}$. By appropriately choosing Γ, also the input and the internal behavior of $\mathcal{T}$ can be defined.

Remark 6.2.8 In [18] it was shown that the behavior of an iteratively composed max-ai team automaton equals that of the max-ai team automaton over the underlying components. For example, considering the automata in Example 6.3, if $\mathcal{T}'$ and $\mathcal{T}''$ are the max-ai team automata over $\{\mathcal{T}'', \mathcal{C}_3\}$ and $\{\mathcal{C}_1, \mathcal{C}_2\}$ respectively, and if $\mathcal{T}$ is the max-ai team automaton over $\{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3\}$, then $\mathbf{B}_{\mathcal{T}'} = \mathbf{B}_{\mathcal{T}}$. ■

6.2.2 Compositionality in Team Automata

A team automaton is said to satisfy *compositionality* if its behavior can be described in terms of that of its constituents *i.e.,* when the behavior of the team automaton automata can be expressed as a *shuffled* version of the sequences that form the behaviors of the set of its components [196].

Definition 6.2.9 *Let Δ_i be alphabets and $L_i \subseteq \Delta_i^*$, with $i \in \{1, \ldots, n\}$. The* fully synchronized shuffle, $\underline{||}_{\{\Delta_i | i \in \{1,\ldots,n\}\}} L_i$ *is defined as* $\underline{||}_{\{\Delta_i | i \in \{1,\ldots,n\}\}} L_i = \{w \in (\bigcup_{i \in \{1,\ldots,n\}} \Delta_i)^* \mid \forall i \in \{1, \ldots, n\} : \mathit{pres}_{\Delta_i}(w) \in L_i\}$.

Example 6.2.10 Let Δ_1, Δ_2 be alphabets. Let $L_1 = \{abc\}$ be a sequence such that $L_1 \subseteq \Delta_1 = \{a, b, c\}$ and $L_2 = \{cd\}$ a second sequence such that $L_2 \subseteq \Delta_2 = \{c, d\}$. Then, the fully synchronized shuffle $abc \ {}_{\Delta_1}\underline{||}_{\Delta_2} \ cd = \{abcd\}$ (*i.e.,* words must synchronize on $\Delta_1 \cap \Delta_2 = \{c\}$). ■

Before continuing, we observe the following property of full synchronized shuffles.

Remark 6.2.11 Let Δ_i, with $i \in \{1, .., 4\}$, be alphabets and let $L_i \subseteq \Delta_i^*$. Then

$$\underline{||}_{\{\Delta_1,\Delta_3\}} \{L_1, L_3\} \subseteq \underline{||}_{\{\Delta_2,\Delta_4\}} \{L_2, L_4\}$$

whenever $L_1 \subseteq L_2$ and $L_3 \subseteq L_4$. ■

In [196] it was shown that the construction of team automata to certain types of synchronization, like the one leading to max-ai team automata, guarantees compositionality.

Theorem 6.2.12 (Compositionality of team automata) *Let $\mathcal{T}$ be the max-ai team automaton over $\mathcal{S}$. Then $\mathbf{B}_{\mathcal{T}} = \underline{||}_{\{\Sigma_i | i \in \{1,\ldots,n\}\}} \mathbf{B}_{\mathcal{C}_i}$.*

6.3 An Insecure Communication with Team Automata

In this section we use team automata to model a generic (insecure) communication system in which to analyze security properties.

We assume all actions to be built over a first order signature σ, where predicate symbols are seen as communication channels and atomic formulas as messages. We assume that σ contains at least the following function symbols: $\{\_\}_\_$ encryption, $\langle \_, \_ \rangle$ paring, $h(\_)$ hashing, and those indicating the secret and public key, $sk(\_)$ and $pk(\_)$ respectively. We let m, m' range over the set Messages of atomic formulas and c, c' over the set Channels of predicate symbols. In the sequel, `Eve`, `Eve'`, `Pub`, `Pub'`, `Reveal`, and `Reveal'` will be used as particular predicate names. An action is denoted by $c(m)$, which represents a message m sent over channel c. Given a set

$M \subseteq$ Messages of messages, we define $c(M) = \{c(m) \mid m \in M\}$. Given a set C of predicate names we define $C(M) = \{c(m) \mid m \in M,\ c \in C\}$. Finally, with a little abuse of notation, we will also write C as an abbreviation for the set $C(\mathsf{Messages})$.

We abstract from the cryptographic details concerning the operations according to which messages can be encrypted, decrypted, hashed, *et cetera*, but we assume the presence of a cryptosystem (defined by a derivation operator $\vdash$) that implements these operations. By applying cryptographic operations from this cryptosystem to a set M of messages, a new set $KS(M) = \{m \mid M \vdash m\}$ of messages (usually called the *deduction set*) can be obtained. This approach is standard in the analysis of (cryptographic) communication protocols [53, 86, 134, 140].

In the sequel, we model a generic cryptographic communication protocol specification involving two roles, *viz.* an *initiator* $\mathcal{T}_S$ and a *responder* $\mathcal{T}_R$. We assume all the communication between $\mathcal{T}_S$ and $\mathcal{T}_R$ to flow through an *insecure channel* (cf. Figure 6.4). This insecure channel may release some messages to an *intruder* which, in turn, can eavesdrop on these messages as well as inject fake messages in the communication channel. This is a standard approach for verifying security properties for (cryptographic) communication protocols. A protocol specification is considered secure with respect to a security property if it satisfies this property despite the presence of the intruder. As in [140], the insecure channel and the intruder are modelled by team automata $\mathcal{T}_{IC}$ and $\mathcal{T}_X$. We thus propose a framework consisting of four types of team automata (see also Figure 6.4):

1. $\mathcal{T}_S$ plays the role of the protocol's initiator,
2. $\mathcal{T}_R$ plays the role of the protocol's responder,
3. $\mathcal{T}_{IC}$ plays the role of the insecure channel, and
4. $\mathcal{T}_X$ plays the role of the active and malicious intruder.

We let the initiator and the responder communicate with the insecure channel through disjoint sets of actions Σ^S_{com} and Σ^R_{com}, respectively, so that a direct communication between them is impossible. The $\mathcal{T}_{IC}$, in turn, can interact with the intruder only through a distinct set Σ^I_{com} of actions. Finally, some particular actions may be used by an honest role to reveal some information to the outside concerning, *e.g.*, a state reached during a run of the protocol.
We let $\mathcal{T}_P$ denote the team automaton representing our protocol specification in the absence of the intruder. We thus define $\mathcal{T}_P$ to be the max-ai team automaton over $\{\mathcal{T}_S, \mathcal{T}_R, \mathcal{T}_{IC}\}$ that is obtained after hiding the actions $\Sigma^P_{com} = \Sigma^S_{com} \cup \Sigma^R_{com}$, *i.e.*, all messages passing through the insecure channel (*e.g.*, $\Sigma^P_{com} = \{\mathtt{Pub}(m), \mathtt{Pub}'(m) \mid \forall\, m \in \mathsf{Messages}\}$ in Figure 6.4). Hence

$$\mathcal{T}_P = \mathrm{hide}_{\Sigma^P_{com}}(\,|||\,\{\mathcal{T}_S, \mathcal{T}_R, \mathcal{T}_{IC}\}).$$

By hiding Σ^P_{com}, $\mathcal{T}_P$ - appears as a black box, possibly with some output actions Σ^S_{sig} and Σ^R_{sig}— signalling the successful reception of messages. Usually such signals are used only for verification purposes and for the sequel we assume that $\Sigma^S_{sig} \cap \Sigma^R_{sig} = \varnothing$ (*e.g.*, $\Sigma^S_{sig} = \{\mathtt{Reveal}\}$ and $\Sigma^R_{sig} = \{\mathtt{Reveal}'\}$ in Figure 6.4).

We let $\mathcal{T}_I$ be the team automaton representing the protocol specification in presence of the intruder. The actions in Σ^I_{com} serve as back-door for intrusion and are added to $\mathcal{T}_{IC}$ (*e.g.*, $\Sigma^I_{com} = \{\mathtt{Eve}, \mathtt{Eve}'\}$ in Figure 6.4). This is what we need to guarantee that the intruder $\mathcal{T}_X$ may communicate with $\mathcal{T}_P$ only through the insecure channel. We define $\mathcal{T}_I$ to be the max-ai team automaton over $\{\mathcal{T}_P, \mathcal{T}_X\}$ that is obtained after hiding the actions Σ^I_{com}, *i.e.*, all messages that the intruder

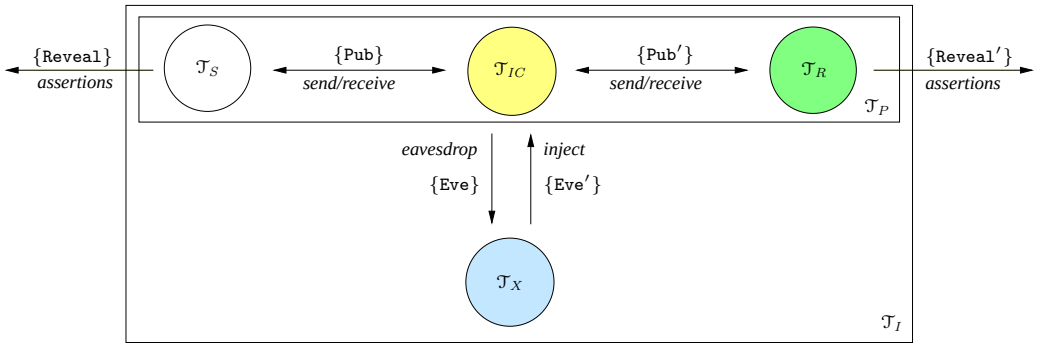


Figure 6.4: The insecure communication scenario for Team Automata. The insecure scenario is represented by the automaton $\mathcal{T}_I$. Within the scenario, $\mathcal{T}_S$ models the protocol's initiator, $\mathcal{T}_{IC}$ models the insecure channel, $\mathcal{T}_R$ models the protocol's responder, and $\mathcal{T}_X$ represents the Dolev-Yao [69] intruder. The team automaton $\mathcal{T}_P$, composed over $\{\mathcal{T}_S, \mathcal{T}_{IC}, \mathcal{T}_R\}$ represents the secure scenario.

can eavesdrop from and inject back into the insecure channel. We thus enforce maximal synchronization between the intruder and the protocol. Hence

$$\mathcal{T}_I = \text{hide}_{\Sigma^I_{com}}(|||\{\mathcal{T}_P, \mathcal{T}_X\})$$

We have now defined an insecure communication scenario for team automata by composing a secure communication scenario with an intruder.

6.4 GNDC Security Analysis for Team Automata

In Chapter 3 (Section 3.2) we have already seen that GNDC is a scheme that has the form

$$P \in \mathit{GNDC}^{\alpha}_{\triangleleft} \text{ iff } \forall X \in \mathcal{E}_C : (P \parallel X) \setminus C \triangleleft \alpha(P)$$

where $(P \parallel X) \setminus C$ denotes the parallel composition of processes P and X restricted to communication over channels other than C. X is an arbitrary (possibly malicious) process in the environment $\mathcal{E}_C$, the set of all processes whose communicating actions are in C. By varying the parameters $\triangleleft$ and α, the GNDC schema can be used to define and verify many security properties—among which secrecy, integrity, and entity authentication [81, 84, 86, 104, 149]. Recently, a slightly extended GNDC schema was defined [85], incorporating the fact that the set of bad behaviors of P may depend on P itself and on the property under scrutiny.

In the specific context of analyzing cryptographic communication protocols, the *static* (initial) knowledge of the hostile environment must be bound to a specific set of messages. This limitation is needed to avoid a hostile intruder that is too strong, and which would therefore be able to corrupt any secret (as it would know all cryptographic keys, *et cetera*). This brings us to the definition of a new environment $\mathcal{E}^{\phi}_C$, based on $\mathcal{E}_C$, of all processes communicating through actions C and having an initial knowledge of at most the messages in $KS(\phi)$. For the analysis of safety properties (*e.g.*, secrecy, integrity, and entity authentication) it is sufficient to consider the trace inclusion relation $\leq$ as a behavioral relation between the terms of the algebra [86]. Hence, let us consider

the GNDC instance

$$P \in GNDC^{\alpha}_{\leq} \text{ iff } \forall X \in \mathcal{E}^{\phi}_{C} : (P \parallel X) \setminus C \leq \alpha(P), \tag{6.4.1}$$

which was, *e.g.*, used in [104] to analyze integrity in stream signature protocols. Informally, (6.4.1) requires traces of process $(P \parallel X) \setminus C$ to be included in the traces of process $\alpha(P)$, representing the expected behavior of P when no adversary is present.

6.4.1 Reformulating GNDC in Terms of Team Automata

We begin by describing the team automaton $\mathcal{T}_P = \{Q, (\Sigma^P_{inp}, \Sigma^P_{out}, \Sigma^P_{int}), \delta, I\}$ which models the system P. Because (6.4.1) requires P to communicate with X through the channels contained in C, we let $C = C_{inp} \cup C_{out}$; the actions in C_{inp} are input to X and the actions in C_{out} that are output to X. In the sequel, we assume C to coincide exactly with Σ^I_{com} and, in particular, C_{inp} with the actions in Σ^I_{com} that are input to $\mathcal{T}_X$ (*e.g.*, $\{\texttt{Eve}\}$ in Figure 6.4) and C_{out} with the actions in Σ^I_{com} that are output to $\mathcal{T}_X$ (*e.g.*, $\{\texttt{Eve}'\}$ in Figure 6.4). We are now able to formalize the hostile environment $\mathcal{E}_C$ in terms of team automata as:

$$\mathcal{E}_C = \{(Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I) \mid \Sigma_{inp} \subseteq C_{inp},\ \Sigma_{out} \subseteq C_{out}\}. \tag{6.4.2}$$

In addition, (6.4.1) requires the initial knowledge of the environment to be bound to a specified set of messages ϕ. This means that the environment should be able to produce, by means of only its internal functioning, at most the messages contained in $KS(\phi)$. In terms of team automata, this means that a component automaton in the environment, when considered as a stand-alone component, can only execute output actions belonging to $C(KS(\phi))$. Formally, the *initial knowledge* of $\mathcal{T}$ is defined as $\{\gamma \in \mathbf{B}_{\mathcal{T}} \mid \gamma \in {\Sigma^{\mathcal{T}}_{out}}^*\}$, and the formal definition of the environment $\mathcal{E}^{\phi}_{C}$ thus becomes:

$$\mathcal{E}^{\phi}_{C} = \{\mathcal{X} \in \mathcal{E}_C \mid \{\gamma \in \mathbf{B}_{\mathcal{X}} \mid \gamma \in {\Sigma^{\mathcal{X}}_{out}}^*\} \subseteq (C(KS(\phi)))^*\}. \tag{6.4.3}$$

Finally, we need a behavioral notion of comparison between team automata which abstracts from their internal and communicating actions. Furthermore, we want to be able to exclude all sequences containing an action occurring in C. Therefore, we hide the output actions involved in the communications and we define the *observational behavior* (with respect to actions not in C) of the resulting team automata as the sequences consisting solely of external actions not in C.

Definition 6.4.1 *Let $\mathcal{T} = (Q, (\Sigma_{inp}, \Sigma_{out}, \Sigma_{int}), \delta, I)$ be a team automaton over $\mathcal{S}$, let $\Sigma_{com} \subseteq \Sigma_{ext}$, and let $\mathcal{T}' = hide_{\Sigma_{com}}(\mathcal{T})$. Then the* observational behavior *of $\mathcal{T}'$ with respect to actions not in C, denoted by $\mathbf{O}^{C}_{\mathcal{T}'}$, is defined as*

$$\mathbf{O}^{C}_{\mathcal{T}'} = \{\gamma \in pres_{\Sigma^{\mathcal{T}'}_{ext}}(\mathbf{B}_{\mathcal{T}'}) \mid \gamma \in \Sigma^{\mathcal{T}'}_{ext} - C^*\}.$$

As a result we are able to reformulate (6.4.1) in terms of team automata.

Definition 6.4.2 *Let $\alpha(\mathcal{T}_P)$ be the expected (correct) behavior of $\mathcal{T}_P$. Then:*

$$\mathcal{T}_P \in GNDC^{\alpha(\mathcal{T}_P)}_{\subseteq} \text{ iff } \forall \mathcal{X} \in \mathcal{E}^{\phi}_{C} : \mathbf{O}^{C}_{hide_{C}(|||\{\mathcal{T}_P, \mathcal{X}\})} \subseteq \alpha(\mathcal{T}_P).$$

Informally, Definition 6.4.2 says that $\mathcal{T}_P$ (*i.e.*, a cryptographic communication protocol specified in the insecure communication scenario) satisfies $GNDC^{\alpha(\mathcal{T}_P)}_{\subseteq}$ if and only if its observational behavior, despite communicating with any intruder $\mathcal{X}$ through the actions C, is included in $\alpha(\mathcal{T}_P)$. A significant instance of α is, *e.g.*, $\alpha_{int}(\mathcal{T}_P) = \mathbf{O}^{C}_{\mathcal{T}_P}$, which will be used in Section 6.5.2 to express integrity.

6.4.2 Security Analysis Strategies for Team Automata

While allowing a uniform approach for specifying security properties, Definition 6.4.2 does not provide us with effective strategies for the analysis of (cryptographic) communication protocols. In particular, the universal quantification over $\mathcal{E}_C^\phi$ causes serious problems when checking $\mathcal{T}_P \in GNDC_{\subseteq}^{\alpha(\mathcal{T}_P)}$. Luckily, the strategies developed for GNDC in the context of process algebras can be transferred to team automata.

The Most General Intruder. To avoid the infinite number of checks that the universal quantification requires, we now show that there exists an attacker that is more powerful (with respect to a chosen behavioral relation) than all the others. In this way one can reduce the analysis against any environment to an analysis against only one, albeit very powerful, so-called *most general intruder*. From the theory of GNDC [85] we know that a sufficient condition for the existence of such a most general intruder, is to have a behavioral relation that is a *pre-congruence* with respect to the (parallel) composition and restriction operators. Restated in our framework we say that $\lhd$ is a pre-congruence (with respect to $|||$ and hide_C) if for every automaton $\mathcal{T}$, $\mathcal{X}$ and $\mathcal{X}'$ in $\mathcal{E}_C$, whenever $\mathbf{B}_{\mathcal{X}}^C \lhd \mathbf{B}_{\mathcal{X}'}^C$ then $\mathbf{O}^C_{\text{hide}_C(|||\{\mathcal{T},\mathcal{X}\})} \lhd \mathbf{O}^C_{\text{hide}_C(|||\{\mathcal{T},\mathcal{X}'\})}$. It is not difficult to prove that this is true in our case, *viz.*

Lemma 6.4.3 *Let $\mathcal{T} = (Q, (\Sigma_{inp}^{\mathcal{T}}, \Sigma_{out}^{\mathcal{T}}, \Sigma_{int}^{\mathcal{T}}), \delta, I)$ be a team automaton and let $\mathcal{X}, \mathcal{X}' \in \mathcal{E}_C$. Then*

$$\mathbf{B}_{\mathcal{X}}^C \subseteq \mathbf{B}_{\mathcal{X}'}^C \textit{ implies } \mathbf{O}^C_{\textit{hide}_C(|||\{\mathcal{T},\mathcal{X}\})} \subseteq \mathbf{O}^C_{\textit{hide}_C(|||\{\mathcal{T},\mathcal{X}'\})}.$$

Proof. Let $a_1 \cdots a_n \in \mathbf{O}^C_{\text{hide}_C(|||\{\mathcal{T},\mathcal{X}\})}$ and let $\mathbf{B}_{\mathcal{X}}^C \subseteq \mathbf{B}_{\mathcal{X}'}^C$. By (6.4.2), $\Sigma_{ext}^{\mathcal{X}} \subseteq C$ because $\mathcal{X} \in \mathcal{E}_C$. Then by Definition 6.4.1, for all $i \in \{1, \ldots, n\}$, $a_i \in \Sigma_{ext}^{\mathcal{T}} - C$. We now use the fact that by definition also all prefixes of $a_1 \cdots a_n$ are included in $\mathbf{O}^C_{\text{hide}_C(|||\{\mathcal{T},\mathcal{X}\})}$ and show by induction that all prefixes of $a_1 \cdots a_n$ are also included in $\mathbf{O}^C_{\text{hide}_C(|||\{\mathcal{T},\mathcal{X}'\})}$. First, consider a_1. By Definition 6.4.1, either $a_1 \in \mathbf{B}_{\text{hide}_C(|||\{\mathcal{T},\mathcal{X}\})}$ or $b_1 \cdots b_m a_1 \in \mathbf{B}_{\text{hide}_C(|||\{\mathcal{T},\mathcal{X}\})}$, for some $m \geq 1$ and where, for all $j \in [m]$, b_j is an internal action of $\text{hide}_C(|||\{\mathcal{T}, \mathcal{X}\})$. In both cases, since $\mathbf{B}_{\mathcal{X}}^C \subseteq \mathbf{B}_{\mathcal{X}'}^C$ and $a_i \in \Sigma_{ext}^{\mathcal{T}} - C$, for all $i \in \{1, \ldots, n\}$, it follows by Definition 6.4.1 that $a_1 \in \mathbf{O}^C_{\text{hide}_C(|||\{\mathcal{T},\mathcal{X}'\})}$. Now assume that $a_1 \cdots a_k \in \mathbf{O}^C_{\text{hide}_C(|||\{\mathcal{T},\mathcal{X}'\})}$, with $k < n$, and consider $a_1 \cdots a_{k+1}$. Using similar arguments as above and the induction hypothesis it follows that $a_1 \cdots a_{k+1} \in \mathbf{O}^C_{\text{hide}_C(|||\{\mathcal{T},\mathcal{X}'\})}$. ■

Since $\mathcal{E}_C^\phi \subseteq \mathcal{E}_C$, this lemma holds for $\mathcal{X}, \mathcal{X}' \in \mathcal{E}_C^\phi$ as well. Based on the approach of [86] we now define a component automaton Top_C^ϕ, representing the most general intruder.

We specify Top_C^ϕ in the way that I/O automata are commonly defined [140, 142]. Its states are thus defined by the current values of the variables listed under States, while its transitions are defined, per action a, as preconditions (Pre) and effect (Eff), *i.e.*, (q, a, q') is a transition of Top_C^ϕ if the precondition of a is satisfied by q, while q' is the transformation of q defined by the effect of a. We omit the precondition (effect) of an action when it is true.

Recall that the set C of predicates that the intruder uses to interact with the insecure channel is partitioned into C_{inp} and C_{out} (*e.g.*, in Figure 6.4, $C_{inp} = \{\mathsf{Eve}\}$ and $C_{out} = \{\mathsf{Eve}'\}$). Recall also that C (respectively, C_{inp} and C_{out}) is an abbreviation for $C(\mathsf{Messages})$ (respectively, $C_{inp}(\mathsf{Messages}) and C_{out}(\mathsf{Messages})$).

Top_C^ϕ

Actions
Inp: C_{inp}(Messages) Out: C_{out}(Messages) Int: $\varnothing$

States
received $\subseteq 2^{C_{inp}(\mathsf{Messages})}$, initially ϕ

Transitions
$c(m) \in C_{inp}$(Messages)
Eff: received := received $\cup \{m\}$

$c(m) \in C_{out}$(Messages)
Pre: $m \in KS$(received)

The general way in which Top_C^ϕ is specified implies that its behavior includes that of any automaton from $\mathcal{E}_C^\phi$.

Lemma 6.4.4 *For all* $\mathcal{X} \in \mathcal{E}_C^\phi$, $\mathbf{B}_{\mathcal{X}}^C \subseteq \mathbf{B}_{Top_C^\phi}^C$.

Proof. Let $\mathcal{X} \in \mathcal{E}_C^\phi$. Then (6.4.3) implies that $\mathcal{X} \in \mathcal{E}_C$ and thus, by (6.4.2) and the specification of Top_C^ϕ, $\Sigma_{inp}^{\mathcal{X}} \subseteq C_{inp} = \Sigma_{inp}^{Top_C^\phi}$ and $\Sigma_{out}^{\mathcal{X}} \subseteq C_{out} = \Sigma_{out}^{Top_C^\phi}$. From (6.4.3) and the specification of Top_C^ϕ it follows immediately that $\mathbf{B}_{\mathcal{X}}^C \subseteq \mathbf{B}_{Top_{\mathcal{E}}^\phi}^C$. ∎

Lemmata 6.4.3 and 6.4.4 directly imply the following result.

Theorem 6.4.5 *For all* $\mathcal{X} \in \mathcal{E}_C^\phi$, $\mathbf{O}_{hide_C(|||\{\mathcal{T}_P,\mathcal{X}\})}^C \subseteq \mathbf{O}_{hide_C(|||\{\mathcal{T}_P,Top_C^\phi\})}^C$.

Together with Definition 6.4.2, this gives us the following result.

Corollary 6.4.6 *Let* $\alpha(\mathcal{T}_P)$ *be as in Definition 6.4.2. Then*

$$\mathcal{T}_P \in GNDC_{\subseteq}^{\alpha(\mathcal{T}_P)} \text{ iff } \mathbf{O}_{hide_C(|||\{\mathcal{T}_P,Top_C^\phi\})}^C \subseteq \alpha(\mathcal{T}_P).$$

Compositional Results. We now report some compositionality results for the insecure communication scenario which, as we will see, can simplify the analysis.

To begin with, we let:

$$\mathcal{T}_1 = \text{hide}_{\Sigma_{com}^P}(|||\{\mathcal{T}_S, \mathcal{T}_{IC}\}) \text{ and } \mathcal{T}_2 = \text{hide}_{\Sigma_{com}^P}(|||\{\mathcal{T}_R, \mathcal{T}_{IC}\}).$$

We then let $\mathcal{T}_P$ be the team automaton defined at the end of Section 6.3, *i.e.*, with $\Sigma_{com}^I = C$ added to $\mathcal{T}_{IC}$. Therefore, $\mathcal{T}_P$ represents the communication scenario in which an initiator and a responder are connected by an insecure channel, but are not connected to the intruder. If we add the most general intruder, some general compositional results can be proved. To this aim we let

$$\mathcal{T}'_1 = \text{hide}_C(|||\{\mathcal{T}_1, Top_C^\phi\}) \text{ and } \mathcal{T}'_2 = \text{hide}_C(|||\{\mathcal{T}_2, Top_C^\phi\}).$$

The following lemma states that the observational behavior of the insecure scenario that sees $\mathcal{T}_1$, $\mathcal{T}_2$ interacting with the intruder Top_C^ϕ, can be obtained as a shuffle of the observational behaviors of $\mathcal{T}'_1$ and $\mathcal{T}'_2$.

Lemma 6.4.7 *Let* $\{m \mid \{c(m) \in \Sigma^P_{com}\} \subseteq \phi$. *Then*

$$\mathbf{O}^C_{\mathit{hide}_C(|||\{|||\{\mathcal{T}_1,\mathcal{T}_2\},\mathit{Top}^\phi_C\})} = \underline{||}_{\{\Sigma^{\mathcal{T}'_1},\Sigma^{\mathcal{T}'_2}\}} \{\mathbf{O}^C_{\mathcal{T}'_1}, \mathbf{O}^C_{\mathcal{T}'_2}\}.$$

Proof. From the way $\mathcal{T}'_1$ and $\mathcal{T}'_2$ are composed it follows that $\Sigma^{\mathcal{T}'_1}_{ext} = \Sigma^S_{sig}$ and $\Sigma^{\mathcal{T}'_2}_{ext} = \Sigma^R_{sig}$. Now let $\mathcal{T}'' = \text{hide}_C(|||\{|||\{\mathcal{T}_1, \mathcal{T}_2\}, \mathit{Top}^\phi_C\})$. It remains to prove that $\mathbf{O}^C_{\mathcal{T}''} = \underline{||}_{\{\Sigma^{\mathcal{T}'_1},\Sigma^{\mathcal{T}'_2}\}} \{\mathbf{O}^C_{\mathcal{T}'_1}, \mathbf{O}^C_{\mathcal{T}'_2}\}$. Since $\Sigma^S_{sig} \cap \Sigma^R_{sig} = \varnothing$, this follows directly from the fact that $\{m \mid \{c(m) \in \Sigma^P_{com}\} \subseteq \phi$, *i.e.*, adding $\mathcal{T}_R$ ($\mathcal{T}_S$) to $\mathcal{T}'_1$ ($\mathcal{T}'_2$) does not change the signals from Σ^S_{sig} (Σ^R_{sig}) which $\mathcal{T}_S$ ($\mathcal{T}_R$) can output because all messages that $\mathcal{T}_R$ ($\mathcal{T}_S$) can send to $\mathcal{T}_{IC}$ have already been included in the initial knowledge of $\mathcal{T}_X$. ■

The previous lemma is used to prove a compositional result over the GNDC specification for team automata. The following theorem says that if $\mathcal{T}_1$ and $\mathcal{T}_2$ satisfy GNDC with respect to two properties $\alpha(\mathcal{T}_1)$ nd $\alpha(\mathcal{T}_2)$ respectively, their composition satisfies GNDC with respect to the shuffle of these properties.

Theorem 6.4.8 *If* $\mathcal{T}_1 \in GNDC^{\alpha(\mathcal{T}_1)}_{\subseteq}$ *and* $\mathcal{T}_2 \in GNDC^{\alpha(\mathcal{T}_2)}_{\subseteq}$, *then*

$$|||\{\mathcal{T}_1, \mathcal{T}_2\} \in GNDC^{\underline{||}_{\{\Sigma^{\mathcal{T}_1},\Sigma^{\mathcal{T}_2}\}}\{\alpha(\mathcal{T}_1),\alpha(\mathcal{T}_2)\}}_{\subseteq}$$

Proof. Let $\mathcal{T}_1 \in GNDC^{\alpha(\mathcal{T}_1)}_{\subseteq}$ and $\mathcal{T}_2 \in GNDC^{\alpha(\mathcal{T}_2)}_{\subseteq}$. Then $\mathbf{O}^C_{\mathcal{T}'_1} \subseteq \alpha(\mathcal{T}_1)$ and $\mathbf{O}^C_{\mathcal{T}'_2} \subseteq \alpha(\mathcal{T}_2)$ and thus, by Lemma 6.4.7 and Remark 6.2.11,

$$\begin{aligned}\mathbf{O}^C_{\text{hide}_C(|||\{|||\{\mathcal{T}_1,\mathcal{T}_2\},\mathit{Top}^\phi_C\})} &= \underline{||}_{\{\Sigma^{\mathcal{T}'_1},\Sigma^{\mathcal{T}'_2}\}}\{\mathbf{O}^C_{\mathcal{T}'_1}, \mathbf{O}^C_{\mathcal{T}'_2}\} \\ &\subseteq \underline{||}_{\{\Sigma^{\alpha(\mathcal{T}_1)},\Sigma^{\alpha(\mathcal{T}_2)}\}}\{\alpha(\mathcal{T}_1), \alpha(\mathcal{T}_2)\}\end{aligned}$$

■

6.5 A Case Study: The EMSS Protocol

The EMSS protocol was introduced in [173] and is used to sign digital streams. It exploits a combination of hash functions and digital signatures and achieves robustness against packet loss, *i.e.*, an incompletely received stream may still allow the user to verify the integrity of the packets that were not lost.

Actually, EMSS is a family of protocols and here we focus on its deterministic (1,2) schema. We assume that a sender S wants to send a stream of payloads $m_0, m_1, \ldots, m_{\mathit{last}}$ to a set of receivers $\{R_n \mid n \geq 1\}$ (as usual for recipients of digital data streams, we assume that receivers are not able to communicate to each other). The protocol then requires S to send tuples built from payloads (called packets) to the receivers.

$$\begin{array}{llrcll}
S \xrightarrow{P_0} \{R_n \mid n \geq 1\} & \text{packet} & P_0 & = & \langle 0, m_0, \varnothing, \varnothing\rangle & \\
S \xrightarrow{P_1} \{R_n \mid n \geq 1\} & \text{packet} & P_1 & = & \langle 1, m_1, h(P_0), \varnothing\rangle & \\
S \xrightarrow{P_i} \{R_n \mid n \geq 1\} & \text{packet} & P_i & = & \langle i, m_i, h(P_{i-1}), h(P_{i-2})\rangle & 2 \leq i \leq \mathit{last} \\
S \xrightarrow{P_{\mathit{sign}}} \{R_n \mid n \geq 1\} & \text{packet} & P_{\mathit{sign}} & = & \langle \{h(P_{\mathit{last}}), h(P_{\mathit{last}-1})\}_{sk(S)}\rangle &
\end{array}$$

After the first two messages, each packet P_i contains a meaningful payload m_i, together with the hashes $h(P_{i-1})$ and $h(P_{i-2})$ of the previous two packets sent. The end of a stream is indicated by a signature packet P_{sign} containing the hashes of the final two packets, along with a digital signature. We assume that the private sender key $sk(S)$ cannot be deduced from $\{m_i \mid 0 \leq i \leq \mathit{last}\}$.

6.5.1 The EMSS Protocol Modeled by Team Automata

In this section, we specify the deterministic (1,2) schema of the EMSS protocol with team automata. As already done for the specification of Top_C^{ϕ}, we omit the precondition (effect) of an action when it is true.

The sender S of the stream is modelled by a CA $\mathcal{T}_S$ and the set $\{R_n \mid n \geq 1\}$ of receivers by n copies of a CA $\mathcal{T}_R$. $\mathcal{T}_S$ uses its private key $sk(\mathcal{T}_S)$ and a public key $pk(\mathcal{T}_S)$ to perform regular digital signature operations. Let Messages denote the set $\{m_0, m_1, \ldots, m_{\mathit{last}}\}$ of meaningful payloads. Then $\mathcal{T}_S$ and $\mathcal{T}_R$ use the hash function h : Messages $\rightarrow$ Hashed. Moreover, $\mathcal{T}_S$ uses the function $s : 2^{\mathsf{Hashed}} \rightarrow$ Signed, defined by $s(H) = H_{sk(\mathcal{T}_S)}$, to sign sets of hashed messages with its private key $sk(\mathcal{T}_S)$, whereas $\mathcal{T}_R$ uses the function $\bar{s}$: Signed $\rightarrow \{\mathsf{true}, \mathsf{false}\}$ and the public key $pk(\mathcal{T}_S)$ to verify whether or not a set of hashed messages was signed by $\mathcal{T}_S$.

In the specification of $\mathcal{T}_S$ we explicitly model that each of its actions is enabled only once during a computation, thus prohibiting loops. For example, as soon as $\mathcal{T}_S$ has sent P_0, then this action's precondition $P_0 \notin \mathsf{sent}$ prohibits this action to be executed again. For the sake of readability, we omit the addition of such preconditions to the specification of $\mathcal{T}_S$ but implicitly assume that all the actions are executed only once during a computation. Note that each packet contains the packet number; in the sequel we denote the packet with packet number i, by P_i.

$\mathcal{T}_S$

Actions

Inp: $\varnothing$

Out: $\{\overbrace{\langle 0, m_0, \varnothing, \varnothing\rangle}^{P_0}, \overbrace{\langle 1, m_1, h(P_0), \varnothing\rangle}^{P_1}\} \cup \{\overbrace{\langle i, m_i, h(P_{i-1}), h(P_{i-2})\rangle}^{P_i} \mid 2 \leq i \leq \mathit{last}\} \cup \{\underbrace{\langle\{h(P_{\mathit{last}}), h(P_{\mathit{last}-1})\}_{sk(\mathcal{T}_S)}\rangle}_{P_{sign}}\}$

Int: $\{\mathit{Hash}_i \mid 0 \leq i \leq \mathit{last}\} \cup \{\mathit{Sign}\}$

States

sent $\subseteq$ Messages, hashed $\subseteq$ Hashed, signed $\subseteq$ Signed, all initially $\varnothing$

Transitions

P_0

Eff: sent := sent $\cup \{P_0\}$

Hash_i, $0 \leq i \leq \mathit{last}$

Pre: $P_i \in$ sent $\wedge\ h(P_i) \notin$ hashed

Eff: hashed := hashed $\cup \{h(P_i)\}$

P_1

Pre: $h(P_0) \in$ hashed $\wedge\ P_1 \notin$ sent

Eff: sent := sent $\cup \{P_1\}$

$P_i,\ 2 \leq i \leq \mathit{last}$
Pre: $\{h(P_{i-1}), h(P_{i-2})\} \subseteq \mathsf{hashed} \wedge P_i \notin \mathsf{sent}$
Eff: $\mathsf{sent} := \mathsf{sent} \cup \{P_i\}$

Sign
Pre: $h(P_{\mathit{last}}) \in \mathsf{hashed} \wedge s(\{h(P_{\mathit{last}}), h(P_{\mathit{last}-1})\}) \notin \mathsf{signed}$
Eff: $\mathsf{signed} := \mathsf{signed} \cup \{s(\{h(P_{\mathit{last}}), h(P_{\mathit{last}-1})\})\}$

P_{sign}
Pre:$\{h(P_{\mathit{last}}), h(P_{\mathit{last}-1})\}_{sk(\mathcal{T}_S)} \in \mathsf{signed} \wedge P_{\mathit{sign}} \notin \mathsf{sent}$
Eff: $\mathsf{sent} := \mathsf{sent} \cup \{P_{\mathit{sign}}\}$

Clearly $\mathcal{T}_S$ has no input, while its output behavior $\mathbf{B}_{\mathcal{T}_S}^{\Sigma_{out}}$ consists of all prefixes of $P_0 P_1 \cdots P_{\mathit{last}} P_{\mathit{sign}}$. To send the packets $P_0, P_1, \ldots, P_{\mathit{last}}, P_{\mathit{sign}}$ in this order, $\mathcal{T}_S$ must perform some internal computations. This is reflected by its internal behavior $\mathbf{B}_{\mathcal{T}_S}^{\Sigma_{int}}$ consisting of all prefixes of $\mathit{Hash}_0 \mathit{Hash}_1 \cdots \mathit{Hash}_{\mathit{last}} \mathit{Sign}$.

We now continue with the specification of $\mathcal{T}_R$. $\mathcal{T}_R$ is capable of receiving as input behavior packets from $P_0, P_1, \ldots, P_{\mathit{last}}$, in the corresponding variables P_i', for $i = 0, \ldots, \mathit{last}$. Eventually $\mathcal{T}_R$ receives the signature packet P_{sign}, that ends the receiving phase. After, $\mathcal{T}_R$ verifies the accompanying digital signature of P_{sign}'(we assume that $\mathcal{T}_R$ has previously retrieved the public key $pk(\mathcal{T}_S)$ corresponding to the private key $sk(\mathcal{T}_S)$); the verification of the signature allows $\mathcal{T}_R$ to be sure of the integrity of the stream of verifiable payloads collected in $\mathsf{xtractedM}$, which are going to be sent to the application as output behavior of $\mathcal{T}_R$. The verification of the digital signature triggers the verification of the stream of the packets received. For $i = \mathit{last}, \ldots, 0$, after $\mathcal{T}_R$ has verified P_i', $\mathcal{T}_R$ verifies whether it has received P_{i-1}'. If it is the case, $\mathcal{T}_R$ extracts the hash h_{i-1} from P_i', computes the hash $h(P_{i-1}')$, and compares these two hashes. If they are equal, then the variable m_{i-1}' that should contain the verifiable payload m_{i-1} is extracted from P_{i-1}'. Otherwise $\mathcal{T}_R$ has no output behavior. On the other hand, if $\mathcal{T}_R$ did not receive P_{i-1}' then $\mathcal{T}_R$ verifies whether it received P_{i-2}'. If $\mathcal{T}_R$ did receive P_{i-2}', then it extracts the hash h_{i-2} from P_i', computes the hash $h(P_{i-2}')$, and compares the two hashes. If they are equal, then the variable m_{i-2}' that should contain the verifiable payload m_{i-2} is extracted from P_{i-2}. Otherwise $\mathcal{T}_R$ has no output behavior. As already done for the specification of $\mathcal{T}_S$ we omit in the specification of $\mathcal{T}_R$ the addition of preconditions that avoid loops; we implicitly assume that all the actions are executed only once during a computation.

$\mathcal{T}_R$

Actions

Inp: $\{\overbrace{\langle 0, m_0', \varnothing, \varnothing\rangle}^{P_0'}, \overbrace{\langle 1, m_1', h_0), \varnothing\rangle}^{P_1'}\} \cup \{\overbrace{\langle i, m_i', h_{i-1}', h_{i-2}\rangle}^{P_i'} \mid 2 \leq i \leq \mathit{last}\}$

$\cup\ \{\underbrace{\langle\{h_{\mathit{last}}, h_{\mathit{last}-1}\}_{sk(\mathcal{T}_S)}\rangle}_{P_{\mathit{sign}}'}\}$

Out: $\mathsf{Payloads}'$
Int: $\{\mathit{XtractH}_i, \mathit{XtractM}_i, \mathit{Hash}_i \mid 0 \leq i \leq \mathit{last}\} \cup \{\mathit{Verify}, \mathit{Stream}\}$

States

received, xtractedM $\subseteq$ Payloads$'$, xtractedH, hashed $\subseteq$ Hashed, all initially $\varnothing$
$\{\{\text{verified}_i | i = 0, \ldots, \mathit{last}, \mathit{sign}\}, \text{send}\} \subseteq \{\text{true}, \text{false}\}$, initially false

Transitions

$P'_i,\ 0 \leq i \leq \mathit{last}$
Pre: $P'_{\mathit{sign}} \notin$ received
Eff: received := received $\cup \{P'_i\}$

$\mathit{XtractH}_{i,1},\ 1 \leq i \leq \mathit{last}$
Pre: $[\{P'_{i-1}, \overbrace{\langle i, m'_i, h_{i-1}, h_{i-2}\rangle}^{P'_i}\} \subseteq \text{received}] \wedge [\text{verified}_i = \text{true}]$
Eff: xtractedH := xtractedH $\cup \{h_{i-1}\}$

$\mathit{XtractH}_{i,2},\ 2 \leq i \leq \mathit{last}$
Pre: $[\{P'_{i-2}, \overbrace{\langle i, m'_i, h_{i-1}, h_{i-2}\rangle}^{P'_i}\} \subseteq \text{received}] \wedge [P_{i-1} \notin \text{received}] \wedge [\text{verified}_i = \text{true}]$
Eff: xtractedH := xtractedH $\cup \{h_{i-2}\}$

P'_{sign}
Eff: received := received $\cup \{P'_{\mathit{sign}}\}$

Verify
Pre: $[P'_{\mathit{sign}} \in \text{received}] \wedge [\bar{s}(\{\overbrace{h_{\mathit{last}}, h_{\mathit{last}-1}}^{P'_{\mathit{sign}}}\}_{sk(\mathcal{T}_S)}) = \text{true}]$
Eff: $\text{verified}_{\mathit{sign}} := \text{true}$, xtractedH := xtractedH $\cup \{h_{\mathit{last}}, h_{\mathit{last}-1}\}$

$\mathit{XtractH}_{\mathit{sign},1}$
Pre: $[\{P'_{\mathit{last}}, \overbrace{\langle h_{\mathit{last}}, h_{\mathit{last}-1}\rangle}^{P'_{\mathit{sign}}}\} \subseteq \text{received}] \wedge [\text{verified}_{\mathit{sign}} = \text{true}]$
Eff: xtractedH := xtractedH $\cup \{h_{\mathit{last}}\}$

$\mathit{XtractH}_{\mathit{sign},2}$
Pre: $[\{P'_{\mathit{last}-1}, \overbrace{\langle h_{\mathit{last}}, h_{\mathit{last}-1}\rangle}^{P'_{\mathit{sign}}}\} \subseteq \text{received}] \wedge [P'_{\mathit{last}} \notin \text{received}] \wedge [\text{verified}_{\mathit{sign}} = \text{true}]$

Eff: xtractedH := xtractedH $\cup \{h'_{\mathit{last}-1}\}$

$\mathit{Hash}_i,\ 0 \leq i \leq \mathit{last}$
Pre: $h_i \in$ xtractedH $\wedge [P'_i \in \text{received}]$
Eff: hashed := hashed $\cup \{h(P'_i)\}$

$\mathit{XtractM}_i,\ 0 \leq i \leq \mathit{last}$
Pre: $[h_i \in \text{xtractedH}] \wedge [h(P'_i) \in \text{hashed}] \wedge [h(P'_i) = h_i]$
Eff: xtractedM := xtractedM $\cup \{m'_i\}$, $\text{verified}_i := \text{true}$

Stream
Pre: $[[m'_{last} \in \mathsf{xtractedM}] \vee [[m'_{last-1} \in \mathsf{xtractedM}] \wedge [P'_{last} \notin \mathsf{received}]]]$
$\wedge[\mathsf{verified}_{sign} = \mathsf{true}]$
Eff: $\mathsf{send} := \mathsf{true}$

m'_0
Pre: $[\mathsf{send} = \mathsf{true}] \wedge [m'_0 \in \mathsf{xtractedM}] \wedge [\mathsf{verified}_0 = \mathsf{true}]$
Eff:$\mathsf{xtractedM} := \mathsf{xtractedM} - \{m'_0\}$

$m'_i,\ 1 \leq i \leq last$
Pre: $[\mathsf{send} = \mathsf{true}] \wedge [m'_i \in \mathsf{xtractedM}]$
$\wedge[\{m'_k \mid i \leq k \leq last\} \cap \mathsf{xtractedM} = \varnothing] \wedge [\mathsf{verified}_i = \mathsf{true}]$
Eff: $\mathsf{xtractedM} := \mathsf{xtractedM} - \{m'_i\}$

Remark 6.5.1 In the $\mathcal{T}_R$ model, we have explicitly inverted (with respect to the specification of EMSS) the order of messages in the output behavior of $\mathcal{T}_R$. Without losing generality and without changing the final results of our analysis, this choice simplifies some technicalities. Observe that the first message of the output sequence of $\mathcal{T}_R$ must necessarily be either m'_{last} or m'_{last-1}. ■

We now go on with the construction the formal model of EMSS, $\mathcal{T}_{EMSS}$. It is defined as the max-ai team automaton over $\{\mathcal{T}_S, \mathcal{T}_R^{(i)} \mid 1 \leq i \leq n\}$. Formally:

$$\mathcal{T}_{EMSS} = |||\,\{\mathcal{T}_S, \mathcal{T}_R^{(i)} \mid 1 \leq i \leq n\},$$

Note that $\mathcal{T}_{EMSS}$ has no input actions, while it has the union of the output (resp.,internal) actions of $\mathcal{T}_S$ and the $\mathcal{T}_R$'s as its output (resp.,internal) actions.

6.5.2 Analysis of the EMSS Protocol

In this section we use the GNDC schema for team automata together with the insecure communication scenario in order to show that the deterministic (1,2) schema of the EMSS protocol guarantees integrity. Note that this has already been validated in [149], where a CCS-like process algebra was used instead. Our goal here is thus to use this particular case study to show the effectiveness of team automata for security analysis.

We model the sender S by $\mathcal{T}_S$ and the receiver by $\mathcal{T}_R$. While here we consider one $\mathcal{T}_R$, this analysis can be extended in a natural way to the case in which there are n copies of $\mathcal{T}_R$. We formally define integrity as the ability of $\mathcal{T}_R$ to accept a message m_i, for any i, only if it has indeed been sent by $\mathcal{T}_S$. We also assume that $\mathcal{T}_R$ signals the acceptance of a stream of messages as a legitimate stream by issuing it as a list of messages $\{\mathtt{Reveal}'\}$. We require the expected (correct) observational behavior $\alpha_{int}(\mathcal{T}_P)$ of $\mathcal{T}_P$ with respect to integrity as the set containing all prefixes of the subsequence (holes are due to packets loss) of $\mathtt{Reveal}'(m_{i_{last}}) \cdots \mathtt{Reveal}'(m_{i_1})\mathtt{Reveal}'(m_{i_0})$. Formally:

$$\alpha_{int} \stackrel{\text{def}}{=} \{\mathtt{Reveal}'(m_{i_{last}}) \cdots \mathtt{Reveal}'(m_{i_1})\mathtt{Reveal}'(m_{i_0}) \mid 0 \leq i_0 < \ldots < i_{last} \leq last\}$$

Further, we equip Top_C^{ϕ} with an initial knowledge ϕ consisting of all output actions of $\mathcal{T}_S$ and the public key $pk(\mathcal{T}_S)$, *i.e.*, $\phi = \{P_0, P_1, P_i, P_{sign} \mid 2 \leq i \leq last\} \cup \{pk(\mathcal{T}_S)\}$, where $P_0 =$

$\langle 0, m_0, \varnothing, \varnothing\rangle$, $P_1 = \langle 1, m_1, h(P_0), \varnothing\rangle$, $P_i = \langle i, m_i, h(P_{i-1}), h(P_{i-2})\rangle$, for all $1 \leq i \leq$ *last*, and $P_{sign} = \langle\{h(P_{last}), h(P_{last-1})\}_{sk(\mathcal{T}_S)}\rangle$. We do so solely for analysis reasons, *viz.* in order to enable Top_C^{ϕ} to send the correct messages to $\mathcal{T}_R$ through the insecure channel. Note that the messages contained in this initial knowledge are exactly those that the intruder is anyway able to collect by eavesdropping what $\mathcal{T}_S$ sends through the insecure channel. As is common in security analysis, we rely on the *perfect encryption assumption*, *i.e.*, Top_C^{ϕ} cannot deduce $sk(\mathcal{T}_S)$ from ϕ nor can it forge hash and encryption functions by guessing. From Section 6.4.2 we recall that

$$\mathcal{T}_1 = \text{hide}_{\Sigma_{com}^P}(|||\{\mathcal{T}_S, \mathcal{T}_{IC}\}), \quad \mathcal{T}_2 = \text{hide}_{\Sigma_{com}^P}(|||\{\mathcal{T}_R, \mathcal{T}_{IC}\})$$

$$\mathcal{T}_1' = \text{hide}_C(|||\{\mathcal{T}_1, Top_C^{\phi}\}), \qquad \mathcal{T}_2' = \text{hide}_C(|||\{\mathcal{T}_2, Top_C^{\phi}\})$$

Hence the observational behavior of the max-ai team automaton over $\mathcal{T}_1$ and Top_C^{ϕ} is empty, therefore

Lemma 6.5.2 $\mathcal{T}_1 \in GNDC_{\subseteq}^{\varnothing}$.

Proof. Directly by Corollary 6.4.6 because $\mathbf{O}^C_{\text{hide}_C(|||\{\mathcal{T}_1, Top_C^{\phi}\})} = \varnothing$. ∎

We now show that the observational behavior of the max-ai team automaton over $\mathcal{T}_2$ and Top_C^{ϕ} is included in the expected observational behavior $\alpha_{int}(\mathcal{T}_P)$ of $\mathcal{T}_P$ with respect to integrity,

Lemma 6.5.3 $\mathcal{T}_2 \in GNDC_{\subseteq}^{\alpha_{int}(\mathcal{T}_P)}$.

Proof. Recall that the behavior of $\mathcal{T}_2$ coincides with $\mathcal{T}_R$ when it interacts with the intruder. Let us concentrate on the observable (output) behavior of $\mathcal{T}_R$ in $\mathcal{T}_2$.

If $\mathcal{T}_R$ shows an empty output behavior $\varnothing$, the theorem is trivially satisfied. Otherwise the output behavior of $\mathcal{T}_R$ is a sequence of messages. From Remark 6.5.1 the first message of this sequence must be either $\texttt{Reveal}'(m'_{last})$ or $\texttt{Reveal}'(m'_{last-1})$. We treat only the case in which the first message is $\texttt{Reveal}'(m'_{last})$; the other case is analogous. In the following we omit the predicate $\texttt{Reveal}'(\_)$, for sake of conciseness.

First we prove the following statement:

Claim 6.5.4 *Let $P_0, \ldots, P_{last}, P_{sign}$ be the correct packets sent by $\mathcal{T}_S$. If $\mathcal{T}_R$ is in state where* $\mathsf{verified}_i = \mathsf{true}$ *and* $\{P_i', P_{sign}\} \subseteq \mathsf{received}$ *then* $P_i' = P_i$.

Proof of Claim 6.5.4 By induction over $i = \mathsf{last}, \ldots, 0$, in which the base case is $i =$ *last*.

base case: ($i = \mathsf{last}$). By the precondition of $XtractM_{last}$, $\mathsf{verified}_{last} = \mathsf{true}$ implies that $h_{last} \in \mathsf{xtractedH}$, $h(P'_{last}) \in \mathsf{hashed}$ and $h_{last} = h(P'_{last})$. By the precondition of *Verify*, $h_{last} \in \mathsf{xtractedH}$ implies that $P'_{sign} \in \mathsf{received}$ and $\mathsf{verified}_{sign} = \mathsf{true}$. From the hypothesis we know that $P_{sign} = \langle h(P_{last}), h(P_{last-1})\rangle$ has been received (in P'_{sing}), so (by the precondition of *Verify*) $\mathsf{verified}_{sign} = \mathsf{true}$ implies that h_{last} is indeed $h(P_{last})$ *i.e.*, it coincides with the hash of the correct packet P_{last}. For the properties of hash functions in cryptography this means that $P'_{last} = P_{last}$.

inductive step: assume that the hypothesis holds for $j > i$. By the precondition of $XtractM_i$, $\mathsf{verified}_i = \mathsf{true}$ provided that $h_i \in \mathsf{xtractedH}$, $h(P_i') \in \mathsf{hashed}$ and $h_i = h(P_i')$. Since $h_i \in \mathsf{xtractedH}$, either the precondition of $XtractH_{i,1}$ or the precondition of $XtractH_{i,2}$) must hold. Assume that the precondition of $XtractH_{i,1}$ holds (the other case is analogous).

We have that $\{P'_i, P'_{i+1}\} \subseteq$ received and $\mathsf{verified}_{i+1} = \mathsf{true}$. By hypothesis, P_{sign} is received and, by the inductive hypothesis, $P'_{i+1} = P_{i+1}$ *i.e.*, P'_{i+1} is authentic. By the precondition of $XtractH_{i,1}$ and by the hypothesis that $\mathsf{verified}_i = \mathsf{true}$ it follows that the hash h_i, extracted from $P'_{i+1} = \langle m_{i+1}, h_i, h_{i-1}\rangle$, is $h(P_i)$ *i.e.*, h_i coincides with $h(P_i)$, the hash of the correct packet P_i. For the properties of hash functions in cryptography this means that $P'_i = P_i$. *End of the proof of Claim* 6.5.4.

Now the result follows from the fact that: (a) the intruder is not able to forge P_{sign} because it doesn't know the private key $sk(\mathcal{T}_S)$ of $\mathcal{T}_S$; (b) by the precondition of actions m'_i (for $i = 0, \ldots,$ *last*) in $\mathcal{T}_R$, messages are revealed by $\mathcal{T}_R$ (hence by $\mathcal{T}_2$) in reverse order (with respect to the packet number). ■

Finally, after an observation on the composition of team automata that have no internal actions, we can show that integrity is guaranteed in the instance of the EMSS protocol under scrutiny.

Remark 6.5.5 If $\{\mathcal{T}, \mathcal{T}\}$ is a composable system, then clearly $\mathbf{B}_{|||\{\mathcal{T},\mathcal{T}\}} = \mathbf{B}_{\mathcal{T}}$. ■

Theorem 6.5.6 $\mathcal{T}_P \in GNDC^{\alpha_{int}(\mathcal{T}_P)}_{\subseteq}$.

Proof. From Lemma 6.5.2 and 6.5.3 and Theorem 6.4.8 it follows that

$$\begin{aligned} |||\{\mathcal{T}_1, \mathcal{T}_2\} &\in GNDC^{\overset{||}{\subseteq}_{\{\Sigma^{\mathcal{T}_1}, \Sigma^{\mathcal{T}_2}\}}\ \{\mathbf{O}^C_{\mathcal{T}_1}, \mathbf{O}^C_{\mathcal{T}_2}\}} \\ &= GNDC^{\overset{||}{\subseteq}_{\{\mathtt{Reveal}'\}}\ \{\varnothing, \alpha_{int}(\mathcal{T}_P)\}} \\ &= GNDC^{\alpha_{int}(\mathcal{T}_P)}_{\subseteq} \end{aligned}$$

Then by Corollary 6.4.6. $\mathbf{O}^C_{\mathrm{hide}_C(|||\{|||\{\mathcal{T}_1,\mathcal{T}_2\}, Top^{\phi}_C\})} \subseteq \alpha_{int}(\mathcal{T}_P)$. Since $\mathcal{T}_{IC}$ has no internal actions, $\{\mathcal{T}_{IC}, \mathcal{T}_{IC}\}$ forms a composable system, the from Remarks 6.2.8 and 6.5.5 it follows that $\mathbf{B}_{|||\{|||\{\mathcal{T}_1,\mathcal{T}_2\}, Top^{\phi}_C\}} = \mathbf{B}_{|||\{\mathcal{T}_S,\mathcal{T}_R,\mathcal{T}_{IC}, Top^{\phi}_C\}} = \mathbf{B}_{|||\{\mathcal{T}_P, Top^{\phi}_C\}}$ and consequently, that $\mathbf{O}^C_{\mathrm{hide}_C(|||\{|||\{\mathcal{T}_1,\mathcal{T}_2\}, Top^{\phi}_C\})} = \mathbf{O}^C_{\mathrm{hide}_C(|||\{\mathcal{T}_P, Top^{\phi}_C\})}$ by Definition 6.4.1.
Hence $\mathbf{O}^C_{\mathrm{hide}_C(|||\{\mathcal{T}_P, Top^{\phi}_C\})} \subseteq \alpha_{int}(\mathcal{T}_P)$, and thus, by Corollary 6.4.6, $\mathcal{T}_P \in GNDC^{\alpha_{int}(\mathcal{T}_P)}_{\subseteq}$. ■

6.6 Conclusions and Future Work

We use team automata to define a framework for security analysis by constructing a general insecure communication scenario for team automata and by reformulating the GNDC schema in terms of team automata. We also define some effective compositional analysis strategies for this insecure communication scenario. We also investigate strategies of analysis in our framework. We Firstly, we define the most general intruder in terms of team automata. By the use of the most general intruder we are able to avoid the universal quantification presents in the re-formulation of the GNDC schema for team automata. Secondly, we define a compositional analysis strategy for team automata, and we show how security properties are preserved by composition over an initiator and a responder. We use the framework to prove that integrity is guaranteed in a case study in which team automata models the EMSS protocol.

A goal for the future is to try to automate the current manual verification process. Since team automata are an extension of I/O automata, the *IOA Language and Toolset* [91] may be of help when trying to achieve this goal. Another goal for the future is to extend the team automata framework with time, probability, or both. Such extensions of automata-based formalisms are well studied in the literature, *e.g.*, for I/O automata [141, 183]. In this respect, also the well-developed theory of *timed automata* needs to be mentioned [11, 129]. Like their I/O automata counterparts, timed team automata could consider time in the systems they model, whereas probabilistic team automata would allow a probabilistic choice of the next state.

Part IV

Epilogue

7

Conclusions and Future Work

In this thesis we answer the following questions:

Question 1: can security protocol analysis and fault-tolerance analysis benefit from a common background and common strategies?

Question 2: which are the differences and similarities between the various strategies used for security protocol analysis?

We answer question 1 by identifying logic-based model checking as a strategy common to both fault-tolerance analysis and security protocol analysis (Chapter 1 and Chapter 4). In the context of industrial applications, we also show how existing tools can be used effectively in both fields (Chapter 1 and Chapter 2). We prove that a scheme developed in security protocol analysis, the Generalized Non Deducibility on Compositions (GNDC), can be re-formulated in the framework of fault-tolerance analysis (Chapter 3). This result implies that any verification strategy used in the GNDC for security analysis can be applied to fault-tolerance analysis as well. In particular, we show that the "fault-tolerant" property is an instance of GNDC known as BNDC (Bisimulation Non Deducibility on Compositions), and this implies that the existing tools for checking BNDC can be used to check fault-tolerance as well.

Question 2 is answered in two different ways. Firstly, we prove that a bisimulation-like relation exists between security protocols modeled as process in a process algebras and as a theory in multiset rewriting systems (Chapter 5). To obtain this result, we use restricted versions of both formalisms. Those versions are specifically tailored to security protocols. Secondly, we consider Team Automata – an emerging automata-based formalism – and we show that the GNDC scheme can be re-formulated in terms of Team Automata (Chapter 6).

7.1 Conclusions

We formulate the conclusions of this thesis in terms of some general principles and subsidiary, specific statements. Statements are valid throughout this thesis, and generally summarize the lessons learned during the experiences reported in this thesis. Principles have a wider validity. Figure 7.1 shows principles and statements in a graphical form.

From Part I and Part II we learn a lesson concerning the concept of attack in security and the concept of fault in fault-tolerance.

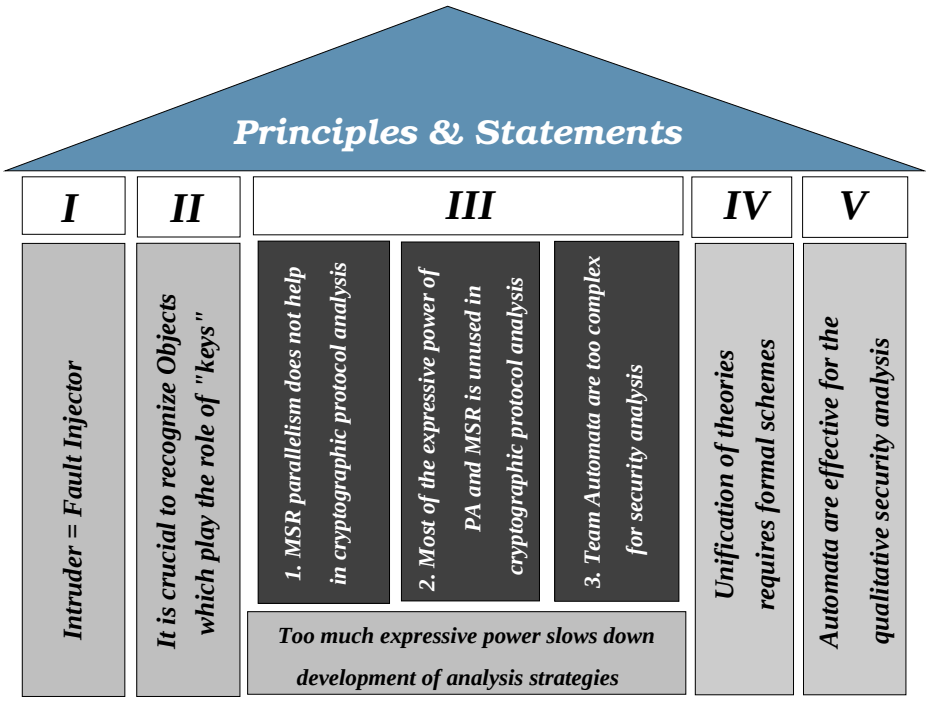


Figure 7.1: The five Principles (in light grey boxes) and the three Statements (in dark boxes) of Integration in Security Protocol Analysis and Fault Tolerance Analysis.

Principle I *The Intruder (in security protocol analysis) and the Fault Injector (in fault-tolerance analysis) are essentially the same entity.*

The intruder and the fault-injector share a fundamental characteristic: they can be modeled as a malicious and active environment trying to subvert system goals. The understanding of this principle is at the basis of the idea of applying techniques from security analysis to fault-tolerance.

The similarities between the intruder and the fault-injector become evident as soon as we make clear the separation between the system model and the environment with which the system interacts. In Chapter 1, where this separation is missing and where the faults are embedded in the system model, this principle is not immediately clear. Contrastingly, in Chapter 3, where this separation is applied to a fault-tolerant system, this similarity is evident; this allows us to bring some strategies from one field to the other. In particular, we are able to characterize fault-tolerance as a logic validation problem in the μ-calculus; in addition, we are able to reformulate the GNDC scheme in the context of fault-tolerant systems. Figure 7.2 illustrates the intersection between security protocol and fault-tolerance analysis that emerges from Principle 1.

From Part II, we learn an important lesson concerning the correct identification of a certain role.

Principle II *In engineering security critical applications, it is essential to realize when an object plays the role of an encryption key.*

This principle emerges from our study of OSA/Parlay architecture (Chapter 2). The simple yet crucial step that leads to the correct analysis is the understanding that interfaces in web applications may play the role of protecting access to a secret: this is exactly the role encryption

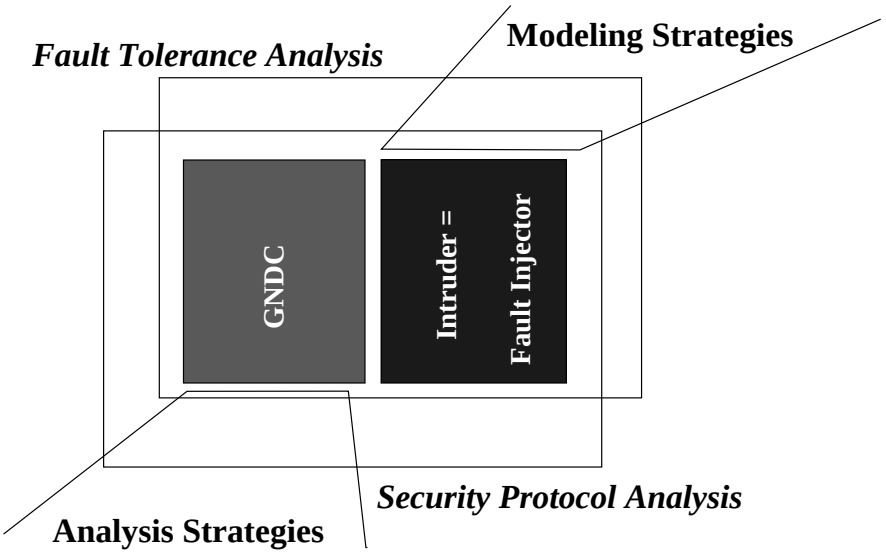


Figure 7.2: Meeting Points between Fault Tolerance Analysis and Security Protocol Analysis

keys have. Every security system has key-like elements, that are sometimes difficult to identify. The application of this principle says that recognizing the "being-key" property in an object is essential, and objects of this kind have to be carefully protected.

We underline also that, while this principle is obvious in theory, in practice things are non-trivial: informal specifications, resulting from long, cooperative development processes, can easily hide a violation of this principle.

Principle III *The use of too much expressive power in a formal modelling notation is counter-productive to the development of analysis strategies* .

This principle says that high expressive power in a formal modelling notation is attractive only when we focus on the modeling activity. High expressiveness hinders when we try to develop effective analysis strategies.

This principle emerges from Chapter 4 and Chapter 5, and it becomes even more evident in Chapter 6. In the following we list three specific statements supporting the principle, each concerning the formal models studied in this thesis:

Statement 1 *Multiset rewriting contains more inherent parallelism than it is required to analyze security protocols.*

Multiset rewriting is a powerful formalism for modeling and analyzing concurrent systems. We do not need all of its power in expressing concurrency when describing and analyzing security protocols. This statement emerges from Chapter 5, where a restricted version of multiset rewriting proves to be sufficient to describe a large class of security protocols.

Statement 2 *Most of the power of process algebras and multiset rewriting is unused when modeling security protocols.*

This statement stems from both Chapter 4 and Chapter 5. In the first, we model traditional

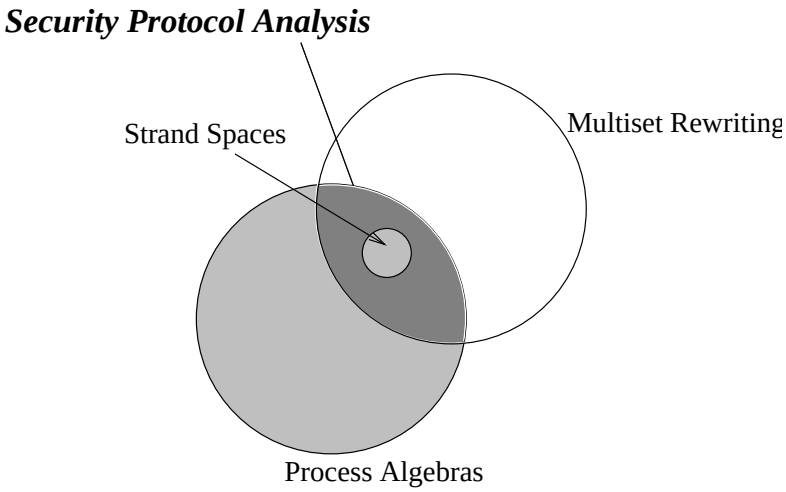


Figure 7.3: Meeting Points between Process Algebra and Multiset Rewriting in Security Protocol Analysis

security protocols without using features like mobility and intra-agent parallelism; security protocols are a parallel composition of sequential agents. Because of this, we are able to build an optimized and more specific model checker. In Chapter 5 we restrict process algebra and multiset rewriting to the security protocol context to obtain a strong result of correlation between (subsets of) the two formalisms.

Statement 1 and Statement 2 are illustrated in Figure 7.3. Here, strand spaces [78] are placed in the intersection. Strand spaces is a formalism that has been proved to be equivalent to a restricted version of multiset rewriting [44]. Indirectly, we prove that strand spaces and process algebras are related as well in the context of security protocol analysis.

Statement 3 *Team Automata models are too complex for the effective support of security protocol analysis.*

Team Automata provides flexible models for the specification of communication in systems, but the flexibility hinders when developing effective strategies of analysis. The possibility of defining different modalities of synchronization among automata brought us to study their application in security protocol analysis. Peer-to-peer and multicast/broadcast communications can be expressed elegantly in team automata. Initially we were thinking of a potential unification between strategies of analysis for security protocols and broadcast/multicast protocols [197]. What seemed an advantage proved to be a disadvantage as soon as we started to develop the GNDC theory required for the development of strategies of security protocol analysis. In the end, in an attempt to control the growth of the number of cases to be considered we were forced to use the very subset of Team Automata required to describe traditional (unicast) security protocols.

Other calculi for the analysis of specific systems – like π-calculus [161] for mobility and (from a certain point of view) spi-calculus [7] for security protocols – show their strength just in their conciseness.

Figure 7.4 depicts the relation between process algebras and Team Automata with respect to the validation of security protocols. GNDC can also be defined in Team Automata terms. Since I/O automata are a special class of Team Automata it follows that GNDC can be re-defined in

terms of I/O automata as well.

Principle IV *In the analysis of security protocols, automata-based languages are more effective in quantitative (real-time and probabilistic) than qualitative analysis of security protocols.*

This principle emerges from our experience of modeling security properties using Team Automata. For example, the flexibility of Team Automata helps in expressing advanced communication paradigms, such as multicast and broadcast [197]. However, in the domain of security protocol analysis, Team Automata tools and analysis techniques are not competitive.

On the other hand, a significant amount of work on automata in timed security (*e.g.*, see [103]), and the existing associated tools (*e.g.*, the real-time model checker UPPAAL) make them suitable for studying timing attacks to security (*e.g.*, see [61]). For what concerns the probabilistic analysis of security, automata based models are, at present, as promising as process-algebraic approaches (*e.g.*, see [131, 130] for automata, and [9] for process algebras).

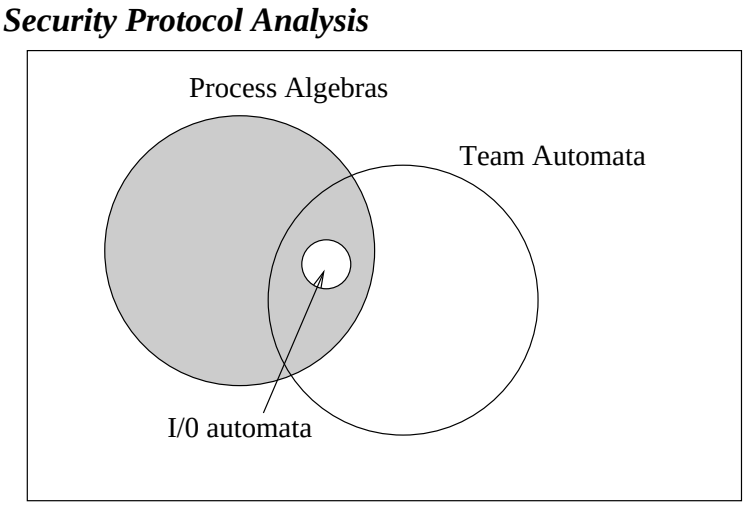


Figure 7.4: Meeting Points between Team Automata and Process Algebra in Security Protocol Analysis

Principle V *Formal schemes of analysis are essential for the unification of analysis techniques.*

Creating a scheme of analysis requires an effort of abstraction, whose goal is to identify the essential entities and their relationships that are required by the analysis. Redefining a scheme in a different formal model requires (only) the modeling of its entities and its relationships in the new formal model. The modeling activity can be technically complex, but the theoretical effort of unification is, in essence, contained in the scheme itself.

For example, for the GNDC scheme, the entities are the system under analysis, its malicious environment, an agent showing the expected behavior of the system, and a notion of "observability" relation. In Chapter 3 we apply the GNDC scheme to fault-tolerance, by identifying the malicious environment with a fault-injector; in Chapter 6 we instantiate GNDC in terms of team automata showing how to model the entities of the scheme as automata. In both cases we reuse analysis techniques that are established for the GNDC scheme.

7.2 Future Work

The research of this thesis can be developed further in at least two different directions: to improve the techniques of analysis, and to extend the principles of integration to emerging disciplines.

First Direction. The analysis techniques developed in each chapter of the thesis can be improved further as follows:

- In Chapter 3 we study a formal specification of fail safe, fail silent, fail stop and fault-tolerance properties in the GNDC scheme. We prove that the "fault-tolerance" property enjoys a precise classification in terms of GNDC, whereas the other properties are completely expressible in GNDC only when we consider their instances in our running examples. A complete formal characterization of these properties is still missing. A goal for the future is to understand better the formal characteristics of fail silent, fail stop and fail safety properties and to conclude the classification of fault-tolerant properties as instances of security (non-interference) properties, by following what has been done in security, for example by Gorrieri and Focardi in [81, 83].

- In Chapter 4 we design a logic based model checker for the analysis of security protocols. Our implementation runs in exponential time in the size of the longest message involved in the protocol. This matches the expected theoretical computational complexity, so it is the best we can expect. We think that our tool performance can be significantly improved by the use of partial order reduction techniques. Moreover, we think that a (front-end module of) static type analysis of the message flow along a protocol specification may help in defining significant transformations, that in turn are used by our tool to improve the efficiency of the dynamic analysis. It is interesting to investigate this area as future work.

- In Chapter 5 we relate process algebras and multiset rewriting in the restricted setting of security protocol analysis. We find a bisimulation-like relation between security protocol models in the two formalisms, that maintains secrecy and authenticity properties. It would be interesting to identify other area of research in where such a 'bisimulation-like" relation can be defined.

- In Chapter 6 we provide Team Automata with a framework for the analysis of security protocols. In this domain, a goal for the future is to automate the current manual verification process. Since team automata are an extension of I/O automata, the *IOA Language and Toolset* [91] may be of help when trying to achieve this goal. Another goal for the future is to extend the team automata framework with time, probability, or both. For this we can benefit from many previous experiences reported in the literature (*e.g.,* see [141, 183, 11, 129]) where such extensions are proposed for different automata-based formalisms.

Second Direction. The principles of integration identified in this thesis have general validity and can be applied to other fields of research as well. One of the possible contexts is privacy control, which we started to investigate in [60]. In this emerging field, the need of instruments for the specification and analysis of privacy policies is compelling. We hope that the usage of the integration principles we have identified in this thesis can help in identifying techniques of process specification and verification techniques also supported by verification tools.

Bibliography

[1] *Open Service Access (OSA) Application Programming Interface (API) Mapping for OSA*. http://www.3gpp.org/ftp/Specs/archive/29_series/29.198-03. Release 5.1.0.

[2] AA.VV. *Failures-Divergence Refinement (FDR2): User Manual and Tutorial*. Formal System (Europe) Ltd, 2000.

[3] ABADI, M. Secrecy by Typing in Security Protocols. *Journal of the ACM 46*, 5 (1999), 749–786.

[4] ABADI, M., AND BLANCHET, B. Analyzing Security Protocols with Secrecy Types and Logic Programs. *ACM SIGPLAN Notices 37*, 1 (2002), 33–44. Proc. of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'02), Jan. 16-18, 2002, Portland, OR, USA.

[5] ABADI, M., AND GORDON, A. D. Reasoning about Cryptographic Protocols in the Spi Calculus. In *Proc. of 8th International Conference on Concurrency Theory (CONCUR 97), July 1-4, 1997, Warsaw, Poland* (1997), A. W. Mazurkiewicz and J. Winkowski, Eds., vol. 1243 of *LNCS*, Springer-Verlag, pp. 59–73.

[6] ABADI, M., AND GORDON, A. D. A Bisimulation Method for Cryptographic Protocols. In *Proc. of the 7th European Symposium on Programming (ESOP'98) – held as part of the Joint European Conferences on Theory and Practice of Software (ETAPS'98) – March 28 - April 4, 1998, Lisbon, Portugal* (1998), C. Hankin, Ed., vol. 1381 of *LNCS*, Springer-Verlag, pp. 12–26.

[7] ABADI, M., AND GORDON, A. D. A Calculus for Cryptographic Protocols. The Spi Calculus. Tech. Rep. 149, Digital Equipment Corporation Systems Research Center, Palo Alto, California, 1998.

[8] ABADI, M., AND NEEDHAM, R. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering 22*, 1 (1996), 6–15.

[9] ALDINI, A., BRAVETTI, M., AND GORRIERI, R. A Process-algebraic Approach fo the Analysis of Probabilistic Non-Interference. *Journal of Computer Security 12*, 2 (March 2004), 191–245.

[10] ALPERN, B., AND SCHNEIDER, F. B. Defining Liveness. *Information Processing Letters 21*, 4 (October 1985), 181–185.

[11] ALUR, R., AND DILL, D. L. A Theory of Timed Automata. *Theoretical Computer Science 126*, 2 (1994), 183–235.

[12] AMADIO, R. M., AND LUGIEZ, D. On the Reachability Problem in Cryptographic Protocols. In *Proc. of 11th International Conference on Concurrency Theory (CONCUR 2000), Aug. 22-25, 2000, University Park, PA, USA* (2000), C. Palamidessi, Ed., vol. 1877 of *LNCS*, Springer-Verlag, pp. 380–394.

[13] AMENDOLA, A. M. Dependability of Railway Control Systems. In *Proc. of the 26th International Symposium on Fault-Tolerant Computing (FTCS-26), June 25-27, 1996, Sendai, Japan* (1996), IEEE Computer Society Press, pp. 150–155. Panel.

[14] AMENDOLA, A. M., IMPAGLIAZZO, L., MARMO, P., AND POLI, F. Experimental Evaluation of Computer-Based Railway Control Systems. In *Proc. of the 27th Annual International Symposium on Fault-Tolerant Computing (FTCS-27), June 24-27, 1997, Seattle, Washington, USA* (1997), IEEE Computer Society Press, pp. 380–384.

[15] ANDERSEN, H. R. *Verification of Temporal Properties of Concurrent Systems*. PhD thesis, Department of Computer Science, Aarhus University, Denmark, 1993.

[16] ANDERSEN, H. R. Partial model checking (extended abstract). In *Proc. of 10th Annual IEEE Symposium on Logic in Computer Science (LICS 1995), June 26-29, 1995, San Diego, California, USA* (1995), IEEE Computer Society Press, pp. 398–407.

[17] ANDERSEN, H. R., AND LIND-NIELSEN, J. Partial model checking of modal equations: A survey. *Software Tools for Technology Transfer 3*, 2 (1999), 242–259.

[18] BEEK, M. H., ELLIS, C. A., KLEIJN, J., AND ROZENBERG, G. Synchronizations in Team Automata for Groupware Systems. *Computer Supported Cooperative Work - The Journal of Collaborative Computing 12*, 1 (2003), 21–69.

[19] BERNARDESCHI, C., FANTECHI, A., AND GNESI, S. Model Checking Fault Tolerant Systems. *Software Testing, Verification and Reliability 12*, 4 (December 2002), 251–275.

[20] BERNARDESCHI, C., FANTECHI, A., GNESI, S., LAROSA, S., MONGARDI, G., AND ROMANO, D. A Formal Verification Environment for Railway Signaling System Design. *Formal Methods in System Design 12*, 2 (March 1998), 139–161.

[21] BERNARDESCHI, C., FANTECHI, A., AND SIMONCINI, L. Formally Verifying Fault Tolerant System Designs. *The Computer Journal 43*, 3 (2000), 191–205.

[22] BEVIER, W., AND YOUNG, W. Machine Checked Proofs of the Design and Implementation of a Fault-Tolerant Circuit. Tech. Rep. NAS1-18878, NASA, 1990.

[23] BHAT, G., AND CLEAVELAND, R. Efficient model checking via the equational μ-calculus. In *Proc. of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 1996), July 27-30, New Brunswick, NJ, USA* (1996), IEEE Computer Society Press, pp. 304–312.

[24] BISTARELLI, S., CERVESATO, I., LENZINI, G., AND MARTINELLI, F. Relating multiset rewriting and process algebras for security protocol analysis. In *Proc. of the IFIP WG 1.7 ACM SIGPLAN and GI FoMSESS Workshop on Issues in the Theory of Security (WITS'03) –*

held as part of the European Joint Conferences on Theory and Practice of Software (ETAPS 2003) – Apr. 5-13, 2003, Warsaw, Poland (2003), R. Gorrieri, Ed., Informal Proceedings, pp. 21–31. (full version selected for publication in the Journal of Computer Security).

[25] BISTARELLI, S., CERVESATO, I., LENZINI, G., AND MARTINELLI, F. Relating process algebras and multiset rewriting for immediate decryption protocols. In *Proc. of the Second International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security (MMM-ACNS 2003), Sept. 20-24, 2003, St. Petersburg, Russia* (2003), V. A. S. V. I. Gorodetski and L. J. Popyack, Eds., vol. 2776 of *LNCS/LNAI*, Springer-Verlag, pp. 86–99.

[26] BISTARELLI, S., CERVESATO, I., LENZINI, G., AND MARTINELLI, F. Relating multiset rewriting and process algebras for security protocol analysis. *Journal of Computer Security 13*, 1 (February 2005), 3–47.

[27] BORÄLV, A. A Case Study: Formal Verification of a Computerized Railway Interlocking. *Formal Aspect of Computing 10*, 4 (1998), 338–360.

[28] BORÄLV, A. A Fully Automated Approach for Proving Safety Properties in Interlocking Software Using Automatic Theorem-Proving. In *Proc. of the 2nd International ERCIM Workshop on Formal Methods Industrial Critical Systems (FMICS 97) – held as part of the 24th International Colloquium on Automata, Languages, and Programming (ICALP '97) – July 4-5, 1997, Cesena, Italy* (February 1998), S. Gnesi and D. Latella, Eds., no. 64 in Bulletin of the Association for Theoretical Computer Science - EATCS, EATCS.

[29] BOREALE, M. Symbolic Trace Analysis of Cryptographic Protocols in the Spi-Calculus. In *Proc. of the Automata, Languages and Programming, 28th International Colloquium (ICALP 2001), July 8-12, 2001, Crete, Greece* (2001), vol. 2076 of *LNCS*, Springer-Verlag, pp. 667–681.

[30] BOREALE, M., AND GORLA, D. Processes calculi and the verification of security properties. *Journal of Telecommunications and Information Technology - Special Issue on Cryptographic Protocol Verification 4* (2002), 28–40.

[31] BOREALE, M., NICOLA, R. D., AND PUGLIESE, R. Process Algebraic Analysis Of Cryptographic Protocols. In *Proc. of the IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XIII) and Protocol Specification, Testing and Verification (PSTV XX), Oct. 10-13, 2000, Pisa, Italy* (2000), T. Bolognesi and D. Latella, Eds., Kluwer Academic Publishers, pp. 375–392.

[32] BOSNACKI, D., AND DAMS, D. Discrete-Time Promela and Spin. In *Proc. of the 5th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'98), Sept., 1998, Lyngby, Denmark* (1998), A. P. Ravn and H. Rischel, Eds., vol. 1486 of *LNCS*, Springer-Verlag, pp. 307–310.

[33] BOUALI, A., GNESI, S., AND LAROSA, S. JACK: Just another concurrency kit. the integration project. In *Bulletin of the EATCS*, vol. 54. ETACS, October 1994, pp. 207–223.

[34] BOWEN, J. P., AND HINCKEY, M. G. Seven More Myths of Formal Methods. *IEEE Software 12* (1995), 34–41.

[35] BRADFIELD, J., AND STIRLING, C. *Handbook of Process Algebra*. Elsevier, North-Holland, 2001, ch. Modal Logics and μ-Calculi: an Introduction, pp. 293–332.

[36] BROADFOOT, P. J., LOWE, G., AND ROSCOE, A. W. Automating data independence. In *Proc. of the 6th European Symposium on Research in Computer Security (ESORICS 2000), October 4-6, 2000, Toulouse, France* (2000), F. Cuppens, Y. Deswarte, D. Gollmann, and M. Waidner, Eds., vol. 1895 of *LNCS*, Springer-Verlag, pp. 175–190.

[37] BRUNS, G. *Distributed Systems Analysis with CCS*. Prentice Hall, 1997.

[38] BRUNS, G., AND SUTHERLAND, I. Model checking and fault tolerance. In *Proc. of the 6th International Conference Algebraic Methodology and Software Technology (AMAST'97), December 13-17, 1997, Sydney, Australia* (1997), M. Johnson, Ed., vol. 1349 of *LNCS*, Springer-Verlag, pp. 45–59.

[39] BÚCHI, J. R. On a decision methods in restricted second order arithmetic. In *Proc. of the International Conference on Logic, Methology and Philosophy of Science, Standford, CA, USA* (1960), Standford University Press, pp. 1–11.

[40] BURROWS, M., ABADI, M., AND NEEDHAM, R. A Logic of Authentication. *ACM Transactions on Computer Systems 8*, 1 (1990), 18–36.

[41] CACHIN, C., CAMENISCH, J., KILIAN, J., AND MÜLLER, J. One-Round Secure Computation and Secure Autonomous Mobile Agents. In *Proc. of the 27th Internation Colloquium on Automata, Languages and Programming (ICALP' 2000), Geneva, July 9-15, 2000, Geneva, Switzerland*, U. Montanari, J. D. P. Rolim, and E. Welzl, Eds., vol. 1853 of *LNCS*. Springer-Verlag, 2000, pp. 512–523.

[42] CERVESATO, I. Typed MSR: Syntax and Examples. In *Proc. of the 1st International Workshop on Mathematical Methods, Models and Architectures for Computer Networks Security (MMM'01), May 21-23, 2001, St. Petersburg, Russia* (2001), V. Gorodetski, V. Skormin, and L. Popyack, Eds., vol. 2052 of *LNCS*, Springer-Verlag, pp. 159–177.

[43] CERVESATO, I., DURGIN, N. A., LINCOLN, P. D., MITCHELL, J. C., AND SCEDROV, A. A Meta-Notation for Protocol Analysis. In *Proc. of the 12th IEEE Computer Security Foundations Workshop (CSFW'99), June 28-30, 1999, Mordano, Italy* (1999), R. Gorrieri, Ed., IEEE Computer Society Press, pp. 55–69.

[44] CERVESATO, I., DURGIN, N. A., LINCOLN, P. D., MITCHELL, J. C., AND SCEDROV, A. Relating Strands and Multiset Rewriting for Security Protocol Analysis. In *Proc. of the 13th IEEE Computer Security Foundations Workshop (CSFW'00), July 3-5, Cambrige, UK* (2000), P. Syverson, Ed., IEEE Computer Society Press, pp. 35–51.

[45] CIMATTI, A., GIUCHIGLIA, F., MONGARDI, G., ROMANO, D., TORIELLI, F., AND TRAVERSO, P. Model Checking Safety Critical Software with SPIN: an Application to a Railway Interlocking System. In *Proc. of the Computer Safety, Reliability and Security, 17th International Conference (SAFECOMP'98), October 5-7, 1998, Heidelberg, Germany* (1997), W. D. Ehrenberger, Ed., vol. 1516 of *LNCS*, Springer-Verlag, pp. 284–295.

[46] CIMATTI, A., GIUNCHIGLIA, F., MONGARDI, G., ROMANO, D., TORIELLI, F., AND TRAVERSO, P. Formal Verification of a Railway Interlocking System using Model Checking. *Formal Aspect of Computing 10*, 4 (1998), 361–380.

[47] CLARKE, E., GRUMBERG, O., JHA, S., LU, Y., AND VEITH, H. Progress on the State Explosion Problem in Model Checking. In *Informatics: 10 Years Back, 10 Years Ahead*, R. Wilhelm, Ed., vol. 2000/2001. Springer-Verlag, 2001, pp. 176–194.

[48] CLARKE, E. M., EMERSON, E. A., AND SISTLA, A. P. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specification. *ACM Transaction on Programming Languages and Systems (TOPLAS) 8*, 2 (April 1986), 244–263.

[49] CLARKE, E. M., GRUMBERG, O., AND PELED, D. A. *Model Checking*. MIT Press, January 2000.

[50] CLARKE, E. M., JHA, S., AND MARRERO, W. A Machine Checkable Logic of Knowledge for Specifying Security Properties of Electronic Protocols. In *Proc. of Workshop on Formal Methods and Security Protocols (FMSP'98) – held as part of the 13th Annual IEEE Symposium on Logic in Computer Science (LICS'98) – June 25, 1998, Indianapolis, IN, USA* (1998). (informal proceedings).

[51] CLARKE, E. M., JHA, S., AND MARRERO, W. Using state space exploration and a natural deduction style message derivation engine to verify security protocols. In *Proc. of IFIP Working Conference on Programming Concepts and Methods (PROCOMET '98), June 8-12, 1998, Shelter Island, NY, USA* (1998), D. Gries and W. P. de Roever, Eds., vol. 125 of *IFIP Conference Proceedings*, Chapman & Hall, pp. 87–106.

[52] CLARKE, E. M., JHA, S., AND MARRERO, W. Partial Order Reductions for Security Protocol Verification. In *Proc. of the International Conference for the Construction and Analysis of Systems (TACAS 2000) March 27 - April 1, 2000, Berlin, Germany* (2000), S. Graf and M. Schwartzbach, Eds., vol. 1785 of *LNCS*, Springer-Verlag, pp. 503–518.

[53] CLARKE, E. M., JHA, S., AND MARRERO, W. Verifying security protocols with Brutus. *ACM Transactions on Software Engineering and Methodology 9*, 4 (2000), 443–487.

[54] CLARKE, E. M., AND WING, J. M. Formal Methods: State of the Art and Future Directions. *ACM Computer Survey 28*, 4 (1996), 626–643.

[55] CLEAVELAND, R., PARROW, J., AND STEFFEN, B. The Concurrency Workbench: A Semantics-Based Tool for the Verification of Concurrent Systems. *ACM Transactions on Programming Languages and Systems (TOPLAS) 15*, 1 (January 1993), 36–72.

[56] COMON, H., AND SHMATIKOV, V. Is It Possible to Decide Whether a Cryptographic Protocol Is Secure Or Not? *Journal of Telecommunications and Information Technology 4* (2002), 5–15. Special Issue on Cryptographic Protocol Verification.

[57] CORIN, R., DI CAPRIO, G., ETALLE, S., GNESI, S., LENZINI, G., AND MOISO, C. Security analysis of parlay/osa framework. In *Proc. of the 9th International Conference on Intelligence in service delivery Networks (ICIN2004), Oct. 18-21 , 2004, Bordeaux, France* (2004), pp. 54–59. (also appeared as a Technical Report TR-CTIT-04-37, Aug., 2004, CTIT, Univ. of Twente, Enschede, The Netherlands).

[58] CORIN, R., DI CAPRIO, G., ETALLE, S., GNESI, S., LENZINI, G., AND MOISO, C. Security analysis of parlay/osa framework. In *Proc. of the 7th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'05) –*

held in conjuntion with the 5th International Conference on Distributed Applications and Interoperable Systems (DAIS'05) – June 15-17 , 2005, Athens, Greece (2005), vol. 3535, pp. 131–146.

[59] CORIN, R., AND ETALLE, S. An Improved Constraint-Based System for the Verification of Security Protocols. In *Proc. of the 9th International Static Analysis Symposium (SAS'02), September 17-20, 2002, Madrid, Spain* (2002), M. Hermenegildo and G. Puebla, Eds., no. 2477 in LNCS, Springer-Verlag, pp. 326–341.

[60] CORIN, R., ETALLE, S., DEN HARTOG, J., LENZINI, G., AND STAICU, I. A logic for auditing accountability in decentralized systems. In *Proc. of the 2nd international Workshop on Formal Aspects in Security and Trust (FAST2004), affiliated with 18th IFIP World Computer Congress(WCC2004) and sponsored by IFIP WG 1.7 "Theoretical Foundations of Security Analysis and Design, 16-17 Aug., 2004, Toulouse, France* (2004), Kluwer Academic Press, p. in press. (also appeared as a Technical Report TR-CTIT-04-37, June, 2004, CTIT, Univ. of Twente, Enschede, The Netherlands).

[61] CORIN, R., ETALLE, S., HARTEL, P. H., AND MADER, A. Timed Model Checking of Security Protocols. In *Proc. of the 2nd ACM workshop on Formal Methods in Security Engineering (FMSE'04) – held as part of the 11th ACM Conference on Computer and Communications Security (CCS'04) – Oct 29, 2004, Washington DC, USA* (2004), ACM Press, pp. 23–32.

[62] CRAZZOLARA, F., AND WINSKEL, G. Events in security protocols. In *Proceedings of the 8th ACM conference on Computer and Communications Security* (2001), ACM Press, pp. 96–105.

[63] DAM, M. CTL* and ECTL* as Fragments of Modal μ-Calculus. *Theoretical Computer Science 126*, 1 (1994), 77–96.

[64] DENKER, G., AND MILLEN, J. K. Capsl intermediate language. In *Proc. of the Workshop on Formal Methods and Security Protocols (FMSP99) – held as a part of the Federated Logic Conference (FLOC99) – June 5, 1999, Trento, Italy* (1999). informal proceedings.

[65] DENKER, G., AND MILLEN, J. K. CAPSL integrated protocol environment. In *Proc. of DARPA Information Survivability Conference and Exposition (DISCEX 2000), January 25-27, 2000, Hilton Head, SC, USA* (2000), IEEE Computer Society Press, pp. 207–221.

[66] DENKER, G., MILLEN, J. K., GRAU, A., AND FILIPE, J. K. Optimizing protocol rewrite rules of CIL specifications. In *Proc. of the 13th IEEE Computer Security Foundations Workshop (CSFW'00), July 3-5, 2000, Cambridge, England, UK* (2000), IEEE Computer Society Press, pp. 52–62.

[67] DIFFIE, W., AND HELLMAN, M. New Direction in Cryptography. *IEEE Transaction on Information Theory 22*, 6 (1976), 644–654.

[68] DIJKSTRA, E. W. Guarded Commands, Non-Determinacy and a Calculus for The Derivation of Programs. *ACM SIGPLAN Notices 10*, 6 (June 1975), 2–14.

[69] DOLEV, D., AND YAO, A. On the security of public-key protocols. *IEEE Transaction on Information Theory 29*, 2 (1983), 198–208.

[70] DONOVAN, B., NORRIS, P., AND LOWE, G. Analyzing Library of Security Protocols using Casper and FDR. In *Proc. of the Workshop on Formal Methods and Security Protocols (FMSP'99) – held as part of the 1999 Federated Logic Conference (FLOC'99), June 30 - July 12, 1999, Trento, Italy* (1999). (informal proceedings).

[71] DURANTE, L., SISTO, R., AND VALENZANO, A. A State-Exploration Technique for Spi-Calculus Testing-Equivalence Verification. In *Proc. of the IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XIII) and Protocol Specification, Testing and Verification (PSTV XX), Oct. 10-13, 2000, Pisa, Italy* (2000), T. Bolognesi and D. Latella, Eds., Kluwer Academic Publishers, pp. 155–170.

[72] EGIDI, L., AND PETROCCHI, M. Modelling a Secure Agent with Team Automata. *Electronic Notes In Computer Science* (2004), 119–134. (in press).

[73] EISNER, C. Using Symbolic Model Checking to Verify the Railway Stations of Hoorn-Keersenboogerd and Heerhugowaard. In *Proc. of the 10th IFIP WG10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods September 27-29, 1999, Bad Herrenalb, Germany* (1999), L. Pierre and T. Kropf, Eds., vol. 1703 of *LNCS*, Springer-Verlag, pp. 97–109.

[74] ELLIS, C. A. Team Automata for Groupware Systems. In *Proc. of the ACM SIGGROUP International Conference on Supporting Group Work (GROUP'97) The Integration Challenge, November 16-19, 1997, Embassy Suites Hotel, Phoenix, AR, USA* (1997), ACM Press, pp. 415–424.

[75] EMERSON, E. A., AND LEI, C. L. Efficient model checking in fragments of the propositional mu-calculus (extended abstract). In *Proc. of the 1st Symposium on Logic in Computer Science (LICS'86), June 16-18, 1986, Cambridge, MA, USA* (1986), IEEE Computer Society, pp. 267–278.

[76] FÁBREGA, F. J. T., HERZOG, J. C., AND GUTTMAN, J. D. Honest ideals on strand spaces. In *Proc. of the 11th Computer Security Foundations Workshop (CSFW'98), June 9-11, 1998, Rockport, MA, USA* (1998), IEEE Computer Society, pp. 66–78.

[77] FÁBREGA, F. J. T., HERZOG, J. C., AND GUTTMAN, J. D. Strand Spaces: Why is a Security Protocol Correct? In *Proc. of the 19th IEEE Computer Society Symposium on Research in Security and Privacy (SSP 98), Oakland, CA, USA, May 3-6, 1998* (1998), IEEE Computer Society, pp. 160–171.

[78] FÁBREGA, F. J. T., HERZOG, J. C., AND GUTTMAN, J. D. Strand Spaces: Proving Security Protocols Correct. *Journal of Computer Security 7*, 2–3 (1999), 191–230.

[79] FIORE, M., AND ABADI, M. Computing Symbolic Models for Verifying Cryptographic Protocols. In *Proc. of the 14th IEEE Computer Security Foundations Workshop (CSFW'01), June 11-13 2001, Cape Breton, Nova Scotia, Canada* (2001), IEEE Computer Society Press, pp. 160–173.

[80] FOCARDI, R., AND GORRIERI, R. An Information Flow Security Property for CCS. In *Proc. of the 2nd North American Process Algebra Workshop (NAPAW '93) – held as part*

of the 12th Annual ACM Symposium on Principles of Distributed Computing (PODC 93) – August 15, 1993, Ithaca, NY, USA (1993), B. Bloom, Ed., published as TR-93/1369, Cornell University, Ithaca, NY.

[81] FOCARDI, R., AND GORRIERI, R. A Classification of Security Properties. *Journal of Computer Security 3*, 1 (1995), 5–34.

[82] FOCARDI, R., AND GORRIERI, R. The Compositional Security Checker: A tool for the Verification of Information Flow Security Properties. *IEEE Transactions on Software Engineering 23*, 9 (1997), 550–571.

[83] FOCARDI, R., AND GORRIERI, R. Classification of Security Properties—Part I: Information Flow. In *Foundations of Security Analysis and Design – Tutorial Lectures*, no. 2171 in LNCS. Springer-Verlag, 2001, pp. 331–396.

[84] FOCARDI, R., GORRIERI, R., AND MARTINELLI, F. Non Interference for the Analysis of Cryptographic Protocols. In *Proc. of the 27th International Colloquium on Automata, Languages and Programming (ICALP' 2000), July 9-15, 2000, Geneva, Switzerland*, U. Montanari, J. D. P. Rolim, and E. Welzl, Eds., vol. 1853 of *LNCS*. Springer-Verlag, 2000, pp. 354–372.

[85] FOCARDI, R., GORRIERI, R., AND MARTINELLI, F. Classification of Security Properties – Part II: Network Security. In *Foundations of Security Analysis and Design II – FOSAD 2001/2002 Tutorial Lectures*, R. Gorrieri and R. Focardi, Eds., vol. 2946 of *LNCS*. Springer-Verlag, 2004, pp. 139–185.

[86] FOCARDI, R., AND MARTINELLI, F. A Uniform Approch for the Definition of Security Properties. In *Proc. of the World Congress on Formal Methods in the Development of Computing Systems (FM'99), September 20-24, 1999, Toulouse, France* (1999), J. M. Wing, J. Woodcock, and J. Davies, Eds., vol. 1708 of *LNCS*, Springer-Verlag, pp. 794–813.

[87] FOKKINK, W. J. Safey Criteria for Hoorn-Keersenboogerd Railway Station. In *Proc. of the 5th Conference on Computers in Railways (COMPRAIL'96), Volume I: Railway Systems and Management, Berlin, Germany* (1996), J. Allan, C. A. Brebbia, R. J. Hill, G. Sciutto, and S. Sone, Eds., Computational Mechanics Publications, pp. 101–110.

[88] FOLEY, S. N. External Consistency and the Verification of Security Protocols. In *Proc. of the 6th International Workshop on Security of Protocols, April 15-17, 1998, Cambridge, UK* (1999), B. Christianson, B. Crispo, W. S. Harbison, and M. Roe, Eds., vol. 1550 of *LNCS*, Springer-Verlag, pp. 24–35.

[89] FOLEY, S. N. A Non-Functional Approach to Systems Integrity. *IEEE Journal on Selected Areas in Communications 21*, 1 (2003), 36–43.

[90] FRENDRUP, U., HÜTTEL, H., AND JENSEN, J. N. Modal Logics for Cryptographic Processes. In *Proc. of the 9th International Workshop on Expressiveness in Cuncurrency (EXPRESS 02) – held as part of 13th International Conference on Concurrency Theory (CONCUR 2002) – August 19, 2002, Brno, Czech Republic* (2002). (informal proceedings).

[91] GARLAND, S., AND LYNCH, N. The IOA Language and Toolset: Support for Designing, Analyzing, and Building Distributed Systems. Tech. Rep. MIT/LCS/TR-762, MIT, 1998.

[92] GIANI, A., MARTINELLI, F., PETROCCHI, M., AND VACCARELLI, A. A Case Study with PaMoChSA: A Tool for the Automatic Analysis of Cryptographic Protocols. In *Proc. the 5th International Conference on Information Systems, Analysis and Synthesis and World Multiconference on Systemics, Cybernetics and Informatics Communications Systems and Internet (SCI 2001/ISAS 2001), July 22-25, 2001, Orlando, FL, USA* (2001), N. Callaos, I. N. da Silva, and J. Molero, Eds., International Institute of Informatics and Systemics, pp. 108–116.

[93] GIANNAKOPOULOU, D., PASAREANU, C. S., AND BARRINGER, H. Assumption Generation for Software Component Verification. In *Proc. of the 17th IEEE International Conference on Automated Software Engineering (ASE 2002) September 23-27, 2002, Edinburgh, Scotland, UK* (2002), IEEE Computer Society Press, pp. 3–12.

[94] GNESI, S., LATELLA, D., AND LENZINI, G. A BRUTUS Model Checking for a Dialect of Spi-Calculus (Extended Abstract). In *Proc. of the Workshop on Formal Methods and Computer Security (FMCS 2000) – held as part of the 12th International Conference on Computer Aided Verification (CAV 2000), July 15-19, 2000, Chicago IL, USA* (2000), Informal Proceedings.

[95] GNESI, S., LATELLA, D., AND LENZINI, G. A BRUTUS logic for a spi-calculus. In *Proc. of the ACM SIGPLAN Workshop on Issues in the Theory of Security (WITS'02) – held as part of the 29th Annual ACM SIGPLAN - SIGACT Symposium on Principles of Programming Languages (POPL'02) – January 14-15, 2002, Portland, OR, USA* (2002), J. Guttman, Ed., Informal Proceedings. (also appeared as TR B4-27 of the Istituto di Elaborazione dell'Informazione (IEI-CNR), Pisa, Italy, December, 2000).

[96] GNESI, S., LATELLA, D., LENZINI, G., AMENDOLA, A., ABBANEO, C., AND MARMO, P. An Automatic SPINCritical Railway Control System. In *Proc. of the International Conference on Dependable Systems and Networks (DSN-2000) – formerly FTCS-30 and DCCA-8 – June 25-28, 2000, New York, NY, USA* (2000), IEEE Computer Society, pp. 119–124.

[97] GNESI, S., LATELLA, D., LENZINI, G., AMENDOLA, A., ABBANEO, C., AND MARMO, P. A formal specification and validation of a control system in presence of byzantine errors. In *Proc. of the 6th International Conference Tool and Algorithms for the Construction and Analysis of Systems (TACAS 2000) – held as part of the Joint European Conferences on Theory and Practice of Software (ETAPS 2000) – March 25 - April 2, 2000, Berlin, Germany* (2000), no. 1785 in LNCS, Springer-Verlag, pp. 535–549.

[98] GNESI, S., LATELLA, D., LENZINI, G., AMENDOLA, A., AND C. ABBANEO, P. M. A formal specification and validation of a safety critical railway control system. In *Proc. of the 5th International ERCIM Workshop Formal Methods in Control Systems (FMICS 2000), April 3-4, 2000, Berlin, Germany* (April 2000), A. R. S. Gnesi, I. Schieferdecker, Ed., GND - Forschungszentrum Informationstechnik GmbH, pp. 305–329. GMD Report, REPFOKUS-2000-91.

[99] GNESI, S., LENZINI, G., AND MARTINELLI, F. Applying generalized non deducibility on compositions (gndc) approach in dependability. In *Proc. of the Formal Methods for Security and Time (MEFISTO Project), August 6, 2003, Pisa, Italy*, vol. 99 of *ENTCS*. Elsevier Science, 2004, pp. 111–126. (also appeared as a Technical Report TR-CTIT-04-36, 2004, CTIT, Univ. of Twente, Enschede, The Netherlands).

[100] GNESI, S., LENZINI, G., AND MARTINELLI, F. Logical characterization and analysis of fault tolerant systems through partial model checking. In *Proc. of the ICLP'03 International Workshop on Software Verification and Validation (SVV 2003) – held as part of the 19th International Conference on Logic Programming (ICLP 2003) – December 14, 2003, Mumbai, India*, S. Etalle, S. Mukhopadhyay, and A. Roychoudhury, Eds., vol. 118 of *ENTCS*. Elsevier Science, February 2005, pp. 57–70.

[101] GOGUEN, J. A., AND MESEGUER, J. Security Policy and Security Models. In *Proc. of the IEEE Symposium on Security and Privacy, April 26-28, 1982, Oakland, CA, USA* (1983), IEEE Computer Society Press, pp. 11–20.

[102] GORDON, A. D., AND JEFFREY, A. Authenticity by Typing for Security Protocols. In *Proc. of the 14th IEEE Computer Security Foundations Workshop (CSFW'01), June 11-13 2001, Cape Breton, Nova Scotia, Canada* (2001), IEEE Computer Society, pp. 145–159.

[103] GORRIERI, R., LANOTTE, R., MAGGIOLO-SCHETTINI, A., MARTINELLI, F., TINI, S., AND TRONCI, E. Automated analysis of timed security: A case study on web privacy. *International Journal of Information Security 2*, 2-3 (2004), 168–186.

[104] GORRIERI, R., MARTINELLI, F., PETROCCHI, M., AND VACCARELLI, A. Compositional Verification of Integrity for Digital Stream Signature Protocols. In *Proc. 3rd International Conference on Application of Concurrency to System Design (ACSD 2003), June 18-20, 2003, Guimaraes, Portugal* (2003), IEEE Computer Society Press, pp. 142–149.

[105] GROOTE, J. F., AND PONSE, A. Proof Theory for μCRL: a Language for Processing with Data. In *Proc. of the International Workshop on Semantics of Specification Languages (SoSL 1993), October 25-27, 1993, Utrecht, The Netherlands* (1994), D. J. Andrews, J. F. Groote, and C. A. Middelburg, Eds., Springer-Verlag, pp. 232–251.

[106] GROOTE, J. F., AND VAN WLIJMEN, S. F. M. A Modal Logic for μCRL. Tech. Rep. Logic Group Preprint Series 114, Utrecht, 1994.

[107] GROOTE, J. F., VAN WLIJMEN, S. F. M., AND KOORN, J. W. C. The Safety Guaranteeing System at Station Hoorn-Kersenboogerd. In *Proc. of 10th Annual Conference on Computer Assurance (COMPASS'95) Systems Integrity, Software Safety and Process Security, June 25-29, 1995, Gaithersburg, MD, USA* (1995), pp. 57–68.

[108] GROUP, P. X. W. Parlay APIs 4.0: Parlay X Web Services - White Paper. Tech. rep., The Parlay Group, 2002. http://www.parlay.org/specs/library/index.asp.

[109] GÜRGENS, S., OCHSENSCHLÄGER, P., AND RUDOLPH, C. Role Based Specification and Security Analysis of Cryptographic Protocols Using Asynchronous Product Automata. In *Proc. of the 13th International Workshop on Database and Expert Systems Applications (DEXA'02), September 2-6, 2002, Aix-en-Provence, France* (2002), IEEE Computer Society Press, pp. 473–482.

[110] GUTTMAN, J. D., AND FÁBREGA, F. J. T. Authentication tests and the structure of bundles. *Theoretical Computer Science 283*, 2 (June 2002), 333–380.

[111] HALL, J. A. Seven Myths of Formal Methods. *IEEE Software 5*, 7 (1990), 11–19.

[112] HAREL, D. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming 8*, 3 (1987), 231–274.

[113] HEATHER, J., LOWE, G., AND SCHNEIDER, S. How to Prevent Type Flaw Attacks on Security Protocols. In *Proc. of the 13th IEEE Computer Security Foundations Workshop (CSFW'00), July 3-5, 2000, Cambridge, England, UK* (2000), IEEE Computer Society Press, pp. 255–268.

[114] HEATHER, J., AND SCHNEIDER, S. Towards Automatic Verification of Authentication Protocols on an Unbounded Network. In *Proc. of the 13th IEEE Computer Security Foundations Workshop (CSFW'00), July 3-5, 2000, Cambridge, England, UK* (2000), IEEE Computer Society Press, pp. 132–143.

[115] HOARE, C. A. R. *Communicating Sequential Processes*. Prentice Hall, 1985.

[116] HOLZMANN, G. J. *Design and Validation of Computer Protocols*. Prentice Hall, 1991.

[117] HOLZMANN, G. J. The Model Checker SPIN. *IEEE Transactions on Software Engineering 23*, 5 (1997), 279–295.

[118] HOLZMANN, G. J. *The SPIN Model Checker: Primer and Reference Manual*. Addison Wesley, 2003.

[119] IEC, I. . Functional Safety of Electrical Electronic Programmable-Electronic Safety-Related Systems.

[120] JANIN, D., AND WALUKIEWICZ, I. Automata for the μ-calculus and related results. In *Proc. of 20th International Symposium on Mathematical Foundations of Computer Science (MFCS'95), August 28 - September 1, 1995, Prague, Czech Republic* (1995), J. Wiedermann and P. Hajek, Eds., vol. 969 of *LNCS*, Springer-Verlag, pp. 552–562.

[121] JONSSON, E. An Integrated Framework For Security and Dependability. In *Proc. of the New Security Paradigms Workshop (NSPW'98), September 22-25, 1998, Charlotteville, VA, USA* (1999), ACM Press, pp. 22–29.

[122] JONSSON, E., STROMBERG, L., AND LINDSKOG, S. On the Functional Relation Between Security and Dependability Impairments. In *Proc. of the New Security Paradigms Workshop (NSPW'99), September 22-24, 1999, Caledon Hills, Ontario, Canada* (2000), ACM Press, pp. 104–111.

[123] KLEIJN, J. Team Automata for CSCW – A Survey. In *Petri Net Technology for Communication-Based Systems: Advances in Petri Nets* (2003), H. Ehrig, W. Reisig, G. Rozenberg, and H. Weber, Eds., no. 2472 in LNCS, Springer-Verlag, pp. 295–320.

[124] KLINT, P. A meta-environment for generating programming environments. *ACM Transactions on Software Engineering and Methodology (TOSEM) 2*, 2 (1993), 176–201.

[125] KOZEN, D. Results on the propositional $\mu-$calculus. *Theoretical Computer Science 27*, 3 (1983), 333–354.

[126] KUPFERMAN, O., AND VARDI, M. Y. Module checking. In *Proc. of the 8th International Conference on Computer Aided Verification (CAV'96), July 31-August 3, 1996, New Brunswick, NJ, USA* (1996), R. Alur and T. A. Henzinger, Eds., vol. 1102 of *LNCS*, Springer-Verlag, pp. 75–86.

[127] LAMPORT, L. Proving the Correctness of Multiprocess Programs. *IEEE Transactions of Software Engineering 21*, 7 (1977), 125–143.

[128] LAMPORT, L., SHOSTAK, R., AND PEASE, M. The Byzantine Generals Problem. *ACM Transaction on Programming Languages and Systems 4*, 3 (1982), 382–401.

[129] LANOTTE, R., MAGGIOLO-SCHETTINI, A., AND TROINA, A. Weak Bisimulation for Probabilistic Timed Automata and Applications to Security. In *Proc. of the 1st International Conference on Software Engineering and Formal Methods (SEFM '03), September 22-27, 2003, Brisbane, Australia* (2003), IEEE Computer Society Press, pp. 34–43.

[130] LANOTTE, R., MAGGIOLO-SCHETTINI, A., AND TROINA, A. Weak bisimulation for probabilistic timed automata and applications to security. In *Proc. of 1st International Conference on Software Engineering and Formal Methods (SEFM'03), Sept. 22-27, 2003, Brisbane, Australia* (2003), IEEE Computer Society Press, pp. 34–43.

[131] LANOTTE, R., MAGGIOLO-SCHETTINI, A., AND TROINA, A. Information flow analysis for probabilistic timed automata. In *Proc. of the 2nd Workshop in Security and Trust (FAST'04), Aug. 26-27, 2004, Toulouse, France* (2004), Kluwer Academic Press. (to appear).

[132] LARSEN, P. G., FITZGERALD, J., AND BROOKERS, T. Applying Formal Specification in Industry. *IEEE Software 13*, 7 (1996), 48–56.

[133] LEDUC, G. Verification of two versions of the Challenge Handshake Authentication Protocol (CHAP). *Annals of Telecommunications 55*, 1-2 (2000), 18–30.

[134] LENZINI, G., GNESI, S., AND LATELLA, D. Spider: a security model checker. In *Proc. of the 1st International Workshop on Formal Aspect of Security and Trust (FAST 2003) – held as a part of the 12th International Formal Methods Europe Symposium (FM 2003) – Sept. 8-14, 2003, Pisa, Italy* (2003), F. Martinelli and T. Dimitrakos, Eds., Istituto di Informatica e Telematica (IIT-CNR), Pisa, Italy, pp. 163–180. Technical Report ITT-CNR-10/2003.

[135] LIGGERSMEYER, P., ROTHFELDER, M., RETTELBACH, M., AND ACKERMANN, T. Qualitätssincherung Software-basierter Technischer Systeme - Problembereiche und Lösungsänsatze. *Informatik Spektrum 21* (1998), 249–258. in German.

[136] LOWE, G. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proc. of 19th IEEE Computer Security Foundations Workshop (CSFW'96)*, vol. 1055 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996, pp. 147–166.

[137] LOWE, G. Some New Attacks upon Security Protocols. In *Proc. of 19th IEEE Computer Security Foundations Workshop (CSFW'96)*, vol. 1055 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996, pp. 147–166.

[138] LOWE, G. Breaking and Fixing the Needham-Schroeder Public-Key Protocol using FDR. *Software Concepts and Tools 3*, 17 (1997), 93–102.

[139] LOWE, G. Casper: A Compiler for the Analysis of Security Protocols. In *Proc. of 10th IEEE Computer Security Foundations Workshop (CSFW'97), June 10-12, 1997, Rockport, MA, USA* (1997), IEEE Computer Society Press, pp. 18–30.

[140] LYNCH, N. I/O Automaton Models and Proofs for Shared-Key Communication Systems. In *Proc. of the 12th IEEE Computer Security Foundations Workshop (CSFW'99), June 28-30, 1999, Mordano, Italy* (1999), R. Gorrieri, Ed., pp. 14–29.

[141] LYNCH, N. Input/output automata: Basic, timed, hybrid, probabilistic, dynamic, ... In *Proc. of the 14th International Conference on Concurrency Theory (CONCUR 2003), September 3-5, 2003, Marseille, France*, R. M. Amadio and D. Lugiez, Eds., vol. 2761 of *LNCS*. Springer-Verlag, 2003, pp. 187–188.

[142] LYNCH, N., AND TUTTLE, M. An Introduction to Input/Output Automata. *CWI Quarterly 2*, 3 (1989), 219–246.

[143] MANNA, Z., AND PNUELI, A. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.

[144] MANOLIOS, P., AND TREFLER, R. J. Safety and Liveness in Branching Time. In *Proc. of 16th Annual IEEE Symposium on Logic in Computer Science (LICS), June 16-19, 2001, Boston, MA, USA* (2001), IEEE Computer Society, pp. 366–375.

[145] MARTINELLI, F. *Formal Methods for the Analysis of Open Systems with Applications to Security Properties*. PhD thesis, University of Siena, December 1998.

[146] MARTINELLI, F. Partial Model Checking and Theorem Proving for Ensuring Security Properties. In *Proc. of the 11th IEEE Computer Security Foundation Workshop (CSFW'98), June 9-11 , 1998, Rockport, MA, USA* (1998), IEEE Computer Society Press, pp. 44–52.

[147] MARTINELLI, F. Encoding Several Security Properties as Properties of the Intruder's Knowledge. Tech. Rep. IAT-B4-2001-20, Istituto di Automatica e Telematica (IAT-CNR), Pisa, Italy, December 2001.

[148] MARTINELLI, F. Analysis of Security Protocols as Open Systems. *Theoretical Computer Science 290*, 1 (2003), 1057–1106.

[149] MARTINELLI, F., PETROCCHI, M., AND VACCARELLI, A. Compositional Verification of Secure Streamed Data: a Case Study with EMSS. In *Proc. of the 8th Italian Conference on Theoretical Computer Science (ICTCS'03), October 13-15, 2003, Bertinoro, Italy* (2003), no. 2841 in LNCS, pp. 383–396.

[150] MATSUI, M. On Correction Between the Order of S-boxes and the Strength of DES. In *Lecture Notes in Computer Science*, vol. 950. Springer-Verlag, 1994, pp. 367–375. Proc. of EUROCRYP'94.

[151] MEADOWS, C., AND MCLEAN, J. Security and Dependability: Then and Now. In *Proc. of the Workshop II on Computer Security, Fault Tolerance, and Software Assurance: From*

Needs to Solutions, Williamsburg, VA, USA, November 11-13, 1998 (1999), IEEE Computer Society Press, pp. 166–70.

[152] MEADOWS, C. A. Formal Verification of Cryptographic Protocols: A Survey. In *Proc. of the International Conference on the Theory and Application of Cryptology Advances in Cryptology and Information Security, (ASIACRYPT 94)* (1994), J. Pieprzyk and R. Safavi-Naini, Eds., vol. 917 of *LNCS*, Springer-Verlag, pp. 135–150.

[153] MEADOWS, C. A. Applying the Dependability Paradigm to Computer Security. In *Proc. of the 1995 Workshop on New Security Paradigms, August 22 - 25, 1995, La Jolla, CA, USA* (1995), IEEE Computer Society Press, pp. 75–79.

[154] MEADOWS, C. A. Analyzing the Needham-Schroeder Public Key Protocol: A Comparison of Two Approaches. In *Proc. of the 4th European Symposium on Research in Computer Security (ESORICS'96), September 1996, Rome, Italy*, E. Bertino, H. Kurth, G. Martella, and E. Montolivo, Eds., vol. 1146 of *LNCS*. Springer-Verlag, 1996, pp. 351–364.

[155] MEADOWS, C. A. The NRL protocol analyzer: an overview. *Journal of Logic Programming 26*, 2 (1996), 113–131.

[156] MILLEN, J. K., AND SHMATIKOV, V. Constraint Solving for Bounded-Process Cryptographic Protocol Analysis. In *Proc. of the 8th ACM Conference on Computer and Communication Security, November 5-8, 2001, Philadelphia, PA, USA* (2001), P. Samarati, Ed., ACM Press, pp. 166–175.

[157] MILLER, D. Higher-Order Quantification and Proof Search. In *Proc. of the 9th International Conference on Algebraic Methodology and Software Technology (AMAST), September 9–13, 2002, Reunion Island, France* (2002), H. Kirchner and C. Ringeissen, Eds., vol. 2422 of *LNCS*, Springer-Verlag.

[158] MILNER, R. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

[159] MILNER, R. Operational and algebraic semantics of concurrent processes. In *Handbook of Theoretical Computer Science*, J. van Leewen, Ed., vol. B: Formal Models and Semantics. The MIT Press, 1990, ch. 19, pp. 1201–1242.

[160] MILNER, R. *Communication and Mobile systems: the π-calculus*. Cambridge University Press, 1999.

[161] MILNER, R., PARROW, J., AND WALKER, D. A calculus of mobile processes. *Information and Computation 100*, 1 (1992), 1–77.

[162] MILNER, R., PARROW, J., AND WALKER, D. A Calculus of Mobile Processes, I and II. *Information and Computation 100*, 1 (1992), 1–40.

[163] MITCHELL, J. C., MITCHELL, M., AND STERN, U. Automated Analysis of Cryptographic Protocols Using Murϕ. In *Proc. of the IEEE Symposium on Security and Privacy, May 4-7, 1998, Oakland, California, USA* (1997), IEEE Computer Society Press, pp. 141–151.

[164] MONGARDI, G. *Dependable Computing for Railway Control System*. Springer-Verlag, 1993, ch. 3.

[165] MORLEY, M. J. Safety-level communication in railway interlockings. *Science of Computer Programming 29*, 1-2 (1997), 147–170.

[166] NAUMOVICH, G., AND CLARKE, L. A. Classifying Properties: an Alternative to The Safety-Liveness Classification. In *Proc. of the 8th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE-8), Nov 6-10, 2000, San Diego, California, USA* (2000), ACM Press, pp. 159–168.

[167] NEEDHAM, R. M., AND SCHROEDER, M. D. Using Encryption for Authentication in Large Network of Computer. *Communications of the ACM 21*, 12 (1978), 993–999.

[168] NIPKOW, T., PAULSON, L., AND WENZEL, M. *Isabelle/HOL—A Proof Assistant for Higher-Order Logic*. No. 2283 in LNCS. Springer-Verlag, 2002.

[169] OHEIMB, D. Interacting State Machines: A Stateful Approach to Proving Security. In *Proc. of 1st International Conference Formal Aspects of Security (FASec 2002), December 16-18, London, UK* (2003), no. 2629 in LNCS, Springer-Verlag, pp. 15–32.

[170] OHEIMB, D., AND LOTZ, V. Formal Security Analysis with Interacting State Machines. In *Proc. of 7th European Symposium on Research in Computer Security (ESORICS'02), October, 14-16, 2002, Zurich, Switzerland* (2002), M. W. D. Gollmann, G. Karjoth, Ed., no. 2502 in LNCS, Springer-Verlag, pp. 212–228.

[171] PAGE, U. R. Unified Modeling Language. http://www.uml.org.

[172] PAULSON, L. C. Proving Properties of Security Protocols by Induction. In *Proc. of 10th IEEE Computer Security Foundations Workshop (CSFW'97), June 10-12, 1997, Rockport, MA, USA* (1997), IEEE Computer Society Press, pp. 70–83.

[173] PERRIG, A., CANETTI, R., TYGAR, J., AND SONG, D. Efficient Authentication and Signing of Multicast Streams over Lossy Channels. In *Proc. of the 21th IEEE Symposium on Security and Privacy* (2000), IEEE Computer Society Press, pp. 56–73.

[174] PNUELI, A. The Temporal Logic of Programs. In *Proc. of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77), Providence, Rhode, Island* (1977), IEEE Computer Society Press, pp. 46–57.

[175] PR EN 50128 CENELEC. Railways Applications: Software for Railway Control and Protection Systems.

[176] ROSCOE, A. W. *Model-Checking CSP*. Prentice Hall International (UK) Ltd., 1994.

[177] ROSCOE, A. W. Modelling and Verifying Key-Exchange Protocols Using CSP and FDR. In *Proc. of The 8th Computer Security Foundations Workshop (CSFW '95), Kenmare, Ireland, March 13-15, 1995* (1995), IEEE Computer Society Press, pp. 98–107.

[178] ROSCOE, A. W. Proving Security Protocols with Model Checkers by Data Independence Techniques. In *Proc. of The 11th Computer Security Foundations Workshop (CSFW'98), June 9-11, 1998, Rockport, MA, USA* (1998), IEEE Computer Society Press, pp. 84–95.

[179] RUMBAUGH, J., JACOBSON, I., AND BOOCH, G. *Unified Modeling Language Reference Manual*. Addison Wesley, 1998.

[180] RUSHBY, J. Partitioning for Safety and Security: Requirements, Mechanisms, and Assurance. Tech. rep., NASA Langley Research Center, June 1999.

[181] SCHNEIDER, S. Security Properties and CSP. In *Proc. of the IEEE Symposium on Security and Privacy, May 6-8, 1996, Oakland, CA* (1996), pp. 174–187.

[182] SCHNEIDER, S. Verifying Authentication Protocols in CSP. *IEEE Transaction on Sofware Engineering 24*, 8 (1998), 743–758.

[183] SEGALA, R. A Compositional Trace-Based Semantics for Probabilistic Automata. In *Proc. of the 6th International Conference on Concurrency Theory (CONCUR '95), August 21-24, 1995, Philadelphia, PA, USA*, I. Lee and S. A. Smolka, Eds., vol. 962 of *LNCS*. Springer-Verlag, 1995, pp. 234–248.

[184] SHEERAN, M., AND STÅLMARCK, G. A tutorial on Stålmarck's proof procedure for propositional logic. In *Proc. of the 2nd International Conference on Formal Methods in Computer-Aided Design (FMCAD)'98, November 4-6, 1998, Palo Alto, CA, USA*, G. Gopalakrishnan and P. Windley, Eds., vol. 1522 of *LNCS*. Springer-Verlag, 1998, pp. 82–99.

[185] SIMPSON, A., WOODCOCK, J., AND DAVIS, J. Safety through security. In *Proc. of the 9th International Workshop on Software Specification and Design, April 16-18, 1998, Ise-Shima (Isobe), Japan* (1998), IEEE Computer Society, pp. 18–23.

[186] SONG, D. X. Athena: A New Efficient Automatic Checker for Security Protocol Analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop (CSFW '99), June 28-30, Mordano, Italy* (1999), IEEE Computer Society Press, pp. 192–202.

[187] STAVRIDOU, V., AND DUTERTRE, B. From security to safety and back. In *Proc. of the Workshop II on Computer Security, Fault Tolerance, and Software Assurance: From Needs to Solutions, November 11-13, 1998, Williamsburg, VA* (1998), IEEE Computer Society Press.

[188] STIRLING, C. An Introduction to Modal and Temporal Logics for CCS. In *Proc. of the International Workshop on Concurrency: Theory, Language, and Architecture, September 25-27, 1989, Oxford, UK* (1989), A. Yonezawa and T. Ito, Eds., vol. 491 of *LNCS*, Springer-Verlag, pp. 2–20.

[189] STIRLING, C. *Modal and Temporal Logics for Processes*, vol. 1043 of *LNCS*. Springer-Verlag, 1996, pp. 149–240.

[190] STIRLING, C. *Modal and Temporal Properties of Processes*. Texts in Computer Science. Springer - Verlag, 2001.

[191] STOREY, N. *Safety Critical Computer Systems*. Addison-Wesley, 1996.

[192] STREETT, R. S., AND EMERSON, E. A. An Automata Theoretic Procedure for the Propositional $\mu-$calculus. *Information and Computation 81*, 3 (1989), 249–264.

[193] T. A. HENZINGER, O. KUPFERMAN, R. M. On the universal and existential fragments of the μ-calculus. In *Proc. of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003), as part of Joint European Conferences on Theory and Practice of Software (ETAPS 2003), April 7-11, 2003, Warsaw, Poland* (2003), H. Garavel and J. Hatcliff, Eds., no. 619 in LNCS, pp. 49–64.

[194] TER BEEK, M. H. *Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components*. PhD thesis, Leiden Universiteit (The Netherlands), December 2003.

[195] TER BEEK, M. H., ELLIS, C. A., KLEIJN, J., AND ROZENBERG, G. Team Automata for Spatial Access Control. In *Proc. of the Seventh European Conference on Computer-Supported Cooperative Work (ECSCW 2001), Sept. 16-20, 2001, Bonn, Germany* (2001), Kluwer Academic Publishers, pp. 59–77.

[196] TER BEEK, M. H., AND KLEIJN, J. Team Automata Satisfying Compositionality. In *Proc. of the 12th International on Formal Methods Europe Symposium (FME'03), Sep. 8-14, 2003, Pisa, Italy* (2003), K. Araki, S. Gnesi, and D. Mandrioli, Eds., no. 2805 in LNCS, Springer-Verlag, pp. 381–400.

[197] TER BEEK, M. H., LENZINI, G., AND PETROCCHI, M. Team automata for security analysis of multicast/broadcast communication. In *Presented at the Workshop on Issues in Security and Petri Nets (WISP) – affiliated to the 24th International Conference on Application and Theory of Petri Nets (ATPN'03), June 23, 2003, Eindhoven, The Netherland* (2003), R. G. N. Busi and F. Martinelli, Eds., Informal Proceedings, pp. 57–72. (also appeared as Technical Report 003-TR-13, 2003, ISTI-CNR, Pisa, Italy).

[198] TER BEEK, M. H., LENZINI, G., AND PETROCCHI, M. Team automata for security analysis. Tech. rep., Centre for Telematics and Information Technology (CTIT), Univ. Twente, Enschede, The Netherlands, 2004. Technical Report TR-CTIT 04-13.

[199] TER BEEK, M. H., LENZINI, G., AND PETROCCHI, M. Team Automata for Security (a Survey). In *Proc. of the 2nd International Conference on Security Issues in Coordination Models, Languages, and Systems (SecCo 2004), affiliated to CONCUR 2004, 30 August 2004, London, United Kingdom*, R. Focardi and G. Zavattaro, Eds., vol. 128:5 of *ENTCS*. Elsevier Science, May 2005, pp. 105–119. (also published as a technical report IIT TR-06/2004, Istituto di Informatica e Telematica CNR).

[200] THOMPSON, H. H., WHITTAKER, J. A., AND MOTTAY, F. E. Software security vulnerability testing in hostile environments. In *Proceedings of the 2002 ACM symposium on Applied computing (SAC 02)* (Madrid, Spain, 2002), A. Press, Ed., pp. 260–264.

[201] VALMARI, A. The State Explosion Problem. In *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets*, vol. 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998, pp. 429–528.

[202] WALUKIEWICZ, I. On Completeness of the μ-Calculus. In *Proceedings 8th Annual IEEE Symp. on Logic in Computer Science (LICS'93), June 20-23, 1993, Montreal, Canada* (1993), IEEE Computer Society Press, pp. 136–146.

[203] WALUKIEWICZ, I. Completeness of Kozen's axiomatization of the propositional λ-calculus. In *Symposium on Logic in Computer Science (LICS '95)* (Los Alamitos, Ca., USA, 1995), IEEE Computer Society Press, pp. 14–24.

[204] WEBER, D. G. Formal specification of fault tolerance and its relation to computer security. *ACM SIGSOFT Engineering Notes 14*, 3 (1989). Proc. of International Workshop on Software Specification and Design.

[205] WING, J. M. A Specifier's Introduction to Formal Methods. *IEEE Computer 23*, 9 (September 1990), 8–24.

[206] WOO, T., AND LAM, S. A Semantic Model for Authentication Protocols. In *Proc. of the IEEE Symposium on Security and Privacy, May 24-26, 1993, Oakland, CA* (1993), IEEE Computer Society Press, pp. 178–194.

Titles in the IPA Dissertation Series

J.O. Blanco. *The State Operator in Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1996-01

A.M. Geerling. *Transformational Development of Data-Parallel Algorithms*. Faculty of Mathematics and Computer Science, KUN. 1996-02

P.M. Achten. *Interactive Functional Programs: Models, Methods, and Implementation*. Faculty of Mathematics and Computer Science, KUN. 1996-03

M.G.A. Verhoeven. *Parallel Local Search*. Faculty of Mathematics and Computing Science, TUE. 1996-04

M.H.G.K. Kesseler. *The Implementation of Functional Languages on Parallel Machines with Distrib. Memory*. Faculty of Mathematics and Computer Science, KUN. 1996-05

D. Alstein. *Distributed Algorithms for Hard Real-Time Systems*. Faculty of Mathematics and Computing Science, TUE. 1996-06

J.H. Hoepman. *Communication, Synchronization, and Fault-Tolerance*. Faculty of Mathematics and Computer Science, UvA. 1996-07

H. Doornbos. *Reductivity Arguments and Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1996-08

D. Turi. *Functorial Operational Semantics and its Denotational Dual*. Faculty of Mathematics and Computer Science, VUA. 1996-09

A.M.G. Peeters. *Single-Rail Handshake Circuits*. Faculty of Mathematics and Computing Science, TUE. 1996-10

N.W.A. Arends. *A Systems Engineering Specification Formalism*. Faculty of Mechanical Engineering, TUE. 1996-11

P. Severi de Santiago. *Normalisation in Lambda Calculus and its Relation to Type Inference*. Faculty of Mathematics and Computing Science, TUE. 1996-12

D.R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. Faculty of Mathematics and Computing Science, TUE. 1996-13

M.M. Bonsangue. *Topological Dualities in Semantics*. Faculty of Mathematics and Computer Science, VUA. 1996-14

B.L.E. de Fluiter. *Algorithms for Graphs of Small Treewidth*. Faculty of Mathematics and Computer Science, UU. 1997-01

W.T.M. Kars. *Process-algebraic Transformations in Context*. Faculty of Computer Science, UT. 1997-02

P.F. Hoogendijk. *A Generic Theory of Data Types*. Faculty of Mathematics and Computing Science, TUE. 1997-03

T.D.L. Laan. *The Evolution of Type Theory in Logic and Mathematics*. Faculty of Mathematics and Computing Science, TUE. 1997-04

C.J. Bloo. *Preservation of Termination for Explicit Substitution*. Faculty of Mathematics and Computing Science, TUE. 1997-05

J.J. Vereijken. *Discrete-Time Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1997-06

F.A.M. van den Beuken. *A Functional Approach to Syntax and Typing*. Faculty of Mathematics and Informatics, KUN. 1997-07

A.W. Heerink. *Ins and Outs in Refusal Testing*. Faculty of Computer Science, UT. 1998-01

G. Naumoski and W. Alberts. *A Discrete-Event Simulator for Systems Engineering*. Faculty of Mechanical Engineering, TUE. 1998-02

J. Verriet. *Scheduling with Communication for Multiprocessor Computation*. Faculty of Mathematics and Computer Science, UU. 1998-03

J.S.H. van Gageldonk. *An Asynchronous Low-Power 80C51 Microcontroller*. Faculty of Mathematics and Computing Science, TUE. 1998-04

A.A. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1998-05

E. Voermans. *Inductive Datatypes with Laws and Subtyping – A Relational Model*. Faculty of Mathematics and Computing Science, TUE. 1999-01

H. ter Doest. *Towards Probabilistic Unification-based Parsing*. Faculty of Computer Science, UT. 1999-02

J.P.L. Segers. *Algorithms for the Simulation of Surface Processes*. Faculty of Mathematics and Computing Science, TUE. 1999-03

C.H.M. van Kemenade. *Recombinative Evolutionary Search*. Faculty of Mathematics and Natural Sciences, UL. 1999-04

E.I. Barakova. *Learning Reliability: a Study on Indecisiveness in Sample Selection*. Faculty of Mathematics and Natural Sciences, RUG. 1999-05

M.P. Bodlaender. *Scheduler Optimization in Real-Time Distributed Databases*. Faculty of Mathematics and Computing Science, TUE. 1999-06

M.A. Reniers. *Message Sequence Chart: Syntax and Semantics*. Faculty of Mathematics and Computing Science, TUE. 1999-07

J.P. Warners. *Nonlinear approaches to satisfiability problems*. Faculty of Mathematics and Computing Science, TUE. 1999-08

J.M.T. Romijn. *Analysing Industrial Protocols with Formal Methods*. Faculty of Computer Science, UT. 1999-09

P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. Faculty of Computer Science, UT. 1999-10

G. Fábián. *A Language and Simulator for Hybrid Systems*. Faculty of Mechanical Engineering, TUE. 1999-11

J. Zwanenburg. *Object-Oriented Concepts and Proof Rules*. Faculty of Mathematics and Computing Science, TUE. 1999-12

R.S. Venema. *Aspects of an Integrated Neural Prediction System*. Faculty of Mathematics and Natural Sciences, RUG. 1999-13

J. Saraiva. *A Purely Functional Implementation of Attribute Grammars*. Faculty of Mathematics and Computer Science, UU. 1999-14

R. Schiefer. *Viper, A Visualisation Tool for Parallel Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1999-15

K.M.M. de Leeuw. *Cryptology and Statecraft in the Dutch Republic*. Faculty of Mathematics and Computer Science, UvA. 2000-01

T.E.J. Vos. *UNITY in Diversity. A stratified approach to the verification of distributed algorithms*. Faculty of Mathematics and Computer Science, UU. 2000-02

W. Mallon. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes*. Faculty of Mathematics and Natural Sciences, RUG. 2000-03

W.O.D. Griffioen. *Studies in Computer Aided Verification of Protocols*. Faculty of Science, KUN. 2000-04

P.H.F.M. Verhoeven. *The Design of the MathSpad Editor*. Faculty of Mathematics and Computing Science, TUE. 2000-05

J. Fey. *Design of a Fruit Juice Blending and Packaging Plant*. Faculty of Mechanical Engineering, TUE. 2000-06

M. Franssen. *Cocktail: A Tool for Deriving Correct Programs*. Faculty of Mathematics and Computing Science, TUE. 2000-07

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

E. Saaman. *Another Formal Specification Language*. Faculty of Mathematics and Natural Sciences, RUG. 2000-10

M. Jelasity. *The Shape of Evolutionary Search Discovering and Representing Search Space Structure*. Faculty of Mathematics and Natural Sciences, UL. 2001-01

R. Ahn. *Agents, Objects and Events a computational approach to knowledge, observation and communication*. Faculty of Mathematics and Computing Science, TU/e. 2001-02

M. Huisman. *Reasoning about Java programs in higher order logic using PVS and Isabelle*. Faculty of Science, KUN. 2001-03

I.M.M.J. Reymen. *Improving Design Processes through Structured Reflection*. Faculty of Mathematics and Computing Science, TU/e. 2001-04

S.C.C. Blom. *Term Graph Rewriting: syntax and semantics*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2001-05

R. van Liere. *Studies in Interactive Visualization*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2001-06

A.G. Engels. *Languages for Analysis and Testing of Event Sequences*. Faculty of Mathematics and Computing Science, TU/e. 2001-07

J. Hage. *Structural Aspects of Switching Classes*. Faculty of Mathematics and Natural Sciences, UL. 2001-08

M.H. Lamers. *Neural Networks for Analysis of Data in Environmental Epidemiology: A Case-study into Acute Effects of Air Pollution Episodes*. Faculty of Mathematics and Natural Sciences, UL. 2001-09

T.C. Ruys. *Towards Effective Model Checking*. Faculty of Computer Science, UT. 2001-10

D. Chkliaev. *Mechanical verification of concurrency control and recovery protocols*. Faculty of Mathematics and Computing Science, TU/e. 2001-11

M.D. Oostdijk. *Generation and presentation of formal mathematical documents*. Faculty of Mathematics and Computing Science, TU/e. 2001-12

A.T. Hofkamp. *Reactive machine control: A simulation approach using χ*. Faculty of Mechanical Engineering, TU/e. 2001-13

D. Bošnački. *Enhancing state space reduction techniques for model checking*. Faculty of Mathematics and Computing Science, TU/e. 2001-14

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects*. Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity*. Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing*. Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing*. Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining*. Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL*. Faculty of Mathematics and Computer Science, TU/e. 2002-16

J.J.D. Aerts. *Random Redundant Storage for Video on Demand*. Faculty of Mathematics and Computer Science, TU/e. 2003-01

M. de Jonge. *To Reuse or To Be Reused: Techniques for component composition and construction*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-02

J.M.W. Visser. *Generic Traversal over Typed Source Code Representations*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-03

S.M. Bohte. *Spiking Neural Networks*. Faculty of Mathematics and Natural Sciences, UL. 2003-04

T.A.C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing*. Faculty of Mathematics and Computer Science, TU/e. 2003-05

S.V. Nedea. *Analysis and Simulations of Catalytic Reactions*. Faculty of Mathematics and Computer Science, TU/e. 2003-06

M.E.M. Lijding. *Real-time Scheduling of Tertiary Storage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-07

H.P. Benz. *Casual Multimedia Process Annotation – CoMPAs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08

D. Distefano. *On Modelchecking the Dynamics of Object-based Software: a Foundational Approach*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-09

M.H. ter Beek. *Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components*. Faculty of Mathematics and Natural Sciences, UL. 2003-10

D.J.P. Leijen. *The λ Abroad – A Functional Approach to Software Components*. Faculty of Mathematics and Computer Science, UU. 2003-11

W.P.A.J. Michiels. *Performance Ratios for the Differencing Method*. Faculty of Mathematics and Computer Science, TU/e. 2004-01

G.I. Jojgov. *Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving*. Faculty of Mathematics and Computer Science, TU/e. 2004-02

P. Frisco. *Theory of Molecular Computing – Splicing and Membrane systems*. Faculty of Mathematics and Natural Sciences, UL. 2004-03

S. Maneth. *Models of Tree Translation*. Faculty of Mathematics and Natural Sciences, UL. 2004-04

Y. Qian. *Data Synchronization and Browsing for Home Environments*. Faculty of Mathematics and Computer Science and Faculty of Industrial Design, TU/e. 2004-05

F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-06

L. Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications*. Faculty of Science, Mathematics and Computer Science, KUN. 2004-07

E.H. Gerding. *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications*. Faculty of Technology Management, TU/e. 2004-08

N. Goga. *Control and Selection Techniques for the Automated Testing of Reactive Systems*. Faculty of Mathematics and Computer Science, TU/e. 2004-09

M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs*. Faculty of Science, Mathematics and Computer Science, RU. 2004-10

A. Löh. *Exploring Generic Haskell*. Faculty of Mathematics and Computer Science, UU. 2004-11

I.C.M. Flinsenberg. *Route Planning Algorithms for Car Navigation*. Faculty of Mathematics and Computer Science, TU/e. 2004-12

R.J. Bril. *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets*. Faculty of Mathematics and Computer Science, TU/e. 2004-13

J. Pang. *Formal Verification of Distributed Systems*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-14

F. Alkemade. *Evolutionary Agent-Based Economics*. Faculty of Technology Management, TU/e. 2004-15

E.O. Dijk. *Indoor Ultrasonic Position Estimation Using a Single Base Station*. Faculty of Mathematics and Computer Science, TU/e. 2004-16

S.M. Orzan. *On Distributed Verification and Verified Distribution*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-17

M.M. Schrage. *Proxima - A Presentation-oriented Editor for Structured Documents*. Faculty of Mathematics and Computer Science, UU. 2004-18

E. Eskenazi and A. Fyukov. *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures*. Faculty of Mathematics and Computer Science, TU/e. 2004-19

P.J.L. Cuijpers. *Hybrid Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2004-20

N.J.M. van den Nieuwelaar. *Supervisory Machine Control by Predictive-Reactive Scheduling*. Faculty of Mechanical Engineering, TU/e. 2004-21

E. Ábrahám. *An Assertional Proof System for Multithreaded Java -Theory and Tool Support- *. Faculty of Mathematics and Natural Sciences, UL. 2005-01

R. Ruimerman. *Modeling and Remodeling in Bone Tissue*. Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. *Experiments in Rights Control - Expression and Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

H. Gao. *Design and Verification of Lock-free Parallel Algorithms*. Faculty of Mathematics and Computing Sciences, RUG. 2005-04

H.M.A. van Beek. *Specification and Analysis of Internet Applications*. Faculty of Mathematics and Computer Science, TU/e. 2005-05

M.T. Ionita. *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures*. Faculty of Mathematics and Computing Sciences, TU/e. 2005-06

G. Lenzini. *Integration of Analysis Techniques in Security and Fault-Tolerance*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-07