UNIVERSITÀ DEGLI STUDI DI NAPOLI "FEDERICO II"
Dipartimento di Matematica e Applicazioni R.Caccioppoli

Dottorato in Matematica Applicata e Informatica - XIV Ciclo

Models and Security Verification of Distributed Systems

Maria Grazia Buscemi

Abstract

The focus of the thesis is on formal paradigms suited for modeling mobility and security aspects in distributed systems. We first investigate calculi for mobility, i.e. models for systems such that the network topology may dynamically change and nodes in the network can be allocated and deallocated as computations proceed.

We study the expressiveness of the join calculus by comparison with Petri nets, using tools from type theory. Specifically, we consider a hierarchy of generalized nets and prove a full correspondence to join calculus.

We also introduce a novel approach for lifting process calculi with structural axioms to bialgebras. Bialgebras enjoy the property that the unique morphism to the final bialgebra induces a bisimilarity relation which is a congruence with respect to the operations. We apply this general result to the π-calculus in order to provide a compositional coalgebraic model for the calculus.

The contribution of the thesis toward security includes a framework for the analysis of security protocols depending on a variety of cryptographic primitives (e.g. public-key and shared-key encryption, digital signature and hashing). Properties of security protocols, such as authentication and secrecy can be verified within this framework. Our method relies on symbolic techniques that provide more accuracy on the protocol model and increased efficiency, with respect to other approaches.

Finally, we present STA, a tool for the automatic analysis of security protocols, based on our symbolic method. We show results of experimenting with STA and contrast the tool with techniques based on finite-state model checking.

alla mia grande famiglia

Aknowledgments

First of all, I wish to thank my supervisor, Antonino Maugeri for his support during my PhD course.

I heartily thank Michele Boreale and Ugo Montanari who followed my research activity during these last two years, and Vladimiro Sassone who introduced me to this research area.

I also thank the Department of Mathematics and Computer Science of the University of Catania and the Department of Systems and Computer Science at the University of Florence, and my colleagues and roommates for the stimulating working environment.

Contents

Chapter 1

Introduction

A crucial step in the evolution of computer technology is due to the introduction of distributed systems and their interconnection via networks. This development is characterized by the increasing importance of data communication or data *mobility*, which may be described as the process of moving information electronically from one point to another.

The diffusion of distributed systems and the use of networks and communications facilities for carrying data among computers have affected computer security. It has given rise to the new notion of *network security* which includes a collection of security tasks such as to protect data and resources from disclosure, to guarantee the authenticity of data and messages, and to protect systems from network-based attacks.

The main models intended to formalize distributed systems began by considering only static connectivity. This is the case for Petri nets [62, 66], for Hoare's Communicating Sequential Processes (CSP) [38], for Milner's Calculus of Communicating Systems (CCS) [51]. These formalisms are best suited for describing the behavior of local area networks connecting autonomous computers for the purpose of resource sharing. Such systems typically consist of a collection of relatively uniform and static environments containing nodes connected by channels. They are characterized by their predictability with respect to bandwidth fluctuations and process failures, and by the availability of a global address space, a global file system, and a centralized administration ensuring reliability, stability and high level protection.

These assumptions do not hold in the context of computational structures such as wide area networks and, in particular, the World-Wide-Web. Access to resources is unreliable and opaque in such networks; there is no support for a global file system, no integrity of addressing and no centralized administration. In contrast to local area networks, these systems can be best described as consisting of a heterogeneous, dynamic collection of computers provided with changing links.

In this scenario, mobility not only denotes exchange of data over communication channels, but also has to do with mobile computation – mobile code that migrates from site to site – and, hence, it cannot abstract away from issues of locality. Moreover, within such a context, security concerns require that access barriers cannot be made transparent and must be treated as a fundamental feature of distributed systems.

Any computational model for this kind of systems must be equipped with primitives that are suitable for structuring the computational space around notions of mobility and security. *Nominal calculi* [36] are powerful models in this perspective. They are entirely based upon the notion of communication, and that computation and communication are equivalent terms. The main novelty of nominal calculi, when compared to earlier paradigms, is that they are able to model mobile systems, i.e., systems that have a dynamic changing topology and such that nodes and communications links can be created. This is achieved by the possibility of generating, communicating 'new' names.

Akin to earlier calculi, such as CCS and CSP, nominal calculi are usually defined by a set of basic actions, that are the elementary operations a system can perform, and by a small number of primitive operators that describe how complex systems are built composing simple ones: classical operators are, for instance, those for parallel composition, restriction, sequenzialization, non-deterministic choice. However, in nominal calculi the combined use of restriction and name-passing provides a much more powerful mechanism, as we will discuss in detail.

Labeled transition systems define the operational semantics of these calculi: a transition represents a possible evolution step of a system, and its label describes the action that is performed at that step. Following the structural operational semantics (SOS) paradigm introduced by Plotkin [65], the labeled transition system is often defined by structural rules. Behavioral equivalences are then defined on top of the labeled transition system: most famous is bisimulation equivalence.

As pointed out by Gordon, the main feature of nominal calculi – the dynamic generation of fresh, unguessable names – is crucial for modeling both security and mobility. The intuition behind names and naming is that in order to communicate with something, one must be able to refer to it, and the basic way of referencing is by naming it. Communicating something is naming something, e.g., a resource, an object, a piece of code, or a communication channel.

More in detail, in nominal calculi, the scope of a restriction – the program text in which a name may be used – may change during computation. When a process sends a restricted name as a message to a process outside the scope of the restriction, the scope is said to *extrude*, that is, it enlarges to embrace the process receiving the name of the channel. Processes are mobile in the sense that their communication possibilities may change over time; they may learn the names of new channels via scope extrusion. Thus, a name is a transferable capability for communication.

From a security viewpoint, the relevance of cryptography for network security makes the generation of fresh random numbers – composed of enough bits to be effectively unguessable – an essential primitive. Such random numbers are pure names; all that matters is their identity. They are used in cryptographic protocols as encryption keys or nonces (stamps used to prove freshness of a message). And, cryptographic channels, i.e., channels whose content is encrypted, are identified by the key used for encryption.

As we will show in the second part of the thesis, CSP, which is not a nominal calculus, has been used successfully to find bugs in cryptographic protocols [44]. However, a formalism representing a model of a programming language for mobile code is required in order to model the resources allocated dynamically by any program.

The focus of the thesis is on mobility and security aspects of models for concurrent systems. Therefore, nominal calculi will have a crucial role throughout the thesis.

In the following three sections we outline three main nominal calculi – π-calculus, join-calculus, and spi-calculus – in order to highlight the key role of names. We conclude the chapter with an overview of the origin of the thesis and a summary of its structure.

The π-Calculus

The π-calculus [53] is the touchstone for several applications concerning high-order mobility, security and object orientation. A formal treatment of the calculus is reported in Chapter 2.

Programs in the π-calculus consist of systems of independent, parallel processes that synchronize via message-passing handshakes on named channels. The channels that the programs know about determine the communication possibilities of the process. Channels are represented by names and names are the only data that may be communicated on channels. Channels may be restricted, so that only certain processes may communicate on them. In this respect the π-calculus is similar to earlier calculi such as CCS and CSP.

A major innovation of the π-calculus is its collection of equational laws along with transition rules for restricted names. The application of both these laws and rules enables the scope of a name to expand and contract as it is passed from one process to another during the course of a computation. In this sense, the scope of a restricted name is mobile. This mechanism allows to model mobility regarded as the passage of the name of a communications channel from one process to another.

The π-calculus can also be used for formalizing cryptographic protocols. Usually, in cryptographic protocols a cryptographic channel, i.e. a channel whose content is encrypted, is identified by the keys used for encryption. And the set of principals allowed to send and receive data on that channel may expand during a protocol run, for example, as a result of channel establishment. It is easy to model this property of channels in the π-calculus: keys correspond to names, key generation to restriction, communication on a cryptographic channel to communication on its key, and the expansion of that set of principals to scope extrusion.

Unfortunately, this encoding is too secure in the sense that it fails to model certain attacks on cryptographic communications. For instance, there is no way to model a process that eavesdrops on a channel or a process that transmits replayed or spoof messages. Moreover, Abadi observes that communication in the π-calculus enjoys *forward secrecy*: the secrecy of completed communications is not compromised by the subsequent disclosure of a previously secret channel.

The Join Calculus

We briefly outline the main ingredients of the distributed join calculus and a variant equipped with cryptographic primitives. In Section 2.3 we will formally define a monadic version of the core calculus.

The join calculus [31, 33] is a calculus of mobile processes with asynchronous name-passing communication. In the spirit of the π-calculus, it treats channel names and location names as first class values with lexical scopes. A location controls its own moves, and can only move towards a location whose name it has received. This provides a sound basis for static analysis and secure mobility.

The join calculus simplifies the π-calculus by enforcing the hypothesis of *unique receptors*. This means that there is at most one process receiving messages on a name. Unique receptors are a distinctive feature of the join calculus that makes it suitable for distributed implementations, as channels may be allocated at their receptor process.

In [2], Abadi and Fournet adopt the join calculus to address the issue of abstracting away from the low-level cryptographic manipulation in distribution applications. They propose a translation of the join-calculus programs into a low-level version of the calculus including cryptographic primitives. Communication on secure channels is implemented by encrypted communication on public channels. The result is that one can reason about programs in the high-level calculus independently of their implementation.

The Spi Calculus

We now give an overview of the spi-calculus, with emphasis on the key role of names to model security properties. In the second part of the thesis we will address more in detail formal methods for the analysis of cryptographic protocols and we will describe our automatic method based on a variant of the spi-calculus.

The spi-calculus is an extension of the π-calculus with cryptographic primitives, suited for specification and verification of security properties of cryptographic protocols.

The main idea behind this calculus is to use restriction and scope extrusion from the π-calculus to model possession and communication of secrets, such as cryptographic keys. In practice, communication on a cryptographic channel is modeled as communication on a public channel of a term that represents a ciphertext. The name of the communications channel is public, while the name representing the secret encryption key is restricted. An attacker intent on disrupting a protocol is modeled as an arbitrary process that runs alongside the protocol, and which can send or receive on the public channels used by the protocol. The encoding is realistic, since message replay is allowed and forward secrecy fails. And, indeed, if a ciphertext is transmitted on a public channel and the key is published, then the attacker can obtain the plaintext.

Contribution of the thesis

The focus of the thesis is on formal paradigms suited for modeling mobility and security aspects in distributed systems. Accordingly, the contributions of the thesis follow two directions.

On Mobility Nominal calculi are a very powerful formalism for describing both mobility and security, by means of mechanisms such generation of fresh and unguessable names, as well as restriction and extrusion. The operational semantics of process calculi is given in terms of labeled transition systems.

Labeled transition systems can be regarded as coalgebras for a functor in the category **Set**. A coalgebraic framework presents several advantages: morphisms between coalgebras (cohomomorphisms) enjoy the property of "reflecting behaviors" and thus they allow, for example, to characterize bisimulation equivalences as kernels of morphisms and bisimilarity as the bisimulation associated to the morphism to the final coalgebra.

However, in the above representation of transition systems, the states are seen as just forming a set, i.e., the algebraic structure modeling the construction of programs and the composition of states is disregarded.

The missing structure may be recovered by integrating coalgebras with algebras, as it is done by Turi and Plotkin [74]. They define *bialgebras* and prove that for them bisimilarity is a congruence. Roughly, bialgebras are structures that can be regarded both as algebras of coalgebras and as coalgebras of algebras. Morphisms between bialgebras are both algebra homomorphisms and coalgebra cohomomorphisms and, thus, the unique morphism to the final bialgebra, which exists under reasonable assumptions, induces a (coarsest) bisimulation congruence on any coalgebra.

In this first part of the thesis we propose a novel approach for lifting the transition systems of nominal calculi with structural axioms to bialgebras.

First, we consider a category $\mathbf{Alg}(\Sigma)$ of algebras where specification Σ does not contain axioms. Thus the powerset functor can be lifted to a functor on $\mathbf{Alg}(\Sigma)$ using the De Simone specification. Then, we construct a complete axiomatization (with auxiliary operators) of the algebra of the calculus and we prove that each axiom bisimulates. This is enough to ensure that the lifting can take place and thus, in particular, that bisimilarity is a congruence.

The compositional structure of our bialgebra also gives advantages in finite state verification methods [27]. Usually, such techniques glue components together and, then, apply model checking or other verification methods. On the other hand, in many cases, state explosion severely limits the applicability. Rather, a compositional approach gives a chance to minimize components before combining them, thus preventing state explosion, at some extent, and yielding a smaller state space.

The restriction operator of specification Σ has no name argument. Also extrusion and fresh input actions have no bound names. The reason is that the extruded, fresh or restricted name is assumed to be always the first one, i.e., x_0. To model name extrusion, we define a first order operator δ à la De Brujin, that shifts any name to the successor, thus generating a new name in x_0. The advantage of this choice is that it does not need a notion of support, i.e., the set of names effectively present, which would yield a second order model. Similarly, Pitts's nominal logics [64] exploits a first order model, by relying on substitution algebras that do not need considering support names.

The construction in [22] yields a category of bialgebras which satisfy the axioms, while in our case the algebraic specification is just a signature, without axioms. Thus our final coalgebra is larger. However, given a coalgebra which satisfies the axioms, its image in the final coalgebra is the same.

As a further step, we apply our general theorem for calculi with structural axioms to the π-calculus in order to provide a compositional coalgebraic semantics of the calculus. However, we stretch that our theorem is suited for any calculus with parallel composition and restriction. In order to lift a calculus with structural axioms to a bialgebra, we first need to define an algebra of the calculus satisfying the specification and then prove that the axioms bisimulate with respect to the operations of the algebra.

The claim of this part of the thesis is also that mobility issues are relevant to study the expressive of nominal calculi. We have mentioned that nominal calculi are more powerful than earlier paradigms for distributed systems. The reason is that names as well as mechanisms like restriction and scope extrusion can model mobility in the sense of dynamic changing network topologies and allocating new nodes during the computations.

According to this intuition, we will study the expressive power of the join calculus, by comparison with (generalized, colored) Petri nets [62, 66], which are a consolidated model in concurrency. Petri nets represent basic distributed systems that may exhibit complex interaction behaviors when processes (transitions) compete for shared resources (tokens).

The idea behind our work is that join processes can be seen as nets: roughly, names, messages, and elementary definitions of the join calculus correspond respectively to places, tokens, and transitions of Petri nets. The correspondence, however, runs short soon because nets are *not* a value-passing formalism and can express *no* mobility. In fact, they have a static, immutable network topology.

In order to achieve the full expressiveness of the join-calculus, we then consider three extensions of the original Petri nets featuring increasing degree of mobility.

On Security Computing agents exchange data and code over networks. One of the objects of cryptographic protocols is to secure these exchanges: they are small protocols that use cryptography to distribute keys and authenticate principals and data over networks. In general, a public, unsafe network is assumed, accessible by adversaries that can read, modify and delete traffic, and that have control of one or more network principals. Cryptographic protocols are often subject to non-intuitive attacks, especially when assumptions about the environment in which the protocol operates change. For example, the Needham-Schroeder public key protocol [57], which was intended to be used for communication between parties that trust each other, and is secure as long as those assumptions hold, becomes subject to a subtle attack, in case one of the communicating parties were dishonest [44], an assumption which has become more likely in an internet-based environment.

It has long been realized that formal methods can be useful for analyzing the correctness of cryptographic protocols. Formal methods allow one both to do a thorough analysis of the different paths which an intruder can take, and to specify precisely the environmental assumptions that have been made. Probably, the first mention of formal methods as a possible tool for cryptographic protocol analysis was made by Needham and Schroeder in [57]. However, the first work in this area was actually done in the late seventies by Dolev and Yao [24]: they developed a formal model which assumes that multiple executions of the protocol in exam can be running concurrently, that cryptographic algorithms behave like black boxes which obey a limited set of algebraic properties (e.g. the encryption and decryption operations cancel each other out), that the environment is identified with the attacker who can read, alter, and destroy traffic, and may also control some legitimate members of the system. Since its introduction, this model has become universally known as the Dolev-Yao model. Throughout the thesis we will adopt it as our conceptual model.

In the late eighties, the publication of the Burrows, Abadi and Needham (BAN) logic [16] brought the problem of the analysis of cryptographic protocols to the attention of a larger research community. BAN logic is an example of a logic of belief: principals can hold a set of possible beliefs (such as a belief that a message was sent by a certain other principal) and they can use a set of inference rules for deriving new beliefs from old ones. An example would be a rule saying that if A believes that a key K is known only by B and himself, and A receives a message encrypted with K then A believes that the message was sent by B to A. The BAN logic consisted of a very simple and intuitive set of rules, which made it easy to use. Even so, as the BAN paper demonstrated, it was possible to use the logic to detect serious flaws in protocols.

More recently, research has focused on state exploration tools and theorem proving techniques based on the Dolev-Yao model. Much of this work sparked by Lowe's demonstration that it was possible to use a general-purpose model checker FDR, to find a man-in-the-middle attack on the Needham-Schroeder public key protocol [44]. Work since then has progressed in applying both model-checkers [48, 54, 67, 68] and theorem provers [61] to the problem, as well as in design of special-purpose model checkers [19, 47].

Protocol analysis via theorem proving is intuitive but not fully automated and general completeness results are difficult to establish. In the second part of the thesis, we shall focus on approaches based on state exploration.

Within model checking, both the honest participants and the adversary are modeled as communicating processes, described in some appropriate process calculus, such as CSP or the spi-calculus, and the whole system is just the parallel composition of all these processes. A finite-state operational model for this system is then explicitly generated and explored, to check whether any insecure state can ever be reached.

To make standard finite-state model checking applicable to protocol analysis, two simplifying requirements are necessary: (a) there is a bound on the number of protocol runs, and (b) there is a bound on the number of possible messages the intruder can generate and send to honest participants at any moment. Discarding one of these two requirements leads to infinite models. Also, these bounds have to be chosen very carefully: due to the combinatorics of message generation, the size of the model tends to explode as multiple principals and data values are considered. It is known that discarding requirement (a) leads to undecidability of protocol analysis, unless severe restrictions are imposed on the analyzed protocols (see e.g.[6, 20, 26]).

On the other hand, it is less clear to what extent requirement (b) can be relaxed, while preserving generality of protocol format, decidability and effectiveness. As mentioned, according to the Dolev-Yao model, all the messages exchanged between participants in a protocol go through the environment which models the attacker at the same time. Thus, the protocol is usually infinite, even assuming requirement (a).

Symbolic analysis [39, 5, 6, 10] offers an approach alternative to finite-state model checking in that it avoids explicit construction of the whole state-space. The idea is to generate a finite symbolic state-space by replacing infinite families of values with variables. The possible ground values that variables may carry are then restricted by means of some kind of constraints.

The second part of the thesis fits into this stream of work. We propose a general framework, based on symbolic techniques, for automatic analysis of security protocols. Similar approaches have been pursued ([5, 10]) for the case of protocols only using shared-key encryption. Specifically, in [10], constraints on the variables in the symbolic model take the form of most general unifiers.

Our method extends the one in [10] to apply to protocols based on a variety of cryptographic primitives. We start from a base language akin to the applied π-calculus[3], equipped with a set of *generic* cryptographic primitives. We propose a symbolic operational semantics that relies on unification and provides finite and effective protocol models. Next, we give a method to carry out trace analysis directly on the symbolic model. Under certain conditions on the given cryptographic primitives, our method is proven complete for the considered class of properties.

As an application of the symbolic method, we consider an instance of our framework whose primitives correspond to the most common cryptographic functions (shared and public key, digital signature and hashing).

Our symbolic approach is efficient in practice, because the symbolic model is compact, and the refinement procedure at its heart is only invoked on demand and on single symbolic traces. However, claims of efficiency should generally be taken with some care, given that the protocol analysis problem is NP-hard even under very mild hypotheses (see, e.g., [69]).

Finally, we present STA (*Symbolic Trace Analyzer*), a tool for the analysis of security protocols, based on the above mentioned method. The use of symbolic techniques in STA avoids explicit construction of the whole, possibly infinite state-space of protocols. This results in accurate protocol modeling, increased efficiency and more direct formalization. We illustrate the use of STA by analyzing two well-known protocols: Needham-Schroeder and Kerberos. We discuss the results of this analysis, and contrast them with previous work based on finite-state model checking.

Overview of the thesis

The main concern of the thesis is on two crucial issues in models for distributed systems: mobility and security. Accordingly, the thesis is divided in two parts. PART I focuses on models of mobile systems: Chapter 2 provides a formal treatment of the main concepts that will be used in the thesis. Chapter 3 focuses on the study of the expressiveness of the join-calculus by comparison with Petri nets and, as a result, defines three extensions of the original Petri nets featuring increasing degree of mobility. The objective of Chapter 4 is twofold: to describe the novel approach for lifting process calculi with structural axioms to coalgebraic models, and to apply such a result to the π-calculus thus providing a compositional coalgebraic semantics of the calculus.

PART II concerns the use of formal methods for the automatic analysis of cryptographic protocols: In Chapter 5, we introduce a formal method for automatic verification of security protocols. The framework generalizes previous work based on symbolic analysis and applies to protocols that rely on a variety of different crypto-functions. Chapter 6 presents STA, the tool for the analysis of security protocols that implements the method described in Chapter 5: the use of STA is illustrated by analyzing two well-known protocols and its performances contrast with other techniques based on finite-state model checking.

Finally, in Chapter7, we present our conclusions and offer directions for future research.

Published Works

Part of this thesis has appeared in joint publications with Michele Boreale, Ugo Montanari and Vladimiro Sassone. More specifically, Chapter 3 is based on [18]. Chapter 4 is a revised version of [17]. Chapter 5 and Chapter 6 are based on [11] and[12].

Part I

Mobility

Chapter 2

Background

In this chapter we introduce some basic concepts and results we will refer to in the following. We present the π-calculus and the join-calculus in Section 2.2 and Section 2.3, respectively. In Section 2.4 we outline the main results of the theory of coalgebras.

2.1 Names and Permutations

We need some basic definitions and properties on names and on name permutations. We denote with $\mathfrak{N} = \{x_0, x_1, x_2, \ldots\}$ the infinite, countable, totally ordered set of *names* and we use $x, y, z \ldots$ to denote names. A *name substitution* is a function $\sigma : \mathfrak{N} \to \mathfrak{N}$. We denote with $\sigma \circ \sigma'$ the composition of substitutions σ and σ'; that is, $\sigma \circ \sigma'(x) = \sigma(\sigma'(x))$. We use σ to range over substitution and we denote with $[y_1 \mapsto x_1, \cdots, y_n \mapsto x_n]$ the substitution that maps x_i into y_i for $i = 1, \ldots, n$ and which is the identity on the other names. We abbreviate by $[y \leftrightarrow x]$ the substitution $[y \mapsto x, x \mapsto y]$. The *identity substitution* is denoted by id. A *name permutation* is a bijective name substitution. We use ρ to denote a permutation. Given a permutation ρ, we define permutation ρ_{+1} as follows:

$$\frac{-}{\rho_{+1}(x_0) = x_0} \qquad\qquad \frac{\rho(x_n) = x_m}{\rho_{+1}(x_{n+1}) = x_{m+1}} \tag{2.1}$$

Essentially, permutation ρ_{+1} is obtained from ρ by shifting its correspondences to the right by one position.

2.2 The π-Calculus

The π-calculus is an extension of CCS where channels can be transmitted over other channels, so that a process can dynamically acquire new channels. Channel transmission mobility is a powerful extension of the basis communication model. The transmission of a channel over another channel gives the recipient the ability to communicate on that channel.

Many versions of π-calculus have appeared in the literature. The π-calculus we present here is *early*, *monadic*, and has *synchronous* communications.

Let $\mathfrak{N}$ be the countable set of names introduced in the previous section. The π-calculus *agent terms*, ranged over by $P, Q, \ldots$, are closed (wrt. variables X) terms defined by the syntax:

$$P \ ::= \ \mathbf{0} \ \big|\ \pi.P \ \big|\ P|P \ \big|\ P{+}P \ \big|\ (\boldsymbol{\nu} x)\, P \ \big|\ [x{=}y]P \ \big|\ \mathtt{rec}\, X.\, P \ \big|\ X$$

where recursion is guarded[1], and *prefixes*, ranged over by π, are defined as:

$$\pi \ ::= \ \tau \ \big|\ \bar{x}y \ \big|\ x(y).$$

The occurrences of y in $x(y).P$ and $(\boldsymbol{\nu} y)\, P$ are bound; *free names* and *bound names* of agent term P are defined as usual and we denote them with $\mathrm{fn}(P)$ and $\mathrm{bn}(P)$, respectively. Also, we denote with $\mathrm{n}(P)$ and $\mathrm{n}(\pi)$ the sets of (free and bound) names of agent term P and prefix π respectively.

If σ is a name substitution, we denote with $\sigma(P)$ the agent term P whose free names have been replaced according to substitution σ, in a capture-free way.

The *static* π-calculus agent terms, ranged over by S, are defined by the syntax:

$$S \ ::= \ \pi.P \ \big|\ P{+}P \ \big|\ [x{=}y]P \ \big|\ \mathtt{rec}\, X.\, P.$$

We define π-calculus *agents* (π*-agents* in brief) as agent terms up to a *structural congruence* $\equiv$; it is the smallest congruence that satisfies the following axioms:

(alpha)	$P \equiv Q$ if P and Q are alpha equivalent
(par)	$P|\mathbf{0} \equiv P \qquad P|Q \equiv Q|P \qquad P|(Q|R) \equiv (P|Q)|R$
(res)	$(\boldsymbol{\nu} x)\,(\boldsymbol{\nu} y)\, P \equiv (\boldsymbol{\nu} y)\,(\boldsymbol{\nu} x)\, P \qquad (\boldsymbol{\nu} x)\, P \equiv P \quad \text{if } x \notin \mathrm{fn}(P)$
	$(\boldsymbol{\nu} x)\,(P|Q) \equiv P|(\boldsymbol{\nu} x)\, Q \quad \text{if } x \notin \mathrm{fn}(P)$

Note that $(\boldsymbol{\nu} x)\, \mathbf{0} \equiv \mathbf{0}$ is a particular case of the last axiom above. We do not consider axioms for summation ($+$), matching ($=$), and recursion ($\mathtt{rec}\,.$), since we aim at introducing an algebra with only parallel composition and restriction. We remark that $P \equiv Q$ implies $\sigma(P) \equiv \sigma(Q)$ and $\mathrm{fn}(P) = \mathrm{fn}(Q)$; so, it is possible to define substitutions and free names also for agents.

[1] Recursion is guarded in P iff in every subterm $\mathtt{rec}\, X.\, Q$ of P, variable X appears within a context $\pi.\_$.

Below, we introduce an example of a static π-agent. Throughout this chapter, we will adopt it as a running example.

Example 2.2.1. Let $P = \mathtt{rec}\, X.\, (\boldsymbol{\nu} y)\, \overline{y} x_2 \mathbf{0} \,|\, \overline{x_2} y.X$ be a static π-agent. Agent P generates new names and extrudes them on a channel x_2; in parallel, P sends name x_2 on each generated channel.

The *actions* an agent can perform are defined by the following syntax:

$$\alpha \ ::= \ \tau \ \big| \ xy \ \big| \ x(z) \ \big| \ \bar{x}y \ \big| \ \bar{x}(z)$$

and are called respectively *synchronization*, *free input*, *bound input*, *free output* and *bound output* actions; x and y are free names of α ($\mathrm{fn}(\alpha)$), whereas z is a bound name ($\mathrm{bn}(\alpha)$); moreover $\mathrm{n}(\alpha) = \mathrm{fn}(\alpha) \cup \mathrm{bn}(\alpha)$.

The standard operational semantics of the π-calculus is defined via labeled transitions $P \xrightarrow{\alpha} P'$, where P is the starting agent, P' is the target one and α is an action. We refer to [53] for further explanations of the transition relation.

(π-Tau) $\tau.P \xrightarrow{\tau} P$	(π-Out) $\bar{x}y.P \xrightarrow{\bar{x}y} P$
(π-Inp) $x(y).P \xrightarrow{xz} [z \mapsto y]P$	(π-Inp′) $x(y).P \xrightarrow{x(y)} P$
(π-Sum) $\dfrac{P_1 \xrightarrow{\alpha} Q_1}{P_1{+}P_2 \xrightarrow{\alpha} Q_1}$ and symmetric	(π-Match) $\dfrac{P \xrightarrow{\alpha} Q}{[x = x]P \xrightarrow{\alpha} Q}$
(π-Rec) $\dfrac{P[\mathtt{rec}\, X.\, P/X] \xrightarrow{\alpha} Q}{\mathtt{rec}\, X.\, P \xrightarrow{\alpha} Q}$	(π-Par) $\dfrac{P \xrightarrow{\alpha} Q}{P \vert R \xrightarrow{\alpha} Q \vert R}$ if $\mathrm{bn}(\alpha) \cap \mathrm{fn}(R) = \varnothing$
(π-Res) $\dfrac{P \xrightarrow{\alpha} Q}{(\boldsymbol{\nu} y)\, P \xrightarrow{\alpha} (\boldsymbol{\nu} y)\, Q}$ if $y \notin \mathrm{n}(\alpha)$	(π-Open) $\dfrac{P \xrightarrow{\bar{x}y} Q}{(\boldsymbol{\nu} y)\, P \xrightarrow{\bar{x}(y)} Q}$ if $x \neq y$
(π-Com) $\dfrac{P_1 \xrightarrow{\bar{x}y} Q_1 \quad P_2 \xrightarrow{xy} Q_2}{P_1 \vert P_2 \xrightarrow{\tau} Q_1 \vert Q_2}$	(π-Close) $\dfrac{P_1 \xrightarrow{\bar{x}(y)} Q_1 \quad P_2 \xrightarrow{x(y)} Q_2}{P_1 \vert P_2 \xrightarrow{\tau} (\boldsymbol{\nu} y)\, (Q_1 \vert Q_2)}$

Table 2.1. Early operational semantics.

The operational semantics we consider here is reported in Table 2.1. It differs from the (standard) *early operational semantics* because it includes a bound input rule (π-Inp′) and rule (π-Close) replaces rule

$$(\pi\text{-Std-Close})\ \frac{P_1 \xrightarrow{\bar{x}(y)} Q_1 \quad P_2 \xrightarrow{xy} Q_2}{P_1 | P_2 \xrightarrow{\tau} (\boldsymbol{\nu} y)\, (Q_1 | Q_2)} \quad \text{if } y \notin \mathrm{fn}(P_2)$$

To understand why the two semantics are equivalent, let us consider the π-agents $P_1 = (\boldsymbol{\nu} y)\, \bar{x}y.\mathbf{0}$ and $P_2 = \bar{z}y.\mathbf{0} \,|\, x\,(z).\mathbf{0}$. A transition $P_1 | P_2 \xrightarrow{\tau} (\boldsymbol{\nu} y)\, \bar{z}y.\mathbf{0}$

should not be allowed because it would cause name y in the first component $\overline{z}y.\mathbf{0}$ of P_2 to be captured by restriction $(\boldsymbol{\nu}y)$. Indeed, on the one hand, if we start with $P_1 \xrightarrow{\overline{x}(y)} \mathbf{0}$ and $P_2 \xrightarrow{\overline{x}y} \overline{z}y.\mathbf{0}$, the above transition is prevented by the side condition of rule (π-STD-CLOSE); on the other hand, if by rule (π-INP) we get $P_2 \xrightarrow{x\,y} \overline{z}y.\mathbf{0}$, rule (π-CLOSE) cannot be applied since the input action is not bound, while if we employ rule (π-INP′) and we obtain $\overline{x}(y).\mathbf{0} \xrightarrow{\overline{x}(y)} \mathbf{0}$, then rule (π-PAR) cannot be applied to P_2, since y is both bound in $x\,(y)$ and free in $\overline{z}\,y.\mathbf{0}$.

2.3 The Join Calculus

In the thesis we shall focus on a monadic version of the join calculus [31, 33], writing its operational semantics in terms of a reduction system as in [60]. For a thorough introduction the reader is referred to the literature. Let $\mathcal{N}m$ be an infinite set of names and let $x, y \ldots, u, v, \ldots$ range over $\mathcal{N}m$. Join terms, definitions, and patters are given by the following grammar. Thus, a join term P is either the

$$\begin{array}{lll}
\text{Terms} & P, Q & ::= \ 0 \ \mid \ x\langle y\rangle \ \mid \ P \mid Q \ \mid \ \texttt{def}\ D\ \texttt{in}\ P \\
\text{Definitions} & D, E & ::= \ J \triangleright P \ \mid \ D \wedge E \\
\text{Join Patterns} & J, K & ::= \ x\langle y\rangle \ \mid \ J \mid K
\end{array}$$

'null' term, an emission $x\langle y\rangle$ of message y on channel x, a parallel composition of terms, or a local definition. A definition D is a set of elementary definitions $J \triangleright P$ matching join patterns J to terms P.

The sets of defined names dn, received names rn, and free names fn are defined below, and then extended to sets of terms in the obvious way. A *renaming*

$$\begin{array}{llll}
\mathrm{rn}(x\langle y\rangle) & = \{y\} & \mathrm{rn}(J \mid K) & = \mathrm{rn}(J) \cup rn(K) \\
\mathrm{dn}(x\langle y\rangle) & = \{x\} & \mathrm{dn}(J \mid K) & = \mathrm{dn}(J) \cup \mathrm{dn}(K) \\
\mathrm{dn}(D \wedge E) & = \mathrm{dn}(D) \cup \mathrm{dn}(E) & \mathrm{dn}(J \triangleright P) & = \mathrm{dn}(J) \\
\mathrm{fn}(x\langle y\rangle) & = \{x\} \cup \{y\} & \mathrm{fn}(P \mid Q) & = \mathrm{fn}(P) \cup \mathrm{fn}(Q) \\
\mathrm{fn}(D \wedge E) & = \mathrm{fn}(D) \cup \mathrm{fn}(E) & \mathrm{fn}(J \triangleright P) & = \mathrm{dn}(J) \cup (\mathrm{fn}(P) \setminus \mathrm{rn}(J))
\end{array}$$
$$\mathrm{fn}(\texttt{def}\ D\ \texttt{in}\ P) = (\mathrm{fn}(P) \cup \mathrm{fn}(D)) \setminus \mathrm{dn}(D)$$

σ is a map from names to names that is the identity except on a finite number of names. We indicate by $\mathrm{dom}(\sigma)$ the set of names on which σ is not the identity, and by $\mathrm{cod}(\sigma)$ its image, using function application for the renaming of free names in terms.

Definition 2.3.1. The *structural congruence* $\equiv$ is the smallest substitutive equivalence relation on terms that satisfies the following rules, where θ and σ are *one-to-one* renamings such that $\mathrm{dom}(\theta) \subseteq \mathrm{rn}(D)$, $\mathrm{cod}(\theta) \cap \mathrm{fn}(D) = \varnothing$ and $\mathrm{dom}(\sigma) \subseteq \mathrm{dn}(D)$, $\mathrm{cod}(\sigma) \cap \mathrm{fn}(\theta D, P) = \varnothing$.

$$
\begin{aligned}
\mathtt{def}\ D\ \mathtt{in}\ P &\equiv \mathtt{def}\ \sigma\theta D\ \mathtt{in}\ \sigma P && && (2.2)\\
P \mid \mathtt{def}\ D\ \mathtt{in}\ Q &\equiv \mathtt{def}\ D\ \mathtt{in}\ P \mid Q && \scriptstyle \mathrm{fn}(P)\cap\mathrm{dn}(D)=\varnothing && (2.3)\\
\mathtt{def}\ D\ \mathtt{in}\ \mathtt{def}\ E\ \mathtt{in}\ P &\equiv \mathtt{def}\ D \wedge E\ \mathtt{in}\ P && \scriptstyle \mathrm{dn}(E)\cap(\mathrm{dn}(D)\cup\mathrm{fn}(D))=\varnothing && (2.4)
\end{aligned}
$$

plus equations stating that 0 is the unit for $\mid$ and that $\mid$ and $\wedge$ are associative and commutative.

Taking $\mathrm{dom}(\sigma) = \varnothing$, rule (2.2) expresses the α-equivalence of definitions up to a renaming of their received names, while considering $\mathrm{dom}(\theta) = \varnothing$ we obtain α-equivalence of terms up to renaming of defined names. Rule (2.3) formalizes the scope extrusion of names, and rule (2.4) states that, under conditions that avoid name clashes, definitions can equivalently be gathered together by the $\wedge$ connective. Interestingly, rule (2.4) is problematic in the design of polymorphic type systems for the join calculus [34, 60], essentially because it may introduce polymorphic (mutual) recursion. This, however, is not an issue in the present setting. On the contrary the law is instrumental in establishing our results.

The operational semantics adopted here corresponds to the original one based on the chemical abstract machine [9]. Terms within a reduction context play the role of 'molecules', definitions determine the 'reactions'. The reduction context determines which join pattern is matched in a definition and binds its received names. The reduction rule replaces in the context the matched pattern by the right-hand side of its definition.

Definition 2.3.2 (cf. [60]). The *reduction* $\longrightarrow$ is the smallest substitutive relation on $\equiv$-classes that satisfies the following rule:

$$\mathtt{def}\ D \wedge J \triangleright P\ \mathtt{in}\ C[\sigma J] \quad \rightarrow \quad \mathtt{def}\ D \wedge J \triangleright P\ \mathtt{in}\ C[\sigma P] \qquad \text{if } \mathrm{dom}(\sigma) \subseteq \mathrm{rn}(J),$$

where C is a *reduction context*, i.e.

$$C ::= [\,] \mid \mathtt{def}\ D\ \mathtt{in}\ C \mid C \mid P \mid P \mid C$$

and $C[P\,]$ denotes the reduction context C with its hole filled by P.

For $x, v \in \mathrm{fn}(P)$, we use $P{\Downarrow}x\langle v\rangle$ to test the ability of P to eventually emit v on x. Formally,

$$P{\Downarrow}x\langle v\rangle \text{ if and only if } \exists Q.\ P \longrightarrow^{*} Q \mid x\langle v\rangle.$$

We write $P{\Downarrow}x$ when, as in the standard case, the value v is not relevant.

On Linearity. Here we no longer require join patterns to be linear, i.e., we allow names to occur several times in patterns. There are two aspects to this. The first one consists of renouncing the linearity of defined names in join patterns. It presents no further difficulties for implementations [43], and we see no reason not to consider it. In terms of Petri nets it corresponds to adding multiplicities to arcs. More problematic in distributed implementations is relaxing the linearity of received names, because this amounts to equating values on different channels. Nevertheless we decided to adopt it, as it corresponds to a form of pattern matching essential in colored Petri nets. It is worth remarking that our work is largely independent of this choice: we could as well keep linearity of received names at the price of disallowing matching in our colored nets, and our results would stay, *mutatis mutandis*.

2.4 Coalgebras

We recall that an algebra A over a signature Σ (Σ-algebra in brief) is defined by a carrier set $|A|$ and, for each operation $op \in \Sigma$ of arity n, by a function $op^A : |A|^n \to |A|$. A homomorphism (or simply a morphism) between two Σ-algebras A and B is a function $h : |A| \to |B|$ that commutes with all the operations in Σ, namely, for each operator $op \in \Sigma$ of arity n, we have $op^B(h(a_1), \cdots, h(a_n)) = h(op^A(a_1, \ldots, a_n))$. We denote by $\mathbf{Alg}(\Sigma)$ the category of Σ-*algebras* and Σ-*morphisms*. The following definition introduces labeled transition systems whose states have an algebraic structure.

Definition 2.4.1 (transition systems). *Let Σ be a signature, and L be a set of labels. A* transition system *over Σ and L is a pair $lts = \langle A, \longrightarrow_{lts}\rangle$ where A is a nonempty Σ-algebra and $\longrightarrow_{lts} \subseteq |A| \times L \times |A|$ is a labeled transition relation. For $\langle p, l, q\rangle \in \longrightarrow_{lts}$ we write $p \xrightarrow{l}_{lts} q$.*

Let $lts = \langle A, \longrightarrow_{lts}\rangle$ and $lts' = \langle B, \longrightarrow_{lts'}\rangle$ be two transition systems. A morphism $h : lts \to lts'$ of transition systems over Σ and L (lts morphism, in brief) is a Σ-morphism $h : A \to B$ such that $p \xrightarrow{l}_{lts} q$ implies $f(p) \xrightarrow{l}_{lts'} f(q)$.

The notion of bisimulation on structured transition systems is the classical one.

Definition 2.4.2 (bisimulation). *Let Σ be a signature, L be a set of labels, and $lts = \langle A, \longrightarrow_{lts}\rangle$ be a transition system over Σ and L.*

A relation $\mathcal{R}$ over $|A|$ is a bisimulation *if $p \mathrel{\mathcal{R}} q$ implies:*

- *for each $p \xrightarrow{l} p'$ there is some $q \xrightarrow{l} q'$ such that $p' \mathrel{\mathcal{R}} q'$;*
- *for each $q \xrightarrow{l} q'$ there is some $p \xrightarrow{l} p'$ such that $p' \mathrel{\mathcal{R}} q'$.*

Bisimilarity $\sim_{lts}$ *is the largest bisimulation.*

Given a signature Σ and a set of labels L, a collection of SOS rules can be regarded as a specification of those transition systems over Σ and L that have a transition relation closed under the given rules.

Definition 2.4.3 (SOS rules). *Given a signature Σ and a set of labels L, a sequent $p \xrightarrow{l} q$ (over L and Σ) is a triple where $l \in L$ is a label and p, q are Σ-terms with variables in a given set X.*

An SOS rule *r over Σ and L takes the form:*

$$\frac{p_1 \xrightarrow{l_1} q_1 \ \cdots \ p_n \xrightarrow{l_n} q_n}{p \xrightarrow{l} q}$$

where $p_i \xrightarrow{l_i} q_i$ as well as $p \xrightarrow{l} q$ are sequents.

We say that transition system $lts = \langle A, \longrightarrow_{lts} \rangle$ satisfies *a rule r like above if each assignment to the variables in X that is a solution*[2] *to $p_i \xrightarrow{l_i} q_i$ for $i = 1, \ldots, n$ is also a solution to $p \xrightarrow{l} q$.*

We represent with

$$\begin{array}{c} p_1 \xrightarrow{l_1} q_1 \ \cdots \ p_n \xrightarrow{l_n} q_n \\ \hline \vdots \\ \hline p \xrightarrow{l} q \end{array}$$

a proof*, with premises $p_i \xrightarrow{l_i} q_i$ for $i = 1, \ldots, n$ and conclusion $p \xrightarrow{l} q$, obtained by applying the rules in R.*

Definition 2.4.4 (transition specifications). *A* transition specification *is a tuple $\Delta = \langle \Sigma, L, R \rangle$ consisting of a signature Σ, a set of labels L, and a set of SOS rules R over Σ and L.*

A transition system over Δ is a transition system over Σ and L that satisfies rules R.

It is well known that ordinary labeled transition systems (i.e., transition systems whose states do not have an algebraic structure) can be represented as coalgebras for a suitable functor [70].

Definition 2.4.5 (coalgebras). *Let $F : \mathcal{C} \to \mathcal{C}$ be a functor on a category $\mathcal{C}$. A* coalgebra for *F, or F-*coalgebra*, is a pair $\langle A, f \rangle$ where A is an object and $f : A \to F(A)$ is an arrow of $\mathcal{C}$. A F-*cohomomorphism *(or simply F-*morphism*) $h : \langle A, f \rangle \to \langle B, g \rangle$ is an arrow $h : A \to B$ of $\mathcal{C}$ such that*

$$h; g = f; F(h). \tag{2.5}$$

We denote with $\mathbf{Coalg}(F)$ *the category of F-coalgebras and F-morphisms.*

[2]Given $h : X \to A$ and its extension $\hat{h} : T_\Sigma(X) \to A$, h is a solution to $p \xrightarrow{l} q$ for lts if and only if $\hat{h}(p) \xrightarrow{l}_{lts} \hat{h}(q)$.

Proposition 2.4.6. *For a fixed set of labels L, let $P_L : \mathbf{Set} \to \mathbf{Set}$ be the functor defined on objects as $P_L(X) = \mathcal{P}(L \times X + X)$, where $\mathcal{P}$ denotes the countable powerset functor, and on arrows as $P_L(h)(S) = \{\langle l, h(p)\rangle \mid \langle l, p\rangle \in S \cap L \times X\} \cup \{h(p) \mid p \in S \cap X\}$, for $h : X \to Y$ and $S \subseteq L \times X + X$. Then P_L-coalgebras are in a one-to-one correspondence with transition systems[3] on L, given by $f_{lts}(p) = \{\langle l, q\rangle \mid p \xrightarrow{l}_{lts} q\} \cup \{p\}$ and, conversely, by $p \xrightarrow{l}_{lts_f} q$ if and only if $\langle l, q\rangle \in f(p)$.*

In [21] the generalized notion of lax cohomomorphism is given, in order to accommodate also the more general definition of lts morphisms in a (lax) coalgebraic framework. To make clear their intuition, let $f : A \to P_L(A)$ and $g : B \to P_L(B)$ be two P_L-coalgebras and let $h : A \to B$ be a P_L-morphism. If we split the morphism condition (2.5) for h in the conjunction of the two inclusions $f; P_L(h) \subseteq h; g$ and $h; g \subseteq f; P_L(h)$, then it is easily shown that the first inclusion expresses "preservation" of transitions, while the second one corresponds to "reflection". Thus, lts morphisms can be seen as arrows (i.e., functions in **Set**) that satisfy the first inclusion, while lts morphisms which also satisfy the reflection inclusion are P_L-morphisms. This observation will be useful in Sect. 4.1.

Definition 2.4.7 (De Simone format). *Given a signature Σ and a set of labels L, a rule r over Σ and L is in* De Simone format *if it has the form:*

$$\frac{\{x_i \xrightarrow{l_i} y_i \mid i \in I\}}{op(x_1, \ldots, x_n) \xrightarrow{l} p}$$

where $op \in \Sigma$, $I \subseteq \{1, \ldots, n\}$, p is linear and the variables y_i occurring in p are distinct from variables x_i, except for $y_i = x_i$ if $i \notin I$.

The following results are due to [74] and concern *bialgebras*, i.e., coalgebras in $\mathbf{Alg}(\Sigma)$. Bialgebras enjoy the property that the unique morphism to the final bialgebra, which exists under reasonable conditions, induces a bisimulation that is a congruence with respect to the operations, as noted in the introduction.

Proposition 2.4.8 (lifting of P_L). *Let $\Delta = \langle \Sigma, L, R\rangle$ be a transition specification with rules in De Simone format.*

Define $P_\Delta : \mathbf{Alg}(\Sigma) \to \mathbf{Alg}(\Sigma)$ as follows:

- $|P_\Delta(A)| = P_L(|A|)$;
- *whenever* $\dfrac{\{x_i \xrightarrow{l_i} y_i \mid i \in I\}}{op(x_1, \ldots, x_n) \xrightarrow{l} p} \in R$ *then*

[3] Notice that this correspondence is well defined also for transition systems with sets of states, rather than with algebras of states as required in Definition 2.4.1.

$$\frac{\langle l_i, p_i \rangle \in S_i,\ i \in I \quad q_j \in S_j,\ j \notin I}{\langle l, p[p_i/y_i,\ i \in I,\ q_j/y_j,\ j \notin I] \rangle \in op^{P_\Delta(A)}(S_1, \ldots, S_n)};$$

- *if $h : A \to B$ is a morphism in $\mathbf{Alg}(\Sigma)$ then $P_\Delta(h) : P_\Delta(A) \to P_\Delta(B)$ and $P_\Delta(h)(S) = \{ \langle l, h(p) \rangle \mid \langle l, p \rangle \in S \cap (L \times |A|) \} \cup \{ h(p) \mid p \in S \cap |A| \}$.*

Then P_Δ is a well-defined functor on $\mathbf{Alg}(\Sigma)$.

Corollary 2.4.9. *Let $\Delta = \langle \Sigma, L, R \rangle$ be a transition specification with rules R in De Simone format.*

Any morphism $h : f \to g$ in $\mathbf{Coalg}(P_\Delta)$ entails a bisimulation $\sim_h$ on lts_f, that coincides with the kernel of the morphism. Bisimulation $\sim_h$ is a congruence for the operations of the algebra.

Moreover, the category $\mathbf{Coalg}(P_\Delta)$ has a final object. Finally, the kernel of the unique P_Δ-morphism from f to the final object of $\mathbf{Coalg}(P_\Delta)$ is a relation on the states of f which coincides with bisimilarity on lts_f and is a congruence.

The theory described so far accounts for the lifting of the functor from **Set** to $\mathbf{Alg}(\Sigma)$; a further step is needed to lift a P_L-coalgebra to be a P_Δ-coalgebra. Indeed, the above step is obvious in the particular case of $f : A \to P_\Delta(A)$, with $A = T_\Sigma$ and f unique by initiality, namely when A has no structural axioms and no additional constants, and lts_f is the minimal transition system satisfying Δ.

As an example that in general the lifting may not succeed, let us consider the case of the chemical abstract machine *CHAM*. For our purposes, the signature Σ_c of a *CHAM* is defined as:

$$\Sigma_c \ ::= \ \mathbf{0} \ \big|\ \_|\_ \ \big|\ a.\_ \ \big|\ \bar{a}.\_ \ \big|\ \mathtt{redex}_{\,a}(\_,\_)$$

and E_c is the set of axioms for commutativity, associativity, *id*, $\mathbf{0}$ plus an axiom $\mathtt{redex}_{\,a}(p,q) = a.p \mid \bar{a}.q$. The only reduction rule $\mathtt{redex}_{\,a}(p,q) \longrightarrow p \mid q$ is in De Simone format. For P_{Δ_c} the usual poweralgebra functor on $\mathbf{Alg}(\Sigma_c)$, the transition system of *CHAM* forms a P_L-coalgebra, but not a P_{Δ_c}-coalgebra [4]. And bisimilarity is not a congruence as, for example, $\mathbf{0} \sim a.p$ but $\bar{a}.p \not\sim a.p \mid \bar{a}.p$.

[4]Indeed, axiom $\mathtt{redex}_{\,a}(p,q) = a.p \mid \bar{a}.q$ does not satisfy the "bisimulation" condition required in Theorem 4.1.4.

Chapter 3

On the Expressiveness of the Join Calculus

The join calculus [31, 33] is an algebra of mobile processes with asynchronous name-passing communication that simplifies the π-calculus by enforcing the hypothesis of *unique receptors*, as noticed in the introduction. This means that there is at most one process receiving messages on a name, and is a distinctive feature of the join calculus that makes it suitable for distributed implementations, as channels may be allocated at their receptor process. This chapter focuses on the version of join calculus introduced in Section2.3, studying its expressiveness by establishing a tight link to Petri nets.

Petri nets [62, 66] are a fundamental model in concurrency, representing basic distributed machines that, although rudimentary, may exhibit complex interaction behaviors when processes (transitions) compete for shared resources (tokens). Operational in nature, Petri nets have been studied extensively from the semantic viewpoint (see [71] for some references).

The analogy between join terms and nets is relatively simple, and was first noticed in [8] and, recently, in [59]. Names, messages, and elementary definitions of the join calculus (cf. 2.3) correspond respectively to places, tokens, and transitions of Petri nets (cf. 3.1). The correspondence, however, runs short soon because nets are *not* a value-passing formalism and can express *no* mobility. Regrettably, they have a static, immutable network topology. This suggests to look for suitable extensions of Petri nets – in particular nets with mobility – that might be profitably applied to the study of mobile networks.

We consider here three extensions of place/transition Petri nets (Π_0) obtained by adding one by one, in a hierarchical fashion, the features needed to achieve the full expressiveness of join calculus. Namely, *value-passing*, accounted for by (a version of) *colored nets* (Π_1), mobility as *network reconfigurability*, achieved by introducing *reconfigurable nets* (Π_2), and dynamically growing, *open networks*, modeled by the notion of *dynamic nets* (Π_3).

High level nets exist in several shapes and versions. The distinctive features of the mobile nets presented here is that input places of transitions are private and presets are immutable. A reconfigurable net can deliver tokens in different places at each firing, according to the input it receives, while a dynamic net can spawn a new net to life. But they *cannot* change the input arcs of any transitions. Following [58], where an algebra with such properties has been considered, we believe that this is the key to a tractable compositional semantic framework, the key to control generality. Most noticeably, it corresponds to unique receptiveness in the join calculus. A further naturality criteria in our view is that, as it turns out, dynamic nets can be proved *equivalent* to the join calculus.

The technical bulk of this work is characterizing classes of join terms corresponding to the classes of the high-level nets discussed above. More precisely, our approach consists of designing four type systems Δ_i, $i = 0, \dots, 3$, that single out the terms corresponding to the nets in Π_i, respectively for $i = 0, \dots, 3$. In particular, for a fixed, semantic-preserving, well-behaved translation $[\![\_]\!]$, a join term P is typeable in Δ_i if and only if $[\![P]\!]$ belongs to Π_i. System Δ_0 is aimed at constraining terms to place/transition nets and, therefore, forbids dynamic process creation and nontrivial messages, Δ_1 relaxes the latter limitation, but enforces a strict distinction between channels and values, supporting value-passing but forbidding any mobility. Such distinction is then relaxed in Δ_2. System Δ_3 relaxes the dynamicity constraint and, therefore, turns out to be trivial, i.e., each term is typeable in Δ_3. That is, dynamic nets *coincide* with join terms. Finally, we provide a system Δ_4 that sums the features of all the others.

It is worth remarking that these systems are very rudimentary from the type theoretic viewpoint: there are no complex types or rules, nor sophisticated issues such as polymorphism [34, 60] or similar. However, we believe that our formalization is quite interesting, suggestive, and worth pursuing, because the nature of join-terms makes it natural to express our conditions in systems of rules. Nevertheless, our work here is about comparing

models. As a matter of fact, relating join calculus and Petri nets may be beneficial for both. On one of the edges of the connection, in fact, the join calculus may provide Petri nets with a compositional framework, together with behavioral semantics such as testing [42] and bisimulation [14]. Also, it may suggest interesting, semantically well-founded extensions of Petri nets, such as reconfigurable and dynamic nets here, mobile nets in [8], and functional nets in [59]. On the other edge, it opens the join calculus to an entire body of results on the semantics of noninterleaving concurrency, such as those supported by monoidal categories (see, e.g., [15]).

In this chapter, we consider a generalization of the join calculus whose join patterns may not be linear (see Subsection 2.3).

Related Work. Although focus and approach here differ radically from [8], our models are clearly related to those of Asperti and Busi. While we aim at representing the join calculus precisely, the *dynamic nets* of [8] are more loosely inspired by it and are more general than ours. In particular, they do not enforce privateness of input places – that is unique receptors – as our nets do. The analogy between Petri nets and join calculus has also provided the inspiration for Odersky's *functional nets* [59]. Although *loc. cit.* is quite different in spirit from our work, the relationships between Odersky's framework and our work here may be worth of further investigation.

Structure of the Chapter. The chapter is organized as follows. Section 3.1 introduces Place Transition nets, the type system Δ_0 and relates typeability to PT nets. The following sections respectively define colored, reconfigurable, dynamic nets and their respective type systems. Finally, Sect. 3.5 introduces Δ_4.

3.1 Place Transition Nets

A *multiset* on a set P is a function $\mu\colon P \to \mathbb{N}$. We shall use $\mu(P)$ to denote the set of finite multisets, i.e., markings, on P. The *sum* of $\mu_0, \mu_1 \in \mu(P)$ is the multiset $\mu = \mu_0 \oplus \mu_1$ such that $\forall p \in P.\ \mu(p) = \mu_0(p) + \mu_1(p)$.

Petri nets consists of places, transitions, and tokens. In view of relating nets and join terms, we identify places with names. In addition to $\mathcal{N}m$ – that will play the role of *public* places (*free* names) – we shall consider an infinite set ω of distinguished *private* places (*bound* names). In the following, $\mathcal{N}m_\omega$ stands for the union $\mathcal{N}m + \omega$.

Definition 3.1.1 (PT Nets). A *place/transition net* is a tuple $N = (T, \partial_0, \partial_1, \mu_0)$, where T is a finite set of transitions, $\partial_0, \partial_1\colon T \to \mu(\mathcal{N}m_\omega)$ are the pre- and post-set functions, and $\mu_0 \in \mu(\mathcal{N}m_\omega)$ is the initial marking. We shall use Π_0 to refer to PT Petri nets.

Thus, to simplify notations, we include in every net all the names of $\mathcal{N}m_\omega$ with the understanding that *only* marked places, i.e., those carrying tokens, and places connected to some transitions are effectively to be considered in the net. In particular, the *empty net*, that we denote by $\varnothing$, formally consists of all the places in $\mathcal{N}m_\omega$, but no tokens and no transitions. Analogously, the net consisting of a distribution of tokens on a *multiset* of places μ, has all places but only $\mu(x)$ tokens in each $x \in \mathcal{N}m_\omega$. With abuse of notations, we shall use μ to denote both the multisets and the corresponding net. Also, for $\mu_0, \mu_1 \in \mu(\mathcal{N}m_\omega)$, **tran** $\mu_0\, \mu_1$ stands for the net with a unique transition t with $\partial_0(t) = \mu_0$ and $\partial_1(t) = \mu_1$.

Following our intuition, isomorphisms preserve free names. This is detailed in the definition below, where we use *id* for identities, $+$ for function coproducts, and take the liberty of identifying functions with their extensions to multisets.

Definition 3.1.2. Nets $(T, \partial_0, \partial_1, \mu_0)$ and $(T', \partial'_0, \partial'_1, \mu'_0)$ are *isomorphic* if there exist isomorphisms $f_t \colon T \to T'$ and $f_p \colon \omega \to \omega$ such that $\partial'_i \circ f_t = (id_{\mathcal{N}m} + f_p) \circ \partial_i$, for $i = 0, 1$, and $(id_{\mathcal{N}m} + f_p)(\mu_0) = \mu'_0$.

Observe that nets that only differ for the use of names in ω are isomorphic, yielding a form of α-conversion. In the following, isomorphic nets will be regarded as equal. In particular, as we shall see, net isomorphism corresponds to structural congruence of join terms.

Definition 3.1.3. Let $N = (T, \partial_0, \partial_1, \mu_0)$ and $N' = (T', \partial'_0, \partial'_1, \mu'_0)$ be nets. Denoted by $\otimes$, *parallel composition* juxtaposes nets, merging public places without confusing the private ones. Formally,

$$N \otimes N' = (T + T', \partial_0 + \partial'_0, \partial_1 + \partial'_1, \mu_0 \oplus \mu'_0), \qquad \text{if } \omega(N) \cap \omega(N') = \varnothing,$$

where $\omega(N)$ denotes the names in ω that are used in N, i.e., that are either marked or connected to some transition. Observe that the side condition on private names can always be achieved up to isomorphism. Clearly, $\otimes$ is commutative and associative.

Denoted by ν_x, *restriction* 'hides' the place $x \in \mathcal{N}m$ replacing it by a *fresh* name $i \in \omega$, i.e., a name not occurring in $\omega(N)$. Formally, for $N = (T, \partial_0, \partial_1, \mu_0)$

$$\nu_x N = N[x \leftrightarrow i], \qquad i \text{ is fresh in } N$$

where $N[x \leftrightarrow i]$ is the net $(T, \partial'_0, \partial'_1, \mu'_0)$ whose μ'_0 and ∂'_j coincide respectively with μ_0 and ∂_j ($j = 0, 1$), but for the values they yield on x and i, that are exchanged, i.e., for $\delta \in \{\partial_0, \partial_1, \mu_0\}$, we have that $\delta'(k)$ is equal to $\delta(i)$ if $k = x$, to $\delta(x)$ if $k = i$, and to $\delta(k)$ otherwise.

For $X = \{x_1, \ldots, x_n\} \subseteq \mathcal{N}m$, we use $\nu_X N$ to mean $\nu_{x_1}(\cdots \nu_{x_n} N)$. This is a correct definition because $\nu_x \nu_y N = \nu_y \nu_x N$ for each x, y, up to isomorphism.

The evolution of nets is described in terms of the 'firing' of its transitions. As usual, the firing of t consumes and produces resources, as prescribed by the pre- and post-set functions ∂_i. In the present setting, this can be expressed as

$$N \otimes \partial_0(t)\ [t\rangle\ N \otimes \partial_1(t).$$

We use $N \Downarrow x$ for $x \in \mathcal{N}m$ to mean that there exist transitions $t_1, \ldots, t_n$ of N and a net N' such that $N\ [t_1\rangle \cdots [t_n\rangle\ N' \otimes x$.

Observe that we stop at the single transition semantics of nets, rather than considering also the usual step semantics. We do so in order to match the standard reduction semantics of the join calculus, and it is not a hard limitation. We could easily extend the results to the classical step semantics, provided we equip join terms with multiple concurrent reductions.

The Type System Δ_0

The purpose of Δ_0 is to single out those join terms that are PT Petri nets. In order to achieve this we need to prevent any form of name passing and mobility, imposing a static network structure to terms. In particular, since definitions represent transitions, it is fundamental that their right-hand sides consists of messages only. Also, corresponding to the fact that there is only one atom (*token*) of information delivered in PT nets, we enforce that only empty messages may be exchanged. For notational convenience, we assume in the following the existence of a distinguished name $\bullet \in \mathcal{N}m$ that represents the empty tuple.

The types of Δ_0 are $\diamond$ and $\diamond\diamond$, with τ ranging over them. There are three kind of type judgment.

$$\vdash P : \diamond \quad (P \text{ is ok}) \qquad\qquad \vdash D : \diamond \quad (D \text{ is ok})$$

$$\vdash P : \diamond\diamond \quad (P \text{ is ok and no } \mathtt{def}\ \_\ \mathtt{in}\ \_ \text{ occurs in it})$$

The typing rules are the following.

(P-ZERO)
$$\vdash 0 : \diamond\diamond$$

(P-MESS)
$$\vdash x\langle\bullet\rangle : \diamond\diamond$$

(P-PAR)
$$\frac{\vdash P : \tau \qquad \vdash Q : \tau}{\vdash P \mid Q : \tau}$$

(P-DEF)
$$\frac{\vdash D : \diamond \qquad \vdash P : \diamond}{\vdash \mathtt{def}\ D\ \mathtt{in}\ P : \diamond}$$

(P-SUB)
$$\frac{\vdash P : \diamond\diamond}{\vdash P : \diamond}$$

(D-PAT)
$$\frac{\vdash P : \diamond\diamond}{\vdash J \triangleright P : \diamond}\ \mathrm{rn}(J){=}\{\bullet\}$$

(D-AND)
$$\frac{\vdash D : \diamond \qquad \Gamma \vdash E : \diamond}{\vdash D \wedge E : \diamond}$$

The rules are elementary. The key is (D-PAT) that allows *only* simple processes as right-hand sides of definitions, while (P-MESS) ensures that only emissions of $\bullet$ are well typed and, together with (P-PAR), that only parallel composition of messages can be used inside definitions. We use $\Delta_0 \vdash P$ as a shorthand for typeability, i.e., $\vdash P : \diamond$, in the system Δ_0 above.

Proposition 3.1.4 (Subject Reduction). *If $\Delta_0 \vdash P$ and $P \longrightarrow Q$, then $\Delta_0 \vdash Q$.*

Translating Δ_0 terms to PT Nets

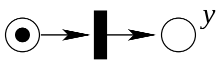
We intend to define a translation from Δ_0-typed join terms to PT nets that can be extended to other type systems and other classes of nets. Therefore, we first give a general map on all join terms, and then prove that it restrict to a well defined encoding of terms typeable in Δ_0 into PT nets. By induction on the structure of terms, $[\![\_]\!]$ is defined as follows.

$$
\begin{array}{llll}
[\![0]\!] & = \varnothing & & \\
[\![x\langle v\rangle]\!] & = (x, v) & [\![J \triangleright P]\!] & = \mathbf{tran}[\![J]\!]\ [\![P]\!] \\
[\![P \mid Q]\!] & = [\![P]\!] \otimes [\![Q]\!] & [\![D \wedge E]\!] & = [\![D]\!] \otimes [\![E]\!] \\
[\![\texttt{def}\ D\ \texttt{in}\ P]\!] & = \nu_{\mathrm{dn}(D)}([\![D]\!] \otimes [\![P]\!]) & &
\end{array}
$$

where $[\![J]\!]$ is J seen as a multiset, i.e., $[\![x\langle v\rangle]\!] = (x, v)$ and $[\![J \mid K]\!] = [\![J]\!] \oplus [\![K]\!]$.

Roughly speaking, we map names to places, messages to markings, and definitions to (groups of) transitions. Since we shall soon consider nets whose tokens carry information, we use (x, v) to denote a marking of place x with value v. Of course, in the current case v always equals $\bullet$, and $(x, \bullet)$ should be read as an exotic way to mean x. In more details, $J \triangleright P$ is mapped into a transition whose pre-set is essentially J and whose post-set is the encoding of P, while $\mid$ and $\wedge$ are mapped homomorphically to $\otimes$. Similarly for $\texttt{def}\ D\ \texttt{in}\ P$, where the use of restriction ensures the correct treatment of local definitions. Here it is important to understand the different roles of $x\langle v\rangle$ because, depending on the context, $(x, \bullet)$ is treated as the net consisting of a single token in place x, or as an arc connecting place x to a transition. We shall get back to this in 3.2.

Example 3.1.5. Let P be the Δ_0-typeable term $\texttt{def}\ x\langle\bullet\rangle \triangleright y\langle\bullet\rangle\ \texttt{in}\ x\langle\bullet\rangle$. Then, using the standard representation of Petri nets enriched with labels that decorate *public* places with their name, the translation of P is

$$[\![P]\!] = \nu_x\big(\mathbf{tran}\{x\}\{y\} \otimes (x, \bullet)\big) =$$ 

.

Observe that, in force the restriction ν_x, the name x does not appear.

As announced, the encoding is well defined on well-typed terms.

Proposition 3.1.6 (Correctness). *If $\Delta_0 \vdash P$, then $[\![P]\!]$ is a PT net.*

Proof. (Sketch.) The proof is based on a simple lemma, namely $\Delta_0 \vdash P$ implies that no process of the form `def _ in _` appears inside a definition. Notice that it follows from Proposition 3.1.4 that, for all x, v, if $P \Downarrow x\langle v \rangle$, then $v = \bullet$. $\square$

In addition to the previous proposition, we can prove a completeness theorem for Δ_0, expressed as follows.

Theorem 3.1.7 (Completeness). *The translation $[\![\_]\!]$ is an isomorphism between the set of terms typeable in Δ_0 and PT nets in Π_0.*

The rest of this section is devoted to illustrating that the $[\![\_]\!]$ is very tight a connection, preserving both structural congruence and dynamic behavior.

Proposition 3.1.8. *Let P and Q be join terms typeable in Δ_0. Then,*

$$P \equiv Q \quad \textit{if and only if} \quad [\![P]\!] = [\![Q]\!].$$

Proof. (Sketch.) Remember that we regard isomorphic nets as equal. The 'only if' direction is proved by structural induction on P, while the 'if' direction by rule induction on $\equiv$, exploiting a lemma that $D \equiv E$ if and only if $[\![D]\!] = [\![E]\!]$, whose proof follows easily from the typeability hypothesis.

$\square$

Proposition 3.1.9. *Let P be a join term typeable in Δ_0. Then,*

$$\forall x \in \mathcal{N}m.\ P \Downarrow x \quad \textit{if and only if} \quad [\![P]\!] \Downarrow x$$

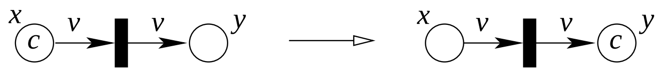
This, together with surjectivity and the compositionality of $[\![\_]\!]$, yields a very strong relationship, allowing to identify (the system represented by) P and (that represented by) $[\![P]\!]$. In other words, it allows us to identify typeability in Δ_0 with Π_0.

3.2 Colored Nets

In colored nets, tokens carry information: the 'colors'. Arcs between places and transitions are labelled by *arc expressions*, which evaluate to multisets of colored tokens after binding free variables to colors. Expressions on input and output arcs describe respectively the resources needed for the firing and those generated by it. Although our treatment below attempts at providing an essential version of colored Petri nets, we maintain one of their fundamental features. Namely, we permit a name to label several of the input arcs of a transition, with the intended meaning that firing is allowed only if the same color can be fetched along all those arcs. This form of pattern matching is the reason why we relaxed the linearity requirement of join patterns. Technically, our model is obtained simply by allowing PT Nets to circulate tokens other than '$\bullet$'. The obvious choice in the present context is to identify colors with names.

Definition 3.2.1 (CNets). A *colored net* is a tuple $(T, \partial_0, \partial_1, \mu_0)$, where T is a finite set of transitions, $\partial_0, \partial_1 \colon T \to \mu(\mathcal{N}m_\omega \times \mathcal{N}m)$, are the pre and post-set functions, and $\mu_0 \in \mu(\mathcal{N}m_\omega \times \mathcal{N}m)$ is the initial marking. We use Π_1 to denote the class of colored Petri nets.

Example 3.2.2. In the above definition we interpret the pairs (p, c) of $\mathcal{N}m_\omega \times \mathcal{N}m$ as representing a place p that carries a color (name) c. The colored net $\mathbf{tran}\{(x, v)\}\{(y, v)\} \otimes (x, c)$ is represented graphically as below on the left.



As usual, the role of names is twofold: in markings they represent colors, e.g., the constant c in x; in transitions they represent bound variables. The net above fetches any token from x, binds it to v, and then delivers the actual value of v to y, as illustrated by the picture above and formalized by the definition below.

Definition 3.2.3. The set of *received colors* of a transition t is

$$\mathtt{rc}(t) = \{x \in \mathcal{N}m \mid\ x \neq \bullet,\ \exists p.\ (p, x) \in \partial_0(t)\}.$$

A *binding* for t is a function $\mathtt{b} \colon \mathcal{N}m \to \mathcal{N}m$ which is the identity on $\mathcal{N}m \smallsetminus \mathtt{rc}(t)$. For $X \in \mu(\mathcal{N}m_\omega \times \mathcal{N}m)$, let $X\langle\mathtt{b}\rangle$ denote the multiset $\{(p, \mathtt{b}(c)) \mid (p, c) \in X\}$.

The firing rule for colored nets is formulated below, for $\mathtt{b}_t$ a binding for t.

$$N \otimes \partial_0(t)\langle\mathtt{b}_t\rangle\ [t\rangle\ N \otimes \partial_1(t)\langle\mathtt{b}_t\rangle$$

Isomorphisms of colored nets are the obvious extension of those of PT nets given in Definition 3.1.2. They allow renaming of private places, but map public ones identically.

It is important to remark that colored nets are considered up to α-conversion, that is renaming of received colors. In particular, colored nets are isomorphic if there is an isomorphism between them after possibly renaming received names.

The Type System Δ_1

The purpose of type system Δ_1 is to characterize colored Petri nets among join terms. Having introduced names, we now face the issue of distinguishing among two kind of names: channels and parameters. In fact, colored nets are a strict value-passing formalism. They are not allowed to use values received along channels as channels themselves, nor to send private names as messages. In order to enforce this, we consider typing environments holding assumptions on free names, *viz.*, if they are channels or messages. Type environments are therefore pairs of *disjoint* sets Γ (the channels) and ∇ (the messages). Type judgments are exactly as before, but for the presence of type environments.

$$\text{(P-ZERO)}\quad \Gamma;\nabla \vdash 0 : \diamond\diamond \qquad \text{(P-MESS)}\quad \Gamma;\nabla \vdash x\langle y\rangle : \diamond\diamond \ (x\notin\nabla,\, y\notin\Gamma) \qquad \text{(P-PAR)}\quad \frac{\Gamma;\nabla \vdash P : \tau \quad \Gamma;\nabla \vdash Q : \tau}{\Gamma;\nabla \vdash P \mid Q : \tau}$$

$$\text{(P-DEF)}\quad \frac{\Gamma,\mathrm{dn}(D);\nabla \vdash D : \diamond \quad \Gamma,\mathrm{dn}(D);\nabla \vdash P : \diamond}{\Gamma;\nabla \vdash \texttt{def}\ D\ \texttt{in}\ P : \diamond} \qquad \text{(P-SUB)}\quad \frac{\Gamma;\nabla \vdash P : \diamond\diamond}{\Gamma;\nabla \vdash P : \diamond}$$

$$\text{(D-PAT)}\quad \frac{\Gamma;\nabla,\mathrm{rn}(J) \vdash P : \diamond\diamond}{\Gamma;\nabla \vdash J \triangleright P : \diamond} \qquad \text{(D-AND)}\quad \frac{\Gamma;\nabla \vdash D : \diamond \quad \Gamma;\nabla \vdash E : \diamond}{\Gamma;\nabla \vdash D \wedge E : \diamond}$$

The structure of the rules matches exactly those of Δ_0, the only difference being the use of Γ in (P-DE) and of ∇ in (D-PAT), so to be able to control the use of names in (P-MESS).

Proposition 3.2.4. *$\Delta_0 \vdash P$ implies $\Delta_1 \vdash P$.*

Proposition 3.2.5 (Subject Reduction). *If $\Delta_1 \vdash P$ and $P \longrightarrow Q$, then $\Delta_1 \vdash Q$.*

A simple inspection of the rules shows that, as Δ_0, system Δ_1 does not allow processes of the form `def _ in _` to appear inside definitions. The other fundamental properties of Δ_1 are expressed in the proposition below.

Proposition 3.2.6. *If $\Delta_1 \vdash P$, then P never emits messages on received names nor emits bound names out of the scope of their definitions.*

Proof. (Sketch.) Exploiting Proposition 3.2.5 it suffices to prove that whenever $P \longrightarrow \texttt{def}\ D \wedge J \triangleright Q\ \texttt{in}\ P'$, if Q may emit on x, i.e., $Q \equiv Q' \mid x\langle v\rangle$, then $x \notin \mathrm{rn}(J)$ and if P' may emit v, i.e., $P' \equiv P'' \mid x\langle v\rangle$, then $v \notin \mathrm{dn}(D) \cup \mathrm{dn}(J)$.

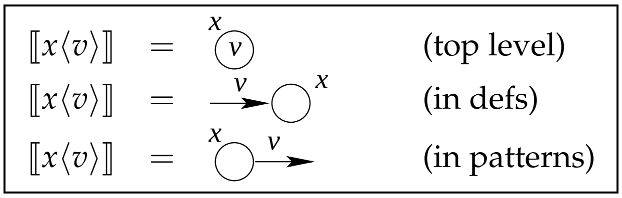
Translating Δ_1 terms to CNets

It follows as a consequence of Proposition 3.2.6 that the translation $[\![\_]\!]$ extends to a well defined map from Δ_1 typeable terms to colored Petri nets.

Proposition 3.2.7 (Correctness). *If $\Delta_1 \vdash P$, then $[\![P]\!]$ is a colored net.*

Here more than before it is important to understand the three different roles played by $x\langle v\rangle$ in the translation. First, when $x\langle v\rangle$ appears outside all definitions, at top level, it is translated as the net consisting of a token v in the place x.

Secondly, when it appears in a join pattern J, it is translated in $[\![J]\!]$ as (x, v), representing the input from place x on a bound variable v. Finally, when $x\langle v\rangle$ appears in the body of a definition, it is translated as (x, v) by the same clause above but, considered as a marking, it represents the output of a free or bound variables in the place x. This is summarized by the pictures below.



Example 3.2.8. Let P be the Δ_1 typeable term $\texttt{def}\ x\langle v\rangle \triangleright y\langle v\rangle\ \texttt{in}\ x\langle c\rangle$. Then $[\![P]\!]$ is the colored net of Example 3.2.2.

Our results about Δ_1 match exactly those for Δ_0. They are listed below.

Theorem 3.2.9 (Δ_1 vs Π_1). *The translation $[\![\_]\!]$ is an isomorphism between the set of terms typeable in Δ_1 and colored nets in Π_1. Moreover, if P and Q are typeable in Δ_1, then*

$$P \equiv Q \quad \text{if and only if} \quad [\![P]\!] = [\![Q]\!]$$

$$P{\Downarrow}x \quad \text{if and only if} \quad [\![P]\!]{\Downarrow}x, \quad \text{for all } x \in \mathcal{N}m.$$

3.3 Reconfigurable Nets

Colored Petri nets have a static structure, i.e., their firings only affects the marking. While this accounts faithfully for value-passing theories, it is completely inadequate to represent dynamically changing structures. Reconfigurable nets generalize colored ones by adding precisely one ingredient: each firing of a transition may deliver to a different set of output places. This equips nets with a mechanism to model networks with reconfigurable topologies, that is networks in which the set of components is fixed, but the connectivity among them may change in time. Formally, this is achieved by a very smooth alteration of the definitions of colored nets – one that allows markings and messages to contain private names – and of their firing rule.

Definition 3.3.1 (RNets). A *reconfigurable net* is a tuple $(T, \partial_0, \partial_1, \mu_0)$, where T is a finite set of transitions, $\partial_0 \colon T \to \mu(\mathcal{N}m_\omega \times \mathcal{N}m)$ and $\partial_1 \colon T \to \mu(\mathcal{N}m_\omega \times \mathcal{N}m_\omega)$, are, respectively, the pre and post-set functions, and $\mu_0 \in \mu(\mathcal{N}m_\omega \times \mathcal{N}m_\omega)$ is the initial marking. We use Π_2 to denote the set of reconfigurable Petri nets.

Restriction ν_x and parallel composition $\otimes$ extends straightforwardly to reconfigurable nets. Also, isomorphisms of reconfigurable Petri nets are obtained extending those of colored nets in the obvious way.

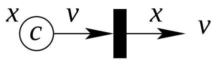
The firing rule is generalized by extending the scope of the binding of received names to include the output places in addition to the names on the output arcs.

Definition 3.3.2. Let $\mathtt{b}$ be a binding. For $X \in \mu(\mathcal{N}m_\omega \times \mathcal{N}m_\omega)$, let $X\langle\!\langle \mathtt{b} \rangle\!\rangle$ denote the multiset $\{(\mathtt{b}_\omega(p), \mathtt{b}_\omega(c)) \mid (p, c) \in X\}$, where $\mathtt{b}_\omega = \mathtt{b} + id_\omega$.

For $\mathtt{b}_t$ a binding for t, the firing rule is then written as

$$N \otimes \partial_0(t)\langle \mathtt{b}_t \rangle \; [t\rangle \; N \otimes \partial_1(t)\langle\!\langle \mathtt{b}_t \rangle\!\rangle$$

Example 3.3.3. The simple reconfigurable net represented below binds v to any name found in x – in this case c – and delivers the name x to the place to which v is bound.



The Type System Δ_2

In order to design a type system Δ_2 corresponding to Π_2 we only need to remove the difference between channels and messages upon which Δ_1 is based, which can be done easily by removing Γ and ∇. This yields a system identical to Δ_0, apart from (P-MESS) that allows any message to be transmitted on any name.

(P-ZERO) $$\vdash 0 : \diamond\diamond$$

(P-MESS) $$\vdash x\langle y \rangle : \diamond\diamond$$

(P-PAR) $$\frac{\vdash P : \tau \qquad \vdash Q : \tau}{\vdash P \mid Q : \tau}$$

(P-DEF) $$\frac{\vdash D : \diamond \qquad \vdash P : \diamond}{\vdash \mathtt{def}\ D\ \mathtt{in}\ P : \diamond}$$

(P-SUB) $$\frac{\vdash P : \diamond\diamond}{\vdash P : \diamond}$$

(D-PAT) $$\frac{\vdash P : \diamond\diamond}{\vdash J \triangleright P : \diamond}$$

(D-AND) $$\frac{\vdash D : \diamond \qquad \vdash E : \diamond}{\vdash D \wedge E : \diamond}$$

Proposition 3.3.4 (Subject Reduction). *If $\Delta_2 \vdash P$ and $P \longrightarrow Q$, then $\Delta_2 \vdash Q$.*

Proposition 3.3.5. $\Delta_1 \vdash P$ *implies* $\Delta_2 \vdash P$.

Translating Δ_2 terms to RNets

The property guaranteed by Δ_2 is that there is no process of the kind `def _ in _` inside definitions, i.e., the topology of the net may change by redirecting output edges, but the components of the net are fixed once and for all. In force of this, we can extend to Δ_2 the correspondence between well-typed terms and nets.

Theorem 3.3.6 (Δ_2 vs Π_2). *The translation $[\![\_]\!]$ is an isomorphism between the set of terms typeable in Δ_2 and reconfigurable nets in Π_2. Moreover, if P and Q are typeable in Δ_2, then*

$$P \equiv Q \quad \text{if and only if} \quad [\![P]\!] = [\![Q]\!]$$
$$P{\Downarrow}x \quad \text{if and only if} \quad [\![P]\!]{\Downarrow}x, \quad \text{for all } x \in \mathcal{N}m.$$

3.4 Dynamic Nets

The obvious generalization of reconfigurable nets is to allow the dynamic creation of components, that we achieve by means of the notions of *dynamic Petri nets*. The idea behind such structures is that the firing of a transition allocates a new net parametric in the actual values of the received names. As the net may consists simply of a marking and no transitions, this includes the standard definition of PT nets. Also colored and reconfigurable nets are, of course, special kinds of dynamic nets. The characteristic feature of dynamic nets, that to our best knowledge distinguishes them by other approaches in the literature and draws a connection to [58], is that, as for reconfigurable nets, input arcs are never modified. While it is possible to modify dynamically the post-set of a transition, and also to spawn new subnets, it is not possible to add places to the presets of transitions. This allows us to formalize our intuition by simply generalizing the post-set functions of nets, allowing $\partial_1(t)$ to be a dynamic net. Of course, this means that the definition of nets becomes recursive.

Definition 3.4.1 (DNets). Let *DNets* be the least set satisfying the equation

$$\mathit{DNets} \cong \big\{(T, \partial_0 \colon T \to \mu(\mathcal{N}m_\omega \times \mathcal{N}m), \partial_1 \colon T \to \mathit{DNets}, \mu_0)\big\},$$

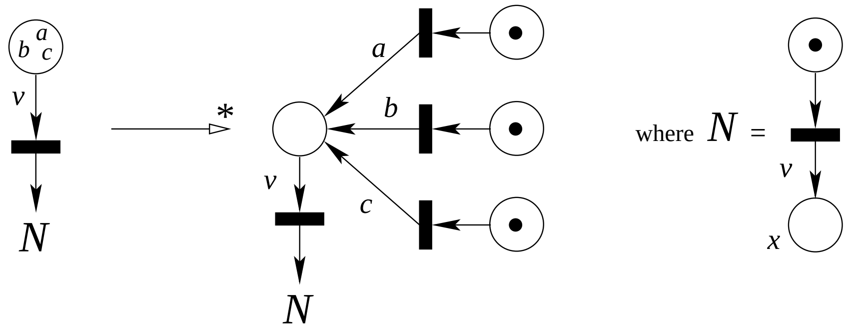
where T is a finite set and μ_0, the initial marking, is in $\mu(\mathcal{N}m_\omega \times \mathcal{N}m_\omega)$. A *dynamic* net *DN* is an element of *DNets*. We use Π_3 to refer to the set of dynamic nets.

Once again, the operations of parallel composition and restriction, and the notions of α-conversion and isomorphism lift smoothly to this setting.

Definition 3.4.2. Dynamic nets $(T, \partial_0, \partial_1, \mu_0)$ and $(T', \partial_0', \partial_1', \mu_0')$ are isomorphic if, up to α-conversion, there exists a pair $(f_t \colon T \to T', f_p \colon \omega \to \omega)$ of isomorphisms such that $(id_{\mathcal{N}m} + f_p id_{\mathcal{N}m} + f_p)(\mu_0) = \mu_0'$, $\partial_0' \circ f_t = (id_{\mathcal{N}m} + f_p id_{\mathcal{N}m}) \circ \partial_0$, and, recursively, $\partial_1(t)$ and $\partial_1'(f_t(t))$ are isomorphic for each $t \in T$.

Example 3.4.3. For $P = \texttt{def}\ x\langle v\rangle \triangleright (\texttt{def}\ y\langle\bullet\rangle \triangleright x\langle v\rangle\ \texttt{in}\ y\langle\bullet\rangle)\ \texttt{in}\ x\langle a\rangle \mid x\langle b\rangle \mid x\langle c\rangle$, the structure $[\![P]\!]$ is the dynamic net illustrated below together with one of its possible firing sequences. The transition may fire three times, binding v in turn to a, b, and c, and spawning three new nets instantiating v appropriately in N.

The free name x of N maintains its meaning across instantiations. This rules are formalized below, and amounts to a recursive extension of binding.



Definition 3.4.4. Let $\mathtt{b}$ be a binding. For $X = (T, \partial_0, \partial_1, \mu_0) \in DNets$, let $X\langle\!\langle\!\langle \mathtt{b} \rangle\!\rangle\!\rangle$ denote the dynamic net obtained from X by substituting free names according to $\mathtt{b}$ in μ_0, $\partial_0(t)$ and, recursively, applying $\langle\!\langle\!\langle \mathtt{b} \rangle\!\rangle\!\rangle$ to $\partial_1(t)$, for all $t \in T$.

The firing rule for dynamic nets is as follows, where $\mathtt{b}_t$ is a binding for t.

$$N \otimes \partial_0(t)\langle \mathtt{b}_t \rangle \; [t\rangle \; N \otimes \partial_1(t)\langle\!\langle\!\langle \mathtt{b}_t \rangle\!\rangle\!\rangle$$

The Type System Δ_3

In order to capture dynamic nets in terms of the join calculus, we simply have to relax the restriction that disallows definitions inside definitions. At the level of type system, this can be achieved by removing the distinction between types $\diamond$ and $\diamond\diamond$ from Δ_2. As an obvious consequence, any join term results typeable.

Proposition 3.4.5. *For each join term P, $\Delta_3 \vdash P$.*

Thus, extending our results for $[\![\_]\!]$ to Δ_3 amounts to proving that dynamic nets coincide with join terms, which is the content of the following.

Theorem 3.4.6 (Δ_3 vs Π_3). *The translation $[\![\_]\!]$ is an isomorphism between join terms (terms typeable in Δ_3) and dynamic nets in Π_3. Moreover, if P and Q are typeable in Δ_3, then*

$$\begin{array}{rll} P \equiv Q & \textit{if and only if} & [\![P]\!] = [\![Q]\!] \\ P{\Downarrow}x & \textit{if and only if} & [\![P]\!]{\Downarrow}x, \quad \textit{for all } x \in \mathcal{N}m. \end{array}$$

3.5 The Type System Δ_4

The information contained in Δ_i, $i = 0, \ldots, 3$ can be summarized in a single type system Δ_4 that tags terms with indices that characterize them as nets of Π_i. Types range over $i^{\diamond\diamond} \in \{0^{\diamond\diamond}, 1^{\diamond\diamond}, 2^{\diamond\diamond}, 3^{\diamond\diamond}\}$ and $i^{\diamond} \in \{0^{\diamond}, 1^{\diamond}, 2^{\diamond}, 3^{\diamond}\}$. For $\tau = i^{?}$ a type, we write $\tau\!\downarrow$ for i and $\tau\!\uparrow$ for ?. By $\tau < \tau'$ we mean that $\tau\!\downarrow < \tau'\!\downarrow$ and $\tau\!\uparrow \leq \tau'\!\uparrow$, with the convention that $\diamond\diamond < \diamond$. Type environments are pairs $\Gamma; \nabla$ as for Δ_1, and type judgments are as follows:

$$\begin{array}{ll}
\Gamma;\nabla \vdash P : i^{\diamond\diamond} & (P \text{ well-typed, in } \Pi_i, \text{ and containing no } \mathtt{def}\ _\ \mathtt{in}\ _) \\
\Gamma;\nabla \vdash P : i^{\diamond} & (P \text{ well-typed and in } \Pi_i) \\
\Gamma;\nabla \vdash D : i^{\diamond} & (D \text{ well-typed and containing terms in } \Pi_i)
\end{array}$$

The typing rules are the following.

(P-MESS$_0$) $\Gamma;\nabla \vdash x\langle\bullet\rangle : 0^{\diamond\diamond}$ $(x\notin\nabla)$

(P-MESS$_1$) $\Gamma;\nabla \vdash x\langle y\rangle : 1^{\diamond\diamond}$ $(x\notin\nabla, y\notin\Gamma)$

(P-MESS$_2$) $\Gamma;\nabla \vdash x\langle y\rangle : 2^{\diamond\diamond}$

(P-DEF)
$$\frac{\Gamma,\mathrm{dn}(D);\nabla \vdash D : i^{\diamond} \quad \Gamma,\mathrm{dn}(D);\nabla \vdash P : i^{\diamond}}{\Gamma;\nabla \vdash \mathtt{def}\ D\ \mathtt{in}\ P : i^{\diamond}}$$

(P-ZERO)
$$\Gamma;\nabla \vdash 0 : 0^{\diamond\diamond}$$

(P-PAR)
$$\frac{\Gamma;\nabla \vdash P : \tau \quad \Gamma;\nabla \vdash Q : \tau}{\Gamma;\nabla \vdash P \mid Q : \tau}$$

(P-SUB)
$$\frac{\Gamma;\nabla \vdash P : \tau \quad \tau < \tau'}{\Gamma;\nabla \vdash P : \tau'}$$

(D-PATT$_0$)
$$\frac{\Gamma;\nabla \vdash P : 0^{\diamond\diamond}}{\Gamma;\nabla \vdash J \triangleright P : 0^{\diamond}}\ (\mathrm{rn}(J)=\{\bullet\})$$

(D-PATT$_1$)
$$\frac{\Gamma;\nabla,\mathrm{rn}(J) \vdash P : i^{\diamond\diamond}}{\Gamma;\nabla \vdash J \triangleright P : i^{\diamond}}$$

(D-PATT$_2$)
$$\frac{\Gamma;\nabla,\mathrm{rn}(J) \vdash P : i^{\diamond}}{\Gamma;\nabla \vdash J \triangleright P : 3^{\diamond}}$$

(D-AND)
$$\frac{\Gamma;\nabla \vdash D : i^{\diamond} \quad \Gamma;\nabla \vdash E : i^{\diamond}}{\Gamma;\nabla \vdash D \wedge E : i^{\diamond}}$$

(D-SUB)
$$\frac{\Gamma;\nabla \vdash D : i^{\diamond} \quad i < j}{\Gamma;\nabla \vdash D : j^{\diamond}}$$

Here (P-MESS) is split into three rules, each behaving as in the corresponding Δ_i; (P-MESS$_2$) ignores type environments, so achieving the effect of Δ_2. Definitions can be typed by i if the terms they contain are typeable by i; (D-PATT$_2$) constrains to 3 the type of definitions containing a `def _ in _` term, so forcing to 3 the type of enclosing processes by means of rule (P-DEF). For Δ_4 we have the following results.

Proposition 3.5.1 (Subject Reduction). *If $\Delta_4 \vdash P$ and $P \longrightarrow Q$, then $\Delta_4 \vdash Q$.*

Theorem 3.5.2. $\Delta_4 \vdash P : i^{\diamond}$ *if and only if* $P \in \Pi_i$, *for* $i = 0, \ldots, 3$.

Proof. It follows by proving that $\Delta_4 \vdash P : i^{\diamond}$ if and only if $\Delta_i \vdash P$. $\square$

Chapter 4

A Coalgebraic Model for Process Calculi

Coalgebraic frameworks for process calculi present several advantages: i.e., morphisms between coalgebras (cohomomorphisms) enjoy the property of "reflecting behaviors" and thus they allow, for example, to characterize bisimulation equivalences as kernels of morphisms and bisimilarity as the bisimulation associated to the morphism to the final coalgebra.

However, in the above representation, the states of transition systems are seen as just forming a set, i.e., the algebraic structure modeling the construction of programs and the composition of states is disregarded. The missing structure may be recovered by integrating coalgebras with algebras, thus gaining *bialgebras* and prove that for them bisimilarity is a congruence.

The goal of this chapter is to propose a novel approach for lifting process calculi with structural axioms to coalgebraic models, and to provide a compositional coalgebraic semantics for the π-calculus by applying this general result to the calculus.

First, we consider a category $\mathbf{Alg}(\Sigma)$ of algebras where specification Σ does not contain axioms. Thus the powerset functor can be lifted to a functor on $\mathbf{Alg}(\Sigma)$ using the De Simone specification. Then, we prove that, given a complete axiomatization (with auxiliary operators) of the algebra of a calculus with parallel composition and restriction, if each axiom bisimulates then the lifting can take place and thus, in particular, bisimilarity is a congruence. Note that the construction in [22] yields a category of bialgebras which satisfy the axioms, while in our case the algebraic specification is just a signature, without axioms. Thus our final coalgebra is larger. However, given a coalgebra which satisfies the axioms, its image in the final coalgebra is the same.

The restriction operator of specification Σ has no name argument. Also extrusion and fresh input actions have no bound names. The reason is that the extruded, fresh or restricted name is assumed to be always the first one, i.e., x_0. To model name extrusion, we define a first order operator δ à la De Brujin, that shifts any name to the successor, thus generating a new name in x_0. The advantage of this choice is that it does not need a notion of support, i.e., the set of names effectively present, which would yield a second order model. Similarly, Pitts's nominal logics [64] exploits a first order model, by relying on substitution algebras that do not need considering support names.

In the π-calculus, bisimilarity fails to be a congruence due to the input prefix. Thus, in order for this property to hold, our algebra structure will include operators of name permutation, parallel composition and restriction, but not prefix. First, we define *static* agents as π-agents whose outmost operation is either prefix, matching, recursion, or summation. We also define a bijective translation of π-agents into terms of our algebra. This function, in particular, maps static agents to constants in the algebra. Then we equip each constant with a set of reduction rules that mimic any possible transition the corresponding static agent can perform. Finally we prove that, for each π-agent, the number of static agents and associated reduction rules needed in all derivations of the agent is finite.

Here our goal is to provide a compositional coalgebraic semantics of the π-calculus. We will define a permutation algebra with parallel composition and restriction for the calculus and, then, we will prove that the labeled transition system of the π-calculus can be lifted to such an algebra, thus obtaining a bialgebra.

A similar task has been considered in [55, 56]. There, a coalgebraic semantics of the π-calculus has been proposed, based on name permutations. The intuition is that the effects of permutations on the behavior of agents are the smallest information required to define an observational equivalence via ordinary bisimilarity, without restrictions due to name generation and passing. However, the proposed model is flat, i.e., it represents the calculus at an operational level, but it does not capture its algebraic structure. Rather, in the present paper we are mainly interested in a compositional interpretation of the π-calculus.

A fully satisfactory coalgebraic theory for the π-calculus takes advantage of well understood verification techniques and of efficient algorithms for computing finite minimal realizations [70, 63, 27].

In the second part of the chapter, we apply the lifting result to the π-calculus. It is well known that, in the π-calculus, bisimilarity fails to be a congruence due to the input prefix. Thus, in order for this property to hold, we construct an algebra for the calculus featuring operators of name permutation, parallel composition and restriction, but not prefix.

First, we define *static* agents as π-agents whose outmost operation is either prefix, matching, recursion, or summation. We also define a bijective translation of π-agents into terms of our algebra. This function, in particular, maps static agents to constants in the algebra. Then we equip each constant with a set of reduction rules that mimic any possible transition the corresponding static agent can perform. Finally we prove that, for each π-agent, the number of static agents and associated reduction rules needed in all derivations of the agent is finite.

In [30], Fiore and Turi propose a semantic framework for name-passing process calculi. Their model is based on functor categories equipped with a differentiation functor δ modeling the introduction of an extra variable – the variable to be bound. Instead we work with permutation algebras enriched with the name extrusion operation δ. While we also rely on a functor category (when the algebraic specification is seen as a Lawvere theory), our setting is more uniform, since δ is at the same level of permutations and of any further operation (here parallel composition) added to the algebra. Another difference is that we deal with name permutations, rather than name substitutions, and, thus, we are able to define observational equivalence (rather than congruence) in a coalgebraic way. A difference with respect to [64] and [35] is that we assume a fixed ordering on names, while they do not need it.

Structure of the chapter In Sect. 4.1 we prove a general result to lift coalgebras in **Set** with structural axioms to coagebras on $\mathbf{Alg}(\Sigma)$. In Sect. 4.2 we define a transition specification and a category of coalgebras for mobile calculi, and in Sect. 4.3 we present a permutation algebra for the π-calculus satisfying the specification. In Sect. 4.3.3 we provide a compositional coalgebraic semantics for the π-calculus, by applying the general result of Sect. 4.1 to the transition system of the calculus.

4.1 Lifting Coalgebras on **Set** to Bialgebras

In this section we prove a general result about lifting a coalgebra g in **Set** with set of states B and equipped with operations Σ, auxiliary constants C, structural axioms E and De Simone rules R to a coalgebra in $\mathbf{Alg}(\Sigma)$. The lifting allows to apply Corollary 2.4.9 and, in particular, to prove that bisimilarity is a congruence.

Theorem 4.1.1. *Let $\mathcal{B}$ be the class of coalgebras g in* **Set** *with the following properties:*

1. $g : |B| \to P_L(|B|)$, *with* $B = T_{(\Sigma \cup C, E)}$.
2. lts_g *satisfies transition specification* $\Delta = (\Sigma, L, R)$, *with R in De Simone format.*

3. *A set of basic transitions* $T \subseteq C \times L \times T_{\Sigma \cup C}$ *exists for constants* C*, namely,* $(c, l, t) \in T$ *implies* $c \xrightarrow{l}_g [t]_E$.

Then, there is an initial coalgebra $\hat{g}$ *in* $\mathcal{B}$*, such that* $\forall g \in \mathcal{B}$*,* $\forall p \in B$*,* $p \xrightarrow{l}_{\hat{g}} q$ *implies* $p \xrightarrow{l}_g q$.

Furthermore, the transitions of $\hat{g}$ *can be derived using the rules* R *and the following additional rules:*

$$\text{(CONST)}\ \frac{(c,l,t) \in T}{c \xrightarrow{l}_{\hat{g}} t} \qquad \text{(STRUCT)}\ \frac{t_1 =_E t_1' \quad t_1' \xrightarrow{l}_{\hat{g}} t_2' \quad t_2' =_E t_2}{t_1 \xrightarrow{l}_{\hat{g}} t_2}$$

where terms t, t_1, t_1', t_2, t_2' *are in* $T_{\Sigma \cup C}$.

Proof. Signature $\Sigma \cup C$, axioms E and transition specification Δ can be considered as a single algebraic specification, equipped with an initial model. The elements of the model can be derived using the proof system associated to the specification. □

Remark 4.1.2. Let A be a Σ-algebra and $h : X \to |A|$ be a function in **Set**. Then, h can be uniquely extended to $\hat{h} : T_\Sigma(X) \to A$ in $\mathbf{Alg}(\Sigma)$. For simplicity, in the sequel, we will also denote $\hat{h}$ by h.

Definition 4.1.3. *Let* $g : |B| \to P_L(|B|)$ *be the initial coalgebra of Theorem 4.1.1, where, however, constants* C *are considered as auxiliary, i.e.,* B *is seen as a* Σ*-algebra. Then, we define the following* Σ*-algebras and* Σ*-morphisms:*

- $A = T_\Sigma(C)$ *and* $h : A \to B$ *as the unique extension in* $\mathbf{Alg}(\Sigma)$ *of* $h(c) = [c]_E$*, for* $c \in C$*;*
- $f : A \to P_\Delta(A)$ *as the unique extension of* $f(c) = \{(l,t) \mid (c,l,t) \in T\}$ *in* $\mathbf{Alg}(\Sigma)$.

In the sequel, we want to find conditions under which P_L-coalgebra g can be lifted to a P_Δ-coalgebra and function h, as above defined, to a P_Δ-morphism. The observation in Subsect. 2.4 allows us to state, without any further condition, that h is a lax morphism between P_L-coalgebras f and g. The reflection inclusion, instead, will require appropriate hypotheses.

Property 1. Function h in Definition 4.1.3 is a lts morphism, namely, a lax P_L-coalgebra morphism. Furthermore, it is surjective.

Proof. Immediate, as every proof of a transition in lts_f holds also in lts_g. □

Theorem 4.1.4. *Let g be the initial coalgebra in $\mathcal{B}$ as specified by Theorem 4.1.1, and let A, h, and f be defined as in Definition 4.1.3. Then, h is surjective. Let us assume that for all equations $t_1 = t_2$ in E, with free variables $\{x_i\}_{i\in I}$, we have De Simone proofs as follows:*

$$\frac{x_i \xrightarrow{l_i} y_i \quad i \in I}{t_1 \xrightarrow{l} t_1'} \quad \textit{implies} \quad \frac{x_i \xrightarrow{l_i} y_i \quad i \in I}{t_2 \xrightarrow{l} t_2'} \quad \textit{and} \quad t_1' =_E t_2' \tag{4.1}$$

and viceversa, using the rules in R and the additional rules:

$$c \xrightarrow{l} t \quad \textit{iff} \quad (c, l, t) \in T.$$

Then, the left diagram below commutes in **Set**, *i.e., $h; g = f; P_L(h)$. Thus, h is a P_L-morphism.*

Proof. We start noticing that $h : A \to B$ is surjective since $B = A_{/=_E}$ and $=_E$ is the kernel of h.

Then, we first prove that the equivalence relation $=_E$ is a bisimulation for lts_f. We use rule induction on the proofs of $=_E$. For axioms in E , the property is guaranteed by Condition 4.1. Rules for reflexivity, symmetry and transitivity are obviously satisfied. Rule for congruence is also easily checked, since if t_i and t_i' bisimulate, for $i = 1, \cdots, n$, also $k(t_1, \cdots, t_n)$ and $k(t_1', \cdots, t_n')$, with $k \in \Sigma_n$, have corresponding transitions by applying the same De Simone rule or the same constant rule.

Since h is a lax P_L-coalgebra morphism, to derive our result it is enough to show that $h(t_1) \xrightarrow{l}_g p$ implies $t_1 \xrightarrow{l}_f t_2$, with $h(t_2) = p$. We prove this property by induction on the rules of lts_g, as specified by Theorem 4.1.1. Rules in R and rule (Const) can be easily checked since they are the same for lts_f and for lts_g. Also, the last rule (Struct) preserves the property. Indeed, given $[t_1']_E \xrightarrow{l}_g [t_2']_E$, i.e., $h(t_1') \xrightarrow{l}_g h(t_2')$, by induction hypothesis we have $t_1' \xrightarrow{l}_f t_2''$, with $h(t_2'') = h(t_2')$. Furthermore, since $t_1 =_E t_1'$ and $=_E$ is a bisimulation for f, we can find $t_1 \xrightarrow{l}_f t_2'''$, with $t_2''' =_E t_2''$ and, thus, $h(t_2''') = h(t_2'')$. Also, $t_2' =_E t_2$ implies $h(t_2) = h(t_2')$. Then, $h(t_2''') = h(t_2)$. $\square$

$$\begin{tikzcd} {|A|} \arrow[r, "h"] \arrow[d, "f"'] & {|B|} \arrow[d, "g"] \\ P_L(|A|) \arrow[r, "P_L(h)"'] & P_L(|B|) \end{tikzcd} \qquad \begin{tikzcd} A \arrow[r, "h"] \arrow[d, "f"'] & B \arrow[d, "g"] \\ P_\Delta(A) \arrow[r, "P_\Delta(h)"'] & P_\Delta(B) \end{tikzcd}$$

Theorem 4.1.5. *Let g be the initial coalgebra in $\mathcal{B}$ as specified by Theorem 4.1.1, and let A, h, and f be defined as in Definition 4.1.3. If the left diagram above commutes in* **Set**, *i.e., $h; g = f; P_L(h)$, then g can be lifted from* **Set** *to* $\mathbf{Alg}(\Sigma)$ *and the right diagram commutes in* $\mathbf{Alg}(\Sigma)$.

Proof. By construction, f, h, and $P_\Delta(h)$ are morphisms in $\mathbf{Alg}(\Sigma)$. Then, we have to prove that g is a morphism, i.e., $g(op^B(p_1, \ldots, p_n)) = op^{P_\Delta(B)}(g(p_1), \ldots, g(p_n))$.

Since h is surjective, there exist $t_1, \ldots, t_n$ such that

$$\begin{aligned}
g(op^B(p_1,\ldots,p_n)) &= g(op^B(h(t_1),\ldots,h(t_n))) \\
&= g(h(op^A(t_1,\ldots,t_n))) && (h \text{ morphism}) \\
&= P_\Delta(h)(f(op^A(t_1,\ldots,t_n))) && (h; g = f; P_L(h)) \\
&= op^{P_\Delta(B)}(P_\Delta(h)(f(t_1)),\ldots,P_\Delta(h)(f(t_n))) && (f; P_\Delta(h) \text{ morphism}) \\
&= op^{P_\Delta(B)}(g(h(t_1)),\ldots,g(h(t_n))) && (h; g = f; P_L(h)) \\
&= op^{P_\Delta(B)}(g(p_1),\ldots,g(p_n))
\end{aligned}$$

□

Corollary 4.1.6. *Let g be the initial coalgebra in $\mathcal{B}$ as specified by Theorem 4.1.1, and let the right diagram above commute. Then in g bisimilarity is a congruence.*

Proof. The claim follows by Theorem 4.1.5 and Corollary 2.4.9. □

The rest of the chapter is devoted to apply the present general result we have proved so far to the π-calculus.

4.2 A Category of Coalgebras for Mobile Calculi

We first define a transition specification Δ_{pr} for mobile calculi with permutations (ρ), parallel composition ($|$), name restriction (ν) and extrusion (δ), and prove that the category $\mathbf{Coalg}(P_{\Delta_{pr}})$ of coalgebras over Δ_{pr} is well-defined. As mentioned in the Introduction, operators ν and δ do not have arguments, as the extruded or restricted name is assumed to be always the first one, i.e., x_0.

Definition 4.2.1. *Signature Σ_{pr} is defined as follows:*

$$\Sigma_{pr} \ ::= \ \mathbf{0} \ \big| \ \_|\_ \ \big| \ \nu.\,\_ \ \big| \ \delta.\,\_ \ \big| \ \rho\_$$

We adopt the convention that operators have decreasing binding power, in the following order: ρ, $|$, ν and δ. Thus, for example, $\nu.\,\delta.\,\rho p|q$ means $\nu.\,(\delta.\,((\rho p)|q))$.

Operators ρ are generic, finite name permutations, as described in Subsect. 2.1. Operator δ is meant to represent the substitution $[x_i \mapsto x_{i+1}]$, for $i = 0, 1, \ldots$. Of course, this substitution is not finite, but, at least in the case of an ordinary agent p, it replaces a finite number of names, i.e., the free names of p.

We define the set L_{pr} of the labels:

$$L_{pr} = \{\tau,\ x\,y,\ x,\ \overline{x}\,y,\ \overline{x} \mid x, y \in \mathfrak{N}\}. \tag{4.2}$$

If $l \in L_{pr}$ then $\delta(l)$ and $\nu(l)$ are the labels obtained from l by respectively applying substitution δ and ν to its names, where $\nu(x_{i+1}) = x_i$ and $\delta(x_i) = x_{i+1}$. Homomorphically, δ and ν are extended to π-agents, with δ and ν inactive on bound names. Note that in $\nu.\, p$, p is a value of a Σ_{pr}-algebra; in $\nu(p)$, p is a π-agent and $\nu(p)$ is defined only if $x_0 \notin \mathrm{fn}(p)$.

The correspondence between labels L_{pr} and the actions of the π-calculus is the obvious one for τ, $x\,y$, and $\overline{x}\,y$. In the case of bound output transitions, only the channel x on which the output occurs is observed in label $\overline{x}$, and similarly in the case of bound input transitions.

Definition 4.2.2 (transition specification Δ_{pr}). *The transition specification Δ_{pr} is the tuple $\langle \Sigma_{pr}, L_{pr}, R_{pr} \rangle$, where the signature Σ_{pr} is as in Definition 4.2.1, labels L_{pr} are defined in (4.2) and the rules R_{pr} are the SOS rules in Table 4.1.*

The most interesting rules are those in the second column, where α is an extrusion or the input of a fresh name. Intuitively, they follow the idea that substitutions on the source of a transition must be

reflected on its destination by restoring the extruded or fresh name to x_0. Rule (DELTA′) applies δ to q and then permutes x_0 and x_1, in order to have the extruded name back to x_0. Conversely, rule (RES′) permutes x_0 and x_1 to make sure that the restriction operation applies to x_0 and not to the extruded name x_1. In rule (PAR′) side condition $\mathrm{bn}(\alpha) \cap \mathrm{fn}(r) = \varnothing$, namely $x_0 \notin \mathrm{fn}(\delta.\, r)$, is *not necessary*. The intuitive reason is that x_0 does not appear in $\delta.\, r$, since δ shifts all the variables to the right.

(RHO) $\dfrac{p \xRightarrow{\alpha} q \quad \alpha \neq \overline{x}, x}{\rho p \xRightarrow{\rho(\alpha)} \rho q}$	(RHO′) $\dfrac{p \xRightarrow{\alpha} q \quad \alpha = \overline{x}, x}{\rho p \xRightarrow{\rho(\alpha)} \rho_{+1} q}$
(PAR) $\dfrac{p \xRightarrow{\alpha} q \quad \alpha \neq \overline{x}, x}{p \vert r \xRightarrow{\alpha} q \vert r}$	(PAR′) $\dfrac{p \xRightarrow{\alpha} q \quad \alpha = \overline{x}, x}{p \vert r \xRightarrow{\alpha} q \vert \delta.\, r}$
(DELTA) $\dfrac{p \xRightarrow{\alpha} q \quad \alpha \neq \overline{x}, x}{\delta.\, p \xRightarrow{\delta(\alpha)} \delta.\, q}$	(DELTA′) $\dfrac{p \xRightarrow{\alpha} q \quad \alpha = \overline{x}, x}{\delta.\, p \xRightarrow{\delta(\alpha)} [x_0 \leftrightarrow x_1]\delta.\, q}$
(RES) $\dfrac{p \xRightarrow{\alpha} q \quad \alpha \neq \overline{x}, x}{\nu.\, p \xRightarrow{\nu(\alpha)} \nu.\, q}$	(RES′) $\dfrac{p \xRightarrow{\alpha} q \quad \alpha = \overline{x}, x}{\nu.\, p \xRightarrow{\nu(\alpha)} \nu.\, [x_0 \leftrightarrow x_1] q}$
(OPEN) $\dfrac{p \xRightarrow{\overline{x}\, x_0} q \quad x \neq x_0}{\nu.\, p \xRightarrow{\nu(\overline{x})} q}$	(CLOSE) $\dfrac{p_1 \xRightarrow{\overline{x}} q_1 \quad p_2 \xRightarrow{x} q_2}{p_1 \vert p_2 \xRightarrow{\tau} \nu.\, q_1 \vert q_2}$
(COM) $\dfrac{p_1 \xRightarrow{\overline{x}\, y} q_1 \quad p_2 \xRightarrow{x\, y} q_2}{p_1 \vert p_2 \xRightarrow{\tau} q_1 \vert q_2}$	

Table 4.1. Structural operational semantics.

Proposition 4.2.3. *Let $\Delta_{pr} = \langle \Sigma_{pr}, L_{pr}, R_{pr} \rangle$ be the transition specification in Definition 4.2.2. Rules R_{pr} are in De Simone format.*

Note that the rules of the π-calculus in Table 2.1 are not in De Simone format: the side condition on $\mathrm{fn}(r)$ prevents rule (π-PAR) from that.

Theorem 4.2.4. *Let $\Delta_{pr} = \langle \Sigma_{pr}, L_{pr}, R_{pr} \rangle$ be the transition specification in Definition 4.2.2. Then, functor $P_{L_{pr}}$ in **Set** can be lifted to functor $P_{\Delta_{pr}}$ in $\mathbf{Alg}(\Sigma_{pr})$ and category $\mathbf{Coalg}(P_{\Delta_{pr}})$ is well defined.*

Proof. The claim follows by Proposition 4.2.3 and Proposition 2.4.8. □

4.3 A Σ_{pr}-Algebra for the π-Calculus

In this section we introduce a Σ_{pr}-algebra for the π-calculus, and we prove a finiteness result. The version of the π-calculus we consider is early, monadic, and with synchronous communications (see Section 2.2).

4.3.1 A Σ_{pr}-Algebra

We first define a Σ_{pr}-algebra and a bijective translation of π-agents to values of such an algebra.

Definition 4.3.1 (Σ_{pr}-algebra for the π-calculus). *Algebra B is defined as the initial algebra $B = T_{(\Sigma_{pr} \cup L, E_{pr} \cup E')}$ where:*

- *Σ_{pr} is as in Definition 4.2.1,*
- *constants L are*

$$L = \{ l_s \mid s \text{ is a static } \pi\text{-agent} \}$$

- *axioms E_{pr} are the axioms below:*

$$\begin{array}{lllll}
(\mathbf{perm}) & (\rho \cdot \rho')p = \rho(\rho'(p)) & \mathrm{id}\, p = p & & \\
(\mathbf{par}) & p|\mathbf{0} = p & p|q = q|p & p|(q|r) = (p|q)|r & \\
(\mathbf{res}) & \nu.\, \mathbf{0} = \mathbf{0} & \nu.\, (\delta.\, p)|q = p|\nu.\, q & \nu.\, \nu.\, [x_0 \leftrightarrow x_1]p = \nu.\, \nu.\, p & \\
(\mathbf{delta}) & \delta.\, \mathbf{0} = \mathbf{0} & \delta.\, p \mid q = (\delta.\, p) \mid \delta.\, q & \delta.\, \nu.\, p = \nu.\, [x_0 \leftrightarrow x_1]\delta.\, p & \\
(\mathbf{rho}) & \rho\, \mathbf{0} = \mathbf{0} & \rho(p|q) = \rho p \mid \rho q & \rho\nu.\, p = \nu.\, \rho_{+1} p & \delta.\, \rho p = \rho_{+1}\delta.\, p
\end{array}$$

- *axioms E' are*

$$\rho l_s = l_{\rho(s)} \qquad \delta.\, l_s = l_{\delta\,(s)}.$$

Axioms (**par**) and (**res**) correspond to the analogous axioms for the π-calculus. The other axioms rule how to invert the order of operators among each other, following the intuition that ν and δ decrease and increase variable indexes respectively. Axioms E_{pr} and E' can be applied from left to right to reduce every term p into a canonical form $p'(l_{s_1}, \ldots, l_{s_n})$, such that ρ and δ do not occur in context $p'(\_, \_, \ldots)$. Notice that other expected properties like $\nu.\, \delta.\, p = p$ and $[x_0 \leftrightarrow x_1]\delta.\, \delta.\, p = \delta.\, \delta.\, p$ can be derived from these axioms.

Definition 4.3.2 (translation $[\![\cdot]\!]$). *We define a translation on π-calculus agent terms $[\![\cdot]\!] : \Pi \to |B|$ as follows:*

$$[\![\mathbf{0}]\!] = \mathbf{0} \quad [\![p|q]\!] = [\![p]\!]|[\![q]\!] \quad [\![(\boldsymbol{\nu} y)\, p]\!] = \nu.\, [\delta\,(y) \leftrightarrow x_0]\delta.\, [\![p]\!] \quad [\![s]\!] = l_s.$$

The translation of the restriction gives the flavor of the De Brujin notation: the idea is to split standard restriction in three steps. First, one shifts all names upwards to generate a fresh name x_0, then swaps $\delta\,(y)$ and x_0, and, finally, applies restriction on x_0, which now stands for what 'used to be' y.

Translation $[\![\cdot]\!]$ is also defined on actions as follows: $[\![\alpha]\!] = \alpha$, if $\alpha \neq \overline{x}\,(y), x\,(y)$; $[\![\overline{x}\,(y)]\!] = \overline{x}$; $[\![x\,(y)]\!] = x$.

Theorem 4.3.3. *If $p \equiv q$ then $[\![p]\!] = [\![q]\!]$.*

Proof. The proof is by structural induction on the axioms of the structural congruence $\equiv$.

First, note that $[\delta\,(x) \leftrightarrow x_0]\delta.\, [\![r]\!] = \delta.\, [\![r]\!]$, if $x \notin \mathrm{fn}(r)$. Indeed, it is easily proved by first applying axioms E' and then substitution $[\delta\,(x) \leftrightarrow x_0]$ to the π-agent $\delta\,(p)$.

The most interesting cases of the proof concern alpha equivalent non-static agents and restriction.

Ax. $(\boldsymbol{\nu} x)\, p \equiv (\boldsymbol{\nu} y)\, [y \mapsto x]p,\ y \notin \mathrm{fn}(p)$.

By the condition, $(\boldsymbol{\nu} y)\, [y \mapsto x]p = (\boldsymbol{\nu} y)\, [y \leftrightarrow x]p$. $[\![(\boldsymbol{\nu} y)\, [x \leftrightarrow y]p]\!] = \nu.\, [\delta\,(y) \leftrightarrow x_0]\delta.\, [\![[x \leftrightarrow y]p]\!] = \nu.\, [\![[\delta\,(y) \leftrightarrow x_0][\delta\,(x) \leftrightarrow \delta\,(y)]\delta\,(p)]\!] = \nu.\, [\![[\delta\,(x) \leftrightarrow x_0]\delta\,(p)]\!] = [\![(\boldsymbol{\nu} x)\, p]\!]$.

Ax. $(\boldsymbol{\nu} x)\, (\boldsymbol{\nu} y)\, p \equiv (\boldsymbol{\nu} y)\, (\boldsymbol{\nu} x)\, p$.

First, $[\![(\boldsymbol{\nu} x)\, (\boldsymbol{\nu} y)\, p]\!] = \nu.\, \nu.\, \rho_1 \delta\delta[\![p]\!]$ and $[\![(\boldsymbol{\nu} y)\, (\boldsymbol{\nu} x)\, p]\!] = \nu.\, \nu.\, \rho_2 \delta\delta[\![p]\!]$, for some ρ_1 and ρ_2 that differ for a permutation $[x_0 \leftrightarrow x_1]$. Then, $[\![(\boldsymbol{\nu} x)\, (\boldsymbol{\nu} y)\, p]\!] = [\![(\boldsymbol{\nu} y)\, (\boldsymbol{\nu} x)\, p]\!]$, by axiom $\nu.\, \nu.\, q = \nu.\, \nu.\, [x_0 \leftrightarrow x_1]q$.

Ax. $(\boldsymbol{\nu} x)\, (p|q) \equiv p|(\boldsymbol{\nu} x)\, q$ if $x \notin \mathrm{fn}(p)$.

$[\![(\boldsymbol{\nu} x)\, (p|q)]\!] = \ldots = \nu.\, ([\delta\,(x) \leftrightarrow x_0]\delta.\, [\![p]\!]\,|[\delta\,(x) \leftrightarrow x_0]\delta.\, [\![q]\!])$. Since $x \notin \mathrm{fn}(p)$, $[\delta\,(x) \leftrightarrow x_0]\delta.\, [\![p]\!] = \delta.\, [\![p]\!]$. Then, $[\![(\boldsymbol{\nu} x)\, (p|q)]\!] = \nu.\, (\delta.\, [\![p]\!]|[\delta\,(x) \leftrightarrow x_0]\delta.\, [\![q]\!]) = [\![p]\!]|\nu.\, [\delta\,(x) \leftrightarrow x_0]\delta.\, [\![q]\!] = [\![p|(\boldsymbol{\nu} x)\, q]\!]$.

Ax. $(\boldsymbol{\nu} x)\, p \equiv p$, if $x \notin \mathrm{fn}(p)$.

It follows by the condition, by the observation on top of the proof, and by axiom $\nu.\,(\delta.\, p|q) = p|\nu.\, q$, with $q = 0$.

$\square$

Theorem 4.3.4. *Function $[\![\cdot]\!]$ is bijective. I.e., let $\{[\cdot]\} : |B| \rightarrow \Pi$ be a translation defined as follows: $\{[\mathbf{0}]\} = \mathbf{0}$; $\{[p|q]\} = \{[p]\}|\{[q]\}$; $\{[\nu.\, p]\} = (\boldsymbol{\nu} x_i)\, \nu\, ([\delta\,(x_i) \leftrightarrow x_0]\{[p]\})$, if $\delta\,(x_i) \notin \mathrm{fn}(\{[p]\})$; $\{[l_s]\} = s$; $\{[\rho p]\} = \rho(\{[p]\})$; $\{[\delta.\, p]\} = \delta\,(\{[p]\})$. Then, for every p in B, $[\![\{[p]\}]\!] = p$ and, for every π-agent q, $\{[[\![q]\!]]\} = q$.*

Proof. We first prove that, for $q \in B$, $[\![\{[q]\}]\!] = q$. We just consider the case $q = \nu.\, p$, which is the most interesting case.
$[\![\{[\nu.\, p]\}]\!] = [\![(\boldsymbol{\nu} x_i)\, \nu\, ([\delta\,(x_i) \leftrightarrow x_0]\{[p]\})]\!] = \nu.\, [\delta\,(x_i) \leftrightarrow x_0]\delta.\, [\![\nu\, ([\delta\,(x_i) \leftrightarrow x_0]\{[p]\})]\!] = \nu.\, [\![[\delta\,(x_i) \leftrightarrow x_0]\delta\, (\nu\, ([\delta\,(x_i) \leftrightarrow x_0]\{[p]\}))]\!]$. Since $[\delta\,(x_i) \leftrightarrow x_0]\delta\, (\nu\, ([\delta\,(x_i) \leftrightarrow x_0]\{[p]\})) = \{[p]\}$ and $[\![\{[p]\}]\!] = p$, then $[\![\{[\nu.\, p]\}]\!] = \nu.\, p$. Note that the condition is necessary for the application of ν. We now prove that, for a π-calculus agent r, $\{[[\![r]\!]]\} = r$. We just consider the case $r = (\boldsymbol{\nu} x)\, p$, for some x.
$\{[[\![(\boldsymbol{\nu} x)\, p]\!]]\} = (\boldsymbol{\nu} x)\, \nu\, ([\delta\,(x) \leftrightarrow x_0]\{[[\delta\,(x) \leftrightarrow x_0]\delta.\, [\![p]\!]]\})$, since $x \notin \mathrm{fn}([\delta\,(x) \leftrightarrow x_0]\delta.\, [\![p]\!])$. Then, $\{[[\![(\boldsymbol{\nu} x)\, p]\!]]\} = \ldots = (\boldsymbol{\nu} x)\, \{[\nu.\, \delta.\, [\![p]\!]]\} = (\boldsymbol{\nu} x)\, p$, by axiom $\nu.\, (\delta.\, p)|q = p|\nu.\, q$, with $q = \mathbf{0}$, and by induction hypothesis. $\square$

Definition 4.3.5 (transition system lts_g). *The transition system for algebra B is $lts_g = \langle B, \Longrightarrow_g \rangle$, where $\Longrightarrow_g$ is defined by the SOS rules in Table 4.1 plus the following axioms:*

$$\text{(STATIC)}\ \frac{s \xrightarrow{\alpha} r \quad \alpha \neq \overline{x}\,(y), x\,(y)}{l_s \xRightarrow{[\![\alpha]\!]}_g [\![r]\!]} \qquad \text{(STATIC}'\text{)}\ \frac{s \xrightarrow{\alpha} r \quad \alpha = \overline{x}\,(y), x\,(y)}{l_s \xRightarrow{[\![\alpha]\!]}_g [\delta\,(y) \leftrightarrow x_0]\delta.\, [\![r]\!]}$$

When no confusion arises, we will simply denote lts_g with lts and $\Longrightarrow_g$ with $\Longrightarrow$.

Theorem 4.3.6. *Transition system lts satisfies the specification Δ_{pr} in Def. 4.2.2.*

Proof. Trivial, as *lts* rules include rules R_{pr}. $\square$

Example 4.3.7. Let us consider again π-agent $p = \texttt{rec}\, X.\, (\boldsymbol{\nu} y)\, \overline{y} x_2.\mathbf{0}|\overline{x}_2 y.X$. By rule ($\pi$-REC) and ($\pi$-OPEN), p can reduce as $p \xrightarrow{\overline{x}_2(x_3)} \overline{x}_3 x_2.\mathbf{0} \mid p$. On the other hand, $[\![p]\!] = l_p$. By rule (STATIC′), l_p can reduce as $l_p \xRightarrow{\overline{x}_2} [\delta\,(x_3) \leftrightarrow x_0]\delta.\, [\![\overline{x}_3 x_2.\mathbf{0} \mid p]\!]$ and $[\delta\,(x_3) \leftrightarrow x_0]\delta.\, [\![\overline{x}_3 x_2.\mathbf{0} \mid p]\!] = [\![\overline{x}_0 x_3.\mathbf{0}]\!] \mid [x_4 \leftrightarrow x_0]\delta.\, l_p$.

At a second step, by rule (π-REC) and (π-OPEN), p can reduce as $p \xrightarrow{\overline{x}_2(x_1)} \overline{x}_1 x_2.\mathbf{0} \mid p$, and by rule ($\pi$-PAR) $\overline{x}_3 x_2.\mathbf{0} \mid p \xrightarrow{\overline{x}_2(x_1)} \overline{x}_3 x_2.\mathbf{0} \mid \overline{x}_1 x_2.\mathbf{0} \mid p$. On the other hand, $[\![\overline{x}_0 x_3.\mathbf{0}]\!] \mid [x_4 \leftrightarrow x_0]\delta.\, l_p \xRightarrow{\overline{x}_3} [\![\overline{x}_1 x_4.\mathbf{0}]\!] \mid [\![\overline{x}_0 x_4.\mathbf{0}]\!] \mid [\![\texttt{rec}\, X.\, (\boldsymbol{\nu} y)\, \overline{y} x_4.\mathbf{0}|\overline{x}_4 y.X]\!]$. Indeed, by rule (STATIC′), $l_p \xRightarrow{\overline{x}_2} [x_2 \leftrightarrow x_0]\delta.\, [\![\overline{x}_1 x_2.\mathbf{0} \mid p]\!]$ and $[x_2 \leftrightarrow x_0]\delta.\, [\![\overline{x}_1 x_2.\mathbf{0} \mid p]\!] = [\![\overline{x}_0 x_3.\mathbf{0}]\!] \mid [x_2 \leftrightarrow x_0]\delta.\, l_p$. Then, by (DELTA′),

$\delta.\, l_p \stackrel{\overline{x}_3}{\Longrightarrow} [x_0 \leftrightarrow x_1]\delta.\, [\![\overline{x}_0 x_3.\mathbf{0}]\!] \mid \delta.\, l_p = [\![\overline{x}_0 x_4.\mathbf{0}]\!] \mid \delta.\,\delta.\, l_p$. By rule (Rho′), $[x_4 \leftrightarrow x_0]\delta.\, l_p \stackrel{\overline{x}_3}{\Longrightarrow} [x_5 \leftrightarrow x_1][\![\overline{x}_0 x_4.\mathbf{0}]\!] \mid \delta.\,\delta.\, l_p = [\![\overline{x}_0 x_4.\mathbf{0}]\!] \mid \delta.\,\delta.\, l_p$. Finally, by rule (Par′), $[\![\overline{x}_0 x_3.\mathbf{0}]\!] \mid [x_4 \leftrightarrow x_0]\delta.\, l_p \stackrel{\overline{x}_3}{\Longrightarrow} [\![\overline{x}_1 x_4.\mathbf{0}]\!] \mid [\![\overline{x}_0 x_4.\mathbf{0}]\!] \mid \delta.\,\delta.\, l_p$, and $\delta.\,\delta.\, l_p = [\![\mathtt{rec}\, X.\, (\boldsymbol{\nu} y)\, \overline{y} x_4.\mathbf{0} | \overline{x}_4 y.X]\!]$.

The two lemmata below will be useful to prove that the transition system of the π-calculus is equivalent to $lts = \langle B, \Longrightarrow \rangle$.

Lemma 4.3.8. *Let p and q be two π-agents. If $p \sim q$, then $\delta(p) \sim \delta(q)$ and $\rho(p) \sim \rho(q)$.*

Lemma 4.3.9.

1. *Let p and q be two π-agents. If $p \stackrel{\alpha}{\longrightarrow} q$ and $\alpha \neq \overline{x}\,(y), x\,(y)$ then $[\![p]\!] \stackrel{[\![\alpha]\!]}{\Longrightarrow} [\![q]\!]$; if $p \stackrel{\alpha}{\longrightarrow} q$ and $\alpha = \overline{x}\,(y), x\,(y)$ then $[\![p]\!] \stackrel{[\![\alpha]\!]}{\Longrightarrow} [\delta\,(y) \leftrightarrow x_0]\delta.\, [\![q]\!]$.*

2. *Let p and q be in B. If $p \stackrel{\alpha}{\Longrightarrow} q$ and $\alpha \neq \overline{x}, x$, then $\{[p]\} \stackrel{\alpha}{\longrightarrow} \{[q]\}$; if $p \stackrel{\overline{x}}{\Longrightarrow} q$ (resp. $p \stackrel{x}{\Longrightarrow} q$), then $\{[p]\} \stackrel{\overline{x}\,(x_i)}{\longrightarrow} \nu\,([\delta\,(x_i) \leftrightarrow x_0]\{[q]\})$, (resp. $\{[p]\} \stackrel{x\,(x_i)}{\longrightarrow} \nu\,([\delta\,(x_i) \leftrightarrow x_0]\{[q]\})$), for every x_i such that $\delta\,(x_i) \notin \mathrm{fn}(\{[q]\})$.*

Proof. The proof of (1) is by induction on the rules $\rightarrow$. We just consider the most interesting cases.

Rule (π-Res). By the condition, $\alpha \neq \overline{x}\,(y), x\,(y)$. By induction hypothesis, $[\![p]\!] \stackrel{[\![\alpha]\!]}{\Longrightarrow} [\![q]\!]$. Then, $[\delta\,(y) \leftrightarrow x_0]\delta.\, [\![p]\!] \stackrel{\delta\,([\![\alpha]\!])}{\Longrightarrow} [\delta\,(y) \leftrightarrow x_0]\delta.\, [\![q]\!]$, by applying rules $\Rightarrow$ and exploiting that $[\delta\,(y) \leftrightarrow x_0]\delta\,([\![\alpha]\!]) = \delta\,([\![\alpha]\!])$, for $y \notin \mathrm{n}([\![\alpha]\!])$. Since $x_0 \notin \delta\,([\![\alpha]\!])$, $[\![\alpha]\!] \neq \overline{x}, x$, and $\nu\,(\delta\,([\![\alpha]\!])) = [\![\alpha]\!]$, $[\![(\boldsymbol{\nu} y)\, p]\!] \stackrel{[\![\alpha]\!]}{\Longrightarrow} [\![(\boldsymbol{\nu} y)\, q]\!]$.

Rule (π-Close). By hypothesis $[\![p_1]\!] \stackrel{\overline{x}}{\Longrightarrow} [\delta\,(y) \leftrightarrow x_0]\delta.\, [\![q_1]\!]$ and $[\![p_2]\!] \stackrel{x}{\Longrightarrow} [\delta\,(y) \leftrightarrow x_0]\delta.\, [\![q_2]\!]$. Then, $[\![p_1 | p_2]\!] \stackrel{\tau}{\Longrightarrow} \nu.\, [\delta\,(y) \leftrightarrow x_0]\delta.\, [\![q_1 | q_2]\!]$.

Rule (π-Open). By hypothesis, $[\![p]\!] \stackrel{\overline{x}\, y}{\Longrightarrow} [\![q]\!]$. Then, $[\delta\,(y) \leftrightarrow x_0]\delta.\, [\![p]\!] \stackrel{\delta\,(\bar{x}) x_0}{\Longrightarrow} [\delta\,(y) \leftrightarrow x_0]\delta.\, [\![q]\!]$, by applying rules

$\Rightarrow$ and noting that $[\delta\,(y) \leftrightarrow x_0]\delta\,(\overline{x}\, y) = \delta\,(\bar{x}) x_0$. It follows that $\nu.\, [\delta\,(y) \leftrightarrow x_0]\delta.\, [\![p]\!] \stackrel{\overline{x}}{\Longrightarrow} [\delta\,(y) \leftrightarrow x_0]\delta.\, [\![q]\!]$, with $\nu\,(\delta\,(\overline{x})) = \overline{x}$.

Rule (π-Par). If $\alpha \neq \overline{x}\,(y), x\,(y)$, the proof is trivial. Suppose $\alpha = \overline{x}\,(y)$ (the proof is similar if $\alpha = \overline{x}\,(y)$). By induction hypothesis, $[\![p]\!] \stackrel{\overline{x}}{\Longrightarrow} [\delta\,(y) \leftrightarrow x_0]\delta.\, [\![q]\!]$. Then, $[\![p]\!] | [\![r]\!] \stackrel{\overline{x}}{\Longrightarrow} [\delta\,(y) \leftrightarrow x_0](\delta.\, [\![q]\!]) | \delta.\, [\![r]\!]$ and $[\![p | r]\!] \stackrel{\overline{x}}{\Longrightarrow} [\delta\,(y) \leftrightarrow x_0]\delta.\, [\![q | r]\!]$, as $y \notin \mathrm{fn}(r)$.

The proof of (2) is by induction on the rules $\Rightarrow$. We just consider the most interesting cases.

Rule (STATIC′). Suppose $\alpha = \overline{x}$ (proof is similar if $\alpha = x$). We apply the rule with $l_s = p$ and $r = \nu([\delta(y) \leftrightarrow x_0]\{[q]\})$ and exploit that $[\![[\delta(y) \leftrightarrow x_0]\delta(\nu([\delta(y) \leftrightarrow x_0]\{[q]\}))]\!] = [\![\{[q]\}]\!]$.

Rule (PAR′). Let $\alpha = \overline{x}$ (similarly, if $\alpha = x$). By hypothesis, $\{[p]\} \xrightarrow{\overline{x}(x_i)} \nu([\delta(x_i) \leftrightarrow x_0]\{[q]\})$, for every x_i, such that $\delta(x_i) \notin \text{fn}(\{[q]\})$. Under the hypothesis $\delta(x_i) \notin \text{fn}(\{[q]\}|\delta(\{[r]\}))$, $x_i \notin \text{fn}(\{[r]\})$. Thus, $\{[p]\}|\{[r]\} \xrightarrow{\overline{x}(x_i)} \nu([\delta(x_i) \leftrightarrow x_0]\{[q]\})|\{[r]\}$ and $\nu([\delta(x_i) \leftrightarrow x_0]\{[q]\})|\{[r]\} = \nu([\delta(x_i) \leftrightarrow x_0](\{[q]\}|\{[\delta.r]\}))$.

Rule (DELTA′). Suppose $\alpha = \overline{x}$. By hypothesis, $\{[p]\} \xrightarrow{\overline{x}(x_i)} \nu([\delta(x_i) \leftrightarrow x_0]\{[q]\})$, for every x_i s.t. $\delta(x_i) \notin \text{fn}(\{[q]\})$. Thus, $\delta(\{[p]\}) \xrightarrow{\alpha'} \delta(\nu([\delta(x_i) \leftrightarrow x_0]\{[q]\}))$, with $\alpha' = \delta(\overline{x})(\delta(x_i))$, and $\delta(\nu([\delta(x_i) \leftrightarrow x_0]\{[q]\})) = \nu([\delta(\delta(x_i)) \leftrightarrow x_0]\ [x_0 \leftrightarrow x_1]\ \delta(\{[q]\}))$.

Rule (CLOSE). By hypothesis, $\{[p_1]\} \xrightarrow{\overline{x}(y)} \nu([\delta(y) \leftrightarrow x_0]\{[q_1]\})$, for every y s.t. $\delta(y) \notin \text{fn}(\{[q_1]\})$; and $\{[p_2]\} \xrightarrow{x(k)} \nu([\delta(k) \leftrightarrow x_0]\{[q_2]\})$, for every k s.t. $\delta(k) \notin \text{fn}(\{[q_2]\})$. If we take $y = k$, then $\{[p_1]\}|\{[p_2]\} \xrightarrow{\tau} (\boldsymbol{\nu}y)(\nu([\delta(y) \leftrightarrow x_0](\{[q_1]\}|\{[q_2]\})))$, by rule ($\pi$-CLOSE), and $(\boldsymbol{\nu}y)(\nu([\delta(y) \leftrightarrow x_0](\{[q_1]\}|\{[q_2]\}))) = \{[\nu.q_1|q_2]\}$.

Rule (RES). By hypothesis, $\{[p]\} \xrightarrow{\alpha} \{[q]\}$. We take x_i s.t. $\delta(x_i) \notin \text{fn}(\{[p]\}) \cup \text{fn}(\{[q]\}) \cup \text{n}(\alpha)$. Then $\nu([\delta(x_i) \leftrightarrow x_0]\{[p]\}) \xrightarrow{\nu(\alpha)} \nu([\delta(x_i) \leftrightarrow x_0]\{[q]\})$, exploiting that $\nu([\delta(x_i) \leftrightarrow x_0]\alpha) = \nu(\alpha)$. Since $x_i \notin \text{n}(\nu(\alpha))$, by rule ($\pi$-RES), $(\boldsymbol{\nu}x_i)\nu([\delta(x_i) \leftrightarrow x_0]\{[p]\}) \xrightarrow{\nu(\alpha)} (\boldsymbol{\nu}x_i)\nu([\delta(x_i) \leftrightarrow x_0]\{[q]\})$, i.e., $\{[\nu.p]\} \xrightarrow{\nu(\alpha)} \{[\nu.q]\}$.

Rule (RES′). Suppose $\alpha = \overline{x}$ (similar proof, otherwise). By hypothesis, $\{[p]\} \xrightarrow{\overline{x}(x_i)} \nu([\delta(x_i) \leftrightarrow x_0]\{[q]\})$, with $\delta(x_i) \notin \text{fn}(\{[q]\})$.
$(\boldsymbol{\nu}x_k)\nu([\delta(x_k) \leftrightarrow x_0]\{[p]\}) \xrightarrow{\alpha'} (\boldsymbol{\nu}x_k)\nu([\delta(x_k) \leftrightarrow x_0]\nu([\delta(x_i) \leftrightarrow x_0]\{[q]\}))$,
with $\alpha' = \nu([\delta(x_k) \leftrightarrow x_0]\overline{x}.(\nu(x_0)))$. It is possible to prove that $(\boldsymbol{\nu}x_k)\nu([\delta(x_k) \leftrightarrow x_0]\nu([\delta(x_i) \leftrightarrow x_0]\{[q]\})) \equiv \nu([x_j \leftrightarrow x_0]\{[\nu.q]\})$, for $x_j \notin \text{fn}(\{[\nu.q]\})$.

Rule (OPEN). By hypothesis, $\{[p]\} \xrightarrow{\overline{x}\,x_0} \{[q]\}$. Suppose $\delta(x_i) \notin \text{fn}(\{[p]\})$. Then, $(\boldsymbol{\nu}x_i)\nu([\delta(x_i) \leftrightarrow x_0]\{[p]\}) \xrightarrow{\alpha} (\boldsymbol{\nu}x_i)\nu([\delta(x_i) \leftrightarrow x_0]\{[q]\})$, with $\alpha = \nu([x_i \leftrightarrow x_0]\overline{x})(\nu(\delta(x_i)))$, noting that $\nu([x_i \leftrightarrow x_0]\overline{x}) \neq x_i$ and $x \neq x_0$.

□

4.3.2 A Finiteness Result

We now prove that, for each π-agent p, the set of static agents and associated reduction rules needed in all derivations of p is finite up to name permutations. It is not larger than the set of the static subagents of p, closed with respect to name fusions.

Definition 4.3.10.

- *Let s be a static π-agent. We define $S(s)$ as follows: $S(q_1 + q_2) = \mathcal{S}(\llbracket q_1 \rrbracket) \cup \mathcal{S}(\llbracket q_2 \rrbracket) \cup \{q_1 + q_2\}$; $S(\mathtt{rec}\, X.\, q) = [\mathtt{rec}\, X.\, q \mapsto X]\mathcal{S}(\llbracket q \rrbracket) \cup \{\mathtt{rec}\, X.\, q\}$; $S(X) = \varnothing$; $S([x = y]q) = \mathcal{S}(\llbracket q \rrbracket) \cup \{[x = y]q\}$.*
- *Let p be a term of B in the canonical form $p(l_{s_1}, \ldots, l_{s_n})$ such that ρ and δ do not occur in context $p(\_, \_, \ldots)$. We define $\mathcal{S}(p)$ as follows: $\mathcal{S}(\nu.\, p) = \mathcal{S}(p)$; $\mathcal{S}(p|q) = \mathcal{S}(p) \cup \mathcal{S}(q)$; $\mathcal{S}(\mathbf{0}) = \varnothing$; $\mathcal{S}(l_s) = S(s)$.*

Notice that $S(s)$ is defined only for static agents, as, for the subagents q of s, it recalls $\mathcal{S}(\llbracket q \rrbracket)$.

Lemma 4.3.11. *Let p and q be terms of B. If $p \stackrel{\alpha}{\Longrightarrow} q$ then for each $q' \in \mathcal{S}(q)$ there is a $p' \in \mathcal{S}(p)$ with $q' = \sigma p'$, for some name substitution σ (not necessarily a permutation).*

Proof. The proof is by structural induction on the rules $\Rightarrow$. We only consider the most interesting cases.

As remarked, by applying axioms E_{pr} and E', every term p can be reduced into a normal form $p'(l_{s_1} \ldots, l_{s_n})$, with ρ and δ not occurring in context $p'(\_, \_, \ldots)$.

Rule (Rho) (similarly for (Rho$'$)). Let p, ρp, q, and ρq be in normal form. It is easy to see that p and ρp only differ on a finite number of name permutations applied to some s_i in their labels l_{s_i} and analogously for q and ρq. Then, $\mathcal{S}(\rho q) \subseteq \mathcal{S}(\rho p)$, by the hypothesis $\mathcal{S}(q) \subseteq \mathcal{S}(p)$.

Rule (Delta) (similarly for (Delta$'$)). Agents p and $\delta.\, p$ differ on a finite number of name substitutions. Then, the proof proceeds as in the previous case.

Rule (Static) (similarly for (Static$'$)). If $s \xrightarrow{\alpha} r$, the subagents of r are the same as s or less, except possibly for permutations and noninjective substitutions (the latter due to rule (π-Inp)).

□

Theorem 4.3.12. *Let p be in B. The number of constants l_s (and respective axioms for $\Longrightarrow$) in all derivations of p is finite, up to name permutations.*

Proof. Set $\mathcal{S}(p)$ is finite for every p, since it is defined by structural recursion. Thus, also its closure with respect to name fusion $\hat{\mathcal{S}}(p) = \{\sigma q \mid q \in \mathcal{S}(p),\ \sigma \text{ name substitution}\}$ is finite, when taken up to name permutations. But Lemma 4.3.11 guarantees that $p \stackrel{\alpha}{\Longrightarrow} q$ implies $\hat{\mathcal{S}}(q) \subseteq \hat{\mathcal{S}}(p)$. $\square$

The condition "up to name permutations" is not restrictive. Indeed, the idea is that, for each agent p, we can give a finite number of clauses $l_q \stackrel{\alpha}{\Longrightarrow} q'$ such that the transitions of static agents in all the derivations of p only use permutations of the form $l_{\rho q} \stackrel{\rho(\alpha)}{\Longrightarrow} \rho' q'$, with $\rho' = \rho$ or $\rho' = \rho_{+1}$ according to α.

Example 4.3.13. Let us consider again agent $p = \texttt{rec}\, X.\, q$, with $q = (\boldsymbol{\nu} y)\, \overline{y} x_2.\mathbf{0} \mid \overline{x}_2 y.X$. $\mathcal{S}([\![p]\!]) = \{p\} \cup [\texttt{rec}\, X.\, q \mapsto X]\mathcal{S}([\![q]\!])$. Since $[\![q]\!] = \nu.\, l_{\overline{x}_0 x_3.\mathbf{0}} \mid l_{\overline{x}_3 x_0.X}$, then $\mathcal{S}([\![q]\!]) = \{\overline{x}_0 x_3.\mathbf{0}\} \cup \{\overline{x}_3 x_0.X\}$ and $\mathcal{S}([\![p]\!]) = \{p\} \cup \{\overline{x}_0 x_3.\mathbf{0}\} \cup \{\overline{x}_3 x_0.\texttt{rec}\, X.\, q\}$.

4.3.3 A Bialgebra for the π-Calculus

In this section, we apply the results of Sect. 4.1 to the permutation algebra B and to the transition system $\langle B, \Longrightarrow_g\rangle$. We will prove that axioms E_{pr} satisfy the "bisimulation" condition in Theorem 4.1.4 and that $\langle B, \Longrightarrow_g\rangle$ is equivalent to the transition system of the π-calculus. Thus, bisimilarity is a congruence with respect to parallel composition and restriction.

Theorem 4.3.14. *If we consider Δ_{pr} as transition specification, algebra B defined in Definition 4.3.1, with $E_{pr} \cup E' = E$ and $L = C$. Then, Condition 4.1 in Theorem 4.1.4 holds.*

Proof. The proof examines all the axioms E_{pr} and E'.

Ax. $\delta.\, \rho p = \rho_{+1}\delta.\, p$. There are two cases.

1. By rule (Delta), $\delta.\, \rho p \stackrel{\delta(\alpha)}{\Longrightarrow} \delta.\, p'$, with $\delta(\alpha) \neq \bar{x}, x$. Then, $\rho p \stackrel{\rho(\alpha')}{\Longrightarrow} \rho p''$ and $p \stackrel{\alpha'}{\Longrightarrow} p''$, with $\rho p'' = p'$, $\alpha = \rho(\alpha')$ and $\alpha' \neq \bar{x}', x'$, for any x'.

   On the other hand, by rule (Delta), $\delta.\, p \stackrel{\delta(\alpha')}{\Longrightarrow} \delta.\, p''$ with $\alpha' \neq \bar{x}', x'$ and, by rule (Rho), $\rho_{+1}\delta.\, p \stackrel{\rho_{+1}(\delta(\alpha'))}{\Longrightarrow} \rho_{+1}\delta.\, p''$.

2. By rule (Delta′), suppose $\delta.\, \rho p \stackrel{\delta(\bar{x})}{\Longrightarrow} [x_0 \leftrightarrow x_1]\delta.\, p'$ (similarly, if $\alpha = \delta(x)$). Necessarily, $\rho p \stackrel{\rho(\bar{x}')}{\Longrightarrow} \rho_{+1} p''$ and $p \stackrel{\bar{x}'}{\Longrightarrow} p''$, with $\rho(\bar{x'}) = \bar{x}$ and $p' = \rho_{+1} p''$.

   On the other hand, by rule (Delta′), $\delta.\, p \stackrel{\delta(\bar{x}')}{\Longrightarrow} [x_0 \leftrightarrow x_1]\delta.\, p''$ and $\rho_{+1}\delta.\, p \stackrel{\rho_{+1}(\delta(\bar{x}'))}{\Longrightarrow} \rho_{+2}[x_0 \leftrightarrow x_1]\delta.\, p''$. Note that $\rho_{+2}[x_0 \leftrightarrow x_1]\delta.\, p'' = [x_0 \leftrightarrow x_1]\rho_{+2}\delta.\, p''$, as ρ_{+2} does not substitute either x_0 or x_1.

Ax. $\nu.\,(\delta.\,p)|q = p|\nu.\,q$. There are three cases.

1. By rule (Res), $\nu.\,(\delta.\,p)|q \stackrel{\nu\,(\alpha)}{\Longrightarrow} \nu.\,p'$. Necessarily, $(\delta.\,p)|q \stackrel{\alpha}{\Longrightarrow} p'$ and there are the following possible cases.
   (a) By rule (Par), suppose $\delta.\,p \stackrel{\alpha}{\Longrightarrow} p''$ (similarly, otherwise) and $p' = p''|q$. Then, $p \stackrel{\nu\,(\alpha)}{\Longrightarrow} p'''$, with $p'' = \delta.\,p'''$.
   On the other hand, by rule (Par), $p|\nu.\,q \stackrel{\nu\,(\alpha)}{\Longrightarrow} p'''|\nu.\,q$.

   (b) By rule (Com), $\delta.\,p \stackrel{\overline{x}\,y}{\Longrightarrow} p''$, $q \stackrel{xy}{\Longrightarrow} q''$, with $\alpha = \tau$ and $p' = p''|q''$. By rule (Delta), $p \stackrel{\nu\,(\overline{x}\,y)}{\Longrightarrow} p'''$, with $x_0 \notin \mathrm{n}(\overline{x}\,y)$ and $p'' = \delta.\,p'''$.
   On the other hand, by rule (Res), $\nu.\,q \stackrel{\nu\,(xy)}{\Longrightarrow} \nu.\,q''$ and, by rule (Com), $p|\nu.\,q \stackrel{\tau}{\Longrightarrow} p'''|\nu.\,q''$.

   (c) By rule (Close), $\alpha = \tau$. Suppose $\delta.\,p \stackrel{\bar{x}}{\Longrightarrow} p''$, $q \stackrel{x}{\Longrightarrow} q''$, and $p' = \nu.\,p''|q''$ (similar proof, otherwise). Then, by rule (Delta′), $p \stackrel{\nu\,(\overline{x})}{\Longrightarrow} p'''$, with $p'' = [x_0 \leftrightarrow x_1]\delta.\,p'''$.
   On the other hand, $\nu.\,q \stackrel{\nu\,(x)}{\Longrightarrow} \nu.\,q''$ and $p|\nu.\,q \stackrel{\tau}{\Longrightarrow} \nu.\,p'''|\nu.\,q''$, by rule (Close). Note that by axioms E_{pr} $\nu.\,\nu.\,[x_0 \leftrightarrow x_1](\delta.\,p''')|q'' = \nu.\,p'''|\nu.\,q''$.
2. By rule (Res′), $\nu.\,(\delta.\,p)|q \stackrel{\nu\,(\alpha)}{\Longrightarrow} \nu.\,[x_0 \leftrightarrow x_1]p'$. By rule (Par′), suppose $\delta.\,p \stackrel{\alpha}{\Longrightarrow} p''$, with $p' = p''|\delta.\,q$. By rule (Delta′), $p \stackrel{\nu\,(\alpha)}{\Longrightarrow} p'''$ and $p'' = [x_0 \leftrightarrow x_1]\delta.\,p'''$.
   On the other hand, by rule (Par′), $p|\nu.\,q \stackrel{\nu\,(\alpha)}{\Longrightarrow} p'''|\delta.\,\nu.\,q$. By axioms E_{pr}, $\nu.\,[x_0 \leftrightarrow x_1]([x_0 \leftrightarrow x_1]\delta.\,p''')|\delta.\,q = p'''|\delta.\,\nu.\,q$.

3. By rule (Open), $\nu.\,(\delta.\,p)|q \stackrel{\nu\,(\overline{x})}{\Longrightarrow} p'$. Necessarily, $\delta.\,p|q \stackrel{\bar{x}x_0}{\Longrightarrow} p'$ and $x \neq x_0$. By rule (Par), suppose $q \stackrel{\bar{x}x_0}{\Longrightarrow} q'$ and $p' = \delta.\,p|q'$.
   On the other hand, by rule (Open), $\nu.\,q \stackrel{\nu\,(\bar{x})}{\Longrightarrow} q'$ and, by rule (Par′), $p|\nu.\,q \stackrel{\nu\,(\bar{x})}{\Longrightarrow} (\delta.\,p)|q'$.

Ax. $\nu.\,\nu.\,[x_0 \leftrightarrow x_1]p = \nu.\,\nu.\,p$. There are the following possible cases.

1. By rule (Res), $\nu.\,\nu.\,[x_0 \leftrightarrow x_1]p \stackrel{\nu\,(\alpha)}{\Longrightarrow} \nu.\,p'$ and $\nu.\,[x_0 \leftrightarrow x_1]p \stackrel{\alpha}{\Longrightarrow} p'$, with $\alpha \neq \bar{x}, x$ and $x_0 \notin \mathrm{n}(\alpha)$. Necessarily, rule (Res) has been applied

to $\nu. [x_0 \leftrightarrow x_1]p$ and $[x_0 \leftrightarrow x_1]p \stackrel{\delta(\alpha)}{\Longrightarrow} p''$, with $p' = \nu. p''$ and $x_0 \notin \mathrm{n}(\delta(\alpha))$. By rule (RHO), $p \stackrel{\delta(\alpha)}{\Longrightarrow} [x_0 \leftrightarrow x_1]p''$.
On the other hand, by rule (RES), $\nu. p \stackrel{\alpha}{\Longrightarrow} \nu. [x_0 \leftrightarrow x_1]p'$ and, again by rule (RES), $\nu. \nu. p \stackrel{\nu(\alpha)}{\Longrightarrow} \nu. \nu. [x_0 \leftrightarrow x_1]p''$.

2. By rule (OPEN), $\nu. \nu. [x_0 \leftrightarrow x_1]p \stackrel{\nu(\overline{x})}{\Longrightarrow} p'$ and $\nu. [x_0 \leftrightarrow x_1]p \stackrel{\overline{x}x_0}{\Longrightarrow} p'$, with $x \neq x_0$. Necessarily, by rule (RES), $[x_0 \leftrightarrow x_1]p \stackrel{\delta(\overline{x})x_1}{\Longrightarrow} p''$, with $p' = \nu. p''$ and $x_0 \neq \delta(\overline{x}), \delta(x_0)$. By rule (RHO), $p \stackrel{\delta(\overline{x})x_0}{\Longrightarrow} [x_0 \leftrightarrow x_1]p''$.
On the other hand, by rule (OPEN), $\nu. p \stackrel{\overline{x}}{\Longrightarrow} [x_0 \leftrightarrow x_1]p''$ and, by rule (RES′), $\nu. \nu. p \stackrel{\nu(\overline{x})}{\Longrightarrow} \nu. p''$.

3. By rule (RES′), $\nu. \nu. [x_0 \leftrightarrow x_1]p \stackrel{\nu(\alpha)}{\Longrightarrow} \nu. [x_0 \leftrightarrow x_1]p'$ and $\nu. [x_0 \leftrightarrow x_1]p \stackrel{\alpha}{\Longrightarrow} p'$ with $\alpha = \bar{x}, x$ and $x_0 \notin \mathrm{n}(\alpha)$. By rule (RES′), $[x_0 \leftrightarrow x_1]p \stackrel{\delta(\alpha)}{\Longrightarrow} p''$, with $p' = \nu. [x_0 \mapsto x_1]p''$ and $x_0 \notin \mathrm{n}(\delta(\alpha))$. By rule (RHO′), $p \stackrel{\delta(\alpha)}{\Longrightarrow} [\delta(x_0) \leftrightarrow \delta(x_1)]p''$.
On the other hand, by rule (RES′), $\nu. p \stackrel{\alpha}{\Longrightarrow} \nu. [x_0 \leftrightarrow x_1][\delta(x_0) \leftrightarrow \delta(x_1)]p''$ and, again by rule (RES′), $\nu. \nu. p \stackrel{\nu(\alpha)}{\Longrightarrow} \nu. [x_0 \leftrightarrow x_1]\nu. [x_0 \leftrightarrow x_1][\delta(x_0) \leftrightarrow \delta(x_1)]p''$.

4. By rule (RES′), $\nu. \nu. [x_0 \leftrightarrow x_1]p \stackrel{\nu(\alpha)}{\Longrightarrow} \nu. [x_0 \leftrightarrow x_1]p'$ and $\nu. [x_0 \leftrightarrow x_1]p \stackrel{\alpha}{\Longrightarrow} p'$ with $\alpha = \bar{x}, x$ and $x_0 \notin \mathrm{n}(\alpha)$. By rule (OPEN), $[x_0 \leftrightarrow x_1]p \stackrel{\delta(\overline{x})x_0}{\Longrightarrow} p'$, with $\alpha = \overline{x}$. By rule (RHO), $p \stackrel{\delta(\overline{x})x_1}{\Longrightarrow} [x_0 \leftrightarrow x_1]p'$.
On the other hand, by rule (RES), $\nu. p \stackrel{\overline{x}x_0}{\Longrightarrow} \nu. [x_0 \leftrightarrow x_1]p'$ and, by rule (OPEN), $\nu. \nu. p \stackrel{\nu(\overline{x})}{\Longrightarrow} \nu. [x_0 \leftrightarrow x_1]p'$.

Ax. $\rho(p|q) = \rho p \mid \rho q$. The most interesting case is the following.

By rule (RHO′), $\rho(p|q) \stackrel{\rho(\overline{x})}{\Longrightarrow} \rho_{+1}p'$ and $p|q \stackrel{\overline{x}}{\Longrightarrow} p'$. Necessarily, by rule (PAR′), $p \stackrel{\overline{x}}{\Longrightarrow} p''$ and $p' = p''|\delta. q$.

On the other hand, by rule (RHO′), $\rho p \stackrel{\rho(\overline{x})}{\Longrightarrow} \rho_{+1}p''$, and by rule (PAR′) $\rho p \mid \rho q \stackrel{\rho(\overline{x})}{\Longrightarrow} (\rho_{+1}p'') \mid \delta. \rho q$.

Ax. $\rho\nu. p = \nu. \rho_{+1}p$. The most interesting case is as follows.

By rule (RHO′), $\rho\nu. p \stackrel{\rho(\alpha)}{\Longrightarrow} \rho_{+1}p'$ and $\nu. p \stackrel{\alpha}{\Longrightarrow} p'$ with $\alpha = \overline{x}, x$. By rule (RES′), $p \stackrel{\delta(\alpha)}{\Longrightarrow} p''$ and $p' = \nu. [x_0 \leftrightarrow x_1]p''$.

On the other hand, by rule (RHO′), $\rho_{+1}p \overset{\rho_{+1}(\delta\,(\alpha))}{\Longrightarrow} \rho_{+2}p''$, with $x_0 \notin \rho_{+1}(\delta\,(\alpha))$. Then, by rule (RES′), $\nu.\,\rho_{+1}p \overset{\rho(\alpha)}{\Longrightarrow} \nu.\,[x_0 \leftrightarrow x_1]\rho_{+2}p''$ and, by axioms E_{pr}, $\rho_{+1}\nu.\,[x_0 \leftrightarrow x_1]p'' = \nu.\,[x_0 \leftrightarrow x_1]\rho_{+2}p''$.

Ax. $\delta.\,p \,|\, q = (\delta.\,p) \,|\, \delta.\,q$. The most interesting case is the following.

By rule (DELTA′), $\delta.\,p|q \overset{\delta\,(\alpha)}{\Longrightarrow} [x_0 \leftrightarrow x_1]\delta.\,p'$ and $p|q \overset{\alpha}{\Longrightarrow} p'$, with $\alpha = \overline{x}, x$. By rule (PAR′), suppose $p \overset{\alpha}{\Longrightarrow} p''$ and $p' = p''|\delta.\,q$.

On the other hand, by rule (DELTA′), $\delta.\,p \overset{\delta\,(\alpha)}{\Longrightarrow} [x_0 \leftrightarrow x_1]\delta.\,p''$ and, by rule (PAR′), $(\delta.\,p) \,|\, \delta.\,q \overset{\delta\,(\alpha)}{\Longrightarrow} [x_0 \leftrightarrow x_1](\delta.\,p'') \,|\, \delta.\,\delta.\,q$.

Ax. $\delta.\,\nu.\,p = \nu.\,[x_0 \leftrightarrow x_1]\delta.\,p$.

By rule (DELTA′), $\delta.\,\nu.\,p \overset{\delta\,(\alpha)}{\Longrightarrow} [x_0 \leftrightarrow x_1]\delta.\,p'$ and $\nu.\,p \overset{\alpha}{\Longrightarrow} p'$ with $\alpha = \overline{x}, x$. Necessarily, by (RES′), $p \overset{\delta\,(\alpha)}{\Longrightarrow} p''$, with $p' = \nu.\,[x_0 \leftrightarrow x_1]p''$.

On the other hand, by rule (DELTA′), $\delta.\,p \overset{\delta\,(\delta\,(\alpha))}{\Longrightarrow} [x_0 \leftrightarrow x_1]\delta.\,p''$. Then, by rule (RHO′), $[x_0 \leftrightarrow x_1]\delta.\,p \overset{\delta\,(\delta\,(\alpha))}{\Longrightarrow} [\delta\,(x_0) \leftrightarrow \delta\,(x_1)][x_0 \leftrightarrow x_1]\delta.\,p''$. Finally, by rule (RES′), $\nu.\,[x_0 \leftrightarrow x_1]\delta.\,p \overset{\delta\,(\alpha)}{\Longrightarrow} \nu.\,[x_0 \leftrightarrow x_1][\delta\,(x_0) \leftrightarrow \delta\,(x_1)][x_0 \leftrightarrow x_1]\delta.\,p''$.
By axioms E_{pr}, $[x_0 \leftrightarrow x_1]\delta.\,\nu.\,[x_0 \leftrightarrow x_1]p'' = \nu.\,[x_0 \leftrightarrow x_1][\delta\,(x_0) \leftrightarrow \delta\,(x_1)][x_0 \leftrightarrow x_1]\delta.\,p''$.

Ax. $\rho\, l_s = l_{\rho(s)}$.

By rule (RHO′), $\rho l_s \overset{\rho(\alpha)}{\Longrightarrow} \rho_{+1}q$ and $l_s \overset{\alpha}{\Longrightarrow} q$, with $\alpha = \overline{x}, x$. By rule (STATIC′), $s \overset{\{[\alpha]\}}{\longrightarrow} \{[q']\}$ and $q = [\delta\,(y) \leftrightarrow x_0]\delta.\,q'$, for $y \notin \mathrm{fn}(\{[q']\})$.
On the other hand, by applying permutation ρ to transition $s \overset{\{[\alpha]\}}{\longrightarrow} \{[q']\}$, we obtain $\rho(s) \overset{\rho(\{[\alpha]\})}{\longrightarrow} \rho(\{[q']\})$. Then, by rule (STATIC′), $l_{\rho(s)} \overset{\rho(\alpha)}{\Longrightarrow} [\delta\,(\rho(y)) \leftrightarrow x_0]\delta.\,\rho q'$.

Ax. $\delta.\,l_s = l_{\delta\,(s)}$.

By rule (DELTA′), $\delta.\,l_s \overset{\delta\,(\alpha)}{\Longrightarrow} [x_0 \leftrightarrow x_1]\delta.\,q$ and $l_s \overset{\alpha}{\Longrightarrow} q$, with $\alpha = \overline{x}, x$. Necessarily, by rule (STATIC′), $s \overset{\{[\alpha]\}}{\longrightarrow} \{[q']\}$ and $q = [\delta\,(y) \leftrightarrow x_0]\delta.\,q'$, for $y \notin \mathrm{fn}(\{[q']\})$.

On the other hand, by applying function δ to each name in transition $s \overset{\{[\alpha]\}}{\longrightarrow} \{[q']\}$, $\delta\,(s) \overset{\delta\,(\{[\alpha]\})}{\longrightarrow} \delta\,(\{[q']\})$. Finally, by rule (STATIC′), $l_{\delta\,(s)} \overset{\delta\,(\alpha)}{\Longrightarrow} [\delta\,(\delta\,(y)) \leftrightarrow$

$x_0]\delta.\,\delta.\,q'$ and $[x_0 \leftrightarrow x_1]\delta.\,[\delta\,(y) \leftrightarrow x_0]\delta.\,q' = [\delta\,(\delta\,(y)) \leftrightarrow x_0]\delta.\,\delta.\,q'$, as $[x_0 \leftrightarrow x_1]$ is inactive on $\delta.\,\delta.\,q'$.

□

Corollary 4.3.15. *Let B be the algebra defined in Definition 4.3.1. In $g : B \to P_{\Delta_{pr}}(B)$ bisimilarity is a congruence.*

Proof. The claim derives by Proposition 4.2.3, Theorem 4.3.14, Theorem 4.1.5 and Theorem 4.1.4. □

Theorem 4.3.16. *Let p and q be π-agents. Then, $p \sim q$ if and only if $[\![p]\!] \sim_g [\![q]\!]$.*

Proof. We first prove that $p \sim q$ implies $[\![p]\!] \sim_g [\![q]\!]$.

We define a binary relation $\mathcal{R}$ as follows: $[\![p]\!]\,\mathcal{R}\,[\![q]\!] \overset{\text{def}}{\Longleftrightarrow} p \sim q$. We prove that $\mathcal{R}$ is a bisimulation. Suppose $[\![p]\!] \overset{\alpha}{\Longrightarrow} p'$. We only prove the most interesting case, i.e., with $\alpha = \bar{x}, x$, and we suppose $\alpha = \bar{x}$. By Lemma 4.3.9, $p \xrightarrow{\overline{x}(x_i)} \nu\,([\delta\,(x_i) \leftrightarrow x_0]\{[p']\})$, $\forall \delta\,(x_i) \notin \mathrm{fn}(\{[p']\})$, and, by definition of bisimilarity, $q \xrightarrow{\overline{x}(x_i)} q'$, with $q' \sim \nu\,([\delta\,(x_i) \leftrightarrow x_0]\{[p']\})$. By Lemma 4.3.9, $[\![q]\!] \overset{\overline{x}}{\Longrightarrow} [\delta\,(x_i) \leftrightarrow x_0]\delta.\,[\![q']\!]$. By Lemma 4.3.8, $q' \sim \nu\,([\delta\,(x_i) \leftrightarrow x_0]\{[p']\})$ implies $[\delta\,(x_i) \leftrightarrow x_0]\delta\,(q') \sim \{[p']\}$. Finally, by definition of $\mathcal{R}$, $[\delta\,(x_i) \leftrightarrow x_0]\delta.\,[\![q']\!]\,\mathcal{R}\,p'$, as required.

We now prove that $[\![p]\!] \sim_g [\![q]\!]$ implies $p \sim q$. We define a binary relation $\mathcal{R}$ as follows: $p\,\mathcal{R}\,q \overset{\text{def}}{\Longleftrightarrow} [\![p]\!] \sim_g [\![q]\!]$. We prove that $\mathcal{R}$ is a bisimulation. Suppose that $p \xrightarrow{\alpha} p'$. We only prove the most interesting case, i.e., with $\alpha = \overline{x}(y), xy$, and we suppose $\alpha = \overline{x}(y)$. By Lemma 4.3.9, $[\![p]\!] \overset{\overline{x}}{\Longrightarrow} [\delta\,(y) \leftrightarrow x_0]\delta.\,[\![p']\!]$, and, by definition of bisimilarity, $[\![q]\!] \overset{\overline{x}}{\Longrightarrow} q'$, with $q' \sim_g [\delta\,(y) \leftrightarrow x_0]\delta.\,[\![p']\!]$. By Lemma 4.3.9, $q \xrightarrow{\overline{x}(y)} \nu\,([\delta\,(y) \leftrightarrow x_0]\{[q']\})$, assuming $\delta\,(y) \notin \mathrm{fn}(\{[q']\})$. We have to prove that $\nu\,([\delta\,(y) \leftrightarrow x_0]\{[q']\})\,\mathcal{R}\,p'$. By Corollary 4.3.15, $\sim_g$ is a congruence and, thus, $q' \sim_g [\delta\,(y) \leftrightarrow x_0]\delta.\,[\![p']\!]$ implies $\nu.\,[\delta\,(y) \leftrightarrow x_0]q' \sim_g [\![p']\!]$. Finally, by definition of $\mathcal{R}$, $\nu\,([\delta\,(y) \leftrightarrow x_0]\{[q']\})\,\mathcal{R}\,p'$, as required. □

Corollary 4.3.17. *Bisimilarity $\sim$ in the π-calculus is a congruence with respect to parallel composition and restriction.*

Proof. First, we prove that if $p_1 \sim q_1$ and $p_2 \sim q_2$, then $p_1|p_2 \sim q_1|q_2$. By Theorem 4.3.16, $[\![p_1]\!] \sim_g [\![q_1]\!]$ and $[\![p_2]\!] \sim_g [\![q_2]\!]$. Since $\sim_g$ is a congruence, $[\![p_1]\!]|[\![p_2]\!] \sim_g [\![q_1]\!]|[\![q_2]\!]$ and, thus, $[\![p_1|p_2]\!] \sim_g [\![q_1|q_2]\!]$. Finally, by Theorem 4.3.16, $p_1|p_2 \sim q_1|q_2$.

Now, we prove that if $p \sim q$ then $(\boldsymbol{\nu}x)\,p \sim (\boldsymbol{\nu}x)\,q$. By Theorem 4.3.16, $[\![p]\!] \sim_g [\![q]\!]$. Since $\sim_g$ is a congruence, $\nu.\,[\delta\,(x) \leftrightarrow x_0]\delta.\,[\![p]\!] \sim_g \nu.\,[\delta\,(x) \leftrightarrow x_0]\delta.\,[\![q]\!]$ and, by definition of $[\![]\!]$, $[\![(\boldsymbol{\nu}x)\,p]\!] \sim_g [\![(\boldsymbol{\nu}x)\,q]\!]$. Finally, by Theorem 4.3.16, $(\boldsymbol{\nu}x)\,p \sim (\boldsymbol{\nu}x)\,q$. □

Part II

Security

Chapter 5

A Framework for the Analysis of Security Protocols

In this chapter we propose a language and a method based on symbolic analysis for automated analysis of security protocols. The method supports a variety of cryptographic functions.

As a base language, we consider a dialect of the spi-calculus [4]. Unlike the original spi-calculus and the languages in [5, 10], where cryptographic functions are modeled as process operators, we adopt a uniform process syntax, much as that of [3], and consider a generic signature Σ of cryptographic functions. The latter may include constructors and destructors for various cryptographic operations, akin to [1]. The meaning of Σ-terms is provided by an evaluation relation $\downarrow$ that maps terms to values (or *messages*). On top of the evaluation relation, we introduce a deduction relation $\vdash$ that describes how the environment synthesizes new messages from known ones. Protocol properties are formalized as correspondence assertions of the kind "every execution of action α must be preceded by some execution of action β", for given α and β.

The ("concrete") operational semantics of this language is, as expected, infinitary, because each input action gives rise to infinitely many transitions. To overcome this problem, we also define a symbolic operational semantics that is finitely branching and yields finite models of protocols with a fixed number of participants. Then, we give a method to carry out trace analysis directly on the symbolic traces.

Next, we provide some reasonable conditions on the given cryptosystem under which we can prove that the method is sound and complete with respect to the concrete semantics. In other words, every attack detected in the symbolic model corresponds to some attack in the concrete one, and vice-versa. Thus our symbolic method makes no approximation with respect to the infinitary, concrete model. For instance, type-dependent flaws (see e.g. [37]), which usually escape finite-state analysis, with our approach naturally emerge when present. The method is rather efficient in practice, because in the symbolic model there is no state-explosion induced by message exchange: every input action gives rise to exactly one symbolic transition. Based on the theory exposed in this chapter, we have developed STA [75], a prototype tool for automated verification of cryptographic protocols. An overview of STA and some examples of its application are given in Chapter 6.

Structure of the chapter. In Section 5.1 we introduce the general framework at the basis of our method. In Section 5.2 we present the language and discuss the symbolic semantics and its relationship to the concrete one. We describe the verification method in Section 5.3 and illustrate its application by means of two examples: the case of public-key cryptography in Section 5.4 and an extended system featuring shared-key, public-key, digital signature and hashing in Section 5.5. We discuss related work, based on symbolic analysis, in Section 5.6.

5.1 A General Framework

In this section, we present the main ingredients of our framework. We introduce the concept of *frame*, i.e., a structure consisting of a signature Σ, a set of (legal) messages and a function that evaluates terms to messages. Then, we generalize the notions of traces, configurations and security property, introduced in [10].

5.1.1 Frames

We consider two countable disjoint sets of *names* $m, n, \ldots \in \mathcal{N}$ and *variables* $x, y, \ldots \in \mathcal{V}$. The set $\mathcal{N}$ is in turn partitioned into a countable set of *local names* $a, b, \ldots \in \mathcal{LN}$ and a countable set of *environmental names* $\underline{a}, \underline{b}, \ldots \in \mathcal{EN}$: these two sets represent the basic data (keys, nonces,...) initially known to any process and to the environment, respectively. The set $\mathcal{N} \cup \mathcal{V}$ is ranged over by letters $u, v, \ldots$. Given a signature Σ of function symbols $f, g, \ldots$, each coming with its arity (constants have arity 0), we denote by $\mathcal{E}_\Sigma$ the algebra of terms (or *expressions*) on $\mathcal{N} \cup \mathcal{V} \cup \Sigma$, given by the grammar:

$$\zeta, \eta \quad ::= \quad u \quad | \quad f(\widetilde{\zeta})$$

where $\widetilde{\zeta}$ is a tuple of terms of the expected length. A *term context* $C[\cdot]$ is a term with a hole that can be filled with any expression ζ, thus yielding an expression $C[\zeta]$.

Definition 5.1.1 (frame). *A* frame $\mathcal{F}$ *is a triple* $(\Sigma, \mathcal{M}, \downarrow)$*, where:*

- Σ *is a signature;*
- $\mathcal{M} \subseteq \mathcal{E}_\Sigma$ *is a set of* messages $M, N, \ldots$*;*
- $\downarrow \subseteq \mathcal{E}_\Sigma \times \mathcal{E}_\Sigma$ *is an* evaluation relation*.*

In the sequel, we write $\zeta \downarrow \eta$ for $(\zeta, \eta) \in \downarrow$ and say that ζ *evaluates to* η. In typical frame instances the relation $\downarrow$ will be both a function and a congruence with respect to the operations in Σ, but we need not to assume these facts in the general framework.

Below, we define a deduction relation which expresses how the environment can generate new messages starting from an initial set of messages S. Unlike other approaches, our definition of deduction relation is not given by a set of deductive rules. Rather, we make use of the set $\mathcal{H}(S)$, which consists of all the expressions inductively built by applying functions of Σ to elements of S and of $\mathcal{EN}$. We denote by $\mathcal{P}_f(X)$ the set of finite subsets of X.

Definition 5.1.2 (deduction relation). *For* $\mathcal{F} = (\Sigma, \mathcal{M}, \downarrow)$ *a frame and* $S \subseteq \mathcal{M}$*, the set* $\mathcal{H}_\mathcal{F}(S)$ *is inductively defined by the following rules:*

$$\begin{array}{rcl}
\mathcal{H}^0_\mathcal{F}(S) & \stackrel{\triangle}{=} & S \cup \mathcal{EN} \\
\mathcal{H}^{i+1}_\mathcal{F}(S) & \stackrel{\triangle}{=} & \mathcal{H}^i_\mathcal{F}(S) \cup \{ f(\widetilde{\zeta}) \, : \, f \in \Sigma, \, \widetilde{\zeta} \subseteq \mathcal{H}^i_\mathcal{F}(S) \} \\
\mathcal{H}_\mathcal{F}(S) & \stackrel{\triangle}{=} & \bigcup_{i \geq 0} \mathcal{H}^i_\mathcal{F}(S)
\end{array}$$

The deduction relation $\vdash_\mathcal{F} \subseteq \mathcal{P}_f(\mathcal{M}) \times \mathcal{M}$ *is defined by:*

$$S \vdash_\mathcal{F} M \quad \stackrel{\triangle}{\Leftrightarrow} \quad \exists \, \zeta \in \mathcal{H}_\mathcal{F}(S) \, : \, \zeta \downarrow M$$

A message M *is* deducible *from* S *if* $S \vdash_\mathcal{F} M$*.*

When no confusion arises, we simply write $\mathcal{H}(S)$ for $\mathcal{H}_\mathcal{F}(S)$ and $\vdash$ for $\vdash_\mathcal{F}$.

5.1.2 Processes

Syntax As a base language, we consider a variant of the spi-calculus [4], parametrized by an arbitrary frame $\mathcal{F}$ (for readability, we omit explicit reference to $\mathcal{F}$ in the notation). The syntax of *agent expressions*, in $\mathcal{A}$, is reported in Table 5.1.

We consider a set $\mathcal{L}$ of *labels* which is ranged over by $\mathsf{a}, \mathsf{b}, \ldots$. The main difference from the spi-calculus is that, here, input and output labels $(\mathsf{a}, \mathsf{b}, \ldots)$ must not be regarded as channels – following the Dolev-Yao model, we assume just one public network – but, rather, as 'tags' attached to process actions for ease of reference. Also, we have a single construct (let) for evaluating expressions that replaces the ad-hoc constructs found in the spi-calculus for encryption, decryption and other cryptographic operations. We do not consider restriction: it might be easily accommodated but it has little semantical significance, in the absence of replication/recursion.

Given the presence of binders for variables, notions of *free variables*, $\mathrm{v}(A) \subseteq \mathcal{V}$, and *alpha-equivalence* arise as expected. We shall identify alpha-equivalent agent expressions. For any M and u, $[^{M}\!/_{u}]$ denotes the operation of substituting the free occurrences of u by M. An agent expression A is said to be *closed* or a *process* if $\mathrm{v}(A) = \varnothing$; the set of processes $\mathcal{P}$ is ranged over by $P, Q, \ldots$. Local names and environmental names occurring in A are denoted by $\mathrm{ln}(A)$ and $\mathrm{en}(A)$, respectively. A process P is *initial* if $\mathrm{en}(P) = \varnothing$.

A, B	$::=$		agents $\mathcal{A}$
		$\mathbf{0}$	(null)
	$\mid$	$\mathsf{a}(x).\,A$	(input)
	$\mid$	$\overline{\mathsf{a}}\langle\zeta\rangle.\,A$	(output)
	$\mid$	$\mathsf{let}\ y = \zeta\ \mathsf{in}\ A$	(evaluation)
	$\mid$	$[\zeta = \eta]A$	(matching)
	$\mid$	$A \parallel B$	(parallel composition)

The occurrences of variables x and y are bound.

Table 5.1. Syntax for agents

Operational Semantics The semantics of the calculus is given in terms of a transition relation $\longrightarrow$, which we will sometimes refer to as 'concrete' (as opposed to the 'symbolic' one we shall introduce later on). We model the state of the system as a pair $\langle s,\ P\rangle$, where s records the current environment's knowledge (i.e., the sequence of messages the environment has "seen" on the network up to a given moment) and P is a process term. An *action* is a term of the form $\mathsf{a}\langle M\rangle$ (*input* action) or $\overline{\mathsf{a}}\langle M\rangle$ (*output* action), for a a label and M a message. The set of actions Act is ranged over by $\alpha, \beta, \ldots$, while the set Act^* of strings of actions is ranged over by $s, s', \ldots$. String concatenation is written '$\cdot$'. We denote by $\mathrm{act}(s)$ and $\mathrm{msg}(s)$ the set of actions and messages, respectively, appearing in s. A string s is *closed* if $\mathrm{v}(s) = \varnothing$ and *initial* if $\mathrm{en}(s) = \varnothing$. In what follows, we write '$s \vdash M$' for $\mathrm{msg}(s) \vdash M$.

We, now, define *traces*, that is, sequences of actions that may result from the interaction between a process and its environment. In traces, each message received by a process (input message) can be synthesized from the knowledge the environment has previously acquired. In *configurations*, the latter is explicitly recorded.

Definition 5.1.3 (traces and configurations). *A* trace *is a closed string* $s \in Act^*$ *such that for each* s_1, s_2 *and* $\mathsf{a}\langle M\rangle$, *if* $s = s_1 \cdot \mathsf{a}\langle M\rangle \cdot s_2$ *then* $s_1 \vdash M$.

A configuration, *written as* $\langle s,\ P\rangle$, *is a pair consisting of a trace s and a process P. A configuration is* initial *if* $\mathsf{en}(s, P) = \varnothing$. *Configurations are ranged over by* $\mathcal{C}, \mathcal{C}', \ldots$.

The concrete transition relation on configurations is defined by the rules below.

(INP)	$\langle s,\ \mathsf{a}(x).\,P\rangle$	$\longrightarrow$	$\langle s\cdot \mathsf{a}\langle M\rangle,\ P[^M/_x]\rangle$	$s \vdash M$, M closed
(OUT)	$\langle s,\ \overline{\mathsf{a}}\langle\zeta\rangle.\,P\rangle$	$\longrightarrow$	$\langle s\cdot \overline{\mathsf{a}}\langle M\rangle,\ P\rangle$	$\zeta \downarrow M$, M closed
(LET)	$\langle s,\ \mathsf{let}\, y = \zeta\, \mathsf{in}\, P\rangle$	$\longrightarrow$	$\langle s,\ P[^\xi/_y]\rangle$	$\zeta \downarrow \xi$, ξ closed
(MATCH)	$\langle s,\ [\zeta = \eta]P\rangle$	$\longrightarrow$	$\langle s,\ P\rangle$	$\zeta \downarrow \xi$, $\eta \downarrow \xi$

(PAR) $$\frac{\langle s,\ P\rangle \longrightarrow \langle s',\ P'\rangle}{\langle s,\ P \parallel Q\rangle \longrightarrow \langle s',\ P' \parallel Q\rangle}$$

plus symmetric version of (PAR).

Each action taken by the process is recorded in the configuration's first component. Rule (INP) makes the transition relation infinitely-branching, as M ranges over the infinite set $\{M : s \vdash M,\ M \text{ closed}\}$. In rule (OUT), ζ is evaluated before the action takes place. By rule (LET), the evaluation of ζ replaces any occurrence of y in P. No handshake communication is provided: all messages go through the environment (rule (PAR)). By $\mathcal{C} \longrightarrow^n \mathcal{C}'$ we mean that $\mathcal{C}$ reduces to $\mathcal{C}'$ in n execution steps.

5.1.3 Properties

We express security properties of a protocol in terms of the traces it generates. In particular, we focus on correspondence assertions of the kind 'for every generated trace, whenever action β occurs in the trace, then action α must have occurred at some previous point in the trace'. Given a configuration $\langle s,\ P\rangle$ and a trace s', we say that $\langle s,\ P\rangle$ *generates* s', written $\langle s,\ P\rangle \searrow s'$, if $\langle s,\ P\rangle \longrightarrow^* \langle s',\ P'\rangle$ for some P'. A substitution θ in a frame $\mathcal{F}$ is a finite partial map from $\mathcal{V}$ to the set of messages $\mathcal{M}$ of frame $\mathcal{F}$ such that $\theta(x) \neq x$, for each variable x. For any object t (i.e. variable, message, process, trace,...), we denote by $t\theta$ the result of simultaneously replacing each $x \in \mathrm{v}(t) \cap \mathrm{dom}(\theta)$ by $\theta(x)$. We let ρ range over ground substitutions, i.e. substitutions that map variables to closed messages.

Definition 5.1.4 (satisfaction relation). *Let α and β be actions and s be a trace. We say that α* occurs prior to *β in s if whenever $s = s' \cdot \beta \cdot s''$ then $\alpha \in \mathrm{act}(s')$. For $\mathrm{v}(\alpha) \subseteq \mathrm{v}(\beta)$, we write $s \models \alpha \hookleftarrow \beta$, and say s* satisfies *$\alpha \hookleftarrow \beta$, if for each ground substitution ρ it holds that $\alpha\rho$ occurs prior to $\beta\rho$ in s. We say that a configuration C* satisfies *$\alpha \hookleftarrow \beta$, and write $C \models \alpha \hookleftarrow \beta$, if all traces generated by C satisfy $\alpha \hookleftarrow \beta$.*

Assertions $\alpha \hookleftarrow \beta$ can express interesting secrecy and authentication properties. As an example, in the final step of many key-establishment protocols, a principal A sends a message of the form $\{N\}_k$ to a responder B, where $\{N\}_k$ is obtained by encrypting some authentication information N under a newly established shared-key k. Our scheme permits expressing that *every* message encrypted with k that is accepted by B during the execution of the protocol indeed originates from A, i.e. that B is really talking to A, and that k is authentic. If we denote by $\mathsf{final_A}$ and $\mathsf{final_B}$ the labels attached to A's and B's final action, respectively, then the above property might be formalized as an assertion $\overline{\mathsf{final_A}}\langle\{x\}_k\rangle \hookleftarrow \mathsf{final_B}\langle\{x\}_k\rangle$, for x a variable. In the next chapter we will show examples of application of the method to two well-known security protocols. In practice, all forms of authentication in Lowe's hierarchy [45] are captured by this scheme, except the most demanding one that requires one-to-one bijection between α's and β's. However, we expect our scheme can be easily adjusted to include this stronger form, by requiring that each β is preceded by *exactly* one occurrence of α.

Another property that can be set within our frame is *secrecy* in the style of [7]. In this case, it is convenient to fix a conventional 'absurd' action $\perp$ that is nowhere used in agent expressions. Thus, the formula $\perp \hookleftarrow \alpha$ means that action α should never take place. Now, the fact that a protocol, say P, does not leak a sensible datum, say d, can be expressed also by saying that the adversary will never be capable of synthesizing d. This can be formalized by extending the protocol to include a 'guardian' that at any time picks up one message from the network, $P \parallel \mathrm{g}(x).\mathbf{0}$, and then requiring that this guardian will never receive d, that is, $\langle \epsilon,\ P \parallel \mathrm{g}(x).\mathbf{0}\rangle \models \perp \hookleftarrow \mathrm{g}\langle d\rangle$.

5.2 Symbolic Semantics

The symbolic semantics we present in this section is based on the notion of symbolic frame. The latter is essentially a frame equipped with an additional symbolic evaluation relation, which is in agreement with its concrete counterpart.

Formally, let us denote by Subst the set of all substitutions in a given frame. A substitution θ is a *unifier* of t_1 and t_2 if $t_1\theta = t_2\theta$. We denote by $\mathrm{mgu}(t_1, t_2)$ a chosen *most general unifier* (mgu) of t_1 and t_2, that is, a unifier θ of t_1 and t_2 such that any other unifier is a composition of substitutions θ and θ', written $\theta\theta'$, for some θ'. Also, for t_1, t_1', t_2, t_2' terms, $\mathrm{mgu}(t_1 = t_1', t_2 = t_2')$ stands for $\theta\,\mathrm{mgu}(t_2\theta, t_2'\theta)$, where $\theta = \mathrm{mgu}(t_1, t_1')$, if such mgu's exist.

We introduce below a symbolic evaluation relation $\downarrow_s$. Intuitively, $\zeta \downarrow_\theta \eta$ means that ζ evaluates to η under any possible instance of θ. The main advantage of this evaluation relation with respect to the "concrete" one ($\downarrow$) relies on the fact that infinitely many pairs (ζ, η) such that $\zeta \downarrow \eta$ can be represented by means of a single judgement $\zeta_0 \downarrow_\theta \eta_0$, for some ζ_0, θ, η_0. This intuition will be clarified in the example on public-key encryption below.

Definition 5.2.1 (symbolic frame). *A* symbolic frame *is a pair* $\mathcal{F}^s = (\mathcal{F}, \downarrow_s)$, *where* $\mathcal{F}$ *is a frame, and* $\downarrow_s \subseteq \mathcal{E}_\Sigma \times \mathsf{Subst} \times \mathcal{E}_\Sigma$ *is a* symbolic evaluation relation *(we write* $\zeta \downarrow_\theta \eta$ *for* $(\zeta, \theta, \eta) \in \downarrow_s$*) such that, for any expression* ζ *and ground substitution* ρ *with* $\mathrm{v}(\zeta) \subseteq \mathrm{dom}(\rho)$, *the followings hold:*

(a) *If* $\zeta\rho \downarrow \eta$, *then there exist* ξ, θ, ρ_0 *such that* $\zeta \downarrow_\theta \xi$, $\rho = \theta\rho_0$ *and* $\eta = \xi\rho_0$. *Furthermore,* $\eta \in \mathcal{M}$ *implies* $\xi \in \mathcal{M}$.

(b) *If* $\zeta \downarrow_\theta \xi$, $\rho = \theta\rho_0$, *for some* ρ_0, *then* $\zeta\rho \downarrow \xi\rho_0$.

We now come to symbolic counterparts of traces and configurations. Note that Condition (b) below states that only the environment can introduce variables into symbolic traces.

Definition 5.2.2 (symbolic traces and configurations). *A* symbolic trace *is a string* $s \in Act^*$ *such that: (a)* $\mathrm{en}(s) = \emptyset$, *and (b) for each* s_1, s_2, α *and* x, *if* $s = s_1 \cdot \alpha \cdot s_2$ *and* $x \in \mathrm{v}(\alpha) - \mathrm{v}(s_1)$ *then* α *is an input action. Symbolic traces are ranged over by* $\sigma, \sigma', \ldots$. *A* symbolic configuration, *written* $\langle \sigma,\ A\rangle_s$, *is a pair composed by a symbolic trace* σ *and an agent* A, *such that* $\mathrm{en}(A) = \emptyset$ *and* $\mathrm{v}(A) \subseteq \mathrm{v}(\sigma)$.

Note that, due to Condition (b) in the Definition 5.2.2, e.g. $\bar{\mathsf{a}}\langle x^+\rangle \cdot \mathsf{a}\langle\{[h]\}_{x^+}\rangle$ is not a symbolic trace, while $\mathsf{a}\langle\{[h]\}_{x^+}\rangle \cdot \bar{\mathsf{a}}\langle x^+\rangle$ is so.

Once a symbolic frame $\mathcal{F}^s$ is fixed, configurations can be equipped with a symbolic transition relation, $\longrightarrow_s$, as defined by the rules below.

$$
\begin{array}{lrcll}
(\textsc{Inp}_\mathrm{s}) & \langle \sigma,\ \mathsf{a}(x).\,A\rangle_\mathrm{s} & \longrightarrow_\mathrm{s} & \langle \sigma\cdot \mathsf{a}\langle x\rangle,\ A\rangle_\mathrm{s} & \\
(\textsc{Out}_\mathrm{s}) & \langle \sigma,\ \bar{\mathsf{a}}\langle \zeta\rangle.\,A\rangle_\mathrm{s} & \longrightarrow_\mathrm{s} & \langle \sigma\theta\cdot \bar{\mathsf{a}}\langle M\rangle,\ A\theta\rangle_\mathrm{s} & \zeta \downarrow_\theta M \\
(\textsc{Let}_\mathrm{s}) & \langle \sigma,\ \mathsf{let}\ y=\zeta\ \mathsf{in}\ A\rangle_\mathrm{s} & \longrightarrow_\mathrm{s} & \langle \sigma\theta,\ A\theta[^{\xi}/_{y}]\rangle_\mathrm{s} & \zeta \downarrow_\theta \xi \\
(\textsc{Match}_\mathrm{s}) & \langle \sigma,\ [\zeta=\eta]A\rangle_\mathrm{s} & \longrightarrow_\mathrm{s} & \langle \sigma\theta,\ A\theta\rangle_\mathrm{s} & \zeta \downarrow_{\theta_1} \xi_1,\ \eta\,\theta_1 \downarrow_{\theta_2} \xi_2, \\
 & & & & \theta_3 = \mathrm{mgu}(\xi_1\theta_2, \xi_2), \\
 & & & & \theta = \theta_1\theta_2\theta_3
\end{array}
$$

$$
(\textsc{Par}_\mathrm{s}) \quad \frac{\langle \sigma,\ A\rangle_\mathrm{s} \longrightarrow_\mathrm{s} \langle \sigma',\ A'\rangle_\mathrm{s}}{\langle \sigma,\ A \parallel B\rangle_\mathrm{s} \longrightarrow_\mathrm{s} \langle \sigma',\ A' \parallel B'\rangle_\mathrm{s}}
$$

plus symmetric version of ($\textsc{Par}_\mathrm{s}$).

In these rules, we assume that: (i) there is a function $\mathrm{new}_\mathcal{V}(\cdot)$ such that, for any given $V \subseteq_{\mathrm{fin}} \mathcal{V}$, $\mathrm{new}_\mathcal{V}(V)$ is a variable not in V; (ii) $x = \mathrm{new}_\mathcal{V}(V)$ – where V is the set of free variables in the source configuration; (iii) $y = \mathrm{new}_\mathcal{V}(V \cup \{x\})$ and $\mathrm{msg}(\sigma)\theta \subseteq \mathcal{M}$; (iv) in rule ($\textsc{Par}_\mathrm{s}$), $B' = B\theta$ where $\langle \sigma,\ A\rangle_\mathrm{s} \xrightarrow{\theta}_\mathrm{s} \langle \sigma',\ A'\rangle_\mathrm{s}$ and $\sigma' = \sigma\theta\cdot\alpha$ or $\sigma' = \sigma\theta$. Note that, differently from the concrete semantics, input variables are *not* instantiated immediately (rule ($\textsc{Inp}_\mathrm{s}$)). Rather, constraints on these variables are added as soon as they are needed, and recorded via mgu's. This may occur due to rules ($\textsc{Out}_\mathrm{s}$), ($\textsc{Let}_\mathrm{s}$) and ($\textsc{Match}_\mathrm{s}$). In the following example, after the first step, variable x gets instantiated to name b by a ($\textsc{Match}_\mathrm{s}$)-reduction:

$$
\langle \epsilon,\ \mathsf{a}(x).\,[x=b]P\rangle_\mathrm{s} \ \longrightarrow_\mathrm{s}\ \langle \mathsf{a}\langle x\rangle,\ [x=b]P\rangle_\mathrm{s} \ \longrightarrow_\mathrm{s}\ \langle \mathsf{a}\langle b\rangle,\ P[^{b}/_{x}]\rangle_\mathrm{s}
$$

Whenever $\langle \sigma,\ A\rangle_\mathrm{s} \longrightarrow_\mathrm{s}^* \langle \sigma',\ A'\rangle_\mathrm{s}$ for some A', we say that $\langle \sigma,\ A\rangle_\mathrm{s}$ *symbolically generates* σ', and write $\langle \sigma,\ A\rangle_\mathrm{s} \searrow_\mathrm{s} \sigma'$. We write $\mathcal{C} \xrightarrow{\theta}_\mathrm{s} \mathcal{C}'$ for $\mathcal{C} \longrightarrow_\mathrm{s} \mathcal{C}'$, where θ is the substitution applied to C' in the reduction step. The relation $\longrightarrow_\mathrm{s}$ is finitely-branching if so is $\downarrow_s$ (this is the case, e.g., for the public key frame $\mathcal{F}^s_{pk}$ we will illustrate in Section 5.4). In this case, each configuration generates a finite number of symbolic traces. It is important to stress that many symbolic traces are in fact 'garbage' – jumbled sequences of actions that cannot be instantiated to any concrete trace. For instance, consider process $P \triangleq \mathsf{a}(y).\,\mathsf{let}\ x=\mathsf{dec}^{\mathsf{pk}}_{k^-}(y)\ \mathsf{in}\ \bar{\mathsf{a}}\langle x\rangle.\,\mathbf{0}$. The initial configuration $\langle \epsilon,\ P\rangle_\mathrm{s}$ symbolically generates the trace $\mathsf{a}\langle \{[z]\}_{k^+}\rangle \cdot \bar{\mathsf{a}}\langle z\rangle$, which is inconsistent, because the environment cannot provide k^+ (i.e. $\epsilon \not\vdash k^+$).

The problem of detecting these inconsistent traces, that might give rise to 'false positives' when checking protocol properties, will be faced in the next section.

Definition 5.2.3 (solutions of symbolic traces). *Given a symbolic trace σ and a ground substitution ρ, we say that ρ* satisfies *σ if $\sigma\rho$ is a trace. If this is the case, we also say that $\sigma\rho$ is a* solution *of σ, and that σ is* consistent.

Based on the notion of consistency, we now prove a theorem that lifts the concrete-symbolic correspondence given in Definition 5.2.1 to the transition relations. Its proof relies on the lemma below.

Lemma 5.2.4. *Let $\mathcal{F}^s$ be a symbolic frame, $\mathcal{C}$ be a symbolic configuration and $\mathcal{C}\rho$ be a configuration. Then:*

(1) *$\mathcal{C}\rho \longrightarrow \mathcal{C}'$ implies that there exist $\mathcal{C}_1$, θ and ρ_0 such that $\rho = \theta\rho_0$, $\mathcal{C} \xrightarrow{\theta}_{\mathrm{s}} \mathcal{C}_1$ and $\mathcal{C}' = \mathcal{C}_1\rho_0$.*

(2) *$\mathcal{C} \xrightarrow{\theta}_{\mathrm{s}} \mathcal{C}_1$ and $\rho = \theta\rho_0$, for some ρ_0 such that $\mathcal{C}_1\rho_0$ is a configuration, imply that $\mathcal{C}\rho \longrightarrow \mathcal{C}_1\rho_0$.*

Proof.

(1) By induction on the rules of the transition relation $\longrightarrow$. The cases (INP) and (PAR) are trivial.

(OUT) $\langle \sigma\rho,\ \bar{\mathsf{a}}\langle\zeta\rho\rangle.\,A\rho\rangle \longrightarrow \langle \sigma\rho\cdot\bar{\mathsf{a}}\langle M\rangle,\ A\rho\rangle$, with $\zeta\rho \downarrow M$. By def. of symbolic frame, $\zeta \downarrow_\theta N$, where $\rho = \theta\rho_0$ and $M = N\rho_0$. Then, $\langle\sigma,\ \bar{\mathsf{a}}\langle\zeta\rangle.\,A\rangle_{\mathrm{s}} \xrightarrow{\theta}_{\mathrm{s}} \langle\sigma\theta\cdot\bar{\mathsf{a}}\langle N\rangle,\ A\theta\rangle_{\mathrm{s}}$, where $\langle\sigma\theta\cdot\bar{\mathsf{a}}\langle N\rangle,\ A\theta\rangle_{\mathrm{s}}\rho_0 = \langle\sigma\rho\cdot\bar{\mathsf{a}}\langle M\rangle,\ A\rho\rangle$.

(LET) $\langle\sigma\rho,\ \mathsf{let}\, y = \zeta\rho \,\mathsf{in}\, A\rho\rangle \longrightarrow \langle\sigma\rho,\ A\rho[\xi/y]\rangle$, with $\zeta\rho \downarrow \xi$. By definition of symbolic frame, $\zeta \downarrow_\theta N$, $\rho = \theta\rho_0$, and $M = N\rho_0$. Then, $\langle\sigma,\ \mathsf{let}\, y = \zeta \,\mathsf{in}\, A\rangle_{\mathrm{s}} \xrightarrow{\theta}_{\mathrm{s}} \langle\sigma\theta,\ A\theta[N/y]\rangle_{\mathrm{s}}$ where $\langle\sigma\theta,\ A\theta[N/y]\rangle_{\mathrm{s}}\rho_0 = \langle\sigma\rho,\ A\rho[M/y]\rangle$.

(MATCH) $\langle\sigma\rho,\ [\zeta\rho = \eta\rho]A\rho\rangle \longrightarrow \langle\sigma\rho,\ A\rho\rangle$, where $\zeta\rho \downarrow M$, $\eta\rho \downarrow N$ and $M = N$. By definition of symbolic frame, $\zeta\rho \downarrow M$ implies $\zeta \downarrow_{\theta_1} M_1$, where $\rho = \theta_1\rho_0{}'$ and $M = M_1\rho_0{}'$, for some ρ_0'. Also $\eta\rho = \eta\theta_1\rho_0{}'$ and, thus, $\eta\theta_1 \downarrow_{\theta_2} N_1$, where $\rho_0{}' = \theta_2\rho_0{}''$ and $N_1\rho_0{}'' = N$, for some $\rho_0{}''$. Since $M = M_1\rho_0{}' = M_1\theta_2\rho_0{}'' = N$, then $M_1\theta_2$ and N_1 are unifiable. Let $\theta_3 = \mathrm{mgu}(M_1\theta_2, N_1)$, i.e. $\rho_0{}'' = \theta_3\rho_0{}'''$ for some $\rho_0{}'''$. We choose $\theta \triangleq \theta_1\theta_2\theta_3$. Then, $\rho = \theta\rho_0{}'''$ and $\langle\sigma,\ [\zeta = \eta]A\rangle_{\mathrm{s}} \xrightarrow{\theta}_{\mathrm{s}} \langle\sigma\theta,\ A\theta\rangle_{\mathrm{s}}$, where $\langle\sigma\theta,\ A\theta\rangle_{\mathrm{s}}\rho_0''' = \langle\sigma\rho,\ A\rho\rangle$.

(2) By induction on $\longrightarrow_{\mathrm{s}}$. The cases (INP$_{\mathrm{s}}$) and (PAR$_{\mathrm{s}}$) are trivial.

(OUT_s) $\langle\sigma,\ \bar{\mathsf{a}}\langle\zeta\rangle.\,A\rangle_s \xrightarrow{\theta}_s \langle\sigma\theta\cdot\bar{\mathsf{a}}\langle N\rangle,\ A\theta\rangle_s$, where $\zeta\ \downarrow_\theta\ N$ and $\rho = \theta\rho_0$, for some ρ_0. By definition of symbolic frame, $\zeta\rho\ \downarrow\ N\rho_0$. Then, by (OUT) $\langle\sigma\rho,\ \bar{\mathsf{a}}\langle\zeta\rho\rangle.\,A\rho\rangle \longrightarrow \langle\sigma\rho\cdot\bar{\mathsf{a}}\langle N\rho_0\rangle,\ A\rho\rangle$ and $\langle\sigma\rho\cdot\bar{\mathsf{a}}\langle N\rho_0\rangle,\ A\rho\rangle = \langle\sigma\theta\cdot\bar{\mathsf{a}}\langle N\rangle,\ A\theta\rangle_s\rho_0$.

(LET_s) $\langle\sigma,\ \mathsf{let}\, y = \zeta\,\mathsf{in}\, A\rangle_s \xrightarrow{\theta}_s \langle\sigma\theta,\ A\theta[{}^{N}\!/y]\rangle_s$, where $\zeta\ \downarrow_\theta\ N$. By definition of symbolic frame, $\zeta\rho\ \downarrow\ N\rho_0$, for some ρ_0. Then, $\langle\sigma\rho,\ \mathsf{let}\, y = \zeta\rho\,\mathsf{in}\, A\rho\rangle \longrightarrow \langle\sigma\rho,\ A\rho[{}^{N\rho_0}\!/y]\rangle$ and $\langle\sigma\rho,\ A\rho[{}^{N\rho_0}\!/y]\rangle = \langle\sigma\theta,\ A\theta[{}^{N}\!/y]\rangle_s\rho_0$.

(MATCH_s) $\langle\sigma,\ [\zeta = \eta]A\rangle_s \xrightarrow{\theta}_s \langle\sigma\theta,\ A\theta\rangle_s$, with $\zeta\ \downarrow_{\theta_1}\ M$, $\eta\theta_1\ \downarrow_{\theta_2}\ N$, $\theta_3 = \mathrm{mgu}(M\theta_2, N)$, $N\theta_3 \in \mathcal{M}$ and $\theta = \theta_1\theta_2\theta_3$. By definition of symbolic frame, $\zeta\rho\ \downarrow\ M\rho_0'$, $\rho = \theta_1\rho_0'$. Also, $\eta\theta_1\ \downarrow_{\theta_2}\ N$ implies $\eta\theta_1{\rho_0}' = \eta\rho\ \downarrow\ N{\rho_0}''$, for some ${\rho_0}''$ such that ${\rho_0}' = \theta_2{\rho_0}''$. We need to prove that $M\rho_0' = N\rho_0''$, i.e. $M\theta_2\rho_0'' = N\rho_0''$. Since $\theta_3 = \mathrm{mgu}(M\theta_2, N)$, i.e. $M\theta_2\theta_3 = N\theta_3$, it is enough to prove that ${\rho_0}''$ is an instance of θ_3. But this follows by the hypothesis that $\rho = \theta\rho_0 = \theta_1\theta_2\theta_3\rho_0$ and $\rho = \theta_1\theta_2{\rho_0}''$.

□

Theorem 5.2.5. *Let $\mathcal{F}^s$ be a symbolic frame, $\mathcal{C}$ be a symbolic configuration and $\mathcal{C}\rho$ be a configuration. Then:*

(1) *$\mathcal{C}\rho \longrightarrow^n \mathcal{C}'$ implies that there exist $\mathcal{C}_1$, $\theta_1, \cdots, \theta_n$, ρ_0 such that $\mathcal{C} \xrightarrow{\theta_1}_s \cdots \xrightarrow{\theta_n}_s \mathcal{C}_1$ and $\mathcal{C}_1\rho_0 = \mathcal{C}'$.*

(2) *$\mathcal{C} \xrightarrow{\theta_1}_s \cdots \xrightarrow{\theta_n}_s \mathcal{C}_1$ and $\rho = \theta_1\cdots\theta_n\rho_0$, for some ρ_0 such that $\mathcal{C}_1\rho_0$ is a configuration, imply that $\mathcal{C}\rho \longrightarrow^n \mathcal{C}_1\rho_0$.*

Proof. The proof is by induction on the number n of execution steps. For $n = 1$, the theorem trivially follows by Lemma 5.2.4.

(1) Suppose $\mathcal{C}\rho \longrightarrow^n \mathcal{C}' \longrightarrow \mathcal{C}''$. By induction hypothesis, there exist $\mathcal{C}_1$, $\theta_1, \cdots, \theta_n$ and ρ_0 such that $\mathcal{C} \xrightarrow{\theta_1}_s \cdots \xrightarrow{\theta_n}_s \mathcal{C}_1$ and $\mathcal{C}_1\,\rho_0 = \mathcal{C}'$. Then, by applying Lemma 5.2.4 to $\mathcal{C}' = \mathcal{C}_1\,\rho_0$ and $\mathcal{C}''$, there exist $\mathcal{C}_2$, θ_{n+1} and ρ_0' such that $\mathcal{C}_1 \xrightarrow{\theta_{n+1}}_s \mathcal{C}_2$ and $\mathcal{C}'' = \mathcal{C}_2\rho_0'$. It follows that $\mathcal{C} \xrightarrow{\theta_1}_s \cdots \xrightarrow{\theta_n}_s \mathcal{C}_1 \xrightarrow{\theta_{n+1}}_s \mathcal{C}_2$, with $\mathcal{C}'' = \mathcal{C}_2\rho_0'$.

(2) Suppose $\mathcal{C} \xrightarrow{\theta_1}_s \cdots \xrightarrow{\theta_n}_s \mathcal{C}_1 \xrightarrow{\theta_{n+1}}_s \mathcal{C}_2$. Let $\rho_0' \stackrel{\text{def}}{=} \theta_{n+1}\rho_0$. $\mathcal{C}_2\rho_0$ is a configuration, hence also $\mathcal{C}_1\rho_0'$ is a configuration. By induction hypothesis, $\mathcal{C}\rho \longrightarrow^n \mathcal{C}_1\rho_0'$. Then, by applying Lemma 5.2.4 to $\mathcal{C}_1 \xrightarrow{\theta_{n+1}}_s \mathcal{C}_2$ and ρ_0', it follows that $\mathcal{C}_1\rho_0' \longrightarrow \mathcal{C}_2\rho_0$. Therefore, $\mathcal{C}\rho \longrightarrow^n \mathcal{C}_1\rho_0 \longrightarrow \mathcal{C}_2\rho_0'$.

□

Corollary 5.2.6 (concrete vs. symbolic semantics). *Let $\mathcal{F}^s$ be a symbolic frame, $\mathcal{C}$ be an initial configuration and s be a trace of $\mathcal{F}$. Then $\mathcal{C} \searrow s$ if and only if there exists σ such that $\mathcal{C} \searrow_s \sigma$ and s is a solution of σ.*

Proof. The proof easily follows by Theorem 5.2.5. In particular, for the 'if' direction, we exploit part (2) and note that $\mathcal{C}\rho = \mathcal{C}$, since $\mathcal{C}$ is ground. □

5.3 A Verification Method

In this section, we first define *regular frames*, i.e., frames for which it is possible to determine a finite *basis* for the synthesis of messages. Next, we introduce a *refinement* procedure that checks consistency of symbolic traces. Finally, we present a verification method which is based on refinement and applies to regular frames.

Regular frames It is convenient to extend the syntax of messages with a new class of variables to be used as placeholders for generic messages known to the environment. Formally, we consider a new set $\widehat{\mathcal{V}}$ of *marked* variables, in bijection with $\mathcal{V}$ via a mapping $\hat{\cdot}$; thus, variables $x, y, z, \ldots$ have marked counterparts $\hat{x}, \hat{y}, \hat{z}, \ldots$. Marked messages (resp., traces) are messages (resp., traces) that may also contain marked variables. Also, for $S \subseteq \mathcal{M}$, the set $\mathcal{H}(S)$ in Definition 5.1.2 is extended to include marked variables, i.e., we re-define $\mathcal{H}^0_{\mathcal{F}}(S)$ as follows:

$$\mathcal{H}^0_{\mathcal{F}}(S) \stackrel{\triangle}{=} S \cup \mathcal{EN} \cup \widehat{\mathcal{V}}.$$

The deduction relation ($S \vdash M$) remains formally unchanged. Note that in case S and M do not contain marked variables, this definition conservatively extends Definition 5.1.2. In practice, marked variables are treated as constants which are known to the environment. The satisfaction relation between marked symbolic traces and ground substitutions is extended according to this intuition. For any $\hat{x}$ and any trace σ, we denote by $\sigma\backslash\hat{x}$ the longest prefix of σ not containing $\hat{x}$.

Definition 5.3.1. *Let σ be a marked symbolic trace and ρ be a ground substitution. We say that ρ* satisfies *σ if $\sigma\rho$ is a trace and, for each $\hat{x} \in \mathrm{v}(\sigma)$, it holds that $(\sigma\backslash\hat{x})\rho \vdash \rho(\hat{x})$. We also say that $\sigma\rho$ is a* solution *of σ, and that σ is* consistent.

The terminology introduced above agrees with Definition 5.2.3 when σ does not contain marked variables. We give now the definition of solved form, that lifts the concept of trace to the non-ground case (note that this definition is formally the same as Definition 5.1.3).

Definition 5.3.2 (solved forms). *Let σ be a marked symbolic trace. We say σ is in* solved form (sf) *if for every σ_1, $\mathsf{a}\langle M\rangle$ and σ_2 s.t. $\sigma = \sigma_1 \cdot \mathsf{a}\langle M\rangle \cdot \sigma_2$ it holds that $\sigma_1 \vdash M$.*

Regular frames enjoy a "finite-basis" property. Basically, this property states the existence of a finite set containing the building blocks of all messages that the attacker can synthesise out of a given σ: this requirement is stated by Condition 1, below. Condition 2 is a technical requirement about substitutions.

Definition 5.3.3 (regular frames). *A symbolic frame $\mathcal{F}^s$ is* regular *if there exists a function $\mathbf{b} : Act^* \longrightarrow \mathcal{P}_f(\mathcal{M})$ such that, for each solved form σ of $\mathcal{F}^s$ and for all ρ that satisfy σ:*

1. *$\sigma\rho \vdash M$ if and only if $M \in \mathcal{H}(\mathbf{b}(\sigma\rho))$;*
2. *$\mathbf{b}(\sigma\rho) \subseteq \mathbf{b}(\sigma)\rho$.*

For each σ, $\mathbf{b}(\sigma)$ is said a basis *of σ.*

Refinement In the refinement procedure, input messages in a symbolic trace are tentatively unified to messages that can be synthesized from the basis. By iterating this procedure, one can check whether a given symbolic trace can eventually be instantiated to a trace in the concrete model. In particular, given any symbolic trace σ, and a basis for it, we can compute the set of the most general instances of σ that are in solved form, denoted by $\mathbf{SF}(\sigma)$.

Definition 5.3.4 (refinement and $\mathbf{SF}(\sigma)$). *We let* refinement, *written $\succ$, be the least binary relation over marked symbolic traces of a regular frame given by the following rules. In* (REF$_1$), *σ' is the longest prefix of σ that is in solved form and $\sigma = \sigma' \cdot \mathsf{a}\langle M\rangle \cdot \sigma''$, for some σ''. Assume $N, N' \notin \mathcal{V} \cup \widehat{\mathcal{V}}$.*

$$(\text{REF}_1)\ \frac{M = C[N] \quad N' \in \mathbf{b}(\sigma') \quad \theta = \mathrm{mgu}(N, N')}{\sigma \ \succ\ \sigma\theta\theta_0}$$

$$(\text{REF}_2)\ \frac{x \in \mathrm{v}(M)}{\sigma \ \succ\ \sigma[\hat{x}/x]}$$

where $\theta \neq \epsilon$, $\theta_0 \triangleq \{\, x/\hat{x} \mid \hat{x} \in \mathrm{v}(\sigma)$ and $|(\sigma\theta)\backslash\hat{x}| < |\sigma\backslash\hat{x}| \,\}$.

For any symbolic trace σ, we let $\mathbf{SF}(\sigma) \triangleq \{\, \sigma' \mid \sigma \ \succ * \sigma'$ and σ' is in sf $\}$.

Rule (REF$_1$) implements the basic step of refinement: the subcomponent N of M gets unified, via θ, with an element of $\mathbf{b}(\sigma')$. E.g., consider $\sigma = \bar{\mathsf{c}}\langle\{[a]\}_{k^+}\rangle \cdot \bar{\mathsf{c}}\langle\{[b]\}_{k^+}\rangle \cdot \mathsf{c}\langle\{[x]\}_{k^+}\rangle$: its possible refinements are $\sigma \succ \sigma[a/x]$ and $\sigma \succ \sigma[b/x]$, and the refined traces are in sf. By rule (REF$_2$) a variable can be marked: this amounts to constraining its possible values to be messages known by the environment. (For technical reasons, marked variables sometimes need to be 'unmarked' back to plain variables, and this is achieved in (REF$_1$) via the renaming θ_0.)

Note that $\mathbf{SF}(\sigma)$ can be effectively computed, and is always finite, as: (a) $\succ$ is a finitely-branching relation, and (b) infinite sequences of refinement steps cannot arise. As to the latter point, note that, since each (REF$_1$)-step eliminates at least one variable any sequence of refinement steps can contain only finitely many (REF$_1$)-steps, after the last of which rule (REF$_2$) can only be applied a finite number of times.

Below we prove that the solutions of a symbolic trace σ can be completely characterized in terms of the solutions of the sf in $\mathbf{SF}(\sigma)$. The proof is based on Lemma 5.3.6, which basically asserts that any consistent symbolic trace that is not sf can be further refined. This lemma relies in turn on Lemma 5.3.5 below, which states that any input message that violates the 'sf-ness' of a trace can be decomposed so to satisfy the premises of either of the refinement rules.

Lemma 5.3.5 (Context Lemma). *Let $\mathcal{F}^s$ be a regular frame and σ be a marked symbolic trace in sf of $\mathcal{F}^s$. If ρ satisfies $\sigma \cdot \mathsf{a}\langle M\rangle$ and $\sigma \nvdash M$, then, either:*

(i) $\exists\, x \in \mathrm{v}(M)$ *s.t.* $\sigma\rho \vdash \rho(x)$, *or*

(ii) $\exists\, C[\cdot], N$ *s.t.* $N \notin \mathbf{b}(\sigma) \cup \mathcal{V} \cup \hat{\mathcal{V}}$, $M = C[N]$, *and* $N\rho \in \mathbf{b}(\sigma)\rho$.

Proof. Since ρ satisfies $\sigma \cdot \mathsf{a}\langle M\rangle$, then $\sigma\rho \vdash M\rho$ and $\forall\, \hat{x} \in v(M)$, $\sigma\rho \vdash \rho(\hat{x})$. Since $\mathcal{F}$ is regular, $M\rho \in \mathcal{H}(\mathbf{b}(\sigma\rho))$. The proof is by induction on the least index i such that $M\rho \in \mathcal{H}^i(\mathbf{b}(\sigma\rho))$.

($i = 0$) $M\rho \in \mathbf{b}(\sigma\rho)$. Depending on the form of M, there are three cases:

- $M = \hat{x}$ or $M = \underline{a}$. These cases cannot arise, as they would imply $\sigma \vdash M$.
- $M = x$. Then, (i) holds.
- $M \notin \mathcal{V} \cup \hat{\mathcal{V}}$. $M \notin \mathbf{b}(\sigma)$ as, otherwise, it would follow $\sigma \vdash M$. Since $\mathcal{F}$ is regular, $\mathbf{b}(\sigma\rho) \subseteq \mathbf{b}(\sigma)\rho$, thus, there exists $N' \in \mathbf{b}(\sigma)$ such that $M\rho = N'\rho$. Then, we define $C \triangleq [\cdot]$, $N \triangleq M$. $N\rho \in \mathbf{b}(\sigma)\rho$, as $N' \in \mathbf{b}(\sigma)$ and $N\rho = N'\rho$; thus, (ii) holds.

($i > 0$) $M = f(\widetilde{M}')$, where $\widetilde{M}'\rho \subseteq \mathcal{H}^{i-1}(\sigma\rho)$. Note that, for some $M_i{}' \in \widetilde{M}'$, $\sigma \nvdash M_i{}'$, as otherwise $\sigma \vdash M$, by definition of $\mathcal{H}(\cdot)$. By induction hypothesis, either: (j) there exists $x \in \mathrm{v}(M_i{}')$ such that $\sigma\rho \vdash \rho(x)$ or (jj) there exist C' and N' s.t. $N' \notin \mathbf{b}(\sigma) \cup \mathcal{V} \cup \hat{\mathcal{V}}$, $M_i{}' = C'[N']$, and $N'\rho \in \mathbf{b}(\sigma)\rho$. Obviously (j) implies (i). If (jj) holds, then we choose $N \triangleq N'$ and $C[\cdot] \triangleq f(\widetilde{M}'_1, C'[\cdot], \widetilde{M}'_2)$, where $\widetilde{M}' = (\widetilde{M}'_1, M_i{}', \widetilde{M}'_2)$.

□

Lemma 5.3.6 (Progression Lemma). *Let $\mathcal{F}^s$ be a regular frame and σ be a marked symbolic trace in $\mathcal{F}$. Suppose that ρ satisfies σ. Than either σ is in sf, or there are σ' and ρ' such that $\sigma \succ \sigma'$, $\sigma\rho = \sigma'\rho'$, and ρ' satisfies σ'.*

Proof. Suppose that σ is *not* in sf and let σ_1 be the longest prefix of σ which is in sf. This means that $\sigma = \sigma_1 \cdot \mathsf{a}\langle M\rangle \cdot \sigma_2$, for some M s.t. $\sigma_1 \not\vdash M$. Since ρ satisfies σ, we must have $\sigma_1\rho \vdash M\rho$. By virtue of Lemma 5.3.5, either (i) there exists $x \in v(M)$ such that $\sigma_1\rho \vdash \rho(x)$, or (ii) there exist C, N, N' such that $N \notin \mathbf{b}(\sigma_1) \cup \mathcal{V} \cup \hat{\mathcal{V}}$, $M = C[N]$, $N' \in \mathbf{b}(\sigma_1)$ and $N'\rho = N\rho$.

If (i) holds, then we can apply (REF$_2$), define $\sigma' \stackrel{\triangle}{=} \sigma[\hat{x}/x]$ and $\rho' \stackrel{\triangle}{=} [x/\hat{x}]\rho$, and the thesis will follow.

If it is the case of (ii), there exists $\theta = \mathrm{mgu}(N, N')$ and ρ is an instance of θ as a substitution. By rule (REF$_1$), $\sigma \succ \sigma' \stackrel{\triangle}{=} \sigma\theta[\widetilde{y}/\widetilde{\hat{y}}]$, for an appropriate renaming $[\widetilde{y}/\widetilde{\hat{y}}]$. Furthermore, $\rho = \theta[\widetilde{y}/\widetilde{\hat{y}}]\rho'$, for some ground substitution ρ'. Also note that, thanks to the renaming $[\widetilde{y}/\widetilde{\hat{y}}]$, for each $\hat{x} \in \mathrm{v}(\sigma')$, $\sigma'\backslash\hat{x}$ is not longer than $\sigma\backslash\hat{x}$, and this guarantees that ρ' satisfies σ'. Thus we have found σ' and ρ' as required by the statement of the lemma. □

Theorem 5.3.7. *Let $\mathcal{F}^s$ be a regular frame, and let σ be a symbolic trace and s a trace of $\mathcal{F}^s$. Then s is a solution of σ if and only if s is a solution of some $\sigma' \in \mathbf{SF}(\sigma)$.*

Proof. Suppose $s = \sigma'\rho$ is a solution of σ': then obviously s is a solution of σ, as $\sigma' = \sigma\theta$, for some θ. On the converse, suppose $s = \sigma\rho$ is a solution of σ. By repeated application of the previous lemma, we find that there is σ' in solved form and ρ' s.t. that $\sigma \succ^\star \sigma'$, $s = \sigma\rho = \sigma'\rho'$ and ρ' satisfies σ'. Hence, by definition, s is a solution of σ'. □

Note that each solved form σ has solutions: a trivial one is obtained by mapping each variable to any name in $\mathcal{EN}$. This fact and the above theorem imply that a symbolic trace σ is consistent if and only if $\mathbf{SF}(\sigma) \neq \emptyset$.

The verification method The method $\mathbf{M}(\mathcal{C}, \alpha \hookleftarrow \beta)$ described below can be used to verify if $\mathcal{C} \models \alpha \hookleftarrow \beta$ or not. If the property is not satisfied, the method computes a trace violating the property, that is, an attack on $\mathcal{C}$.

```
    M(C, α ↩ β)
1.  compute Mod_C = {σ | C ↘s σ};
2.  foreach σ ∈ Mod_C do
        foreach action γ in σ do
            if ∃θ = mgu(β, γ) and ∃σ' = (σθθ') ∈ SF(σθ) s.t.
                    αθθ' does not occur prior to βθθ' in σ'
            then return(No, σ');
3.  return(Yes);
```

To understand how the method works, it is better to consider the simple case $\alpha = \bot$, i.e. verification of $\mathcal{C} \models \bot \hookleftarrow \beta$. This means verifying that, in the concrete semantics, no instance of action β is ever executed starting from $\mathcal{C}$. By the correspondence between symbolic and concrete semantics (Theorem 5.2.6), this amounts to checking that for each σ symbolically generated by $\mathcal{C}$, no solution of σ contains an instance of β. To verify this, the method proceeds as follows: for each such σ, and for each action γ in σ, it is checked whether there is a mgu θ for γ and β. If, for each σ, such a θ does not exist, or if it exists but $\sigma\theta$ is not consistent (i.e. $\mathbf{SF}(\sigma\theta) = \varnothing$, by the considerations following Theorem 5.3.7), then the property is true, otherwise it is not.

In practice, rather than generating the whole set of symbolic traces at once (step 1) and then checking the property, it is more convenient to work 'on-the-fly' and comparing every last symbolic action γ taken by the configuration against action β of the property $\alpha \hookleftarrow \beta$; the refinement procedure $\mathbf{SF}(\cdot)$ is invoked only when β and γ are unifiable.

The correctness of the method in the general case is stated by Theorem 5.3.8 below. Its proof relies on Corollary 5.2.6 and on Theorem 5.3.7, plus routine calculations on unifiers. Remark that the proof of part 1 of Theorem 5.3.8 (correctness) only relies on Corollary 5.2.6. Hence, the correctness of the method holds for the more general class of symbolic frames. Also, note that the method always terminates, considering that $\longrightarrow_{\mathrm{s}}$ is finite-branching.

Theorem 5.3.8 (correctness and completeness). *Let $\mathcal{F}^s$ be a regular frame, $\mathcal{C}$ be an initial configuration of $\mathcal{F}^s$ and α and β be actions of $\mathcal{F}^s$ such that* $\mathrm{v}(\alpha) \subseteq \mathrm{v}(\beta)$.

(1) *If $\mathbf{M}(\mathcal{C}, \alpha \hookleftarrow \beta)$ returns* (No, σ') *then $\mathcal{C} \not\models \alpha \hookleftarrow \beta$. In particular, for any injective ground substitution $\rho : \mathrm{v}(\sigma') \rightarrow \mathcal{EN}$, $\mathcal{C} \searrow (\sigma'\rho)$ and $(\sigma'\rho) \not\models \alpha \hookleftarrow \beta$.*

(2) *If $\mathcal{C} \not\models \alpha \hookleftarrow \beta$ then $\mathbf{M}(\mathcal{C}, \alpha \hookleftarrow \beta)$ returns* (No, σ') *and for any injective ground substitution $\rho : \mathrm{v}(\sigma') \rightarrow \mathcal{EN}$, $\mathcal{C} \searrow (\sigma'\rho)$ and $(\sigma'\rho) \not\models \alpha \hookleftarrow \beta$.*

Proof.

(1) According to the algorithm, there exist a trace σ, an action $\gamma \in \sigma$, a unifier $\theta = \mathrm{mgu}(\gamma, \beta)$ and θ' such that $\sigma' = \sigma\theta\theta' \in \mathbf{SF}(\sigma\theta)$ and $\mathcal{C} \searrow_{\mathrm{s}} \sigma$ and

$$\beta\theta\theta' \text{ occurs in } \sigma' \text{ but } \alpha\theta\theta' \text{does not occur prior to } \beta\theta\theta' \text{ in } \sigma'. \qquad (\star)$$

Given ρ as in the hypotheses, $s \stackrel{\triangle}{=} \sigma'\rho$ is a solution of σ and, by applying Corollary 5.2.6, it follows that $\mathcal{C} \searrow s$. Also, by $(\star)$ and by the injectivity of ρ on $\mathrm{v}(\sigma')$, $\alpha\theta\theta'\rho$ does not occur prior to $\beta\theta\theta'\rho$ in s, i.e. $s \not\models \alpha \hookleftarrow \beta$.

(2) Suppose $\mathcal{C} \searrow s$ and $s \not\models \alpha \hookleftarrow \beta$. By definition, there exists ρ_1 such that $\alpha\rho_1$ does not occur prior to $\beta\rho_1$ in s. Let i be the position of the leftmost

occurrence of $\beta\rho_1$ in s. By $\mathcal{C} \searrow s$ and by Corollary 5.2.6, it follows that there exists σ such that $\mathcal{C} \searrow_{\mathrm{s}} \sigma$ and s is a solution of σ, i.e. $s = \sigma\rho_2$, for some ρ_2. Since $\mathrm{v}(\alpha)$ and $\mathrm{v}(\beta)$ universally quantified, we can assume $\mathrm{v}(\alpha, \beta) \cap \mathrm{v}(\sigma) = \varnothing$ and so $\mathrm{dom}(\rho_1) \cap \mathrm{dom}(\rho_2) = \varnothing$. Let $\rho_0 \stackrel{\triangle}{=} \rho_1 \cup \rho_2$ and $\gamma \stackrel{\triangle}{=} \sigma[i]$; it follows that $\gamma\rho_0 = \beta\rho_0$. Thus, we can consider $\theta = \mathrm{mgu}(\beta, \gamma)$. By definition, ρ_0 is an instance of θ, i.e. $\rho_0 = \theta\rho_0'$, for some ρ_0'. Hence, $s = \sigma\theta\rho_0'$ is a solution of $\sigma\theta$. By Theorem 5.3.7, there exists a sf $\sigma' = \sigma\theta\theta' \in \mathbf{SF}(\sigma\theta)$ such that s is a solution of σ', i.e. $s = \sigma'\rho_0''$, for some ρ_0''. Necessarily

$$\alpha\theta\theta' \text{does not occur prior to } \gamma\theta\theta' \text{in } \sigma', \qquad (\star\star)$$

because otherwise $\alpha\theta\theta'\rho_0'' = \alpha\rho_1$ would occur prior to $\gamma\theta\theta'\rho_0'' = \beta\rho_1$, in contradiction to the hypotheses. Therefore, we have found σ, γ, θ, and σ' such that $\mathbf{M}(\mathcal{C}, \alpha \hookleftarrow \beta)$ returns $(\mathsf{No},\ \sigma')$. Finally, given ρ as in the hypotheses, $\sigma'\rho$ is a solution of σ. Thus, $\mathcal{C} \searrow (\sigma'\rho)$ by the fact that $\mathcal{C} \searrow_{\mathrm{s}} \sigma$ and by Corollary 5.3.7. Also, $(\sigma'\rho) \not\models \alpha \hookleftarrow \beta$ as $\beta\theta\theta'\rho$ ($= \gamma\theta\theta'\rho$) occurs in $\sigma'\rho$ at position i, but $\alpha\theta\theta'\rho$ does not occur in $\sigma'\rho$ prior to position i, by $(\star\star)$ and by the injectivity of ρ.

□

5.4 Example: Public Key Encryption

In this section we illustrate an example of application of the approach introduced in the previous sections. We focus on a system based on public-key cryptography. In particular, in the system we consider primitives for pairing and public key encryption of messages can be arbitrarily nested, but such that non-atomic keys are forbidden. The frame $\mathcal{F}_{pk} = (\Sigma, \mathcal{M}, \downarrow)$ is defined below.

$$\Sigma \quad = \quad \{(\cdot)^+,\quad (\cdot)^-,\quad \{[\cdot]\}_{(\cdot)},\quad \langle\cdot,\cdot\rangle,\quad \pi_i(\cdot)\quad (i=1,2),\quad \mathsf{dec}^{\mathsf{pk}}_{(\cdot)}(\cdot)\} \qquad \text{Signature}$$

$$M, N \quad ::= \quad u \ \mid\ u^+ \ \mid\ u^- \ \mid\ \{[M]\}_{u^+} \ \mid\ \langle M, N\rangle \qquad \text{Messages}$$

$$\pi_i(\langle M_1, M_2\rangle) \ \rightsquigarrow\ M_i \quad (i=1,2)$$

$$\mathsf{dec}^{\mathsf{pk}}_{u^-}(\{[M]\}_{u^+}) \ \rightsquigarrow\ M$$

$$\frac{\zeta \rightsquigarrow \zeta'}{C[\zeta] \rightsquigarrow C[\zeta']} \qquad \zeta \downarrow M \ \stackrel{\triangle}{\Leftrightarrow}\ \zeta \rightsquigarrow^* M \qquad \text{Evaluation}$$

The functions of Σ are: generation of public $((\cdot)^+)$ and private $((\cdot)^-)$ keys, encryption with a public key $(\{[\cdot]\}_{(\cdot)})$, decryption using a private key $(\mathsf{dec}^{\mathsf{pk}}_{(\cdot)}(\cdot))$, pairing $(\langle\cdot,\cdot\rangle)$ and selection $(\pi_i(\cdot))$. Public and private keys are represented by u^+ and u^-, respectively. Names and variables can be used to build compound messages via public-key encryption and pairing. In particular, $\{[M]\}_{m^+}$ represents the message obtained by encrypting M under m^+. The definition of evaluation relation makes use of an auxiliary relation $\rightsquigarrow$, that models the mechanisms of public key encryption under the perfect cryptography assumption. Note that the codomain of the evaluation relation $\downarrow$ is restricted to messages.

As an example of deducible message in $\mathcal{F}_{pk}$, if $S = \{\ \{[\langle a,b\rangle]\}_{k^+}, k^-\ \}$ then $S \vdash a$, since $\zeta = \pi_1(\mathsf{dec}^{\mathsf{pk}}_{k^-}(\{[\langle a,b\rangle]\}_{k^+})) \in \mathcal{H}(S)$ and $\zeta \downarrow a$. Note that, whatever S, the set of messages deducible from S is infinite.

The symbolic frame $\mathcal{F}^s_{pk}$ is defined as the pair $(\mathcal{F}_{pk}, \downarrow_s)$, where $\downarrow_s$ is the reflexive and transitive closure of the relation $(\rightsquigarrow_s)$, as given below.

$$\mathsf{dec}^{\mathsf{pk}}_N(M) \overset{\theta\setminus\{x_1,x_2\}}{\rightsquigarrow_s} x_1\,\theta \qquad\qquad \theta = \mathrm{mgu}(M = \{[x_1]\}_{x_2^+}, N = x_2^-)$$

$$\pi_i(M) \overset{\theta\setminus\{x_1,x_2\}}{\rightsquigarrow_s} x_i\,\theta \qquad (i = 1,2) \quad \theta = \mathrm{mgu}(M, \langle x_1, x_2\rangle)$$

$$\{[M]\}_x \overset{\theta}{\rightsquigarrow_s} \{[M\theta]\}_{x^+} \qquad\qquad \theta = [x^+/x]$$

$$\frac{\zeta \overset{\theta}{\rightsquigarrow_s} \zeta'}{C[\zeta] \overset{\theta}{\rightsquigarrow_s} C\theta[\zeta']}$$

$$\zeta \downarrow_\theta M \quad \overset{\triangle}{\Leftrightarrow} \quad \zeta \overset{\theta_1}{\rightsquigarrow_s} \cdots \overset{\theta_n}{\rightsquigarrow_s} M \ \text{ and } \ \theta = \theta_1\cdots\theta_n$$

Variables x_1 and x_2 are chosen fresh according to some arbitrary but fixed rule.

$\mathcal{F}^s_{pk}$ is indeed a symbolic frame. The proof is based on the lemma below. We adopt the following notation: $\theta \setminus V$ means $\theta \setminus (V \times \mathcal{E}_\Sigma)$.

Lemma 5.4.1. *Suppose $\rightsquigarrow$ and $\rightsquigarrow_s$ are the evaluation relations defined for $\mathcal{F}^s_{pk}$ Then:*

(1) *$\zeta\rho \rightsquigarrow \eta$ implies that there exist η', θ and ρ_0 such that $\zeta \overset{\theta}{\rightsquigarrow_s} \eta'$, $\rho = \theta\rho_0$ and $\eta = \eta'\rho_0$. Furthermore, $\eta \in \mathcal{M}$ implies $\eta' \in \mathcal{M}$.*

(2) *$\zeta \overset{\theta}{\rightsquigarrow_s} \eta$ and $\rho = \theta\rho_0$, for some ρ_0, imply that $\zeta\rho \rightsquigarrow^* \eta\rho_0$.*

Proof.

(1) By structural induction on the rules of $\rightsquigarrow$.

$\zeta\rho = \pi_i(\langle M_1\rho, M_2\rho\rangle) \rightsquigarrow M_i\rho$. By the rules of $\rightsquigarrow_s$, $\zeta = \pi_i(\langle M_1, M_2\rangle) \overset{\theta}{\rightsquigarrow}_s x_i$, with $\theta = \text{mgu}(\langle M_1, M_2\rangle, \langle x_1, x_2\rangle)$. The thesis follows by taking $\eta' = x_i\theta$ and ρ_0 such that $\rho = \theta\rho_0$.

$\zeta\rho = \mathsf{dec}^{\mathsf{pk}}_{u^-\rho}(\{[M\rho]\}_{v^+\rho})$, with $u\rho = v\rho$. By the rules $\rightsquigarrow_s$, $\zeta = \mathsf{dec}^{\mathsf{pk}}_{u^-}(\{[M]\}_{v^+})$ $\overset{\theta}{\rightsquigarrow}_s x_1$, with $\theta = \text{mgu}(\{[M]\}_{v^+} = \{[x_1]\}_{x_2^+}, u = x_2)$. We take $\eta' = x_1\theta$ and ρ_0 such that $\rho = \theta\rho_0$. Thus $\eta'\rho_0 = M\rho$, as $(x_1\theta)\rho_0 = M\theta\rho_0 = M\rho$.

$\dfrac{\zeta\rho \rightsquigarrow \zeta'}{C[\zeta\rho] \rightsquigarrow C[\zeta']}$. By induction hypothesis on $\zeta\rho$, there exists η' and ρ such that $\zeta \overset{\theta}{\rightsquigarrow}_s \eta'$ and $\eta'\rho_0 = \zeta'$. By applying rules of $\rightsquigarrow_s$, $C[\zeta] \overset{\theta}{\rightsquigarrow}_s C\theta[\eta']$. The thesis follows by the fact that $(C\theta[\eta'])\rho_0 = C\rho[\eta'\rho] = C\rho[\zeta']$.

(2) By structural induction on the rules $\rightsquigarrow_s$.

$\{[M]\}_x \overset{\theta}{\rightsquigarrow}_s \{[M\theta]\}_{x^+}$, with $\theta = [x^+/x]$. We prove that $\{[M\rho]\}_{x\rho} \rightsquigarrow^* \{[M\theta]\}_{x^+}\rho_0$ in 0 steps. And, indeed, $\{[M\theta]\}_{x^+}\rho_0 = \{[M\theta]\}_{\theta(x)}\rho_0 = \{[M\rho]\}_{\rho(x)}$.

$\pi_i(M) \overset{\theta}{\rightsquigarrow}_s x_i\theta$, with $\theta = \text{mgu}(M, \langle x_1, x_2\rangle)$. By definition of θ, $\pi_i(M\rho) = \pi_i(M\theta\rho_0) = \pi_i(\langle x_1\theta\rho_0, x_2\theta\rho_0\rangle)$ and, by applying rules of $\rightsquigarrow$, $\pi_i(\langle x_1\theta\rho_0, x_2\theta\rho_0\rangle) \rightsquigarrow x_i\theta\rho_0$.

$\mathsf{dec}^{\mathsf{pk}}_N(M) \overset{\theta}{\rightsquigarrow}_s x_1\theta$, with $\theta = \text{mgu}(M = \{[x_1]\}_{x_2^+}, N = x_2^-)$. By definition of θ, $\mathsf{dec}^{\mathsf{pk}}_{N\rho}(M\rho) = \mathsf{dec}^{\mathsf{pk}}_{N\theta\rho_0}(M\theta\rho_0) = \mathsf{dec}^{\mathsf{pk}}_{x_2^-\theta\rho_0}(\{[x_1\theta\rho_0]\}_{x_2^+\theta\rho_0})$. Finally, by applying rules of $\rightsquigarrow$, $\mathsf{dec}^{\mathsf{pk}}_{x_2^-\theta\rho_0}(\{[x_1\theta\rho_0]\}_{x_2^+\theta\rho_0}) \rightsquigarrow x_1\theta\rho_0$.

$\dfrac{\zeta \overset{\theta}{\rightsquigarrow}_s \zeta'}{C[\zeta] \overset{\theta}{\rightsquigarrow}_s C\theta[\zeta']}$. By induction hypothesis, $\zeta\rho \rightsquigarrow^* \zeta'\rho_0$ and, by rules of $\rightsquigarrow$, $C\rho[\zeta\rho] \rightsquigarrow^* C\rho[\zeta'\rho_0]$, i.e. $(C[\zeta])\rho \rightsquigarrow^* (C\theta[\zeta'])\rho_0$.

$\square$

Proposition 5.4.2. *$\mathcal{F}^s_{pk}$ is a symbolic frame.*

Proof. We need to prove Conditions (a) and (b) of Definition 5.2.1.

(a) By induction on the number n of steps of relation $\rightsquigarrow$, such that $\zeta\rho \rightsquigarrow^n M$. The case $n = 0$ is trivial. Suppose $\zeta\rho \rightsquigarrow^n \eta \rightsquigarrow M$. By induction hypothesis, there exist η', θ, ρ_0 such that $\zeta \downarrow_\theta \eta'$, and $\eta = \eta'\rho_0$. Then, by applying Lemma 5.4.1 to $\eta'\rho$ and M, there exist N, θ', and ρ_0' such that $\eta' \downarrow_{\theta'} N$ and $N\rho_0' = M$. It follows that $\zeta \downarrow_{\theta\theta'} N$ and $M = N\rho_0'$.

(b) By induction on the number n of steps of relation $\leadsto_s$, i.e. such that $\zeta \overset{\theta_1}{\leadsto}_s \zeta_1 \overset{\theta_2}{\leadsto}_s \cdots \overset{\theta_n}{\leadsto}_s N$, with $\rho = \theta_1\theta_2\cdots\theta_n\rho_0$, for some ρ_0. The case $n = 0$ is trivial. Suppose $\zeta \overset{\theta_1}{\leadsto}_s \zeta_1 \overset{\theta_2}{\leadsto}_s \cdots \overset{\theta_n}{\leadsto}_s \zeta_n \overset{\theta_{n+1}}{\leadsto}_s N$. By induction hypothesis, $\zeta\rho \downarrow \zeta_n\rho_0$, for some ρ_0 such that $\rho = \theta_1\cdots\theta_n\rho_0$. By applying Lemma 5.4.1 to $\zeta_n\rho_0$, it follows that $\zeta\rho \leadsto^* \zeta_n\rho_0 \leadsto N\rho_0'$, for some ρ_0'.

□

We now prove that $\mathcal{F}^s_{pk}$ is a regular frame. We specify a function $\mathbf{b}_{pk}$ and show that it is indeed a basis. Propositions 5.4.4 and 5.4.8 below state that $\mathbf{b}_{pk}$ satisfies respectively Conditions 1 and 2 of Definition 5.3.3.

Notation By $M = (u)^\pm$ we mean $M = u$ or $M = u^+$ or $M = u^-$.

Definition 5.4.3 (function $\mathbf{b}_{pk}$).

$$\mathbf{b}_{pk}(\sigma) \stackrel{\triangle}{=} \{\, M \mid (\sigma \vdash M) \textit{ and } (M = (u)^\pm, \textit{ for some } u \in \mathcal{LN} \cup \mathcal{V}, \textit{ or } \exists\, N, u \;:\; M = \{\![N]\!\}_{u^+} \textit{ and } \sigma \nvdash \langle N, u^+\rangle \;\;)\;\; \}.$$

In practice, for a given σ, the set $\mathbf{b}_{pk}(\sigma)$ can be effectively computed by an iterative procedure, which repeatedly applies destructors ($\mathsf{dec}^{\mathsf{pk}}_{(\cdot)}(\cdot)$ and $\pi_i(\cdot)$) to messages in σ, until some fixed point is reached. This procedure always terminates.

Proposition 5.4.4. *Let σ be a sf in $\mathcal{F}^s_{pk}$ and ρ be a substitution that satisfies σ. Then, $\sigma\rho \vdash M$ iff $M \in \mathcal{H}(\mathbf{b}_{pk}(\sigma\rho))$.*

Proof. Suppose $\sigma\rho \vdash M$. By induction on the structure of M.

- $M \in \mathcal{EN}$. Then, $M \in \mathcal{H}^0(\mathbf{b}_{pk}(\sigma\rho))$.
- $M = (\hat{x})^\pm$. Then, $M \in \mathcal{H}^1(\mathbf{b}_{pk}(\sigma\rho))$.
- $M = (m)^\pm$. Then, $(m)^\pm \in \mathbf{b}_{pk}(\sigma\rho)$, by definition.
- $M = \langle M_1, M_2\rangle$. By induction hypothesis M_1, $M_2 \in \mathcal{H}(\mathbf{b}_{pk}(\sigma\rho))$ and, consequently, $M \in \mathcal{H}(\sigma\rho)$.
- $M = \{\![M']\!\}_{u^+}$. If $\sigma\rho \nvdash \langle M', u^+\rangle$, $M \in \mathbf{b}_{pk}(\sigma\rho)$, by definition. Otherwise, by induction hypothesis, u^+, $M' \in \mathcal{H}(\mathbf{b}_{pk}(\sigma\rho))$ and, thus, $\{\![M']\!\}_{u^+} = M \in \mathcal{H}(\mathbf{b}_{pk}(\sigma\rho))$.

On the other hand, suppose $M \in \mathcal{H}(\mathbf{b}_{pk}(\sigma\rho))$. By induction on the minimum j such that $M \in \mathcal{H}^j(\mathbf{b}_{pk}(\sigma\rho))$.

($j = 0$) There are two cases. If $M \in \widehat{\mathcal{V}} \cup \mathcal{EN}$, trivially $\sigma\rho \vdash M$. Else $M \in \mathbf{b}_{pk}(\sigma\rho)$ and it follows by definition that $\sigma\rho \vdash M$.

$(j > 0)$ Suppose $M = \{[M']\}_{u^+}$ (the case $M = \langle M_1, M_2 \rangle$ is analogous), with M', $u^+ \in \mathcal{H}^{j-1}(\mathbf{b}_{pk}(\sigma\rho))$. Then, the thesis follows by induction hypothesis on M' and u^+.

□

The proof of Proposition 5.4.8 relies on the two technical lemmata below.

Lemma 5.4.5. *Let σ be in sf and ρ be a substitution that satisfies σ. If $\sigma \vdash M$, then:*

(1) $\mathrm{v}(M) \subseteq \widehat{\mathcal{V}}$.

(2) $\sigma\rho \vdash M\rho$.

Proof. Let σ_0 be the shortest prefix of σ such that $\sigma_0 \vdash M$. The proof is by induction on $|\sigma_0|$. The case $|\sigma_0| = 0$ is obvious. If $|\sigma_0| > 0$, we proceed by induction on the minimum index j such that there exists $\eta \in \mathcal{H}^j(\sigma_0)$, with $\eta \downarrow M$.

$(j = 0)$ Necessarily $\eta = M$. The case $M \in \widehat{\mathcal{V}} \cup \mathcal{EN}$ is obvious. If $M \in \sigma_0$ then $M\rho \in \sigma_0\rho$ and thus $\sigma_0\rho \vdash M\rho$, i.e. (2). We now prove (1). It is not the case that M is an input message, i.e. $\sigma_0 = \sigma_1 \cdot \mathsf{a}\langle M \rangle \cdot \sigma_2$, as it would imply $\sigma_1 \vdash M$ (by definition of sf), in contradiction to the hypothesis on σ_0. Then necessarily M is an output message, i.e. $\sigma_0 = \sigma_1 \cdot \bar{\mathsf{a}}\langle M \rangle \cdot \sigma_2$. By definition of symbolic trace, it follows that $\mathrm{v}(M) \cap \mathcal{V} \subseteq \mathrm{v}(\sigma_1) \cap \mathcal{V}$. By induction hypothesis on σ_1, $\mathrm{v}(\sigma_1) \cap \mathcal{V} = \varnothing$ (otherwise, (1) would be violated for σ_1) and, thus, $\mathrm{v}(M) \subseteq \widehat{\mathcal{V}}$.

$(j > 0)$ Suppose $\eta = \mathsf{dec}^{\mathsf{pk}}_{\zeta_2}(\zeta_1)$ with $\zeta_1 \downarrow \{[M]\}_{u^+}$ and $\zeta_2 \downarrow u^-$. By induction hypothesis, $\mathrm{v}(\{[M]\}_{u^+}, u^-) \subseteq \widehat{\mathcal{V}}$, which implies (1) for M. Also, by induction hypothesis, $\sigma_0\rho \vdash \{[M\rho]\}_{(u^+\rho)}$ and $\sigma_0\rho \vdash (u^-)\rho$. It follows that $\sigma_0\rho \vdash M\rho$. The other cases ($\eta = \zeta^+$, $\eta = \zeta^-$, $\eta = \langle \zeta_1, \zeta_2 \rangle$, $\eta = \pi_i(\zeta)$ are similar).

□

Remark 5.4.6. 1. Let s be a trace and M a message such that $s \vdash M$. Then, $\mathrm{v}(M) \subseteq \widehat{\mathcal{V}}$.

2. Let σ be a sf in $\mathcal{F}^s_{pk}$. Then $\mathrm{v}(\sigma) \subseteq \widehat{\mathcal{V}}$. This fact trivially follows by Lemma 5.4.5.

Lemma 5.4.7. *Let σ be in sf, ρ satisfies σ and $A = \{m, m^+, m^- \mid m \in \mathcal{N}\}$. If $\sigma\rho \vdash M$, M ground, then there exists N, with $\mathrm{v}(N) \subseteq \mathrm{v}(\sigma)$ such that $\sigma \vdash N$ and $N\rho = M$. Also, if $M \in A$ then $N = M$.*

Proof. Let σ_0 be the shortest prefix of σ such that $\sigma_0\rho \vdash M$. The proof is by induction on $|\sigma_0|$. If $|\sigma_0| = 0$, we take $N \triangleq M$. And, indeed, $\sigma_0 \vdash N$ since $\mathrm{n}(M) \subseteq \mathcal{EN}$. If $|\sigma_0| > 0$, we proceed by induction on the minimum index j such that there exists $\eta \in \mathcal{H}^j(\sigma_0\rho)$, with $\eta \downarrow M$.

$(j = 0)$ The case $M \in \mathcal{EN}$ is obvious. If $M \in \sigma_0\rho$, there exists $N \in \sigma_0$ such that $N\rho = M$ and obviously $\mathrm{v}(N) \subseteq \mathrm{v}(\sigma)$. Furthermore, it is not the case that $N = (\hat{x})^{\pm}$, as it would imply $(\sigma_0 \setminus \hat{x})\rho \vdash \rho(\hat{x})^{\pm} = M$, with $\sigma_0 \setminus \hat{x}$ shorter than σ_0, in contradiction to the hypothesis on σ_0. Also, this is not the case that $N = (x)^{\pm}$, as $\mathrm{v}(\sigma_0) \subseteq \widehat{\mathcal{V}}$, by Observation 5.4.6. It follows that $M = (m)^{\pm} \in A$ implies $N = M$, since $N\rho = M$.

$(j > 0)$ There are different cases, depending on the structure of η. Here we consider the only non-trivial case, i.e. $\eta = \mathsf{dec}^{\mathsf{pk}}_{\zeta_2}(\zeta_1)$, with $\zeta_1 \downarrow \{[M]\}_{m^+}$ and $\zeta_2 \downarrow m^-$. (Note that we can assume ζ_1 and ζ_2 be ground and thus evaluate to messages.) We need to prove that there exists $N' = \{[N]\}_{u^+}$, for some N, such that $\mathrm{v}(N') \subseteq \mathrm{v}(\sigma_0)$, $\sigma_0 \vdash N'$ and $N'\rho = \{[M]\}_{m^+}$. Let σ_0' be the shortest prefix of σ_0 such that $\sigma_0'\rho \vdash \{[M]\}_{m^+}$. By external induction hypothesis if $|\sigma_0'| < |\sigma_0|$ or by internal induction hypothesis (on $\zeta_1 \downarrow \{[M]\}_{m^+}$) if $|\sigma_0'| = |\sigma_0|$, it follows that there exists N' such that $\mathrm{v}(N') \subseteq \mathrm{v}(\sigma_0')$, $\sigma_0' \vdash N'$ and $N'\rho = \{[M]\}_{m^+}$. It cannot be $N' = \hat{x}$, as it would otherwise imply $(\sigma_0' \setminus \hat{x}) \vdash \rho(\hat{x}) = \{[M]\}_{m^+}$, in contradiction to the hypothesis on σ_0' ($\sigma_0' \setminus \hat{x}$ shorter than σ_0'). Therefore, necessarily $N' = \{[N]\}_{u^+}$, for some N and u and obviously $\mathrm{v}(N) \subseteq \mathrm{v}(\sigma_0')$. Also, if $u^+ = \hat{y}^+$, for some $\hat{y} \in \widehat{\mathcal{V}}$, trivially $\sigma_0 \vdash N$. Otherwise, necessarily $u = m$, since $u\rho = m$ and $\mathrm{v}(N') \subseteq \mathrm{v}(\sigma_0) \subseteq \widehat{\mathcal{V}}$, i.e. $u \notin \mathcal{V}$. Therefore, we can apply the internal induction hypothesis to $\zeta_2 \downarrow m^-$, thus obtaining $\sigma_0 \vdash m^-$. Since also $\sigma_0 \vdash \{[N]\}_{m^+}$, it follows that $\sigma_0 \vdash N$. Finally, we prove that if $M \in A$, then $N = M$. Indeed, if it were $N = (\hat{y})^{\pm}$ with $\hat{y} \in \mathrm{v}(\sigma_0)$, it would also be $(\sigma_0 \setminus \hat{y})\rho \vdash \rho(\hat{y})^{\pm} = M$, in contradiction to the minimality of σ_0.

□

Proposition 5.4.8. *Let σ be in sf in $\mathcal{F}^s_{pk}$. Then, $\mathbf{b}_{pk}(\sigma\rho) \subseteq \mathbf{b}_{pk}(\sigma)\rho$, for any ρ that satisfies σ.*

Proof. Suppose $M \in \mathbf{b}_{pk}(\sigma\rho)$, M ground. We have to prove that there exists $N \in \mathbf{b}_{pk}(\sigma)$ such that $N\rho = M$. Let σ_0 be the shortest prefix of σ such that $\sigma_0\rho \vdash M$. Clearly, $M \in \mathbf{b}_{pk}(\sigma_0\rho)$. We distinguish two cases, depending on the structure of M.

- $M = (m)^{\pm}$. Then $\sigma_0 \vdash (m)^{\pm}$ by Lemma 5.4.7 and obviously $\sigma \vdash m^+$. In this case, we take $N \stackrel{\triangle}{=} (m)^{\pm} \in \mathbf{b}_{pk}(\sigma)$.
- $M = \{[M']\}_{m^+}$, with $\sigma\rho \nvdash \langle M', m^+\rangle$, by Lemma 5.4.7 there exists N such that $\sigma_0 \vdash N$, $\mathrm{v}(N) \subseteq \mathrm{v}(\sigma_0)$ and $N\rho = M$. By the hypotheses on σ_0, $N \neq \hat{x}$ and so $N = \{[N']\}_{u^+}$. Necessarily, $\sigma \nvdash \langle N', u^+\rangle$, as it would otherwise follow by Lemma 5.4.5 that $\sigma\rho \vdash \langle N'\rho, u^+\rho\rangle = \langle M', m^+\rangle$, in contradiction to the hypothesis that $M \in \mathbf{b}_{pk}(\sigma\rho)$. Therefore, $N \in \mathbf{b}_{pk}(\sigma)$ and $N\rho = M$.

□

Theorem 5.4.9. *$\mathcal{F}^s_{pk}$ is a regular frame.*

Proof. The theorem follows by Proposition 5.4.4 and Proposition 5.4.8. □

5.5 Other Cryptographic Primitives

We now consider extending the symbolic frame $\mathcal{F}^s_{pk}$, introduced in Section 5.4 to deal with some of the most common cryptographic operations.

The set Σ is enriched by means of appropriate operators for shared-key encryption $\{\cdot\}_{(\cdot)}$ and decryption $\mathsf{dec}^{\mathsf{sk}}_{(\cdot)}(\cdot)$, digital signing $[\![\{\cdot\}]\!]_{(\cdot)}$ and verifying $\mathsf{dec}^{\mathsf{ds}}_{(\cdot)}(\cdot)$ and hashing $H(\cdot)$. The syntax of messages is extended via the following additional clauses:

$$\begin{array}{lcll} M, N & ::= & \ldots & \text{as in Table 5.1} \\ & | & \{M\}_u \;\;|\;\; [\![\{M\}]\!]_{u^-} \;\;\Big|\;\; H(M)\,. & \end{array}$$

The symbolic and concrete evaluations are given in terms of an auxiliary relation $\rightsquigarrow$, defined as expected. In particular, hashing has no rules, digital signature rules are just the same as for public key, but the roles of u^+ and u^- are swapped. For shared key, the concrete and symbolic rules are given below:

$$\mathsf{dec}^{\mathsf{sk}}_u(\{M\}_u) \rightsquigarrow M \qquad \mathsf{dec}^{\mathsf{sk}}_v(\{M\}_u) \overset{\theta}{\rightsquigarrow}_s M\theta \quad \text{where } \theta = \mathrm{mgu}(u, v)\,.$$

A basis for this frame can be given by including into the basis given for public key also all messages of the form $\{M\}_u$ (resp. $[\![\{M\}]\!]_{u^-}$, $H(M)$) s.t. $\sigma \not\vdash u$ (resp. $\sigma \not\vdash \langle M, u^-\rangle$, $\sigma \not\vdash M$).

5.6 Related Works

Early work on symbolic analysis is due to Huima. In [39], the execution of a protocol generates a set of equational constraints. Only an informal description is provided of the kind of equational rewriting needed to solve these constraints. More recent approaches based on symbolic analysis are exploited in [6, 29]. The paper [6] extends the symbolic reachability analysis of [5] to hashing functions and public key cryptography and establishes some results on the complexity of the problem. Unlike our approach, symbolic execution and consistency check are not kept separate, and this may have a relevant impact on the size of the computed symbolic model. Another point worth noting is that, in [6], a brute-force method is needed to resolve variables in key position: such variables have to be instantiated to every possible name used by the participants; this fact may lead to state explosion, too.

In [29], a procedure is provided to analyze the knowledge of the environment, based on a symbolic semantics akin to [10]. The approach applies to protocols with shared-key encryption and arbitrary messages as keys, but, like ours, it is proven complete only for atomic keys. Also, the method suffers from the same problem as [6] concerning brute-force instantiation.

Other recent developments of the symbolic approach are presented in [20] and [50]. Neither of them relies on unification to build the symbolic model. The decision technique in [20] is based on a reduction to a set constraint problem which is in turn reduced to an automata-theoretic problem. Completeness is proven by imposing rather severe restrictions upon protocol syntax. The technique in [50] focuses on reachability properties and is based on constraint solving. The symbolic reduction and the knowledge analysis are separated and the latter is performed by a procedure for solving a system of constraints.

Chapter
6

STA, a Tool for Automatic Analysis of Security Protocols

Symbolic analysis presents advantages with respect to finite-state model checking in the verification of security protocols, as mentioned above. Indeed, the main goal of the method we described in the previous chapter is to avoid explicit construction of the whole state-space of the protocol in analysis. This results in:

- More accurate protocol models. In particular, we require a bounded number of protocol runs, but we do not restrict the number of possible messages the intruder can generate and send to honest participants at any moment. The whole, possibly infinite, protocol model can be searched for attacks.

- Increased efficiency, both in terms of memory occupation and execution time. In particular, state-explosion induced by message exchange between participants is avoided.

- More straightforward protocol modelling. In particular, there is no need for carefully crafted message types or bounds, and no explicit description of the adversary must be provided.

The present chapter introduces STA, a prototype tool implementing the symbolic method described in Chapter 5. We illustrate the significance of the above listed features in practice. We adopt STA to analyze two well-known protocols, i.e. the asymmetric Needham-Schroeder protocol [57] and Kerberos [41], in order to explain the STA's model and specification method, commenting on the results of the analysis and contrasting them with those obtained using finite-state model checkers.

Structure of the chapter. In Section 6.1 we briefly outline the syntax of STA's specifications. Sections 6.2 and 6.3 respectively contain our results of the STA analysis of the Needham-Schroeder and Kerberos protocols. Finally, in section 6.4 we discuss STA's main features, contrasted with finite-state model-checkers.

6.1 STA Specifications

As mentioned above, protocol specification in STA follows closely the syntax and semantics of the formal model, with a few minor differences. Concerning cryptographic operations, shared-key encryption of `X` with `K` is written `{X}K`. Asymmetric encryption is written `(X)^+K`. Here, `+K` is the encryption key; the corresponding decryption key is written `-K`. Asymmetric encryption can be used to model both public-key cryptography (if `+K` is public and `-K` is kept private) and digital signature (if `+K` is kept private and `-K` is public). Hashing of `X` is written as `H(X)`. For process syntax, we have:

- output actions are written as `a!M`;
- input actions are written as `a?M`. Here, `M` can be a generic message pattern, with variables. This means input of any adversary-generated message whose form matches up `M`.
- sequencing of actions is written `>>`;
- testing for equality of two messages `M` and `N` is written `(M is N)`. Operationally `(M is N) >> P` means 'unify `M` and `N`, if possible, and then proceed like `P`, otherwise stop'. Test for syntactic equality of two messages is a special case of this operation.
- parallel composition of `P` and `Q` is written `P||Q`.

There are a few more process operations (non-deterministic choice and a facility for generating fresh names), but those listed above are enough to illustrate our approach. Configurations are written as `(L @ P)`, where `L` is a list of input/output actions, which typically provides the adversary with its initial knowledge. Finally, properties are written like `A <-- B`, with `A` and `B` being actions.

It is worthwhile to note that, in the formal model, an input pattern can be rendered as an input action followed by an appropriate sequence of decryptions and/or equality tests. For example, reception of a piece of information encrypted with `+K` (i.e., reception of anything that can be correctly decrypted using the key `-K`) is written in the formal model as $\mathsf{a}(y).\mathsf{case}\ y\ \mathsf{of}\ \{x\}_{-K}\ \mathsf{in}\ P$. In STA, this can be simply written as `a?(x)^+K >> P`. STA provides explicit shared- and asymmetric-key decryption operations, but they will not be necessary in the examples taken into account throughout the chapter.

6.2 Example: Needham-Schroeder

We consider here two versions of the asymmetric Needham-Schroeder protocol. The first one is a simplified version (the same considered e.g. in [44, 54]), that leaves out the initial steps necessary to distribute the participants' public keys. The second version considers these steps too. The extended version is interesting because it exhibits a type-dependent flaw (which was well-known, though).

6.2.1 Basic Needham-Schroeder

Below, we give the informal description of the protocol. A acts as the initiator and B as the responder. The notation $\{X\}_{+KY}$ denotes encryption of X with the public key of Y. As the protocol begins, $+KA$ and $+KB$ are assumed to be already known by any participant.

$$
\begin{array}{llll}
1. & A \longrightarrow B: & \{NA, A\}_{+KB} & (NA \text{ fresh nonce}) \\
2. & B \longrightarrow A: & \{NA, NB\}_{+KA} & (NB \text{ fresh nonce}) \\
3. & A \longrightarrow B: & \{NB\}_{+KB} &
\end{array}
$$

There is a well-known attack on this protocol (see [44]) in case A may also run the protocol with a malicious insider I, i.e. a principal that has disclosed its keys to the adversary. Below we give the complete STA script for the above version of the protocol. (By convention, names begin by a capital letter, variables begin by one of letters `u`, `x`, `y`, `w` or `z`; the rest are labels. The name of each variable is meant to remind its expected value.) `InA` plays the role of A and consists of two parallel threads, one for talking to B and one for talking to I. In both threads, `InA` sends a nonce challenge and then expects a pair of encrypted nonces in response, the first of which must be the one `InA` previously issued. The third step of both these threads consists in sending out the second nonce, encrypted with the responder's public key. `InA` uses two different nonces (`NA` and `N'A`) for its two threads. `ReB` plays a role somehow specular to `InA`'s first thread. The insider I is not explicitly described, because its role is implicitly impersonated by the adversary, which knows I's keys. In other words, the behavior of I is subsumed by the behavior of the adversary. Thus, the whole system is the parallel composition of initiator and responder, `(InA || ReB)`. The initial configuration, `NS`, consists of a list containing one action plus the system. The `disclose!` action supplies the environment with the appropriate initial knowledge: identities of the participants (`A,B,I`), public keys of A (`+KA`) and B (`+KB`), and the 'seed' `KI` to synthesize both the public (`+KI`) and the private (`-KI`) key of I. The property `AuthAtoB` means that *any* message accepted by B at step 3 should indeed originate from A, since A is supposedly the only initiator: this means that B is really talking to A. `AuthBtoA` can be explained similarly.

```
(* Initiator and Responder *)
val InA = a1!(NA,A)^+KB >> a2?(NA,xNB)^+KA >>
          a3!(xNB)^+KB >> stop
        ||
          a'1!(N'A,A)^+KI >> a'2?(N'A,xNI)^+KA >>
          a'3!(xNI)^+KI >> stop;
val ReB = b1?(yNA,A)^+KB >> b2!(yNA,NB)^+KA >>
          b3?(NB)^+KB >> stop;
(* The initial configuration *)
val NS  = ( [ disclose!(KI,+KA,+KB,A,B,I) ]
          @ (InA || ReB) ) ;
(* Prop.1:  A is correctly authenticated to B *)
val AuthAtoB = ( a3!u <-- b3?u);
(* Prop.2:  B is correctly authenticated to A *)
val AuthBtoA = ( b2!u <-- a2?u);
```

When required to check whether `NS` satisfies `AuthAtoB`, STA finds a trace of `NS`, reported below, that violates the property `AuthAtoB`. The adversary, which intercepts all messages, re-uses the nonce `N'A` (issued by A when interacting with I) when impersonating the role of A talking to B. Then, the adversary induces A to decrypt message `(N'A,NB)^+KA`, thus getting `NB` (actions `a'2` and `a'3`). This attack was found after examining 26 symbolic configurations, which took slightly more than half a second. (This and the following examples were run on a PC with a 350 MHz Pentium III and 64Mb RAM.) The actual output of STA is given below. Note that the symbolic execution causes `yNA` to be instantiated to `N'A`.

```
An attack was found:
   disclose!(KI,+KA,+KB,A,B,I). a'1!(N'A,A)^+KI.
   a1!(NA,A)^+KB. b1?(N'A,A)^+KB. b2!(N'A,NB)^+KA.
   a'2?(N'A,NB)^+KA. a'3!(NB)^+KI. b3?(NB)^+KB.
26 symbolic configurations reached.
```

After repairing the flaw as suggested in [44], i.e. by inserting explicit identities inside each encrypted message, STA finds no additional attack, either on property `AuthAtoB`, or on `AuthBtoA`. In both cases, the exploration reached all the 60 configurations that constitute the complete symbolic state-space of the protocol, and this took again slightly more than half a second. We also tried other instances of the protocol (see below), without finding any attack.

6.2.2 A Type Flaw

We now consider a more accurate version of Needham-Schroeder, in which both A and B also communicate with a trusted server S to get a signed certificate of the other party's public key. Below, $Sig_Y(X)$ denotes X digitally signed by agent Y.

$$
\begin{array}{llll}
1. & A \longrightarrow S: & B & \\
2. & S \longrightarrow A: & Sig_S(+KB, B) & \\
3. & A \longrightarrow B: & \{NA, A\}_{+KB} & (NA \text{ fresh nonce}) \\
4. & B \longrightarrow S: & A & \\
5. & S \longrightarrow B: & Sig_S(+KA, A) & \\
6. & B \longrightarrow A: & \{NA, NB, B\}_{+KA} & (NB \text{ fresh nonce}) \\
7. & A \longrightarrow B: & \{NB\}_{+KB} &
\end{array}
$$

This example is interesting because it exhibits a type-dependent flaw. This kind of flaws is normally not detected using traditional finite-state techniques, unless the specification is tailored towards finding specific, hence, *a priori* known bugs (see, e.g., [25]).

We analyze two parallel runs of the protocol: A acts as responder and as initiator, respectively, while B acts only as responder. The interaction of S with A and B can be interleaved. This version of the protocol can be specified in STA by modifying the scripts for the previous example. E.g., the specification of the responder B is:

```
val ReB = b1?(yNA,A)^+KB >> b2!A >>
    b3?(ypkA,A)^+SigS >> b4!(yNA,NB,B)^(ypkA) >>
    b5?(NB)^+KB >> stop;
```

Note that, in action `b3`, `ReB` implicitly uses `-SigS` as a verification key for messages signed by `S` (the signing key is `+SigS`). We asked STA to check the property `a5!u <-- b5?u`, being `a5` the label of `InA`'s final action (when acting as an initiator). Such property guarantees the initiator A is correctly authenticated to B. After searching 16,275 configurations, STA found a trace violating the property, reported below. In this trace, `InA` is involved in two parallel runs. In the first of these runs, where A acts as the initiator, the adversary gets the `b4` message `(yNA,NB,B)^+KA`. In the second run, where A acts as the responder, the adversary replays this message to A (action `a'1`). A interprets the pair (`NB,B`) as an agent's name and sends it to the server (`a'2`). The adversary can then intercept (`NB,B`) and reply the nonce `NB` to the nonce challenge (`b5`). Note that in the trace below the variable `yNA` can be instantiated to any value.

```
disclose!(A,B,-SigS,+KA,+KB). a1!B. b1?(yNA,A)^+KB.
b2!A.s'1?A. s'2!(+KA,A)^+SigS. b3?(+KA,A)^+SigS.
b4!(yNA,NB,B)^+KA. s1?B. s2!(+KB,B)^+SigS.
```

```
a'1?(yNA,NB,B)^+KA. a'2!(NB,B). b5?(NB)^+KB
```

This attack can be prevented by simply changing $\{NA, A\}_{+KB}$ in step 3 of the protocol into $\{NA, (\mathsf{id}, A)\}_{+KB}$, for id an arbitrary tag indicating its intended type. This tag avoids any confusion between messages in step 3 and 6 of different runs.

6.3 Example 2: Kerberos

Kerberos [41] is a protocol based on shared-key cryptography for mutual authentication between a client C and a server S. In order to authenticate itself, C needs a ticket issued by a Ticket Granting Server (TGS) and, to get this ticket, another ticket is required from a Key Distribution Center (KDC). We analyze the simplified version considered in [54], that leaves out timestamping and nonces. First, C requests from the KDC a ticket for the TGS. Then, the KDC sends to C a newly generated session key for the communication between C and the TGS, and a ticket for the TGS (encrypted with a symmetric key K_{KT}, shared between KDC and TGS). C decrypts the session key, generates a so-called authenticator and sends it to the TGS, together with the received ticket and a request of a ticket for S. C is authenticated to the TGS, since C encrypts the authenticator under the session key that is contained in the ticket for the TGS. In a corresponding fashion, C is authenticated to the server in the further two steps. K_{TS} is the symmetric key shared between TGS and S. K_{s2} is the session key, shared between C and S at the completion of the protocol. The informal description of the protocol is reported below.

$$
\begin{array}{lrcll}
1. & C & \longrightarrow & KDC & : \; C, TGS \\
2. & KDC & \longrightarrow & C & : \; \{K_{s1}\}_{K_c},\, \{C, K_{s1}\}_{K_{KT}} \\
3. & C & \longrightarrow & TGS & : \; \{C\}_{K_{s1}},\, \{C, K_{s1}\}_{K_{KT}},\, S \\
4. & TGS & \longrightarrow & C & : \; \{K_{s2}\}_{K_{s1}},\, \{C, K_{s2}\}_{K_{TS}} \\
5. & C & \longrightarrow & S & : \; \{C\}_{K_{s2}},\, \{C, K_{s2}\}_{K_{TS}}
\end{array}
$$

The present version of the protocol authenticates C only. A further step is required for S's authentication. We split our STA analysis of Kerberos into two parts, client authentication and server authentication.

6.3.1 Client Authentication

We begin by analyzing a run that has one participant for each role. The STA script for this version of the protocol is reported below.

```
val Client = c1!(C,TGS) >>
             c2?({xKs1}Kc, xTkt) >>
             c3!({C}xKs1,xTkt,S) >>
             c4?({xKs2}xKs1,xTktS) >>
             c5!({C}xKs2,xTktS) >> stop;

val KDC    = kdc1?(C,TGS) >>
             kdc2!({Ks1}Kc,{C,Ks1}KKT) >>
             stop;

val TGS    = t1?({wC}wKs1,{wC,wKs1}KKT,S) >>
             t2!({Ks2}wKs1,{wC,Ks2}KTS) >>
             stop;

val Server = s1?({zC}zKs2,{zC,zKs2}KTS) >>
             stop;

val System = Client || KDC || TGS || Server;

val KERB   = ([disclose!(C,C',S)] @ System);

val AuthC  = (c5!(u,u'') <-- s1?(u,u'));
```

Declarations of labels, variables and names are omitted. As for the case of Needham-Schroeder protocol, we adopt input patterns to get a compact specification. In fact, certain input actions would take more than one step in the formal model (and in real protocol's implementations too). For instance, the execution of the server's action `s1` implies decrypting the second component (`{zC,zKs2}KTS`) under `KTS`; extracting the key (`zKs2`); decrypting with `zKs2` the first component (`{zC}zKs2`); getting the client's ID therein (`zC`); comparing `zC` to the ID found in the second component. The property `AuthC` requires that for each value of `u` and `u'`, if `s1?(u,u')` occurs in a trace, there is some value of `u''` s.t. `c5!(u,u'')` occurred at some previous point in that trace. (In general, variables that appear only to the left of `<--`, like `u''` in this example, are existentially quantified.) In other words, the first component of the message accepted by *S* at step `s1` is required to originate from the *C*'s message at step `c5`. This implies that, at the protocol's completion, *S* is really talking to *C*, and they agree on the newly established key too. The property has been verified by STA and the result is as follows:

```
No attack was found.  307 symbolic configurations
reached.
```

Note that there is no guarantee that C and S agree on the second component of their final messages, bound to xTktS and to {zC,zKs2}KTS, respectively: in fact, these two expressions may assume different values in some run of the protocol, even though there is agreement on the first components, {C}xKs2 and {zC}zKs2 respectively. The reason is that here C just forwards the value bound to xTktS, but cannot inspect it. Thus, while the adversary provides S with the right TGS ticket, xTktS might actually be anything different. STA pointed out this anomaly when required to verify the property (c5!u <-- s1?u), i.e. an agreement on the whole final message. It is dubious whether it should be regarded as a flaw. A similar anomaly, i.e., the lack of connection between the first two components of c3, is at the basis of a more interesting attack on the server's authentication, as explained in the next Subsection.

This version of the protocol is also subject to a (well-known) attack based on the use of compromised session keys and tickets – not surprisingly, due to the absence of timestamps and nonces. This attack shows up in case the initial adversary's knowledge is augmented with the compromised information. Here KOld is an old session key and {C,KOld}KTS the corresponding TGS ticket: it is assumed that these two quantities have been issued by the TGS at some time in the past and KOld has been somehow disclosed to the adversary.

```
val KERB =  ([disclose!(C,S, KOld, {C,KOld}KTS )]
              @ System);
```

When fed with this example, STA almost immediately found an attack, reporting the trace:

```
disclose!(C,S,KOld,{C,KOld}KTS). c1!(C,TGS).
s1?({C}KOld,{C,KOld}KTS).
```

The attack arises from the absence of a check on the freshness of the TGS ticket. Traditionally, timestamps are adopted to repair this flaw. However, presently, timestamps are not handled within our symbolic method.

6.3.2 Server Authentication

We consider now extending the above protocol with a sixth step, which lets S authenticate itself to C:

$$6. \quad S \longrightarrow C: \ \{C, S\}_{K_{s2}}.$$

In STA, this corresponds to adding one more action to both Server and Client in the STA script specification of the protocol.

```
val Server = ··· >> s2!{zC,S}zKs2 >> stop;
val Client = ··· >> c6?{C,S}xKs2 >> stop;
```

The property `AuthServer = s2!u <-- c6?u` expresses that the final message accepted by C indeed originates from S. If so, C is really talking to S, and the two parties agree on their reciprocal identities and on the session key. When asked to verify `AuthServer`, STA found an attack after exploring 298 symbolic configurations. The relevant attack's actions are as follow (the trace does not contain any `s2`-action):

```
··· c3!({C}Ks1, xTkt,S). ··· c4?({C}Ks1,xTktS).
 c5!({C}C,xTktS).c6?{C,S}C.
```

Note that the free variables in the trace can be instantiated to any value. The point here is that, in action `c4`, C accepts anything encrypted with `Ks1` as first component of the received message, including `{C}Ks1`, issued by C itself in `c3`. Thus, it mistakes its own ID, `C`, for the new session key. Of course, `C` is known to the adversary, who can, therefore, build a message `{C,S}C` and hand it to C, no matter if S is involved in the run. Again, this is the case of a type flaw, which we were not aware of before checking the property. This anomaly can be repaired by simply changing in step 4 $\{K_{s2}\}_{K_{s1}}$ into $\{0, K_{s2}\}_{K_{s1}}$: the addition of 0 avoids messages with the same format in steps 3 and 4. After this adjustment, STA found no additional attack. The whole symbolic state-space of 622 configurations has been explored in about 3 seconds.

Next, we considered a run of the corrected protocol with two servers, S and S', and a client. Again no attack was found by STA, after exploring 7,516 symbolic configurations. The authentication step 6 prevents the attack described in [54], based on the adversary's replacing the ID S in step 3 with a different S'. In fact, in the final step, each server explicitly declares its identity.

6.4 Discussion

We try an assessment of our approach, by discussing what we think are its main benefits, and by contrasting our work with those of finite-state methods (e.g. [44, 47, 54]).

6.4.1 Accuracy of the model

Finite-state model checking of security protocol relies on finite approximations of the actual model obtained, e.g., by imposing some fixed typing to the adversary-generated messages [54]. For example, it might be assumed that, at a given stage, the adversary can only send messages that fit in the format {nonce}key. The number of such possible messages is finite, as long as the number of distinct nonces and keys is finite. These typing assumptions may be sensible under certain circumstances (see [37]), but they cannot be established automatically: it seems that some a priori knowledge is required of how the protocol works. This makes the whole analysis process potentially error-prone.

On the contrary, our approach makes no assumption on typing, or on the number of messages the adversary may generate. In fact, as a consequence of the completeness result, the symbolic model that STA explores makes no approximation with respect to the infinite-state standard model.

6.4.2 Efficiency

Symbolic analysis does not suffer from any state explosion problem depending on message exchange. Conversely, in finite-state model checkers, even if clever assumptions may reduce the state space to explore (see e.g. [73]), branching factor still remains. For instance, if in some state the adversary can only send messages of type $\{\mathsf{nonce},\mathsf{nonce}\}\mathsf{key}$, then at least $n \times n \times k$ transitions will branch from that state (being n and k the number of possible nonces and keys). As the number of participants, and consequently of possible data values, increases, the size of the model is expected to increase dramatically.

In view of these considerations, STA seems to have some advantage over finite-state model checkers. This is reflected on those instances of Needham-Schroeder analyzed here and in [54]. Some figures on the number of reachable states for the correct version of protocol are reported below. Execution times heavily depend on the number of states, but also on a variety of technological factors, thus exact figures for them are less meaningful. Indicatively, our execution times are from 5 to 70 times shorter than those reported in [54]. Memory occupation is very well controlled in STA because a depth first search strategy is adopted.

Initiators	**Responders**	**Murφ**	**STA**
1	1	1706	60
2	1	17,277	411
2	2	514,550	24,655

It is well known that a form of state-explosion is also induced by interleaving. However, this is not specific to security protocols. There exist standard techniques to deal with this problem, but they have not been considered in our implementation.

The aspect of efficiency deserves, anyway, a comprehensive comparison study. Of course, the word ‘efficiency’ should be taken in a practical sense, here, rather than in the formal sense of ‘worst-case complexity efficiency’. In fact, the problem is NP-hard [69], and pathological examples can always be exhibited, for STA like for other tools.

6.4.3 Usability

The user interface of STA is at the moment rather rudimentary, thus specifying a protocol requires a certain acquaintance with process algebras. However, no deep understanding of security is needed; in particular, no a priori knowledge of the protocol to analyze is required. Another point worth noticing is that no description of the adversary must be explicitly given, apart form its initial knowledge. For an experienced user, once the configuration to be tested has been chosen, translating the informal description of the protocol into the STA specification takes about half an hour, and then everything is automatic. The whole specification process could be made more user-friendly by developing some high-level user interface in the same fashion as Lowe's Casper [46].

Chapter 7

Conclusions and Future Work

The main objective of the thesis has been to study mobility and security issues in models for distributed systems.

In the first part of the thesis, we have pursued a study on the expressiveness of the join calculus, by comparison with (generalized, colored) Petri nets, using tool from type theory. We have proposed two natural notions of high-level nets accounting for dynamic reconfiguration and process creation that fully correspond to the join calculus.

We have also presented a general result about lifting calculi with structural axioms to bialgebras and we have applied such a result to the π-calculus. We have enriched the permutation algebra for the calculus defined in [55, 56] with parallel composition, restriction and an extrusion operation δ. Since this algebra satisfies the condition required by our general theory (i.e., structural axioms bisimulate), the associated transition system can be lifted to obtain a bialgebra and, as a consequence, in the π-calculus, bisimilarity is a congruence with respect to parallel composition and restriction.

In the second part of the thesis we have addressed the issue of the analysis of security protocols. We have proposed a framework for the automatic analysis of security protocols and provided some sufficient conditions under which verification can be performed via a symbolic method. In contrast to finite-state model checking, our method can analyze the whole infinite state space generated by a limited number of participants. The method is efficient in practice, because the symbolic model is compact. However, claims of efficiency should generally be taken with some care, given that the protocol analysis problem is NP-hard even under very mild hypotheses (see e.g. [69]).

Finally, we have described STA, a tool based on this symbolic method for automatic verification of security protocols. We have tested our tool by analyzing Needham-Schroeder and Kerberos protocols. We have then compared our method and results to those obtained by other authors with finite-state model checking.

The work presented in this thesis can be extended in different directions.

We are rather confident that π-calculus observational congruences (both early and late) can be easily handled in our approach by considering algebras of name substitutions rather than name permutations. The same extension should accommodate also calculi with name fusions. More challenging would be to introduce general substitutions (on some first order signature), yielding models rather close to logic programming. By varying the underlying algebra, but otherwise employing similar constructions, we plan to show the flexibility of our uniform bialgebraic approach, where substitutions and extrusion are at the same level as the other operations of the algebra.

Concerning the analysis of cryptographic protocols, we are concentrating on setting low-level cryptographic operations, such as modular exponentiation within our symbolic framework. STA can be currently considered little more than a prototype. Future work includes development of a user-friendly interface and optimizations of the data structures.

Bibliography

[1] M. Abadi and B. Blanchet. Analyzing Security Protocols with Secrecy Types and Logic Programs. In *Conf. Rec. of POPL'02*, 2002.

[2] M. Abadi and C. Fournet. Secure Implementation of Channel Abstractions. In *Proc. of LICS'00*, IEEE. Computer Society Press, 2001.

[3] M. Abadi and C. Fournet. Mobile Values, New Names, and Secure Communication. In *Conf. Rec. of POPL'01*, 2001.

[4] M. Abadi and A.D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1-70, 1999.

[5] R.M. Amadio S. Lugiez. On the reachability problem in cryptographic protocols. In *Proc. of Concur'00*, LNCS 1877, 2000. Full version: RR 3915, INRIA Sophia Antipolis.

[6] R.M. Amadio, S. Lugiez, and V. Vanackère. On the symbolic reduction of processes with cryptographic functions. RR 4147, INRIA Sophia Antipolis, March 2001.

[7] R. Amadio and S. Prasad. The game of the name in cryptographic tables. In *Proc. of Asian'00*, LNCS 1742, Springer-Verlag, 2000. RR 3733 INRIA Sophia Antipolis.

[8] A. Asperti and N. Busi. Mobile Petri Nets. Technical Report UBLCS 96-10, Università di Bologna, 1996.

[9] G. Berry and G. Boudol. The Chemical Abstract Machine *Theoretical Computer Science*, 96:217–248, 1992.

[10] M. Boreale. Symbolic Trace Analysis of Cryptographic Protocols. In *Proc. of ICALP'01*, LNCS 2076, Springer-Verlag, 2001.

[11] M. Boreale and M. Buscemi. Experimenting with STA, a Tool for Automatic Analysis of Security Protocols. In *Proc. of SAC'02*, ACM Press, 2002.

[12] M. Boreale and M. Buscemi. A Framework for the Analysis of Security Protocol. In *Proc. of CONCUR '02*, LNCS 2421. Springer-Verlag, 2002.

[13] M. Boreale, R. De Nicola, and R. Pugliese. Proof Techniques for Cryptographic Processes. *SIAM Journal on Computing*, 31(3):947-986, 2002.

[14] M. Boreale, C. Fournet, and C. Laneve. Bisimulations for the Join-Calculus. In *Proc. of PROCOMET'98*, IFIP. Chapman & Halls, 1998.

[15] R. Bruni, J. Meseguer, U. Montanari, and V. Sassone Functorial Models for Petri Nets. *Information and Computation*, 170(2):207-236, 2001.

[16] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions in Computer Science*, 8(1):18–36, 1990.

[17] M. Buscemi and U. Montanari. A First Order Coalgebraic Model of Pi-Calculus Early Observational Equivalence. In *Proc. of CONCUR '02*, LNCS 2421, Springer-Verlag, 2002.

[18] M. Buscemi and V. Sassone. High-Level Petri Nets as Type Theories in the Join Calculus. In *Proc. of FoSSaCS '01*, LNCS 2030, Springer-Verlag, 2001.

[19] E.M. Clarke, S. Jha, and W. Marrero. Using State Space Exploration and a Natural Deduction Style Message Derivation Engine to Verify Security Protocols. In *Proc. of IFIP PROCOMET*, 1998.

[20] H. Comon, V. Cortier, and J. Mitchell. Tree automata with one memory, set constraints and ping-pong protocols. In *Proc. of ICALP'01*, LNCS 2076, Springer-Verlag, 2001.

[21] A. Corradini, M. Große-Rhode, R. Heckel. Structured transition systems as lax coalgebras. In *Proc. of CMCS'98*, ENTS 11. Elsevier Science, 1998.

[22] A. Corradini, R. Heckel, and U. Montanari. Compositional SOS and beyond: A coalgebraic view of open systems. *Theoretical Computer Science* 280:163-192, 2002.

[23] R. De Simone. Higher level synchronising devices in MEIJE-SCCS. *Theoretical Computer Science* 37(3):245–267, 1985.

[24] D. Dolev, A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 2(29):198-208, 1983.

[25] B. Donovan, P. Norris, and G. Lowe. Analyzing a library of security protocols using Casper and FDR. In *Proc. of Workshop on Formal Methods and Security Protocols*, Trento, 1999.

[26] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. of Workshop on Formal Methods and Security Protocols*, 1999.

[27] G. Ferrari, U. Montanari, and M. Pistore. Minimizing Transition Systems for Name Passing Calculi: A Co-algebraic Formulation. In *Proc. of FoSSaCS'02*, LNCS 2303. Springer, 2002.

[28] G. Ferrari, U. Montanari, and P. Quaglia. A pi-calculus with Explicit Substitutions. *Theoretical Computer Science* 168(1):53–103, 1996.

[29] M.P. Fiore and M. Abadi. Computing Symbolic Models for Verifying Cryptographic Protocols. In *Proc. of 14th Computer Security Foundations Workshop*, IEEE Computer Society Press, 2001.

[30] M. Fiore and D. Turi. Semantics of name and value passing. In *Proc. of LICS'01*, IEEE. Computer Society Press, 2001.

[31] C. Fournet and G. Gonthier. The Reflexive Chemical Abstract Machine and the Join-Calculus. In *Proc. of POPL'96*, 372–385. ACM 1996.

[32] C. Fournet and G. Gonthier. A Hierarchy of Equivalences for Asynchronous Calculi. In *Proc. of ICALP'98*, LNCS 1443, 844–855. Springer-Verlag, 1996.

[33] C. Fournet, G. Gonthier, J. Lévy, L. Maranget, and D. Rémy. A Calculus of Mobile Agents. In *Proc. of CONCUR'96*, LNCS 1119, 406–421, Springer-Verlag, 1996.

[34] C. Fournet, C. Laneve, L. Maranget, and D. Rémy. Implicit Typing à la ML for the Join-Calculus, In *Proc. of CONCUR'97*, LNCS 1243, 196–212. Springer-Verlag, 1997.

[35] M. Gabbay and A. Pitts. A new approach to abstract syntax involving binders. In *Proc. of LICS'99*, IEEE. Computer Society Press, 1999.

[36] A. Gordon. Notes on nominal calculi for security and mobility In *Foundations of Security Analysis and Design*, LNCS 2171, Springer-Verlag, 2002.

[37] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *Proc. of 13th Computer Security Foundations Workshop*, IEEE Computer Society Press, 2000.

[38] C. Hoare. *Communicating Sequential Processes*. International Series in Computer Science. Prentice Hall, 1985.

[39] A. Huima. Efficient infinite-state analysis of security protocols. In *Proc. of Workshop on Formal Methods and Security Protocols*, Trento, 1999.

[40] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. Monographs on Theoretical Computer Science, Springer-Verlag, 1992.

[41] J. Kohl and B. Neuman. The Kerberos network authentication service (version 5). Internet Request For Comment RFC-1510, 1993.

[42] C. Laneve. May and Must Testing in the Join-Calculus. Technical Report UBLCS 96-04, Università di Bologna, 1996.

[43] F. Le Fessant and L. Maranget. Compiling Join-Patterns. In *Proc. HLCL'98*, ENCS 16(3), Elsevier Science, 1998.

[44] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-Key Protocol Using FDR. In *Proc. of TACAS'96*, LNCS 1055, Springer-Verlag, 1996.

[45] G. Lowe. A Hierarchy of Authentication Specifications. In *Proc. of 10th IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press, 1997.

[46] G. Lowe. Casper, a compiler for the analysis of security protocols. In *Proc. of 10th IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press, 1997.

[47] W. Marrero, E.M. Clarke, and S. Jha. Model checking for security protocols. Technical Report TR-CMU-CS-97-139, Carnegie Mellon University, 1997.

[48] C. Meadows. The NRL protocol analyzer: An overview. Journal of Logic Programming, 26(2):113–131, 1996.

[49] C. Meadows. Open issues in formal methods for cryptographic protocol analysis. In *Proc. of DARPA Information Survivability Conference and Exposition*, 2000.

[50] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proc. of 8th ACM Conference on Computer and Communication Security*, ACM Press, 2001.

[51] R. Milner. *A Calculus of Communicating Processes*. n. 92 of Lecture Notes in Computer Science. Springer-Verlag, 1980.

[52] R. Milner. Action Calculi, or Syntactic Action Structures. In *Proc. of MFCS'93*, LNCS 711, 105-121. Springer-Verlag, 1993.

[53] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (parts I and II). *Information and Computation*, 100(1):1–77, 1992.

[54] J.C. Mitchell, M. Mitchell, and U Stern. Automated Analysis of Cryptographic Protocols Using Murφ. In *Proc. of Symp. Security and Privacy*, IEEE Computer Society Press, 1997.

[55] U. Montanari and M. Pistore. Pi-Calculus, Structured Coalgebras and Minimal HD-Automata. In *Proc. of MFCS'00*, LNCS 1983. Springer, 2000.

[56] U. Montanari and M. Pistore. Structured Coalgebras and Minimal HD-Automata for the pi-Calculus. Technical Report 0006-02, IRST-ITC, 2000. Available at the URL: `http://sra.itc.it/paper.epl?id=MP00`.

[57] R. M Needham and M. D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, 1978

[58] M. Nielsen, L. Priese, and V. Sassone. Characterizing Behavioural Congruences for Petri Nets. In *Proc. of CONCUR'95*, LNCS 962. Springer-Verlag, 1995.

[59] M. Odersky. Functional Nets. In *Proc. of ESOP'00*, LNCS 1782. Springer-Verlag, 2000.

[60] M. Odersky, C. Zenger, M. Zenger, and G. Chen. A Functional View of Join. Technical Report ACRC-99-016, University of South Australia, 1999.

[61] L.C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[62] C.A. Petri. *Kommunikation mit Automaten*. Ph.D. thesis, Institut fr Instrumentelle Mathematik. Bonn, 1962.

[63] M. Pistore. *History Dependent Automata*. PhD. Thesis TD-5/99. Università di Pisa, Dipartimento di Informatica, 1999. Available at the URL: `http://www.di.unipi.it/phd/tesi/tesi_1999/TD-5-99.ps.gz`.

[64] A. M. Pitts. Nominal Logic: A First Order Theory of Names and Binding. In *Proc. of TACS'01*, LNCS 2215. Springer, 2001.

[65] G.D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981..

[66] W. Reisig. *Petri Nets: An Introduction*. EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1985.

[67] A.W. Roscoe. Modelling and verifying key-exchange using CSP and FDR. In *Proc. of 8th Computer Security Foundations Workshop*, IEEE Computer Society Press, 1995.

[68] A.W. Roscoe. Proving security protocols with model checkers by data independent techniques. In *Proc. of 11th Computer Security Foundations Workshop*, IEEE Computer Society Press, 1998.

[69] M. Rusinowitch and M Turuani. Protocol Insecurity with Finite Number of Sessions in NP-Complete. In *Proc. of 14th Computer Security Foundations Workshop*, IEEE Computer Society Press, 2001.

[70] J.J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science* 249(1):3–80, 2000.

[71] V. Sassone. On the Algebraic Structure of Petri Nets. *Bulletin of EATCS*, n. 73, 2001.

[72] S. Schneider. Verifying Authentication Protocols in CSP. *IEEE Transactions on Software Engineering*, 24(8):743-758, 1998.

[73] V. Shmatikov and U. Stern. Efficient finite-state analysis for large security protocols. In *Proc. of 11th Computer Security Foundations Workshop*, IEEE Computer Society Press, 1998.

[74] D. Turi and G. Plotkin. Towards a mathematical operational semantics. In *Proc. of LICS'97*, IEEE. Computer Society Press, 1997.

[75] STA: a tool for trace analysis of cryptographic protocols. ML object code and examples, 2001. Available at the URL:
`http://www.dsi.unifi.it/∼boreale/tool.html`.