Formal Framework for Security Policy Enforcement in Computer Systems

by

Liviu Pene

Thesis submitted in partial fulfillment
of the requirements for the degree of
Philosophiæ Doctor in Computer Science

Département d'informatique et d'ingénierie
UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS
Gatineau, Québec

JANUARY 2018

Cadre formel pour le renforcement des politiques de sécurité dans les systèmes informatiques

par

Liviu Pene

Thèse présenté comme exigence partielle
pour l'obtention du grade de
Philosophiæ Doctor en Informatique

Département d'informatique et d'ingénierie
UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS
Gatineau, Québec

JANVIER 2018

Examination Committee

Dr. Jurek Czyzowicz	*Examination Committee Chair*
Dr. Kamel Adi	*Director of Research*
Dr. Luigi Logrippo	*Internal Examiner*
Dr. Mohamed Mejri	*External Examiner*
Dr. Marc Frappier	*External Examiner*

To my beloved wife Raluca and my wonderful son George.

Abstract

The swift evolution of networks and computer systems has generated substantial improvements and marked benefits in several aspects of our personal and professional lives. However, these advantages come at the expense of an increased complexity of the security mechanisms protecting them. Defending the systems has become a big challenge both for individuals and for enterprises. The age of cloud computing and the Internet of Things significantly aggravated the problem. The main reason is that these technologies call into question the classic centralized security models and require contemplating completely distributed approaches. Given this context, computer systems security can no longer be assured solely by the application of best practices, and a formal and rigorous approach is henceforth necessary.

This thesis tackles the question of automatic protection of computer systems by exploring the use of formal methods for policy specification, verification and enforcement. In order to build our formal framework, we have defined algebraic formalisms and modal logics that allow specifying information systems and their behaviour in an elegant and concise manner. We have also defined formal verification techniques for assessing the compliance of the systems with the security policies. Finally, we have devised an enforcement operator capable of generating an automatic enforcement process. The latter has the ability to rewrite the algebraic specification of a computer system in a way that renders it compliant with a security policy.

Résumé

Le développement accéléré des systèmes et réseaux informatiques a engendré des améliorations et avantages substantielles dans plusieurs aspects de notre vie quotidienne et professionnelle. Cependant, ces avantages viennent souvent au détriment d'une complexité accrue dans les mécanismes de sécurité de ces systèmes dont la protection représente un grand défi aussi bien pour les individus que pour les organisations. À l'ère de l'infonuagique et de l'internet des objets, le problème s'est grandement amplifié. La raison principale est que ces technologies remettent en cause les modèles de sécurité centralisés classiques et exigent de considérer des approches totalement distribuées. Dans ce contexte, la sécurité informatique ne peut plus être assurée par l'unique application de règles de bonnes pratiques et une approche formelle et rigoureuse est désormais nécessaire.

Cette thèse traite de la question de la protection automatique des systèmes informatiques en explorant l'utilisation des méthodes formelles pour la spécification, la vérification et le renforcement automatique des politiques de sécurité. Pour construire notre cadre formel, nous avons défini des formalismes algébriques et des logiques modales pour spécifier de manière élégante et concise les systèmes informatiques et leurs comportements. Nous avons aussi défini des techniques de vérification formelle pour vérifier la conformité des systèmes par rapport aux politiques de sécurité. Finalement, nous avons élaboré un opérateur de renforcement capable de générer un processus de renforcement automatique. Ce dernier a la capacité de réécrire une spécification algébrique d'un système informatique de manière à la rendre conforme à une politique de sécurité.

Acknowledgments

First and foremost, I wish to thank my research director and thesis advisor, Dr. Kamel Adi, who has guided my work on this project from beginning to end. His classes may have sparked my interest in computer systems security, but his knowledge, dedication, and passion for the field inspired me to seek answers through research. Throughout the thesis, I have often asked for his advice about a plethora of good and not-so-good (or outright bad) ideas I may have had. Dr. Adi has supervised the development of all published articles, from the early stages of validating the approach to the selection of conferences and journals. His valuable remarks and constructive criticism have always pushed me to think and write clearer and more concise. His assistance with thesis content, macros, templates, and style in general was invaluable. It is largely due to him that my thesis does not look like a 200 pages-long phrase.

My thanks are extended to the members of the doctoral committee, who accepted the tedious and ungrateful task of evaluating this thesis: Dr. Jurek Czyzowicz, Dr. Marc Frappier, Dr. Mohamed Mejri, and Dr. Luigi Logrippo. I am fortunate to have my research assessed through the lens of their vast and inspiring expertise. Their comments and questions are anxiously expected, as they will undoubtedly improve the quality and legibility of the thesis.

Some of the work described in chapters 5, 6 and 7 was done in collaboration with Lamia Hamza, a fellow PhD student from University of Bejaia, Algeria. She is also responsible for several aspects of the development of the PEA application. PEA inherited topology and system specification features from *GenSpec*, which was originally developed by Marc St-Laurent based on my requirements.

Finally, I would like to thank my family for being very supportive and concerned all along. Their genuine interest in the subject of this thesis helped me feel less guilty for not sharing enough of my time with them. I am so grateful for having them always by my side. My late father's excellent coffee kept me awake many nights when needed and my mother-in-law's cakes gave me the desired energy supplement. My brother and in-laws constantly bugged me about the status of the thesis, reminding me that I will eventually have to finish it. My son added a sense of urgency to the situation, giving me the greatest motivation for the final push required. Above all, my wife showed me unquestioning faith in my abilities, relentless support, and unconditional love.

List of Papers

1. K. Adi, A. El-Kabbal, and L. Pene. Distributed firewalls verification with mobile ambients. In Proceedings of the Workshop on Practice and Theory of Access Control Technologies, pages 29-34, Jan 2005.

2. K. Adi and L. Pene. Secrecy Correctness for Security Protocols. In Proceedings of the First International Conference on Distributed Frameworks for Multimedia Applications (DFMA '05), pages 22-29. IEEE Computer Society, 2005.

3. L. Pene and K. Adi. Calculus for distributed firewall specification and verification. In Proceedings of the 5th International Conference on Software Methodologies. SoMeT '06, pages 301-315. IOS Press, 2006.

4. K. Adi, L. Hamza, and L. Pene. Formal Modeling for Security Behavior Analysis of Computer Systems. In Proceedings of the 2008 International MCETECH Conference on e-Technologies (MCETECH '08), pages 49-59. IEEE Computer Society, 2008.

5. K. Adi and L. Pene. Formal Reasoning for Security Protocol Correctness. In Proceedings of the 7th International Conference on Software Methodologies. SoMeT '08, pages 63-83. IOS Press, 2008

6. K. Adi, L. Pene, and L. Sullivan. Games for non-repudiation protocol correctness. In International Journal of Wireless and Mobile Computing - IJWMC , vol. 4, no. 4, pages 305-313, Oct 2010.

7. L. Pene, L. Hamza, and K. Adi. Compliance Verification Algorithm for Computer Systems Security Policies. In Proceedings of the 2017 International MCETECH Conference on e-Technologies (MCETECH '17), pages 96-115. IEEE Computer Society, 2017.

8. L. Pene, L. Hamza, and Kamel Adi. Automatic Security Policy Enforcement in Computer Systems. In print, to appear in Computers & Security Journal, vol. 73, pages 156-171. Elsevier, Mar 2018.

Contents

List of Tables

List of Figures

Introduction

Context and Motivation

There are no such things as secure networks. They only exist in the imagination of those willing to believe it: IT support personnel, company managers, shareholders, business partners, etc. As soon as two or more computers systems are connected, there is a risk associated with the act of communication. Eavesdropping, content alteration and impersonation have been used for centuries. Nowadays, such hazards are mitigated by implementing basic mechanisms in the hope to ensure, at least partially, security properties such as confidentiality, integrity, availability, etc.

The new age of identity theft, cyber-harassment, and cyber-war has settled in and is here to stay. As the amount of stored digital information is increasing, the rate of security incidents is constantly on the rise. High-profile exploits are uncovered daily and they impact the lives of billions of direct or collateral victims. All incidents have one thing in common: illegal access to a host computer system. Leaked confidential information, be it personal or corporate, may cause embarrassment (e.g. private photos), damage to reputation (e.g. Hillary Clinton's email hack), financial loss (e.g. stolen prototypes or patents), or strained inter-governmental relations (Snowden, Wikileaks). The inability to access a system may lead to widespread panic (unavailability of Amazon, Twitter, Paypal, Sony's Playstation Network, and Netflix) or even serious safety consequences (Stuxnet, ransomware in the hospital infrastructure, identity of undercover agents). This climate imperatively requires to be addressed with sound mitigation measures.

This thesis is motivated by the dire need for better defined and managed

cyber protection. Numerous access control models have been created and implemented, with various degrees of success. User interfaces for defining security rules are also common, particularly for perimeter security devices (routers, switches, firewalls, etc.). Best practices and guidelines are useful references whenever plain language policies need to be translated into rules. Nonetheless, there are still serious challenges to overcome. The rapid pace of technological advances makes it difficult to cope with intricate new features and standards that become obsolete in a matter of months. Extensive testing is often ignored due to tight implementation timelines. Moreover, policy implementation is not a fully matured process and this is often done manually.

Meanwhile, significant progress has been made in the area of security enforcement mechanisms. Such mechanisms can be used to prevent behavior that is deemed unacceptable by the security policy. They require a description of the computer system and authorized behavior and formal methods have been successfully employed for developing accurate representations of both. Security policy enforcement has been formalized through frameworks that involve, among other methods, the use of process algebras and various modal logics. One advantage of this approach is that static and dynamic policy enforcement can be automated. The other major advantage is that the resulting framework enjoys useful properties that can be mathematically proven. This aspect is essential in policy compliance exercises. Therefore, we strongly believe that formal methods are ideally positioned for addressing security policy enforcement issues.

Research Goals

The problem of accurate implementation of security policies is very hard, especially for large and distributed networks. In the present thesis, we address this problem by elaborating a formal framework for security policies specification, automated verification and enforcement. These major issues justify and motivate a serious academic discussion and their resolution is, in our opinion, a coherent and considerable contribution to the field.

The main components of formal framework we envisioned must meet the following goals:

- *Specification for computer systems and security policies*: the process algebra and modal logic developed to specify systems and policies, respectively, must be suitable for representing mobility, process interactions, administrative domains, and permission to access those domains;
- *Policy verification technique*: the formal technique must determine, in a provable and automatic manner, whether the system specification satisfies a logic formula that models a security policy;
- *Policy enforcement methodology*: the methodology must allow the automatic calculation of necessary enforcements for policy compliance.

Original Contributions

The merits of the related research efforts are undeniable. Their limitations, however, prompted us for further investigation and eventually led to the development of the ideas presented here. The approach we adopted contains a number of substantial differences from previous works. The most notable are the support for both formal verification and enforcement, automation, and software implementation. Automation, in particular, was imperative in order to make the framework easy to implement, practical, and more than just another theoretical result. The complete list of research contributions in the thesis includes:

1. **Extensive literature review**: The related work in the areas of process algebras, logics, and security policies is presented and assessed from the mobility and access control perspectives. In order to showcase the variety of options, we covered in more detail the techniques and methodologies that are relevant for our research.
2. **In-depth analysis**: Critical reviews of the related research point out the merits and shortcomings of the various approaches. The analysis of their strengths and weaknesses influenced our choice of techniques and led to the development of the present study.
3. **Distributed firewall calculus**: The *FPC* calculus represents our original effort for understanding system specification and verification. We considered firewalls a good fit as they implement security policies. The

specific challenge we tackled is the consistency of filtering rules among distributed firewalls. While useful for system specification, our calculus also facilitates resolution of conflicting rules.

4. **Intruder-oriented calculus**: The intruder's perspective is often ignored while applying protective measures. The main effort usually goes into the implementation of the strongest possible authentication, encryption, etc. The *SSC* intuder-oriented calculus offers a different point of view and focuses on whether a malicious entity can take advantage of system flaws to advance their exploitation process. Therefore, it gives an analyst the ability to identify what series of vulnerabilities is required for compromising a system.

5. *CS2* **calculus**: This calculus introduces a different vision to system specification. The implementation-related constructs of *FPC* and *SSC* are discarded in favour of a richer illustration of process interaction. The syntax accommodates the notions of successful termination, process mobility and ambient protection with access keys. The semantics gains expressiveness and *CS2* becomes more appropriate for model checking, which is its intended purpose.

6. *SPL* **logic**: The logic is tailored for specifying security policies. The concepts of the *CS2* calculus are also supported in our *SPL* logic. Together, the calculus and logic allow for computer systems specification and verification with respect to a given security policy.

7. **Tableau-based proof system**: The tableau system devised for security policy verification enables formal model checking of *SPL*-expressed policies. Tableau finiteness, soundness, and completeness proofs support and augment the validity of the result. Nevertheless, proofs have to be produced manually, which is not ideal, especially for complex systems. This elicited the next contribution in the list.

8. **Implementation of the verification algorithm**: The tableau-based verification algorithm has been completely automated through an implementation in LoTREC, which is a tableau theorem prover. The implementation produces almost-instant formal proofs of policy compliance, eliminating the need for manual work.

9. **Extended $CS2^+$ calculus**: Additional syntactic constructs have been added to *CS2* in order to support dynamic enforcement of security policies. The extended calculus gains the ability to model protection changes, enforcement processes, and communication channels.

10. **Quotient operator**: The computation of enforcement with the aid of our quotient operator is a novelty. The corrective enforcement required to satisfy a security policy can be easily calculated with the technique we propose. The results of the enforcement can be verified by applying the calculus reduction rules. However, the enforcement correctness proof presented in this thesis eliminates the need for such manual verification.

11. **Automatic enforcement implementation**: A Java-based implementation has been developed under our guidance. The application is capable of producing topology diagrams through an intuitive drag-and-drop GUI. Given a security policy and system description, the application calculates automatically the enforcement needed.

12. **Policy enforcement framework**: To conclude, the integration of all the specification, verification, and enforcement components into one cohesive and comprehensive framework is another original contribution.

Thesis Outline

The structure of this thesis follows a classic layout, as described below:

- The Introduction outlines the thesis content: the motivation for our research, an overview of the problem to be addressed, our objectives, and a detailed account of our original contributions.

- Chapter 1 provides a literature review of the main concepts we employ throughout the thesis, grouped in distinct sections for process algebras, logics, and access control.

- Chapter 2 covers the related work in the area of policy verification and enforcement. The main focus is on policy classes, static enforcements, execution monitors and program rewriting. We also assess in this chapter the advantages and limitations of each technique.

- Chapter 3 proposes a calculus for firewall specification and verification. The *FPC* calculus permits detection of conflicting firewall rules in single and distributed firewall configurations. This was the first of a series of calculi we experimented with while searching for the most suitable syntax and semantics for compliance assessment.

- Chapter 4 presents our intruder-oriented analysis of a system's behavior. The *SSC* calculus we developed is used for computer system specification and for simulating the exploratory power of a malicious adversary. Additionally, it helps identifying system changes that would block the intruder's unauthorized access to system components.

- Chapter 5 introduces *CS2* and *SPL*, a process algebra and a logic devised for capturing mobility aspects in computer systems. The two formalisms are used in conjunction with a tableau-based proof system to verify policy compliance. The verification algorithm works on *SPL* formulas and is automated through an implementation in the LoTREC theorem prover.

- Chapter 6 proposes further enhancements to the calculus and logic that enable policy verification and, if required, policy enforcement through a quotient operator. The addition of the automatic enforcement calculation completes our formal framework. Moreover, all aspects of the framework, from system and policy specification to enforcement calculation, are implemented in an application called *PEA*.

- Chapter 7 summarizes the research and results presented in the thesis and suggests potential developments of our work.

- Finally, the Bibliography section lists all the articles, reports, books, and all other references that were cited in this thesis.

Building Blocks: Main Concepts and Literature Review

Chapter 1

Process Algebras and Logics

Abstract

The key concepts presented in this chapter, such as mobile ambients and modal logics, are at the core of our research. The comprehensive review of those concepts gathers the background knowledge needed to fully comprehend our aim, methodology and results. The merits and limitations of the various theories are also debated.

Introduction

There is no shortage of studies on the generic concept of *computer processes*. New results are always surfacing, enhancing previous work or stimulating completely new directions of research. Some of the methodologies come from maturing approaches, while some others are spawned by newer technology paradigms such as grid and cloud computing.

Our interest in the topics covered in the current chapter comes from the necessity to carry forward useful results and the desire to avoid their limitations. Our methodology involves defining a process calculus for specifying computer systems, a logic for describing formally security policy requirements and automatic enforcement of security policies based on system description and security policies. The reviewed literature covers specifically those issues over the lengths of the current chapter and the next.

1.1 Process Algebras

Processes are used to describe a system's behavior. This definition is common across all theories, irrespective of the domain. There are however different representations of processes. The oldest representation is the simple function input/output. In automata theory, a process is an automaton with states and transitions. The representations became more detailed as the field steadily matured. With concurrency theory, for instance, processes gain the ability to interact within parallel or distributed systems.

Process algebras represent a formal manner of concurrent systems modeling. Process interaction is depicted by means of more complex processes and dedicated operators for sequential, parallel, and alternate composition, along with some neutral elements. All aspects of process interaction and communication can be illustrated through algebraic laws that can be further analyzed. An exhaustive depiction of the capabilities of the system is obtained from the formal analysis of the model. The addition of non-algebraic methods, such as modal logic, produced the more appropriate term *process calculus*. Still, the original *process algebra* denomination is preferred by most. The relatively large number of existing process algebras and calculi that exist today is due to the variety of problems they are attempting to address: parallel composition, process equivalence, mobility, expressiveness, tooling, verification, timing, etc.

The most prominent early process algebras are CCS [76], CSP [56], ACP [13], and LOTOS [14, 71]. They share common features such as operators, atomic actions, operational semantics, etc. There are however distinct traits that make them fit for different applications. Milner and Hoare developed CCS and CSP around the same time, with CCS being published earlier and gaining notoriety as the first modern process algebra. An interesting comparison between CCS, CSP, and LOTOS is provided in [34].

The popularity of process algebras spans across several decades - from late 1970's to mid 1990's - and new variants or implementation tools are still being published. They are often the main topic of academic curriculum and constitute an excellent choice for demonstrating principles and concepts through small system case studies. A brief history of process algebras and

a comprehensive bibliography are presented in [5]. We only introduce the main concepts of CSP, the π-Calculus, and the mobile ambients calculus at this stage as the most relevant for the scope of this thesis. CSP has been given preference over CCS as we chose to highlight another one of Milner's creation, the π-Calculus.

1.1.1 CSP

The final version of Hoare's CSP (Communicating Sequential Processes) [56] was published in 1985, although some ideas have been presented in a 1976 academic report and an early, model lacking, CSP was introduced in 1978. The algebra was inspired by Pascal and by Dijkstra's guarded command language. It promotes the use of message passing in synchronous communications depicted in a trace-based model where events are treated as instantaneous. Process specification (or behavior) is described in terms of action sequences, which constitute the actual traces. Processes can be assembled into systems, where they coexist and interact, hence the concurrency and communication aspects of the algebra. The elements of the CSP syntax are:

- *Prefix*: for specifying events that precede a certain process;
- *Recursion*: for representing repetitive tasks;
- *Choice*: for expressing alternate behavior;
- *Mutual recursion*: for accommodating multiple solutions.

Let s and t be traces and let A be a set of events. The following operations are defined on traces:

- Catenation $s^\frown t$: for trace construction from individual events;
- Restriction $t \upharpoonright A$: for extracting the subset of events in A from a trace t;
- Head s_0 and tail s': for identifying the first and last elements of a trace;
- Star $A*$: for denoting all the finite traces with symbols of A;

- Ordering $s \leq t$: for defining an ordering relation between traces s and t;
- Length $\#t$: for counting the number of events of a trace t.

Other trace operations include symbol changing, interleaving, selection, reversal, and composition. CSP has dedicated operators for expressing both deterministic and non-deterministic choice, and is the lone well established process algebra that has them. It also has operators for alternating processes, subordinate processes and restart after catastrophe.

The aim of CSP as declared by its author is to be the simplest mathematical theory that will "provide clear assistance to the programmer in his tasks of specification, design, implementation, verification and validation of complex computer systems" [56]. While not optimal for all these tasks, the results yielded by this approach are proof of the considerable success CSP still enjoys.

1.1.2 The π-Calculus

The π-Calculus was introduced by Milner [77, 78] and further developed by Parrow [86] and Walker [104]. The calculus is based primarily on CCS, but it is not necessarily considered an extension of it, since it introduces many new concepts. For instance, new communication channel names can be transmitted along established channels. This brings forward the notion of mobility, which is central to our research and many others. The syntax and operational semantics of the π-Calculus are presented below.

π-calculus Syntax

The syntax of the π-calculus consists of agents, prefixes, and definitions. It differs from the syntax of CCS by the ability to send and receive communication port names through actions. Port names range over $a, b, ..., z$ and they intuitively represent access rights. Names are also used for denoting data values and variables. An agent can take one of the following eight forms:

- *Nil*: the empty agent 0 that does not execute any action;

- *Prefix*: denoted by α and used for describing communication; an agent can send and receive data (the name x) along a port (channel a), or can evolve without interacting with the environment (through the silent prefix τ);
- *Sum*: denotes the choice of an agent to act as either P or Q;
- *Parallel*: represents concurrency through the parallel execution of P and Q;
- *Match* and *Mismatch*: define conditions for behaving like P;
- *Restriction*: describes the creation of a new name, which can be used as a communication channel;
- *Identifier*: represents an invocation of an agent P where the declared parameters are replaced by the actual parameters $y_1, ..., y_n$.

Later versions of the π-Calculus also consider agent replication, $!P$, which can be used for representing persistent services. The syntax only introduces a bare minimum of constructs, making it impractical for programming purposes. Still, several variants of the calculus exist that either lack Sum or the $Match$ and $Mismatch$ pair. Other variants appropriately extended it to address data types, loops, recursion, asynchronous communication, etc.

Operational Semantics of π-Calculus

Table 1.2 presents the operational semantics of π-Calculus. Most rules are similar to those in CCS. The main differences are in the OPEN, RES, and PAR rules. The OPEN rule allows lifting the restriction of a channel a and transforms an unbound Output action denoted by $\xrightarrow{\overline{a}x}$ into a bound one denoted by $\xrightarrow{\overline{a}\nu x}$. The bn (bound names) and fn (free names) functions appear in the RES and PAR. $bn(P)$ represents the set of bound occurrences in P, while $fn(P)$ represents the set of free occurrences in P. Similarly, $bn(\alpha)$ and $fn(\alpha)$ correspondent respectively to the set of bound and free occurrences in a prefix α.

Table 1.1: The syntax of the π-calculus

Prefixes	$\alpha :=$	$\overline{a}x$	Output
	$\mid$	$a(x)$	Input
	$\mid$	τ	Silent
Agents	$P, Q :=$	0	Nil
	$\mid$	$\alpha.P$	Prefix
	$\mid$	$P + Q$	Sum
	$\mid$	$P \mid Q$	Parallel
	$\mid$	if $x = y$ then P	Match
	$\mid$	if $x \neq y$ then P	Mismatch
	$\mid$	$(\nu x)P$	Restriction
	$\mid$	$A(y_1, ..., y_n)$	Identifier
Definitons		$A(x_1, ..., x_n) \stackrel{def}{=} P$	where$(i \neq j \Rightarrow x_i \neq x_j)$

Table 1.2: The operational semantics of π-calculus

STRUCT	$\dfrac{P' \equiv P,\ P \xrightarrow{\alpha} Q,\ Q \equiv Q'}{P' \xrightarrow{\alpha} Q'}$
PREFIX	$\dfrac{}{\alpha.P \xrightarrow{\alpha} P}$
SUM	$\dfrac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$
MATCH	$\dfrac{P \xrightarrow{\alpha} P'}{\text{if } x = x \text{ then } P \xrightarrow{\alpha} P'}$
MISMATCH	$\dfrac{P \xrightarrow{\alpha} P',\ x \neq y}{\text{if } x \neq y \text{ then } P \xrightarrow{\alpha} P'}$
PAR	$\dfrac{P \xrightarrow{\alpha} P',\ \ \mathrm{bn}(\alpha) \cap \mathrm{fn(Q)}=\emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$
COM	$\dfrac{P \xrightarrow{a(x)} P',\ Q \xrightarrow{\overline{a}u} Q'}{P \mid Q \xrightarrow{\tau} P'\{u/x\} \mid Q'}$
RES	$\dfrac{P \xrightarrow{\alpha} P',\ x \notin \alpha}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'}$
OPEN	$\dfrac{P \xrightarrow{\overline{a}x} P',\ a \neq x}{(\nu x)P \xrightarrow{\overline{a}\nu x} P'}$

The sets fn and bn are defined as follows:

$$\begin{array}{lcl} fn(0) & = & \emptyset \\ fn(ax.P) & = & \{a, x\} \cup fn(P) \\ fn(a(x).P) & = & \{a\} \cup fn(P) \\ fn((vx)P) & = & fn(P) \end{array} \qquad \begin{array}{lcl} bn(0) & = & \emptyset \\ bn(ax.P) & = & bn(P) \\ bn(a(x).P) & = & \{x\} \cup bn(P) \\ fn((vx)P) & = & \{x\} \cup bn(P) \end{array}$$

The π-Calculus is the main focus of a book on mobile systems analysis published by Sangiorgi and Walker [92]. Their landmark study provides a solid framework for understanding theoretical and practical aspects of process mobility. The book is among the most respected references for using π-Calculus to express systems and reasoning about their behaviors and properties.

Many computational models, languages, and calculi are based on the π-Calculus. Some are focused on security (Spi), other on typing systems, or replication issues. Milner himself developed different variants of his original calculus. However, none of them treat the security aspects of mobility in a manner that is suitable for our work.

1.1.3 The Ambient Calculus

The concept of ambient calculus was first introduced by Cardelli and Gordon in [21]. An ambient is a named domain and it can contain processes that operate inside it. The delimited space representing an ambient has a name, an interior and an exterior. Administrative boundaries and processes can be expressed in terms of ambients that can travel from one site to another. The movement of an ambient process is governed by its capabilities, including the possibility to move inside or outside another ambient.

The ambient calculus aims to capture the notion of process mobility by modeling the various movement possibilities between different administrative sites. In particular, security is expressed through movement capacity rather than access controls or cryptographic primitives. Processes can go in and out of ambients or open them. Moreover, an ambient can move inside or outside another ambient, carrying the enclosed processes with it. The

corresponding capabilities of the enclosed ambient processes are: *in*, *out* and *open*.

The Language

We present in Table 1.3 the syntax of the Mobile Ambient calculus (or simply ambient calculus) [21, 22, 43]. The syntax consists in three syntactic categories: names, processes, and capabilities. Names are used for denoting administrative sites (or locations). Six primitives are employed for describing processes and their interaction: restriction, inactivity, composition, replication, ambient and action. The restriction primitive captures local scope. Inactivity indicates that a process does nothing. Composition allows processes to execute in parallel. Replication permits generating several copies of a process. An ambient denotes a process that operates inside a named zone. The use of a movement capability by a process is called an action. The three capabilities are straightforward: *in* n facilitates entrance into a co-located ambient called n, *out* n enables a process to leave its parent ambient n, and *open* n dissolves the borders around the ambient n.

The operational semantics of the calculus comprises a structural congruence between processes, denoted by $\equiv$, and an action regulating reduction relation, denoted by $\rightarrow$. The process equivalence classes in Table 1.4 define structural congruence as identity rather than structural equivalence. The comprehensive list of classes is derived from the following properties: reflexivity, symmetry, transitivity, restriction, parallelism, replication, ambient, action, commutativity, associativity and replication of parallelism, restriction, parallelism and ambient for restriction and parallelism, restriction and replication for inactivity. The ambient calculus reduction relation is defined by the rules in Figure 1.1. Summarily, the rules stipulate that the reduction manages restrictions, ambient processes and parallel compositions of processes, and that structural congruence can help rearranging expressions of ambient processes.

Applications

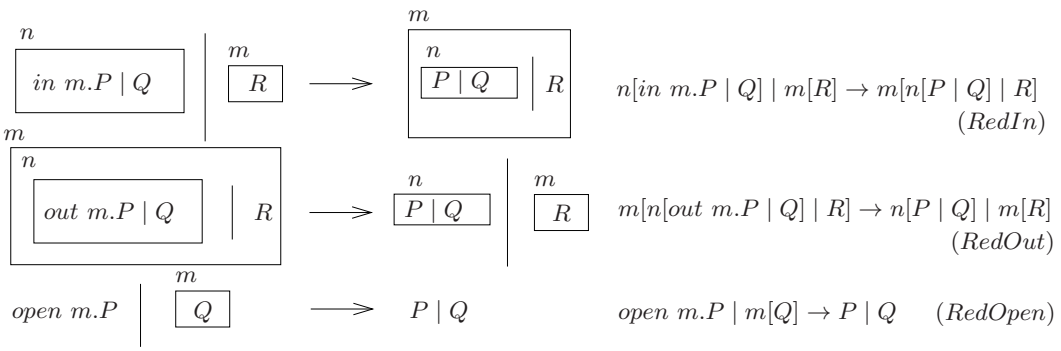
There are two known software implementations of the ambient calculus: one

Table 1.3: Ambient calculus syntax

n		names
P, Q	::=	processes
	$(\nu n)P$	restriction
	0	inactivity
	$P \mid Q$	composition
	$!P$	replication
	$n[P]$	ambient
	$M.P$	action
M	::=	capabilities
	in n	can enter n
	out n	can exit n
	open n	can open n

Table 1.4: Structural congruence in ambient calculus

$P \equiv P$	(Struct Refl)
$P \equiv Q \Rightarrow Q \equiv P$	(Struct Symm)
$P \equiv Q \wedge Q \equiv R \Rightarrow P \equiv R$	(Struct Trans)
$P \equiv Q \Rightarrow (\nu n)P \equiv (\nu n)Q$	(Struct Res)
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	(Struct Par)
$P \equiv Q \Rightarrow !P \equiv !Q$	(Struct Repl)
$P \equiv Q \Rightarrow n[P] \equiv n[Q]$	((Struct Amb)
$P \equiv Q \Rightarrow M.P \equiv M.Q$	(Struct Action)
$P \mid Q \equiv Q \mid P$	(Struct Par Comm)
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(Struct Par Assoc)
$!P \equiv P \mid !P$	(Struct Repl Par)
$(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$	(Struct Res Res)
$(\nu n)(P \mid Q) \equiv P \mid (\nu n)Q$ if $n \notin fn(P)$	(Struct Res Par)
$(\nu n)(m[P]) \equiv m[(\nu n)P]$ if $n \neq m$	(Struct Res Amb)
$P \mid 0 \equiv P$	(Struct Zero Par)
$(\nu n)0 \equiv 0$	(Struct Zero Res)
$!0 \equiv 0$	(Struct Zero Repl)



(a) The basic reduction rules

$$P \to Q \Rightarrow (\nu n)P \to (\nu n)Q \qquad (RedRes)$$

$$P \to Q \Rightarrow n[P] \to n[Q] \qquad (RedAmb)$$

$$P \to Q \Rightarrow P \mid R \to Q \mid R \qquad (RedPar)$$

$$P' \equiv P, P \to Q, Q \equiv Q' \Rightarrow P' \to Q' \qquad (Red \equiv)$$

(b) Other reduction rules

Figure 1.1: The reduction relation in ambient calculus

non-distributed - in Java (by Cardelli [23]), and one distributed - in JoCaml (by Fournet, Levy, and Schmitt *et al.* [39, 93]). Several developments of the initial calculus were introduced by Cardelli, Gordon and their collaborators, such as typed ambients [22], equational properties [43], a modal logics for mobile ambients [24], etc.

Gordon and Cardelli provided some examples [21] for the expressiveness of their calculus. They use the capabilities of mobile ambients to model locks, choice, dissolution, renaming, iteration, synchronization, firewall access and some other complex actions. We present in this section the interesting examples of locks and firewalls, cited directly from [21] in the authors' own words.

Locks

Let $release\ n.P$ be a non-blocking operation that releases a lock n and continues with P. Let $acquire\ n.P$ be a potentially blocking operation that attempts to acquire a lock n, and that continues with P if and when the lock is released. These operations can be defined as follows:

$$\begin{aligned} acquire\ n.P &\triangleq open\ n.P \\ release\ n.P &\triangleq n[] \mid P \end{aligned}$$

Given two locks n and m, two processes can "shake hands" before continuing with their execution:

$$acquire\ n.release\ m.P \mid release\ n.acquire\ m.Q$$

Firewall Access

This is another example of a mobile agent trying to gain access to an ambient. In this case, though, we assume that the ambient, a firewall, keeps its name completely secret, thereby requiring authentication prior to entry. The agent crosses a firewall by means of previously arranged passwords k, k', and k''. The agent exhibits the password k' by using a wrapper ambient that has k' as its name. The firewall, which has a secret name w, sends out a pilot

ambient, $k[out\ w.in\ k'.in\ w]$, to guide the agent inside. The pilot ambient enters an agent by performing in k' (therefore verifying that the agent knows the password), and is given control by being opened. Then, in w transports the agent inside the firewall, where the password wrapper is discarded. The third name, k'', is needed to confine the contents Q of the agent and to prevent Q from interfering with the protocol.

The final effect is that the agent physically crosses into the firewall; this can be seen below by the fact that Q is finally placed inside w. (For simplicity, this example is written to allow a single agent to enter.) Assume $(fn(P) \cup fn(Q)) \cap k, k', k'' = \varnothing$ and $w \notin fn(Q)$:

$$\begin{aligned} Firewall &\triangleq (\nu w)w[k[out\ w.in\ k.in\ w] \mid open\ k'.open\ k''.P] \\ Agent &\triangleq k'[open\ k.k''[Q]] \end{aligned}$$

$$\begin{aligned} & Agent \mid Firewall \\ \equiv\ & (\nu w)(k'[open\ k.k''[Q]] \mid w[k[out\ w.in\ k'.in\ w] \mid open\ k'.open\ k''.P]) \\ \rightarrow^*\ & (\nu w)(k'[openk.k''[Q] \mid k[in\ w]] \mid w[open\ k'.open\ k''.P]) \\ \rightarrow^*\ & (\nu w)(k'[k''[Q] \mid in\ w] \mid w[open\ k'.open\ k''.P]) \\ \rightarrow^*\ & (\nu w)(w[(k'[k''[Q]] \mid open\ k'.open\ k''.P]) \\ \rightarrow^*\ & (\nu w)w[Q \mid P] \end{aligned}$$

There is no guarantee here that any particular agent will make it inside the firewall. Rather, the intended guarantee is that if any agent crosses the firewall, it must be one that knows the passwords.

We use an equation to express the security property of the firewall. If $(fn(P) \cup fn(Q)) \cap k, k', k'' = \varnothing$ and $w \notin fn(Q)$, then we can show that the interaction of the agent with the firewall produces the desired result up to contextual equivalence.

$$(\nu\ k\ k'\ k'')(Agent \mid Firewall) \simeq (\nu w)w[Q \mid P]$$

Since contextual equivalence takes into account all possible contexts, the equation above states that the firewall crossing protocol works correctly in the

presence of any possible attacker that may try to disrupt it. The assumption that an attacker does not already know the password is represented by the restricted scoping of k, k', k''.

1.2 Process Logics

The modal logic that we define in this thesis is inspired from the ambient logic of Cardelli and Gordon [24, 25], which has been further developed by Sangiorgi [91] and Hirschkoff [54]. Our logic also builds on the mobile processes application of the Milner's π-Calculus [77, 78] due to Sangiorgi [92], and the spatial logic for mobile processes of Hirschkoff [55]. Milner's contribution to the field of process logics, adding to his famous process calculus, is briefly presented in a subsection dedicated to the Hennessy-Milner logic [51, 52]. The ADM logic for security protocols of Adi *et al.* [2], closely related to our area of interest, is also reviewed here. The BAN logic [20], a modal logic of belief devised by Burrows, Abadi, and Needham, is another fine example of related work, as it tackles the specification and verification of cryptographic protocols. The μ-logic [63] and the modal logic for mobile processes of De Nicola and Loreti [81] are among our other related resources.

1.2.1 The Hennessy-Milner Logic

The Hennessy-Milner logic [51, 52, 99] is used for depicting local capabilities of a process. The syntax of the Hennessy-Milner logic, presented in Table 1.5, involves boolean connectives for modal formulas Φ_1 and Φ_2 and two modal operators for a set of actions $\{K_1, \ldots, K_n\}, n \geq 0$. The abbreviation $\langle K \rangle \Phi$ is used for the formula $\langle K_1 \rangle \Phi \wedge \ldots \wedge \langle K_n \rangle \Phi$. The abbreviation $[K]\Phi$ is used for the formula $[K_1]\Phi \vee \ldots \vee [K_n]\Phi$.

The fact that a process E satisfies (or not) a property described by the formula Φ is defined inductively on the structure of formulas. Formula satisfaction, denoted by $\models$, is presented in Table 1.6. All processes satisfy tt and none satisfy ff. A process satisfies $\Phi_1 \wedge \Phi_2$ (and respectively, $\Phi_1 \vee \Phi_2$) if it satisfies both Φ_1 and Φ_2 (respectively either Φ_1 or Φ_2). A process satisfies $\langle K \rangle \Phi$ if it is possible to perform action K and thereby satisfy property Φ. A

Table 1.5: The syntax of the Hennessy-Milner logic

$$\Phi ::= tt \mid ff \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid [K]\Phi \mid \langle K \rangle \Phi$$

Table 1.6: The satisfaction relation of the Hennessy-Milner logic

$$\begin{array}{lll}
E \vDash tt & & \\
E \nvDash ff & & \\
E \vDash \Phi_1 \wedge \Phi_2 & \text{iff} & E \vDash \Phi_1 \text{ and } E \vDash \Phi_2 \\
E \vDash \Phi_1 \vee \Phi_2 & \text{iff} & E \vDash \Phi_1 \text{ or } E \vDash \Phi_2 \\
E \vDash [K]\Phi & \text{iff} & \forall F \in \{E' : E \xrightarrow{a} E' \text{ and } a \in K\}.F \vDash \Phi \\
E \vDash \langle K \rangle \Phi & \text{iff} & \exists F \in \{E' : E \xrightarrow{a} E' \text{ and } a \in K\}.F \vDash \Phi
\end{array}$$

process satisfies $[K]\Phi$ if the state it reaches satisfies Φ irrespective of which process K is executed.

1.2.2 The ADM Logic

The ADM logic [2] developed by Adi, Debbabi and Mejri allows the specification of security protocols properties. Both classical security properties (authentication, secrecy and integrity) and electronic commerce properties (non-repudiation, anonymity, good atomicity, money atomicity, certified delivery, etc.) can be expressed with this modal logic. The logic is compact, linear, dynamic, and expressive.

The classic view of a protocol as a distributed algorithm is employed by the authors. Thus, it is specified as a finite sequence of statements that describe internal and external actions performed as messages are transmitted.

Protocol executions (or runs) generate sequences of events (or traces) related to send and receive actions. Dynamic executions representation is achieved through a trace-based model. A trace t belongs to a set of traces $\mathcal{T}$ and can be empty (ε). Traces reflect valid protocol executions in the presence of a malicious smart intruder.

Several other elements support the construction of a model suitable for verification. A pattern, denoted by p, is a trace abstraction where some actions are replaced by variables, while $p_1 \looparrowright p_2$ is a modal operator indexed by the patterns p_1 and p_2. Substitutions, represented by σ, are internal parameters used for giving a semantics to $p_1 \looparrowright p_2$. An environment e helps dealing with recursive formulae and giving a semantics to X. Finally, the formula $\nu X.\Phi$ is a recursive formula in which the greatest fixed point operator ν binds all free occurrences of X in Φ. The resulting model is suitable for the detection of security flaws, which are identified as traces that violate security properties.

The logic's linearity and use of recursive formulas allows actions in a trace to be counted. Security properties can be specified in terms of modalities supported by the logic (before, after, necessary, possible, etc.). The construction of formulas is based on patterns. Each variable in a pattern abstracts a possibly empty sequence of actions. The syntax of the logic is depicted in Table 1.7 below, where X is a formula variable, $\neg$, $\wedge$, and ν represent negation, conjunction, and a recursive formula.

Table 1.7: The syntax of the ADM logic

$$\Phi ::= X \mid \neg\Phi \mid [p_1 \looparrowright p_2]\Phi \mid \Phi_1 \wedge \Phi_2 \mid \nu X.\Phi$$

The denotational semantics of the logic is presented in Table 1.8, where σ, σ' denote substitutions, $e[X \mapsto U]$ is an environment where the formula variable X has been replaced with U, and $t_{\downarrow}$ contains all traces that can be extracted from t by eliminating some actions.

The tableau-based proof system of the logic leads to a modular denotational

Table 1.8: The denotational semantics of the ADM logic

$$
\begin{array}{rcl}
[\![X]\!]_e^{t,\sigma} & = & e(X) \\
[\![\neg\Phi]\!]_e^{t,\sigma} & = & t_{\downarrow} - [\![\Phi]\!]_e^{t,\sigma} \\
[\![\Phi_1 \wedge \Phi_2]\!]_e^{t,\sigma} & = & [\![\Phi_1]\!]_e^{t,\sigma} \cap [\![\Phi_2]\!]_e^{t,\sigma} \\
[\![[p_1 \looparrowright p_2]\Phi]\!]_e^{t,\sigma} & = & \{u \in t_{\downarrow} \mid \forall \sigma' : p_1\sigma\sigma' = u \Rightarrow p_2\sigma\sigma' \in [\![\Phi]\!]_e^{p_2\sigma\sigma', \sigma'\circ\sigma}\} \\
[\![\nu X.\Phi]\!]_e^{t,\sigma} & = & \nu f \text{ where } \left\{ \begin{array}{l} f : \ 2^{\mathcal{T}} \rightarrow 2^{\mathcal{T}} \\ \qquad U \rightarrow [\![\Phi]\!]_{e[X \mapsto U]}^{t,\sigma} \end{array} \right.
\end{array}
$$

semantics and local model checking. This is a significant advantage, since not all traces in $t_{\downarrow}$ need to be verified. Instead, only the traces used in the construction of the formula-labeled tree are visited. This backward-chaining proof search method involves a set of inference rules that constitute the tableau proof system. The system includes inference rules for all syntactic constructs: negation, conjunction, patterns and recursive formulas. Tableau finiteness, which is formally proven, ensures that the verification is bounded. The expressiveness of the logic is demonstrated by specifying security properties such as authentication, secrecy, money atomicity and goods atomicity.

1.2.3 The Ambient Logic

The ambient logic [24, 25] is a modal logic based solidly on the ambient calculus and it is closely related to intuitionistic linear logic and to bunched logics. The ambient logic is specifically devised for expressing name, time and location properties of mobile processes with the aid of logical operators.

Syntax and Semantics of The Ambient Logic

The syntax of logical formulas is summarized in Table 1.9. The propositional logic is given by the true, negation and disjunction formulas. The tree-like structures of locations are modeled through five formulas: void, composition and its adjunct, and location and its adjunct. Revelation and its adjunct are used for publicizing or hiding a name. The sometime and somewhere modalities state that certain states are possible "further away" in space or time, respectively. Only names can contain quantified variables, denoted by η in the table. Therefore, such variables can only be part of the location, placement, revelation and hiding formulas.

The satisfaction relation $P \vDash \mathcal{A}$ defined inductively in Table 1.10 means that the process P satisfies the closed relation $\mathcal{A}$. Each syntactic formula has a corresponding satisfaction relation. The sort of processes is represented by Π, the sort of formulas by Φ, the sort of variables by ϑ, and the sort of names by Λ. The relations $P \downarrow P'$ and $P \downarrow^* P'$ mean that P contains P' within one level or at some nesting level, respectively. The meaning of the connectives in Table 1.10 is the following:

- The first three connectives (for the true, negation and disjunction formulas) give the classical propositional logic
- A process P satisfies the formula 0 if $P \equiv 0$
- A process P satisfies the formula $\mathcal{A} \mid \mathcal{B}$ if there exist processes P' and P'' such that P has the shape $P' \mid P''$ with P' satisfying $\mathcal{A}$ and P'' satisfying $\mathcal{B}$
- A process P satisfies the formula $n[\mathcal{A}]$ if there exists a process P' such that P has the shape $n[P']$ with P' satisfying $\mathcal{A}$
- A process P satisfies the formula $\mathcal{A} \triangleright \mathcal{B}$ if P (together with its context) manages to satisfy $\mathcal{B}$ under any possible attack by an opponent that is bound to satisfy $\mathcal{A}$
- A process P satisfies the formula $n[\mathcal{A}]$ if $P \equiv 0$
- A process P satisfies the formula $\mathcal{A}@n$ if P (together with its context) manages to satisfy $\mathcal{A}$ even when placed into a location called n

Table 1.9: Logical formulas in ambient logic

η, μ	a name n or a variable x
$\mathcal{A}, \mathcal{B}, \mathcal{C}$ ::=	
$\mathbf{T}$	true
$\neg\mathcal{A}$	negation
$\mathcal{A} \vee \mathcal{B}$	disjunction
$\mathbf{0}$	void
$\mathcal{A} \mid \mathcal{B}$	composition
$\mathcal{A} \rhd \mathcal{B}$	guarantee
$\eta[\mathcal{A}]$	location
$\mathcal{A}@\eta$	placement
$\eta ® \mathcal{A}$	revelation
$\mathcal{A} \oslash \eta$	hiding
$\diamond\mathcal{A}$	sometime modality
$\Diamond\mathcal{A}$	somewhere modality
$\forall x.\mathcal{A}$	universal quantification

Table 1.10: Satisfaction in ambient logic

$\forall P \in \Pi$	$P \models \mathbf{T}$		
$\forall P \in \Pi, \mathcal{A} \in \Phi$	$P \models \neg\mathcal{A}$	$\triangleq$	$\neg P \models \mathcal{A}$
$\forall P \in \Pi, \mathcal{A}, \mathcal{B} \in \Phi$	$P \models \mathcal{A} \vee \mathcal{B}$	$\triangleq$	$P \models \mathcal{A} \vee P \models \mathcal{B}$
$\forall P \in \Pi$	$P \models \mathbf{0}$	$\triangleq$	$P \equiv \mathbf{0}$
$\forall P \in \Pi, \mathcal{A}, \mathcal{B} \in \Phi$	$P \models \mathcal{A} \mid \mathcal{B}$	$\triangleq$	$\exists P', P'' \in \Pi. P \equiv P' \mid P'' \wedge P' \models \mathcal{A} \wedge P'' \models \mathcal{B}$
$\forall P \in \Pi, \mathcal{A}, \mathcal{B} \in \Phi$	$P \models \mathcal{A} \rhd \mathcal{B}$	$\triangleq$	$\forall P' \in \Pi. P' \models \mathcal{A} \Rightarrow P \mid P' \models \mathcal{B}$
$\forall P \in \Pi, n \in \Lambda, \mathcal{A} \in \Phi$	$P \models n[\mathcal{A}]$	$\triangleq$	$\exists P' \in \Pi. P \equiv n[P'] \wedge P' \models \mathcal{A}$
$\forall P \in \Pi, n \in \Lambda, \mathcal{A} \in \Phi$	$P \models \mathcal{A}@n$	$\triangleq$	$n[P] \models \mathcal{A}$
$\forall P \in \Pi, n \in \Lambda, \mathcal{A} \in \Phi$	$P \models n®\mathcal{A}$	$\triangleq$	$\exists P' \in \Pi. P \equiv (\nu n)P' \wedge P' \models \mathcal{A}$
$\forall P \in \Pi, n \in \Lambda, \mathcal{A} \in \Phi$	$P \models \mathcal{A} \oslash n$	$\triangleq$	$(\nu n)P \models \mathcal{A}$
$\forall P \in \Pi, n \in \Lambda, \mathcal{A} \in \Phi$	$P \models \diamond\mathcal{A}$	$\triangleq$	$\exists P' \in \Pi. P \rightarrow^* P' \wedge P' \models \mathcal{A}$
$\forall P \in \Pi, \mathcal{A} \in \Phi$	$P \models \Diamond\mathcal{A}$	$\triangleq$	$\exists P' \in \Pi. P \downarrow^* P' \wedge P' \models \mathcal{A}$
$\forall P \in \Pi, x \in \vartheta, \mathcal{A} \in \Phi$	$P \models \forall x.\mathcal{A}$	$\triangleq$	$\forall m \in \Lambda. P \models \mathcal{A}\{x \leftarrow m\}$

- A process P satisfies the formula $®\mathcal{A}$ if it is possible to pull a restricted name of P to the top and call it n, and then strip off the restriction to leave a residual that satisfies $\mathcal{A}$
- A process P satisfies the formula $\mathcal{A} \oslash n$ if $(\nu n)P$ satisfies $\mathcal{A}$; this connective is used for hiding a name
- A process P satisfies the formula $\diamond\mathcal{A}$ if $\mathcal{A}$ holds in the future for some residual P' of P, where $P \rightarrow^* P'$
- A process P satisfies the formula $\Diamond\mathcal{A}$ if $\mathcal{A}$ holds at some sublocation P' within P, where $P \downarrow^* P'$
- A process P satisfies the formula $\forall x.\mathcal{A}$ if for all names m we have that P satisfies $\mathcal{A}\{x \leftarrow m\}$.

Many other connectives have been derived, of which the most useful are: logical equivalence, every component satisfies $\mathcal{A}$, some component satisfies $\mathcal{A}$, everytime and everywhere modalities, $\mathcal{A}$ is unsatisfiable. The logic can be used to derive rules for the directives and to express properties guaranteed by type systems. The logical rules for abstract reasoning freshness have been provided by fresh-name quantification. A model checking algorithm for a fragment of the logic has also been developed by the authors.

Expressiveness, Extensionality and Intensionality of the Ambient Logic

The ambient logic was proposed by Cardelli and Gordon for the purpose of expressing properties of process mobility in the ambient calculus. The results obtained by Hirschkoff *et al.* [54] demonstrate that ambient logic is a very expressive formalism. They were capable to derive, using non-trivial technical developments, some formulas that express capabilities of processes for movement and for communication, the persistence of processes, and free occurrences of names in processes. The results are remarkable given that there is no connective in the logic that is directly related to such properties (neither a construct, nor an infinitary operator).

Hirschkoff and his co-authors use the temporal modality and exploit adjunct

connectives to introduce a form of contextual reasoning in order to analyze the desired properties. Their first important result is the definition of characteristic formulas for image-finite ambient processes. Those formulas capture the equivalence class of a process with respect to the induced logical equivalence, which is unusual in modal logics since the ambient logic has no fixed-point operator. Also, they demonstrate that the image-finiteness condition on processes is weaker than finite-state, since computations containing visible actions (such as input and output actions) are not taken into consideration.

Their second expressiveness result proves that ambient logic is an intensional logic. This is due to the fact that the logic allows inspecting the structure of processes by capturing its interaction capabilities. More formally, the equivalence induced by the logic coincides with structural congruence on processes. This result says that ambient logic is a very fine grained logic.

The work presented in [54] is a direct continuation of Hirschkoff's previous study on spatial logics [55]. He defines a logic for concurrency that is not only intensional, but also extensional. The intensional aspect is expected, as most existing spatial logics induce an equivalence that coincides with structural congruence. The contextual spatial logic for the π-calculus developed by Hirschkoff only has composition adjunct and hiding, but it induces an equivalence that coincides with strong early bisimilarity. This particular property allows decomposing spatial logics into two parts: one that counts and one that observes. The first part is extensively used in the later study [54] for counting the parallel components of a process.

The intensionality and extensionality of the ambient logic have also been discussed by Sangiorgi [91]. The intensional and extensional aspects are verified by comparing the equivalence $=_L$ induced by the logic on processes with the structural congruence $\equiv$ of processes from the ambient calculus and the behavioral equivalence $\approx$. The logic permitted the observation in very fine detail of the internal structure of processes similar to that of the structural congruence $\equiv$. The author demonstrates that $=_L$ is strictly included in $\approx$ and that, for a synchronous version of the ambient calculus, $=_L$ is precisely the structural congruence $\equiv$.

1.3 Conclusion

We chose to build on the foundation of the powerful ambient calculus because of its expressiveness. The ability to model mobility aspects such as location, entering, exiting and restricting access are its strong points and underline its potential. The flexibility and scalability offered by representing computer systems with ambients is also an important factor in our decision. We can depict each machine by an ambient, each message on the network by an ambient, each application by an ambient, and the network itself by an ambient. Likewise, to represent a file or an external hard drive we could employ ambients.

However, the ambient calculus was not built for the purpose of modeling security aspects of mobility. Specifying anything beyond the basic name protection can be cumbersome and more or less accurate. For instance, the mobile ambients fail to capture the actual firewall behavior. The firewall example offered by Cardelli and Gordon, involving an agent that brings the processes on the other side of the firewall by using predefined keys, resembles more to the creation of a secure channel used by a server-agent pair. The addition of new syntactic components such as keys, protection actions, communication channels, and policy enforcement in our calculus facilitates a more natural manner of modeling a much richer process behavior.

The ambient logic was our main inspiration for the logic that we devised. Logic formulas can be used for describing security policies, which can be imposed on computer systems specified with the process algebra. The differences from our logic are, however, very obvious to the attentive reader. The correlation between our logic and calculus prompted for several changes. For instance, we have eliminated all primitives that were not relevant to our calculus (guarantee, revelation, hiding, etc.), and introduced new ones (capability, protected location).

Once the process algebra and logic were chosen as the basic building blocks for modeling computer systems and security policies, we changed our focus on the formal model checking aspect of the framework, which we review in the next chapter.

Chapter 2

Security Policy Verification and Enforcement Techniques

Abstract

The main focus of our research are the formal verification and enforcement of security policies. This chapter introduces the concept of security policy and presents several approaches to formal enforcement, including the state-of-the-art theories and implementations. The shortcomings of each methodology are highlighted and analyzed.

Introduction

Security policies have always been associated with computer systems, in manners more or less obvious or formal. There are two main types of limitations for system use: capability based and restriction based. The capability based limitations come from either hardware (simply not able to perform certain tasks), design (lack of functionality), or user proficiency. The restriction based limitations can be largely associated with security policies.

Restrictions on system resources use can be imposed at the physical level (printers, network connections, USB ports) or the logical level (files, CPU utilization, bandwidth). They can also refer to data flow manipulation and distinct access permissions. Given the diversity of systems and requirements,

it is difficult to define and enforce a standard, universally applicable fashion for stating such restriction in the form of security policies and enforcing them. The opinions on the best implementation stage and strategy range from application design to runtime execution, covering mostly every intermediate stage. Section 2.3 in the previous chapter deals with some aspects of access control and information flow in mobile ambients. This chapter introduces some state of the art studies on various techniques for security policy enforcement in a more generic context.

The term of security policy was introduced in 1982 by Goguen and Meseguer [42] as a response to the need to restrict the rights to read, add, modify, or delete information in specific contexts. Their innovative approach to defining and verifying security involves static and dynamic security policies that stated which information flows are to be denied. The information system is modeled as an automaton called a capability system. The authors also advocated the utility of a security policy definition language based on simple noninterference assertions. A sketch of possible verification models is also presented. Their short article has a surprising density of excellent new ideas and presents a remarkably clear vision of a field that was barely born at the time.

The modern, more practical definition of security policies is given by Schneider [94]:

"A *security policy* defines an execution that, for one reason or another, has been deemed unacceptable. For example, a security policy might concern:

- *access control*, and restrict what operations principals can perform on objects,
- *information flow*, and restrict what principals can infer about objects from observing system behavior, or
- *availability*, and restrict principals from denying others the use of a resource."

2.1 Formal Verification of Security Policies

Verification or validation of security policies is one proactive incident mitigation measure. Routine changes in system operation or slight adjustments of policies can have disastrous consequences in critical environments such as aerospace systems, nuclear plants, traffic control consoles, medical institution infrastructure, and so on. In a society that relies more and more on automation, verification is crucial. Its object is the relation between a system's observed and intended behavior. The formal tools presented in Chapter 1 enable us to specify both. Formal verification can be done on a system specification (through a process algebra, for instance) and a description of the security policy (through logic formulas, for example). The aim of such verification is to prove that the system satisfies the specified properties.

The type of properties to be verified depends on the nature of the program. Termination and correction need to be proven for transformational system. Reactive systems, on the other hand, require proofs for safety and liveness. The verification technique will also differ for the various purposes. Viable solutions include testing, simulation, deductive methods, and model-checking. Testing and simulation take into account a finite number of test cases or scenarios. While the coverage might be broad, it is rarely comprehensive enough and often misses less obvious behaviors or system flaws. In this section we focus on a couple of model-checking techniques involving the semantic tableaux.

Model-checking implies an exhaustive evaluation of all possible paths, meaning that all possible inputs and system states need to be assessed. The staggering number of possibilities may be intimidating, but the assurance given by the result is worth the effort. The approach requires three components: a model to be analyzed, a logic formula to assess against, and a verification algorithm that produces a truth statement about the satisfaction relation between the model and the formula.

A semantic tableau, or simply a tableau, refers to a model-checking technique that analyses all possible interpretations of a logic formula. Handwritten proofs have been produced since the 1950's. Gentzen's sequent calculi and Kripke's explicit accessibility relation [64] were instrumental in the ad-

vancement of the field. Smullyan and Fitting also contributed by proposing a uniform notation for the proofs. Fitting's technique [36] is presented as an alternative to the modal logic proof methods from Fitch [35] and Kripke. His methodology involves explicit tableau rules for every logical construct.

The verification process is straightforward. The exhaustive application of rules, called saturation, leads to a proof tree rooted in an initial statement. A tree branch is considered closed iff both formulas Φ and $\neg\Phi$ are in that branch. If all branches are closed, than the corresponding tableau is closed. If, however, there is at least one branch that is not closed, but no further rule applies, then the tableau is open.

Tableau proofs were further developed and diversified by Fitting [37, 38] and others. Among many noteworthy contributions, we mention Cleaveland's implicit tableau rules [28], Avron's hypersequents [4], the free variable tableaux of Beckert [11], and Lellmann's linear nested sequents [68]. The research area is still very active, with field-specific conferences such as the TABLEAUX International Conference on Automated Reasoning with Analytic Tableaux and Related Methods.

The tableau systems lend themselves easily to automation. Software implementations of the rules, both explicit and implicit, has been successfully attempted by several research groups. Some of the most popular applications are either dedicated tableau frameworks, such as LoTREC [27, 41] and TWB (Tableau Work Bench) [1], or more generic theorem provers such as Isabelle [87, 88].

We review three of the works that influenced our research: Cleaveland and Adi's tableaux systems, and the LoTREC Tableaux Theorem Prover.

2.1.1 Tableau-Based Model Checking in Mu-Calculus

The procedure introduced by Cleaveland in [28] is used for determining whether properties expressed with the propositional mu-calculus hold for given finite-systems. The choice is motivated by the logic's expressiveness and its ability to characterize the behavior of finite-state processes. The syntax of mu-calculus is presented in Table 2.1.

Table 2.1: Syntax of the propositional mu-calculus

$$
\begin{array}{rcl}
\Phi & ::= & A \\
 & | & X \\
 & | & \neg\Phi \\
 & | & \Phi \vee \Psi \\
 & | & \langle \mathsf{a} \rangle \Phi \\
 & | & \nu X.\Phi
\end{array}
$$

The calculus is parameterized with respect to a set $\mathcal{A}$ of atomic formulas, a set *Act* of actions, and a set $\mathcal{V}$ of propositional variables, ranged over by A, a, and X, respectively. The models to be verified against mu-calculus formulas involve sets of process states, transitions and actions. Cleaveland proposes a proof system that operates on sequents of the form $H \vdash s \in \Phi$, where s is a state, H is a set of hypotheses on state s, and Φ is the formula that needs to be satisfied. The tableau rules, with premises below conclusions in typical tableau-based top-down fashion, are presented in Table 2.2.

The eight rules address mu-logic's connectors and operators and their negations. They are employed for building the tableau for a given sequent. The original premise serves as the basis for branches, which are extended by further application of the rules. This creates parent-child pairs and the procedure continues until no more children can be obtained. The last child in each chain is called a leaf and is considered successful if one of four defined conditions are met. If all leaves of a tree are successful, then the tableau is successful. This means that the original sequent is correct, therefore the formula is satisfied by the model.

Cleaveland proves tableau finiteness by showing that for models with a finite set of states, every sequent has a maximum height tableau. He demonstrates that for any given sequent, there are a finite number of distinct tableaux rooted in that sequent. Moreover, proofs of soundness and completeness

Table 2.2: Tableau rules for the propositional mu-calculus

$R1$ $$\boxed{\frac{H \vdash s \in \neg\neg\Phi}{H \vdash s \in \Phi}}$$

$R2$ $$\boxed{\frac{H \vdash s \in \Phi_1 \vee \Phi_2}{H \vdash s \in \Phi_1}}$$

$R3$ $$\boxed{\frac{H \vdash s \in \Phi_1 \vee \Phi_2}{H \vdash s \in \Phi_2}}$$

$R4$ $$\boxed{\frac{H \vdash s \in \neg(\Phi_1 \vee \Phi_2)}{H \vdash s \in \neg\Phi_1, H \vdash s \in \neg\Phi_2}}$$

$R5$ $$\boxed{\frac{H \vdash s \in \langle \mathsf{a} \rangle \Phi}{H \vdash s' \in \Phi}(s' \in \{s \xrightarrow{a} s'\})}$$

$R6$ $$\boxed{\frac{H \vdash s \in \neg\langle \mathsf{a} \rangle \Phi}{H \vdash s_1 \in \neg\Phi, H \vdash s_2 \in \neg\Phi, \ldots}(\{s_1, s_2, \ldots\} \in \{s \xrightarrow{a} s'\})}$$

$R7$ $$\boxed{\frac{H \vdash s \in \nu X.\Phi}{H' \cup \{s \ : \ \nu X.\Phi\} \vdash s \in \Phi[\nu X.\Phi/X]}(s \ : \ \nu X.\Phi \notin H)}$$

$R8$ $$\boxed{\frac{H \vdash s \in \neg\nu X.\Phi}{H' \cup \{s \ : \ \nu X.\Phi\} \vdash s \in \neg\Phi[\nu X.\Phi/X]}(s \ : \ \nu X.\Phi \notin H)}$$

round up an impressive approach to formal verification that inspired some of the work in Chapter 5.

2.1.2 Tableau-Based Proof System for a E-Commerce Protocol

The tableau method is not restricted to modal logics. It may be applied in contexts that involve satisfaction of formulas specified, for instance, in terms of temporal, belief, intuitionistic, and hybrid logics. Once a logic is defined, the tableau rules can be derived and then the model checking can occur.

Adi *et al.* employ the tableau methodology for their ADM logic [2], which has already been reviewed in section 2.3.2 of chapter 1. Their model checking technique is performed on traces of the e-commerce protocol defined in the same article. We remind the reader that a protocol trace is, in the authors' own words, a sequence of protocol events resulting from any interleaving of (possibly partial) protocol runs. The definitions of the other elements of the logic are as presented in section 2.3.2. Local modal checking is used in this case as opposed to global modal checking. This allows visiting only the sub-traces that are required by the tableau, which is significantly more efficient than verifying all possible protocol traces.

The ADM logic's tableau-based proof system is shown in Table 2.3. The tableau rules $R_{\neg}, R_{\wedge}, R_{\nu}$, and $R_{[]}$ verify whether a trace t satisfies a formula Φ or not. The construction of the tableau is different than Cleaveland's in several ways. Firstly, the sequents have a richer syntax, which allows for a more compact, but equally expressive set of rules. Secondly, the rules' premise and conclusion are presented in reversed positions, with the premise at the bottom and the conclusion on top.

The rules' sequents are of the form $H, b, e, \sigma \ \vdash t \in \Phi$, where the trace t ranges over $\mathcal{T}$, H is a mapping in $[\mathcal{V} \rightarrow 2^{\mathcal{T}}]$, b is a variable ranging over $\{\varepsilon, \neg\}$, e is an environment, and σ is a substitution. The flag b is particularly useful, as it alleviates the need for negative forms of the rules. Tableaux are derived by applying rules to sequents and obtaining new formula-labelled nodes until saturation, and the resulting leaves are assessed for success. If

Table 2.3: Tableau proof system for the ADM logic

$R_{\neg}$	$\dfrac{H, b, e, \sigma \vdash t \in \neg\Phi}{H, \neg b, e, \sigma \vdash t \in \Phi}$	
$R_{\wedge}$	$\dfrac{H, b, e, \sigma \vdash t \in (\Phi_1 \wedge \Phi_2)}{H, b_1, e, \sigma \vdash t \in \Phi_1 \quad H, b_2, e, \sigma \vdash t \in \Phi_2}$	$b_1 \times b_2 = b$
R_{ν}	$\dfrac{H, b, e, \sigma \vdash t \in \nu X.\Phi}{H[H \mapsto H(X) \cup \{t\}], b, e, \sigma \vdash t \in \Phi[\nu X.\Phi/X]}$	$T \notin H(X)$
$R_{[]}$	$\dfrac{H, b, e, \sigma \vdash t \in [p_1 \looparrowright p_2]\Phi}{\theta_1 \ldots \theta_n}$	C
where :		
θ_i	$= H, b_i, e, \sigma_i \circ \sigma \vdash p_2\sigma\sigma_i \in \Phi,$	$i \in \{1, \ldots, n\}$
and		
C	$\left(\begin{array}{c} \{\sigma_1, \ldots, \sigma_n\} = \{\sigma' \mid p_1\sigma\sigma' = t\} \neq \emptyset \\ \text{and} \\ b_1 \times \ldots \times b_n = b, n > 0 \end{array}\right)$	

all leaves are successful, then the tableau is successful.

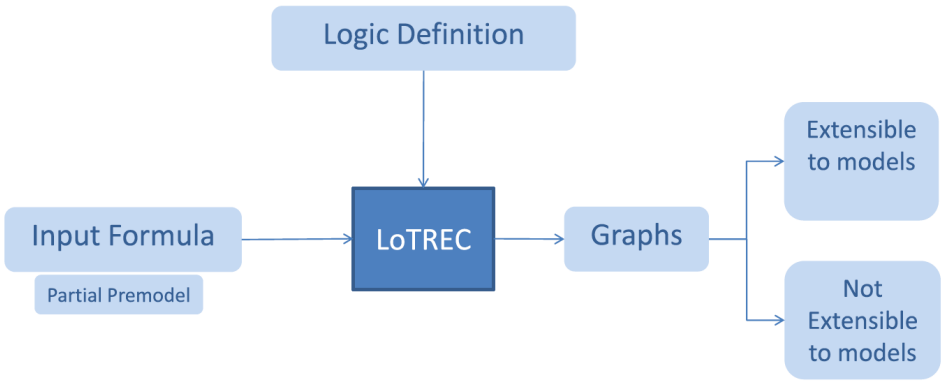
Adi *et al.* formally prove the finiteness, soundness and completeness of their tableau-based system. These essential properties demonstrate that the methodology produces finite models that are suitable for verification against ADM logic formulas.

2.1.3 *LoTREC* Tableaux Theorem Prover

Theorem proving is a deductive technique that relies on inferences or rewriting to automatically generate proofs and verify their correctness. *LoTREC* [27, 41] is an automated theorem prover developed by researchers at IRIT

(Institut de Recherche en Informatique de Toulouse). The theoretical basis of the framework have been laid out by L. Farias del Cerro, O. Gasquet, and A. Herzig in the late 1990s. Significant features have been added by D. Fathoux, M. Sahade, and B. Saïd. The improvements include, among others, model checking and a host of predefined logics, along with their completeness and termination proofs. Tableaux proofs can also be built for user-defined logics. The basic operating principles of *LoTREC* are depicted in Figure 2.1 below.

Figure 2.1: LoTREC's black box



The syntax allows for atomic propositions and formulas involving logical connectives such as not, or, and, always, etc., for which the arity and priority must be specified. The priority corresponds to the order of precedence for the connectives, with 0 being the highest. The precedence order of some classical connectives is: $\neg > \wedge > \vee > \rightarrow > \oplus$. The order can, of course, be different for user-defined logics. For example, the formula $A \wedge B$ can be written as "or variable A variable B", will have an arity of 2, a priority of 1, and will be displayed by LoTREC as the label "A & B".

Rules for a user-defined logic take into account truth conditions and structural constraints. A tableau for a given sequent can then be drawn based on the ruleset applicable. The rules can either modify the graph by adding nodes (i.e. structural rules) or can alter formulae used for node labelling (i.e. propagation rules). They are applied to every formula in every node, which helps expediting the proofs. The graphs are expanded by adding or removing nodes, links, formulas, or by duplicating the graph. There is a restricted set of predefined constructs that can be employed for rule writing. The formal semantics of LoTREC ensure that correctly specified rules always produce "correct" results.

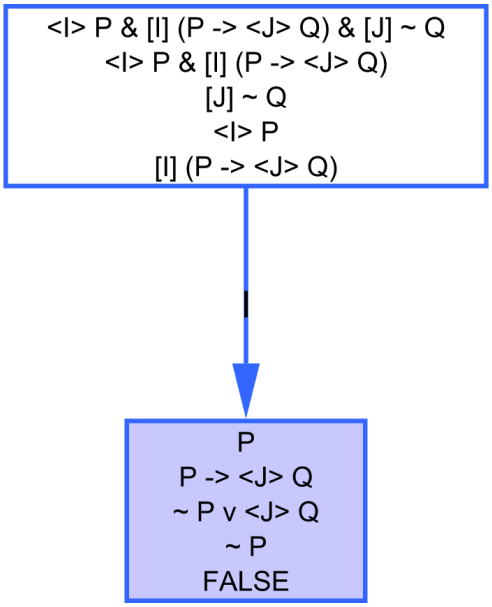
Rules are consolidated in strategies, which define what rules are applied, in which order, and when does the strategy stop. Depending on how the strategy is built, a rule can be applied once, several times, or never. This is achieved by means of rule groupings called blocks and predefined combinators (repeat, allRules, firstRule) in conjunction with end, which prompts for the conclusion of the strategy's execution. A repeatable block of rules may however be sufficient for simple logics. Multiple strategies may be built for the same logic either for efficiency reasons or for demonstrating specific purposes. However, note that termination, soundness, and completeness must be proven for each strategy.

The tableau successfulness is evaluated as in the previous sections, and the truth assessment can be marked in the associated graph by the TRUE or FALSE labels. The following figure shows a relatively simple LoTREC graph.

2.2 Formal Enforcement of Security Policies

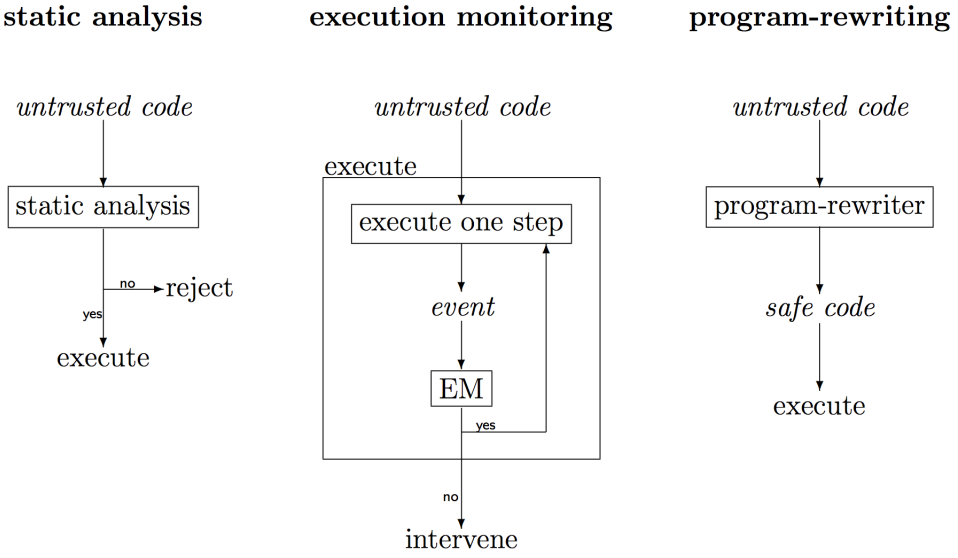
Stating the expected behavior of the system is a necessary requirement. However, the simple enunciation of a policy is not very useful in the absence of an enforcing mechanism. Different approaches for implementing or guaranteeing security policies have been proposed over the years. The enforcement can be applied at various levels of the computing infrastructure, and can support policies ranging from a single security property (e.g. authentication, confidentiality, integrity) to a complete set of system constraints.

Figure 2.2: Sample LoTREC graph



There are multiple manners of enforcing security policies, as depicted in Fig. 2.3. The system can be subjected to an assessment of the underlying code prior to execution. This technique is called static analysis and can be applied, for instance, through model checking or in languages that permit type-checking. Dynamic analysis employs reference monitors that constantly watch untrusted programs for suspicious or non-compliant behavior. Program rewriting is another enforcement mechanism that modifies an untrusted program prior to the execution to prevent policy violations. The remainder of this section is dedicated to the most notable approaches to computer security policy enforcement.

Figure 2.3: Security policy enforcement methods



2.2.1 Static Analysis

Portions of a programs that do not change during execution can be statically analyzed for compliance with security policies. Running an assessment at this stage has several advantages. Static analysis is decidable, meaning in this context that it would provide an answer with certainty, in finite time, about whether the system satisfies a security policy or not. There is no time contraint, since the analysis can be planned and performed prior to the actual execution. Also, there is no impact on the performance of the program. No cycles are wasted by running execution monitors, for instance, to check whether the behavior is acceptable or not. Finally, static analysis can complement, rather than exclude, the use of other enforcement methods, as it has the ability to evaluate policies that other methods can't. This last aspect will become more evident in the next section where some light is

shed onto the unsatisfiable policy. The are several techniques for performing a static analysis of a system against a given policy. Model checking and automatic theorem provers have been covered in the previous section. We briefly introduce here two other interesting approaches to static analysis: proof-carrying code and type systems.

2.2.1.1 Proof-Carrying Code

Necula [80] devised a software mechanism that allows to determine whether code from an untrusted source can be executed on a target system. The author uses the terms "code producer" and "code consumer" to identify the party that compiles and certifies the code and the beneficiary of the formally validated code, respectively. The mechanism, called proof-carrying code, is illustrated in Figure 2.4. The principle is simple, yet powerful. The producer's code implements security properties requested by the consumer and, in addition, carries a certificate of compliance. The consumer verifies that the certificate is acceptable, pertinent, and actually matches the producer's code. If all three conditions are met, the code is run with assurance that the security properties are satisfied.

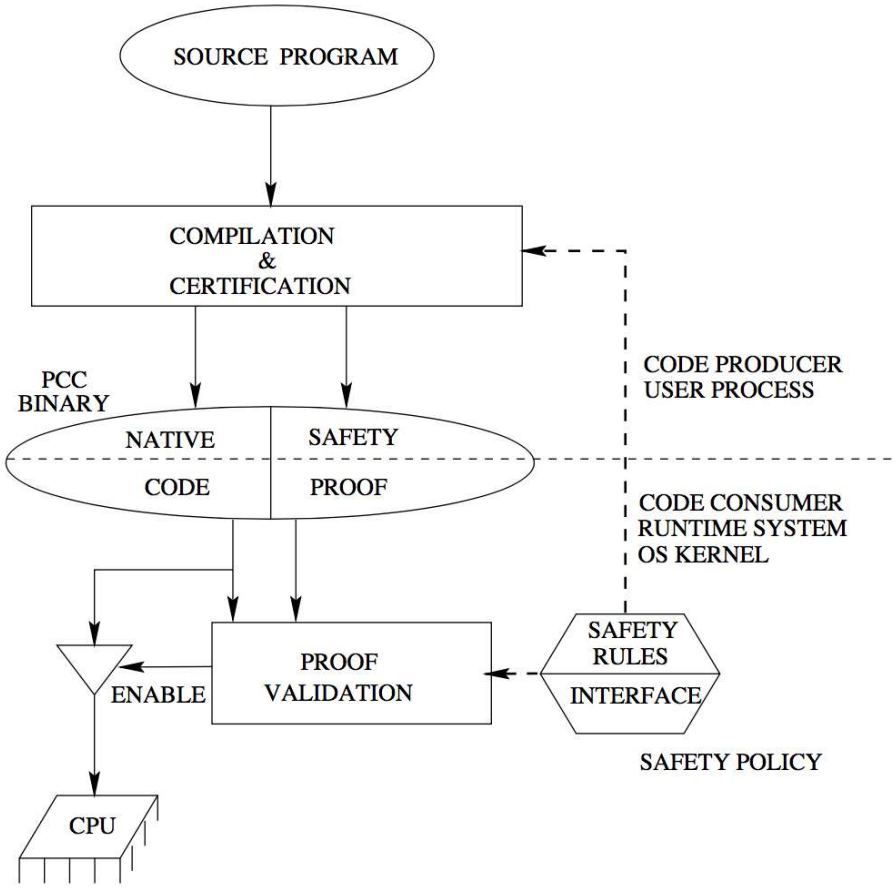
PCC is well fitted for enforcing safety policies. It features typical advantages of static analysis approaches:

- decidability: verification completes in finite time, and it give a clear answer (either safe or not safe to execute);
- no performance impact: since it operates at load time, the execution is not slowed down by runtime validation;
- complements other approaches, such as cryptographic techniques (digital signatures can be used for freshness and authenticity, rather than safety).

However, the original PCC system need to use a verification-condition generator (VCgen) on both the producer and the consumer systems. The VCgen employs typing rules and is quite substantial (over 20,000 lines of code). Moreover, it has to be invoked for each program and each new version of

the program. The Foundational PCC, a type-independent variant of PCC, sidesteps the necessity of the VCgen. It employs the more-expressive higher-order logic, allowing the producer to "explain" safety arguments, but requires machine-checked proofs generated with the help of typed assembly language.

Figure 2.4: Proof-carrying code



2.2.1.2 Type Systems

Sablefeld and Myers [90] offered a solution for ensuring end-to-end information flow confidentiality through programming language techniques. Their method protects the information against an attack through observations of the system's output. The security-typed language they propose uses policy-specifying annotations for the types of program variables and expressions. The associated compile-time security-type checking adds almost no overhead and is compositional: a system composed of secure subsystems is inherently secure. Nevertheless, the solution does not allow verifying policy compliance and no policy manager errors introduced at system's inception can be detected.

Another security policy-enforcing type system was presented by Gorla and Pugliese [45]. Their system addresses issues related to code mobility in distributed systems such as e-commerce and online banking. The type is expressly designed for the process calculus μ-Klaim [44], a calculus of distributed and mobile processes. The syntax of μ-Klaim is presented in Fig. 2.4. It allows the assignment of different privileges to users over different resources, which mimics the behavior of a real-life system. This makes the system practical, as demonstrated by their bank account management system model. In contrast to Sablefeld and Myers' system [90], μ-Klaim also allows dynamic type checking along with the static one. The approach is interesting and a variant adapted to our calculus could be used to complement our framework.

Martins [72] introduced a different type system applicable to complex networks in need of better security. He uses Dπ [50, 53] to control the migration of code between the nodes of large distributed systems. Resource policies are expressed with types and enforced via a type system. The network topology is reflected by shared security policies within sites. The result is a system that is free of any policy violation at runtime.

The framework presented by Miksad in [75] has some common traits with the approach of this thesis. This is to be expected, as both involve formal means of verifying security policies. The verification process requires policy specification, system specification, and policy verification components. The

Table 2.4: μ-KLAIM syntax

$N ::=$	0	(empty net)
	$l ::^{\triangle} P$	(single node)
	$N_1 \parallel N_2$	(net composition)
$P ::=$	nil	(null process)
	$a.P$	(action prefixing)
	$P_1 \mid P_2$	(parallel composition)
	A	(process invocation)
$e ::=$	$V \quad x \quad \ldots$	(expressions)
$a ::=$	$\text{read}(T)@\, l$	(process actions)
	$\text{in}(T)@\, l$	
	$\text{out}(t)@\, l$	
	$\text{eval}(P)@\, l$	
	$\text{newloc}(u : \triangle)$	
$T ::=$	$F \quad F, T$	(templates)
$F ::=$	$f \quad !x \quad !u : \pi$	(template fields)
$t ::=$	$f \quad f, t$	(tuples)
$f ::=$	$e \quad l$	(tuple fields)

main purpose of Peri's study (security policy enforcement) and the framework functionality (two specification pieces and a verification part) play a major role in our framework, too. However, his approach differs greatly from ours. Peri uses Z [32] for system specification, a temporal logic based language for security policies specification, and a trace based system for policy verification.

2.2.2 Execution Monitoring

Schneider [94] initiated the research on the class of runtime-enforceable security policies. He precisely characterized the EM (Execution Monitoring) class of enforcement mechanisms and specified it with security automata. A security automaton (named *recognizer* by Schneider and Alpern in their previous work) is defined by:

- a countable set Q of automaton states,
- a countable set $Q_0 \subseteq Q$ of initial automaton states,
- a countable set I of input symbols, and
- a transition function $\delta : (Q \times I) \rightarrow 2^Q$.

The automaton starts in state Q_0 and reads one symbol s_t at a time from a list of input symbols $s_1 s_2 \ldots$. The symbols correspond to security policy requirements (actions or system states). The current state Q' of the automaton evolves to:

$$\bigcup_{q \in Q'} \delta(q, s_t)$$

A security automaton is used for modeling a particular policy. The automaton contains two sections: one for the declaration of state variables and one for the definition of allowed transitions. Security policy enforcements in EM are accomplished through security automata simulations, which are executed in tandem with the target. The simulations are fed with input symbols generated by each step that the target is about to take. The state of the automaton changes if it can make the transition on the input symbol, otherwise

the simulation ends and the target is terminated. In terms of security policy enforcement, the security automaton grants the target the right to perform the step if the transition is possible, or else it denies execution because of a policy violation.

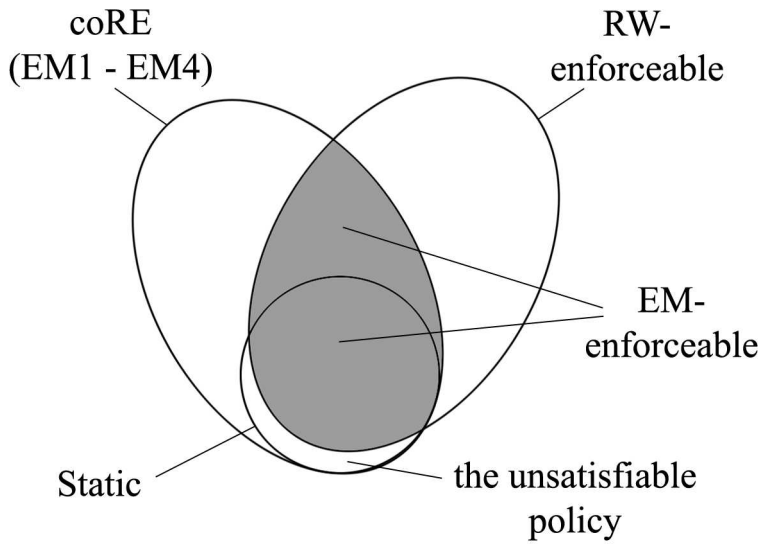
The EM class includes security kernels, reference monitors, firewalls, and most other enforcement mechanisms, but excludes compilers and theorem provers. Since a security policy is specified by giving a predicate on sets of executions, the monitored execution steps vary from single actions to complex configuration changes.

Schneider *et al.* [49] enhanced the result in [94] by defining a new class of policies enforceable by program rewriting. In this context, program rewriting refers to any enforcement mechanism that, in a finite time, modifies an untrusted program prior to execution. The RW class enables some policies that are not enforceable through program monitoring. The use of program rewriting for enforcing security policies originated in the late 1960's, but it has been revived in recent years. Schneider and Erlingson's analysis of software-based fault isolation (SFI) led to the development of SASI (Security Automata SFI Implementation) [103]. SASI enforces security policies by modifying object code for a target system before that system is executed. The approach has been prototyped for two architectures: Intel's x86 and SUN's JVM (Java Virtual Machine). The prototypes explored the use of an SFI-like approach for EM-enforceable policies.

The taxonomy of enforceable security policies described in [49] is depicted in figure Fig. 2.5. This new approach corrects some subtle flaws in Schneider's previous work [94] by eliminating the group of EM class policies that cannot be implemented. A formal model of security enforcement is built and used for characterizing three enforcement methods: static analysis, execution monitoring, and program rewriting. The untrusted programs are modeled as a particular form of a Turing Machine. They need to be prevented from executing steps that would violate a given security policy through one of the enforcement methods. The coRE (co-recursively enumerable) subclass of EM policies illustrated in Fig. 2.5 represents the policies that satisfy a set of constraints (EM1-EM4) and can determine, at any step of the program's execution, an appropriate policy violation detector. The authors of the study formally demonstrate that the statically enforceable policies are a

subset of the coRE class. They are also a subset of the RW class, except for the unsatisfiable policy. The EM-enforceable class is characterized by policies that are at the same time coRE and $RW_{\approx}$-enforceable. Finally, there are EM policies that cannot be enforced through program rewriting, as well as RW-enforceable that cannot be enforced through execution monitoring.

Figure 2.5: A taxonomy of security policies



The runtime enforcement of policies through automata enjoys some popularity among many other researchers. Bauer, Ligatti and Walker [8, 9, 10] issued a series of studies on the subject. Their enforcement technique relies on security automata working as program monitors that examine and, if needed, transform the sequence of program actions. The syntax of security monitors is presented in Table 2.5. The automata enforce a certain class of Schneider's security policies discussed in [49] by inserting or suppressing actions.

A new language and system that supports the development of easy-to-maintain,

but complex runtime security policies for Java applications is also introduced. Policies are first-class objects and consist of two types of methods: one for security-sensitive actions and one for state updates. Complex systems can be modeled by means of parameterized meta policies [10] or higher-order policies [9]. A library of powerful policy combinators is supplied, demonstrating the use of programming features that permit policies composition. A formal semantics has been defined for a subset of the language consisting of the key features only. The system is fully implemented in a large-scale security policy for an email client.

Table 2.5: Security monitors syntax

(types)	$\tau ::= \text{Bool} \mid (\overrightarrow{\tau}) \mid \tau\ \text{Ref} \mid \tau_1 \to \tau_2 \mid \text{Poly} \mid \text{Sug} \mid \text{Act} \mid \text{Res}$
(programs)	$P ::= (\overrightarrow{F}, M, e_{\text{pol}}, e_{\text{app}})$
(monitored functions)	$F ::= \text{fun} f(x : \tau_1) : \tau_2\{e\}$
(memories)	$M ::= \cdot \mid M, l : v$
(values)	$v ::= \text{true} \mid \text{false}$
(expressions)	$e ::= v \mid x \mid (\overrightarrow{e}) \mid e_1; e_2$
(patterns)	$p ::= x \mid \text{true} \mid \text{false}$

One of the newest problems in distributed environments is the dynamic of

its mobile components. On-demand computing in academic and industrial grids and the outbreak of laptops and mobile devices are just two examples of challenging security issues. Orlovsky [83] analyzed security policies in the context of distributed environments. Global and local security policies have been specifically defined for this reason. The author proposes a security mechanism based on smart sandboxes (which are multiple instances of execution monitors) that can share security information. The decentralized mechanism is efficient and scalable. Global information flow policies are an example of a subclass of EM policies enforceable by this mechanism.

2.2.3 Program Rewriting

Program rewriting is another interesting approach, well worth investigating since it complements other methodologies. Irrespective of the fact that a given program (or application) satisfies or not a certain security property or policy, the application is rewritten to ensure that the policy is satisfied. The resulting rewritten program has to be equivalent with the original one in the sense that no new behavior can be added, and only those behaviors in conflict with the security policy are removed. Several approaches to program rewriting and an enforcement technique are discussed in this section.

Edit Automata and Algebraic Approaches

Rewriting can be done, for instance, by automata injection, as shown by Ould-Slimane *et al.* in [84]. The authors turn to edit automata for enforcing security properties through a formal rewriting technique. Edit automata have the capability of feigning the execution of sensitive actions, which is an advantage over the conventional Execution Monitoring (EM) security mechanisms. Feigning refers to an undetectable blocking of the execution of a program actions. However, classic EMs are not always reliable in the sense that they cannot feign all actions. Ould-Slimane's framework uses Extended Finite State Machines (EFSMs) for representing a program and injecting an edit automaton that targets enforced security properties. The authors demonstrate that the program operates in a transparent manner. Moreover, the injection produces a version of the program that is formally proven to

be secure, meaning that all observable outputs respect the intended security property.

Enforcement of security policies on concurrent programs can also be achieved by transformation of formulas and processes, as demontrated by Langar *et al* in [66, 73, 67]. The authors transform the security policy into a process that is executed in parallel with the target system. The original program is rewritten to allow only control and approval of its actions by an execution monitor that implements the enforced security policy. The language used for the specification of security policies, shown in Table 2.6 and denoted by $BPA^*_{\delta,1}$, is an extension of BPA (Basic Process Algebra).

Table 2.6: Syntax of $BPA^*_{\delta,1}$

$$P_1, P_2 ::= \delta \mid 1 \mid a \mid P_1.P_2 \mid P_1 + P_2 \mid {P_1}^* P_2$$

The algebraic framework for the specification of the concurrent systems is provided by ACP^ϕ, displayed in Table 2.7. ACP^ϕ is an extended version of ACP (Algebra for Communicating Process). The meaning of most operators names in Table 2.7 is obvious. The three merge operators correspond to different interaction manners that model synchronization, execution priority, and communication. Encapsulation imposes restrictions on what actions can be executed. The process $\partial_H(P)$ can only evolve if its actions are not part of the set H. Abstraction replaces all actions of the process $\tau_I(P)$ from the set I, called internal actions, by the silent action τ. Finally, enforcement allows a process to evolve only if its actions comply with the security policy.

Langar defines a Kleen algebras-inspired L_φ logic that allows the specification of security properties. The semantics of L_φ is presented in Table 2.8. The operator $\otimes$ is used for defining enforcements. The composition $P \otimes \phi$ of program P and a security policy ϕ is an enforced program denoted by $\delta^\xi_\phi(P)$. The secure program preserves only those traces of the original program P that do not violate the security policy ϕ.

Table 2.7: Syntax of ACP^{ϕ}

$P ::=$	1	(Constant representing successful termination)
	$\mid \delta$	(Constant representing deadlock)
	$\mid a$	(Atomic action)
	$\mid P_1.P_2$	(Sequential composition)
	$\mid P_1 + P_2$	(Non $-$ deterministic choice or sum)
	$\mid P_1 \mid\mid_{\gamma} P_2$	(Merge, parallel composition)
	$\mid P_1 \mid\mid\lfloor P_2$	(left merge)
	$\mid P_1 \mid_{\gamma} P_2$	(Communication merge)
	$\mid P_1{}^* P_2$	(Iteration operator)
	$\mid \delta_H(P)$	(Encapsulation operator, H $\subseteq$ A)
	$\mid \tau_I(P)$	(Abstraction operator, I $\subseteq$ A)
	$\mid \partial_{\phi}^{\xi}(P)$	(Enforcement operator)

Table 2.8: Syntax of L_{φ}

$$\varphi_1, \varphi_2 ::= tt \mid ff \mid 1 \mid a \mid \varphi_1.\varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi_1 \mid \varphi_1{}^*\varphi_2$$

This is achieved by applying the following two transformations. Firstly, each action a of a policy ϕ is replaced by a sequence $\overline{a_d}.\overline{a_f}$ that marks start and end of the action. Consequently, the L_φ policy specification is transformed into a synchronization ACP^ϕ process by a specialized function. Synchronization actions allow the controlled program to evolve when the actions it intends to execute are allowed by the formal monitor. Secondly, each action a of the synchronization process is transformed into $\overline{a_d}.a.\overline{a_f}$ by another specialized function. The properties are then checked against infinite behavior through the formal monitor. An optimization technique reduces the number of required tests for a certain class of security properties. The innovative approach can be applied to commonly used programming languages like Java.

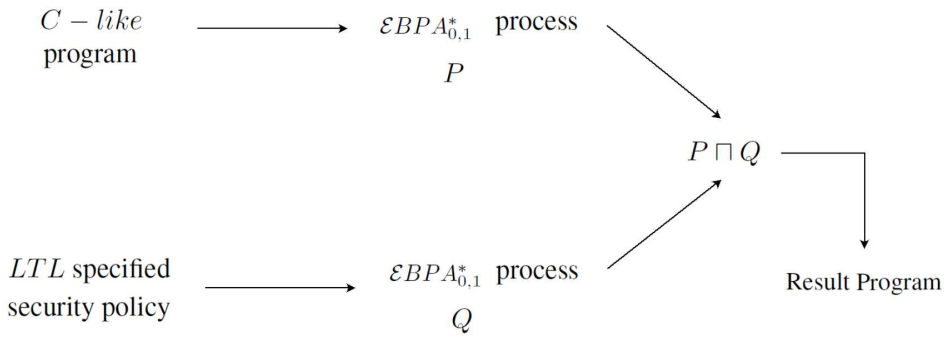
Sui *et al.* carry forward the work of Langar in [101]. They modify $BPA^*_{\delta,1}$ slightly to obtain $BPA^*_{0,1}$ and eventually extend it into the new algebra $\varepsilon BPA^*_{0,1}$, which adds syntactic constructs for variables, conditions and environment. Since BPA is not well suited for addressing concurrent programs, their rewriting approach targets for enforcement only some critical points in the program. The approach is presented in Figure 2.6.

The program P is expressed in $BPA^*_{0,1}$ either directly or by transformation from its C-like language form. The security policy Φ to be enforced is normally given as a νLTL (Linear Temporal Logic) formula and will also be translated into a $BPA^*_{0,1}$ formula. The policy Φ is enforced on the program P by means of the $\sqcap$ operator. The notation $P \sqcap Q$, called the greatest common factor (gcf) of P and Q, is a process R such that all traces of R are both in P and Q, and for any other process R' with all traces in P and in Q, $R' \sqsubseteq R$. Security policy enforcement becomes in this case a problem of computing the gcf of a process P representing the system and a process Q representing the policy.

The enforcement operation consists in solving the set of $BPA^*_{0,1}$ equations obtained by applying $\sqcap$. In order to convey the notion of trace equivalence, the authors use an adapted definition of Brzozowski's derivates, denoted by $\partial_a(x)$, and a termination function $o(x)$ to know whether a process terminates immediately or not. The computation algorithm is displayed in Figure 2.7.

The program obtained is sound, and therefore guarantees that all traces of

Figure 2.6: Approach to formal and automatic security policy enforcement



the secured process satisfy the security policy. Moreover, the enforced program is complete and preserves all compliant traces of the original program. The expressiveness of the $BPA^*_{0,1}$ algebra makes it applicable to some scripting languages such as PHP and Perl, and to a simplified version of C-like programs. A combination with approaches like Langar's may also permit better enforcement of concurrent programs. A web application prototype called FASER (Formal and Automatic Security Enforcement and Rewriting) implements the methodology proposed. The theoretical foundations for the Java and JSP environment FASER are introduced in [100].

Towards Enforcement through Interface Equations

Norris [79] was the first to use the expression "interface equation", in 1985, while Shields [95, 96, 97] strongly promoted it and helped firmly establishing the term. It was considered a potential solution to the problem of general system synthesis as it encapsulates the general problem of concurrently operating systems which have to communicate via some interface. In our context, an interface equation can be formulated as follows: for two processes P and Q and a given equivalence relation denoted by $\sim$, we need to find another

Figure 2.7: Algorithm for formal security policy enforcement

Algorithm 1 calculate $P \sqcap Q$ in $BPA^*_{0,1}$ where $E = \emptyset$

1: $E \longleftarrow E \cup \{P \sqcap Q \sim o(P) \times o(Q) + \sum_{a \in \delta(P) \cap \delta(Q)} a.(\partial_a(P) \sqcap \partial_a(Q))\}$

2: **while** there exists $P_i \sqcap Q_i$ in the right side of any equation in E that does not appear (modulo commutativity of $\sqcap$) in the left side on any equation **do**

$E \longleftarrow E \cup \{P_i \sqcap Q_i \sim o(P_i) \times o(Q_i) + \sum_{a \in \delta(P_i) \cap \delta(Q_i)} a.(\partial_a(P_i) \sqcap \partial_a(Q_i))\}$

end while

3: Return the solution of the linear system E.

process X such that P | X (read "P combined to X using a given operator |") is equivalent to Q:

$$\mathsf{P} \mid \mathsf{X} \sim \mathsf{Q}$$

where, P and Q cannot communicate.

The success of the equation as a specification generator is due to two main research groups. The first one, composed of Cerny, Marin and Zafiropulo *et al.*, focused on the applicability of this theory to logic circuits and protocol generation. The second group, lead by Merlin and Bochmann, formulated the interface equation in the modern shape known today. Cerny and Marin [26] used Boolean equations for fault detection in the case of logic circuits. Their methodology consisted in the deduction of all the receive actions when all the send actions are known, along with a certain set of properties about

the searched module.

Zafiropulo [17, 105] and his collaborators concentrated more on protocols generation, while Merlin and Bochmann [60, 74, 89] focused on generating specifications. Ince [30, 57] addressed the problem of protocol converters by blending together automata theory and topology. His technique employs topological graphs for deriving quotient graphs as solutions to the interface equation. Khedri [62] attempted a general formulation of the equation in the case of parallel processes.

The relatively small number of techniques for solving the equation proves the complexity of the issue and shows that the most important factor is a proper definition of the problem. We briefly present two methods: the specifications generators of Merlin and Bochmann and the relational processes approach of Khedri.

The research work of Merlin, Bochmann [74], and their followers in the field of protocol specification covers more than two decades. The extraction of protocol specifications from a given service specification [60] seems to enjoy more popularity than the other approaches. The service primitives have to be executed in a particular order using operators for sequential, parallel and alternative executions. The proper ordering is ensured through synchronization messages between the derived protocol entities. The derivation algorithm, which is elegant and efficient, allows for finite and infinite behaviors and results in optimal protocol specifications.

A comprehensive review of the protocol synthesis methods was given by Probert in [89]. The eleven methods presented, including those of Bochmann, are classified and compared based on their features: synchronous communication, interactivity, semantic correctness, error-recovery patterns, etc. The protocol synthesis methodology inspired some of our less successful attempts in our own quest for the proper interface equation solution. However, the approach soon became of limited interest to us in the specification area, as we made the choice to employ a calculus and a logic for the purpose of system and policy specification.

A general formulation of the interface equation has been provided by Khedri in [62]. He analyzes process modeling, concurrency, integration, simulation

and L-bisimulation. The use of relational formalisms helps solving the challenging problem. In his work, the interface equation for the interlaced parallel composition of processes is denoted by $\mathsf{P} \wr \mathsf{X} \approx_\sigma \mathsf{Q}$, where P, Q and X are processes, $\wr$ is a parallel composition operator and σ is a known relation between the processes.

A slightly different shape of the equation, devised for the totally synchronous parallel composition of processes, involves another composition operator ($\Downarrow\Downarrow$) and presents a different solution. The unknown process X can designate a controller, a protocol converter or a specification. Processes are represented by a mathematical entity called a relational process which is denoted by the quintuple $((J, K \ : \ J, K \in \mathcal{U} \ : \ P_{JK}),\ \alpha_P,\ E_P,\ \omega_P,\ F_P)$. The first component represents the resources of a process and its actions. The other four represent the input and output relations (α_P and ω_P) and the variables associated with them (E_P and F_P), respectively.

The interface equation can take various forms: protocol conversion, controller generation, reutilization, etc.. Khedri manages to gather them all as a set of solutions in a comprehensive framework. The benefit of having such a set available is that one can easily choose the appropriate solution to their application domain. It is possible that one solution fits a certain purpose, but not another. For instance, for the same interface equation, a solution can work very well in a reutilization context with constraints to use specific hardware components, while it fails to be useful in a protocol conversion context. However, the framework is not particularly well suited for the broad scope of security policies.

Interface equations provide a calculation technique for applicable enforcements. Once the system and the security policy have been specified, we want to extract automatically an enforcement process that imposes the behavior stipulated by the policy. The main intent of this approach is to automatically identify the components that enforce a given policy on a system. In Chapter 6, we state the problem in terms of interface equations as defined in [62, 85, 97] and use a process algebra to solve it.

2.3 Security Policy Enforcement with Ambients and Related Calculi

Several research activities have used ambient calculus to address a number of issues related to access control and information flow policies. The solutions came in the form of information and control flow analysis, ambient guardians, safe ambients, and specification calculi. We present in this section the representative works in these areas.

2.3.1 Control Flow Analysis

This approach employs the ambient calculus as a base for an information flow analysis. The defining factor of the investigation is given by mobility aspect of ambients. Policies are used for determining which flows are permitted, therefore establishing in a formal manner if a certain property of the flow is maintained. Two such analysis are presented: the information flow analysis of Cortesi and Focardi [29] and the firewall validation of Nielson *et al.* [82]. The same technique, but applied to the safe ambients, has been used by Degano *et al.* [31] and is presented in the next section.

Information Flow Analysis

Cortesi and Focardi [29] focus on a particular case of Mandatory Access Control in the context of the classical ambient calculus. The policy in case, called Multilevel Security, requires that every entity is bound to a security level. Only two levels are considered, for simplicity (high and low), and information may just flow from the low level to the high one. Two access rules are imposed: No Read Up (a low level entity cannot access information of a high level entity) and No Write Down (a high level entity cannot leak information to a low level entity). The scenario considered by the authors involves moving agents that could potentially inadvertently leak confidential data. The agents are modeled in the context of mobile ambients. The information flow property is defined in terms of the possibility to move a confidential ambient outside a security boundary through covert channels.

The analysis is defined by the representation function β_l^{CF} and the specification $(\hat{I}, \hat{H})$. Information about all nesting ambients yielded by a process in its initial state is collected by the process representation function defined in Table 2.9. Labels $l^a \in \mathbf{Lab}^a$ on ambients and $l^t \in \mathbf{Lab}^t$ on transitions indicate program points, and l is either a capability or an ambient inside l^a.

Table 2.9: Representation function for Control Flow Analysis

(res)	$\beta_l^{CF}((\nu n)P)$	$= \beta_l^{CF}(P)$
$(zero)$	$\beta_l^{CF}(\mathbf{0})$	$= (\emptyset, \emptyset)$
(par)	$\beta_l^{CF}(P \mid Q)$	$= \beta_l^{CF}(P) \sqcup \beta_l^{CF}(Q)$
$(repl)$	$\beta_l^{CF}(!P)$	$= \beta_l^{CF}(P)$
(amb)	$\beta_l^{CF}(n^{l^a}[P])$	$= \beta_{l^a}^{CF}(P) \sqcup (\{(l, l^a)\}, \{(l^a, n)\})$
(in)	$\beta_l^{CF}(in^{l^t} n.P)$	$= \beta_l^{CF}(P) \sqcup (\{(l, l^t)\}, \emptyset)$
(out)	$\beta_l^{CF}(out^{l^t} n.P)$	$= \beta_l^{CF}(P) \sqcup (\{(l, l^t)\}, \emptyset)$
$(open)$	$\beta_l^{CF}(open^{l^t} n.P)$	$= \beta_l^{CF}(P) \sqcup (\{(l, l^t)\}, \emptyset)$

The specification recursively checks subprocesses for the pair $(\hat{I}, \hat{H})$, as described in Table 2.10. $\hat{I}$ determines which pairs ambient-subambient or ambient-capability belong to the flow. $\hat{H}$ handles the label-name correspondence: if a process contains an ambient labeled l^a with name n, then (l^a, n) is expected to belong to $\hat{H}$. Special labels are assigned to the *in* and *out* capabilities of the ambients containing sensitive data, called border ambients. The verification process consist in proving that a very simple syntactic property (i.e. a labeled capability) is sufficient to guarantee the absence of unwanted information flows.

The analysis becomes very complex and cumbersome in the case of more security levels or a combination of security properties, which makes it impractical for the applications we considered.

Firewall Validation

In [82], Nielson *et al.* used ambient calculus to specify a firewall and agents that attempt to cross it. The firewall model is taken from Cardelli and Gordon's original article on mobile ambients. The premise is that a properly specified firewall should not permit entry to attackers lacking the required passwords.

Four predicates are defined for the acceptability of the analysis:

$(I,H) \models^{l}_{me} P$	for checking a process $P \in \mathbf{Proc}$
$(I,H) \trianglerighteq_{me} M : \tilde{M}$	for translating a capability $\mathrm{M} \in \mathbf{Cap}$ into a set $\tilde{\mathrm{M}} \in \mathcal{P}(\mathbf{SCap})$ of stable capabilities;
$(I,H) \Vdash_{me} N : \tilde{N}$	for decoding a capability $\mathrm{N} \in \mathbf{Nam}$ into a set $\tilde{\mathrm{N}} \in \mathcal{P}(\mathbf{SNam})$ of stable names;
$(I,H) \Vdash^{l} \tilde{m}$	for checking a stable capability $\tilde{\mathrm{m}} \in \mathbf{SCap}$

A polynomial time algorithm is used for rejecting the firewalls that fail to provide adequate protection. The control flow analysis of firewall validation is presented in Table 2.11, which contains separate sections for the four acceptability predicates. The algorithm is based on control flow analysis of processes that are capable of crossing other ambients' boundaries. The verification process examines all possible reduction sequences in order to determine that an agent holding the appropriate keys is allowed to enter a site protected by a firewall. The operations permitted inside each site and the actions allowed for each process are determined as part of the checking technique.

There are, in our opinion, several inconveniences with this approach. The original ambient firewall model of Cardelli and Gordon, which is used by

Table 2.10: Specification of Control Flow Analysis

(res) $(\hat{I},\hat{H}) \models^{CF} (\nu n)P$ iff $(\hat{I},\hat{H}) \models^{CF} P$

$(zero)$ $(\hat{I},\hat{H}) \models^{CF} \mathbf{0}$ always

(par) $(\hat{I},\hat{H}) \models^{CF} P \mid Q$ iff $(\hat{I},\hat{H}) \models^{CF} P \ \wedge \ (\hat{I},\hat{H}) \models^{CF} Q$

$(repl)$ $(\hat{I},\hat{H}) \models^{CF} !P$ iff $(\hat{I},\hat{H}) \models^{CF} P$

(amb) $(\hat{I},\hat{H}) \models^{CF} n^{l^a}[P]$ iff $(\hat{I},\hat{H}) \models^{CF} P$

(in) $(\hat{I},\hat{H}) \models^{CF} in^{l^t} n.P$ iff $(\hat{I},\hat{H}) \models^{CF} P \wedge$

$$\forall\, l^a, l^{a'}, l^{a''} \in \mathbf{Lab}^a : ((l^a, l^t) \in \hat{I} \wedge (l^{a''}, l^a) \in \hat{I} \wedge (l^{a''}, l^{a'}) \in \hat{I}$$

$$\wedge\, (l^{a'}, n) \in \hat{H}) \Longrightarrow (l^{a'}, l^a) \in \hat{I}$$

(out) $(\hat{I},\hat{H}) \models^{CF} out^{l^t} n.P$ iff $(\hat{I},\hat{H}) \models^{CF} P \wedge$

$$\forall\, l^a, l^{a'}, l^{a''} \in \mathbf{Lab}^a : ((l^a, l^t) \in \hat{I} \wedge (l^{a'}, l^a) \in \hat{I} \wedge (l^{a''}, l^{a'}) \in \hat{I}$$

$$\wedge\, (l^{a'}, n) \in \hat{H}) \Longrightarrow (l^{a''}, l^a) \in \hat{I}$$

$(open)$ $(\hat{I},\hat{H}) \models^{CF} open^{l^t} n.P$ iff $(\hat{I},\hat{H}) \models^{CF} P \wedge$

$$\forall\, l^a, l^{a'} \in \mathbf{Lab}^a : ((l^a, l^t) \in \hat{I} \wedge (l^a, l^{a'}) \in \hat{I} \wedge (l^{a'}, n) \in \hat{H})$$

$$\Longrightarrow \{(l^a, l') \mid (l^{a'}, l') \in \hat{I}\} \subseteq \hat{I}$$

Table 2.11: Control flow analysis of firewall validation

$(I,H) \models^l_{me} (\nu n^\mu)P$	iff $(I,H) \models^l_{me[n \mapsto \mu]} P$
$(I,H) \models^l_{me} 0$	iff true
$(I,H) \models^l_{me} P \mid P'$	iff $(I,H) \models^l_{me} P \ \wedge \ (I,H) \models^l_{me} P'$
$(I,H) \models^l_{me} !P$	iff $(I,H) \models^l_{me} P$
$(I,H) \models^l_{me} N^{l^a}[P]$	iff $(I,H) \models^l_{me} P \ \wedge \ l^a \in I(l) \ \wedge$
	$(I,H) \models^l_{me} N : \tilde{N} \ \wedge \ \tilde{N} \subseteq H(l^a)$
$(I,H) \models^l_{me} M.P$	iff $(I,H) \models^l_{me} P \ \wedge \ (I,H) \models^l_{me} M : \tilde{M}$
$(I,H) \trianglerighteq_{me} \mathsf{in}^{l^t} \ N : \tilde{M}$	iff $(I,H) \Vdash_{me} \ N : \tilde{N} \ \wedge \ \tilde{M} \supseteq \{\mathsf{in}^{l_t} \ \mu \mid \mu \in \tilde{N}\}$
$(I,H) \trianglerighteq_{me} \mathsf{out}^{l^t} \ N : \tilde{M}$	iff $(I,H) \Vdash_{me} \ N : \tilde{N} \ \wedge \ \tilde{M} \supseteq \{\mathsf{out}^{l_t} \ \mu \mid \mu \in \tilde{N}\}$
$(I,H) \trianglerighteq_{me} \mathsf{open}^{l^t} \ N : \tilde{M}$	iff $(I,H) \Vdash_{me} \ N : \tilde{N} \ \wedge \ \tilde{M} \supseteq \{\mathsf{open}^{l_t} \ \mu \mid \mu \in \tilde{N}\}$
$(I,H) \Vdash_{me} \ n : \tilde{N}$	iff $\tilde{N} \supseteq \{me(n)\}$
$(I,H) \Vdash^l \ \mathsf{in}^{l^t} \ \mu$	iff $\mathsf{in}^{l^t} \ \mu \ \wedge \ \forall l^a \in I^{-1}(\mathsf{in}^{l^t} \ \mu) : \forall l^{a'} \in I^{-1}(l^a)$
	$\forall l^{a''} \in I^{-1}(l^a) \cap H^{-1}(\mu) : l^a \in I(l^{a''})$
$(I,H) \Vdash^l \ \mathsf{out}^{l^t} \ \mu$	iff $\mathsf{out}^{l^t} \ \mu \ \wedge \ \forall l^a \in I^{-1}(\mathsf{out}^{l^t} \ \mu) : \forall l^{a'} \in I^{-1}(l^a)$
	$\forall l^{a''} \in I^{-1}(l^a) \cap H^{-1}(\mu) : l^a \in I(l^{a''})$
$(I,H) \Vdash^l \ \mathsf{open}^{l^t} \ \mu$	iff $\mathsf{open}^{l^t} \ \mu \ \wedge \ \forall l^a \in I^{-1}(\mathsf{open}^{l^t} \ \mu) : \forall l^{a'} \in I^{-1}(l^a)$
	$\forall l^{a''} \in I^{-1}(l^a) \cap H^{-1}(\mu) : l^a \in I(l^{a''})$

Nielson and his collaborators, does not accurately represent the way firewalls behave. The presence of a guiding pilot and the three passwords required to cross a firewall are unnecessary complications. Details of firewall rules, such as the format recommended by NIST [59], are not even considered. The complexity of the analysis makes it less than ideal for implementation in real-time operating machines such as firewalls. Moreover, considerable developments have been witnessed in the field of firewall policies. In [12, 58], Bellovin introduced the concept of distributed firewalls as a flexible policy control mechanism. Guttman [47] explored issues related to automatically generating local policies from a given network topology and a global policy. Automatic management of network security policies through a policy enforcement tool is addressed by Burns *et al.* in [19]. In [3], Al-Shaer *et al.* tackle the problem of detecting filtering rules anomalies both within an individual firewall and in basic forms of distributed firewalls, such as a three-level cascading firewall. The problems to solve can however become far more complex with different topologies such as stars and rings, where ambient nesting leads to several potential paths to a destination and the order of local policies is capital. We address these issues in Chapter 3, in the shape of a dedicated firewall calculus that provides a more natural model and adds a distributed aspect to the verification process.

2.3.2 Safe Ambients and Derived Approaches

The calculus of mobile ambients has inspired a number of other researchers to develop it further. A different approach called safe ambients is addressed in computer security related papers due to Degano, Levi, Bugliesi, Sangiorgi *et al.* [69, 70, 18, 31].

Safe Ambients

A new calculus called safe ambients is introduced by Levi and Sangiorgi [69, 70]. They examine process interference, which is defined as the possibility for a process to encounter at least two processes willing to react to his request or service. Process interference is one of the most complicated problems associated with concurrency. The solution for interference presented in the paper is the addition of three new primitives called cocapabilities $\overline{\mathsf{in}}\ n, \overline{\mathsf{out}}\ n$

and $\overline{\mathsf{open}}\ n$. The idea is to provide agreement between the participating parties by associating any action of a process with a corresponding permission in the target ambient. The reduction semantics of Safe Ambients is displayed in Table 2.12.

Table 2.12: Reduction semantics of Safe Ambients

$$(R-in) \quad n[\mathsf{in}\ m.P_1 \mid P_2] \mid m[\overline{\mathsf{in}}\ m.Q_1 \mid Q_2] \longrightarrow m[n[P_1 \mid P_2] \mid Q_1 \mid Q_2]$$

$$(R-out) \quad m[n[\mathsf{out}\ m.P_1 \mid P_2] \mid \overline{\mathsf{out}}\ m.Q_1 \mid Q_2] \longrightarrow n[P_1 \mid P_2] \mid m[Q_1 \mid Q_2]$$

$$(R-open) \quad \mathsf{open}\ n.P \mid n[\overline{\mathsf{open}}\ n.Q_1 \mid Q_2] \longrightarrow P \mid Q_1 \mid Q_2$$

$$(R-Msg) \quad < M > \mid (x:W)P \longrightarrow P\{M/x\}$$

$$(R-Eps) \quad \epsilon.P \longrightarrow P$$

$$(R-Rec) \quad \mathsf{rec}\ X.P \longrightarrow P\{\mathsf{rec}\ X.P/X\}$$

$$(R-Res) \quad \frac{P \longrightarrow P'}{(\nu n : Amb)P \longrightarrow (\nu n : Amb)P'}$$

$$(R-Par) \quad \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q}$$

$$(R-Amb) \quad \frac{P \longrightarrow P'}{n[P] \longrightarrow n[P']}$$

$$(R-Str) \quad \frac{P \longrightarrow P' \quad P' \equiv P'' \quad P'' \longrightarrow P'''}{P \longrightarrow P'''}$$

The addition of cocapabilities also facilitates proving behavioral properties and writing more robust programs that behave as expected in all contexts.

Safe ambients are further enhanced with a type system that provides mobility controls and removes all dangerous interferences.

Secure Safe Ambients

The Secure Safe Ambients (SSA) defined by Bugliesi and Castagna in [18] are a typed variant of Safe Ambients. The type system permits the expression and verification of behavioral invariants of ambients. Moreover, the type system is also capable to capture both explicit and implicit process and ambient behavior.

The authors introduce the type system of SSA, define algorithms for type checking and type reconstruction, define a language for expressing security properties, and assess a distributed version of SSA and its type system. The typing rules for SSA are presented in Table 2.13, where E is a type environment, Π is a domain environment, and $Img(E)$ and $Dom(\Pi)$ represent the image of E and the domain of Π, respectively. The rules for a well-typed process P are derived from judgements of the form: $\Pi, E \vdash P : \mathsf{P}$. The two (Action) rules are interesting, as they stipulate that the effect of executing actions can be observed either at the level of the enclosing ambient, or in the continuation process.

Therefore, types apply for both the immediate behavior of processes and the behavior of process residuals, covering every possible evolution of processes in a given context. This particular characteristic allows the type system of SSA to detect potential for security attacks such as Trojan Horses and other combinations of malicious agents in ill-typed contexts.

Control Flow Analysis of Safe Ambients

Degano, Levi, and Bodei [31] obtained some noteworthy results on control flow analysis in the context of safe ambients. They were able to predict the parent ambient, for each ambient n, whenever a capability or ambient α are contained at the top level. This allowed them to restrict significantly the potential movements of an ambient and to use static techniques for proving security properties of programs.

Table 2.13: Typing rules for Secure Safe Ambients

$$\text{(Type proc)}\ \frac{\Pi \vdash \diamond \quad fn(\mathsf{P}) \subseteq \mathsf{Dom}(\Pi) \quad \Pi \vdash \mathsf{P}\ \mathsf{closed}}{\Pi \vdash \mathsf{P}}$$

$$\text{(Env)}\ \frac{\Pi \vdash \diamond \quad \mathsf{Img}(E) \subseteq \mathsf{Dom}(\Pi)}{\Pi, E \vdash \diamond}$$

$$\text{(Name)}\ \frac{\Pi \vdash \diamond \quad a \in \mathsf{Dom}(E)}{\Pi, E \vdash a : E(a)}$$

$$\text{(Dead)}\ \frac{\Pi, E \vdash \diamond}{\Pi, E \vdash 0 : \{\oslash, \oslash, \oslash\}}$$

$$\text{(Par)}\ \frac{\Pi, E \vdash P : \mathsf{P} \quad \Pi, E \vdash Q : \mathsf{P}}{\Pi, E \vdash P \mid Q : \mathsf{P}}$$

$$\text{(Repl)}\ \frac{\Pi, E \vdash P : \mathsf{P}}{\Pi, E \vdash !P : \mathsf{P}}$$

$$\text{(Action}^{\uparrow}\text{)}\ \frac{\Pi \vdash P : \mathsf{P} \quad \Pi, E \vdash a : D \quad \mathsf{cap}\ D \in \mathsf{P}^{\uparrow}}{\Pi, E \vdash \mathsf{cap}\ a.P : \mathsf{P}}\ \mathsf{cap}\ \in\ \{\mathsf{in}, \overline{\mathsf{in}}, \mathsf{out}, \overline{\mathsf{open}}\}$$

$$\text{(Action}^{=}\text{)}\ \frac{\Pi \vdash P : \mathsf{P} \quad \Pi, E \vdash a : D \quad \mathsf{cap}\ D \in \mathsf{P}^{=}}{\Pi, E \vdash \mathsf{cap}\ a.P : \mathsf{P}}\ \mathsf{cap}\ \in\ \{\overline{\mathsf{out}}, \mathsf{open}\}$$

$$\text{(Restr)}\ \frac{\Pi, E, a : D \vdash P : \mathsf{P} \quad a \notin \mathsf{Dom}(E)}{\Pi, E \vdash (\nu a : D)P : \mathsf{P}}$$

$$\text{(Amb)}\ \frac{\Pi, E \vdash P : \mathsf{P} \quad \Pi, E \vdash a : D \quad \Pi \vdash D\ \mathsf{bounds}\ \mathsf{P}}{\Pi, E \vdash a[P] : \Pi(D)}$$

$$\text{(Subsumption)}\ \frac{\Pi, E \vdash P : \mathsf{P} \quad \Pi \vdash Q \quad P \subseteq Q}{\Pi, E \vdash \mathsf{P} : \mathsf{Q}}$$

The clauses of the Control Flow Analysis on SA are introduced in Table 2.14. They operate on the structure of a process P and update its current ambient k, its label L, and its current element I. The study formally proves that there exist a least solution valid according to the clauses, which remains valid under process reduction.

Table 2.14: Control Flow Analysis of Safe Ambients

$\mathsf{nil} : \phi \models_I^{k,L} 0$	iff true
$\mathsf{amb} : \phi \models_I^{k,L} n[P]$	iff $n^L \in I \ \wedge \ \exists I', \phi \models_{I'}^{n,\{k,k\}} P \ \wedge \ I' \sqsubseteq \phi(n)$
$\mathsf{pref} : \phi \models_I^{k,L} \mu.P \ \wedge \mu \neq \mathsf{open}\ n$	iff $\begin{cases} \mu^L \in I \ \wedge \ \phi \models_I^{k,(l_1 \cup t(\mu,k),t(\mu,k))} P \wedge \\ \forall h \in t(\mu,k) : k^{(f(h),f(h))} \in \phi(h) \end{cases}$
$\overline{\mathsf{pref}} : \phi \models_I^{k,L} \overline{\mu}.P$	iff $\overline{\mu}^L \in I \ \wedge \ \phi \models_I^{k,L} P$
$\mathsf{open}\ \phi \models_I^{k,L} \mathsf{open}\ n.P$	iff $\begin{cases} (\mathsf{open}\ n^L \in I) \ \wedge \ \exists I_1 \ : \ \phi \models_{I'}^{k,L} P \wedge \\ I_1 \otimes_k (\gamma, (k, \phi(n), n) \oplus L) \sqsubseteq I \wedge \\ \forall h \in \mathrm{ENAB}(\overline{open}\ n, n), h \in \mathcal{N}, \\ \alpha^L \in \phi(h) : n \in l_i \Rightarrow k \in l_i, \forall i = 1, 2 \end{cases}$
$\mathsf{par}\ \phi \models_I^{k,L} p \mid Q$	iff $\begin{cases} \exists I_1, I_2 \ : \ \phi \models_{I_1}^{k,L} P \ \wedge \ x\phi \models_{I_2}^{k,L} Q \wedge \\ I_1 \otimes_k I_2 \sqsubseteq I \end{cases}$
$\mathsf{repl}\ \phi \models_I^{k,L} !P$	iff $\exists I', \phi \models_{I'}^{k,L} P \ \wedge \ I' \otimes_k I' \sqsubseteq I$

Ambients are classified as either trustworthy or untrustworthy, depending on how they preserve secrecy. Simply put, if an untrustworthy ambient cannot open a trustworthy one, than the program dynamically preserves secrecy. The tests are dependent on some contextual information and are not valid in all circumstances, but ambients passing them can be easily identified with the aid of a tester process E which simulates the most hostile environment matching the requirements. The fact that the tests are not universally valid is a serious limitation of the approach.

2.3.3 Guarded Boxed Ambients

Ferrari *et al.* [33] propose an extension of the ambient calculus by considering the concept of ambient monitoring and coordination policy by attaching a guardian to each ambient. Their formal model, called Guarded Boxed Ambients, separates computational mechanisms (communication and mobility primitives) from policy enforcers. This allows them to support the specification of multiple dynamic security policies. The segregation of primitives and enforcers gives flexibility and finer granularity in expressing policies. Guardians define a local security context and monitor the activity of processes and sub-ambients as well as the interaction with the external environment. They implement the set of permissions that may be granted to the agents entering, exiting, and communicating in the monitored environment. Guardians also have coordination abilities and they can successfully cooperate for an effective propagation of a policy change across an environment. The approach provides an alternative to our idea of automatic reconfiguration of policies.

Lacasse *et al.* in [65] propose a similar idea of controlled processes forced to respect a security policy. Their approach does not rely on ambients, but on a CCS [76] process algebra extension with new syntactic constructs for monitored processes. The potential of program monitors and their role in implementing run-time security policies is examined by Schneider [94].

Table 2.15: Categories in Guarded Boxed Ambients

$$
\begin{array}{rcl}
W \in \mathsf{Gld} ::= & \ldots & \\
g \in \mathsf{Perm} ::= & new \mid spawn \mid box \mid in_1 \mid in_2 \mid in_3 \mid out_1 \mid out_2 \mid out_3 \mid comm & \\
\mid & rdup_1 \mid rdup_2 \mid wrup_1 \mid wrup_2 \mid rddn_1 \mid rddn_2 \mid wrdn_1 \mid wrdn_2 & \\
G \in \mathsf{Gard} ::= & g(\overline{x}) \text{ when } b \text{ then} \mid G \mid G_1 \ \vee \ G_2 \mid W(\overline{v}) \mid & \\
& G \backslash g(\overline{x}) \text{ when } b \mid \bot \mid \top \mid G_1 \ \wedge \ G_2 & \\
v, P \in \mathsf{Proc} ::= & \ldots \mid G &
\end{array}
$$

2.3.4 Controlled Ambients

In [102], Teller *et al.* present a formalism used to control resources in parallel and distributed mobile systems. The authors developed an extension of the mobile ambient calculus named controlled ambients and use this extension to express safety and reliability issues. The language of Controlled Ambients uses cocapabilities for expressing interaction, similar to those found in the theory of Safe Ambients. However, the Controlled Ambients cocapabilities are stronger in the sense that an ambient must have knowledge about the name of all moving ambients. The syntax of Controlled Ambients is presented in Table 2.17.

Resource control features required for the declared purpose of modeling Denial of Service attacks are embedded into ambients with the aid of two new parameters called capacity and weight. These parameters determine how many resources are available to subambients and how many resources are required from the parent ambient. A typing system, shown in Table 2.18, introduces judgements for types (ambients, processes, and messages) and rules

Table 2.16: Transition rules in Guarded Boxed Ambients

$$new\ \frac{P \xrightarrow{new(n_1,G_1)} P' \quad G \xrightarrow{new(n_1,G_1)} G'}{F, n(G)[M,P] \Longrightarrow F(n_1 \mapsto G_1), n(G')[M,P']}\ n_1 \notin dom(F)$$

$$spawn\ \frac{P \xrightarrow{spawn(P_1)} P' \quad G \xrightarrow{spawn(P_1)} G'}{F, n(G)[M,P] \Longrightarrow F(n_1 \mapsto G_1), n(G')[M,P']}$$

$$box\ \frac{P \xrightarrow{box(n_1,G_1)} P' \quad G \xrightarrow{box(n_1,G_1)} G'}{F, n(G)[M,P] \Longrightarrow F(n_1 \mapsto G_1), n(G')[M,P']}$$

$$in\ \frac{P \xrightarrow{in(n_1,G_1)} P' \quad G \xrightarrow{in_1(n_1,G_1)} G' \quad P \xrightarrow{in_2(n_1,G_1)} P' \quad G \xrightarrow{in_3(n_1,G_1)} G'}{F, n(G)[M,P] \Longrightarrow F(n_1 \mapsto G_1), n(G')[M,P']}$$

$$out\ \frac{P \xrightarrow{out(n_1,G_1)} P' \quad G \xrightarrow{out_1(n_1,G_1)} G' \quad P \xrightarrow{out_2(n_1,G_1)} P' \quad G \xrightarrow{out_3(n_1,G_1)} G'}{F, n(G)[M,P] \Longrightarrow F(n_1 \mapsto G_1), n(G')[M,P']}$$

$$comm\ \frac{P \xrightarrow{vr(n_1,G_1)} P' \quad G \xrightarrow{rd(n_1,G_1)} G' \quad P \xrightarrow{comm(n_1,G_1)} P'}{F, n(G)[M,P] \Longrightarrow F(n_1 \mapsto G_1), n(G')[M,P']}$$

$$rdup\ \frac{P \xrightarrow{wr(n_1,G_1)} P' \quad G \xrightarrow{rdup(n_1,G_1)} G' \quad P \xrightarrow{rdup_1(n_1,G_1)} P' \quad G \xrightarrow{rdup_2(n_1,G_1)} G'}{F, n(G)[M,P] \Longrightarrow F(n_1 \mapsto G_1), n(G')[M,P']}$$

$$wrup\ \frac{P \xrightarrow{rd(n_1,G_1)} P' \quad G \xrightarrow{wrup(n_1,G_1)} G' \quad P \xrightarrow{wrup_1(n_1,G_1)} P' \quad G \xrightarrow{wrup_2(n_1,G_1)} G'}{F, n(G)[M,P] \Longrightarrow F(n_1 \mapsto G_1), n(G')[M,P']}$$

$$rddn\ \frac{P \xrightarrow{rddn(n_1,G_1)} P' \quad G \xrightarrow{wr(n_1,G_1)} G' \quad P \xrightarrow{rddn_1(n_1,G_1)} P' \quad G \xrightarrow{rddn_2(n_1,G_1)} G'}{F, n(G)[M,P] \Longrightarrow F(n_1 \mapsto G_1), n(G')[M,P']}$$

$$wrdn\ \frac{P \xrightarrow{wrdn(n_1,G_1)} P' \quad G \xrightarrow{rd(n_1,G_1)} G' \quad P \xrightarrow{wrdn_1(n_1,G_1)} P' \quad G \xrightarrow{wrdn_2(n_1,G_1)} G'}{F, n(G)[M,P] \Longrightarrow F(n_1 \mapsto G_1), n(G')[M,P']}$$

Table 2.17: Controlled Ambients syntax

P	::=	0	null process
	|	$M.P$	capability
	|	$m[P]$	ambient
	|	$P_1 \mid P_2$	parallel comp.
	|	$(\nu n : A)P$	restriction
	|	$\mathsf{rec}\ X.P$	recursion
	|	X	ambient variable
	|	$(n : A)P$	abstraction
	|	$< m >$	message emission
M	::=	$\mathsf{in}\ m$	enter m
	|	$\mathsf{out}\ m$	leave m
	|	$\mathsf{open}\ m$	open m
	|	$\overline{\mathsf{in}_\uparrow}\ m$	m may climb in upwards
	|	$\overline{\mathsf{in}_\downarrow}\ m$	m may climb in downwards
	|	$\overline{\mathsf{out}_\uparrow}\ m$	m may climb out upwards
	|	$\overline{\mathsf{out}_\downarrow}\ m$	m may climb out downwards
	|	$\overline{\mathsf{open}}\{m, h\}$	h may open m

for resource allocation and communication.

Resource policy compliance is defined in terms of capacity and weight with the aid of types. The construction of the model is rather complex. The authors claim that their formalism is "more reasonable" and "more realistic" than other approaches, such as safe ambients, because it can model a listening action on behalf of the system. This main benefit is useful for the authors' application to Denial of Service attacks, but not necessary for most practical purposes: the systems are ready and transparently listening as soon as they are implemented, since by definition (syntax and semantics of the specification language) they can react to some form of interaction (e.g. *in* or *out* capabilities). The resource policies consist only in availability statements, rather than access control (security) requirements. We find the approach very limiting for real life implementations, where such controls are a fundamental part of the system requirements.

2.4 Critical Remarks

The various security policy enforcement mechanisms contributed to solving the specific purpose they were created for and may not apply to other contexts. For instance, while the Intel x86 architecture is still very popular, it is not the only one used. On the contrary, with the overwhelming abundance and diversity of mobile computing platforms, from smartphones to laptops and tablets, Intel themselves are moving away from their well-established roots. In this case, the results presented by Erlingson and Schneider [103] are impractical and not universally applicable, as the authors themselves admit.

The access control approaches that use some variant of ambient calculus (control flow analysis, safe, controlled, and guarded ambients, etc.) involve elaborated constructs for verifying relatively simple properties and therefore have limited scalability. While the concepts are intuitive and clearly proven, they lack in modularity and applicability to most real-world complex systems.

We do not support some of the statements in the reviewed work. Sablefeld and Myers [90] claim that access control conventional mechanisms do

Table 2.18: Typing rules for Controlled Ambients

$$\frac{\Gamma(n) = A}{\Gamma \vdash n : A}\, T-name \qquad \frac{\Gamma(X) = CAPR(t)[T]}{\Gamma \vdash X : CAPR(t')[T]}\, T-var$$

$$\frac{\Gamma, X : CAPR(t)[T] \vdash P : CAPR(t)[T]}{\Gamma \vdash \mathsf{rec}\ X.P : CAPR(t')[T]}\, T-rec$$

$$\frac{\Gamma \vdash P : CAPR(t)[T]}{\Gamma \vdash \mathsf{in}\ m.P : CAPR(t)[T]}\, T-in \qquad \frac{\Gamma \vdash P : CAPR(t)[T]}{\Gamma \vdash \mathsf{out}\ m.P : CAPR(t)[T]}\, T-out$$

$$\frac{\Gamma \vdash P : CAPR(t)[T]\ \Gamma \vdash m : CAAM(s,e)[T']}{\Gamma \vdash \overline{\mathsf{in}_\delta}\ m.P : CAPR(t+e)[T]}\, T-coin$$

$$\frac{\Gamma \vdash P : CAPR(t)[T]\ \Gamma \vdash m : CAAM(s,e)[T']}{\Gamma \vdash \overline{\mathsf{out}_\delta}\ m.P : CAPR(t-e)[T]}\, T-coout$$

$$\frac{\Gamma \vdash m : CAAM(s,e)[T]\ \Gamma \vdash P : CAPR(t)[T]}{\Gamma \vdash \mathsf{open}\ m.P : CAPR(t-e+s)[T]}\, T-open$$

$$\frac{\Gamma \vdash m : CAAM(s,e)[T]\ \Gamma \vdash R : CAPR(t)[T]}{\Gamma \vdash \mathsf{open}\{m,h\}.R : CAPR(t)[T]}\, T-coopen$$

$$\frac{}{\Gamma \vdash 0 : U}\, T-nil \qquad \frac{\Gamma \vdash m : CAAM(s,e)[T]\ \Gamma \vdash P : CAPR(a)[T]}{\Gamma \vdash m[P] : CAPR(t)[T']}\, T-amb$$

$$\frac{\Gamma, n : A \vdash P : U}{\Gamma \vdash (\nu n : A)P : U}\, T-res \qquad \frac{\Gamma \vdash P : CAPR(t)[T]\ \Gamma \vdash Q : CAPR(t')[T]}{\Gamma \vdash P|Q : CAPR(t+t')[T]}\, T-par$$

$$\frac{\Gamma \vdash m : A}{\Gamma \vdash < m > : CAPR(t')[t, A]}\, T-snd \qquad \frac{\Gamma, x : A \vdash P : CAPR(t)[t, A]}{\Gamma \vdash (x : A)P : CAPR(t')[t, A]}\, T-rcv$$

not directly address the enforcement of information flow policies, which we strongly disagree with. While the programming language-based technique they propose adds another layer of protection, it does not dismiss the need for the other layers (encryption, access control, etc.). In our opinion, the collaborative use of all means of policy enforcement only strengthens the target system. Moreover, we feel that there is a need for additional verification, as the error-prone programming process may require appropriate checking.

Martins [72] states that the type system he proposed eliminates policy violations. Still, the security policies manually specified by a network administrator are the potential weak link. There is no guarantee that the policies correspond to the desired implementation unless formal verification or comprehensive testing are performed.

The runtime enforcement automata approach of Bauer, Ligatti and Walker [8, 9, 10] only concerns dynamic enforcement at runtime, rather than providing persistent controls. Every subsequent execution of a non-compliant program still requires the effort to modify the behavior, unless the source code is adjusted following an initial run. Our methodology could be used in conjunction with it, and not as a substitute.

The interface equation provides an interesting formulation for the security enforcement problem. The ideas presented to date refer more to the extraction of specifications and converters, rather than offer a fitting answer to our issue. We adopt a different approach that is detailed in Chapter 6 of the thesis.

The translation of local security policies as defined by Orlovsky [83] is not as straightforward as desired. By contrast, system and policy specification with our calculus is simple and can be easily automated. An implementation of our system specification technique has already been obtained with *PEP*, an application developed under our supervision.

The work of Miksad [75] does tackle policy specification, but not the enforcement aspects. We are employing our own calculus and logic for the system and policy specification and the interface equation for verification and enforcement.

The merits of earlier papers on linking security policy specification and the ambient calculus, such as the article of Braghin *et al.* [15], are obvious. However, our methodology is among the first instances we are aware of where policy enforcement is also added to the mix in order to guarantee policy compliance through enforcement. The unifying framework that we built provides a coherent platform for all stages of system design and maintenance, from specification, to resource protection and security policies enforcement. Such a framework elevates the level of confidence and trust associated with the underlying system.

Applicability to Security Behaviour Analysis

Chapter 3

A Calculus for Distributed Firewall Specification and Verification

Abstract

This chapter proposes a firewall specification calculus suited for expressing security policies implemented in distributed firewalls. Our syntax and semantics, inspired from the ambient calculus, allow the specification of filtering rules for both single and distributed configurations. We show how the calculus can be used to address the problem of conflict detection and how our approach facilitates the analysis of the effect that network topologies have on distributed firewall policies.

3.1 Introduction

Security has become a major component of network planning in recent years and the infrastructure supporting it is more and more complex. Most corporate networks are divided into several internal and external sub-networks protected by firewalls collaborating to implement a global security policy. The problem of enforcing a global security policy through distributed firewalls can be very challenging. Each firewall implements a local policy that needs to be in harmony with the global one. Otherwise, conflicts will arise and security can be breached.

This chapter is structured as follows. Section 3.2 describes briefly the ambient calculus and firewall policies. Section 3.3 presents our calculus suited for representing network packets, network topologies and firewall policies. Section 3.4 provides the framework needed for verifying distributed firewall policies compatibility. The case study provided in Section 3.5 illustrates how our calculus can be applied to specify a network example and to detect a conflict in a distributed firewall policy of the network. Section 3.6, concludes this chapter and states some possible extensions for this work.

3.2 Ambients and Firewall Policies

The concept of ambient calculus was originally introduced in [21]. An ambient is a delimited space that has a name, an interior and an exterior and can contain processes. The movement of the ambient processes is governed by their capabilities to go in and out of an ambient or open them. Moreover, an ambient can move inside or outside another ambient, carrying the enclosed processes with it. The syntax of ambient calculus is based on three components: processes, capabilities and names. The mobility of an ambient depends on the capabilities of the enclosed processes: *in*, *out* and *open*. The *in* capability gives an ambient the capacity to enter a co-located ambient and the *out* capability allows an ambient to leave its parent. The *open* capability dissolves the borders around an ambient. The operational semantics of the ambient calculus is based on a structural congruence between processes and reduction relations.

Firewalls implement security policies in order to provide protection from unwanted and potentially harmful packets. Firewall policies are sets of rules that filter network packets and control access of inbound traffic to various parts of the network, specifying through an *accept* or *deny* action what goes through behind the firewall. Besides their actions, rules have several filtering fields. The most commonly used fields are the source and destination IP address, the name of the protocol used for communication and the source and destination ports used by the protocol. When a new packet arrives at the firewall, those fields are read and checked against the rules of the firewall. The NIST recommendation for firewall policies [59] proposes the following

common format for packet filtering rules:

$$order\ prot\ src\_ip\ src\_port\ dest\_ip\ dest\_port\ act$$

Rules are organized hierarchically, the most important ones being at the top of the set. They are applied in that order to all the packets. The order is highly important, as it can dramatically change the effect of the policy. The right collection of rules applied in the wrong order can simply cancel the advantages of using a firewall for filtering. Policy design is not an easy task, especially if the rule-set is fairly large. It is not uncommon to have more than 20 rules and that can generate internal conflicts. The types of anomalies inherent to local filtering packet firewalls, as described in [3], are: shadowing, correlation, generalization, redundancy and irrelevance.

Local security policies are implemented in single firewalls. A global security policy employs several firewalls in a distributed configuration and takes into account the interaction between them. Hence, a firewall policy can be distributed across several sub-networks, depending on the trust relationship between sub-networks and the topology used. For instance, in a simple bus topology with three sub-networks A, B and C, the security administrators of A may have the option to choose between employing single or distributed firewalls. Implementing a single firewall at the border between A and B results in filtering all traffic from both B and C. If, however, A trusts B, the administrators can rely on B to protect A from all the unwanted traffic coming from C. Then they only need to specify the rules filtering traffic from B in the local firewall separating A and B. Such a configuration that requires cooperation of local firewalls to achieve a common security goal identifies a distributed firewall.

3.3 Distributed Firewall Specification

In this section we define the syntax and semantics of a calculus suited for distributed firewall specification. Following [21], we use the concept of ambient that provides a flexible way for modeling mobility aspects in hierarchical structures, such as networks. We consider as ambient a delimited space protected by a filtering rule. The basic idea is to treat networks as processes protected by firewalls, which are also seen as processes. This uniform ap-

proach allows an intuitive examination of the filtering capabilities of single and distributed firewall policies.

3.3.1 Syntax

We present in Table 3.1 the syntax of our calculus called *Firewall Policy Calculus* (*FPC*).

The following can be processes:

- **inactivity**: 0 is a process that does nothing;
- **parallel composition**: P | Q refers to parallel execution of processes P and Q; it is a commutative and associative operator;
- **capability**: the in cond.P allows a process P to enter in some cases an ambient protected by the condition cond.
- **ambient**: $\overset{\mathsf{act}}{\underset{\mathsf{cond}}{}}[\mathsf{P}]$ denotes a network P protected by a filtering rule for the inbound traffic; in this syntactic construct act denotes the action of the filtering rule deny or accept and cond denotes the condition of the filtering rule: a conjunction of predicates; each predicate is used to specify packet header information (IPs and ports expressed as src_ip, src_port, dest_ip, dest_port, protocols expressed as prot);
- **conditional choice**: $\otimes$ is a binary operator used to build ordered composed filtering rules.

We denote by $\mathcal{P}$ the set of all processes described by our semantics and by $\mathcal{C}$ the set of all conditions described by our semantics. Conditions are made of a single predicate or conjunctions of predicates and they determine if rules apply or not to packets. The decimal value num is in the range 0-255 for addresses and in the range 1-10000 for port numbers. For instance, the following is a predicate: src_ip = 132.213.10.12, dest_ip = 132.210.1.12 ... 132.210.1.254, prot = tcp and dest_port = 80.

Intuitively, the process $\overset{\mathsf{act}}{\underset{\mathsf{cond}}{}}[\mathsf{P}]$ describes a piece of a network P protected by a filtering rule cond $\rightarrow$ act. Any network packet can enter P if the packet

Table 3.1: Syntax of *FPC*

P, Q	::=		**processes**
	|	0	**inactivity**
	|	P | Q	**parallel composition**
	|	!P	**replication**
	|	in cond.P	**capability**
	|	$\overset{\text{act}}{\text{cond}}[\text{P}]$	**ambient**
	|	P $\otimes$ Q	**conditional choice**
act	::=	accept | deny	**rule actions**
cond	::=	predicate | cond $\wedge$ cond	**condition**
predicate	::=		**predicate**
	|	address = adr_val [. . . adr_val]	
	|	port = val_port	
	|	prot = name_prot	
address	::=	src_ip | dest_ip	**addresses**
port	::=	src_port | dest_port	**ports**
val_port	::=	val [. . . val] | $*$	**port numbers**
name_prot	::=	tcp | udp | icmp | $*$	**protocol names**
adr_val	::=	val.val.val.val	**address value**
val	::=	num | $*$	**value**

header satisfies cond and act is an accept. This guarantees that all legitimate traffic will be allowed in and all unwanted traffic will be filtered out.

While a rule alone can be a firewall policy, it is not often the case in practice. Policies are ordered sets of rules and they can be combined for modeling networks that implement more than one policy. With our syntax, we use the $\otimes$ operator to build firewall policies from rules. For instance, $\mathsf{P} \otimes \mathsf{Q}$ could describe a process where P has the form $\overset{\mathsf{act}}{\mathsf{cond}}[\mathsf{P}']$ and Q symbolises an ambient firewall process.

3.3.2 Semantics

We now give the operational semantics of the firewall policy calculus. It consists of two main sections: structural congruence, denoted by $\equiv$, and reduction relation, denoted by $\rightarrow$. The structural congruence $\equiv$ is defined in Table 3.2 and illustrates process equivalence in the context of our calculus. Properties such as reflexivity, symmetry, transitivity, parallelism, commutativity and associativity are quite obvious. Replication and replication parallelism are used for expressing iterations and recursion. This is important for describing, for instance, that a certain process attempts repeatedly to access a protected domain. Note that, since rule composition order is relevant, the composition operator $\otimes$ is not commutative.

The reduction relation defined in Table 3.3 is based on the ambient capabilities of the processes embedded in the matched packets. We denote by $\rightarrow^*$ a sequence of reduction relations. If there is no further reduction possible for a process P, we say that the process is in its normal form with respect to the reduction relation and denote it by $\mathsf{P}_{\Downarrow}$. For instance, let $\mathsf{P} = \mathsf{in\ c.Q} \mid \overset{\mathsf{accept}}{c'}[\mathsf{R}]$. Its normal form $\mathsf{P}_{\Downarrow}$, obtained by applying the reduction relation in Table 3.3, is $\overset{\mathsf{accept}}{c'}[\mathsf{Q} \mid \mathsf{R}]$. A normal form exists for any process and it is unique.

The semantics of a condition c, denoted $[\![\ c\ ]\!]$, is the set of all packets satisfying c. More formally:

$$\begin{cases} [\![\ p\ ]\!] & = & \{t \in \mathcal{T} \ :\ p(t) = true\} \\ [\![\ c_1 \wedge c_2\ ]\!] & = & [\![\ c_1\ ]\!] \cap [\![\ c_2\ ]\!] \end{cases}$$

where p is a predicate and $\mathcal{T}$ is the set of all possible packets. The "_"

Table 3.2: Structural Congruence

$P \equiv P$	(**Reflexivity**)
$P \equiv Q \Rightarrow Q \equiv P$	(**Symmetry**)
$P \equiv Q \wedge Q \equiv R \Rightarrow P \equiv R$	(**Transitivity**)
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	(**Parallelism**)
$P \equiv Q \Rightarrow !P \equiv !Q$	(**Replication**)
$P \equiv Q \Rightarrow \mathsf{in}\ \mathsf{cond}.P \equiv \mathsf{in}\ \mathsf{cond}.Q$	(**Capability**)
$P \mid 0 \equiv P$	(**Zero Parallelism**)
$P \mid Q \equiv Q \mid P$	(**Commutativity**)
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(**Associativity**)
$!P \equiv P \mid !P$	(**Replication Parallelism**)
$P \otimes P \equiv P$	(**Idempotent**)
$P \equiv Q \Rightarrow P \otimes R \equiv Q \otimes R$	(**Comp Distributivity**)

Table 3.3: Reduction Relation

(1)	P'	$\rightarrow$	Q'	$if\ \mathsf{P}' \equiv \mathsf{P}\ ,\ \mathsf{P} \rightarrow \mathsf{Q},\ \mathsf{Q} \equiv \mathsf{Q}'$
(2)	$\mathsf{P} \mid \mathsf{R}$	$\rightarrow$	$\mathsf{Q} \mid \mathsf{R}$	$if\ \mathsf{P} \rightarrow \mathsf{Q}$
(3)	$\mathsf{in}\ c.\mathsf{P} \mid {}^{\mathsf{accept}}_{c'}[\mathsf{Q}]$	$\rightarrow$	${}^{\mathsf{accept}}_{c'}[\mathsf{P} \mid \mathsf{Q}]$	$if\ [\![\ c\ ]\!] \subseteq [\![\ c'\ ]\!]$
(4)	$\mathsf{in}\ c.\mathsf{P} \mid {}^{\mathsf{deny}}_{c'}[\mathsf{Q}]$	$\rightarrow$	${}^{\mathsf{deny}}_{c'}[\mathsf{Q}]$	$if\ [\![\ c\ ]\!] \subseteq [\![\ c'\ ]\!]$
(5)	$\mathsf{in}\ c.\mathsf{P} \mid \overline{}_{\overline{c'}}[\mathsf{Q}]$	$\rightarrow$	$\overline{}_{\overline{c'}}[\mathsf{Q}]$	$if\ [\![\ c\ ]\!] \cap [\![\ c'\ ]\!] = \emptyset$
(6)	$\mathsf{in}\ c.\mathsf{P} \mid {}^{\mathsf{accept}}_{c'}[\mathsf{Q}] \otimes \mathsf{R}$	$\rightarrow$	${}^{\mathsf{accept}}_{c'}[\mathsf{P} \mid \mathsf{Q}] \otimes \mathsf{R}$	$if\ [\![\ c\ ]\!] \subseteq [\![\ c'\ ]\!]$
(7)	$\mathsf{in}\ c.\mathsf{P} \mid {}^{\mathsf{deny}}_{c'}[\mathsf{Q}] \otimes \mathsf{R}$	$\rightarrow$	${}^{\mathsf{deny}}_{c'}[\mathsf{Q}] \otimes \mathsf{R}$	$if\ [\![\ c\ ]\!] \subseteq [\![\ c'\ ]\!]$
(8)	$\mathsf{in}\ c.\mathsf{P} \mid \overline{}_{\overline{c'}}[\mathsf{Q}] \otimes \mathsf{R}$	$\rightarrow$	$\overline{}_{\overline{c'}}[\mathsf{Q}] \otimes (\mathsf{in}\ c.\mathsf{P} \mid \mathsf{R})$	$if\ [\![\ c\ ]\!] \subseteq [\![\ c'\ ]\!]$

operator is used to symbolize either an *accept* or a *deny* action in rules (4) and (7) from Table 3.3, where both actions would have the same effect.

In our calculus, we code a network packet by a process $\mathsf{in}\ c.\mathsf{P}$ where c represents the information header (source and destination addresses, ports, and protocol) and P stands for the packet's body. The in capability allows the packet to travel without restriction within the network it originated in. However, in order to enter another network protected by a firewall, it needs to satisfy the conditions imposed by the firewall policy rules. Note that the packet body P forms a process by itself and contains the data that will be interpreted by the destination host.

Intuitively, the process $\mathsf{in}\ c.\mathsf{P} \mid {}^{\mathsf{accept}}_{c}[\mathsf{Q}]$ can be reduced to ${}^{\mathsf{accept}}_{c}[\mathsf{P} \mid \mathsf{Q}]$ since the packet header c satisfies condition c. In this case, the processes contained within the packet can execute inside the protected ambient.

3.4 Distributed Firewall Verification

In this section we introduce the notion of policy compatibility and describe our technique for distributed firewall verification. We employ our formalism for specifying network topologies and provide the framework for assessing if local policies comply or not with the global firewall policy.

Security policies needed for protecting a certain network can be implemented in a single firewall or in several distributed firewalls. Policy specification is not a simple task and it is error-prone. Even simple firewalls can contain intra-policy anomalies. Switching the order of two rules can lead to shadowing, correlation, generalization and coordination, as detailed in [3]. Firewalls in distributed environments face even greater challenges. Besides the aforementioned anomalies, there could be conflicts between the policy of a single firewall and the global policy of the rest of the environment. For the scope of this chapter we take into account inter-firewall incompatibilities. The intra-firewall anomalies will affect individual local firewall policies at a lower level and their effect does not impact directly the distributed policy.

The notion of security policy *compatibility* is now introduced in order to assess if a policy transgresses or not another policy. Let ϕ_1 and ϕ_2 be two

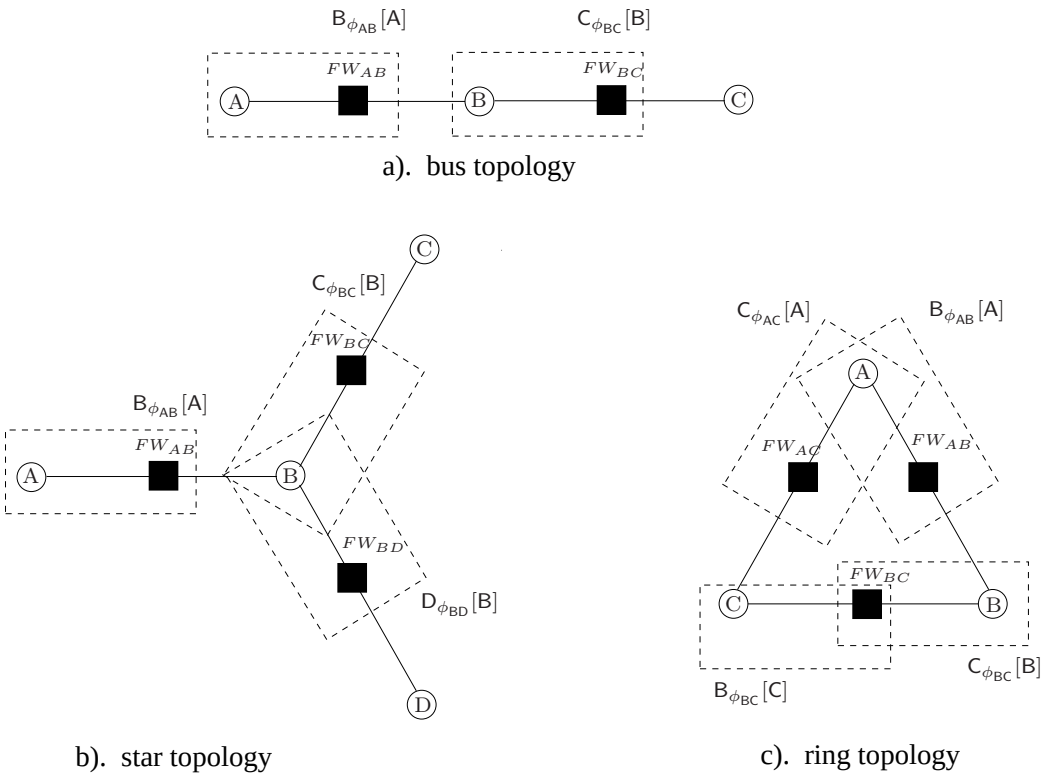


Figure 3.1: Network topologies and distributed firewalls

FPC security policies. The notation $\Downarrow$ is used to show that the policy is in its normal form. We say that ϕ_1 is compatible with ϕ_2, noted $\phi_1 \geq \phi_2$, if ϕ_1 is equally or more permissive than ϕ_2. In other words, if a packet is not allowed to pass by ϕ_1, then it is certainly not allowed by ϕ_2. Formally:

$$\phi_1 \geq \phi_2 \Rightarrow \forall\, \mathsf{c} \in \mathcal{C},\ \mathsf{P} \in \mathcal{P} \mid \mathsf{in\ c.P} \mid (\phi_1)_{\Downarrow} \rightarrow^* (\phi_1)_{\Downarrow} \Rightarrow \mathsf{in\ c.P} \mid (\phi_2)_{\Downarrow} \rightarrow^* (\phi_2)_{\Downarrow}$$

Solving any compatibility issue is vital in the context of distributed firewalls. A badly implemented local filtering policy can endanger the whole network. We address this problem by proposing a technique for distributed firewalls verification. We evaluate the effect of each individual local policy and com-

pare it with the combined policies of the other firewall policies. If the local policy is more permissive than the global one, then they are compatible. If, on the contrary, the local policy is less permissive (i.e. blocks packets that are otherwise allowed by the global policy), then the global policy transgresses (or violates) the local policy and one of them needs to be corrected.

Let FW_{AB} be a local firewall in a distributed firewall configuration implementing the global policy $\mathsf{G_{net}}$. In the following, $\mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}]$ denotes that network A is protected from traffic from network B by the filtering policy ϕ_{AB} implemented in firewall FW_{AB}. We denote by $\mathsf{G}_{\overline{\mathsf{AB}}}$ the global policy for the rest of the network. Unlike the case of the local firewall FW_{AB}, we cannot specify $\mathsf{G}_{\overline{\mathsf{AB}}}$ as it depends on network topology.

The three basic network topologies (bus, star and ring) are presented in Figure 3.1. All relations are expressed from network A's perspective. As expected and confirmed by the topology specification presented in Table 3.4, more connections between networks translate into more paths for a packet to reach its destination. Any network, irrespective of its complexity, can be represented starting from those three basic topologies. An efficient graph algorithm can be implemented in order to automate the topology specification, as the intricacy of a large, well-connected network would make the manual enumeration of all the components very difficult. However, distributing a global policy over a large number of local firewalls is not very practical and a combination of distributed policies may be desirable.

Table 3.4: Distributed firewall topology specification

bus:	$\mathsf{A} \mid \mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}] \mid \mathsf{C}_{\phi_{\mathsf{BC}}}[\mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}]]$
star:	$\mathsf{A} \mid \mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}] \mid \mathsf{C}_{\phi_{\mathsf{BC}}}[\mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}]] \mid \mathsf{D}_{\phi_{\mathsf{BD}}}[\mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}]]$
ring:	$\mathsf{A} \mid \mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}] \mid \mathsf{C}_{\phi_{\mathsf{BC}}}[\mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}]] \mid \mathsf{A}_{\phi_{\mathsf{AC}}}[\mathsf{C}_{\phi_{\mathsf{BC}}}[\mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}]]] \mid \mathsf{C}_{\phi_{\mathsf{AC}}}[\mathsf{A}] \mid$ $\mathsf{B}_{\phi_{\mathsf{BC}}}[\mathsf{C}_{\phi_{\mathsf{AC}}}[\mathsf{A}]] \mid \mathsf{A}_{\phi_{\mathsf{AB}}}[\mathsf{B}_{\phi_{\mathsf{BC}}}[\mathsf{C}_{\phi_{\mathsf{AC}}}[\mathsf{A}]]]$

The policy $\mathsf{G}_{\overline{\mathsf{AB}}}$ is extracted from the network topology specification by eliminating all policies that contain $\mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}]$ and the process A, which contains no policy and has no effect on traffic filtering. For instance, for the simple ring topology presented in Figure 3.1, we have:

$$\mathsf{G}_{\overline{\mathsf{AB}}} : \ \mathsf{C}_{\phi_{\mathsf{AC}}}[\mathsf{A}] \mid \mathsf{B}_{\phi_{\mathsf{BC}}}[\mathsf{C}_{\phi_{\mathsf{AC}}}[\mathsf{A}]] \mid \mathsf{A}_{\phi_{\mathsf{AB}}}[\mathsf{B}_{\phi_{\mathsf{BC}}}[\mathsf{C}_{\phi_{\mathsf{AC}}}[\mathsf{A}]]]$$

The verification process consists then in checking if a packet $\mathsf{in\ c.P}$ has at least the same capability to cross FW_{AB} as it has to cross the rest of the network, protected by the policy $\mathsf{G}_{\overline{\mathsf{AB}}}$. Otherwise, there is a conflict between the two policies. In case the global distributed policy is not compatible with the local policy(written $\mathsf{G}_{\mathsf{net}} \not\geq \phi_{AB}$), we have the following verification relation:

$$\begin{aligned}&\exists\, \mathsf{c} \in \mathcal{C}, \mathsf{P} \in \mathcal{P} \mid \mathsf{in\ c.P} \mid \mathsf{G}_{\overline{\mathsf{AB}}} \ \rightarrow^* (\mathsf{G}'_{\overline{\mathsf{AB}}})_{\Downarrow},\ (\mathsf{G}_{\overline{\mathsf{AB}}})_{\Downarrow} \ \neq (\mathsf{G}'_{\overline{\mathsf{AB}}})_{\Downarrow} \\ &\wedge\, \mathsf{in\ c.P} \mid (\mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}])_{\Downarrow} \ \rightarrow^* (\mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}])_{\Downarrow} \ \Rightarrow \mathsf{G}_{\overline{\mathsf{AB}}} \not\geq \mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}]\end{aligned}$$

The bus topology is a special case. Since local policies are cascaded, adding a new layer of protection on top of the previous ones, compatibility is inherent to this topology. Therefore, compatibility propagates from the innermost firewall policy towards the peripheral ones, as stated in Proposition 3.4.1 below.

Proposition 3.4.1 *In a bus network topology, there are no incompatibilities between any local policy and the global firewall policy.*

Proof
The proof is done by induction on the number n of networks composing the bus, with $n \geq 3$. For $n = 3$, it is easy to see that the global policy of the rest of the network, $\mathsf{G}_{\overline{\mathsf{AB}}} = \mathsf{C}_{\phi_{\mathsf{BC}}}[\mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}]]$, contains the local policy $\mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}]$. Hence, the global policy $\mathsf{G}_{\overline{\mathsf{AB}}}$ cannot allow a traffic prohibited by the local policy implemented by FW_{AB}. Therefore, $\mathsf{G}_{\mathsf{net}} \geq \phi_{AB}$.

3.5 Case Study

In this section we present in detail how our calculus is used to specify and verify the example network illustrated in Figure 3.2. The network is composed of three subnetworks A, B and C, connected through the distributed

firewalls FW_{AB}, FW_{AC} and FW_{BC} respectively. An unfitting local security policy is implemented in one firewall (FW_{AC} in our case). We detect the filtering policy incompatibility by using our firewall policy calculus.

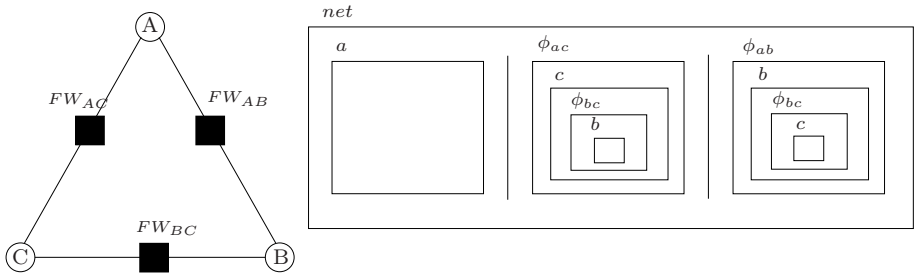


Figure 3.2: Example network with distributed firewalls

The ambient representation of the network from network A's perspective is also depicted in Figure 3.2. We denote by *net* the whole network and by a, b and c the subnetworks corresponding to A, B and C. The presence of sub-ambients c and b inside ambients b and c, respectively, indicates the alternate paths that processes can take in order to reach a.

The IP addresses range for nodes of network A is 132.156.0.1 - 132.156.255.255. For network B, the range is 132.155.21.1 - 132.155.60.255, and for C it is 132.155.0.1 - 132.155.20.255. An ftp server with the IP address 132.156.40.100 exists in A along with a www server with the IP address 132.155.10.100 in C. The local security policies for the three distributed firewalls are as follows. Firewall FW_{AC} allows traffic from port 80 of all hosts in A to the www server in C and blocks any traffic from C to A. Firewall FW_{AB} allows traffic from port 21 of hosts with src_ip = 132.155.*.* to port 21 of the ftp server in A and denies all traffic from A to B. Firewall FW_{BC} allows traffic from port 80 all hosts in B to the www server in C and also allows traffic from port 21 of hosts in C with src_ip = 132.155.*.* to port 21 of the host with src_ip = 132.156.40.100. Using the format recommended in [59] for firewall rules specification, we obtain the local firewall policies shown in Tables 3.5, 3.6 and 3.7.

Table 3.5: Local policy of firewall FW_{AC}

order	prot	src_ip	src_port	dest_ip	dest_port	act
1 :	tcp	132.156.0.1 − 132.156.255.255	80	132.155.10.100	80	accept
2 :	tcp	132.155.0.1 − 132.155.20.255	∗	132.156.0.1 − 132.156.255.255	∗	deny

Table 3.6: Local policy of firewall FW_{AB}

order	prot	src_ip	src_port	dest_ip	dest_port	act
1 :	tcp	132.155. ∗ .∗	21	132.156.40.100	21	accept
2 :	tcp	132.156.0.1 − 132.156.255.255	∗	132.155.21.1 − 132.155.60.255	∗	deny

Table 3.7: Local policy of firewall FW_{BC}

order	prot	src_ip	src_port	dest_ip	dest_port	act
1 :	tcp	132.155.21.1 − 132.155.60.255	80	132.155.10.100	80	accept
2 :	tcp	132.155. ∗ .∗	21	132.156.40.100	21	accept

The example network is a basic ring topology as in Figure 3.1 and therefore the topology specification is:

$$\mathsf{A} \mid \mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}] \mid \mathsf{B}_{\phi_{\mathsf{BC}}}[\mathsf{C}_{\phi_{\mathsf{AC}}}[\mathsf{A}] \mid \mathsf{C}_{\phi_{\mathsf{AC}}}[\mathsf{A}] \mid \mathsf{C}_{\phi_{\mathsf{BC}}}[\mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}]| \mathsf{A}_{\phi_{\mathsf{AC}}}[\mathsf{C}_{\phi_{\mathsf{BC}}}[\mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}]]$$
$$\mid \mathsf{A}_{\phi_{\mathsf{AB}}}[\mathsf{B}_{\phi_{\mathsf{BC}}}[\mathsf{C}_{\phi_{\mathsf{AC}}}[\mathsf{A}]]]$$

We use the notations a_{ij}^k and c_{ij}^k to denote the action and condition of rule number k of the local policies ϕ_{ij}, with $i \neq j$ and $i, j \in \{A, B, C\}$. This allows us to specify the local firewall policies using our calculus. For instance, the policy for the firewall FW_{AB} protecting network A from inbound traffic from network B has the is specified as:

$$\mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}] \;=\; {}^{\mathsf{a}^1_{\mathsf{AB}}}_{\mathsf{c}^1_{\mathsf{AB}}}[\mathsf{A}] \;\otimes\; {}^{\mathsf{a}^2_{\mathsf{AB}}}_{\mathsf{c}^2_{\mathsf{AB}}}[\mathsf{A}]$$

where $\mathsf{c}^1_{\mathsf{AB}}, \mathsf{a}^1_{\mathsf{AB}}, \mathsf{c}^2_{\mathsf{AB}}$ and $\mathsf{a}^2_{\mathsf{AB}}$ are obtained from Table 3.6:

$$\begin{aligned} \mathsf{c}^1_{\mathsf{AB}}: \quad & \mathsf{src\_ip} = 132.155.*.* \;\wedge\; \mathsf{src\_port} = 21 \\ & \wedge\; \mathsf{dest\_ip} = 132.156.40.100 \\ & \wedge\; \mathsf{dest\_port} = 21 \;\wedge\; \mathsf{prot} = \mathsf{tcp} \end{aligned}$$

$$\begin{aligned} \mathsf{c}^2_{\mathsf{AB}}: \quad & \mathsf{src\_ip} = 132.156.0.1 - 132.156.255.255 \;\wedge\; \mathsf{src\_port} = * \\ & \wedge\; \mathsf{dest\_ip} = 132.155.21.1 - 132.155.60.255 \\ & \wedge\; \mathsf{dest\_port} = * \wedge \mathsf{prot} = \mathsf{tcp} \end{aligned}$$

$$a^1_{AB} = \mathsf{accept};\; a^2_{AB} = \mathsf{deny}$$

The other two local policies can be specified in a similar manner using our calculus and the corresponding NIST format from Tables 3.5 and 3.7.

Proposition 3.5.1 *The local firewall policy ϕ_{AC} is not compatible with the global firewall policy $\mathsf{G}_{\mathsf{net}}$ of the whole network.*

Proof:
The formalism we developed in this chapter is used to verify that the network contains a security policy incompatibility. We prove this statement by choosing a particular packet that is denied direct access

to network A through firewall FW_{AC}, but can reach A by crossing FW_{BC} and FW_{AB}, as allowed by the global policy. Therefore, the global policy of the rest of the network is more permissive than the local policy implemented by FW_{AC} and transgresses it: $\mathsf{G}_{\overline{\mathsf{AC}}} \not\geq \mathsf{C}_{\phi_{\mathsf{AC}}}[\mathsf{A}]$.

We examine the local policy of firewall FW_{AC} and compare it with the global policy of the rest of the network. The point is to demonstrate that even though direct access for packets from C to A is denied, the transgressing global policy $\mathsf{G}_{\overline{\mathsf{AC}}}$ allows the process P from a particular packet $\mathsf{in\ c.P}$ to enter network A and be executed in parallel with the process A.

Let *in c.P* be the packet originating in network C, with c being a conjunction of the following predicates: $\mathsf{src\_ip} = 132.155.5.25$, $\mathsf{src\_port} = 21$, $\mathsf{dest\_ip} = 132.156.40.100$, $\mathsf{dest\_port} = 21$, and $\mathsf{prot} = \mathsf{tcp}$.

The expressions for $\mathsf{c}^1_{\mathsf{AC}}, \mathsf{a}^1_{\mathsf{AC}}, \mathsf{c}^2_{\mathsf{AC}}$ and $\mathsf{a}^2_{\mathsf{AB}}$ are obtained from Table 3.5:

$$\begin{aligned} \mathsf{c}^1_{\mathsf{AC}}: \quad & \mathsf{src\_ip} = 132.156.0.1 - 132.156.255.255 \wedge \mathsf{src\_port} = 80 \\ & \wedge\ \mathsf{dest\_ip} = 132.155.10.100 \wedge \mathsf{dest\_port} = 80 \wedge \mathsf{prot} = \mathsf{tcp} \end{aligned}$$

$$\begin{aligned} \mathsf{c}^2_{\mathsf{AC}}: \quad & \mathsf{src\_ip} = 132.155.0.1 - 132.155.20.255 \ \wedge\ \mathsf{src\_port} = * \\ & \wedge\ \mathsf{dest\_ip} = 132.156.0.1 - 132.156.255.255 \\ & \wedge\ \mathsf{dest\_port} = * \ \wedge\ \mathsf{prot} = \mathsf{tcp} \end{aligned}$$

$$a^1_{AC} = \mathsf{accept};\ a^2_{AC} = \mathsf{deny}$$

We compare the attributes of the conditions c, $\mathsf{c}^1_{\mathsf{AC}}$ and $\mathsf{c}^2_{\mathsf{AC}}$ and observe that $[\![\ \mathsf{c}\ ]\!] \cap [\![\ \mathsf{c}^1_{\mathsf{AC}}\ ]\!] = \emptyset$ and $[\![\ \mathsf{c}\ ]\!] \subseteq [\![\ \mathsf{c}^2_{\mathsf{AC}}\ ]\!]$. Then, the first rule does not apply. However, the action of the second rule is deny, so there is no reduction and no packet from C can enter A through FW_{AC}. In this case:

$$\mathsf{in\ c.P} \mid \begin{smallmatrix}\mathsf{a}^1_{\mathsf{AC}}\\ \mathsf{c}^1_{\mathsf{AC}}\end{smallmatrix}[\mathsf{A}] \ \rightarrow \mathsf{in\ c.P} \mid \begin{smallmatrix}\mathsf{a}^1_{\mathsf{AC}}\\ \mathsf{c}^1_{\mathsf{AC}}\end{smallmatrix}[\mathsf{A}]$$

Then, the process $\mathsf{C}_{\phi_{\mathsf{AC}}}[\mathsf{A}]$ is already in its normal form with respect to the reduction relation:

$$\mathsf{in\ c.P} \mid (\mathsf{C}_{\phi_{\mathsf{AC}}}[\mathsf{A}])_{\Downarrow} \ \rightarrow^* (\mathsf{C}_{\phi_{\mathsf{AC}}}[\mathsf{A}])_{\Downarrow} \qquad (8)$$

We now examine the network without the local firewall FW_{AC}. The global policy $\mathsf{G}_{\overline{\mathsf{AC}}}$ is obtained by removing all instances containing ϕ_{AC} from the topology specification:

$$\mathsf{G}_{\overline{\mathsf{AC}}} = \mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}] \mid \mathsf{C}_{\phi_{\mathsf{BC}}}[\mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}]] \qquad (9)$$

First, we need to check if the packet can at least reach inside network B. To do so, we execute it in parallel with the local policy protecting network B from traffic coming from C, $\mathsf{C}_{\phi_{\mathsf{BC}}}[\mathsf{B}]$:

$$\mathsf{in\ c.P} \mid \mathsf{C}_{\phi_{\mathsf{BC}}}[\mathsf{B}] = \mathsf{in\ c.P} \mid \begin{smallmatrix}\mathsf{a}^1_{\mathsf{BC}}\\ \mathsf{c}^1_{\mathsf{BC}}\end{smallmatrix}[\mathsf{B}] \otimes \begin{smallmatrix}\mathsf{a}^2_{\mathsf{BC}}\\ \mathsf{c}^2_{\mathsf{BC}}\end{smallmatrix}[\mathsf{B}]$$

For our packet, we have already specified c as:

$$\mathsf{c}: \ \mathsf{src\_ip} = 132.155.5.25 \ \wedge \ \mathsf{src\_port} = 21 \ \wedge \ \mathsf{dest\_ip} = 132.156.40.100 \ \wedge \ \mathsf{dest\_port} = 21 \ \wedge \ \mathsf{prot} = \mathsf{tcp}$$

We compare one by one the predicates of conditions c and c^1_{BC} and conclude that $[\![\ c\ ]\!] \subseteq [\![\ c^1_{BC}\ ]\!]$. In this case, the reduction rule (5) from Table 3.3 applies and we obtain:

$$\mathsf{in\ c.P} \mid \begin{smallmatrix}\mathsf{a}^1_{\mathsf{BC}}\\ \mathsf{c}^1_{\mathsf{BC}}\end{smallmatrix}[\mathsf{B}] \rightarrow \begin{smallmatrix}\mathsf{a}^1_{\mathsf{BC}}\\ \mathsf{c}^1_{\mathsf{BC}}\end{smallmatrix}[\mathsf{P} \mid \mathsf{B}]$$

Since rule 1 matches our packet, rule 2 can be ignored as it will have no effect on the packet. This means that the process P is able to enter network B, which is what we intended to verify as a first step. The reduction relation for the local policy becomes:

$$\mathsf{in\ c.P} \mid \mathsf{C}_{\phi_{\mathsf{BC}}}[\mathsf{B}] \rightarrow \begin{smallmatrix}\mathsf{a}^1_{\mathsf{BC}}\\ \mathsf{c}^1_{\mathsf{BC}}\end{smallmatrix}[\mathsf{P} \mid \mathsf{B}] \otimes \begin{smallmatrix}\mathsf{a}^2_{\mathsf{BC}}\\ \mathsf{c}^2_{\mathsf{BC}}\end{smallmatrix}[\mathsf{B}]$$

Next, we need to find out if our special packet can reach inside network A. In order to evaluate the capabilities our chosen packet with respect to the process A, we execute it in parallel with the local policy protecting network A, $\mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}]$:

$$\mathsf{in\ c.P} \mid \mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}] = \mathsf{in\ c.P} \mid \begin{smallmatrix}\mathsf{a}^1_{\mathsf{AB}}\\ \mathsf{c}^1_{\mathsf{AB}}\end{smallmatrix}[\mathsf{A}] \otimes \begin{smallmatrix}\mathsf{a}^2_{\mathsf{AB}}\\ \mathsf{c}^2_{\mathsf{AB}}\end{smallmatrix}[\mathsf{A}]$$

The comparison of the conditions c and c^1_{AB} reveals that $[\![\ c\ ]\!] \subseteq [\![\ c^1_{AB}\ ]\!]$. Again, the reduction rule (5) from Table 3.3 applies and we obtain:

$$\mathsf{in\ c.P} \mid \begin{smallmatrix}\mathsf{a}^1_{\mathsf{AB}}\\ \mathsf{c}^1_{\mathsf{AB}}\end{smallmatrix}[\mathsf{A}] \rightarrow \begin{smallmatrix}\mathsf{a}^1_{\mathsf{AB}}\\ \mathsf{c}^1_{\mathsf{AB}}\end{smallmatrix}[\mathsf{P} \mid \mathsf{A}]$$

This means that the process P is able to enter network A and execute inside in parallel with process A. Since rule 1 matches our packet, rule 2 can be ignored as it will have no effect on the packet. The reduction relation for the local policy becomes:

$$\mathsf{in\ c.P} \mid \mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}] \ \rightarrow\ {}^{\mathsf{a}^1_{\mathsf{AB}}}_{\mathsf{c}^1_{\mathsf{AB}}}[\mathsf{P} \mid \mathsf{A}] \ \otimes\ {}^{\mathsf{a}^2_{\mathsf{AB}}}_{\mathsf{c}^2_{\mathsf{AB}}}[\mathsf{A}] \qquad (10)$$

From (9) and (10) we obtain:

$$\mathsf{in\ c.P} \mid \mathsf{G}_{\overline{\mathsf{AC}}} \ \rightarrow\ (\ {}^{\mathsf{a}^1_{\mathsf{AB}}}_{\mathsf{c}^1_{\mathsf{AB}}}[\mathsf{P} \mid \mathsf{A}] \ \otimes\ {}^{\mathsf{a}^2_{\mathsf{AB}}}_{\mathsf{c}^2_{\mathsf{AB}}}[\mathsf{A}]\ ) \mid \mathsf{C}_{\phi_{\mathsf{BC}}}[\mathsf{B}_{\phi_{\mathsf{AB}}}[\mathsf{A}]] \qquad (11)$$

Then, the global firewall policy $\mathsf{G}_{\overline{\mathsf{AC}}}$ allows P to enter A and it transgresses the local policy FW_{AC}. At this point, taking into account expressions (8) and (11) and our verification relation from Section 3.4, we state that the local firewall FW_{AC} is not compatible with the global policy of the distributed firewall configuration:

$$\mathsf{G}_{\overline{\mathsf{AC}}} \not\geq \mathsf{C}_{\phi_{\mathsf{AC}}}[\mathsf{A}]$$

The problem can be solved either by opening FW_{AC} to accept some traffic from C to A (local policy change) or by blocking packets from C in the other firewalls (global policy change). □

3.6 Conclusion

In this chapter we have presented a new approach for specifying and verifying distributed security policies implemented in distributed firewalls. Our contributions include the formal syntactic and semantic definitions of a new description formalism inspired from the ambient calculus and a formal reasoning framework for distributed firewall verification. We demonstrate by a case study how our new calculus can be used to address the problem of conflict detection in distributed firewalls and how it facilitates the analysis of the effect that network topologies have on distributed firewall policies. The point of our research is to identify, where applicable, how legitimate packets can be stopped from reaching their destination and potentially harmful packets are able to get through. The new concepts of order and compatibility allow us to compare policies. For instance, using these concepts, we proved that no conflicts can arise between local policies in a bus topology network.

Further development of the work presented here can be considered. Refinements in firewall policy specification could allow the elimination of intra-firewall anomalies. Also, the extraction of the appropriate local policies from a given global policy seems to be a complementary extension of this work. In addition, dynamic reconfiguration of firewall policies based on topology changes would be an exciting prospect.

Chapter 4

Intruder Oriented Security Behavior Analysis of Computer Systems

Abstract

This chapter proposes a formal approach for specifying and verifying computer systems security behavior. Using our methodology, systems and their interactions are modeled through processes with a new dedicated calculus inspired from the ambient calculus. We demonstrate how, given a network security policy implementation, our dedicated calculus allows to verify that the specification offers or not sufficient protection from a malicious intruder.

4.1 Introduction

Proper implementation of a security policy has always been a crucial step in providing the level of protection required for a certain computer system or network. There is no shortage of effective implementations of such specifications. However, due to the abstractions inherent to the very high level of those specifications, there is often a significant mismatch between what was intended and the actual result of the applied policy. Several factors come into play when defining constraints, including the depth of the knowledge of the systems, their complexity, architectural changes etc. In many cases the compound effect of permissions and restrictions amounts to unforeseen sce-

narios that can jeopardize the overall security of the system. We propose a formal technique for computer systems specification and verification. A new dedicated calculus is defined for the purpose of system specification. Our calculus builds on the basic concepts of the ambient calculus. Our work focuses on the issues of modeling the system's behavior and checking if the system satisfies a given security policy. This is only the first phase of a comprehensive approach towards guaranteed policy compliance. It consists in the development of a specification language that is powerful enough to express the dynamic behavior of system security at both system and network levels.

The remainder of this chapter is organized as follows. Section 4.2 presents our new calculus suited for system and network specification. The case study, depicted in section 4.3, illustrates how our calculus can be applied both to specify a network and to correct an inadequate implementation of a security policy. Section 4.4 concludes this chapter and discusses an envisioned extension of this work.

4.2 Computer Systems Security Specification

In this section we define the syntax and semantics of a calculus suited to specify, at an abstract level, a given network with the behaviors of network components, including network protection through security policies and the behavior of a malicious intruder.

4.2.1 Syntax

We present in Table 4.1 the syntax of our calculus, which we call *Security Specification Calculus* (or *SSC*, in short). Contrary to the one of Gordon[21], our syntax allows the specification of resources, protection, rights, and control policies. Moreover, it is capable to describe the interaction with a potential intruder. We model these components in terms of processes and process interaction.

Let $\mathcal{N}$ be a set of names, $\mathcal{K}$ be a set of keys and $\mathcal{X}$ be a set of variables. The following sub-categories can be processes:

- **inactivity**: 0 is a process that does nothing;

- **parallel composition**: P | Q refers to parallel execution of processes P and Q; it is a commutative and associative operator;
- **replication**: The process !P denotes the unbounded replication of the process P;
- **ambient**: $\overset{k_e,k_s}{n}$[P] denotes an ambient called n containing a resource (specified by the process P) protected by the keys necessary for entering or exiting from the protected ambient in order to interact with the resource.
- **action**: a.P depicts the sequential behavior of a process as a sequence: it first uses its a capability and than behaves like P.

The following sub-categories refer to process capabilities:

- **movement**: $\texttt{mov}_n^k$ refers to the capability of a process to move using the appropriate ambient key. Depending on the value of the key k, there are two types of movements: an access movement and an exit movement. The access movement allows process to enter an ambient n protected by the access key k_e. The exit movement allows a process to exit from an ambient n protected by the exit key k_s;
- **key request** : $\texttt{req}_{n,t}^x$ allows a process to make a request for an access key k_e or an exit key k_s dépending on the value of the parameter t;
- **exploration**: exp allows representation of dynamic processes that can non-deterministically choose what their next action is;
- **key publication**: $\texttt{pub}_{n,t}^k$ refers to the publication of the access or exit key of an ambient depending on the value of the parameter t;
- **compositional choice**: $\texttt{a} \oplus \texttt{b}$ refers to the choice operator that allows the future evolution of a process to be defined as a choice involving all possible combination of capabilities a and b of a process $(\texttt{a} \oplus \texttt{b})$.P;
- **nondeterministic choice**: $\texttt{a} \sqcap \texttt{b}$ refers to a choice operator that allows the future evolution of a process to be defined as a choice involving two component capabilities of a process $(\texttt{a} \sqcap \texttt{b})$.P, but does not allow the environment any control over which capability will be selected.

Table 4.1: Syntax of *SSC*

n	$\in$	$\mathcal{N}$	**name**
x	$\in$	$\mathcal{X}$	**variable**
k	$\in$	$\mathcal{K}$	**key**
t	$\in$	$\{e, s\}$	**type of key**
P, Q	$::=$		**processes**
		0	**inactivity**
	$\mid$	$\mathsf{P} \mid \mathsf{Q}$	**parallel composition**
	$\mid$	$!\mathsf{P}$	**replication**
	$\mid$	$\overset{k_e,k_s}{n}[\mathsf{P}]$	**ambient**
	$\mid$	$\mathtt{a}.\mathsf{P}$	**sequence action**
$\mathtt{a}, \mathtt{b}$	$::=$		**process capabilities**
	$\mid$	$\mathtt{mov}_n^k$	**movement**
	$\mid$	$\mathtt{pub}_{n,t}^k$	**key publication**
	$\mid$	$\mathtt{req}_{n,t}^x$	**request key**
	$\mid$	$\mathtt{exp}$	**exploration**
	$\mid$	$\mathtt{a} \oplus \mathtt{b}$	**compositional choice**
	$\mid$	$\mathtt{a} \sqcap \mathtt{b}$	**nondeterministic choice**

In our approach an ambient has a name and it is protected by access and exit keys (k_e and k_s) that denote controlled access to resources. Therefore, processes must know the appropriate key in order to be allowed to enter an ambient that contains the resource it seeks. The permissions assigned to a processes reside in its movement capabilities (namely `mov`). A process will take the action invoked by such capabilities in order to continue its execution. The `exp` operator is a dedicated construct that mimics an intruder's behavior. The behavior of the intruder is modelled by the set of capabilities that it can acquire through exploration and use whenever it pleases (i.e. nondeterministic). This is what differentiates legitimate, regular processes from the one representing the intruder. However, the fact that a process is able to access a protected ambient does not automatically translate into an exit capability. In many cases, a process will cease to move around once it reaches its final destination. This possible process behavior is reflected by the requirement for defining distinctive access and exit keys. Security (or control) policies are responsible for granting the rights to use resources through key publication.

Our formal calculus can be used to specify an abstraction of both the internal (inter-process) and the external (inter-system) physical connections. For instance, direct links between computers are simply assimilated to a supplemental level of protection for the target computer. This corresponds in practice to permissions from both ends to use the link. A process running inside computer A attempting to reach another process inside computer B will need to exit its home ambient and be allowed by B's ambient to enter. Our calculus is also suitable for representing complex network devices such as routers and switches, that can have their own intricate access policies.

The fact that processes are represented sequentially in our model means that the very first action has to be taken before the next one is executed. If, for any reason, the action cannot be completed, the process is blocked until the right conditions exist to allow it to continue. Some processes have a predictable behavior in the sense that their sequence of actions is predefined. For instance, the process $\texttt{mov}_n^{k_e}.0$ will attempt to enter the ambient n protected by the key k_e and then stops. Moreover, the `exp` operator has been introduced in order to capture the behavior of dynamic processes like the intruder, for instance, which is not forced to execute a specific action, and

can instead carefully plan his next move for maximum gain as he attempts to build a complex attacks. Such processes have a different behavior depending on their interaction with the rest of the system. We detail this aspect in section 4.2.2, dedicated to the semantics of our calculus.

4.2.2 Semantics

In this section we present the operational semantics of our calculus. It consists of the definition of a structural congruence between processes, denoted by $\equiv$, and a reduction relation, denoted by $\rightarrow$. The structural congruence is defined in Table 4.2 and illustrates process equivalence in the context of our calculus. Properties such as reflexivity, symmetry, transitivity, parallelism, commutativity and associativity are quite obvious. The capabilities are also straightforward. Replication is used for expressing iterations and recursion. The latter is important for describing, for instance, that a certain process attempts repeatedly to access a protected domain.

The reduction relations defined in Table 4.3 captures the process behavior depending on the process capabilities and the configuration in term of protections of its environment. Rules (1) and (2) are trivial. Rule (3) depicts the fact that, given the right capability, a processes can enter a protected ambient. Ability to exit a protected ambient is presented by rule (4). A request for the proper key from the appropriate publication service resulting in the communication of the key is captured by rule (5). The key is not automatically used by a `mov` form in the process. Processes can know a key and never use it. If, at some points in its execution thread, an action calls for that particular key, it is available to the processes and it will be used. Rule (6) describes the behavior of the intruder. It captures the way that the intruder process can, at will, use an access or exit movement if it leads to any further advantage. This relation adds to the intruder's knowledge and gives him new movement capabilities. The expression $(\mathtt{mov}_n^{k_t} \oplus \mathtt{exp})$ from rule(6) can take one of four values, as described by the (**Compositional Choice**) expression from the structural congruence relation presented in Table 4.2. Further combined with the (**Execution Choice**) expression from the same table, we have:

$$(\mathtt{mov}_n^{k_t} \oplus \mathtt{exp}).\mathsf{P} \equiv (\mathtt{mov}_n^{k_t}.\mathsf{P}) \sqcap (\mathtt{exp}.\mathsf{P}) \sqcap (\mathtt{mov}_n^{k_t}.\mathtt{exp}.\mathsf{P}) \sqcap (\mathtt{exp}.\mathtt{mov}_n^{k_t}.\mathsf{P})$$

Table 4.2: Structural Congruence

$\mathsf{P} \equiv \mathsf{P}$	(**Reflexivity**)
$\mathsf{P} \equiv \mathsf{Q} \Rightarrow \mathsf{Q} \equiv \mathsf{P}$	(**Symmetry**)
$\mathsf{P} \equiv \mathsf{Q} \wedge \mathsf{Q} \equiv \mathsf{R} \Rightarrow \mathsf{P} \equiv \mathsf{R}$	(**Transitivity**)
$\mathsf{P} \equiv \mathsf{Q} \Rightarrow \mathsf{P} \mid \mathsf{R} \equiv \mathsf{Q} \mid \mathsf{R}$	(**Parallelism**)
$\mathsf{P} \equiv \mathsf{Q} \Rightarrow {}^{k_e,k_s}_{n}[\mathsf{P}] \equiv {}^{k_e,k_s}_{n}[\mathsf{Q}]$	(**Ambient**)
$\mathsf{P} \equiv \mathsf{Q} \Rightarrow \mathsf{a}.\mathsf{P} \equiv \mathsf{a}.\mathsf{Q}$	(**New Capability**)
$\mathsf{P} \mid 0 \equiv \mathsf{P}$	(**Zero Parallelism**)
$(\mathsf{P} \mid \mathsf{Q}) \mid \mathsf{R} \equiv \mathsf{P} \mid (\mathsf{Q} \mid \mathsf{R})$	(**Associativity**)
$\mathsf{P} \equiv \mathsf{Q} \Rightarrow !\mathsf{P} \equiv !\mathsf{Q}$	(**Replication**)
$!0 \equiv 0$	(**Zero Replication**)
$(\mathsf{a} \oplus \mathsf{b}).\mathsf{P} \equiv \mathsf{a}.\mathsf{P} \sqcap \mathsf{b}.\mathsf{P} \sqcap \mathsf{a}.\mathsf{b}.\mathsf{P} \sqcap \mathsf{b}.\mathsf{a}.\mathsf{P}$	(**Compositional Choice**)
$(\mathsf{a} \oplus \mathsf{a}).\mathsf{P} \equiv \mathsf{a}.\mathsf{P}$	(**Idempotence**)
$(\mathsf{a} \oplus \mathsf{b}).\mathsf{P} \equiv (\mathsf{b} \oplus \mathsf{a}).\mathsf{P}$	(**Commutativity Comp Choice**)
$((\mathsf{a} \oplus \mathsf{b}) \oplus \mathsf{c}).\mathsf{P} \equiv (\mathsf{a} \oplus (\mathsf{b} \oplus \mathsf{c})).\mathsf{P}$	(**Associativity Comp Choice**)
$\mathsf{a}.\mathsf{P} \sqcap \mathsf{b}.\mathsf{Q} \equiv \mathsf{b}.\mathsf{Q} \sqcap \mathsf{a}.\mathsf{P}$	(**Commutativity Nondst Choice**)
$(\mathsf{a}.\mathsf{P} \sqcap \mathsf{b}.\mathsf{Q}) \sqcap \mathsf{c}.\mathsf{R} \equiv \mathsf{a}.\mathsf{P} \sqcap (\mathsf{b}.\mathsf{Q} \sqcap \mathsf{c}.\mathsf{R})$	(**Associativity Nondst Choice**)
$(\mathsf{a} \sqcap \mathsf{b}).\mathsf{P} \equiv \mathsf{a}.\mathsf{P} \sqcap \mathsf{b}.\mathsf{P}$	(**Execution Choice**)

Therefore, the intruder has a choice to do one of the following:

- stop exploring and use the newly acquired key as its next move
- ignore the key acquired and continue exploring
- use the newly acquired key as its next move and then continue exploring
- continue to explore and use the key at a future moment.

If the intruder chooses, for instance, the third option ($\texttt{mov}_n^{k_t}.\texttt{exp}.\mathsf{P}$), then the result is a movement either inside or outside of an ambient n, depending wether t takes the value e or s. This is illustrated in Fig.4.1.

If there is no publication service running in parallel with the intruder or if he already gained all keys from the services available, he will further look into accessing protected ambients for which he has the key.

4.3 Case Study

The subject of our case study is a simple system consisting of three computers connected through a router: one workstation (A) and two servers (B and C), all with their own security policies implemented. An SSH client is running on A. An SSH server is running on B, along with an FTP client. An FTP server and a database reside on C. The implemented security policies provide protection with a pair of keys for each machine. Access to A, protected by the access key k_1, is governed by the authentication mechanism built in the operating system running on the workstation. The exit key k_2 represents the only allowed way to exit from A by using the SSH client. The access key k_3 summarizes the conditions needed for being allowed to enter B: knowledge of the address of the SSH server on B and the proper credentials to login (a valid username/password combination). The exit key of A and the access key for B are known to all processes originating in A. In order to exit from B, processes have to have permissions to use the FTP client, represented through the exit key k_4. Access to C is granted to processes aware of the key k_5, which gathers the knowledge needed to access the FTP server it hosts: server address, user name and password. Only some of the

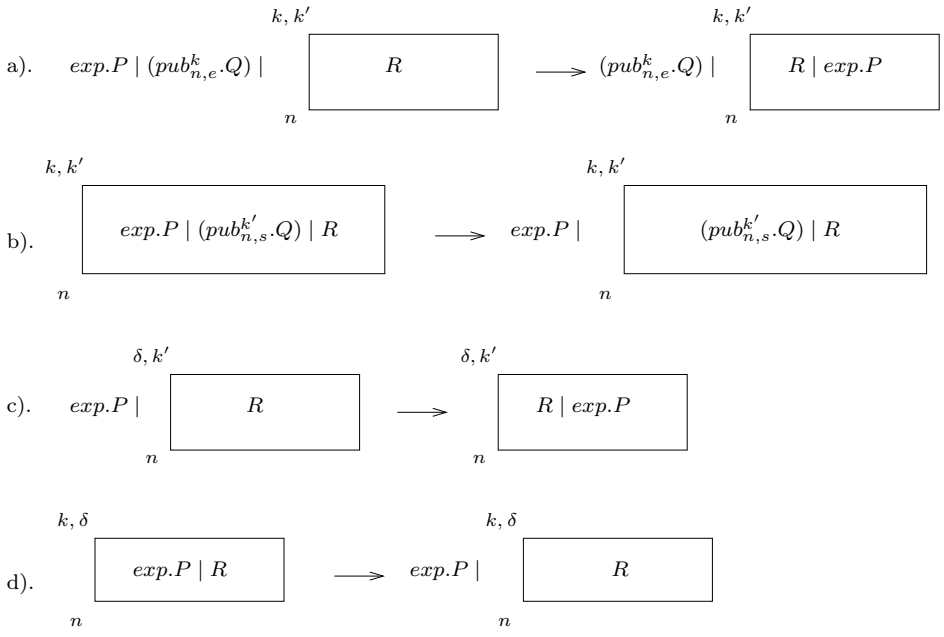


Figure 4.1: Intruder Network Exploration Capabilities

Table 4.3: Reduction Relations

$$\mathsf{P}' \to \mathsf{Q}' \qquad \text{if } \mathsf{P}' \equiv \mathsf{P}, \mathsf{P} \to \mathsf{Q}, \mathsf{Q} \equiv \mathsf{Q}' \quad (1)$$

$$\mathsf{P} \mid \mathsf{R} \to \mathsf{Q} \mid \mathsf{R} \qquad \text{if } \mathsf{P} \to \mathsf{Q} \quad (2)$$

$$\mathtt{mov}_n^{k_e}.\mathsf{P} \mid {}_n^{k_e,k_s}[\mathsf{Q}] \to {}_n^{k_e,k_s}[\mathsf{P} \mid \mathsf{Q}] \quad (3)$$

$${}_n^{k_e,k_s}[\mathtt{mov}_n^{k_s}.\mathsf{P} \mid \mathsf{Q}] \to \mathsf{P} \mid {}_n^{k_e,k_s}[\mathsf{Q}] \quad (4)$$

$$\mathtt{req}_{n,t}^{x}.\mathsf{P} \mid (\mathtt{pub}_{n,t}^{k_t}.\mathsf{Q}) \to \mathsf{P}[x \leftarrow k_t] \mid (\mathtt{pub}_{n,t}^{k_t}.\mathsf{Q}) \quad (5)$$

$$\mathtt{exp}.\mathsf{P} \mid (\mathtt{pub}_{n,t}^{k_t}.\mathsf{Q}) \to (\mathtt{mov}_n^{k_t} \oplus \mathtt{exp}).\mathsf{P} \mid (\mathtt{pub}_{n,t}^{k_t}.\mathsf{Q}) \quad (6)$$

processes from B, including those started with *root* privileges, have such permissions. The access to database DB does not require any special privilege and is available to all users authenticated by server C. Finally, the exit key k_6 is granted automatically to all processes that attempt to exit from C. However, some processes bear no significance for our example. In order to simplify our presentation, we will represent all those uninteresting processes by one abstract process per machine. We will now focus on those that are essential for our purpose. There are two key servers running on B, depicted as key publication services inside the protected ambient b. No direct access to the ambient c is given to processes from the ambient a, meaning there is no publication of the access key k_5 of c in a.
Several processes are running inside each machine at any given moment and they are denoted by a global process for each machine: A, B and C respectively for A, B and C. This is illustrated by the following expression:

$$\overset{k_1,k_2}{a}[\mathsf{A}] \mid \overset{k_3,k_4}{b}[\mathsf{B}] \mid \overset{k_5,k_6}{c}[\mathsf{C}]$$

The process A is viewed as the composition of two sub process M and A', i.e.: $\mathsf{A} \equiv \mathsf{M} \mid \mathsf{A}'$. The process M denotes a regular process running on machine A whereas A' represent an abstraction of the rest of the processes running on A. The process C in C is the composition of two processes T and C', where T represents a database of sensitive information that should only be available for some processes from B. This process is selected for evaluating if the regular and intruder processes can access the database or if they comply with the security policy that intends to deny access to that data to all users from A. The same observation also applies to B and its two key publication servers: $\mathsf{B} \equiv !(\mathtt{pub}^{k_4}_{b,s}.0) \mid !(\mathtt{pub}^{k_5}_{c,e}.0) \mid \mathsf{B}'$. The publication of the key k_5 captures the documented SSH vulnerability that gives *root* access to authenticated SSH users.

For this case study, we show how the intruder can enter c and execute some actions in parallel with T. This clearly corresponds to a violation of the implemented security policy.

The system specification is described by the following process:

$$\overset{k_1,k_2}{a}[\mathsf{M} \mid \mathsf{A}'] \mid \overset{k_3,k_4}{b}[!(\mathtt{pub}^{k_4}_{b,s}.0) \mid !(\mathtt{pub}^{k_5}_{c,e}.0) \mid \mathsf{B}'] \mid \overset{k_5,k_6}{c}[\mathsf{T} \mid \mathsf{C}']$$

4.3.1 Regular Process Behavior

Regular processes are executed by a normal (or regular) user. Since their purpose is always unambiguous, they have a predefined structure that states the order in which different actions will take place. Let M be a regular process executed on computer A, with the following predefined structure:

$$\mathsf{M} \;=\; \mathtt{mov}_a^{k_2}.\mathtt{mov}_b^{k_3}.\mathtt{req}_{b,s}^{x}.\mathtt{req}_{c,e}^{y}.\mathtt{mov}_b^{x}.\mathtt{mov}_c^{y}.\mathsf{M}'$$

This structure will allow the process, as demonstrated here, to exit from a, enter b, acquire two keys that it will further use for exiting b and entering c and then behave like M'. The process M' abstracts the rest of the behavior of M after reaching the server C. The example specification that takes into account this particular format of M is:

$$\begin{array}{c} {}^{k_1,k_2}_{a}[\mathtt{mov}_a^{k_2}.\mathtt{mov}_b^{k_3}.\mathtt{req}_{b,s}^{x}.\mathtt{req}_{c,e}^{y}.\mathtt{mov}_b^{x}.\mathtt{mov}_c^{y}.\mathsf{M'} \mid \mathsf{A'}] \;\mid\; {}^{k_3,k_4}_{b}[!(\mathtt{pub}_{b,s}^{k_4}.0) \;\mid \\ !(\mathtt{pub}_{c,e}^{k_5}.0) \;\mid\; \mathsf{B'}] \;\mid\; {}^{k_5,k_6}_{c}[\mathsf{T} \mid \mathsf{C'}] \end{array}$$

The application of reduction rules from Table 4.3 allows the process to execute the actions shown in Table 4.4 and lead to the parallel execution of $\mathsf{M'}$ and T.

4.3.2 Intruder

We apply the same technique for a process I representing the intruder. The `exp` operator will help the intruder to explore the system and gain capabilities in term of `mov` actions. The `exp` operator is a dedicated syntactic construct that mimics a non deterministic intruder's behavior. As stated in the introduction to this example, all processes originating in A, including the intruder's process, know the keys k_2 of the ambient a and k_3 of the ambient b. The intruder has an option to use those keys at will and start his exploration. He will gain further movement capacities from the two key publication servers (without making requests, contrary to the regular process).

Table 4.4: Reduction Relations for the Regular Process

$$
\begin{aligned}
&{}^{k_1,k_2}_{a}[\mathtt{mov}^{k_2}_{a}.\mathtt{mov}^{k_3}_{b}.\mathtt{req}^{x}_{b,s}.\mathtt{req}^{y}_{c,e}.\mathtt{mov}^{x}_{b}.\mathtt{mov}^{y}_{c}.\mathsf{M'} \mid \mathsf{A'}] \mid {}^{k_3,k_4}_{b}[!(\mathtt{pub}^{k_4}_{b,s}.0) \mid !(\mathtt{pub}^{k_5}_{c,e}.0) \mid \mathsf{B'}] \mid {}^{k_5,k_6}_{c}[\mathsf{T} \mid \mathsf{C'}] \\
\rightarrow\ &{}^{k_1,k_2}_{a}[\mathsf{A'}] \mid \mathtt{mov}^{k_3}_{b}.\mathtt{req}^{x}_{b,s}.\mathtt{req}^{y}_{c,e}.\mathtt{mov}^{x}_{b}.\mathtt{mov}^{y}_{c}.\mathsf{M'} \mid {}^{k_3,k_4}_{b}[!(\mathtt{pub}^{k_4}_{b,s}.0) \mid !(\mathtt{pub}^{k_5}_{c,e}.0) \mid \mathsf{B'}] \mid {}^{k_5,k_6}_{c}[\mathsf{T} \mid \mathsf{C'}] \\
\rightarrow\ &{}^{k_1,k_2}_{a}[\mathsf{A'}] \mid {}^{k_3,k_4}_{b}[\mathtt{req}^{x}_{b,s}.\mathtt{req}^{y}_{c,e}.\mathtt{mov}^{x}_{b}.\mathtt{mov}^{y}_{c}.\mathsf{M'} \mid !(\mathtt{pub}^{k_4}_{b,s}.0) \mid !(\mathtt{pub}^{k_5}_{c,e}.0) \mid \mathsf{B'}] \mid {}^{k_5,k_6}_{c}[\mathsf{T} \mid \mathsf{C'}] \\
\rightarrow\ &{}^{k_1,k_2}_{a}[\mathsf{A'}] \mid {}^{k_3,k_4}_{b}[\mathtt{req}^{y}_{c,e}.\mathtt{mov}^{k_4}_{b}.\mathtt{mov}^{y}_{c}.\mathsf{M'} \mid !(\mathtt{pub}^{k_4}_{b,s}.0) \mid !(\mathtt{pub}^{k_5}_{c,e}.0) \mid \mathsf{B'}] \mid {}^{k_5,k_6}_{c}[\mathsf{T} \mid \mathsf{C'}] \\
\rightarrow\ &{}^{k_1,k_2}_{a}[\mathsf{A'}] \mid {}^{k_3,k_4}_{b}[\mathtt{mov}^{k_4}_{b}.\mathtt{mov}^{k_5}_{c}.\mathsf{M'} \mid !(\mathtt{pub}^{k_4}_{b,s}.0) \mid !(\mathtt{pub}^{k_5}_{c,e}.0) \mid \mathsf{B'}] \mid {}^{k_5,k_6}_{c}[\mathsf{T} \mid \mathsf{C'}] \\
\rightarrow\ &{}^{k_1,k_2}_{a}[\mathsf{A'}] \mid {}^{k_3,k_4}_{b}[!(\mathtt{pub}^{k_4}_{b,s}.0) \mid !(\mathtt{pub}^{k_5}_{c,e}.0) \mid \mathsf{B'}] \mid \mathtt{mov}^{k_5}_{c}.\mathsf{M'} \mid {}^{k_5,k_6}_{c}[\mathsf{T} \mid \mathsf{C'}] \\
\rightarrow\ &{}^{k_1,k_2}_{a}[\mathsf{A'}] \mid {}^{k_3,k_4}_{b}[!(\mathtt{pub}^{k_4}_{b,s}.0) \mid !(\mathtt{pub}^{k_5}_{c,e}.0) \mid \mathsf{B'}] \mid {}^{k_5,k_6}_{c}[\mathsf{M'} \mid \mathsf{T} \mid \mathsf{C'}]
\end{aligned}
$$

The initial knowledge of the two keys is equivalent to keys publication and is illustrated by the following expression:

$$\mathsf{I} \;=\; (\mathtt{mov}_a^{k_2} \;\oplus\; \mathtt{mov}_b^{k_3} \;\oplus\; \mathtt{exp}).\mathsf{I'}$$

There are four possible behaviors of the intruder introduced by the $\oplus$ operator, but only one that is useful in his attack, namely $\mathtt{mov}_a^{k_2}.\mathtt{mov}_b^{k_3}.\mathtt{exp}.\mathsf{I'}$. The other three are ignored for the following reasons: two of them paralyses the attack (those prefixed by $\mathtt{mov}_b^{k_3}$, the process has to exit a before it can enter b). The third ($\mathtt{mov}_a^{k_2}.\mathsf{I'}$) is already implied by the choice made. Similarly to the case of the regular process M, we say that there is a process $\mathsf{A''}$so that $\mathsf{A} \equiv \mathsf{I} \mid \mathsf{A''}$. The network specification becomes:

$$_a^{k_1,k_2}[\mathtt{mov}_a^{k_2}.\mathtt{mov}_b^{k_3}.\mathtt{exp}.\mathsf{I'} \mid \mathsf{A''}] \;\mid\; _b^{k_3,k_4}[!(\mathtt{pub}_{b,s}^{k_4}.0) \mid !(\mathtt{pub}_{c,e}^{k_5}.0) \mid \mathsf{B'}] \mid\; _c^{k_5,k_6}[\mathsf{T} \mid \mathsf{C'}]$$

The application of intruder reduction rules from section 6.3.2 allows the intruder to exit a and enter b, as illustrated in Table 4.5.

Table 4.5: Reduction Relations for the Intruder - From A to B

$$_a^{k_1,k_2}[\mathtt{mov}_a^{k_2}.\mathtt{mov}_b^{k_3}.\mathtt{exp}.\mathsf{I'} \mid \mathsf{A''}] \mid\; _b^{k_3,k_4}[!(\mathtt{pub}_{b,s}^{k_4}.0) \mid !(\mathtt{pub}_{c,e}^{k_5}.0) \mid \mathsf{B'}] \mid\; _c^{k_5,k_6}[\mathsf{T} \mid \mathsf{C'}]$$
$$\rightarrow\; _a^{k_1,k_2}[\mathsf{A''}] \mid \mathtt{mov}_b^{k_3}.\mathtt{exp}.\mathsf{I'} \mid\; _b^{k_3,k_4}[!(\mathtt{pub}_{b,s}^{k_4}.0) \mid !(\mathtt{pub}_{c,e}^{k_5}.0) \mid \mathsf{B'}] \;\mid\; _c^{k_5,k_6}[\mathsf{T} \mid \mathsf{C'}]$$
$$\rightarrow\; _a^{k_1,k_2}[\mathsf{A''}] \mid\; _b^{k_3,k_4}[\mathtt{exp}.\mathsf{I'} \mid !(\mathtt{pub}_{b,s}^{k_4}.0) \mid !(\mathtt{pub}_{c,e}^{k_5}.0) \mid\; \mathsf{B'}] \mid\; _c^{k_5,k_6}[\mathsf{T} \mid \mathsf{C'}]$$

The intruder finds itself in front of two vulnerable services that provide two more keys. The intruder reduction relations table presents two possible alternatives for the exploration paths and the reduced expressions are presented in Table 4.6.

Table 4.6: Intruder Alternatives Inside B

Case 1:

$\rightarrow {}^{k_1,k_2}_{a}[\mathsf{A"}] \mid {}^{k_3,k_4}_{b}[(\mathtt{mov}^{k_4}_{b} \oplus \mathtt{exp}).\mathsf{I'} \mid !(\mathtt{pub}^{k_4}_{b,s}.0) \mid !(\mathtt{pub}^{k_5}_{c,e}.0) \mid \mathsf{B'}] \mid {}^{k_5,k_6}_{c}[\mathsf{T} \mid \mathsf{C'}]$

where $(\mathtt{mov}^{k_4}_{b} \oplus \mathtt{exp}).\mathsf{I'} \equiv \mathtt{mov}^{k_4}_{b}.\mathsf{I'} \sqcap \mathtt{exp}.\mathsf{I'} \sqcap \mathtt{mov}^{k_4}_{b}.\mathtt{exp}.\mathsf{I'} \sqcap \mathtt{exp}.\mathtt{mov}^{k_4}_{b}.\mathsf{I'}$

Case 2:

$\rightarrow {}^{k_1,k_2}_{a}[\mathsf{A"}] \mid {}^{k_3,k_4}_{b}[(\mathtt{mov}^{k_5}_{c} \oplus \mathtt{exp}).\mathsf{I'} \mid !(\mathtt{pub}^{k_4}_{b,s}.0) \mid !(\mathtt{pub}^{k_5}_{c,e}.0) \mid \mathsf{B'}] \mid {}^{k_5,k_6}_{c}[\mathsf{T} \mid \mathsf{C'}]$

where $(\mathtt{mov}^{k_5}_{c} \oplus \mathtt{exp}).\mathsf{I'} \equiv \mathtt{mov}^{k_5}_{c}.\mathsf{I'} \sqcap \mathtt{exp}.\mathsf{I'} \sqcap \mathtt{mov}^{k_5}_{c}.\mathtt{exp}.\mathsf{I'} \sqcap \mathtt{exp}.\mathtt{mov}^{k_5}_{c}.\mathsf{I'}$

Table 4.7: Intruder's Choice

$\rightarrow {}^{k_1,k_2}_{a}[\mathsf{A"}] \mid {}^{k_3,k_4}_{b}[\mathtt{mov}^{k_4}_{b}.\mathtt{mov}^{k_5}_{c}.\mathsf{I'} \mid !(\mathtt{pub}^{k_4}_{b,s}.0) \mid !(\mathtt{pub}^{k_5}_{c,e}.0) \mid \mathsf{B'}] \mid {}^{k_5,k_6}_{c}[\mathsf{T} \mid \mathsf{C'}]$

$\rightarrow {}^{k_1,k_2}_{a}[\mathsf{A"}] \mid {}^{k_3,k_4}_{b}[!(\mathtt{pub}^{k_4}_{b,s}.0) \mid !(\mathtt{pub}^{k_5}_{c,e}.0) \mid \mathsf{B'}] \mid \mathtt{mov}^{k_5}_{c}.\mathsf{I'} \mid {}^{k_5,k_6}_{c}[\mathsf{T} \mid \mathsf{C'}]$

$\rightarrow {}^{k_1,k_2}_{a}[\mathsf{A"}] \mid {}^{k_3,k_4}_{b}[!(\mathtt{pub}^{k_4}_{b,s}.0) \mid !(\mathtt{pub}^{k_5}_{c,e}.0) \mid \mathsf{B'}] \mid {}^{k_5,k_6}_{c}[\mathsf{I'} \mid \mathsf{T} \mid \mathsf{C'}]$

However, both cases lead to the same end result, since the next exploration step will bring the other move capability for the intruder and the next reduction is:

$$\rightarrow {}^{k_1,k_2}_{a}[\mathsf{A''}] \mid {}^{k_3,k_4}_{b}[(\mathtt{mov}^{k_4}_{b} \oplus \mathtt{mov}^{k_5}_{c} \oplus \mathtt{exp}).\mathsf{I'} \mid !(\mathtt{pub}^{k_4}_{b,s}.0) \mid$$
$$!(\mathtt{pub}^{k_5}_{c,e}.0) \mid \mathsf{B'}] \mid {}^{k_5,k_6}_{c}[\mathsf{T} \mid \mathsf{C'}]$$

Within the ten possibilities arising from the new computational choice $(\mathtt{mov}^{k_4}_{b} \oplus \mathtt{mov}^{k_5}_{c} \oplus \mathtt{exp})).\mathsf{I'}$, the intruder can choose to execute the sequence $\mathtt{mov}^{k_4}_{b}.\mathtt{mov}^{k_5}_{c}.\mathsf{I'}$ as it allows him to exit b, enter c and execute $\mathsf{I'}$ in parallel with T and to achieve its purpose. The relation denoting the consequences of the intruder's choice is displayed in Table 4.7.

4.3.3 Security Correction

Our study of the example network revealed that the implemented security policy is not sufficient for the intended purpose of protecting the process T. There are several solutions for rendering it secure, corresponding to the potential enforcement places. They all involve access and exit keys availability. On one hand, there is the option to refrain from publishing the keys that were available in the original setup. On the other hand, more restriction can be implemented by adding new pairs of keys. The simplest method is to suppress the key publication server denoted by $!(\mathtt{pub}^{k_5}_{c,e}.0)$ within b. This may be feasible if the server is not absolutely needed for the usual system and network operations. However, given the fact that the server is active and defined by the initial security policy, there may be an actual requirement for it. The removal of the access key to b from the initial knowledge of processes in a can also be considered. This alternative is easy to implement, but also prevents the access to b for any process, which is not specifically denied in the original security policy. A similar effect would be obtained by enforcing a second overall level of protection for b. Another solution would be to protect the process T itself. The access key can be published in a sub-ambient of b to which no process from a can get access. Alternately, the server $!(\mathtt{pub}^{k_5}_{c,e}.0)$ could be protected by creating a new Sub-ambient d of b for which the key pair is not published within a. We chose this last solution as it only requires

one change in b's policy and does not imply modifications of the policies for ambients a and c. The network specification for the newly enforced security policy is:

$$\overset{k_1,k_2}{a}[\mathsf{A}] \mid \overset{k_3,k_4}{b}[!(\mathtt{pub}^{k_4}_{b,s}.0) \mid \overset{k_e^d,k_s^d}{d}[!(\mathtt{pub}^{k_5}_{c,e}.0)] \mid \mathsf{B'}] \mid \overset{k_5,k_6}{c}[\mathsf{T} \mid \mathsf{C'}]$$

4.4 Conclusion

In this chapter, we have presented a new approach for specifying computer systems that capture in an effective and elegant way security system behavior. Hence, we defined a new calculus that draws from the power and versatility of the ambient calculus and adds several new concepts such as protection keys and the ability to explore and find system vulnerabilities. The benefit of using our methodology for system modeling is twofold. It allows both the study of the security system behavior and the evaluation of an implementation of a given security policy. Moreover, it identifies changes required to ensure such correctness in order to guarantee policy compliance.

The scalability of our solution is also a strong point in its favor: it can be used to specify a single computer systems as well as very complex systems. However, our approach is different from previous works on the issue in many fashions. For instance, the introduction of the specific intruder semantics allows policy compliance verification and identification of flaws: both their nature and corrective actions. Nevertheless, the approach could benefit from the additional definition of a logic for specifying policies. Such a logic would contain all ingredients needed for both verifying and enforcing policies. Furthermore, it would help pointing out changes needed to correct a specification and the exact location of the changes needed for policy compliance.

Security Policy Verification

Chapter 5

Tableau Based Verification Algorithm for Security Policies

Abstract

This chapter introduces another original contribution: a framework for modeling computer systems, defining security policies, and verifying policy compliance. The CS2 process algebra captures the mobility aspects of systems and the SPL logic allows expressing access control in terms of formulas. The tableau-based proof system enables validation of the security policy implementation through formal model checking.

5.1 Introduction

Computer systems security requirements are implemented at the network, system, user, or application level. The most common ones are easily integrated based on guidelines, standards, and best practices. Still, they are rarely a perfect match of the actual computer system's protection. The rulesets are assumed to be correct because they are derived from a valid policy. Oftentimes this is not true and the network is left vulnerable to attacks from malicious intruders. Formal verification of the policy implementation's effectiveness would solve the problem. However, this requires a precise formal description of computer systems and policies, and a methodology for validating compliance. In this chapter we address the problems of computer system

specification and security policies verification.

Access control security policies imply regulated movement in and out of protected boundaries with the purpose of using various resources. Within computer systems, this can be represented in terms of process interactions. A formal specification of such interactions can be accomplished with a process algebra (calculus). While several calculi have been used for this purpose, we found that the mobile ambients of Cardelli and Gordon [21, 24] represent a good basis for our objective. Ambients are named locations containing processes and they capture the concepts of administrative scope and mobility in a simple and effective manner. Their hierarchical structures mirror the administrative domains of the system they model, which makes them suitable for representing computers and networks. Properties of processes described with the ambient calculus can be specified using the ambient logic [25].

Furthermore, the mandated behaviour of a system can be prescribed through sets of requirements bundled as security policies. A security policy can be a simple security constraint, such as the ability to execute a particular action, or a collection of conditions for the various components and the whole system. The logic makes use of temporal and spacial modalities to capture the static structure of a system as well as its dynamic evolution [54]. The idea of employing calculi and logics for policy compliance monitoring has been revisited in [7].

The various extensions of the original calculus, such as safe ambients [31, 70], security boundary analysis [16], or the ambient guardians [33] show limitations that make them less than ideal for the purpose of security policy verification. The original ambient calculus, for instance, is not refined enough for modeling firewalls and resorts to artificial concepts to simulate a simple accept/deny rule. The derived calculi involve elaborated constructs for verifying relatively simple properties and therefore have limited scalability. The original calculus and logic we propose here are better equipped for expressing access controlled mobility and the associated security policies.

As processes execute actions and systems evolve, they reach states that may or may not comply with a security policy. In order to determine which case stands, static or dynamic policy enforcement needs to be applied. Static enforcement is fairly common and entails verification of system properties

applied to a system description (or model). The properties are expressed with logic formulas. The model can be formally analyzed for proof of formula satisfaction with the method of semantic tableaux used by Fitting [37, 38] and Cleaveland [28]. Manual proofs can be produced through application of the tableau rules. Universal theorem provers such as Isabel [88], or tableau-specific ones such as Tableau Work Bench [1] and LoTREC [27, 41], can be further employed to completely automate the verification process. This is the area addressed by the algorithm proposed in this paper. We define the tableau inference rules for our logic and formally prove the tableau's finiteness, soundness and completeness. A LoTREC implementation of the tableau proof system for our logic is also produced.

Runtime enforcements are preferable for certain policies, as demonstrated by Schneider et al.[94, 49], Bauer [10], and Basin et al. [6]. Moreover, non-compliant systems can be modified to accommodate the required policies through program-rewriting, as shown by Hamlen [48], Langar et al. [67], Khoury et al. [61], and by Sui et al. [101]. Our verification technique can be applied independently or corroborated with dynamic enforcement, either at runtime or by program rewriting. The main benefit is that it can lower the number of policies to be enforced dynamically, improving the efficiency of the system.

The remainder of the chapter is organized as follows. The new calculus and logic are introduced in Sections 5.2 and 5.3 respectively. The tableau-based proof system for policy satisfaction is presented in Section 5.4. The case study depicted in Section 5.5 illustrates how the tableau-based proof system works. The implementation of the tableau proof system is presented in Section 5.6. Finally, Section 5.7 summarizes our conclusions and hints at directions for future work.

5.2 System Specification Calculus

In this section we define the syntax and the semantics of the *Calculus for Specification of Computer Systems* (abbreviated as *CS2*). The calculus allows the description of the relevant components of a computer system's structure in terms of hierarchical domains, communication capabilities and protection mechanisms.

Table 5.1: Syntax of *CS2*

$n \in \mathcal{N}$			**domain name**
$k \in \mathcal{K}$			**security key**
$\mathtt{a} \in \mathcal{A}$			**process action**
P, Q	::=		**processes**
		0	**deadlock**
	|	1	**successful termination**
	|	A	**action**
	|	$\mathsf{P}.\mathsf{Q}$	**sequence**
	|	$\mathsf{P} \mid \mathsf{Q}$	**parallel composition**
	|	$\mathsf{P} + \mathsf{Q}$	**choice**
	|	${}^{k}_{n}[\mathsf{P}]$	**ambient**
A	::=		**process capabilities**
		$\mathtt{mov}^{k}_{n}$	**movement**
	|	$\mathtt{a}$	**other actions**

5.2.1 Specification Syntax

The syntax of *CS2* is presented in Table 5.1. Let $\mathcal{N}$ be a set of names, $\mathcal{A}$ be a set of all possible process actions, and $\mathcal{K}$ be a set of keys used for protecting domains. In the proposed syntax, process constants 0 and 1 represent deadlock (or blocking) and successful process termination, respectively. A number of operators are defined as well: "." for sequence, "|" for parallel composition, and "+" for nondeterministic choice. Ambients are used for delimiting administrative domains. They are identified by names and protected by access keys. For instance, the expression ${}^{k}_{n}[\mathsf{P}]$ depicts an ambient named n, protected by the key k, and containing a process P.

Permission to access resources in actual computer systems can be easily modelled with key possession. For instance, a process is allowed to enter a domain protected by a key k if it has a movement capability with the key k. Let $(\mathcal{K}, \geq)$ be a partial ordered set, and let k, k' in $\mathcal{K}$. The expression $k \geq k'$ means that k is comparable to k', but more powerful, as it can open at least any ambient k' can open. For default ambient protection we use the public key δ, which is the $glb(\mathcal{K})$. The ability of processes to move is implemented by the $\mathtt{mov}^{k}_{n}$ action, where n is the name of a domain for which the access is requested and k is the access key. Note that the same action is used to exit from a domain.

General process interactions are expressed through a communication function, γ, which is a partial function of $\mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A}$ that satisfies the two following conditions:

1. $\forall a, b \in \mathcal{A} : \gamma(a, b) = \gamma(b, a)$ (commutativity)
2. $\forall a, b, c \in \mathcal{A} : \gamma(\gamma(a, b), c) = \gamma(a, \gamma(b, c))$ (associativity)

5.2.2 Specification Semantics

In the remainder of this chapter, we note $\mathcal{P}$ the set of processes that can be built using *CS2*. The operational semantics of the calculus is presented in terms of a structural congruence "≡" and a reduction relation "→". Table 5.2 displays a structural congruence on processes and includes, among others, reflexivity, symmetry, transitivity, associativity, and distributivity properties of the $+, |$ and . operators, which are common to most process algebras. The

Table 5.2: Structural Congruence for *CS2*

$$\mathsf{P} \equiv \mathsf{P} \tag{5.2.1}$$

$$\mathsf{P} \equiv \mathsf{Q} \Rightarrow \mathsf{Q} \equiv \mathsf{P} \tag{5.2.2}$$

$$\mathsf{P} \equiv \mathsf{Q}, \mathsf{Q} \equiv \mathsf{R} \Rightarrow \mathsf{P} \equiv \mathsf{R} \tag{5.2.3}$$

$$\mathsf{P} \equiv \mathsf{Q} \Rightarrow \mathsf{R}.\mathsf{P} \equiv \mathsf{R}.\mathsf{Q} \tag{5.2.4}$$

$$\mathsf{P} \equiv \mathsf{Q} \Rightarrow \mathsf{P}.\mathsf{R} \equiv \mathsf{Q}.\mathsf{R} \tag{5.2.5}$$

$$\mathsf{P} \equiv \mathsf{Q} \Rightarrow \mathsf{P} \mid \mathsf{R} \equiv \mathsf{Q} \mid \mathsf{R} \tag{5.2.6}$$

$$(\mathsf{P} \mid \mathsf{Q}) \mid \mathsf{R} \equiv \mathsf{P} \mid (\mathsf{Q} \mid \mathsf{R}) \tag{5.2.7}$$

$$\mathsf{P} \equiv \mathsf{Q} \Rightarrow \mathsf{P} + \mathsf{R} \equiv \mathsf{Q} + \mathsf{R} \tag{5.2.8}$$

$$\mathsf{P} \equiv \mathsf{Q} \Rightarrow {}^{k}_{n}[\mathsf{P}] \equiv {}^{k}_{n}[\mathsf{Q}] \tag{5.2.9}$$

$$0.\mathsf{P} \equiv 0 \tag{5.2.10}$$

$$\mathsf{P} + 0 \equiv \mathsf{P} \tag{5.2.11}$$

$$1.\mathsf{P} \equiv \mathsf{P} \equiv \mathsf{P}.1 \tag{5.2.12}$$

$$\mathsf{P} \mid 1 \equiv \mathsf{P} \tag{5.2.13}$$

$$\mathsf{P} + 1 \equiv \mathsf{P} \tag{5.2.14}$$

$$\mathsf{P}.(\mathsf{Q} + \mathsf{R}) \equiv \mathsf{P}.\mathsf{Q} + \mathsf{P}.\mathsf{R} \tag{5.2.15}$$

$$(\mathsf{P} + \mathsf{Q}).\mathsf{R} \equiv \mathsf{P}.\mathsf{R} + \mathsf{Q}.\mathsf{R} \tag{5.2.16}$$

$$\mathsf{P} \mid \mathsf{Q} \equiv \mathsf{Q} \mid \mathsf{P} \tag{5.2.17}$$

$$\mathsf{P} \mid (\mathsf{Q} + \mathsf{R}) \equiv (\mathsf{P} \mid \mathsf{Q}) + (\mathsf{P} \mid \mathsf{R}) \tag{5.2.18}$$

$$\mathsf{P} + \mathsf{P} \equiv \mathsf{P} \tag{5.2.19}$$

$$\mathsf{P} + \mathsf{Q} \equiv \mathsf{Q} + \mathsf{P} \tag{5.2.20}$$

effect of deadlock is expressed by (5.2.10) and (5.2.11). A process will stop any further execution once it encounters a deadlock, and the rest of its actions can be ignored. In case of a choice between a non-blocking process and a deadlock, the non-blocking process will execute. Successful termination, on the other hand, acts as a neutral element and does not change the sequential, parallel, or choice execution of a process. Its properties are shown in congruences (5.2.12) - (5.2.14).

Process evolutions are captured by the reduction relation defined in Table 5.3. Most rules are standard and need no explanation. Movement inside and outside ambients is captured by rules (5.3.8) and (5.3.9). For instance, rule (5.3.8) shows that a process can enter an ambient n protected by a key k provided it executes a movement directed into n with a key k' that is equal or superior to k.

System specifications are easy to build and read with our calculus. For instance, the expression ${}^{\delta}_{n}[\mathtt{mov}^{\delta}_{n}.\mathsf{A}]$ models an administrative domain n (a computer, a service, a network, etc.) that is publicly open (uses the public key δ), and contains a process which can exit the domain n and then continue to run as A.

5.3 Security Policy Logic

This section introduces a logic tailored for specifying security policies for computer systems described with *CS2*. The logic, named *Security Policy Logic* (or *SPL* in short), needs to be expressive enough to formulate any safety property.

5.3.1 Logic Syntax and Sematics

The syntax of *SPL* is summarized in Table 5.4. We define a modal logic with standard propositional connectives ($\neg, \vee$), a temporal operator ($.$), and a capability operator ($\langle\ \_\ \rangle$). Spatial connectives ($|$ and $\_[\ \_\ ]$) are also part of the syntax.

The following standard macros are used in the remainder of the chapter:

$$\Phi \wedge \Psi \quad \equiv \quad \neg(\neg\Phi \vee \neg\Psi) \qquad\qquad ff \quad \equiv \quad \neg tt$$

Table 5.3: Reduction Relation for *CS2*

$$\frac{\mathsf{P}' \equiv \mathsf{P},\ \mathsf{P} \xrightarrow{a} \mathsf{Q},\ \mathsf{Q} \equiv \mathsf{Q}'}{\mathsf{P}' \xrightarrow{a} \mathsf{Q}'} \tag{5.3.1}$$

$$\frac{\mathsf{P} \xrightarrow{a} \mathsf{P}'}{\mathsf{P}+\mathsf{Q} \xrightarrow{a} \mathsf{P}'} \tag{5.3.2}$$

$$\frac{\mathsf{P} \xrightarrow{a} \mathsf{P}'}{\mathsf{P}.\mathsf{Q} \xrightarrow{a} \mathsf{P}'.\mathsf{Q}} \tag{5.3.3}$$

$$\frac{\mathsf{P} \xrightarrow{a} \mathsf{P}'}{\mathsf{P} \mid \mathsf{Q} \xrightarrow{a} \mathsf{P}' \mid \mathsf{Q}} \tag{5.3.4}$$

$$\frac{\mathsf{P} \xrightarrow{a} \mathsf{P}',\ \mathsf{Q} \xrightarrow{b} \mathsf{Q}'}{\mathsf{P} \mid \mathsf{Q} \xrightarrow{\gamma(a,b)} \mathsf{P}' \mid \mathsf{Q}'} \tag{5.3.5}$$

$$\frac{\mathsf{P} \xrightarrow{a} \mathsf{P}'}{{}^{k}_{n}[\mathsf{P}] \xrightarrow{a} {}^{k}_{n}[\mathsf{P}']} \tag{5.3.6}$$

$$\frac{\mathtt{a} \neq \mathtt{mov}^{-}_{-}}{\mathtt{a} \xrightarrow{a} 1} \tag{5.3.7}$$

$$\frac{k' \geq k}{{}^{k}_{n}[\mathsf{P}] \mid \mathtt{mov}^{k'}_{n}.\mathsf{Q} \xrightarrow{\mathtt{mov}^{k'}_{n}} {}^{k}_{n}[\mathsf{P} \mid \mathsf{Q}]} \tag{5.3.8}$$

$$\frac{k' \geq k}{{}^{k}_{n}[\mathtt{mov}^{k'}_{n}.\mathsf{Q} \mid \mathsf{R}] \xrightarrow{\mathtt{mov}^{k'}_{n}} \mathsf{Q} \mid {}^{k}_{n}[\mathsf{R}]} \tag{5.3.9}$$

We define the semantics of *SPL* in Table 5.5, where $\mathtt{a} \in \mathcal{A}$, $n \in \mathcal{N}$, and $k \in \mathcal{K}$. The set of logical formulas specified in *SPL* is denoted by $\mathcal{F}$. The semantics of *SPL* is given by the meaning function $[\![\_]\!] : \mathcal{F} \rightarrow 2^{\mathcal{P}}$ defined inductively on the structure of formulas. We say that a process P satisfies the formula Φ and we note $\mathsf{P} \models \Phi$ if $\mathsf{P} \in [\![\Phi]\!]$.

All processes except for the blocking process satisfy the formula tt. Processes that are not part of the semantics of a certain formula Φ satisfy the negation of the formula. A process satisfies the formula $\Phi \vee \Psi$ if it satisfies either the formula Φ or the formula Ψ. The formula $\langle \mathsf{a} \rangle \Phi$ is satisfied by any process of the form $\mathsf{a}.\mathsf{P}$, provided that P satisfies Φ. Processes that satisfy the sequence logical formula involve the consecutive execution of two subprocesses, with each one satisfying the respective components of the formula. A process P' satisfies $\Phi \mid \Psi$ if there exists a process P satisfying Φ and a process Q satisfying Ψ such that $\mathsf{P}' = \mathsf{P} \mid \mathsf{Q}$. The protected location logical formula reflects the case when a specific behavior is required inside an ambient. A compliant process must match both the environment's external parameters (name and key) and the internal formula's semantics.

The syntax and semantics of our logic enable the construction of complex security policies. The logic can be used for expressing safety properties. For instance, to specifically deny access to resources inside a domain n protected by a key k', the following simple formula can be used:

$$\neg \langle \mathtt{mov}_n^k \rangle tt \;\mid\; {}_n^{k'}[tt]$$

Let's take another example by considering a system composed of an anti-virus server and a client:

$${}_c^{k_c}[tt \;\mid\; {}_v^{k_v}[tt]] \;\mid\; {}_s^{k_s}[\langle \mathtt{mov}_s^{k_s} \rangle \langle \mathtt{mov}_c^{k_c} \rangle \langle \mathtt{mov}_c^{k_c} \rangle tt \;\mid\; tt]$$

Table 5.4: Syntax of *SPL*

Φ, Ψ	$::=$		
		$\#\!\!t$	True
	$\mid$	$\neg\Phi$	Negation
	$\mid$	$\langle \mathsf{a} \rangle \Phi$	Capability
	$\mid$	$\Phi.\Psi$	Sequence
	$\mid$	$\Phi \mid \Psi$	Parallel Composition
	$\mid$	$\Phi \vee \Psi$	Disjunction
	$\mid$	${}^{k}_{n}[\Phi]$	Protected location

Table 5.5: Semantics of *SPL*

$$
\begin{array}{lcl}
[\![\#\!\!t]\!] & = & \mathcal{P} \setminus \{0\} \\
[\![\neg\Phi]\!] & = & \mathcal{P} \setminus [\![\Phi]\!] \\
[\![\langle \mathsf{a} \rangle \Phi]\!] & = & \{\mathsf{a}.\mathsf{P} \in \mathcal{P} \ : \ \mathsf{P} \in [\![\Phi]\!]\} \\
[\![\Phi.\Psi]\!] & = & \{\mathsf{P}.\mathsf{Q} \in \mathcal{P} \ : \ \mathsf{P} \in [\![\Phi]\!] \ \wedge \ \mathsf{Q} \in [\![\Psi]\!]\} \\
[\![\Phi \mid \Psi]\!] & = & \{\mathsf{P} \mid \mathsf{Q} \in \mathcal{P} \ : \ \mathsf{P} \in [\![\Phi]\!] \ \wedge \ \mathsf{Q} \in [\![\Psi]\!]\} \\
[\![\Phi \vee \Psi]\!] & = & [\![\Phi]\!] \cup [\![\Psi]\!] \\
[\![{}^{k}_{n}[\Phi]]\!] & = & \{{}^{k'}_{n}[\mathsf{P}] \in \mathcal{P} \ : \ \mathsf{P} \in [\![\Phi]\!], \ k' \geq k\}
\end{array}
$$

The security policy needs to account for the client and server administrative domains c and s and the anti-virus subsystem v within the client. This translates into three locations protected with the keys k_c, k_v, and k_s, of which two are nested. Another requirement is the the server's ability to push virus signature updates. That requirement is represented through a series of movement capabilities. Finally, the rest of the client and server's subsystems have to be represented. As we are not interested in the details of those components, they can be abstracted as True.

Now that the syntax and semantics of the algebraic calculus and the logic are presented, we are able to specify a computer system's structure and behavior and a desired security policy.

5.3.2 Formula Closure

In order to perform logical formula manipulations, we introduce some new definitions. They are useful for reasoning about the finite nature of formulas, which plays an important role in model checking. The following definitions are inspired by [28, 2] and are given in support of the proofs in the next section.

The evaluation of a system specification with respect to formula satisfaction will be made in Section 5.4 by breaking down complex formulas up to atomic formulas.

Definition 5.3.1 *Let Φ and Ψ be two formulas. Ψ is an immediate subterm of Φ, denoted by $\Psi \prec \Phi$, if exactly one of the following hold:*

$$\Phi = \neg\Psi$$

$$\Phi = \langle \mathsf{a} \rangle \Psi$$

$$\Phi = \Psi.\Phi' \text{ for some } \Phi'$$

$$\Phi = \Psi \mid \Phi' \text{ for some } \Phi'$$

$$\Phi = \Psi \vee \Phi' \text{ for some } \Phi'$$

$$\Phi = {}^{k}_{n}[\Psi]$$

The following definition allows us to determine the length of a formula.

Definition 5.3.2 *The size of a formula Φ, denoted by $|\Phi|$, is equal to the number of subterms and operations, i.e.:*

$$\begin{aligned}
|\not\!\bot| &= 1 \\
|\neg\Phi| &= 1 + |\Phi| \\
|\langle \mathsf{a} \rangle \Phi| &= 1 + |\Phi| \\
|\Phi.\Psi| &= 1 + |\Phi| + |\Psi| \\
|\Phi \mid \Psi| &= 1 + |\Phi| + |\Psi| \\
|\Phi \vee \Psi| &= 1 + |\Phi| + |\Psi| \\
|{}^{k}_{n}[\Phi]| &= 1 + |\Phi|
\end{aligned}$$

The closure of a formula is defined as the set of all its subterms and their subterms, up to atomic formulas.

Definition 5.3.3 *The closure $CL(\Phi)$ of a formula Φ is defined as follows:*

$$\begin{aligned}
\Phi &\in CL(\Phi) \\
CL(\neg\Phi) &= \{\neg\Phi\} \cup CL(\Phi) \\
CL(\langle \mathsf{a} \rangle \Phi) &= \{\langle \mathsf{a} \rangle \Phi\} \cup CL(\Phi) \\
CL(\Phi.\Psi) &= \{\Phi.\Psi\} \cup CL(\Phi) \cup CL(\Psi) \\
CL(\Phi \mid \Psi) &= \{\Phi \mid \Psi\} \cup CL(\Phi) \cup CL(\Psi) \\
CL(\Phi \vee \Psi)| &= \{\Phi \vee \Psi\} \cup CL(\Phi) \cup CL(\Psi) \\
CL({}^{k}_{n}[\Phi]) &= \{{}^{k}_{n}[\Phi]\} \cup CL(\Phi)
\end{aligned}$$

The previous definitions are useful for proving the finiteness of the tableau methodology. Closure can be interpreted as the set of all *SPL* formulas that can be obtained using all the subterms of a formula. Therefore, the cardinal of $CL(\Phi)$ is bounded by the size of the formula, as shown in the following proposition.

Proposition 5.3.4 *Let Φ be a formula. Then,*

$$|CL(\Phi)| \leq |\Phi|$$

Proof:
The proof is by structural induction on Φ. We demonstrate it below for Capability and Disjunction.

• $\langle \mathtt{a} \rangle \Phi$

Since $CL(\langle \mathtt{a} \rangle \Phi) = \{\langle \mathtt{a} \rangle \Phi\} \cup CL(\Phi)$, then

$$|CL(\langle \mathtt{a} \rangle \Phi)| \; = \; 1 \; + \; |CL(\Phi)|$$

$\Rightarrow$ {[Induction on the hypothesis $|CL(\Phi)| \leq |\Phi|$]}

$$|CL(\langle \mathtt{a} \rangle \Phi)| \; \leq \; 1 \; + \; |\Phi|$$

$\Rightarrow$ {[Definition of $|\langle \mathtt{a} \rangle \Phi|$]}

$$|CL(\langle \mathtt{a} \rangle \Phi)| \leq |\langle \mathtt{a} \rangle \Phi|$$

• $\Phi \vee \Psi$

Since $CL(\Phi \vee \Psi) = \{\Phi \vee \Psi\} \cup CL(\Phi) \cup CL(\Psi)$, then

$$|CL(\Phi \vee \Psi))| \; = \; 1 \; + \; |CL(\Phi)| \; + \; |CL(\Psi)|$$

$\Rightarrow$ {[Induction on the hypothesis $|CL(\Phi)| \leq |\Phi|$, $|CL(\Psi)| \leq |\Psi|$]}

$$|CL(\Phi \vee \Psi)| \; \leq \; 1 \; + \; |\Phi| \; + \; |\Psi|$$

$\Rightarrow$ {[Definition of $|\Phi \vee \Psi|$]}

$$|CL(\Phi \vee \Psi)| \leq |(\Phi \vee \Psi)|$$

The cases for Negation and Protected location follow the same steps as for the Capability proof. The cases for Sequence and Parallel Composition follow the steps for the Conjunction proof. □

5.4 Tableau-based Proof System for *SPL*

In this section we define a tableau-based proof system for *SPL*. The tableau system is a model-checking technique that relies on a set of inference rules used for determining automatically whether properties specified as logic formulas are satisfied. Tableaux (or semantic tableaux, as they are also called) have been used successfully in conjunction with modal logics [37, 28, 2] and are therefore suitable for *SPL*.

5.4.1 Building the Tableau

Using the logic semantics as defined in Section 5.3 is not very practical. Given a formula Φ, we have to calculate the (possibly infinite) set of processes that satisfy Φ and demonstrate that the model considered is included into that set of processes. However, the tableau-based technique considers local model checking. A deductive tableau system needs to provide formal assurance for all scenarios, therefore the reasoning about satisfaction must be made on a tautology. It is often easier to prove a contradiction, so it is a common practice to demonstrate the opposite, namely that the formula negation is unsatisfiable.

Model checking for a given model M is performed on sequents of the form $b \vdash_M s \in [\![\Phi]\!]$, where $s \in \mathcal{S}$ is a specification and $\Phi \in \mathcal{F}$ is a finite formula. The variable b is ranging over $\{\top, \bot\}$. We introduce the operations bb', $b \times b'$, and $b\Phi$ as follows:

bb'	$b \times b'$	$b\Phi$
$\top\top = \top$	$\top \times \top = \top$	$\top\Phi = \Phi$
$\top\bot = \bot$	$\top \times \bot = \bot$	$\bot\Phi = \neg\Phi$
$\bot\top = \bot$	$\bot \times \top = \bot$	$\top\neg\Phi = \neg\Phi$
$\bot\bot = \top$	$\bot \times \bot = \bot$	$\bot\neg\Phi = \Phi$

The inclusion of the variable b is crucial for reasoning on negative forms of the terms and reduces the number of inference rules needed for the tableau proof system introduced in Table 5.6. Note that our inference rules have the premise at the bottom and the conclusion on top, which permits a more

natural way of building the proof tree from the root up, rather then upside down.

The proof is actually based on a refutation tableau that starts with the negation of the formula: $\neg\Phi$. The backward-chaining proof is a tree labeled with formulas at every node and rooted in the original assumption. Branches are built through application of the inference rules and extend until no further inference can be made. The last formula on a branch is called a leaf.

A leaf $b \vdash s \in [\![\Phi]\!]$ is successful if :

- $b = \top \wedge \Phi = tt$
- $b = \bot \wedge \Phi = ff$

Leaves for which no rule can be applied are also successful leaves. A sequent is an unsuccessful leaf, if one of the following conditions holds:

- $b = \top \wedge \Phi = ff$
- $b = \bot \wedge \Phi = tt$
- $s = \mathtt{a}.s' \wedge \Phi = \langle \mathtt{a}' \rangle \Phi' \wedge \mathtt{a} \neq \mathtt{a}'$

A tableau is successful when all its leaves are successful. A tableau is unsuccessful if it contains at least one unsuccessful leaf. This means that the original assumption about $\neg\Phi$ is not satisfied, hence the formula Φ holds for the assessed model.

The rule $R_\neg$ makes verification of negative formulas straightforward. Instead of verifying whether $\neg\Phi$ is satisfied by a specification, we verify Φ and conclude the contrary for $\neg\Phi$. This is where variable b comes in handy. The other rules of Table 5.6 are intuitive enough to require no particular explanations. For instance the rule $R_{\langle \mathtt{a} \rangle}$ means that if $s \in [\![\Phi]\!]$ then $\mathtt{a}.s \in [\![\langle \mathtt{a} \rangle \Phi]\!]$.

Table 5.6: Tableau rules for *SPL*

$$R_{\neg} \qquad \frac{\bot b \vdash s \in [\![\Phi]\!]}{b \vdash s \in [\![\neg\Phi]\!]}$$

$$R_{\langle \mathsf{a} \rangle} \qquad \frac{b \vdash s \in [\![\Phi]\!]}{b \vdash \mathsf{a}.s \in [\![\langle \mathsf{a} \rangle \Phi]\!]}$$

$$R_{.} \qquad \frac{b \vdash s \in [\![\Phi]\!] \quad b \vdash s' \in [\![\Psi]\!]}{b \vdash s.s' \in [\![\Phi.\Psi]\!]}$$

$$R_{|} \qquad \frac{b' \vdash s \in [\![\Phi]\!] \quad b'' \vdash s' \in [\![\Psi]\!]}{b' \times b'' \vdash s \mid s' \in [\![\Phi \mid \Psi]\!]}$$

$$R_{1\vee} \qquad \frac{b \vdash s \in [\![\Phi]\!]}{b \vdash s \in [\![\Phi \vee \Psi]\!]}$$

$$R_{2\vee} \qquad \frac{b \vdash s \in [\![\Psi]\!]}{b \vdash s \in [\![\Phi \vee \Psi]\!]}$$

$$R_{[]} \qquad \frac{b \vdash s \in [\![\Phi]\!]}{b \vdash {}^{k}_{n}[s] \in [\![{}^{k}_{n}[\Phi]]\!]}$$

5.4.2 Tableau Finiteness, Soundness, and Completeness

In this section we demonstrate the tableau system's finiteness, soundness, and completeness. The results are formulated as theorems.

In order to prove the finiteness of the tableau, we have to demonstrate that there is no infinite ascending chain for any sequent that is part of the tableau. The ordering relation for formulas $\prec$ is extended to sequents as follows:

$$\Phi \prec \Phi' \;\Rightarrow\; b \vdash s \in [\![\ \Phi\ ]\!] \prec b' \vdash s' \in [\![\ \Phi'\ ]\!]$$

Let $R \in \{R_\neg, R_{\langle \mathtt{a} \rangle}, R_., R_|, R_{1\vee}, R_{2\vee}, R_{[]}\}$ be one of the inference rules from Table 5.6. The notation $\theta_1 \rightarrow_R \theta_2$ is used for showing that a parent sequent θ_1 reduces to a child sequent θ_2. The following proposition demonstrates that the number of possible children of a sequent is finite.

Proposition 5.4.1 *Let* $\theta_1 = b \vdash s_1 \in [\![\ \Phi_1\ ]\!]$ *and* $\theta_2 = b \vdash s_2 \in [\![\ \Phi_2\ ]\!]$. *Then,*

$$\theta_1 \rightarrow_R \theta_2 \Rightarrow CL(\Phi_1) \supseteq CL(\Phi_2)$$

Proof:
The proof is by induction on formulas and their immediate subterms.

- $R_\neg$: $\theta_1 \rightarrow_{R_\neg} \theta_2$

  $\Rightarrow$ {[Rule $R_\neg$]} $\Phi_1 = \neg\Phi_2$

  $\Rightarrow$ {[Definition of $CL(\neg\Phi_2)$]}

  $CL(\Phi_1) = CL(\neg\Phi_2) \supseteq CL(\Phi_2)$

- $R_{\langle \mathtt{a} \rangle \Phi}$: $\theta_1 \rightarrow_{R_{\langle \mathtt{a} \rangle \Phi}} \theta_2$

  $\Rightarrow$ {[Rule $R_{\langle \mathtt{a} \rangle \Phi}$]} $\Phi_1 = \langle \mathtt{a} \rangle \Phi$ and $\Phi_2 = \Phi$

  $\Rightarrow$ {[Definition of $CL(\langle \mathtt{a} \rangle \Phi)$]}

  $CL(\Phi_1) = CL(\langle \mathtt{a} \rangle \Phi) \supseteq CL(\Phi_2)$

- $R_{.}: \quad \theta_1 \rightarrow_{R_{.}} \theta_2$

  $\Rightarrow \{[\text{ Rule } R_{.} \text{ }]\} \quad \Phi_1 = \Phi.\Psi \text{ and } \Phi_2 \in \{\Phi, \Psi\}$

  $\Rightarrow \{[\text{ Definition of } CL(\Phi.\Psi) \text{ }]\}$

  $CL(\Phi_1) = CL(\Phi.\Psi) \supseteq CL(\Phi) \cup CL(\Psi) \supseteq CL(\Phi_2)$

- $R_{|}: \quad \theta_1 \rightarrow_{R_{|}} \theta_2$

  $\Rightarrow \{[\text{ Rule } R_{|} \text{ }]\} \quad \Phi_1 = \Phi \mid \Psi \text{ and } \Phi_2 \in \{\Phi, \Psi\}$

  $\Rightarrow \{[\text{ Definition of } CL(\Phi \mid \Psi) \text{ }]\}$

  $CL(\Phi_1) = CL(\Phi \mid \Psi) \supseteq CL(\Phi) \cup CL(\Psi) \supseteq CL(\Phi_2)$

- $R_{1\vee}: \quad \theta_1 \rightarrow_{R_{1\vee}} \theta_2$

  $\Rightarrow \{[\text{ Rule } R_{1\vee} \text{ }]\} \quad \Phi_1 = \Phi \vee \Psi \text{ and } \Phi_2 = \Phi$

  $\Rightarrow \{[\text{ Definition of } CL(\Phi \mid \Psi) \text{ }]\}$

  $CL(\Phi_1) = CL(\Phi \vee \Psi) \supseteq CL(\Phi) \cup CL(\Psi) \supseteq CL(\Phi_2)$

- $R_{2\vee}$ - same as for $R_{1\vee}$, but $\Phi_2 = \Psi$

- $R_{[]}: \quad \theta_1 \rightarrow_{R_{[]}} \theta_2$

  $\Rightarrow \{[\text{ Rule } R_{[]} \text{ }]\} \quad \Phi_1 =_n^k [\Phi] \text{ and } \Phi_2 = \Phi$

  $\Rightarrow \{[\text{ Definition of } CL(\binom{k}{n}[\Phi]) \text{ }]\}$

  $CL(\Phi_1) = CL(\binom{k}{n}[\Phi]) \supseteq CL(\Phi_2)$ $\square$

We conclude that for a tableau with sequent θ as root, formulas in all sequents derived directly by inference rules belong in $CL(\Phi)$. Informally, since Φ is finite, then $CL(\Phi)$ is finite, which means that there is a bounded number of first-level branches of the tree. Moreover, there will be a bounded number

of branches at every level. Therefore, the proposition demonstrates that a tableau built to prove formula Φ from the premise sequent θ has a maximum width. It remains to show that each branch has a finite height.

Proposition 5.4.2 *Let $\theta_1 = b \vdash s \in [\![\Phi_1]\!]$ be a sequent and let Θ be the ordered set $\{\theta_i : \theta_i \prec \theta_{i+1}, i \geq 1\}$ of terms of the θ_1's chain in a tableau proof, then,*

$$|\Theta| \leq |CL(\Phi_1)|$$

Proof:
The proof is done on sequent chains. Let R_i be inference rules $R_i \in \{R_\neg, R_{\langle \mathsf{a} \rangle}, R_., R_|, R_{1\vee}, R_{2\vee}, R_{[]}\}$. The elements of set $\Theta = \{\theta_1, \theta_2, \theta_3, \ldots\}$ come from the following chain of ordered sequents:

$\theta_1 \rightarrow_{R_1} \theta_2 \rightarrow_{R_2} \theta_3 \ldots$

$\Rightarrow$ {[Definition of ordered sequents]}

$\Phi_1 \prec \Phi_2 \prec \Phi_3 \prec \ldots$

$\Rightarrow$ {[Definitions of immediate subterms and size of formulas]}

$|\Phi_1| > |\Phi_2| > |\Phi_3| > \ldots$

$\Rightarrow$ {[Definition of equivalent sets]}

$\{\theta_1, \theta_2, \theta_3 \ldots\} \sim \{\Phi_1, \Phi_2, \Phi_3, \ldots\}$

$\Rightarrow$ {[Definition of set cardinal]}

$|\{\theta_1, \theta_2, \theta_3 \ldots\}| = |\{\Phi_1, \Phi_2, \Phi_3, \ldots\}|$

$\Rightarrow$ {[Definitions of immediate subterm, formula closure, Proposition 5.4.1]}

$\{\Phi_1, \Phi_2, \Phi_3, \ldots\} \subseteq CL(\Phi_1) \Rightarrow |\{\Phi_1, \Phi_2, \Phi_3, \ldots\}| \leq |CL(\Phi_1)|$ **(1)**

$\Rightarrow$ {[Proposition 5.3.4]} $|CL(\Phi_1)| < |\Phi_1| < \infty$ (**2**)

$\Rightarrow$ {[(**1**) and (**2**)]}

$|\Theta| \leq |CL(\Phi_1)| < \infty$ □

The two propositions in this section lead to the conclusion that a finite tableau can be built for a given sequent, therefore an assessment of the validity of the formula can be made. In practical terms, this means that we can verify formally whether a set of restrictions (i.e.: security policy) is satisfied by a given model (i.e.: system specification).

Theorem 5.4.3 (Finiteness) *For any sequent $\theta = b \vdash s \in [\![\Phi]\!]$, there is a tableau rooted in θ with finite maximum height and width.*

Proof:
The proof stems directly from Propositions 5.4.1 and 5.4.2. The tableau for a sequent with a finite formula Φ has a maximum width and a maximum height bounded by $|CL(\Phi)|$. □

Theorem 5.4.4 (Soundness) *If $\theta = b \vdash s \in [\![\Phi]\!]$ has a successful tableau, then $s \in [\![b\Phi]\!]$.*

Proof:
We need to prove that all successful leaves are semantically valid and all inference rules preserve soundness.

Leaves:

- $b = \top \wedge \Phi = tt$

  $b = \top \rightarrow b\Phi = tt$

  By hypothesis, $\top \vdash s \in [\![tt]\!]$, so $s \in [\![b\Phi]\!]$.

- $b = \bot \wedge \Phi = ff$

  $b = \bot \rightarrow \bot b = \top \wedge b\Phi = \neg ff = tt$

  By hypothesis and $R_\neg$, $\top \vdash s \in [\![\neg ff]\!]$, so $s \in [\![b\Phi]\!]$.

Inference rules:

We interpret soundness for the inference rules as follows. By hypothesis, the sequent of the denominator is sound and the sequent of the numerator has a successful tableau. It remains to prove that the numerator is also sound.

- $R_{\neg}$

  $b = \top \Rightarrow R_{\neg}$ has the form $\dfrac{\bot \vdash s \in [\![\ \Phi\ ]\!]}{\top \vdash s \in [\![\ \neg\Phi\ ]\!]}$

  By hypothesis,

  $$\begin{cases} \top \vdash s \in [\![\ \neg\Phi\ ]\!] \Rightarrow s \in [\![\ \neg\Phi\ ]\!] \\ \\ \bot \vdash s \in [\![\ \Phi\ ]\!] \Rightarrow s \notin [\![\ \Phi\ ]\!] \end{cases}$$

  Since $s \notin [\![\ \Phi\ ]\!] \equiv s \in [\![\ \neg\Phi\ ]\!] \equiv s \in [\![\ \top\neg\Phi\ ]\!]$, soundness is preserved.

  $b = \bot \Rightarrow R_{\neg}$ has the form $\dfrac{\top \vdash s \in [\![\ \neg\Phi\ ]\!]}{\bot \vdash s \in [\![\ \Phi\ ]\!]}$

  By hypothesis,

  $$\begin{cases} \bot \vdash s \in [\![\ \Phi\ ]\!] \Rightarrow s \in [\![\ \neg\Phi\ ]\!] \\ \\ \top \vdash s \in [\![\ \neg\Phi\ ]\!] \Rightarrow s \notin [\![\ \Phi\ ]\!] \end{cases}$$

  Since $s \notin [\![\ \Phi\ ]\!] \equiv s \in [\![\ \neg\Phi\ ]\!] \equiv s \in [\![\ \bot\Phi\ ]\!]$, soundness is preserved.

- $R_{\langle \mathsf{a} \rangle}$

  $b = \top \Rightarrow R_{\langle \mathsf{a} \rangle}$ has the form $\dfrac{\top \vdash s \in [\![\ \Phi\ ]\!]}{\top \vdash \mathsf{a}.s \in [\![\ \langle \mathsf{a} \rangle\Phi\ ]\!]}$

  By hypothesis,

  $$\begin{cases} \top \vdash \mathsf{a}.s \in [\![\ \langle \mathsf{a} \rangle\Phi\ ]\!] \Rightarrow \mathsf{a}.s \in [\![\ \langle \mathsf{a} \rangle\Phi\ ]\!] \\ \\ \top \vdash s \in [\![\ \Phi\ ]\!] \Rightarrow s \in [\![\ \Phi\ ]\!] \end{cases}$$

Since $s \in [\![\ \Phi\ ]\!] \equiv s \in [\![\ \top\Phi\ ]\!]$, soundness is preserved.

$b = \bot \Rightarrow R_{\langle \mathtt{a} \rangle}$ has the form $\dfrac{\bot \vdash s \in [\![\ \Phi\ ]\!]}{\bot \vdash \mathtt{a}.s \in [\![\ \langle \mathtt{a} \rangle \Phi\ ]\!]}$

By hypothesis,

$$\begin{cases} \bot \vdash \mathtt{a}.s \in [\![\ \langle \mathtt{a} \rangle \Phi\ ]\!] \Rightarrow \mathtt{a}.s \in [\![\ \neg\langle \mathtt{a} \rangle \Phi\ ]\!] \\ \bot \vdash s \in [\![\ \Phi\ ]\!] \end{cases}$$

The case for $\bot \vdash s \in [\![\ \Phi\ ]\!]$ has already been proven for $R_\neg$ when $b = \top$. Since $s \notin [\![\ \Phi\ ]\!] \equiv s \in [\![\ \bot\Phi\ ]\!]$, soundness is preserved.

- $R_.$

  $b = \top \Rightarrow R_.$ has the form $\dfrac{\top \vdash s \in [\![\ \Phi\ ]\!] \quad \top \vdash s' \in [\![\ \Psi\ ]\!]}{\top \vdash s.s' \in [\![\ \Phi.\Psi\ ]\!]}$

  By hypothesis,

$$\begin{cases} \top \vdash s.s' \in [\![\ \Phi.\Psi\ ]\!] \Rightarrow s.s' \in [\![\ \Phi.\Psi\ ]\!] \\ \top \vdash s \in [\![\ \Phi\ ]\!] \\ \top \vdash s' \in [\![\ \Psi\ ]\!] \end{cases}$$

  The case for $\top \vdash s \in [\![\ \Phi\ ]\!]$ has already been proven for $R_{\langle \mathtt{a} \rangle}$ when $b = \top$. It also applies for $\top \vdash s' \in [\![\ \Psi\ ]\!]$. Since $s \in [\![\ \Phi\ ]\!] \equiv s \in [\![\ \top\Phi\ ]\!]$ and $s' \in [\![\ \Psi\ ]\!] \equiv s' \in [\![\ \top\Psi\ ]\!]$, soundness is preserved.

- $R_|$, $R_{1\vee}$, $R_{2\vee}$, $R_{[]}$

  For the remaining tableau rules, proofs are similar to those for $R_\neg, R_{\langle \mathtt{a} \rangle}$ and $R_.$. The sequents to be proven sound can have one of the following four forms:

  1. $\top \vdash s \in [\![\ \Phi\ ]\!]$
  2. $\top \vdash s \in [\![\ \neg\Phi\ ]\!]$
  3. $\bot \vdash s \in [\![\ \Phi\ ]\!]$
  4. $\bot \vdash s \in [\![\ \neg\Phi\ ]\!]$

The first three forms have been addressed already in the proofs for $R_{\neg}, R_{\langle \mathtt{a} \rangle}$ and $R_{\cdot}$. For the fourth:

$\bot \vdash s \in [\![\neg\Phi]\!] \Rightarrow s \notin [\![\neg\Phi]\!]$, but $s \notin [\![\neg\Phi]\!] \equiv s \in [\![\Phi]\!]$, so the sequent is equivalent to $\bot \vdash s \in [\![\Phi]\!]$.

□

Proposition 5.4.5 *Let s be a system specification and Φ be a formula. Then, $b \vdash s \in [\![\Phi]\!]$ has a successful tableau $\Leftrightarrow$ $b \vdash s \in [\![\neg\Phi]\!]$ has no successful tableau.*

Proof:

- $\Rightarrow$
  By hypothesis, $b \vdash s \in [\![\Phi]\!]$ has a successful tableau and, by Theorem 5.4.4, $s \in [\![b\Phi]\!]$.

  Suppose that $b \vdash s \in [\![\neg\Phi]\!]$ also has a successful tableau. In this case, Theorem 5.4.4 states that $s \in [\![b\neg\Phi]\!]$.

  Since $b\neg = \neg b$, then $[\![b\neg\Phi]\!] = [\![\neg b\Phi]\!]$ and, by definition of $[\![\neg b\Phi]\!]$, we deduce that $s \notin [\![b\Phi]\!]$.

  This contradicts the hypothesis that $s \in [\![b\Phi]\!]$, and therefore the supposition that $b \vdash s \in [\![\neg\Phi]\!]$ also has a successful tableau is false.

- $\Leftarrow$
  We need to prove that if $b \vdash s \in [\![\neg\Phi]\!]$ has no successful tableau, then $b \vdash s \in [\![\Phi]\!]$ must have one.

  By hypothesis, $b \vdash s \in [\![\neg\Phi]\!]$ has no successful tableau. This means, by Theorem 5.4.4, that $s \notin [\![b\neg\Phi]\!]$. Since $[\![b\neg\Phi]\!] = [\![\neg b\Phi]\!]$, we deduce that which is equivalent $s \notin [\![\neg b\Phi]\!]$.

  From the definition of semantics, $s \in [\![\neg\neg b\Phi]\!] = s \in [\![b\Phi]\!]$. Therefore, $b \vdash s \in [\![\Phi]\!]$ has a successful tableau.

□

Theorem 5.4.6 (Completeness) *Let s be a system specification and Φ be a formula. Then,*

$s \in [\![\ b\Phi\ ]\!] \Rightarrow b\ \vdash s \in [\![\ \Phi\ ]\!]$ *has a successful tableau.*

Proof:
Suppose that $s \in [\![\ b\Phi\ ]\!]$, but $b\ \vdash s \in [\![\ \Phi\ ]\!]$ does not have a successful tableau.

By Proposition 5.4.5, this means that $b\ \vdash s \in [\![\ \neg\Phi\ ]\!]$ has a successful tableau.

By Theorem 5.4.4, we deduce that $s \in [\![\ \neg b\Phi\ ]\!]$ which is equivalent to $s \notin [\![\ b\Phi\ ]\!]$, but this contradicts the hypothesis that $s \in [\![\ b\Phi\ ]\!]$. □

5.5 Case Study

In this section, we illustrate our technique with an example of a medical computer system that allows nurses and doctors to read, create and change prescriptions for their patients. The example demonstrates the use of the *CS2* algebra, the *SPL* logic, and the tableau-based proof method. We show that the system satisfies the security policy.

5.5.1 System Specification

The emphasis of the case study is the application of the tableau rules and therefore the computer system is kept simple. It includes a desktop computer for the nurse, a tablet for the doctor and a file server for the patient's current prescription, medical test results, allergies, and their medical history.

The hospital system's specification, expressed in *CS2* terms, is as follows:

$$\mathsf{H} \;=\; {}^{k_n}_{n}[\mathtt{mov}^{k_n}_{n}.\mathtt{mov}^{k_p}_{p}.\mathsf{N}] \;\mid\; {}^{k_d}_{d}[\mathtt{mov}^{k_d}_{d}.\mathtt{mov}^{k_p}_{p}.(\mathsf{D_1} \;\mid\; \mathtt{mov}^{k_f}_{f}.\mathsf{D_2})] \;\mid\; {}^{k_p}_{p}[\mathsf{P} \;\mid\; {}^{k_f}_{f}[\mathsf{F}]]$$

$$\textit{where}\ \mathsf{N}, \mathsf{D_1}, \mathsf{D_2}, \mathsf{P}, \mathsf{F} \neq 0,\ k_n, k_d, k_p, k_f \neq \delta$$

The three devices, depicted hrough ambients n, d, and p respectively, are properly safeguarded according to the best industry standards, as symbolized

by their access keys k_n, k_d, and k_p. The file server has a general folder for current prescriptions and a restricted one for the patient's more sensitive information. The restricted folder is protected by the key k_f. The nurse can only view current prescriptions (process N), while the doctor can do the same (process D_1) and additionally access the patient's medical history and write new prescriptions or enter new test results (process D_2).Finally, processes P and F represent the prescription and the sensitive information.

The security policy for the system reflects the system's purpose: we require that the nurse and the doctor have their appropriate access levels and that the file server has a restricted area for sensitive data. The *SPL* formula for the system's security policy is:

$$\Phi \;=\; {}^{k_n}_{n}[\langle \mathtt{mov}^{k_n}_{n}\rangle\langle \mathtt{mov}^{k_p}_{p}\rangle \not\Box] \;\mid\; {}^{k_d}_{d}[\langle \mathtt{mov}^{k_d}_{d}\rangle\langle \mathtt{mov}^{k_p}_{p}\rangle(\not\Box \;\mid\; \langle \mathtt{mov}^{k_f}_{f}\rangle \not\Box)] \;\mid\; {}^{k_p}_{p}[\not\Box \;\mid\; {}^{k_f}_{f}[\not\Box]]$$

The policy allows us demonstrate the application of 4 different tableau rules and verification of the conditions for both successful and unsuccessful leaves. The case study has a reasonable degree of complexity for a manual proof. At the same time, it is kept legible to prevent from splitting the proof tree on several pages. The same scenario is treated in Sect. 5.6, where the advantages of automation over manual proofs are highlighted.

5.5.2 Proof Tree

The hypothesis in this case study have already been presented with the system specification. The initial value of the variable b is $\top$. The sequent to use for producing an unsuccessful tableau is $\top \vdash \mathsf{H} \in [\![\, \neg\Phi \,]\!]$. The inference rules in Table 5.6 are applied to the premise and produce the proof tree shown in Table 5.7.

The premise of the tableau involves the negation of a formula, so the rule $R_\neg$ is invoked first. This implies changes both to the formula and to the variable b. The value of variable b of the sequent is modified from $\top$ to $\bot\top = \bot$. The resulting formula Φ involves parallel composition and the rule $R_|$ is applied. Note the change of the sequent variables corresponding to b' and b'' in the $R_|$ inference rule. They take the values $\bot$ and $\top$, respectively, to comply with the operation $\bot \times \top = \bot$.

Table 5.7: Tableau system proof for case study

successful

$$\top \vdash \mathsf{D}_2 \in \llbracket\, \mathit{tt}\, \rrbracket$$

successful

$$R_{\langle \mathtt{a} \rangle}$$

$$\top \vdash \mathsf{D}_1 \in \llbracket\, \mathit{tt}\, \rrbracket \qquad \bot \vdash \mathtt{mov}_f^{k_f}.\mathsf{D}_2 \in \llbracket\, \langle \mathtt{mov}_f^{k_f} \rangle \mathit{tt}\, \rrbracket$$

successful

$$R_{|}$$

$$\top \vdash \mathsf{F} \in \llbracket\, \mathit{tt}\, \rrbracket$$

$$\bot \vdash \mathsf{D}_1 \mid \mathtt{mov}_f^{k_f}.\mathsf{D}_2 \in \llbracket\, \mathit{tt} \mid \langle \mathtt{mov}_f^{k_f} \rangle \mathit{tt}\, \rrbracket$$

unsuccessful

successful

$$R_{\langle \mathtt{a} \rangle} \qquad R_{[]}$$

$$\bot \vdash \mathsf{N} \in \llbracket\, \mathit{tt}\, \rrbracket \qquad \top \vdash \mathtt{mov}_p^{k_p}.\mathsf{D} \in \llbracket\, \langle \mathtt{mov}_p^{k_p} \rangle (\mathit{tt} \mid \langle \mathtt{mov}_f^{k_f} \rangle \mathit{tt})\, \rrbracket \qquad \top \vdash \mathsf{P} \in \llbracket\, \mathit{tt}\, \rrbracket \qquad \top \vdash {}^{k_f}_{f}[\mathsf{F}] \in \llbracket\, {}^{k_f}_{f}[\mathit{tt}]\, \rrbracket$$

$$R_{\langle \mathtt{a} \rangle} \qquad R_{\langle \mathtt{a} \rangle} \qquad R_{|}$$

$$\bot \vdash \mathtt{mov}_p^{k_p}.\mathsf{N} \in \llbracket\, \langle \mathtt{mov}_p^{k_p} \rangle \mathit{tt}\, \rrbracket \qquad \top \vdash \mathtt{mov}_d^{k_d}.\mathtt{mov}_p^{k_p}.\mathsf{D} \in \llbracket\, \langle \mathtt{mov}_d^{k_d} \rangle \langle \mathtt{mov}_p^{k_p} \rangle (\mathit{tt} \mid \langle \mathtt{mov}_f^{k_f} \rangle \mathit{tt})\, \rrbracket \qquad \top \vdash \mathsf{P} \mid {}^{k_f}_{f}[\mathsf{F}] \in \llbracket\, \mathit{tt} \mid {}^{k_f}_{f}[\mathit{tt}]\, \rrbracket$$

$$R_{\langle \mathtt{a} \rangle} \qquad R_{[]} \qquad R_{[]}$$

$$\bot \vdash \mathtt{mov}_n^{k_n}.\mathtt{mov}_p^{k_p}.\mathsf{N} \in \llbracket\, \langle \mathtt{mov}_n^{k_n} \rangle \langle \mathtt{mov}_p^{k_p} \rangle \mathit{tt}\, \rrbracket \qquad \top \vdash {}^{k_d}_{d}[\mathtt{mov}_d^{k_d}.\mathtt{mov}_p^{k_p}.\mathsf{D}] \in \llbracket\, {}^{k_d}_{d}[\langle \mathtt{mov}_d^{k_d} \rangle \langle \mathtt{mov}_p^{k_p} \rangle (\mathit{tt} \mid \langle \mathtt{mov}_f^{k_f} \rangle \mathit{tt})]\, \rrbracket \qquad \top \vdash {}^{k_p}_{p}[\mathsf{P} \mid {}^{k_f}_{f}[\mathsf{F}]] \in \llbracket\, \mathit{tt} \mid {}^{k_f}_{f}[\mathit{tt}]\, \rrbracket$$

$$R_{[]} \qquad R_{|}$$

$$\bot \vdash {}^{k_n}_{n}[\mathtt{mov}_n^{k_n}.\mathtt{mov}_p^{k_p}.\mathsf{N}] \in \llbracket\, {}^{k_n}_{n}[\langle \mathtt{mov}_n^{k_n} \rangle \langle \mathtt{mov}_p^{k_p} \rangle \mathit{tt}]\, \rrbracket \qquad \top \vdash {}^{k_d}_{d}[\mathtt{mov}_d^{k_d}.\mathtt{mov}_p^{k_p}.\mathsf{D}] \mid {}^{k_p}_{p}[\mathsf{P} \mid {}^{k_f}_{f}[\mathsf{F}]] \in \llbracket\, {}^{k_d}_{d}[\langle \mathtt{mov}_d^{k_d} \rangle \langle \mathtt{mov}_p^{k_p} \rangle (\mathit{tt} \mid \langle \mathtt{mov}_f^{k_f} \rangle \mathit{tt})] \mid {}^{k_p}_{p}[\mathit{tt} \mid {}^{k_f}_{f}[\mathit{tt}]]\, \rrbracket$$

$$R_{|}$$

$$\bot \vdash {}^{k_n}_{n}[\mathtt{mov}_n^{k_n}.\mathtt{mov}_p^{k_p}.\mathsf{N}] \mid {}^{k_d}_{d}[\mathtt{mov}_d^{k_d}.\mathtt{mov}_p^{k_p}.\mathsf{D}] \mid {}^{k_p}_{p}[\mathsf{P} \mid {}^{k_f}_{f}[\mathsf{F}]] \in \llbracket\, {}^{k_n}_{n}[\langle \mathtt{mov}_n^{k_n} \rangle \langle \mathtt{mov}_p^{k_p} \rangle \mathit{tt}] \mid {}^{k_d}_{d}[\langle \mathtt{mov}_d^{k_d} \rangle \langle \mathtt{mov}_p^{k_p} \rangle (\mathit{tt} \mid \langle \mathtt{mov}_f^{k_f} \rangle \mathit{tt})] \mid {}^{k_p}_{p}[\mathit{tt} \mid {}^{k_f}_{f}[\mathit{tt}]]\, \rrbracket$$

$$R_{\neg}$$

$$\top \vdash {}^{k_n}_{n}[\mathtt{mov}_n^{k_n}.\mathtt{mov}_p^{k_p}.\mathsf{N}] \mid {}^{k_d}_{d}[\mathtt{mov}_d^{k_d}.\mathtt{mov}_p^{k_p}.\mathsf{D}] \mid {}^{k_p}_{p}[\mathsf{P} \mid {}^{k_f}_{f}[\mathsf{F}]] \in \llbracket\, \neg({}^{k_n}_{n}[\langle \mathtt{mov}_n^{k_n} \rangle \langle \mathtt{mov}_p^{k_p} \rangle \mathit{tt}] \mid {}^{k_d}_{d}[\langle \mathtt{mov}_d^{k_d} \rangle \langle \mathtt{mov}_p^{k_p} \rangle (\mathit{tt} \mid \langle \mathtt{mov}_f^{k_f} \rangle \mathit{tt})] \mid {}^{k_p}_{p}[\mathit{tt} \mid {}^{k_f}_{f}[\mathit{tt}]])\, \rrbracket$$

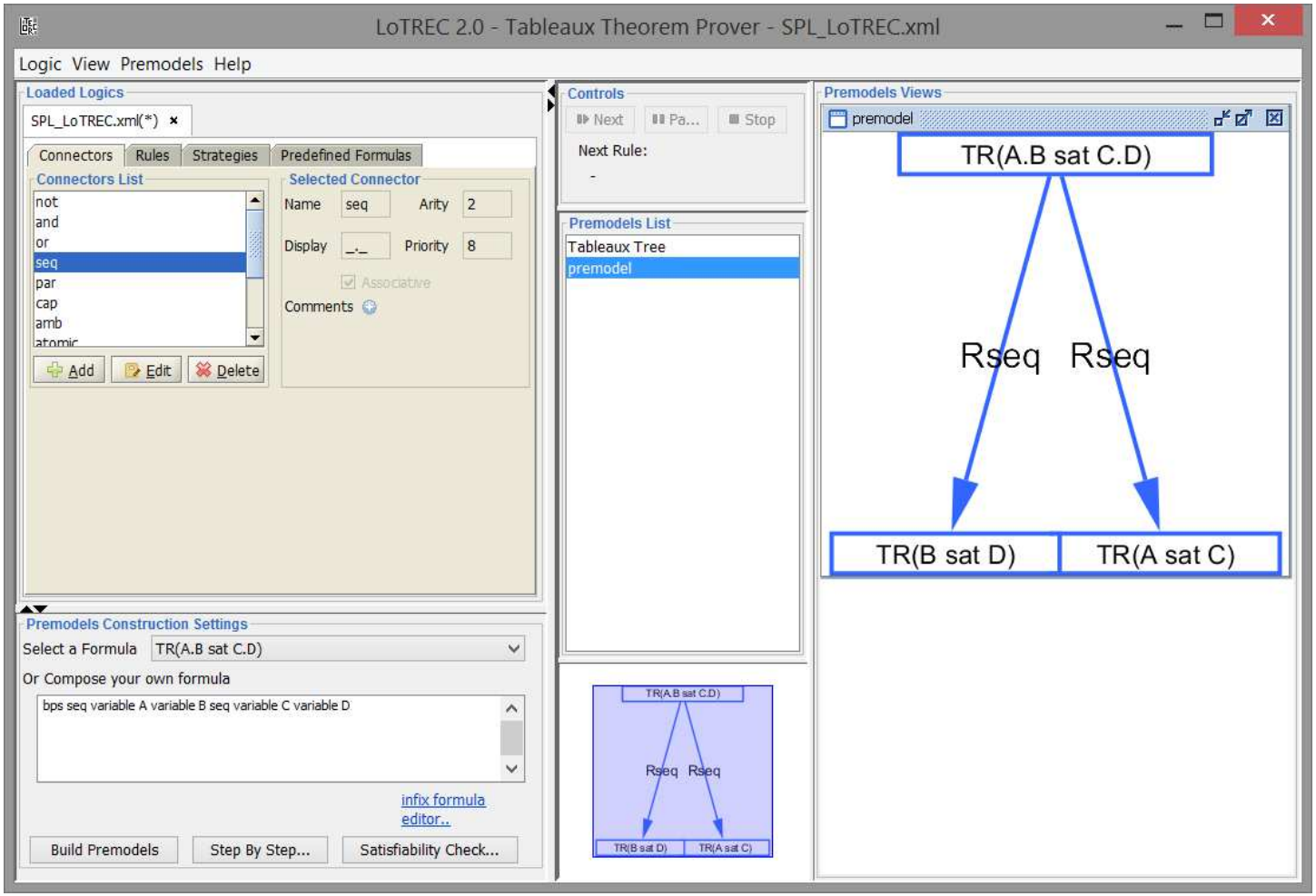
The successive application of the $R_{\langle \mathtt{a} \rangle}, R_{|}$, and $R_{[]}$ tableau rules results in five branches. Finally, at the top of the branches, the proof leads to leafs where processes are evaluated for satisfaction of formula $\not\equiv$. Four of the leaves are successful, since any process belongs to $[\![\not\equiv]\!]$. The only exception is $\bot \vdash \mathsf{N} \in [\![\not\equiv]\!]$, which is unsuccessful. The tableau is then unsuccessful, the original sequent does not hold, and the proof by refutation tableau is complete. Therefore, the system depicted by the case study satisfies the security policy.

5.6 LoTREC Implementation of the Tableau Proof System for *SPL*

The generic tableau theorem prover LoTREC [41] has been introduced in chapter 2. In this section we show our LoTREC implementation of the tableau proof system for *SPL* and apply it to the case study from the previous section. The application allows model checking on predefined or user-defined logics. The resulting system can be applied to formulas to verify whether they have successful tableaux or not. The formal semantics of LoTREC ensures that correctly specified rules always produce "correct" results. The tableau successfulness is evaluated as in the previous sections, and the truth assessment can be marked in the associated graph by the TRUE or FALSE labels.

A tableau system implementation requires the definition of the logic and the tableau inference rules. The syntax uses operators for defining formulas, sequents, and rules. In our case, we needed to express both logical formulas and system specifications. Therefore, elements of both *CS2* and *SPL* were accommodated. The syntax and semantics of the logic, along with tableau rules and a strategy, are saved as an XML file. The resulting system can be applied to formulas to verify whether they have successful tableaux or not. The underlying components that make LoTREC a formal prover are transparent to the user. A graphic interface, depicted in Fig. 5.1, enables definition of new logics, tableau systems, or custom formulas to be verified. The truth result is shown in the shape of an upside-down proof tree rooted in the original formula.

Figure 5.1: LoTREC implementation of the tableau proof system for *SPL*



5.6.1 *SPL* Connectors

The LoTREC implementation starts with the definition of the logic. The syntax implies operators for defining formulas, sequents, and rules. In our case, we need to express both logical formulas and system specifications. Therefore, elements of both *CS2* and *SPL* have to be accommodated. A combination of variable, constants, and custom operators is used for the implementation.

The system specification implies the definition of processes, process capabilities and operations, and the ambient structure. Termination and deadlock are simply represented through the constants 0 and 1. The implementation of connectors includes definitions for sequence, parallel composition, choice, movement, and ambient.

Each connector requires a name, a display symbol, an arity and a priority. The arity is determined by the number of terms linked by a connector. The terms can be a constant, a variable, or another connector. A priority is set for each connector. The priorities chosen for the *SPL* tableau implementation reflect the order of precedence in classic process algebras and logics. LoTREC allows specifying whether a connector is associative. The connectors corresponding to parallelism, choice, and conjunction, for instance, are associative.

System specification connectors reflect the elements of the *CS2* syntax. A sequence, for instance, is represented by the connector with the name seq, has an arity of 2 (since it connects two terms), a priority of 2, and is displayed as "_._", with each "_" being replaced by the actual term. The ambient connector is called amb, has an arity of 3 and a priority of 5, and is displayed as "_, _[_]". The other *CS2*-related connectors are as follows: par, choice, repl, mov and amb.

The connectors for the *SPL* logic follow the same structure. Capability is represented by cap, a connector with arity and priority 2, displayed as "< _ > _". The connector for negation is named not, has arity 1, priority 5, and is displayed as " ∼ _". Disjunction is modelled as or, has arity 2, priority 3, and is displayed as "_v_". Conjunction is often used in proofs and it earns a dedicated connector that is much easier to handle than as a

double negation involving a disjunctive formula. The associated connector, called and, has arity 2, priority 4, and is displayed as "_&_". Due to the structural similarity with the algebra connectors, Sequence, Parallel Composition, and Protected Location can be represented through the seq, par and amb connectors previously mentioned. True is represented through constant sf TT. Note that in LoTREC lower case names are reserved for variables, therefore the more adequate symbol tt could not be used.

Two additional connectors are added for defining sequents: sbp and sbn, standing for "sequent - b positive" and "sequent - b negative", respectively. A single LoTREC rule accommodating both values of the variable b would have become very complex for some tableau rules and would have been impossible to express for others. Therefore we chose to include two LoTREC rules for each inference rule.

The sbp and sbn connectors solve the issue by abstracting the variable b completely. A sequent is simply presented as a formula satisfaction evaluation prefixed by a true or false unary operator. Both connectors have arity 2 and priority 0 and are not associative. They are displayed as "TR(_sat_)" and "FL(_sat_)", respectively. For instance, the sequent $\top \vdash \mathsf{P.Q} \in [\![\Phi.\Psi]\!]$ would be displayed as TR(P.Q sat $\Phi.\Psi$).

5.6.2 *SPL* Rules and Strategies

LoTREC rules consist of conditions and actions. As mentioned in the previous section, each inference rule is represented by a pair of rules in the LoTREC implementation. We used intuitive names for the rules: R_Not and R_NotN correspond to $R_{\neg}$, R_Par and R_ParN correspond to $R_{|}$, etc.

The *SPL* tableau system rules use the premise sequent at the denominator as their conditions. There are 20 condition names, which are actually predefined conditions on elements, links, nodes, parents, successors, etc. Rules are applied to formulas that label the nodes. The condition name, node, and formula have to be defined when creating a new rule. The default node name node is used in most circumstances, with node' being defined when new nodes have to be created. Rule actions have a structure similar to conditions, requiring a name, node, and formula. Among the 15 predefined action names, the most useful for our implementation are add, link and createNewNode.

They allow adding a new label to the node, creating a successor (or child) node, or linking two nodes.

The rule corresponding to $\mathsf{R}_{[]}$ for $b = \top$, for example, has the following components:

- Rule name: R_Amb
- Condition name: hasElement
- Condition node: node
- Condition formula: bps amb variable N variable K variable A amb variable N variable K variable B
- Action name: hasElement
- Action node: node
- Action formula: bps variable A variable B

Two more implementation rules are depicted in Fig. 5.2 and Fig. 5.3. The complete ruleset is compiled into an XML file. The formal semantics of LoTREC ensures that correctly specified rules always produce "correct" results. Rules are consolidated in strategies, which define what rules are applied, in which order, and when does the strategy stop. The default strategy we built for the *SPL* tableau proof system, pictured in Fig. Fig. 5.4 includes all the rules we defined.

The tableau successfulness is evaluated as in the previous sections, and the truth assessment can be marked in the associated graph by the TRUE or FALSE labels. Verification can be performed on formulas composed directly in the dedicated interface window or on predefined formulas, as shown in Fig. 5.5. The resulting proof tree displays both the applicable inference rules and the labels.

The proof tree for the case study in the previous section is shown in Fig. 5.6. Each tableau rule application corresponds to a label. Branching caused by the parallelism in the security policy is clearly marked in the graph. A

Figure 5.2: LoTREC *SPL* tableau rule for $R_{\neg}$

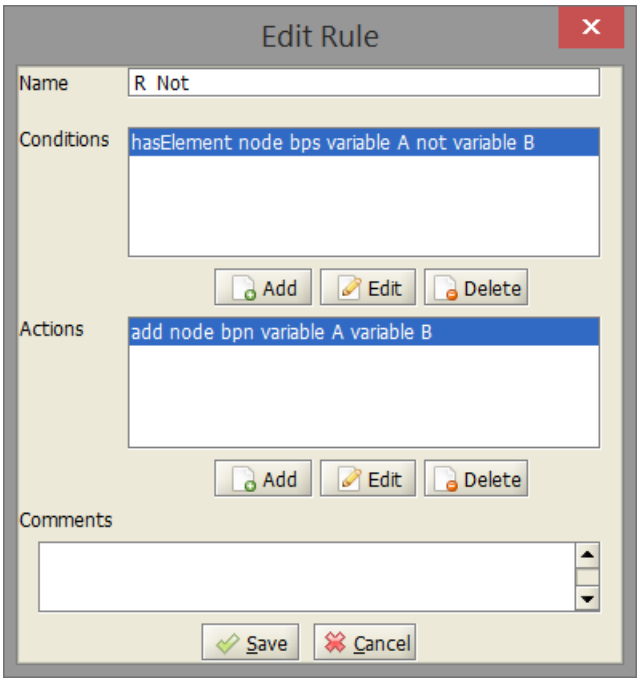


Figure 5.3: LoTREC *SPL* tableau rule for $R_{|}$

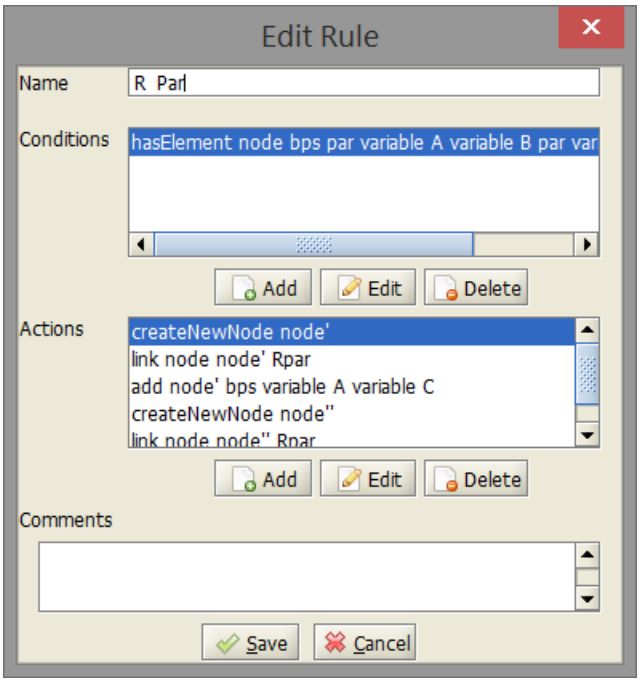


Figure 5.4: LoTREC *SPL* tableau strategy

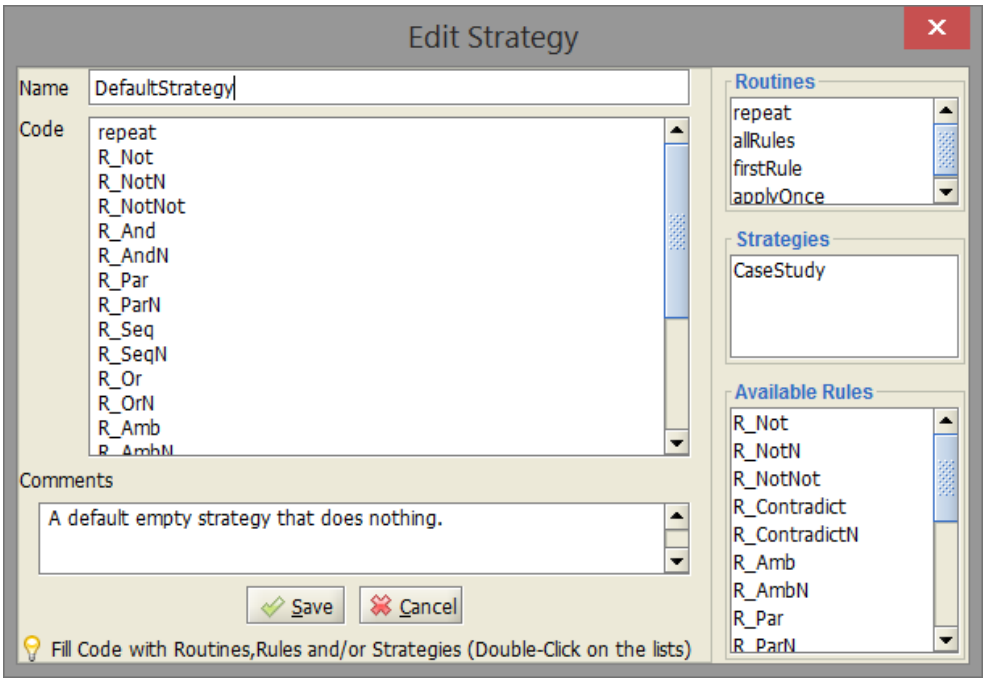
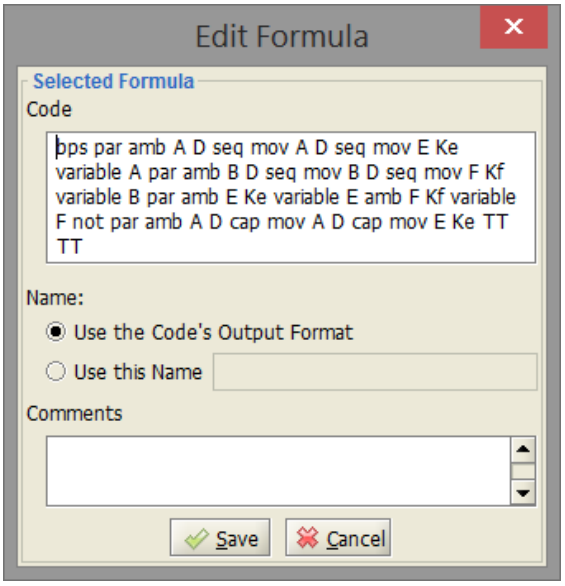


Figure 5.5: LoTREC *SPL* tableau predefined formula



step-by-step run of the proof will reveal that the same rules as in the manual proof in Table 5.7 are applied, exactly in the same order. The tree contains one successful and one unsuccessful leaf, as expected. Therefore, the tree is unsuccessful and the refutation tableau proof is complete. Note that proof generation was significantly faster than building the proof by hand, underlining once more the benefit of automation.

5.7 Conclusion

Formal verification of security policies can ensure that access is provided as intended, with no accidental backdoors or latent vulnerabilities. We are proposing in this chapter an automated algorithm for assessing policy implementation. The algorithm employs inference rules applied to logic formulas and systems specifications that we define with the *CS2* process algebra and the *SPL* modal logic, respectively. The statement to be evaluated is presented in the form of a sequent, which takes into account a model of the analyzed system and a formula representing the policy. A tableau is then built on back-chaining premises and conclusions to determine whether the original statement holds. Formal proofs are given for the finiteness, soundness, and completeness of our tableau methodology. If the tableau is closed, then a conclusion can be reached about the initial sequent, meaning that the verified formula is satisfied (or not) by the model.

The implementation of our tableau-proof system in LoTREC, a tableau theorem prover, is also presented. It allows automatic verification of policy compliance, eliminating the need for manual application of the inference rules. The four components - the algebra, the logic, the proof algorithm, and the implementation - integrate optimally as they are built specifically for the purpose. A set of access permissions and restrictions built as security policies can be checked individually or together. Moreover, the methodology allows verification of any reasoning about the model that can be expressed with *SPL* formulas.

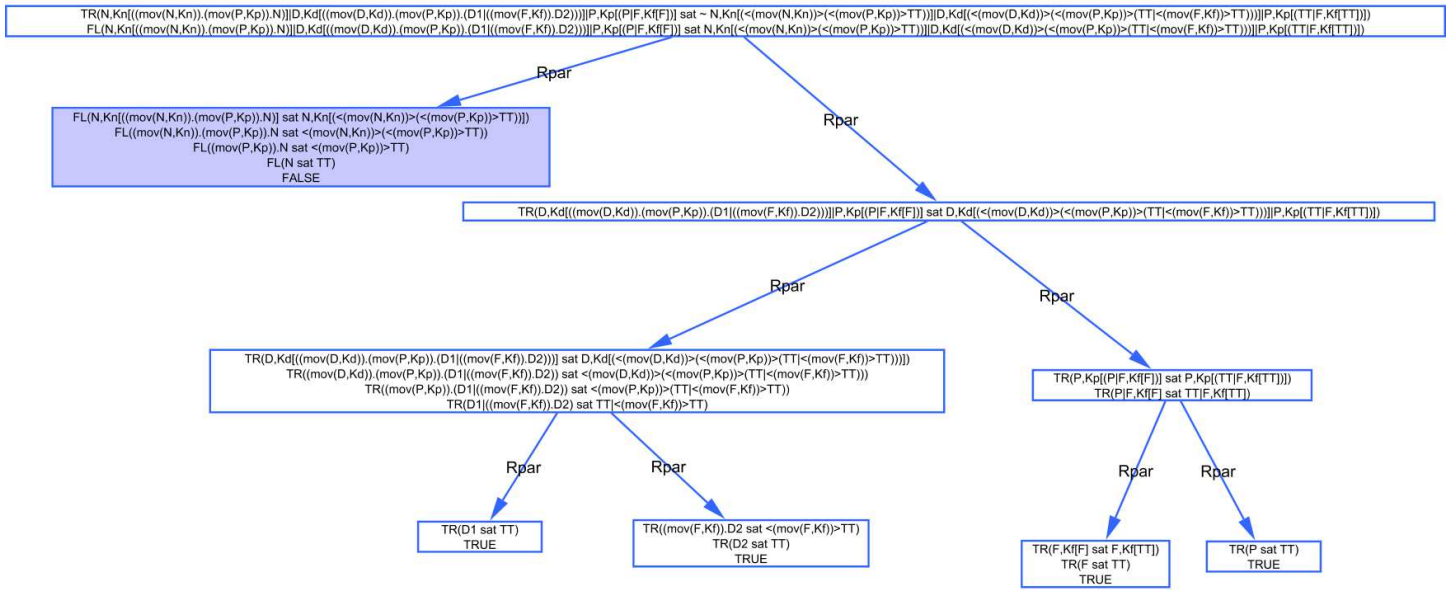


Figure 5.6: LoTREC *SPL* tableau proof for the case study

One of the merits of our approach is that it unifies all components in a formal, homogeneous framework. The solutions we have come across so far and referenced herein fall short of our intended objective. In some cases, the system specification and policy logic may be well developed, but there is no formal verification. Conversely, formal verification methods are available for several calculi and logics, but none cover mobility. The framework we propose addresses these shortcomings and advances the solution yet another step through automation. Moreover, tableau finiteness is a guarantee that the algorithm terminates, no matter how complex the initial sequent is. The effectiveness of the implementation has been demonstrated in Sect.5.6.

Among the potential directions for extending this work, the natural candidate is the dynamic enforcement of security policies, which is the subject of the following chapter.

Security Policy Enforcement

Chapter 6

Formal Framework for Security Policy Enforcement

Abstract

This chapter presents additional contributions that complete our formal framework. The CS2+ process algebra is a more expressive version of CS2 enriched with an enforcement operator, protection capabilities and communication interfaces. The SPL logic gains a manner of eliminating negative formulas. A quotient operator calculates the required enforcements for non-compliant computer systems. A software prototype has been implemented to show the practical feasibility and the effectiveness of our security policy enforcement framework.

6.1 Introduction

This chapter carries on the system and logic specifications from the previous one and proposes a technique for automatic security policy enforcement in computer systems. New syntactic constructs are added in order to enforce compliance. We demonstrate how, for a given security policy expressed by a logical formula, our calculus allows to asses whether the specification meets the security policy requirements. If it does not, the optimal enforcement for the system is automatically generated using our enforcement operator. A software prototype has been implemented to show the practical feasibility

and the effectiveness of our security policy enforcement framework.

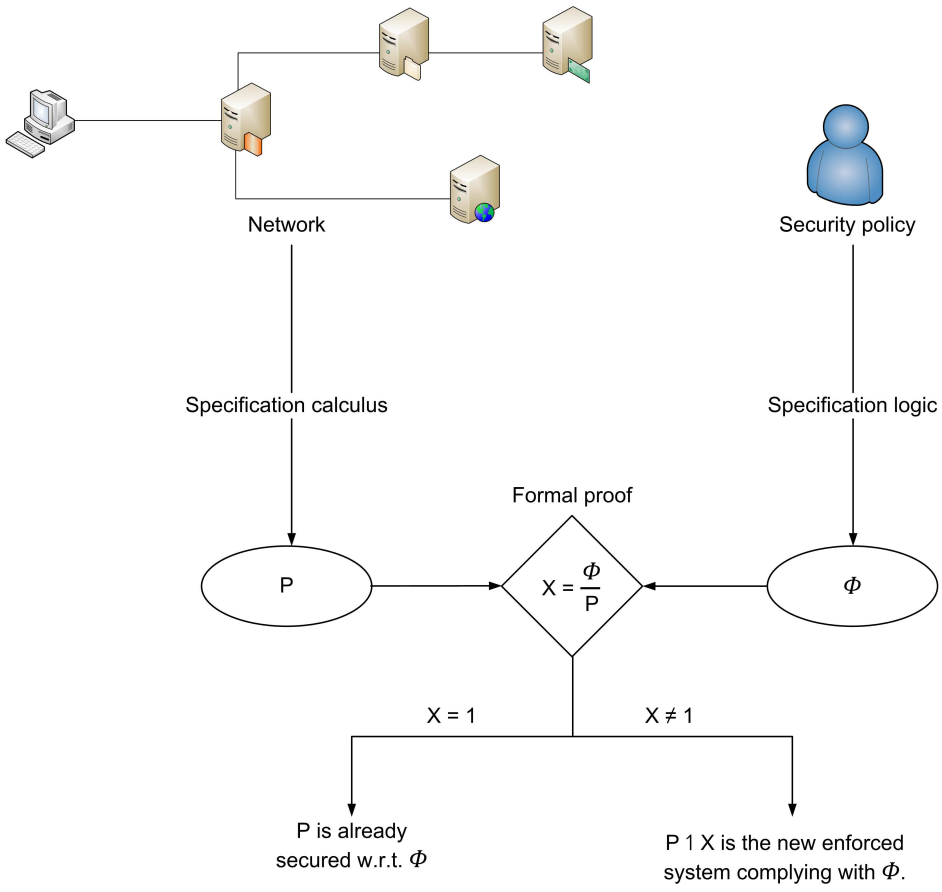
The remainder of this chapter is organized as follows. Section 6.2 summarizes our approach. The new calculus we introduce in Section 6.3 is suitable for specifying systems and policy enforcements. Section 6.4 details our dedicated logic for security policy specification. The quotient operator, defined in Section 6.5, describes our technique for deriving an enforcement from a given security policy and system's specification. The case study depicted in Section 6.6 illustrates how a system can be evaluated for security policy compliance. The case study is expanded to demonstrate how to obtain the enforcement and prove that it does indeed implement the policy. Section 6.7 presents a software prototype implementing the proposed approach. Finally, Section 6.8 expresses our conclusions and some directions for future work.

6.2 Our Approach

The related literature and concepts have already been covered in Chapters 2 and 3. A common observation is that many existing approaches have limited applicability. The informal manner for specifying input information of both system and policy representation impacts the accuracy of the specification. In practical terms, we need to make sure that none of the network components or system's interactions are missing. Access to network shares, for instance, implies the definition of components (users, shared volumes, communication channels, access control lists, etc.), and interactions (allowed operations such as file retrieval and submission, system response in case of insufficient access permissions, etc).

The absence of the aforementioned components or insufficient granularity in describing interactions would make the specification incomplete. It is therefore important to develop formal methods that allow appropriate representations of computer systems and their behavior with respect to a given security policy. The original technique we propose in this chapter addresses the issue of policy compliance for computer networks through formal specification and assessment of the system's security configuration. The approach allows the automatic generation of enforcement processes that have the ability to rewrite a system specification to make it satisfy a security policy.

Figure 6.1: Our approach



The computer network analyzed is specified with our dedicated calculus, which is designed to easily capture the behaviors of the various network components. The security policy is specified using a dedicated logic. Policy compliance can be verified in terms of system and policy specifications. If a policy change is required, our quotient operator allows us to compute the enforcements for the non-compliant components of the system.

Figure 6.1 provides an outline of our approach. The major steps involved are the following:

- the system is specified using the calculus defined in section 6.3; the result is a process P which models, at an abstract level, the system;
- the security policy is specified as a formula Φ with the aid of the *SPL* logic;
- the system's specification P is evaluated for compliance with the formula Φ; if P does not satisfy Φ, an enforcement process X has to be calculated so that the resulting enforced system $\mathsf{P} \upharpoonright \mathsf{X}$ satisfies Φ.

6.3 Security Enforcement Calculus

In this section we define the syntax and the semantics of a calculus suited for specifying, at an abstract level, a given network with the behaviors of network components, including policy enforcement and network protections. We named the calculus $CS2^+$ as it inherits all the elements of *CS2*'s syntax and their associated semantics.

6.3.1 Syntax

The syntax of our specification calculus is presented in Table 6.1. All sets of *CS2* have been preserved. The set of communication channels is denoted by $\mathcal{C}$. The new syntactical components are as follows. The expression $\mathsf{P} \upharpoonright \mathsf{Q}$ describes a process P enforced by a process Q. Two ambients can communicate only if their interfaces share at least a common channel.

Table 6.1: Syntax of *CS2*$^+$

n	$\in$	$\mathcal{N}$	**domain name**
k	$\in$	$\mathcal{K}$	**security key**
a	$\in$	$\mathcal{A}$	**process action**
i	$\subseteq$	$\mathcal{C}$	**communication interface**
P, Q	::=		**processes**
		0	**deadlock**
	$\mid$	1	**successful termination**
	$\mid$	$\mathsf{M}\ \mathsf{Act}$	**action**
	$\mid$	$\mathsf{P}.\mathsf{Q}$	**sequence**
	$\mid$	$\mathsf{P} \mid_\gamma \mathsf{Q}$	**parallel composition**
	$\mid$	$\mathsf{P} \upharpoonright \mathsf{Q}$	**enforcement**
	$\mid$	$\mathsf{P} + \mathsf{Q}$	**choice**
	$\mid$	${}^{k}_{n}[\mathsf{P}]^{i}$	**ambient**
M	::=		**action modalities**
		$\bullet$	**preserve**
	$\mid$	$\blacktriangle$	**remove**
	$\mid$	$\blacktriangledown$	**insert**
Act	::=		**process capabilities**
		$\mathtt{mov}^{k}_{n}$	**movement**
	$\mid$	$\mathtt{prot}^{k}_{n}$	**protection**
	$\mid$	$\mathtt{a}$	**other actions**

There are two types of actions: regular and enforcing. The regular actions enable the execution of process capabilities. The enforcing actions are used for modifying process behaviour. We consider the following particular regular actions in order to define process capabilities: $\mathtt{mov}_n^k$ (movement) and $\mathtt{prot}_n^k$ (protection). Movement represents the ability of a process to circulate in and out of ambients, provided it uses the appropriate key. It has been introduced in the previous chapter within the syntax of *CS2*. Protection refers to an access key change, and is always applied from within an ambient. The process $\mathtt{prot}_n^k.\mathsf{P}$ within an ambient n protected by k' will modify its parent ambient's key to k and then continue executing as P.

We use the modalities "$\bullet\mathsf{a}$", "$\blacktriangle\mathsf{a}$", and "$\blacktriangledown\mathsf{a}$" to depict enforcing actions. They preserve, prevent or to enforce the execution of some action a, respectively. The $\bullet\mathsf{a}$ modality used within an enforcement process ensures that the first action a of a process $\mathsf{a}.\mathsf{P}$ is preserved and executes as scheduled. The opposite effect is achieved through $\blacktriangle\mathsf{a}$, which triggers the removal of the action a from the same process. The $\blacktriangledown\mathsf{a}$ modality allows us to insert the action a as the first executable action of a process P, effectively transforming it into $\mathsf{a}.\mathsf{P}$. Movement actions can be used to describe a system behaviour, so they can be part of a system specification. They are also used for transporting enforcements to the desired location, so movements can also be part of enforcement processes. The action modalities can only exist during the enforcement phase, so they cannot be part of a system specification. Moreover, in order to keep the specification legible, we will denote $\mathtt{mov}_n^\delta$ by $\mathtt{mov}_n$.

6.3.2 Semantics

The operational semantics of *CS2*$^+$ preserves all structural congruences "$\equiv$" and reduction relations "$\rightarrow$" corresponding to elements common with *CS2*. Table 6.2 presents a structural congruence on processes, which includes additional relations for enforcement. Note that unlike parallelism, enforcement is not symmetrical. Moreover, when an enforcement is applied to a process, the right hand term has to terminate before the left hand term starts executing. Table 6.3 defines the predicate "$\_\downarrow$", which is used in the definition the reduction relation. $\mathsf{P}\downarrow$ means that P has the option to terminate successfully.

The reduction relation in Table6.4 captures all possible process evolutions.

Table 6.2: Structural Congruence for $CS2^{+}$

$P \equiv P$	(6.2.1)
$P \equiv Q \Rightarrow Q \equiv P$	(6.2.2)
$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$	(6.2.3)
$P \equiv Q \Rightarrow R.P \equiv R.Q$	(6.2.4)
$P \equiv Q \Rightarrow P.R \equiv Q.R$	(6.2.5)
$P \equiv Q \Rightarrow P \mid_\gamma R \equiv Q \mid_\gamma R$	(6.2.6)
$(P \mid_\gamma Q) \mid_\gamma R \equiv P \mid_\gamma (Q \mid_\gamma R)$	(6.2.7)
$P \equiv Q \Rightarrow P + R \equiv Q + R$	(6.2.8)
$P \equiv Q \Rightarrow R \restriction P \equiv R \restriction Q$	(6.2.9)
$P \equiv Q \Rightarrow P \restriction R \equiv Q \restriction R$	(6.2.10)
$P \equiv Q \Rightarrow {}^{k}_{n}[P]^{i} \equiv {}^{k}_{n}[Q]^{i}$	(6.2.11)
$0.P \equiv 0$	(6.2.12)
$P \restriction 0 \equiv 0$	(6.2.13)
$P + 0 \equiv P$	(6.2.14)
$1.P \equiv P \equiv P.1$	(6.2.15)
$P \mid_\gamma 1 \equiv P$	(6.2.16)
$P \restriction 1 \equiv P \equiv 1 \restriction P$	(6.2.17)
$P + 1 \equiv P$	(6.2.18)
$P.(Q + R) \equiv P.Q + P.R$	(6.2.19)
$(P + Q).R \equiv P.R + Q.R$	(6.2.20)
$P \restriction Q.R \equiv (P \restriction Q) \restriction R$	(6.2.21)
$P \mid_\gamma Q \equiv Q \mid_\gamma P$	(6.2.22)
$P \mid_\gamma (Q + R) \equiv (P \mid_\gamma Q) + (P \mid_\gamma R)$	(6.2.23)
$P + P \equiv P$	(6.2.24)
$P + Q \equiv Q + P$	(6.2.25)
$P \restriction (Q \mid_\gamma R) \equiv (P \restriction Q) \restriction R \equiv (P \restriction R) \restriction Q$	(6.2.26)
$P \restriction (Q + R) \equiv (P \restriction Q) + (P \restriction R)$	(6.2.27)
$(P + Q) \restriction R \equiv (P \restriction R) + (Q \restriction R)$	(6.2.38)
$(P \mid_\gamma Q) \restriction R \equiv (P \restriction R) \mid_\gamma (Q \restriction R)$	(6.2.29)

We use "$\rightarrow^*$" as a transitive closure of "$\rightarrow$". Our syntax allows a complete control over process behaviour with the aid of enforcement processes. Given a process $\mathsf{a}.\mathsf{P}$, an enforcement can allow a to run as scheduled (6.4.10) or remove it (6.4.11). The use of the remove modality is shown in the following example:

$$\mathtt{mov}_n^k.(\mathsf{P}+\mathsf{Q}) \upharpoonright \blacktriangle\mathtt{mov}_n^k.\mathsf{R} \xrightarrow{\blacktriangle\mathtt{mov}_n^k} (\mathsf{P}+\mathsf{Q}) \upharpoonright \mathsf{R}$$

Alternately, an enforcement process can insert a completely new action to be executed as the first action of a process (6.4.12). A combination of removal and insertion is useful, for example, when the network topology changes due to a link being added or failing. In such a case, security policies need to be updated to reflect the affected communication interfaces and alternate paths. They will in turn be implemented through enforcement processes that prescribe the corresponding movement capabilities.

The actual mechanisms for movement inside, out, and across ambients are captured by rules (6.4.13) to (6.4.15). The example below illustrates both (6.4.13) and (6.4.14).

$$\mathtt{mov}_n^k.\mathsf{P} \mid_\gamma {}^k_n[\mathsf{Q} \mid_\gamma \mathsf{R}]^i \mid_\gamma {}^{k'}_m[\mathtt{mov}_m^{k'}.\mathsf{S} \mid_\gamma \mathsf{T}]^j \xrightarrow{\mathtt{mov}_n^k,\, \mathtt{mov}_m^{k'}} {}^k_n[\mathsf{P} \mid_\gamma \mathsf{Q} \mid_\gamma \mathsf{R}]^i \mid_\gamma \mathsf{S} \mid_\gamma {}^{k'}_m[\mathsf{T}]^j$$

Ambient access keys can be modified as well, whenever required by a policy change, in order to lower or elevate the ambient's protection level (6.4.16).

Definition 6.3.1 (Normal form) *A process* P *is in its normal form, denoted by* $\mathsf{P}_\Downarrow$*, iff there is no action* $\mathsf{a} \in \mathcal{A}$ *so that* $\mathsf{P} \xrightarrow{a} \mathsf{P}'$*.*

We extend the structural congruence relation as follows:

$$(\mathsf{P} \upharpoonright \mathsf{Q})_\Downarrow \Rightarrow \mathsf{P} \upharpoonright \mathsf{Q} \equiv \mathsf{P} \qquad (6.2.30)$$

6.4 Security Enforcement Logic

In this section, we define a dedicated logic suited to specify enforceable security policies for computer systems described with our *CS2*$^+$ calculus. The

Table 6.3: Process Termination for $CS2^{+}$

$$1\downarrow \qquad (6.3.1)$$

$$\frac{\mathsf{P}\downarrow}{(\mathsf{P}+\mathsf{Q})\downarrow} \qquad (6.3.2)$$

$$\frac{\mathsf{Q}\downarrow}{(\mathsf{P}+\mathsf{Q})\downarrow} \qquad (6.3.3)$$

$$\frac{\mathsf{P}\downarrow,\ \mathsf{Q}\downarrow}{(\mathsf{P}.\mathsf{Q})\downarrow} \qquad (6.3.4)$$

$$\frac{\mathsf{P}\downarrow,\ \mathsf{Q}\downarrow}{(\mathsf{P}\ |_{\gamma}\ \mathsf{Q})\downarrow} \qquad (6.3.5)$$

$$\frac{\mathsf{P}\downarrow}{{}^{k}_{n}[\mathsf{P}]^{i}\downarrow} \qquad (6.3.6)$$

Table 6.4: Reduction Relation for $CS2^{+}$

$$\frac{\mathsf{P}' \equiv \mathsf{P},\ \mathsf{P} \xrightarrow{a} \mathsf{Q},\ \mathsf{Q} \equiv \mathsf{Q}'}{\mathsf{P}' \xrightarrow{a} \mathsf{Q}'} \quad (6.4.1)$$

$$\frac{\mathsf{P} \xrightarrow{a} \mathsf{P}'}{\mathsf{P}+\mathsf{Q} \xrightarrow{a} \mathsf{P}'} \quad (6.4.2)$$

$$\frac{\mathsf{P} \xrightarrow{a} \mathsf{P}'}{\mathsf{P}.\mathsf{Q} \xrightarrow{a} \mathsf{P}'.\mathsf{Q}} \quad (6.4.3)$$

$$\frac{\mathsf{P} \xrightarrow{a} \mathsf{P}'}{\mathsf{P} \mid_{\gamma} \mathsf{Q} \xrightarrow{a} \mathsf{P}' \mid_{\gamma} \mathsf{Q}} \quad (6.4.4)$$

$$\frac{\mathsf{Q} \xrightarrow{a} \mathsf{Q}'}{\mathsf{P} \upharpoonright \mathsf{Q} \xrightarrow{a} \mathsf{P} \upharpoonright \mathsf{Q}'} \quad (6.4.5)$$

$$\frac{\mathsf{P} \xrightarrow{a} \mathsf{P}',\ \mathsf{Q}\downarrow}{\mathsf{P} \upharpoonright \mathsf{Q} \xrightarrow{a} \mathsf{P}'} \quad (6.4.6)$$

$$\frac{\mathsf{P} \xrightarrow{a} \mathsf{P}',\ \mathsf{Q} \xrightarrow{b} \mathsf{Q}'}{\mathsf{P} \mid_{\gamma} \mathsf{Q} \xrightarrow{\gamma(a,b)} \mathsf{P}' \mid_{\gamma} \mathsf{Q}'} \quad (6.4.7)$$

$$\frac{\mathsf{P} \xrightarrow{a} \mathsf{P}'}{{}^{k}_{n}[\mathsf{P}]^{i} \xrightarrow{a} {}^{k}_{n}[\mathsf{P}']^{i}} \quad (6.4.8)$$

$$\frac{\Box}{\mathsf{a} \xrightarrow{a} 1} \quad (6.4.9)$$

$$\frac{\Box}{\mathsf{a}.\mathsf{P} \upharpoonright \bullet\mathsf{a}.\mathsf{Q} \xrightarrow{\bullet\mathsf{a}} \mathsf{a}.(\mathsf{P} \upharpoonright \mathsf{Q})} \quad (6.4.10)$$

$$\frac{\Box}{\mathsf{a}.\mathsf{P} \upharpoonright \blacktriangle\mathsf{a}.\mathsf{Q} \xrightarrow{\blacktriangle\mathsf{a}} \mathsf{P} \upharpoonright \mathsf{Q}} \quad (6.4.11)$$

$$\frac{\Box}{\mathsf{P} \upharpoonright \blacktriangledown\mathsf{a}.\mathsf{Q} \xrightarrow{\blacktriangledown\mathsf{a}} \mathsf{a}.(\mathsf{P} \upharpoonright \mathsf{Q})} \quad (6.4.12)$$

$$\frac{k' \geq k}{{}^{k}_{n}[\mathsf{P}]^{i} \ddagger \mathtt{mov}^{k'}_{n}.\mathsf{Q} \xrightarrow{\mathtt{mov}^{k'}_{n}} {}^{k}_{n}[\mathsf{P} \ddagger \mathsf{Q}]^{i}} \quad (6.4.13)$$

$$\frac{k' \geq k}{\mathsf{P} \ddagger {}^{k}_{n}[\mathtt{mov}^{k'}_{n}.\mathsf{Q} \mid_{\gamma} \mathsf{R}]^{i} \xrightarrow{\mathtt{mov}^{k'}_{n}} \mathsf{P} \ddagger \mathsf{Q} \mid {}^{k}_{n}[\mathsf{R}]^{i}} \quad (6.4.14)$$

$$\frac{k'' \geq k, k'' \geq k', i \cap j \neq \emptyset}{{}^{k}_{n}[\mathsf{P}]^{i} \ddagger {}^{k'}_{m}[\mathtt{mov}^{k''}_{n}.\mathsf{Q} \mid_{\gamma} \mathsf{R}]^{j} \xrightarrow{\mathtt{mov}^{k''}_{n}} {}^{k}_{n}[\mathsf{P} \ddagger \mathsf{Q}]^{i} \ddagger {}^{k'}_{m}[\mathsf{R}]^{j}} \quad (6.4.15)$$

$$\frac{\Box}{{}^{k}_{n}[\mathsf{P} \upharpoonright \mathtt{prot}^{k'}_{n}.\mathsf{Q}]^{i} \xrightarrow{\mathtt{prot}^{k'}_{n}} {}^{k'}_{n}[\mathsf{P} \upharpoonright \mathsf{Q}]^{i}} \quad (6.4.16)$$

where $\ddagger \in \{|_{\gamma}, \upharpoonright\}$

logic is very similar to the *SPL* logic defined in the previous chapter. However, due to the removal of sequence and the presence channels in the syntax and semantics, we had to give it a different name. The obvious choice was SPL^+, which we are using for consistency with the calculus naming. The syntax of SPL^+ is summarized in Table 6.5 and its semantics are shown in Table 6.6.

Table 6.5: Syntax of SPL^+

Φ, Ψ	$::=$		
	$\not\perp$		True
$\mid$	$\neg\Phi$		Negation
$\mid$	$\langle \mathsf{a} \rangle \Phi,\ \mathsf{a} \in \mathcal{A}$		Capability
$\mid$	$\Phi \mid \Psi$		Parallel Composition
$\mid$	$\Phi \vee \Psi$		Disjunction
$\mid$	${}^{k}_{n}[\Phi]^{i}$		Protected location

Lemma 6.4.1 *Let* Φ *and* Ψ *be two logical formulas and let* P *be a process. Then:*

$$\mathsf{P} \in [\![\ \Phi\ ]\!] \quad \wedge \quad [\![\ \Phi\ ]\!] \subseteq [\![\ \Psi\ ]\!] \ \Rightarrow\ \mathsf{P} \models \Psi$$

Proof:
The proof is trivial. □

One problem that may arise during enforcements is the interpretation of the negation operator when used with complex formulas. We need to propagate the negation operator inside the formulas in order to limit its scope to atomic actions. This transformation will be of great help to simplify the definition of the quotient operator given in Table 6.8 in the next section. Table 6.7 presents a rewriting system that allows the calculation of such transformations. It is easy to verify that all applied transformations lead to equivalent logical formulas.

Table 6.6: Semantics of SPL^{+}

$$[\![\not\!\!t]\!] = \mathcal{P} \setminus \{0\} \quad (6.6.1)$$

$$[\![ff]\!] = \{0\} \quad (6.6.2)$$

$$[\![\neg\Phi]\!] = \mathcal{P} \setminus [\![\Phi]\!] \quad (6.6.3)$$

$$[\![\langle \mathsf{a} \rangle \Phi]\!] = \{\mathsf{a}.\mathsf{P} \in \mathcal{P} \ : \ \mathsf{P} \in [\![\Phi]\!]\} \quad (6.6.4)$$

$$[\![\Phi \mid \Psi]\!] = \{\mathsf{P} \mid \mathsf{Q} \in \mathcal{P} \ : \ \mathsf{P} \in [\![\Phi]\!] \ \wedge \ \mathsf{Q} \in [\![\Psi]\!]\} \quad (6.6.5)$$

$$[\![\Phi \vee \Psi]\!] = [\![\Phi]\!] \cup [\![\Psi]\!] \quad (6.6.6)$$

$$[\![{}^{k}_{n}[\Phi]^{i}]\!] = \{{}^{k}_{n}[\mathsf{P}]^{i} \in \mathcal{P} \ : \ \mathsf{P} \in [\![\Phi]\!]\} \quad (6.6.7)$$

Table 6.7: The elimination of the $\neg\Phi$ form

$$\neg\not\!\!t \ \rightarrow \ ff \quad (6.7.1)$$

$$\neg ff \ \rightarrow \ \not\!\!t \quad (6.7.2)$$

$$\neg\neg\Phi \ \rightarrow \ \Phi \quad (6.7.3)$$

$$\neg(\langle \mathsf{a} \rangle \Phi) \ \rightarrow \ (\neg\langle \mathsf{a} \rangle)\not\!\!t \vee \langle \mathsf{a} \rangle \neg\Phi \quad (6.7.4)$$

$$\neg(\Phi \mid \Psi) \ \rightarrow \ \not\!\!t \mid \neg\Psi \vee \neg\Phi \mid \not\!\!t \vee \neg\Phi \mid \neg\Psi \quad (6.7.5)$$

$$\neg(\Phi \vee \Psi) \ \rightarrow \ \neg\Phi \wedge \neg\Psi \quad (6.7.6)$$

$$\neg({}^{k}_{n}[\Phi]^{i}) \ \rightarrow \ {}^{k}_{n}[\neg\Phi]^{i} \quad (6.7.7)$$

The following example demonstrates how the rewriting system can be applied to push the negation operator down to atomic actions.

$$\begin{aligned}
\neg(\langle \mathtt{mov}_n^k \rangle \langle \mathtt{mov}_m^{k'} \rangle tt) &\xrightarrow{(7.4)} (\neg\langle \mathtt{mov}_n^k \rangle) tt \vee \langle \mathtt{mov}_n^k \rangle \neg(\langle \mathtt{mov}_m^{k'} \rangle tt) \\
&\xrightarrow{(7.4)} (\neg\langle \mathtt{mov}_n^k \rangle) tt \vee \langle \mathtt{mov}_n^k \rangle((\neg\langle \mathtt{mov}_m^{k'} \rangle) tt) \vee \langle \mathtt{mov}_m^{k'} \rangle \neg tt) \\
&\xrightarrow{(7.1)} (\neg\langle \mathtt{mov}_n^k \rangle) tt \vee \langle \mathtt{mov}_n^k \rangle(\neg\langle \mathtt{mov}_m^{k'} \rangle) tt \vee \langle \mathtt{mov}_n^k \rangle \langle \mathtt{mov}_m^{k'} \rangle ff
\end{aligned}$$

6.5 Security Policy Enforcement

Once the system and the security policy have been specified with our algebraic calculus and logic, the next important step is to extract automatically an enforcement process that imposes the behavior of the system's model as stated by the policy. The main intent of this approach is to automatically identify the enforcement components that allow to impose a given policy on a system.

This can be viewed as an alternative representation of the interface equation problem [97, 62, 85, 79]. The idea is to derive a quotient process from two given processes. The derived quotient process represents, then, the missing part for the two processes to be equivalent. The quotient is defined in terms of a process and a logical formula. Therefore, the equation we need to solve has the form:

$$\mathsf{P} \upharpoonright \mathsf{X} \models \Phi$$

where P is the formal description of a system and Φ is the security policy to be enforced.
In the equation shown above, X can be expressed as a quotient. The solution X of this equation is the enforcement we look for:

$$\mathsf{X} = \frac{\Phi}{\mathsf{P}}$$

We have shown in Table 6.2 that enforcements don't have any effect on a deadlock process. Also, it is obvious that there is no need for an enforcement if a process already satisfies the policy. For all other situations, the quotient operator $\frac{=}{-}$ is formally defined in Table 6.8. Several examples of process enforcements are also provided in this section.

Table 6.8: Quotient Operator
$\frac{\overline{}}{\overline{}} : \mathcal{F} \times (\mathcal{P} \setminus \{0\}) \to \mathcal{P}$

$$\frac{\#}{\mathsf{P}} = 1 \tag{6.8.1}$$

$$\frac{ff}{\mathsf{P}} = 0 \tag{6.8.2}$$

$$\frac{\langle \mathtt{a} \rangle \Phi}{\mathtt{a}.\mathsf{P}} = \bullet \mathtt{a}.\frac{\Phi}{\mathsf{P}} \tag{6.8.3}$$

$$\frac{\langle \mathtt{a} \rangle \Phi}{\mathtt{b}.\mathsf{P}} = \blacktriangle \mathtt{b}.\frac{\Phi}{\mathsf{P}}.\blacktriangledown \mathtt{a} \qquad \mathtt{a} \neq \mathtt{b} \tag{6.8.4}$$

$$\frac{\neg\langle \mathtt{a} \rangle \Phi}{\mathtt{a}.\mathsf{P}} = \blacktriangle \mathtt{a}.\frac{\Phi}{\mathsf{P}} \tag{6.8.5}$$

$$\frac{\neg\langle \mathtt{a} \rangle \Phi}{\mathtt{b}.\mathsf{P}} = \bullet \mathtt{b}.\frac{\Phi}{\mathsf{P}} \qquad \mathtt{a} \neq \mathtt{b} \tag{6.8.6}$$

$$\frac{\langle \mathtt{a} \rangle \Phi}{1} = \frac{\Phi}{1}.\blacktriangledown \mathtt{a} \tag{6.8.7}$$

$$\frac{\Phi \vee \Psi}{\mathsf{P}} = \frac{\Phi}{\mathsf{P}} + \frac{\Psi}{\mathsf{P}} \tag{6.8.8}$$

$$\frac{\Phi}{\mathsf{P} + \mathsf{Q}} = \frac{\Phi}{\mathsf{P}} + \frac{\Phi}{\mathsf{Q}} \tag{6.8.9}$$

$$\frac{{}^{k}_{n}[\Phi]^{i}}{{}^{k}_{n}[\mathsf{P}]^{i}} = \mathtt{mov}^{k}_{n}.\frac{\Phi}{\mathsf{P}} \tag{6.8.10}$$

$$\frac{{}^{k'}_{n}[\Phi]^{i}}{{}^{k}_{n}[\mathsf{P}]^{i}} = \mathtt{mov}^{k}_{n}.\mathtt{prot}^{k'}_{n}.\frac{\Phi}{\mathsf{P}} \qquad k \neq k' \tag{6.8.11}$$

$$\frac{{}^{k'}_{n}[\Phi]^{i} \mid \Psi}{{}^{k}_{n}[\mathsf{P}]^{i} \mid_{\gamma} \mathsf{Q}} = \frac{{}^{k'}_{n}[\Phi]^{i}}{{}^{k}_{n}[\mathsf{P}]^{i}} \mid_{\gamma} \frac{\Psi}{\mathsf{Q}} \tag{6.8.12}$$

All processes, except the blocking process, satisfy the formula tt (6.8.1), so there is nothing to enforce (i.e. the value of the quotient is 1). Since no valid process can satisfy ff, the generated enforcement at (6.8.2) is 0, which has the effect of blocking the system. The rules (6.8.3)-(6.8.7) are related to the enforcement of process actions. The rule (6.8.3) applies when the process starts by the action specified in the formula. The enforcement process must then keep the action in place, therefore the action $\bullet\mathtt{a}$ is generated for the enforcement process.

The rule (6.8.6) is quite similar, with $\bullet\mathtt{b}$ preserving in this case an action which is different from the action $\mathtt{a}$ prohibited by the policy. If the desired action is different, as in (6.8.4), that particular action needs to be enforced. This means that the non-compliant action $\mathtt{b}$ needs to be neutralized first, followed by the insertion of the new action $\mathtt{a}$. An fitting example would be the use of a new access key k'' for a movement action instead of the obsolete key k. Let $\mathsf{P} = \mathtt{mov}_n^k.\mathtt{mov}_m^{k'}.1$ and let $\Phi = \langle \mathtt{mov}_n^{k''} \rangle tt$. Notice that the formula specifies that the movement action needs to use the new key. The enforcement in this case is given by the expression:

$$\frac{\Phi}{\mathsf{P}} = \mathsf{X} = \blacktriangle\mathtt{mov}_n^k.\frac{tt}{\mathtt{mov}_m^{k'}.1}.\blacktriangledown\mathtt{mov}_n^{k''} = \blacktriangle\mathtt{mov}_n^k.1.\blacktriangledown\mathtt{mov}_n^{k''} \equiv \blacktriangle\mathtt{mov}_n^k.\blacktriangledown\mathtt{mov}_n^{k''}$$

Therefore, $\mathsf{P} \upharpoonright \mathsf{X} = \mathtt{mov}_n^{k''}.\mathtt{mov}_m^{k'}$.

The rule (6.8.5) follows the same reasoning, but in this case we seek satisfaction of the complementary action. The rule (6.8.7) applies to the enforcement of a policy on the process 1. The rule (6.8.8) concerns the enforcement of a disjunction of two formulas forming the policy. Either $\frac{\Phi}{\mathsf{P}}$ or $\frac{\Psi}{\mathsf{P}}$ will produce a compliant process, when enforced on P. In the case of the indeterminate choice (6.8.9), the formula can be applied to either P or Q. Let $\mathsf{P} = \mathtt{mov}_n^k.\mathsf{P}'$, $\mathsf{Q} = \mathtt{mov}_m^{k'}.\mathsf{Q}'$ and let $\Phi = \langle \mathtt{mov}_n^k \rangle tt$. The enforcement for $\mathsf{P} + \mathsf{Q}$ is:

$$\frac{\Phi}{\mathsf{P}+\mathsf{Q}} = \bullet\mathtt{mov}_n^k.\frac{tt}{\mathsf{P}'} + \blacktriangle\mathtt{mov}_m^{k'}.\frac{tt}{\mathsf{Q}'}.\blacktriangledown\mathtt{mov}_n^k = \bullet\mathtt{mov}_n^k.1 + \blacktriangle\mathtt{mov}_m^{k'}.\blacktriangledown\mathtt{mov}_n^k$$

Ambient enforcement, with and without access key changes, is defined in

rules (6.8.10) and (6.8.11). The rule (6.8.10) allows the enforcement process to go inside an ambient and apply the enforcement. The modification of the ambient protection key is given through rule (6.8.11). Targeted enforcement is also supported by our enforcement operator, as shown by (6.8.12). The ambient name permits the delivery of dedicated enforcements on different locations within the process. Let $\Phi = \not\Box$, $\Psi = \langle \mathtt{mov}_m^{k'} \rangle \not\Box$, $\mathsf{P} = \mathtt{mov}_m^{k''}.\mathsf{P}'$, and $\mathsf{Q} = \mathtt{mov}_n^{k}.\mathsf{Q}'$. The enforcement X for the formula ${}_n^k[\Phi]^i \mid {}_m^{k''}[\Psi]^j$ on the process ${}_m^{k''}[\mathsf{P}]^j \mid_\gamma {}_n^{k'''}[\mathsf{Q}]^i$ is calculated as follows:

$$\frac{{}_n^k[\Phi]^i \mid {}_m^{k''}[\Psi]^j}{{}_m^{k''}[\mathsf{P}]^j \mid_\gamma {}_n^{k'''}[\mathsf{Q}]^i} = \frac{{}_n^k[\not\Box]^i}{{}_n^{k'''}[\mathtt{mov}_n^k.\mathsf{Q}']^i} \mid_\gamma \frac{{}_m^{k''}[\langle \mathtt{mov}_m^{k'} \rangle \not\Box]^j}{{}_m^{k''}[\mathtt{mov}_m^{k''}.\mathsf{P}']^j}$$

The results of the two new quotients obtained are:

$$\frac{{}_n^k[\not\Box]^i}{{}_n^{k'''}[\mathtt{mov}_n^k.\mathsf{Q}']^i} = \mathtt{mov}_n^{k'''}.\mathtt{prot}_n^k.\frac{\not\Box}{\mathtt{mov}_n^k.\mathsf{Q}'} = \mathtt{mov}_n^{k'''}.\mathtt{prot}_n^k.1$$

$$\frac{{}_m^{k''}[\langle \mathtt{mov}_m^{k'} \rangle \not\Box]^j}{{}_m^{k''}[\mathtt{mov}_m^{k''}.\mathsf{P}']^j} = \mathtt{mov}_m^{k''}.\blacktriangle \mathtt{mov}_m^{k''}.\frac{\not\Box}{\mathsf{P}'}.\blacktriangledown \mathtt{mov}_m^{k'} = \mathtt{mov}_m^{k''}.\blacktriangle \mathtt{mov}_m^{k''}.\blacktriangledown \mathtt{mov}_m^{k'}.$$

Therefore value of the enforcement is:

$$\mathsf{X} = \mathtt{mov}_n^{k'''}.\mathtt{prot}_n^k.1 \mid_\gamma \mathtt{mov}_m^{k''}.\blacktriangle \mathtt{mov}_m^{k''}.\blacktriangledown \mathtt{mov}_m^{k'}$$

Theorem 6.5.1 (Enforcement Correctness) *Let* $\Phi \in \mathcal{F}, \mathsf{P} \in \mathcal{P} \setminus \{0\}$, *and let* $\mathsf{X} = \dfrac{\Phi}{\mathsf{P}}$. *Then:*

$$\mathsf{P} \upharpoonright \mathsf{X} \models \Phi$$

Proof:
The proof is done by structural induction on Φ.

☑ $\Phi = \not\Box$:

$$\mathsf{X} = \frac{\not\Box}{\mathsf{P}} = 1 \qquad (by\ 6.8.1)$$

$$\mathsf{P} \upharpoonright 1 \equiv \mathsf{P} \qquad (by\ 6.2.17)$$

$$\mathsf{P} \models \not\Box \qquad (by\ 6.6.1)$$

☑ $\Phi = ff$:

$\mathsf{X} = \dfrac{ff}{\mathsf{P}} \;=\; 0$ (*by* 6.8.2)

$\mathsf{P} \upharpoonright 0 \;\equiv\; 0$ (*by* 6.2.13)

$0 \models ff$ (*by* 6.6.2)

☑ $\Phi = \langle \mathsf{a} \rangle \Psi$:

✓ $\mathsf{P} = \mathsf{a}.\mathsf{Q}$

$\mathsf{X} = \dfrac{\langle \mathsf{a} \rangle \Psi}{\mathsf{a}.\mathsf{Q}} \;=\; \bullet\mathsf{a}.\dfrac{\Psi}{\mathsf{Q}}$ (*by* 6.8.3)

$\mathsf{a}.\mathsf{Q} \upharpoonright \bullet\mathsf{a}.\dfrac{\Psi}{\mathsf{Q}} \xrightarrow{\bullet\mathsf{a}} \mathsf{a}.(\mathsf{Q} \upharpoonright \dfrac{\Psi}{\mathsf{Q}})$ (*by* 6.4.10)

$\mathsf{Q} \upharpoonright \dfrac{\Psi}{\mathsf{Q}} \models \Psi$ (*by hypothesis*)

$\mathsf{a}.(\mathsf{Q} \upharpoonright \dfrac{\Psi}{\mathsf{Q}}) \models \langle \mathsf{a} \rangle \Psi$ (*by* 6.6.4)

✓ $\mathsf{P} = \mathsf{b}.\mathsf{Q}, \mathsf{b} \neq \mathsf{a}$

$\mathsf{X} = \dfrac{\langle \mathsf{a} \rangle \Psi}{\mathsf{b}.\mathsf{Q}} \;=\; \blacktriangle\mathsf{b}.\dfrac{\Psi}{\mathsf{Q}}.\blacktriangledown\mathsf{a}$ (*by* 6.8.4)

$\mathsf{b}.\mathsf{Q} \upharpoonright \blacktriangle\mathsf{b}.\dfrac{\Psi}{\mathsf{Q}}.\blacktriangledown\mathsf{a} \xrightarrow{\blacktriangle\mathsf{b}} \mathsf{Q} \upharpoonright \dfrac{\Psi}{\mathsf{Q}}.\blacktriangledown\mathsf{a}$ (*by* 6.4.11)

$\mathsf{Q} \upharpoonright \dfrac{\Psi}{\mathsf{Q}}.\blacktriangledown\mathsf{a} \;\equiv\; (\mathsf{Q} \upharpoonright \dfrac{\Psi}{\mathsf{Q}}) \upharpoonright \blacktriangledown\mathsf{a}$ (*by* 6.2.21)

$(\mathsf{Q} \upharpoonright \dfrac{\Psi}{\mathsf{Q}}) \upharpoonright \blacktriangledown\mathsf{a} \xrightarrow{\blacktriangledown\mathsf{a}} \mathsf{a}.(\mathsf{Q} \upharpoonright \dfrac{\Psi}{\mathsf{Q}})$ (*by* 6.4.12)

$\mathsf{a}.(\mathsf{Q} \upharpoonright \dfrac{\Psi}{\mathsf{Q}}) \models \langle \mathsf{a} \rangle \Psi$ (*by* 6.6.4)

☑ $\Phi = \neg\langle \mathsf{a}\rangle\Psi$:

✓ $\mathsf{P} = \mathsf{a}.\mathsf{Q}$

$\mathsf{X} = \dfrac{\neg\langle \mathsf{a}\rangle\Psi}{\mathsf{a}.\mathsf{Q}} = \blacktriangle\mathsf{a}.\dfrac{\Psi}{\mathsf{Q}}$ (*by* 6.8.5)

$\mathsf{a}.\mathsf{Q} \upharpoonright \blacktriangle\mathsf{a}.\dfrac{\Psi}{\mathsf{Q}} \xrightarrow{\blacktriangle\mathsf{a}} \mathsf{Q} \upharpoonright \dfrac{\Psi}{\mathsf{Q}}$ (*by* 6.4.11)

$\mathsf{Q} \upharpoonright \dfrac{\Psi}{\mathsf{Q}} \models \Psi$ (*by hypothesis*)

$\mathsf{Q} \upharpoonright \dfrac{\Psi}{\mathsf{Q}} \not\models \langle \mathsf{a}\rangle\Psi$ (*by* 6.6.4)

$\mathsf{Q} \upharpoonright \dfrac{\Psi}{\mathsf{Q}} \models \neg\langle \mathsf{a}\rangle\Psi$ (*by* 6.6.3)

✓ $\mathsf{P} = \mathsf{b}.\mathsf{Q}, \mathsf{b} \neq \mathsf{a}$

$\mathsf{X} = \dfrac{\neg\langle \mathsf{a}\rangle\Psi}{\mathsf{b}.\mathsf{Q}} = \bullet\mathsf{b}.\dfrac{\Psi}{\mathsf{Q}}$ (*by* 6.8.5)

$\mathsf{b}.\mathsf{Q} \upharpoonright \bullet\mathsf{b}.\dfrac{\Psi}{\mathsf{Q}} \xrightarrow{\bullet\mathsf{b}} \mathsf{b}.(\mathsf{Q} \upharpoonright \dfrac{\Psi}{\mathsf{Q}})$ (*by* 6.4.10)

$\mathsf{Q} \upharpoonright \dfrac{\Psi}{\mathsf{Q}} \models \Psi$ (*by hypothesis*)

$\mathsf{b}.(\mathsf{Q} \upharpoonright \dfrac{\Psi}{\mathsf{Q}}) \models \langle \mathsf{b}\rangle\Psi$ (*by* 6.6.4)

$\mathsf{b} \neq \mathsf{a} \Rightarrow \mathsf{b}.(\mathsf{Q} \upharpoonright \dfrac{\Psi}{\mathsf{Q}}) \not\models \langle \mathsf{a}\rangle\Psi$ (*by* 6.6.4)

$\mathsf{b}.(\mathsf{Q} \upharpoonright \dfrac{\Psi}{\mathsf{Q}}) \models \neg\langle \mathsf{a}\rangle\Psi$ (*by* 6.6.3)

$$\boxtimes\ \Phi\ =\ {}^{k'}_{n}[\Phi']^i \mid \Psi \text{ and } \mathsf{P}\ =\ {}^{k}_{n}[\mathsf{P}']^i\ \mid_\gamma\ \mathsf{Q}\text{:}$$

$$\mathsf{X} = \frac{{}^{k'}_{n}[\Phi']^i \mid \Psi}{{}^{k}_{n}[\mathsf{P}']^i\ \mid_\gamma\ \mathsf{Q}} = \frac{{}^{k'}_{n}[\Phi']^i}{{}^{k}_{n}[\mathsf{P}']^i}\ \mid_\gamma\ \frac{\Psi}{\mathsf{Q}} \qquad (by\ 6.8.12)$$

$$\mathsf{P} \upharpoonright \mathsf{X} \equiv ({}^{k}_{n}[\mathsf{P}']^i \upharpoonright (\frac{{}^{k'}_{n}[\Phi']^i}{{}^{k}_{n}[\mathsf{P}']^i}\ \mid_\gamma\ \frac{\Psi}{\mathsf{Q}})) \mid_\gamma (\mathsf{Q} \upharpoonright (\frac{{}^{k'}_{n}[\Phi']^i}{{}^{k}_{n}[\mathsf{P}']^i}\ \mid_\gamma\ \frac{\Psi}{\mathsf{Q}}))$$

$$(by\ 6.2.29)$$

$$\equiv (((({}^{k}_{n}[\mathsf{P}']^i \upharpoonright \frac{{}^{k'}_{n}[\Phi']^i}{{}^{k}_{n}[\mathsf{P}']^i}) \upharpoonright \frac{\Psi}{\mathsf{Q}}) + (({}^{k}_{n}[\mathsf{P}']^i \upharpoonright \frac{\Psi}{\mathsf{Q}}) \upharpoonright \frac{{}^{k'}_{n}[\Phi']^i}{{}^{k}_{n}[\mathsf{P}']^i})) \mid_\gamma$$

$$(((\mathsf{Q} \upharpoonright \frac{\Psi}{\mathsf{Q}}) \upharpoonright \frac{{}^{k'}_{n}[\Phi']^i}{{}^{k}_{n}[\mathsf{P}']^i}) + ((\mathsf{Q} \upharpoonright \frac{{}^{k'}_{n}[\Phi']^i}{{}^{k}_{n}[\mathsf{P}']^i}) \upharpoonright \frac{\Psi}{\mathsf{Q}}))$$

$$(by\ 6.2.26)$$

$${}^{k}_{n}[\mathsf{P}']^i \upharpoonright \frac{{}^{k'}_{n}[\Phi']^i}{{}^{k}_{n}[\mathsf{P}']^i} \models {}^{k'}_{n}[\Phi']^i \qquad (by\ induction)$$

By the quotient operator and the reduction relation, we have:

$$\frac{{}^{k'}_{n}[\Phi']^i}{{}^{k}_{n}[\mathsf{P}']^i} = \mathtt{mov}^{k}_{n}.\mathsf{R} \text{ and } {}^{k}_{n}[\mathsf{P}']^i \upharpoonright \frac{{}^{k'}_{n}[\Phi']^i}{{}^{k}_{n}[\mathsf{P}']^i} \rightarrow^* {}^{k'}_{n}[\mathsf{S}]^i \quad \mathsf{R}, \mathsf{S} \in \mathcal{P} \quad (6.\mathbf{9}.\mathbf{1})$$

Since domains are uniquely identified, we have:

$$\frac{\Psi}{\mathsf{Q}} \neq \mathtt{mov}^{-}_{n}.\mathsf{T} \quad \mathsf{T} \in \mathcal{P} \qquad (6.\mathbf{9}.\mathbf{2})$$

$$({}^{k}_{n}[\mathsf{P}']^i \upharpoonright \frac{{}^{k'}_{n}[\Phi]^i}{{}^{k}_{n}[\mathsf{P}']^i}) \upharpoonright \frac{\Psi}{\mathsf{Q}} = {}^{k'}_{n}[\mathsf{S}]^i \upharpoonright \frac{\Psi}{\mathsf{Q}} \qquad (6.9.2.1)$$

By (6.9.2), there is no possible reduction for (6.9.2.1). We then have:

$$({}^{k'}_{n}[\mathsf{S}]^i \upharpoonright \frac{\Psi}{\mathsf{Q}})_{\Downarrow} \qquad (6.9.2.2)$$

By (6.9.2.1) and(6.9.2.2) we have:

$$({}^{k}_{n}[\mathsf{P}']^{i} \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}}) \upharpoonright \frac{\Psi}{\mathsf{Q}} \equiv {}^{k}_{n}[\mathsf{P}']^{i} \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}} \qquad (6.\mathbf{9.3})$$

$$({}^{k}_{n}[\mathsf{P}']^{i} \upharpoonright \frac{\Psi}{\mathsf{Q}})_{\Downarrow} \qquad (by\ 6.9.2)$$

$${}^{k}_{n}[\mathsf{P}']^{i} \upharpoonright \frac{\Psi}{\mathsf{Q}} \equiv {}^{k}_{n}[\mathsf{P}']^{i} \qquad (by\ 6.2.30)$$

$$({}^{k}_{n}[\mathsf{P}']^{i} \upharpoonright \frac{\Psi}{\mathsf{Q}}) \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}} \equiv {}^{k}_{n}[\mathsf{P}']^{i} \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}} \qquad (6.\mathbf{9.4})$$

Since domains are uniquely identified, we have:

$$\mathsf{Q} \neq {}^{k}_{n}[\mathsf{U}]^{i} \quad \mathsf{U} \in \mathcal{P}$$

By the quotient operator and (6.9.1), we have:

$$(\mathsf{Q} \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}})_{\Downarrow} \qquad (by\ 6.2.30)$$

$$(\mathsf{Q} \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}}) \upharpoonright \frac{\Psi}{\mathsf{Q}} \equiv \mathsf{Q} \upharpoonright \frac{\Psi}{\mathsf{Q}} \qquad (6.\mathbf{9.5})$$

Since domains are uniquely identified, we have:

$$\mathsf{Q} \upharpoonright \frac{\Psi}{\mathsf{Q}} \neq {}^{k}_{n}[\mathsf{V}]^{i} \quad \mathsf{V} \in \mathcal{P}$$

$$((\mathsf{Q} \upharpoonright \frac{\Psi}{\mathsf{Q}}) \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}})_{\Downarrow} \qquad (by\ 6.9.1, 6.2.30)$$

$$(\mathsf{Q} \upharpoonright \frac{\Psi}{\mathsf{Q}}) \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}} \equiv \mathsf{Q} \upharpoonright \frac{\Psi}{\mathsf{Q}} \qquad (6.\mathbf{9.6})$$

$$\mathsf{P} \upharpoonright \mathsf{X} \rightarrow ({}^{k}_{n}[\mathsf{P}']^{i} \upharpoonright \frac{{}^{k'}_{n}[\Phi']^{i}}{{}^{k}_{n}[\mathsf{P}']^{i}}) \mid_{\gamma} (\mathsf{Q} \upharpoonright \frac{\Psi}{\mathsf{Q}}) \models {}^{k'}_{n}[\Phi']^{i} \mid \Psi$$

$$(by\ 6.9.3 - 6.9.6, 6.6.5, 6.6.7)$$

☑ $\Phi = \Phi' \vee \Psi$:

$$\mathsf{X} = \frac{\Phi' \vee \Psi}{\mathsf{P}} = \frac{\Phi'}{\mathsf{P}} + \frac{\Psi}{\mathsf{P}} \qquad (by\ 6.8.8)$$

$$\mathsf{P} \upharpoonright (\frac{\Phi'}{\mathsf{P}} + \frac{\Psi}{\mathsf{P}}) \equiv (\mathsf{P} \upharpoonright \frac{\Phi'}{\mathsf{P}}) + (\mathsf{P} \upharpoonright \frac{\Psi}{\mathsf{P}}) \qquad (by\ 6.2.27)$$

$$\mathsf{P} \upharpoonright \frac{\Phi'}{\mathsf{P}} \models \Phi' \quad \text{and} \quad \mathsf{P} \upharpoonright \frac{\Psi}{\mathsf{P}} \models \Psi \qquad (by\ \textit{hypothesis})$$

$$\mathsf{P} \upharpoonright \frac{\Phi'}{\mathsf{P}} \models \Phi' \vee \Psi \quad \text{and} \quad \mathsf{P} \upharpoonright \frac{\Psi}{\mathsf{P}} \models \Phi' \vee \Psi \qquad (by\ 6.6.6)$$

$$(\mathsf{P} \upharpoonright \frac{\Phi'}{\mathsf{P}}) + (\mathsf{P} \upharpoonright \frac{\Psi}{\mathsf{P}}) \models \Phi' \vee \Psi \qquad (by\ 6.6.6)$$

☑ $\Phi =_n^k [\Psi]^i$:

✓ $\mathsf{P} =_n^k [\mathsf{Q}]^i$

$$\mathsf{X} = \frac{{}_n^k[\Psi]^i}{{}_n^k[\mathsf{Q}]^i} = \mathtt{mov}_n^k.\frac{\Psi}{\mathsf{Q}} \qquad (by\ 6.8.10)$$

$${}_n^k[\mathsf{Q}]^i \upharpoonright \mathtt{mov}_n^k.\frac{\Psi}{\mathsf{Q}} \xrightarrow{\mathtt{mov}_n^k} {}_n^k[\mathsf{Q} \upharpoonright \frac{\Psi}{\mathsf{Q}}]^i \qquad (by\ 6.4.13)$$

$$\mathsf{Q} \upharpoonright \frac{\Psi}{\mathsf{Q}} \models \Psi \qquad (by\ \textit{hypothesis})$$

$${}_n^k[\mathsf{Q} \upharpoonright \frac{\Psi}{\mathsf{Q}}]^i \models {}_n^k[\Psi]^i \qquad (by\ 6.6.7)$$

✓ $\mathsf{P} =_n^{k'} [\mathsf{Q}]^i, k' \neq k$

$$\mathsf{X} = \frac{{}_n^k[\Psi]^i}{{}_{k'}^{n}[\mathsf{Q}]^i} = \mathtt{mov}_n^{k'}.\mathtt{prot}_n^k.\frac{\Psi}{\mathsf{Q}} \qquad (by\ 6.8.11)$$

$${}_n^{k'}[\mathsf{Q}]^i \upharpoonright \mathtt{mov}_n^{k'}.\mathtt{prot}_n^k.\frac{\Psi}{\mathsf{Q}} \xrightarrow{\mathtt{mov}_n^{k'}} {}_n^{k'}[\mathsf{Q} \upharpoonright \mathtt{prot}_n^k.\frac{\Psi}{\mathsf{Q}}]^i \qquad (by\ 6.4.13)$$

$$ {}^{k'}_{n}[\mathsf{Q} \upharpoonright \mathtt{prot}^{k}_{n}.\frac{\Psi}{\mathsf{Q}}]^{i} \xrightarrow{\mathtt{prot}^{k}_{n}} {}^{k}_{n}[\mathsf{Q} \upharpoonright \frac{\Psi}{\mathsf{Q}}]^{i} \quad (by\ 6.4.16) $$

$$ \mathsf{Q} \upharpoonright \frac{\Psi}{\mathsf{Q}} \models \Psi \quad (by\ hypothesis) $$

$$ {}^{k'}_{n}[\mathsf{Q} \upharpoonright \frac{\Psi}{\mathsf{Q}}]^{i} \models {}^{k}_{n}[\Psi]^{i} \quad (by\ 6.6.7) $$

□

6.6 Case Study

In this section, we illustrate our technique with the example of the network depicted in Fig.6.2. The example demonstrates the use of the new calculus for specifying the system, policy changes, enforcement and verification. We show that the generated enforcement policy produces the expected changes.

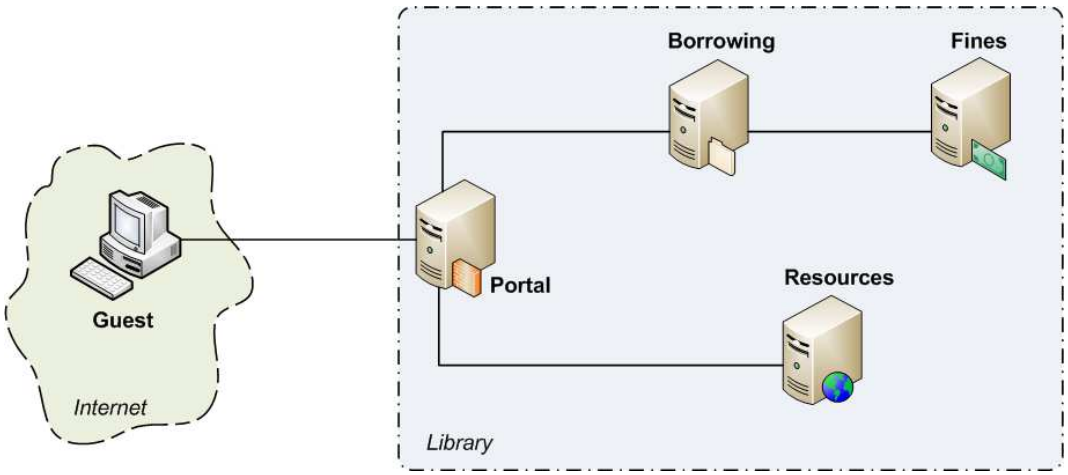
6.6.1 System Specification

The subject of the case study is a simplified version of a library system as depicted in Fig. 6.2. In order to make the example easy to comprehend, the number of computer systems has been kept to a minimum: one for a library guest and four for the library. We identified two logical zones: the *Internet* logical zone containing the Guest system and the *Library* logical zone containing the library's own computer systems. The *Library* zone initially contains one computer system for the library's portal server (Portal), two for the reservation and fine payments (Borrowing and Fines) and one for online resources (Resources).

Each computer system is represented by a non-blocking process running in a protected ambient. Specific keys are only defined for ambients requiring safeguards: k_l and k_f corresponding to portal and borrowing systems, respectively. All other keys are set to the default value δ, which states that there are no access restrictions. For ease of reading, we have chosen to omit δ from the specification. This means that processes $\mathtt{mov}^{\delta}_{n}.\mathsf{P}$ and ${}^{\delta}_{n}[\mathsf{P}]^{i_n}$ will be represented as $\mathtt{mov}_n.\mathsf{P}$ and ${}_n[\mathsf{P}]^{i_n}$, respectively. For the same reason, by abuse of notation, we will denote "$|_\gamma$" by "$|$", since all communications are

well defined. Access to the library portal (provided that k_l is known) does not grant access to Borrowing, but is required as a preliminary step.

Figure 6.2: Case study - Library network



Currently, guests can browse online resources, reserve items from the library and pay fines for late returns. Fine payments are dependant on the borrowing system, as they are linked to the library catalogue. The workflow is straightforward. Guests initiate a session by authenticating to the portal with credentials provided by the library. Once authenticated, they are presented with the choices (browse or borrow) and carry on their intended tasks until they decide to close the session.

Browsing online resources does not require special permissions. Consulting the library catalogue and borrowing items, however, involve an additional password. Paying fines does not require special permissions, but can only be done after accessing the borrowing section. The specification for the guest system is as follows:

$$\mathsf{G} \;=\; {}_g[\mathtt{mov}_l^{k_l}.(\mathtt{mov}_r.\mathsf{G}_1 \;+\; \mathtt{mov}_b^{k_b}.(\mathsf{G}_2 \;+\; \mathtt{mov}_f.\mathsf{G}_3))]^{i_g}$$

$$where\ \mathsf{G}_1, \mathsf{G}_2, \mathsf{G}_3 \neq 0$$

The library allows access to its online resources on the Resources web server through the Portal. The web server handles reservation requests for items from its Borrowing catalogue. Late return fines for borrowed items are processed through Fines. The specification for the library system's ambient is:

$$\mathsf{L} \;=\; {}^{k_l}_{l}[\mathsf{L}_1 \;\mid\; {}^{k_b}_{b}[\mathsf{B} \;\mid\; {}_{f}[\mathsf{F}]^{i_f}]^{i_b} \;\mid\; {}_{r}[\mathsf{R}]^{i_r}]^{i_l}$$
$$where\ \mathsf{L}_1, \mathsf{B}, \mathsf{F}, \mathsf{R} \neq 0$$

The whole system is composed of the library and guest systems L and G. It is therefore specified by the process S below:

$$\begin{aligned}\mathsf{S} \;&=\; \mathsf{L} \mid \mathsf{G} \\ &=\; {}^{k_l}_{l}[\mathsf{L}_1 \;\mid\; {}^{k_b}_{b}[\mathsf{B} \;\mid\; {}_{f}[\mathsf{F}]^{i_f}]^{i_b} \;\mid\; {}_{r}[\mathsf{R}]^{i_r}]^{i_l} \;\mid\; {}_{g}[\mathtt{mov}^{k_l}_{l}.(\mathtt{mov}_{r}.\mathsf{G}_1 \;+\; \mathtt{mov}^{k_b}_{b}.(\mathsf{G}_2 \;+\; \mathtt{mov}_{f}.\mathsf{G}_3))]^{i_g}\end{aligned}$$

To summarize, the signification of processes is as follows:

- G: represents the Guest process;
- G_1: symbolizes the process used by the Guest to access online resources;
- G_2: denotes the activities required by the Guest for browsing the catalogue and borrowing items;
- G_3: stands for the activities required by the Guest for paying fines for late returns;
- L: represents the Library process;
- L_1: corresponds to the Library portal authentication and choice of service;
- B: represents the Library's catalogue browsing and borrowing services;
- F: expresses to the process associated with fine payments;
- R: stands for the online resources web server.

The interfaces share different channels to accommodate communication between ambients: $i_g = \{gl\}, i_l = \{gl, lb, lr\}, i_b = \{lb, bf\}, i_f = \{bf\}$ and $i_r = \{lr\}$.

This enables the following:

- guest can communicate with the portal;
- the portal can communicate with the borrowing and resources systems;
- the borrowing and fines systems can communicate;
- no other communication is defined.

The library decides to finally upgrade their ageing catalogue system. The library managers decide to reduce guest access temporarily to online resources only. For a certain period, guests will no longer be allowed to consult the catalogue and borrow items. In order to do that, the Borrowing system's key k_b needs to be changed to a new value $k_b{}'$. The new policy for the library system is:

$$\Phi_l \;=\; {}^{k_l}_{l}[\not\Vdash \;\mid\; {}^{k_b{}'}_{b}[\not\Vdash]^{i_b}]^{i_l}, where\ k_b{}' \neq k_b$$

The formula Φ_l would be satisfied if the Library process meets several conditions. First, the process must be an ambient named l, protected by the key k_l and communicating across an interface i_l. Second, the ambient must contain a parallelism that involves a non-blocking process and an ambient b. Finally, ambient b must have an interface i_b and must be protected by a new key k_b'.

6.6.2 Security Policy Enforcement

The enforcement required to make the library system compliant with the new policy is given by:

$$X_l \;=\; \frac{\Phi_l}{\mathsf{L}}$$

The enforcement process for the library system is calculated as follows:

$$
\begin{aligned}
X_l &= \frac{\Phi_l}{\mathsf{L}} = \frac{{}^{k_l}_{l}[\not\Vert \mid {}^{k_b\prime}_{b}[\not\Vert]^{i_b}]^{i_l}}{{}^{k_l}_{l}[\mathsf{L}_1 \mid {}^{k_b}_{b}[\mathsf{B} \mid {}_{f}[\mathsf{F}]^{i_f}]^{i_b} \mid {}_{r}[\mathsf{R}]^{i_r}]^{i_l}} \\
&= \mathtt{mov}^{k_l}_{l}.(\frac{\not\Vert \mid {}^{k_b\prime}_{b}[\not\Vert]^{i_b}}{\mathsf{L}_1 \mid {}^{k_b}_{b}[\mathsf{B} \mid {}_{f}[\mathsf{F}]^{i_f}]^{i_b} \mid {}_{r}[\mathsf{R}]^{i_r}}) && (by\ 6.8.11) \\
&= \mathtt{mov}^{k_l}_{l}.(\frac{\not\Vert}{\mathsf{L}_1 \mid {}_{r}[\mathsf{R}]^{i_r}} \mid \frac{{}^{k_b\prime}_{b}[\not\Vert]^{i_b}}{{}^{k_b}_{b}[\mathsf{B} \mid {}_{f}[\mathsf{F}]^{i_f}]^{i_b}}) && (by\ 6.8.8) \\
&= \mathtt{mov}^{k_l}_{l}.(1 \mid \mathtt{mov}^{k_b}_{b}.\mathtt{prot}^{k_b\prime}_{b}.\frac{\not\Vert}{\mathsf{B} \mid {}_{f}[\mathsf{F}]^{i_f}}) && (by\ 6.8.1, 6.8.12) \\
&= \mathtt{mov}^{k_l}_{l}.\mathtt{mov}^{k_b}_{b}.\mathtt{prot}^{k_b\prime}_{b}.1 && (by\ 6.2.16, 6.8.1)
\end{aligned}
$$

We have computed the necessary enforcement corresponding to the new policies based on the quotient operator table defined in the previous section. It is now time to verify that the enforcements work as expected and the modified systems satisfies the new policy: $\mathsf{L} \setminus \mathsf{X_l} \models \Phi_l$. We proceed by calculating the system specification for the enforced Library system by using the reduction relation defined in Table 4.3:

$$
\begin{aligned}
\mathsf{L} \setminus \mathsf{X_l} &= {}^{k_l}_{l}[\mathsf{L}_1 \mid {}_{b}[\mathsf{B} \mid {}_{f}[\mathsf{F}]^{i_f}]^{i_b} \mid {}_{r}[\mathsf{R}]^{i_r}]^{i_l} \setminus \mathtt{mov}^{k_l}_{l}.\mathtt{mov}^{k_b}_{b}.\mathtt{prot}^{k_b\prime}_{b}.1 \\
&\xrightarrow{\mathtt{mov}^{k_l}_{l}} {}^{k_l}_{l}[\mathsf{L}_1 \mid {}_{b}[\mathsf{B} \mid {}_{f}[\mathsf{F}]^{i_f}]^{i_b} \mid {}_{r}[\mathsf{R}]^{i_r} \setminus \mathtt{mov}^{k_b}_{b}.\mathtt{prot}^{k_b\prime}_{b}.1]^{i_l} && (by\ 6.4.13) \\
&\xrightarrow{\mathtt{mov}^{k_b}_{b}} {}^{k_l}_{l}[\mathsf{L}_1 \mid {}_{b}[\mathsf{B} \mid {}_{f}[\mathsf{F}]^{i_f} \setminus \mathtt{prot}^{k_b\prime}_{b}.1]^{i_b} \mid {}_{r}[\mathsf{R}]^{i_r}]^{i_l} && (by\ 6.4.13) \\
&\xrightarrow{\mathtt{prot}^{k_b\prime}_{b}} {}^{k_l}_{l}[\mathsf{L}_1 \mid {}^{k_b\prime}_{b}[\mathsf{B} \mid {}_{f}[\mathsf{F}]^{i_f}]^{i_b} \mid {}_{r}[\mathsf{R}]^{i_r}]^{i_l} && (by\ 6.4.16)
\end{aligned}
$$

The policy satisfaction relation for Φ_l can then be written as:

$$
\begin{aligned}
\mathsf{L} \setminus \mathsf{X_l} \models \Phi_l &\Leftrightarrow {}^{k_l}_{l}[\mathsf{L}_1 \mid {}^{k_b{}'}_{b}[\mathsf{B} \mid {}_{f}[\mathsf{F}]^{i_f}]^{i_b} \mid {}_{r}[\mathsf{R}]^{i_r}]^{i_l} \models {}^{k_l}_{l}[\not\Vert \mid {}^{k_b{}'}_{b}[\not\Vert]^{i_b}]^{i_l} \\
&\Leftrightarrow \mathsf{L}_1 \mid {}^{k_b{}'}_{b}[\mathsf{B} \mid {}^{k_f}_{f}[\mathsf{F}]^{i_f}]^{i_b} \mid {}_{r}[\mathsf{R}]^{i_r} \models \not\Vert \mid {}^{k_b{}'}_{b}[\not\Vert]^{i_b} && (by\ 6.6.7)
\end{aligned}
$$

We can verify that indeed $\mathsf{L} \upharpoonright \mathsf{X_l} \models \Phi_l$, since $\mathsf{L_1} \mid {}_r[\mathsf{R}]^{i_r} \models \not\Box$ (by 6.6.1) and ${}^{k_b'}_{b}[\mathsf{B} \mid {}_f[\mathsf{F}]^{i_f}]^{i_b} \models {}^{k_b'}_{b}[\not\Box]^{i_b}$ (by 6.6.7 and 6.6.1). Therefore, the library system has been successfully enforced to satisfy the new policy.

6.7 Software Implementation

The main purpose of developing PEA (short for "Policy Enforcement Application") was to mechanize the method developed in this chapter and to demonstrate its applicability. Given a computer network system, no matter how complex, the application allows us to build the network topology, generate the network specification using the calculus, specify policies with the logic and calculate required enforcements based on the quotient operator formulas.

The application was developed in Java [46] using Eclipse [98], with Swing libraries [40] being used for the GUI development. The architecture of our application contains different roles, a GUI and three modules that implement the elements of our framework. One module translates topology information into an interprocess algebra-based specification. The second module is used for defining system security policies with logical formulas. Enforcement calculations are performed by the third module. Since we used Eclipse, it is easy to extend the application by adding new modules. The addition of libraries of other elements (services, other network hardware, etc.) is also straightforward. The internationalization feature of Java allows interface components (menu items, labels, etc.) to be defined in your language of choice. Currently, there are English and French versions of the application. The software and the complete code will be made available under GPL license.

The interface is intuitive and easy to use. Fig. 6.3 displays the network topology described in our case study, along with the associated network specification. The system's topology is built by simply dragging predefined elements (computers, routers, firewalls, etc.) into the main window of the GUI. Associated details such as ambient names and keys, or processes, can be added as properties of the elements via a context-aware box. Once the components have been inserted, a dedicated button can be used to connect them by pointing to the ends of the link.

Figure 6.3: PEA: network specification

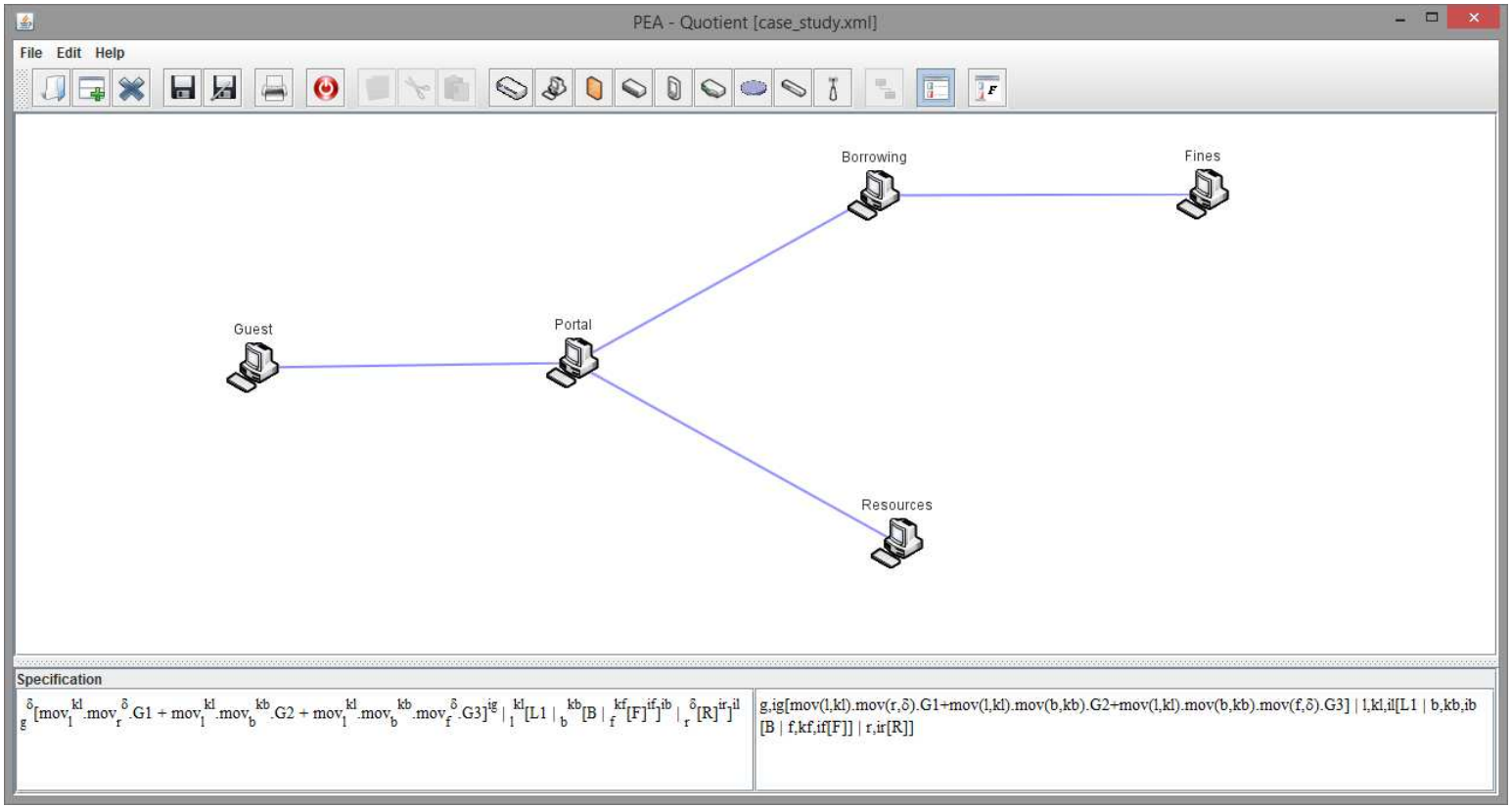


Figure 6.4: PEA: ambient modification

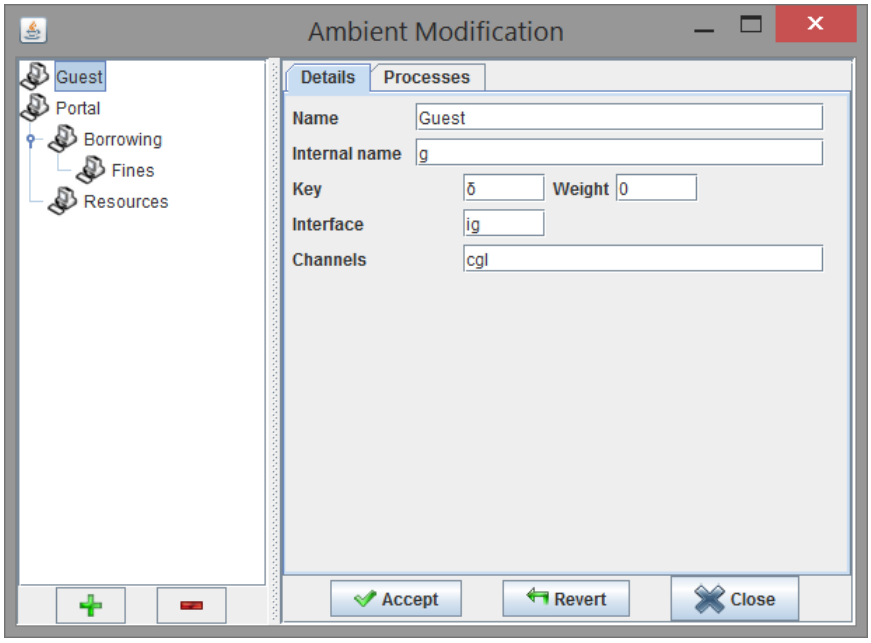


Figure 6.5: PEA: process modification

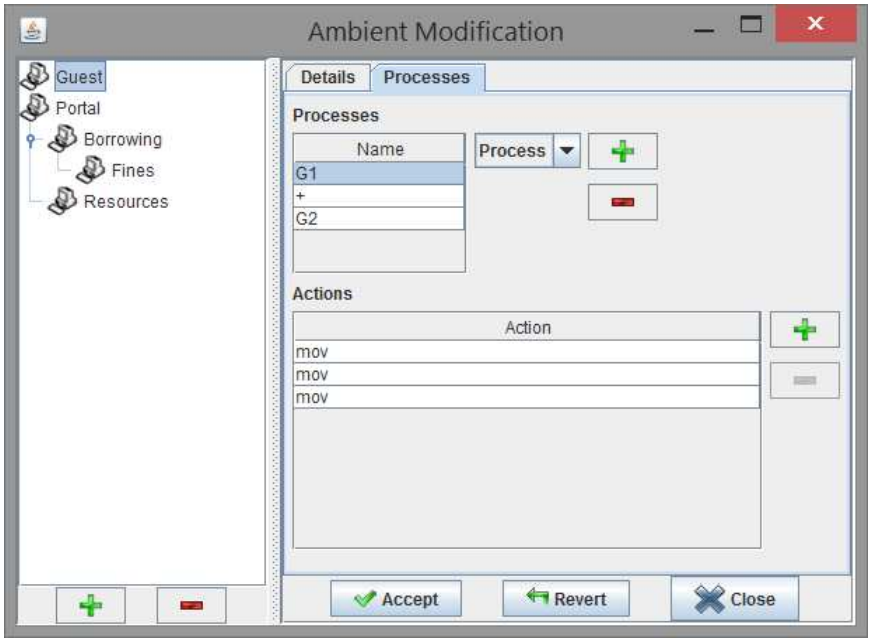
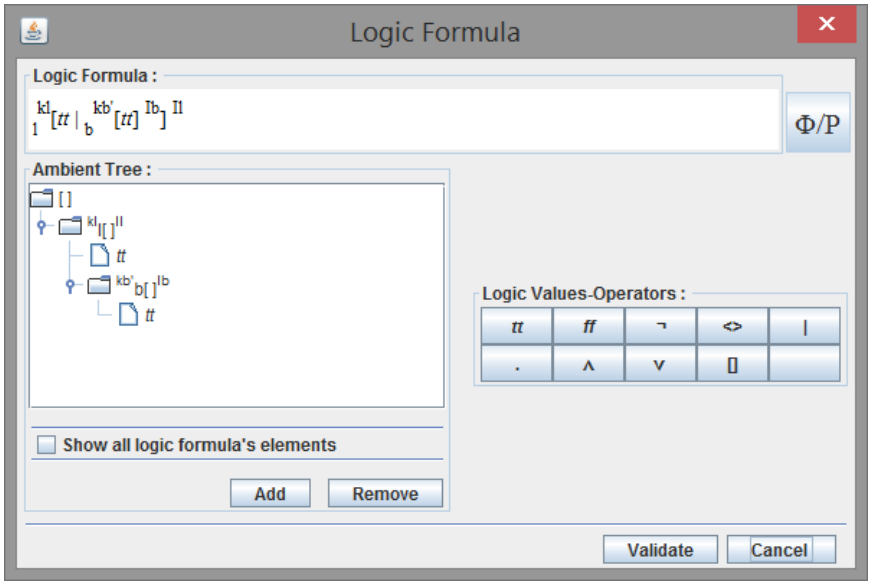


Figure 6.6: PEA: library security policy



The network specification for the system built is displayed at the bottom of the main window. Any change to the topology or process details is automatically applied to the specification. All applicable process reductions are also taken into account and are transparently performed in the background.

Systems built with the aid of the application can be modified, copied, exported to XML files and imported for re-utilization. Links between components can be added and removed through a simple right-click. Parent-child ambient relations can be changed via drag-and-drop. Ambient names, keys, interfaces, and process details are easily defined, as illustrated in Fig. 6.4 and Fig. 6.5.

PEA also allows specifying security policies by using a GUI-style method. The policy is displayed both in logic formulas format and as an XML style structure showing the components, as in Fig. 6.6.

The main window displays the topology and system specification. If the policy window, showing the logic formulas, is also open, one can calculate the necessary enforcement to be applied to the system. The quotient button permits the automated generation of an adequate enforcement, as demonstrated in Fig. 6.7.

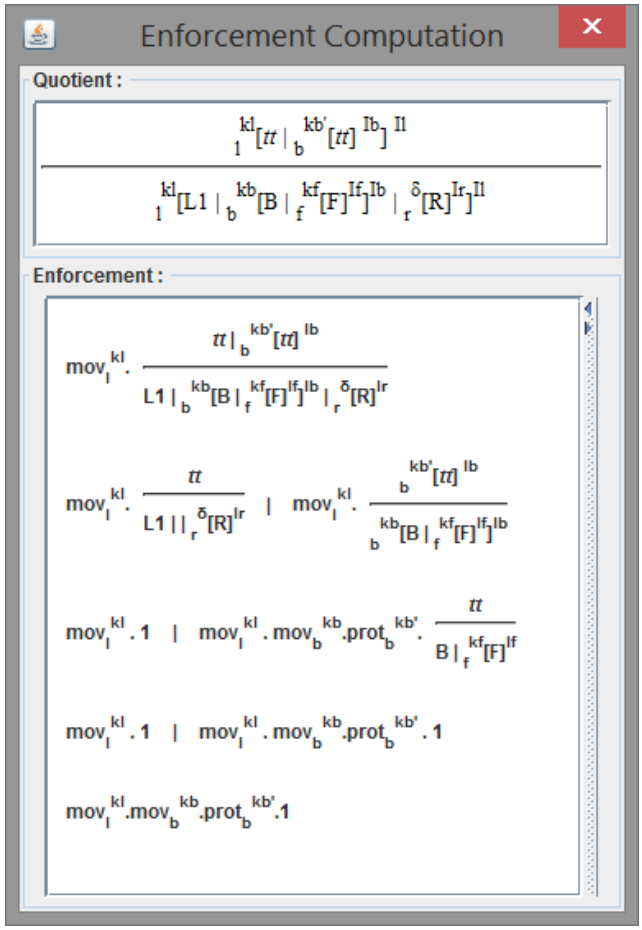
The application is the first step in our effort to obtain automatic enforcements for the implementation of desired security policies. It can be employed for various network simulations: system design, policy verification, disaster recovery scenarios, etc.

6.8 Conclusion

In this chapter, we have developed a new formal framework for computer system specification and security policy enforcement. This framework consists of three main components. The first is a formalism for specifying computer systems that captures in an effective and elegant way the system's behaviour and topology. We improve our *CS2* calculus with constructs for communication interfaces, protection capabilities, and an enforcement operator. The new calculus facilitates the specification of a computer system's current state and its evolution. The second component of the framework is a dedicated logic for defining security policies. The logic formulas allow expressing current and desired security policies. The semantics of the logic links policy satisfaction to compliant processes. Again, the system's evolution can be followed, in this case from the point of view of applicable constraints. The third component is the quotient operator that allows an automatic calculation of required enforcements. Given a process P and a security policy Φ, we calculate $\mathsf{X} = \frac{\Phi}{\mathsf{P}}$ as a first step. If $\mathsf{X} = 1$, then the process P satisfies the policy Φ. Otherwise, the enforcement can be applied to update the process and make it policy compliant. The formal foundation of all components ensures that the enforcement produces a secured system, free of incomplete specifications, arbitrary interpretations or faulty implementation of policies.

Note that our approach is different from previous works as it allows both the verification of policy compliance and the application of corrective actions. The automated aspect of the methodology further enhances its value. A

Figure 6.7: PEA: library enforcement process



software implementation demonstrates its effectiveness and proves its applicability to practical problems. The versatility of the framework is supported by the numerous potential uses. It could be a free and lightweight tool employed by educators to simulate basic or complex network topologies, or to build and test security policies. A detailed specification that includes all potentially vulnerable systems is a worthwhile effort. Intrusion prevention systems of firewalls could take advantage of it by blocking exploits of known

vulnerabilities through access key changes for the affected components. This should be done in conjunction with patch management systems, which could signal when access can be re-enabled. Cyber attacks could be modelled to determine whether critical systems can be reached, and under which conditions. They can be further analyzed to decide the best detection and prevention strategy. The approach can also be beneficial for business continuity planning while designing alternate configurations or disaster recovery scenarios. Such cases imply system specifications adapted to the potential crisis (natural or human-triggered disasters, hardware failure, wars, etc.). Our methodology could be used to validate whether the expectations (i.e. security policy) can be satisfied by the available computer systems (i.e. part of the new system specification).

Further extensions of the calculus syntax, in particular, would increase the range of applications. In its current state, the access key incorporates all conditions necessary to enter an ambient. In practice, they correspond to more than just system credentials or access control lists. A finer granularity would allow a more precise control over process movement. A new action could model direct communications between a sender and a receiver that are not neighbours (encrypted channels, remote connections, etc). Other actions that do not relate specifically to mobility would enrich specifications to better reflect the complex set of computer system behaviours.

Chapter 7

Conclusion and Future Work

Abstract

This chapter positions the research within the current security context, summarizes the results included in this thesis, and examines potential further extensions of the results obtained so far.

The "always on, anywhere" paradigm has inadvertently benefited the malicious intruders, from script kiddies to hacktivists and state-sponsored agents. Hacking as a Service is a commodity available at the end of a mouse click. More than ever before, being connected has become synonymous with being vulnerable. For the consumers of network services, whether they use servers, laptops, tablets, or smartphones, the whole concept of *secure computing* is overwhelming. They have two alternatives: becoming an expert in all aspects of security, or ignoring them blissfully. The first option is unrealistic, while the second one leaves everybody open to attacks.

How do we solve this problem? We do it by providing built-in security mechanisms out of the box, along with tools to enforce new safeguards as the system evolves. Yet, this may not be sufficient to ensure proper protection. Although the implementation of some mechanisms is straightforward, this generally does not apply to the complex and distributed systems of large organizations. Then, there is also the subjective human factor that intervenes in policy definition, implementation and evaluation.

Formal methods are well positioned to address such concerns, as demon-

strated throughout this thesis. The framework we built enables us to specify systems and security policies, assess policy compliance and automatically calculate necessary enforcements for non-compliant systems. The research goals were achieved through incremental advances, as shown by the gradual refinement of the calculi. The first calculus we propose, *FPC*, employs a large number of firewall-specific constructs. *SSC*, the second process algebra presented, includes multiple capabilities for specifying the intruder's behavior. While they contribute to an exact representation of the environment, these syntactic elements were not carried on in the general-purpose *CS2* and $CS2^{+}$.

As a result, the individual components of the final framework - the process algebra and the associated modal logic - are as lean as possible. This renders the whole solution concise, but accurate and elegant. Nonetheless, it manages to capture all the subtleties of system specification and policy assessment. It is worth reiterating that the formal foundation of our approach produces an outcome that is proven to be correct for both verification and enforcement. Furthermore, both forms of policy assessment are automated. The LoTREC implementation of the verification algorithm eliminates the need for a manual application of inference rules. PEA, the software implementation of the policy enforcement methodology, goes even further. Through its graphic interface, it facilitates the specification of the computer system and security policies, draws the associated topology, and calculates the enforcement needed to satisfy a given policy.

The afore-mentioned results can be the basis for new research. We are contemplating the following directions for extending the work presented in this thesis:

- Modelling vulnerabilities: whenever an exploit exists for an information system or one of its components, this can be expressed as a protection change where the new key is public (δ).
- Access control models: modelling roles and/or security levels with our calculus would make it possible to analyze their behaviour with respect to security policies. Configuration of roles and access levels could be easily implemented based on our algebra.
- Additional functionality in PEA: the provision of a library of prebuilt

templates for computer systems and subsystems would greatly improve its usability.

Modelling vulnerabilities. Information systems are secure as long as none of their components are vulnerable. The workflow is rather simple. Once a vulnerability has been discovered for a subsystem and an exploit was made available, a malicious adversary can attempt to attack it. A successful attack gives access to the subsystem concerned. Once compromised, privilege escalation can be the next step. The exploit may expose other subsystems and allow an intruder to elevate their privileges to a higher level. In order to fix the vulnerability, a patch needs to be installed.

The "Shellshock" vulnerability of the very popular GNU Bash shell, for instance, allowed remote attackers to run arbitrary code. Computers, servers, firewalls and security appliances using Bash were potential victims of cleverly crafted messages. In successful attacks, the intruders were able to execute any command and take full control over the target system.

Modelling would have helped understanding the overall impact (direct and through related subsystems). The emergence of an exploit can be expressed in terms of our *CS2* process algebra as the enforcement for the ambient representing the subsystem. The enforcement would impose a change of the access key from k (private) to δ (public). Patches for correcting vulnerabilities can be seen as subsequent enforcements of a fresh private key.

Potential research avenues: determine which policy implementations are affected by a vulnerability, evaluate mitigation solutions (e.g. temporarily disabling the subsystem), run simulations of a potential compromise for impact assessment.

Access control models. Our representation of ambient protection stands in for a whole class of access control concepts such as authentication credentials, privileges, access control lists. This is by design, as the level of abstraction allows a security practitioner to focus on the mobility aspects. There is an opportunity, however, to fragment the protection keys into more granular components.

The partial order on keys may be useful for expressing different levels of privileges, possibly associated with different roles or user profiles. A role would set consistent permissions across processes representing a category of users. Processes P and Q denoting two regular users, for instance, would employ movements with the same key k_r for accessing a certain ambient.

Identification information (e.g. account name, user ID, IP address etc.) can be used for compiling access control lists (ACLs) associated with ambient protection keys. Movements executed by all processes corresponding to ACL items would employ a key that is equal to or superior to the protection key of the ambient. Alternately, roles could be involved in defining an acceptable partial order for an access key. Let's consider the case of a system with three roles: regular user, power user, and administrator. Let the keys k_r, k_p and k_a be the keys associated with the roles, with $k_r < k_p < k_a$. Configuring the level of privilege required to access an ambient would be as simple as setting the value of the protection key to one of the three values: k_r for everyone, k_p to exclude regular users, and k_a to allow administrators only. This could significantly simplify the system and policy specification, as only three keys would be used.

The separation of duties can also be addressed by using the minimum level of privileges required to access a resource. This applies to situations when a process is executed by a user with multiple roles. Someone with both regular user and administrator roles could switch to either one for running certain tasks. Their processes can therefore use either k_r or k_a. During the system specification phase, the processes could be defined considering the ambient protection key (i.e. the least privilege) rather than the most powerful role.

Additional functionality in PEA. The graphical interface of the application works well for drawing network topology maps thanks to its icons library. Routers, servers, computers and links can be selected and added with one click. This does not translate, however, into a rich system specification. All network components are simply represented as bare ambients. This may be discouraging for a security practitioner who tries to model a large system. A library of pre-built specialized components would be beneficial. For instance, there could be templates for a typical workstation with an Internet browser, anti-virus, and local files, or a web server with FTP and remote management services. The templates could be extended to include small LANs, such as a

development laboratory or a virtual classroom.

The subject of automatic security policy enforcement is vast and only partially explored. This thesis is by no means an exhaustive coverage of the issue, but contributes by mapping another area of the domain. We hope that it motivates and inspires further research.

Bibliography

[1] Pietro Abate and Rajeev Goré. System Description: The Tableaux Work Bench. Available at: http://csl.anu.edu.au/ abate/twb. In *TABLEAUX, LNCS*, pages 164–175. Springer, 2003.

[2] K. Adi, M. Debbabi, and M. Mejri. A new logic for electronic commerce protocols. In *International journal of Theoretical Computer Science (TCS) 291*, 2003.

[3] E. S Al-Shaer and H. H Hamed. Discovery of policy anomalies in distributed firewalls. In *INFOCOM '04: Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, pages 2605–2616. IEEE, 2004.

[4] Arnon Avron. The method of hypersequents in the proof theory of propositional non-classical logics. *In Logic: From Foundations To Applications, European Logic Colloquium*, pages 1–32, 1994.

[5] J. C. M. Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335(2-3):131–146, 2005.

[6] David Basin, Vincent Jugé, Felix Klaedtke, and Eugen Zălinescu. Enforceable security policies revisited. *ACM Trans. Inf. Syst. Secur.*, 16(1):3:1–3:26, June 2013.

[7] David Basin, Felix Klaedtke, and Samuel Müller. Monitoring security policies with metric first-order temporal logic. In *Proceedings of the 15th ACM Symposium on Access Control Models and Technologies*, SACMAT '10, pages 23–34, New York, NY, USA, 2010. ACM.

[8] L. Bauer, J. Ligatti, and D. Walker. More enforceable security policies. In *In Foundations of Computer Security, Copenhagen, Denmark*, 2002.

[9] Lujo Bauer. Composing security policies with polymer. In *In Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation*, pages 305–314. ACM, 2005.

[10] Lujo Bauer, Jay Ligatti, and David Walker. A language and system for composing security policies. Technical report, Princeton University, 2004.

[11] Bernhard Beckert and Rajeev Goré. Free variable tableaux for propositional modal logics. *TABLEAUX-97 LNCS 1227*, pages 91–106, 1997.

[12] Steven M. Bellovin. Distributed firewalls. *;login:*, 24(Security):39–47, November 1999.

[13] Jan A. Bergstra and Jan Willem Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37:77–121, 1985.

[14] Tommaso Bolognesi and Ed Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–59, 1987.

[15] C. Braghin, A. Cortesi, R. Focardi, and S. van Bakel. Boundary inference for enforcing security policies in mobile ambients. In *Proceedings of the 2nd IFIP International Conference on Theoretical Computer Science (TCS)*, pages 383–395. Kluwer Academic Publisher, 2002.

[16] Chiara Braghin, Agostino Cortesi, and Riccardo Focardi. Control flow analysis of mobile ambients with security boundaries. In *FMOODS '02: Proceedings of the Fifth International Conference on Formal Methods for Open Object-Based Distributed Systems V*, pages 197–212. Kluwer, B.V., 2002.

[17] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.

[18] Michele Bugliesi and Giuseppe Castagna. Secure safe ambients. In *POPL '01: Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 222–235, New York, NY, USA, 2001. ACM.

[19] J. Burns, A. Cheng, P. Gurung, S. Rajagopalan, P. Rao, D. Rosenbluth, A.V. Surendran, and D. Martin-Jr. Automatic management of network security policy. *DARPA Information Survivability Conference and Exposition (DISCEX II'01)*, 02:1012, 2001.

[20] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, SOSP '89, pages 1–13, New York, NY, USA, 1989. ACM.

[21] L. Cardelli and A. D. Gordon. Mobile ambients. In *Foundations of Software Science and Computation Structures: First International Conference, FOSSACS '98*. Springer-Verlag, Berlin Germany, 1998.

[22] L. Cardelli and A. D. Gordon. Types for mobile ambients. In *Symposium on Principles of Programming Languages*, pages 79–92, 1999.

[23] Luca Cardelli and Luca Cardelli. Mobile ambient synchronization. Technical report, Digital Systems Research, 1997.

[24] Luca Cardelli and Andrew D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '00, pages 365–377, New York, NY, USA, 2000. ACM.

[25] Luca Cardelli and Andrew D. Gordon. Ambient logic. *Mathematical Structures in Computer Science*, 2006.

[26] E. Cerny. Controllability and fault observability in modular combinational circuits. *IEEE Transactions on Computers*, 27(10):896–903, October 1978.

[27] Luis Fariñas del Cerro, David Fauthoux, Olivier Gasquet, Andreas Herzig, Dominique Longin, and Fabio Massacci. Lotrec: The generic tableau prover for modal and description logics. In *Proceedings of the First International Joint Conference on Automated Reasoning*, IJCAR '01, pages 453–458, London, UK, UK, 2001. Springer-Verlag.

[28] Rance Cleaveland. Tableau-based model checking in the propositional mu-calculus. *Acta Informatica*, 27(8):725–747, September 1990.

[29] A. Cortesi and R. Focardi. Information flow security in mobile ambients. *Electronic Notes in Theoretical Computer Science*, 54, 2001.

[30] Ince D.C. A graph theoretic solution to the interface equation. In *Applications of Combinatorial Mathematics*, volume 0, pages 185–197, 1997.

[31] Pierpaolo Degano, Francesca Levi, and Chiara Bodei. Safe ambients: Control flow analysis and security. In *ASIAN '00: Proceedings of the 6th Asian Computing Science Conference on Advances in Computing Science*, pages 199–214, London, UK, 2000. Springer-Verlag.

[32] Antoni Diller. *Z: An Introduction to Formal Methods*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[33] G. Ferrari, E. Moggi, and R. Pugliese. Guardians for ambient-based monitoring. *F-WAN: Foundations of Wide Area Network Computing*, 66, 2002.

[34] Colin Fidge. A comparative introduction to CSP, CCS and LOTOS. Technical report, University of Queensland, 1994.

[35] F.B. Fitch. Tree proofs in modal logic. *Journal of Symbolic Logic*, 31:152, 1966.

[36] Melvin Fitting. Tableau methods of proof for modal logics. *Notre Dame Journal of Formal Logic*, 13(2):237–247, 04 1972.

[37] Melvin Fitting. *First-order Logic and Automated Theorem Proving*. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[38] Melvin Fitting. Prefixed tableaus and nested sequents. *Annals of Pure and Applied Logic*, 163(3):291 – 313, 2012.

[39] Cédric Fournet, Jean-Jacques Lévy, and Alan Schmitt. An asynchronous, distributed implementation of mobile ambients. In *TCS '00: Proceedings of the International Conference IFIP on Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics*, pages 348–364, London, UK, 2000. Springer-Verlag.

[40] Amy Fowler. A swing architecture overview. *SUN/Oracle*, 2004.

[41] Olivier Gasquet, Andreas Herzig, Dominique Longin, and Mohamad Sahade. Lotrec: Logical tableaux research engineering companion. In *Proceedings of the 14th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, TABLEAUX'05, pages 318–322, Berlin, Heidelberg, 2005. Springer-Verlag.

[42] J. A. Goguen and J. Meseguer. Security policies and security models. *1982 IEEE Symposium on Security and Privacy*, 00:11, 1982.

[43] A. D. Gordon and L. Cardelli. Equational properties of mobile ambients. In *Foundations of Software Science and Computation Structure*, pages 212–226, 1999.

[44] Daniele Gorla and Rosario Pugliese. Resource access and mobility control with dynamic privileges acquisition. In *In Proc. of ICALPŠ03, volume 2719 of LNCS*, pages 119–132. Springer-Verlag, 2003.

[45] Daniele Gorla and Rosario Pugliese. Enforcing security policies via types. *Security in Pervasive Computing: First International Conference, Boppard, Germany, March 12-14, 2003. Revised Papers*, pages 86–100, 2004.

[46] James Gosling, Bill Joy, and Guy L. Steele. *The Java Language Specification*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1996.

[47] J. D. Guttman. Filtering postures: Local enforcement for global policies. In *IEEE Symposium on Security and Privacy*, pages 120–129, 1997.

[48] Kevin Hamlen. *Security Policy Enforcement by Automated Program-rewriting*. PhD thesis, Cornell University, Ithaca, NY, USA, 2006. AAI3227141.

[49] Kevin W. Hamlen, Greg Morrisett, and Fred B. Schneider. Computability classes for enforcement mechanisms. *ACM Transactions on Programming Languages and Systems*, 28(1):175–205, 2006.

[50] M. Hennessy, M. Merro, and J. Rathke. Towards a behavioural theory of access and mobility control in distributed systems. In *Theoretical Computer Science*, pages 282–299. Springer-Verlag, 2003.

[51] Matthew Hennessy and Robin Milner. On observing nondeterminism and concurrency. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming*, pages 299–309, London, UK, 1980. Springer-Verlag.

[52] Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *Journal of ACM*, 32(1):137–161, 1985.

[53] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173:2002, 1998.

[54] D. Hirschkoff, E. Lozes, and D. Sangiorgi. On the expressiveness of the ambient logic. In *Logical Methods in Computer Science*, 2006.

[55] Daniel Hirschkoff. An extensional spatial logic for mobile processes. In *CONCUR*, pages 325–339, 2004.

[56] C. A. R. Hoare and C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21:666–677, 1985.

[57] Pengelly A. Ince D.C. Quotient machines, the interface equation and protocol conversion. In *Computer Journal*, volume 43, pages 24–39, 2000.

[58] Sotiris Ioannidis, Angelos D. Keromytis, Steven M. Bellovin, and Jonathan M. Smith. Implementing a distributed firewall. In *ACM Conference on Computer and Communications Security*, pages 190–199, 2000.

[59] Ken Cutler John Wack and Jamie Pole. Guidelines on firewalls and firewall policy. Technical report, NIST(National Institute of Standards and Technology), January 2002. Special Publication 800-41.

[60] F. Khendek, G. von Bochmann, and C. Kant. New results on deriving protocol specifications from service specifications. *SIGCOMM Comput. Commun. Rev.*, 19(4):136–145, 1989.

[61] Raphaël Khoury and Nadia Tawbi. Equivalence-preserving corrective enforcement of security properties. *Int. J. Inf. Comput. Secur.*, 7(2/3/4):113–139, November 2015.

[62] R. Khédri. *Concurrence, bisimulations et équation d'interface : une approche relationnelle*. PhD thesis, Université Laval, 1998.

[63] Dexter Kozen. Results on the propositional μ-calculus. In *Proceedings of the 9th Colloquium on Automata, Languages and Programming*, pages 348–359, London, UK, 1982. Springer-Verlag.

[64] Saul Kripke. Semantic considerations on modal logic. *Acta Philosophica Fennica*, 16:83–94, 1963.

[65] Alexandre Lacasse, Mohamed Mejri, and Béchir Ktari. Formal implementation of network security policies. In *The Second Annual Conference on Privacy, Security and Trust (PST04). New Brunswick, Canada*, 2004.

[66] Mahjoub Langar and Mohamed Mejri. Formal and efficient enforcement of security policies. In *Foundations of Computer Science*, pages 143–149, 2005.

[67] Momamed Langar, Mohamed Mejri, and Kamel Adi. Formal enforcement of security policies on concurrent systems. *Journal of Symbolic Computation*, 3:997–1016, 2011.

[68] Björn Lellmann. Linear nested sequents, 2-sequents and hypersequents. In *Automated Reasoning with Analytic Tableaux and Related Methods - 24th International Conference, TABLEAUX 2015, Wrocław, Poland, September 21-24, 2015. Proceedings*, pages 135–150, 2015.

[69] Francesca Levi and Davide Sangiorgi. Controlling interference in ambients. In *Symposium on Principles of Programming Languages*, pages 352–364, 2000.

[70] Francesca Levi and Davide Sangiorgi. Mobile safe ambients. *ACM Transactions on Programming Languages and Systems*, 25(1):1–69, 2003.

[71] L. Logrippo, M. Faci, and M. Haj-Hussein. An introduction to lotos: learning by examples. *Computer Networks and ISDN Systems*, 23(5):325–342, 1992.

[72] F. Martins and V. Vasconcelos. Controlling security policies in a distributed environment, 2004.

[73] T. Mechri, Mahjoub Langar, Mohamed Mejri, Hamido Fujita, and Yutaka Funyu. Automatic enforcement of security in computer networks. In *SoMeT*, pages 200–222, 2007.

[74] Philip Merlin and Gregor V. Bochmann. On the construction of submodule specifications and communication protocols. *ACM Transactions on Programming Languages and Systems*, 5(1):1–25, 1983.

[75] W. Miksad and William A. Wulf. Specification and verification of security policies. Technical report, University of Virginia Charlottesville, 1996.

[76] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982.

[77] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts. *I and II. Information and Computation*, 100:1–77, 1992.

[78] Robin Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, June 1999.

[79] M.T.Norris. The role of formal methods in system design. *British Telecom Technical Journal*, 1985.

[80] George C. Necula. Proof-carrying code. pages 106–119. ACM Press, 1997.

[81] Rocco De Nicola and Michele Loreti. A modal logic for mobile agents. *ACM Trans. Comput. Logic*, 5(1):79–128, 2004.

[82] F. Nielson, H. R. Nielson, R. R. Hansen, and J. G. Jensen. Validating firewalls in mobile ambients. In *International Conference on Concurrency Theory*, pages 463–477, 1999.

[83] Arie Orlovsky and Danny Raz. Decentralized enforcement of security policies for distributed computational systems. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 241–248, New York, NY, USA, 2007. ACM.

[84] Hakima Ould-Slimane, Mohamed Mejri, and Kamel Adi. Using edit automata for rewriting-based security enforcement. In *DBSec*, pages 175–190, 2009.

[85] J. Parrow. Submodule construction as equation solving in ccs. *Theoretical Computer Science*, 29(1):175–202, 1989.

[86] J Parrow. An introduction to the pi-calculus. In *Dans Handbook of process algebra, Bergstra, Ponse et Smolka éditeurs, Elsevier*, pages 479–543, 2001.

[87] Lawrence C. Paulson. Introduction to isabelle, 1998.

[88] Lawrence C. Paulson. The isabelle reference manual, 2008.

[89] Robert L. Probert and Kassem Saleh. Synthesis of communication protocols: Survey and assessment. *IEEE Trans. Comput.*, 40(4):468–476, 1991.

[90] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21:2003, 2003.

[91] Davide Sangiorgi. Extensionality and intensionality of the ambient logics. In *Symposium on Principles of Programming Languages*, pages 4–13, 2001.

[92] Davide Sangiorgi and David Walker. *Pi-Calculus: A Theory of Mobile Processes*. Cambridge University Press, New York, NY, USA, 2001.

[93] Alan Schmitt and Inria Rocquencourt. An implementation of ambients in jocaml. position paper for the 5th mobile object systems workshop, 1999.

[94] Fred B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(1):30–50, 2000.

[95] M. W. Shields. Extending the interface equation. In *Technical Report SE/079/3, Electronic Engineering Laboratory*. University of Kent at Canterbury, August 1986.

[96] M. W. Shields. Solving the interface equation. In *Technical Report SE/079/2, Electronic Engineering Laboratory*. University of Kent at Canterbury, July 1986.

[97] M. W. Shields. Implicit system specification and the interface equation. *Comput. J.*, 32(5):399–412, 1989.

[98] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd edition, 2009.

[99] Colin Stirling. Modal and temporal logics for processes. In *Banff Higher Order Workshop*, pages 149–237, 1995.

[100] Guangye Sui and Mohamed Mejri. Faser formal and automatic security enforcement by rewriting by bpa algebra with test. *International Journal of Grid and Utility Computing*, 4(2/3):204–211, September 2013.

[101] Guangye Sui and Mohamed Mejri. Security enforcement by rewriting: An algebraic approach. In *Foundations and Practice of Security*, pages 311–321. Springer International Publishing, 2015.

[102] D. Teller, P. Zimmer, and D. Hirschkoff. Using ambients to control resources. *International Journal of Information Security*, 2:126–144. Springer, 2004.

[103] Úlfar Erlingsson and Fred B. Schneider. Sasi enforcement of security policies: a retrospective. In *NSPW '99: Proceedings of the 1999 workshop on New security paradigms*, pages 87–95, New York, NY, USA, 2000. ACM.

[104] D. Walker and D. Sangiori. The pi-calculus : A theory of mobile processes. In *Cambridge University Press*, 2003.

[105] P Zafiropulo, C H West, H Rudin, and D D Cowan. Brand,d.: Towards analyzing and synthesizing protocols. *IEEE Transactions on Communications*, 4(28):651–661, 1980.