Università degli Studi di Pisa
Dipartimento di Informatica

Dottorato di Ricerca in Informatica

Ph.D. Thesis: TD–2/00

Security Issues in Process Calculi

Chiara Bodei

March 2000

Addr: Corso Italia 40, 56125, Pisa, Italy
Tel: +39-050-887267 — Fax: +39-050-887226 — E-mail: chiara@di.unipi.it
Home page: http://www.di.unipi.it/~chiara

Thesis supervisor:
Prof. Pierpaolo Degano

Abstract

We study security properties of systems, in a process algebraic framework, paying particular attention to the analysis of cryptographic protocols and to the development of secure code for distributed, concurrent and mobile systems. Security problems are addressed both from a static and a dynamic point of view, having operational semantics as a common context. In one case we exploit static techniques, mainly based on Control Flow Analysis, while in the other we rely on certain forms of enhanced operational semantics.

We choose the pure setting of concurrency calculi, in particular the π-calculus, a foundational calculus based on the notion of naming. We also consider the spi-calculus, a security-oriented extension of the π-calculus, enriched with primitives for encryption and decryption.

On the static side, Control Flow Analysis is used to verify simple security properties both for π-calculus and spi-calculus processes. For the first calculus, the static property of *confinement* ensures that, at run-time, information devised to be secret is never disclosed via a communication to unauthorized external users, i.e that a process has *no leaks*. For the *no read-up/no write-down* property instead, processes are classified according to security levels and those at high level must not send information to those at low level. The static property that guarantees that information flow in the right direction is called *discreetness*.

For the spi-calculus, the property of confinement is recast and then compared with another notion of secrecy, proposed by Abadi. Finally a form of message authentication, called *authorship* is considered.

On the dynamic side, we exploit an enhanced semantics for the π-calculus and the spi-calculus that considers names localized to their generators and that offers a primitive of message authentication.

To Federico

Securitas est laetitia,
orta ex idea rei futurae,
vel praeteritae,
de qua dubitandi causa
sublata est.

Spinoza (Etica, III)

Acknowledgments

Considering the debts that I accumulated during my PhD course and in writing this thesis, I can hardly express my deepest gratitude to all the people who helped me directly and indirectly.

First of all, I wish to thank Pierpaolo Degano, who so much inspired, taught and followed my research and this work. His patience and encouragement were essential to me, as well as his irony and humor.

I owe thanks to Corrado Priami, Hanne Riis Nielson, Flemming Nielson and Riccardo Focardi, with whom was so nice and stimulating to work with. My efforts were enormously assisted by their support and guidance.

I am glad to have spent the first term of my PhD at BRICS, Denmark, under the supervision of Mogens Nielson.

I really enjoyed discussing and exchanging ideas with Paolo Baldan, Roberto Bruni, Luca Cattani, Francesca Levi and Vladimiro Sassone. Furthermore, I would like to thank all the people at the Computer Science Department, in particular, my roommates Davide Marchignoli, Fabio Martinelli, Simone Tini and my collegues Stefano Bistarelli, Marco Comini, Simone Contiero, Roberta Gori, Nadia Pisanti, Alessandra Raffaetà and Andrea Zavanella.

I am grateful to Giorgio Levi, Andrea Maggiolo-Schettini e Ugo Montanari, whose comments and questions have been a most valuable source of insight during the development of my work.

I am also indebted with my external referees Martín Abadi and Chris Hankin for their careful reading of a preliminary draft of the thesis and for their constructive criticisms. I owe twice to Martín Abadi, whose accurate and detailed comments were incomparable in improving the presentation.

Finally, thanking all my relatives (included my relatives in law) is by no means a matter of form: their warm support never failed. A big thank to my husband Mario and to my little baby Federico who shared me with my work, especially in this last year (for Federico, his whole life).

Contents

List of Figures

List of Tables

Introduction

Security, from the Latin *securitas*, indicates the condition of being protected from or not exposed to a danger, i.e. of being safe. Nevertheless, as soon as the name "computer" or "information" qualifies this term, what seems to be an obvious concept elusively refuses to assume a precise meaning. The security community itself has not yet reached a consensus on its exact definition, despite a long debate. Security has to do with confidentiality, integrity and availability, but also with non-repudiation, authenticity and even more, depending on the application one has in mind. Technology as well influences security, introducing new problems and updating some of the old ones. In particular, the widespread diffusion of distributed systems and networks has furnished a great number of interesting scenarios in which security plays a significant role. Anyway, in these systems, security problems are in general undecidable [51], for their dynamic behaviour: this leads to the search for properties able to guarantee that a system is secure.

Literature is strewn with a variety of properties and measures proposed to achieve security objectives, different in style and in quality and often especially tailored for a certain purpose.

Formal methods and techniques can help generalizing and comparing these different formulations, driving the delicate translation of intuitive notions into formal specifications, essential for a careful design and analysis. Furthermore, the choice of different abstraction levels given by formal approaches supports reasoning on security issues. Finally, also implementations may benefit from formal methods: theories in which specifications can be refined are consequently required as a guideline.

I.1 Formal Framework

There are essentially three orthogonal aspects that are crucial for security. The first aspect regards the protection of information devised to be confidential from unauthorized disclosure (*secrecy* or *confidentiality*) and thus deals with *who* may receive or read the data. The second aspect regards the protection of information devised to have integrity against unauthorized modification (*integrity*) and thus deals with *who* may send or write the data. Closely related to this second aspect is that of ensuring that the agents do not lie about their identity (*authentication*) and thus deals with *whom* one is willing to communicate with. A third aspect concerns the availability of services at all times (avoiding *denial of service*) and thus deals with overall system performance. These, in rough and rapid outlines, are the main problems confronting us in the area of communication systems and, in particular, for the analysis of security protocols.

Process algebras offer a pure framework to study concurrent and distributed systems and, in turn, the security issues connected to them. Systems are specified as expressions of the calculus, called *processes*. Processes are obtained by combining via a few operators (sequential and parallel composition, nondeterministic choice, declarations) the basic actions of sending and of receiving messages between processes along channels. Furthermore there are some scope operators, such as restrictions and hiding.

By focusing on the essence of interactions among processes, these calculi make their well established theory available to inquiry the subtle aspects of communications, in which security problems or flaws often hide.

Moreover, process algebras furnish a common background, where comparing different formulations and models for the same property is possible.

Among process algebras, the π-calculus [88], a foundational calculus based on the notion of name passing, seems particularly suitable to address security problems, especially to model the usage of secret information. In its setting, names – that represent values, or messages, and also channels – can be created and passed. Processes can only communicate on the channels they know: for them, learning the name of a channel amounts to possessing the capability to communicate on the channel. A process can widen its communication possibilities, via an explicit mechanism of the calculus, called *scope extrusion*, that enlarges the scope of names. Consequently, the semantic rules may explicitly control the access to channels and to data.

Nevertheless, the π-calculus is yet too abstract to explicitly represent cryptographic operations, that are an important ingredient of security protocols. To fill this gap, Abadi and Gordon [8, 7] defined the spi-calculus, an extension of the π-calculus with primitives for encryption and decryption.

Once cast in process algebraic framework, concurrent and distributed system can be studied using either static techniques or choosing a dynamic approach.

Traditionally used for code optimization, program analysis offers many static techniques that are useful for security issues, for it provides a repertoire of automatic and decidable methods and tools for analysing properties of system. This feature allows to circumvent the undecidability issues that generally arise in security [51].

In this field, two static techniques are particularly relevant, namely, Type Systems and Control Flow Analysis. In both cases, the analysis of systems can be oriented to define security static tests: if a system passes security checks at compile-time, then it will not violate the security requirements at run-time.

There are various notions of (bisimulation based) behavioural equivalence defined for process algebras, which facilitate both the statement of properties and their check. These represent a possible basis for a dynamic approach to security, where the focus usually relies on the external observable behaviour of processes.

I.2 Object and Objectives

Well established and founded, the theories and techniques just mentioned offer a fertile ground to study security properties of distributed and concurrent systems, and also – even though at a very abstract level – cryptographic protocols. Bringing protocols into a merely theoretical and abstract perspective allows to check security properties in isolation and to focus on their very nature.

We address security problems, both from a static and a dynamic point of view, having operational semantics as a common context. In one case we exploit static techniques, mainly based on Control Flow Analysis, while in the other we rely on certain forms of enhanced operational semantics.

Our attention is focused on a limited set of simple properties that we study in the pure setting of the π-calculus and of the spi-calculus.

On the static side, the starting point is the use of Control Flow Analysis, in place of the more used Type Systems. The analysis returns solutions that statically predict how names will be bound to actual channels at run-time. This feature makes Control Flow Analysis suitable to verify security properties, by defining static tests on solutions, that guarantee, if passed that the properties hold. The technical basis to obtain this kind of result is offered by the possibility to prove semantic correctness. Remarkably, existence of best solutions is provable and, for the π-calculus case, there is a constructive procedure that generates solutions in low polynomial time.

We study first a property of secrecy, similar to the property of no leaking, introduced by Abadi in [1] for the spi-calculus. Both notions consider data classified in "secret" and "public". In our framework, a process has *no leaks*, if secret information is never disclosed. Our static test (called *confinement*) checks that secret information may only be communicated over secret channels.

At this point, the next natural step is the comparison with our notion of secrecy and Abadi's in the spi-calculus context. The two notions are tightly related, and their comparison gives some useful insights on the secrecy properties and may contribute to understand the interplay between Type Systems and Control Flow Analysis techniques.

Both our and Abadi's notions of secrecy deal with the flow of information. Controlling flows is one of the main issue in security literature and it is usually based on the classification of processes and data in security levels, with the requirement that no highly classified information could flow to lower levels.

In this context, we present another property for the π-calculus, a variant of the *no read-up/no write-down* property of Bell and LaPadula [14]. Processes are given levels of security clearance, and the property demands that those at high level never send information to those at low level. A little extension to the already mentioned Control Flow Analysis for π-calculus is sufficient to define a static check (called *discreetness*) for when a process respects the classification hierarchy, and to prove it safe with respect to the dynamic notion.

As we have seen, the other relevant class of security properties regards authentication. We study here, in the spi-calculus framework, *message authentication*, i.e. the property requiring that the receiver of a message can ascertain its origin. Message authentication problems seem more difficult to cope with, at compile-time, because no static technique can predict the exact sequence of message exchanges. As a consequence, we start with a limited form of message authentication, called *authorship* that states when a message is made of packets of information generated by a selected process, that is therefore authenticated. Authorship can be statically guaranteed.

On the dynamic side, we still remain on message authentication. We use an enhanced semantics of the π-calculus and of the spi-calculus that offers a kind of *authentication primitive*. The new semantics provides each sequential process with its own local space of names and local name manager that generates a fresh name whenever necessary. We use here the adjective *local* in the sense of Algol-like languages, with no connection with the physical localities where processes run. When a name is sent, it is equipped with the information, called *relative address*, needed to point back to the local environment where it has been generated. The reference to the generator of a name is maintained during all the communications, by a semantic mechanism *not* available to the users. Therefore, the relative address of a message uniquely identifies its originator, even after the message has been maliciously intercepted and forwarded. Availability of localized names guarantees by construction that a message has been generated by a given entity, making message authentication immediate.

The following sections go a bit more in depth as far as the main tools used are concerned. The last one offers an overview of the thesis.

I.3 Operational Semantics

One of the major approaches to the formal semantics is given by the operational one. Introduced in the sixties, [82, 79], it describes the meaning of a programming language in terms of elementary steps on an abstract machine, formalised as transitions between states. Behaviours are graphically represented as *transition systems*, i.e. oriented graphs, where nodes represent states (or configurations) and arcs represent transitions between states. Transitions may be labelled by additional information on the activity performed.

Operational semantics provides quite a natural and intuitive way to understand how a program behaves. This aspect, joint with the simplicity of its mathematical basis, makes operational semantics helpful both in the design and implementation phases.

The main drawback is instead that the meaning of programs is not directly modelled, because it necessarily passes through their execution sequences. Therefore it is more difficult to reason about programs *in se*, without concern of implementation and execution problems. This may lead to a lack of structure and of compositionality, i.e. the possibility to give the semantics of a construct in terms of the semantics of its components. Compositionality, essential for making the semantic definitions finite, appears in this framework only in the seventies, [39], leading Plotkin [106] to propose a structural approach.

Structural Approach The introduction of Structural Operational Semantics or SOS [106] arises from the cross-fertilization with denotational techniques, borrowing from them notions such as compositionality and abstract syntax. The semantics of compound constructs is defined in terms of the semantics of their components, following the various syntactic possibilities, i.e. it is *syntax-directed.* Transitions are deduced in a logically-based way, by inducing on the syntactic constructs. Transitions themselves are defined using axioms and inference rules of the form

$$\frac{Premises}{Conclusion}$$

where if the premises are satisfied, so does the conclusion. From a formal point of view, this amounts to having a logically-based proof system, that, in particular, provides a reliable support to the development of automatic tools.

The structural operational semantics, still preserving the simplicity of the traditional operational semantics, exploits compositionality, making specifications modular and modules suitable for re-use.

Finally, it is sufficiently general to allow to tune the level of abstraction, without drastically changing the method.

I.4 Process Algebras

Process Algebras are executable specification languages for the description of concurrent and distributed systems. These systems are made up of different active components, called processes or *agents.* Their behaviour is represented by a set of atomic I/O actions that they can perform, both independently and in interaction with each other. A specification composes these actions via a few basic operators, like sequentialization (or prefix), non-deterministic choice and parallel composition, usually allowing communications. Furthermore there are some scope operators, such as restrictions and hiding. Process algebras are, usually, interpreted on an interleaving model, making concurrency not directly observable.

Process algebras have been often given an operational semantics in SOS style. The behaviour and evolution of a program (*process*) are described through a transition system. So computations are formally defined as paths in the transition system. Also, there are various notions of (bisimulation based) behavioural equivalence defined on transition systems, which facilitate both the statement of properties and their check.

Classical process algebras (among the others *CCS* [85], *CSP* [28], *ACP* [15], *Meije* [12]), have been successively extended to cope with the possibility to exchange names (*name passing*) and processes (*Higher Order* calculi) in communications. These features express *mobility*, i.e. the dynamic change of the control structure of processes, and are present in the π-calculus [88], *HO-π* [114], Plain CHOCS [122], the spi-calculus [7] the *join calculus* [59], the *ambient calculus* [30] and many others.

Cryptographic protocols can be naturally described in a process algebra, yet at a very abstract level. The desired security properties are then studied by checking their specification on all the computations. A great amount of specifications have been given in languages such as CSP, CCS, LOTOS and, more recently in the π-calculus and spi-calculus [8, 7]. Since, the transition system associated with a protocol has often a finite number of states, the analysis is mechanizable and provides complete answers. Considerable research has been done in recent years using various process algebras (without mobility) and equivalences, e.g., to establish properties about the information flow and to detect flows in protocols [56, 78]. More recent works extend the above to study the foundations of secure mobile languages (see e.g. the papers on the spi-calculus [8, 7], on the ambient-calculus [98], on the Join calculus [6] and on KLAIM [41]).

I.5 Enhanced Operational Semantics

Descriptions of systems need to be more detailed and concrete in the implementation phase. They require to consider also information on the external environment, concerning architecture and topology. Aspects, such as locality or causality, become of interest, from this point of view. In particular, locality-based semantics model distributed systems according to their physical or geographical distribution, localising each activity in the site it is performed.

To capture this kind of low level information one often resorts to true concurrent models, because the interleaving ones only capture more extensional aspects. Unfortunately, the non interleaving approach presents a more difficult formal treatment (see for instance *Petri Nets* [109]).

This little *excursus* shows that both the approaches are essential to understand distributed systems, for complementary reasons.

Different strategies have been used to integrate the two philosophies (e.g. [44]). Some extensions to transition systems towards true concurrency have been presented, among which that of *Proved transition systems* [45, 47, 46] (See Section 1.3). A proved transition is labelled by an encoding of its deduction tree or *proof term*. Proved transition systems can be considered as a sort of ologrammatic representation of computations, containing all the possible encodable information. From this concrete and unifying model, interleaving in style, it is possible to retrieve a large number of different semantic models by abstracting from undesired information [48, 23, 22]. Aspects like causality or locality may be retrieved in this way. This approach has the advantage that the theory, the techniques and the tools valid for the interleaving models are still applicable.

I.6 Program Analysis

Program Analysis (see [97] for a presentation comprising the major approaches) offers static techniques for predicting safe and computable approximations to the set of values or behaviours arising dynamically. Program analysis aims at checking properties of a program, and more generally of a system, that hold in all executions — regardless of the actual data upon which the program operates and regardless of the specific environment in which it executes. Therefore static analysis can serve as a basis for intervening at the source of the problem, i.e. directly on the code. Consequently, it helps tuning up programming languages themselves. As far as network computing is concerned this could drive in the search for new programming languages that embody security properties, without enforcing them at implementation level. Furthermore, static analysis provides a repertoire of automatic and decidable methods and tools for analysing properties of systems. Most of them implicitly involve termination, and thus the properties are intended to "err on the safe side". In other words, static analysis provides its users with *safe approximate* answers. Additionally, for each analysis it makes sense to impose an ordering on the properties, for example stipulating that a property is larger than another if more values satisfy the former than the latter. The properties are then interpreted in such a way that an analysis remains correct even when it produces a larger property than ideally possible. This corresponds to producing a valid inference in a program logic for partial correctness. However, program analysis is generally more efficient than program verification, and for that reason more approximate, because the focus is on the fully automatic processing of large programs.

There are several approaches to program analysis, among which: *Data Flow Analysis* [72], *Control Flow Analysis* [94], *Abstract Interpretation* [34] and *Type Systems* [69].

Actually, there is a large amount of commonality among these different approaches. On the one hand, their comparison could help choosing the most suitable technique to check the property under consideration. On the other hand, each approach may shed further light on the others, leading to their mutual enrichment.

We concentrate our attention mainly on Control Flow Analysis, a specific static technique, based on Flow Logic for studying the security of systems. It is built around the more "classical" approaches to static analysis – namely Data Flow Analysis, Constraint Based Analysis and Abstract Interpretation – and thus links up with the pioneering approach taken in the very early studies by Denning [49, 50]. At the same time it offers a way to exploiting the quite advanced state-of-the-art in static analysis also in analyses of security.

I.7 Flow Logic for Control Flow Analysis

Control Flow Analysis predicts safe and computable approximations to the set of values that the objects of a program may assume during its execution. It was mainly developed for functional languages [120] but can be used for virtually all programming paradigms, e.g. for analysing imperative or object-oriented languages [102] and languages with concurrency [61] and mobility [18]. The approach is related to Data Flow Analysis and can be seen as an auxiliary analysis needed to establish the information about the interprocedural flow of control assumed when specifying the familiar equations of Data Flow Analysis. As pointed out in [97] it can also be viewed as an interprocedural version of the Definition-Use and Use-Definition Chainings that are commonly computed by optimizing compilers.

The formulation we will use here, is based on Flow Logics [95]. This is an approach that separates the specification of *whether or not* a piece of analysis information is acceptable for a given program from *actually computing* the least (or best) such piece of information. By now this approach has been developed to a level where it links up with state-of-the-art techniques from Data Flow Analysis, Constraint Based Analysis and Abstract Interpretation [94, 96, 99, 100]. This approach naturally leads to a general treatment of semantic correctness (e.g. in the form of subject reduction results) and to the existence of least solutions (in the manner of model intersection properties). This means that all systems can be analysed and that if a system passes a static security check, then all its computations are indeed secure.

I.8 Security Protocols

Security protocols are specifications for exchanging messages on a possibly insecure network, such as the Internet, to generate cryptographic keys, or to achieve authentication of the parties in communication sessions. Parties in a security protocol interact with an uncertain environment, where some of the participants are not fully trusted or maybe hostile. Consequently, even though carefully designed, protocols may have flaws, allowing malicious agents or *intruders* to violate security. An intruder, gaining some control over the communication network, is able to intercept or forge or invent messages to convince agents to reveal sensitive information or to believe it is one of the legitimate agents in the session.

Both providing a general framework to describe security protocols and basic techniques to analyse and verify them is quite hard. Due to their intrinsic fragility, protocols can always harbour some flaws and are thus prone to intruders' attacks. This indicates how important is to have rigorous ways to prove their correctness, with respect to security properties.

I.9 Outline of the Work

In **Chapter 1**, we introduce some technical background, required for the following chapters. More precisely:

- Section 1.1 introduces some of the basic notions on transition systems;
- Section 1.2 introduces some of the basic notions and results on the π-calculus;
- Section 1.3 introduces the proved version of the π-calculus;
- Section 1.4 introduces some of the basic notions and results on the spi-calculus;
- Section 1.5 surveys the parts of paper [1] relevant to our treatment in Section 3.4.

The original contributions of the thesis are organized in two parts, each divided in chapters. Each chapter begins with an introduction, that describes its contents and its main results. Each part is followed by a short review of related works. The first part, *Static Techniques for Security*, is devoted to the study of security properties from the static point of view and it is structured in chapters as follows:

- **Chapter 2** proposes a Control Flow Analysis for the π-calculus, that shows how names will be bound to actual channels at run-time. The analysis is applied for statically checking of the properties of *no leaking* and of *no read-up/no write-down*.
- **Chapter 3** proposes a Control Flow Analysis for the spi-calculus. In this setting, the notion of confinement introduced for the π-calculus is extended and compared with the notion of secrecy introduced by Abadi. Finally, a property of message authentication, called *Authorship* is considered.
- **Chapter 4** mentions some of the related works.

The second part, *A Dynamic Approach to Security*, is devoted to the study of security properties from the dynamic point of view and it is structured in chapters as follows:

- **Chapter 5** introduces localized names. An enhanced (structural) operational semantics, based on proved transition systems, is given to the π-calculus which considers names as being *localized* to their owners. It offers a built-in mechanism, i.e. a primitive, that guarantees *message authentication*
- In **Chapter 6** the same schema of Chapter 5 is followed for the spi-calculus: the operational semantics is adapted to this calculus and an analogous primitive for authentication is given.
- **Chapter 7** mentions some of the related works.

I.10 The Origin of the Chapters

Many chapters of this thesis are based on already published papers, jointly with Pierpaolo Degano, Riccardo Focardi, Flemming Nielson and Hanne Riis Nielson and Corrado Priami. In particular:

- Chapter 2 is based on the ideas developed in [18, 19, 17].
- Chapter 5 is based on the ideas developed in [21, 20, 16].
- Chapter 6 is based on the ideas developed in [16].

Chapter 1

Preliminaries

We introduce some technical background, required for the next chapters. More precisely:

- Section 1.1 introduces some of the basic notions on transition systems;
- Section 1.2 introduces some of the basic notions and results on the π-calculus;
- Section 1.3 introduces the proved version of the π-calculus;
- Section 1.4 introduces some of the basic notions and results on the *spi*-calculus;
- Section 1.5 surveys the parts of paper [1] relevant to our treatment in Section 3.4.

1.1 Transition Systems

A system is usually a collection of entities that cooperate to carry out an activity. We assume that a system is a mathematical model of the phenomenon under study. It is characterized at time t by its internal *state* (*configuration*) that is made up of a *control program* and some *data*. This is a static description of a system. It changes its internal state according to stimuli from the environments or to particular values of data. The set of configurations that a system may pass through and their relations characterize the dynamic behaviour. Since we are intersted in the dynamic behaviour of systems, we call our models *transition systems*.

Definition 1.1.1 (transition system) *A* transition system *is a structure* $(\Gamma, \longrightarrow)$*, where* Γ *is a set of elements* γ *called* configurations *and the binary relation* $\longrightarrow \subseteq \Gamma \times \Gamma$ *is called* transition relation.

Hereafter, $\gamma \longrightarrow \gamma'$ stands for $\langle \gamma, \gamma' \rangle \in \longrightarrow$, where γ and γ' are called the *source* and, respectively, the *target* of the transition.

Sometimes more information on transitions is required. This leads to the definition of labelled transition systems.

Definition 1.1.2 (labelled transition system) *A* labelled transition system *is a structure* $(\Gamma, A, \longrightarrow)$ *where* Γ *is a set of* configurations*,* A *is a set of* labels *(sometimes called* actions *or* operations*) and* $\longrightarrow \subseteq \Gamma \times A \times \Gamma$ *is the* transition relation*.*

Hereafter, $\gamma \xrightarrow{a} \gamma'$ stands for $\langle \gamma, a, \gamma' \rangle \in \longrightarrow$, $\gamma \not\xrightarrow{a} \gamma'$ for $\langle \gamma, a, \gamma' \rangle \notin \longrightarrow$, $\gamma \longrightarrow \gamma'$ for $\exists a \in A \,.\, \langle \gamma, a, \gamma' \rangle \in \longrightarrow$, and $\gamma \not\longrightarrow$ for $\forall a \in A, \forall \gamma' \in \Gamma \,.\, \gamma \not\xrightarrow{a} \gamma'$.

We now show that a transition system can be conveniently represented through a graph in which the configurations form the nodes and the arcs represent the possible transitions between them. The dynamic behaviour of a system define the form of this graph, that we call *transition graph.* Some auxiliary definitions are needed. Hereafter, we assume as given a labelled transition system $(\Gamma, A, \longrightarrow)$.

Definition 1.1.3 (derivative) *Configuration* γ' *is an* immediate derivative *of* γ*, if* $\gamma \xrightarrow{a} \gamma'$*. It is simply a* derivative*, if* $\gamma \xrightarrow{w}{}^{*} \gamma'$*.*

We define configurations that do not occur within cycles.

Definition 1.1.4 (acyclic configurations) *Configuration* γ *is* acyclic *if there is no* w *such that* $\gamma \xrightarrow{w}{}^{+} \gamma$*.*

Sometimes is useful to identify the set of derivatives of a given configuration.

Definition 1.1.5 (set of derivatives) *The* derivative set *of a configuration* γ *is*

$$ds(\gamma) = \{\gamma' \mid \gamma \xrightarrow{w}{}^{*} \gamma'\}.$$

Note that the union of the set of derivatives of the initial states of a system define all reachable configurations.

We can now formalize the notion of derivation graph.

Definition 1.1.6 (derivation graph) *Given a configuration* γ *and its set of derivatives* $ds(\gamma)$*, the* derivation graph *of* γ *is*

$$dg(\gamma) = \langle ds(\gamma), A, \{(\gamma_i, a, \gamma_j) \mid \gamma_i \in ds(\gamma) \;\wedge\; \gamma_i \xrightarrow{a} \gamma_j\}\rangle$$

where $ds(\gamma)$ *is the set of nodes,* A *is the labelling alphabet, and the third set defines the arcs.*

The possible patterns of behaviour of a system are obtained by visiting its derivation graph. The sequences of consecutive transitions that describes the behaviour are called computations.

Definition 1.1.7 (computation) *Let $\gamma_i \xrightarrow{a} \gamma_j$ be a transition. Then, γ_i is the* source *of the transition and γ_j is its* target. *A* computation *of γ is a sequence of transitions $\gamma = \gamma_0 \xrightarrow{a_0} \gamma_1 \xrightarrow{a_1} \ldots$ starting from γ, and such that the target of any transition coincides with the source of the next one. We let $\xi, \xi', \xi_1, \ldots$ range over computations, and we write ϵ for the empty computation. The notions of source and target are extended in the obvious way to computations. We let*

$$C(\gamma) = \{\xi_i \mid \gamma \xrightarrow{\xi_i}{}^{*} \gamma_i\}$$

be the set of computations with source γ, and

$$C(\gamma, \gamma') = \{\xi_i \mid \gamma \xrightarrow{\xi_i}{}^{*} \gamma'\}$$

be the set of computations with source γ and target γ'.

Sometimes is useful to have a linearization of all computations that a system may engage in. A possibility is to get the unfolding of the derivation graph, thus yielding a tree of computations.

Definition 1.1.8 (derivation tree) *Given a configuration γ and its derivation graph $dg(\gamma)$, the* derivation tree *of γ is defined inductively as follows*

$$dt(\gamma) = \sum_{\{(\gamma, a_i, \gamma_i) \in dg(\gamma)\}} a_i \cdot dt(\gamma_i)$$

Hereafter, isomorphic derivation trees will be identified.

It is clear from the above definition that any path in the derivation tree of a configuration γ represents a computation starting at γ.

1.2 The π-calculus

In this section we briefly recall the π-calculus, a model of concurrent communicating processes based on the notion of *naming*. We follow [89].

Definition 1.2.1 (syntax) *Let $\mathcal{N}$ be a countable, infinite set of* names *ranged over by $a, b, \ldots, x, y, \ldots$, and let τ be a distinguished element such that $\mathcal{N} \cap \{\tau\} = \emptyset$.* Processes *(denoted by $P, Q, R, \ldots \in \mathcal{P}$) are built from names according to the syntax*

$$
\begin{array}{ll}
P ::= & processes \\
\quad \mathbf{0} & nil \\
\quad \pi.P & prefix \\
\quad P+P & summation \\
\quad P|P & parallel\ composition \\
\quad P+P & summation \\
\quad (\nu x)P & restriction \\
\quad [x=y]P & matching \\
\quad A(y_1,\ldots,y_k) & constant\ definition
\end{array}
$$

Alternatively, the last production can be replaced by

$$
\begin{array}{ll}
\quad !P & replication
\end{array}
$$

$$
\begin{array}{ll}
\pi ::= & prefix \\
\quad \tau & silent\ prefix \\
\quad x(y) & input \\
\quad \overline{x}y & output
\end{array}
$$

where A is an agent identifier. Hereafter, the trailing $\mathbf{0}$ *will be omitted (i.e. we will write π instead of $\pi.\mathbf{0}$).*

The prefix π is the first atomic action that the process $\pi.P$ can perform. The input prefix binds the name y in the prefixed process. Intuitively, some name y is received along the link named x. The output prefix does not bind the name y which is sent along x. The silent prefix τ denotes an action which is invisible to an external observer of the system. Summation denotes non-deterministic choice. The operator $|$ describes parallel composition of processes. The operator (νx) acts as a static binder for the name x in the process P that it prefixes. In other words, x is a unique name in P which is different from all the external names. Matching $[x = y]P$ is an `if-then` operator: process P is activated if $x = y$. Finally, each agent identifier A has a unique defining equation of the form $A(y_1, \ldots, y_k) = P$, where the y_i's are the only free names (see below) of P and $y_i \neq y_j$ if $i \neq j$. Alternatively, $!P$ behaves as many times as needed copies of P running in parallel.

The π-calculus can be equipped with an early as well as a late semantics. They essentially differ in the time in which variables are bound to names. In the late approach the placeholder of the input is instantiated when a communication occurs (see rule *Com* in Tab. 1.1), while in the early semantics it is instantiated in the axiom for the input action (see rule *Ein* in Tab. 1.2).

Late semantics

The late operational semantics of the π-calculus is defined in *SOS* (Structural Operational Semantics) style, and the labels of transitions are τ for silent actions, $x(y)$ for input, $\overline{x}y$ for free output, and $\overline{x}(y)$ for bound output. We will use μ as a metavariable for the labels of transitions (it is distinct from π, the metavariable for prefixes, though it coincides in three cases). We recall the notion of free names $fn(\mu)$, bound names $bn(\mu)$, and names $n(\mu) = fn(\mu) \cup bn(\mu)$ of a label μ. Two functions, *sbj* and *obj*, are also defined giving the subject and the object of input and output actions, respectively.

Kind	μ	$fn(\mu)$	$bn(\mu)$	$sbj(\mu)$	$obj(\mu)$
Silent move	τ	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
Free output	$\overline{x}y$	$\{x, y\}$	$\emptyset$	$\{x\}$	$\{y\}$
Input and bound output	$x(y), \overline{x}(y)$	$\{x\}$	$\{y\}$	$\{x\}$	$\{y\}$

Functions *fn*, *bn* and *n* are extended in the obvious way to processes.

We briefly recall substitutions that will be used later on. A *substitution* is a function $\sigma : \mathcal{N} \to \mathcal{N}$ which is almost everywhere the identity. If $x_i\sigma = y_i$ for all i with $1 \leq i \leq n$ (and $x\sigma = x$ for all other names x), we sometimes write $\{y_1/x_1, \ldots, y_n/x_n\}$ or $\{\tilde{y}/\tilde{x}\}$ for σ. Then, $P\sigma$ denotes the agent obtained from P by simultaneously substituting $z\sigma$ for each free occurrence of z in P for each z, with change of bound names to avoid captures. In particular the following two rules hold where $\equiv$ denotes syntactic identity

$$(x(y).P)\sigma \equiv x\sigma(y').P\{y'/y\}\sigma \text{ where } y' \notin fn((\nu\, y)P, P\sigma) \ \wedge\ y'\sigma = y'$$

and

$$(\nu\, y)P \equiv (\nu\, y')P\{y'/y\}\sigma \text{ where } y' \notin fn((\nu\, y)P, P\sigma) \ \wedge\ y'\sigma = y'.$$

Table 1.1 shows the late transition system of the π-calculus.

Early semantics

In the *early* version of the operational semantics the placeholder of the input is instantiated in the axiom for early input (Ein): $x(y).P \xrightarrow{xw} P\{w/x\}$.

Here the labels of transitions are τ for silent actions, xy for free input, $\overline{x}y$ for free output, and $\overline{x}(y)$ for bound output.

Kind	μ	$fn(\mu)$	$bn(\mu)$	$sbj(\mu)$	$obj(\mu)$
Silent move	τ	$\emptyset$	$\emptyset$		
Free input and output	$xy, \overline{x}y$	$\{x, y\}$	$\emptyset$	$\{x\}$	$\{y\}$
Bound output	$\overline{x}(y)$	$\{x\}$	$\{y\}$	$\{x\}$	$\{y\}$

Alternatively, Ein can be split in two, following [89]:

$$act : \mu.P \xrightarrow{\mu} P$$

$$par : \frac{P \xrightarrow{\mu} P'}{P|Q \xrightarrow{\mu} P'|Q}, bn(\mu) \cap fn(Q) = \emptyset \qquad sum : \frac{P \xrightarrow{\mu} P'}{P + Q \xrightarrow{\mu} P'}$$

$$res : \frac{P \xrightarrow{\mu} P'}{(\nu x)P \xrightarrow{\mu} (\nu x)P'}, x \notin n(\mu) \qquad open : \frac{P \xrightarrow{\overline{x}y} P'}{(\nu y)P \xrightarrow{\overline{x}(y)} P'}, y \neq x$$

$$close : \frac{P \xrightarrow{\overline{x}(y)} P', Q \xrightarrow{x(y)} Q'}{P|Q \xrightarrow{\tau} (\nu y)(P'|Q')} \qquad com : \frac{P \xrightarrow{\overline{x}y} P', Q \xrightarrow{x(z)} Q'}{P|Q \xrightarrow{\tau} P'|Q'\{y/z\}}$$

Table 1.1: Late transition system for the π-calculus.

$$\begin{array}{ll} FreeIn : & x(y).P \xrightarrow{xw} P\{w/y\} \\ BoundIn : & x(y).P \xrightarrow{x(y)} P \end{array}$$

Table 1.2 shows the annotated *early* transition system of the π-calculus.

1.2.1 Structural Congruence

The *structural congruence* $\equiv_P$ on processes[1] is defined as the least congruence satisfying the following clauses:

- $P \equiv_P Q$ if P and Q are α-equivalent (in symbols $P =_\alpha Q$); to be more precise: $(\nu x)P \equiv_P (\nu y)(P\{y/x\})$ if $y \notin fn((\nu x)P)$, and $x(y)P \equiv_P x(z)(P\{z/y\})$ if $z \notin fn(x(y)P)$;
- $(\mathcal{P}/_{\equiv_P}, +, \mathbf{0})$ and $(\mathcal{P}/_{\equiv_P}, |, \mathbf{0})$ are commutative monoids,
- $[x = x]P \equiv_P P$,
- $(\nu\, x)(\nu\, y)P \equiv_P (\nu\, y)(\nu\, x)P$, $(\nu\, x)P \equiv_P P$ if $x \notin fn(P)$, $(\nu\, x)(P \,|\, Q) \equiv_P (\nu\, x)P \,|\, Q$ if $x \notin fn(Q)$, and
- $A(a_1, \ldots, a_k) \equiv_P P\{a_1/y_1, \ldots, a_k/y_k\}$, if $A(y_1, \ldots, y_k) = P$
- $!P \equiv_P P|!P$.

[1]We will omit $_P$, whenever clear from the context.

$$Tau : \tau.P \xrightarrow{\tau} P \qquad Out : \overline{x}y.P \xrightarrow{\overline{x}y} P$$

$$Ein : x(y).P \xrightarrow{xw} P\{w/y\}$$

$$Par : \frac{P_1 \xrightarrow{\mu} Q_1}{P_1|P_2 \xrightarrow{\mu} Q_1|P_2}, bn(\mu) \cap fn(P_2) = \emptyset \qquad Sum : \frac{P_1 \xrightarrow{\mu} Q_1}{P_1 + P_2 \xrightarrow{\mu} Q_1}$$

$$Res : \frac{P \xrightarrow{\mu} Q}{(\nu x)P \xrightarrow{\mu} (\nu x)Q}, x \notin n(\mu) \qquad Open : \frac{P \xrightarrow{\overline{x}y} Q}{(\nu y)P \xrightarrow{\overline{x}(y)} Q}, y \neq x$$

$$Close : \frac{P_1 \xrightarrow{\overline{x}(y)} Q_1,\ P_2 \xrightarrow{x(y)} Q_2}{P_1|P_2 \xrightarrow{\tau} (\nu y)(Q_1|Q_2)} \qquad Com : \frac{P_1 \xrightarrow{\overline{x}y} Q_1,\ P_2 \xrightarrow{xy} Q_2}{P_1|P_2 \xrightarrow{\tau} Q_1|Q_2}$$

Table 1.2: Early transition system for the π-calculus.

We call a *variant* of $P \xrightarrow{\mu} Q$, a transition which only differs in that P and Q have been replaced by structurally congruent processes, and μ has been α-converted, where a name bound in μ includes Q in its scope.

The transition in the conclusion of each rule, as well as in the axioms, stands for all its variants. Alternatively, variants can be explicitly handled with a rule like

$$\frac{P' \equiv_P P \xrightarrow{\mu} Q \equiv_P Q'}{P' \xrightarrow{\mu} Q'}$$

Also, matching can be explicitly handled by semantics with the rule:

$$Match : \frac{P \xrightarrow{\mu} Q}{[x = x]P \xrightarrow{\mu} Q}$$

In this case, the congruence rule $[x = x]P \equiv_P P$ is no longer necessary.

The following proposition shows how late and early semantics are related. The proof proceeds by induction on the rules of early and late operational semantics.

Proposition 1.2.2 *Let $\longrightarrow_E$ and $\longrightarrow_L$ denote the early and the late transition relation. Then,*

1. $P \xrightarrow{\overline{x}y}_E P' \;\Leftrightarrow\; P \xrightarrow{\overline{x}y}_L P'$

2. $P \xrightarrow{\overline{x}(y)}_E P' \;\Leftrightarrow\; P \xrightarrow{\overline{x}(y)}_L P'$

3. $P \xrightarrow{xy}_E P' \;\Leftrightarrow\; \exists P'', w \,.\, P \xrightarrow{x(w)}_L P'' \wedge P' \equiv P''\{y/w\}$

4. $P \xrightarrow{\tau}_E P' \;\Leftrightarrow\; P \xrightarrow{\tau}_L P'$

1.2.2 Equivalences

The operational semantics of agents is sometimes too intensional and detailed to study the behaviour of distributed concurrent systems. Therefore behavioural equivalences based on the notion of bisimulation [103] are introduced. As usual, there is a *strong* and a *weak* version of bisimulation. The former treats all actions uniformly, while the latter abstracts from invisible or silent actions because it discards them. We only define the strong version of bisimulations, because the weak ones can be derived in the standard way.

The classical definition of bisimulations for π-calculus are the *early* and *late* ones. As for the operational semantics, the two notions differ for the moment in which instantiation of names occurs (input for the early case and communication for the late one). We start with the definition of the late equivalence.

Definition 1.2.3 (late bisimulation) *A binary relation $\mathcal{S}$ on agents is a* late simulation *if $P\,\mathcal{S}\,Q$ implies that*

- *If $P \xrightarrow{\mu} P'$ and μ is τ, $\overline{x}z$ or $\overline{x}(y)$ with $y \notin \mathrm{fn}(P,Q)$, then for some Q', $Q \xrightarrow{\mu} Q'$ and $P'\,\mathcal{S}\,Q'$*
- *if $P \xrightarrow{x(y)} P'$ and $y \notin \mathrm{fn}(P,Q)$, then for some Q', $Q \xrightarrow{x(y)} Q'$ and for all w, $P'\{w/y\}\,\mathcal{S}\,Q'\{w/y\}$*

The relation $\mathcal{S}$ is a late bisimulation *if both $\mathcal{S}$ and $\mathcal{S}^{-1}$ are late simulations. P is late bisimilar to Q (written $P \approx_L Q$) if there exists a late bisimulation $\mathcal{S}$ such that $P\,\mathcal{S}\,Q$.*

Note that late simulations have a strong requirement on bound input actions.

Definition 1.2.4 (early bisimulation) *A binary relation $\mathcal{S}$ on agents is an* early simulation *if $P\,\mathcal{S}\,Q$ implies that*

- *If $P \xrightarrow{\mu} P'$ and μ is any action with $bn(\mu) \cap \mathrm{fn}(P,Q) = \emptyset$, then for some Q', $Q \xrightarrow{\mu} Q'$ and $P'\,\mathcal{S}\,Q'$*

The relation $\mathcal{S}$ is an early bisimulation *if both $\mathcal{S}$ and $\mathcal{S}^{-1}$ are early simulations. P is early bisimilar to Q (written $P \approx_E Q$) if there exists an early bisimulation $\mathcal{S}$ such that $P\,\mathcal{S}\,Q$.*

1.2.3 Higher order π-calculus

The objects in π-calculus can be only first-order. By dropping this limitation we obtain the higher order π-calculus $HO\pi$ [114]. Names can represent processes, and thus communications may cause processes to migrate.

Syntax The syntax of π-calculus is extended as follows. Let $\mathcal{V}$ be a set of process variables ranged over by $X, Y, \ldots$. Let K stand for a process or for a name, and let U stand for a variable or for a name. We substitute V and U for objects in prefix, and U for names in constant definition of π-calculus. Thus, we have the following syntax

$$P ::= \mathbf{0} \mid \pi.P \mid P + P \mid P|P \mid (\nu x)P \mid [x = y]P \mid P(U_1, \ldots, U_n)$$

where π may be either $x(U)$ for *input*, or $\overline{x}V$ for *output* or τ for *silent* moves.

1.2.4 Operational semantics

The late operational semantics of $HO\pi$ is defined in *SOS* style and the label of the transitions are τ for silent actions, $x(K)$ for input, $\overline{x}K$ for free output and $\overline{x}(K)$ for bound output. We will still use μ as a metavariable for the actions and we assume that the notions of free names, bound names and names of a label μ are tuned according to the new syntax. Similarly, we adapt the structural congruence on processes and the notion of variant of transitions.

The transition relation of $HO\pi$ is defined by the axioms and rules in Tab. 1.3, where an auxiliary transition relation $\xrightarrow{\theta}_I$ is used. The set I contains names that can occur in a communicated process and are extruded. Rule *Close* uses I to include the receiving process as well in their scope. Of course, rule *Open* updates the set of these names, that *Close* empties (Note that in rule *Close* it is $I \subseteq fn(K)$). The actual transitions are generated by rule $Ho\pi$ that discards index I. Note that an early semantics can be defined for $Ho\pi$, as well.

$$Act: \mu.P \xrightarrow{\mu}_{\emptyset} P,\ \mu \text{ not input}$$

$$Par_0: \frac{P \xrightarrow{\mu}_I P'}{P|Q \xrightarrow{\mu}_I P'|Q}, bn(\ell(\mu)) \cap fn(Q) = \emptyset \quad Sum: \frac{P \xrightarrow{\mu}_I P'}{P+Q \xrightarrow{\mu}_I P'}$$

$$Open: \frac{P \xrightarrow{\overline{x}K}_{\emptyset} P'}{(\nu I)P \xrightarrow{\overline{x}(K)}_I P'}, x \notin I \subseteq fn(K) \qquad Com_0: \frac{P \xrightarrow{\overline{x}U}_{\emptyset} P', Q \xrightarrow{x(K)}_{\emptyset} Q'}{P|Q \xrightarrow{\tau}_{\emptyset} P'|Q'\{K/U\}}$$

$$Ide: \frac{P\{\tilde{K}/\tilde{U}\} \xrightarrow{\mu}_I P'}{Q(\tilde{K}) \xrightarrow{\mu}_I P'}, Q(\tilde{U}) = P$$

$$Close_0: \frac{P \xrightarrow{\overline{x}(K)}_I P', Q \xrightarrow{x(K)}_{\emptyset} Q'}{P|Q \xrightarrow{\tau}_{\emptyset} (\nu I)(P'|Q')},\ fn(K) \cap fn(Q) = \emptyset$$

$$Res: \frac{P \xrightarrow{\mu}_I P'}{(\nu J)P \xrightarrow{\mu}_I (\nu J)P'}, J \cap n(\ell(\mu)) = \emptyset$$

$$Ho\pi: \frac{P \xrightarrow{\mu}_I P'}{P \xrightarrow{\mu} P'}$$

Table 1.3: Late transition system of $HO\pi$.

1.3 The Proved π-calculus

In this section, we present a very concrete transition system for π-calculus (see [108]), in which labels of transitions encode (a portion of) their deduction tree. The proved transition system has the same states of the standard transition system of π-calculus.

Following [43] and [26], transitions are labelled with encodings of their proofs.

Enriched labels (*proof terms*) are defined as follows, using a function (ℓ) that takes a proof term to the corresponding standard action label.

Definition 1.3.1 (proof terms) *Let $\vartheta \in \{||_0, ||_1, +_0, +_1, (\nu\, x)\}^*$. Then the set Θ of* proof terms *(with metavariable θ) is defined by the following syntax*

$$\theta ::= \vartheta\mu \mid \vartheta\langle ||_0\vartheta_0\mu_0, ||_1\vartheta_1\mu_1\rangle$$

with $\mu_0 = xz$ iff μ_1 is either $\overline{x}z$ or $\overline{x}(z)$, or vice versa.
Function $\ell : \Theta \to Act$ is defined as

$$\ell(\theta\mu) = \mu; \quad \ell(\vartheta\langle ||_0\vartheta_0\mu_0, ||_1\vartheta_1\mu_1\rangle) = \tau.$$

Symbol $||_0$ ($||_1$) means that the left (right) component of a parallel composition is the active one in the firing of a transition. Actually, tags $||_i$ records the applications of rules for parallel composition during a derivation. Similarly, tag $+_0$ ($+_1$) records that the left (right) alternative of a nondeterministic choice is chosen. Symbol $(\nu\, x)$ says that there is a filter over name x. For example, the proof term

$$||_0(\nu\, b) +_1 a$$

describes an action a that is the right alternative of a nondeterministic choice ($+_1$), passes successfully through the filter $(\nu\, b)$, and occurs in the left location of a compound site ($||_0$).

Since labels of transitions may be long wired, we write $||_i^h$ for a sequence of h $||_i$'s. A similar convention holds for $+_i$, as well.

There is a proof term constructor for any operator of the language apart from prefix and constant definitions.

Functions *fn*, *bn* and n are extended to proof terms by letting them work on the action part alone. Furthermore, we assume the structural congruence $\equiv$ on processes defined as the least congruence satisfying the following clauses:

- P and Q α-equivalent (they only differ in the choice of bound names) $\Rightarrow P \equiv Q$,

- $[x = x]P \equiv P$,

- $(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P \equiv (\nu x, y)P$, $(\nu x)(R \mid S) \equiv (\nu x)R \mid S$ if $x \notin fn(S)$, and $(\nu x)P \equiv P$ if $x \notin fn(P)$.

Note that $|$ and $+$ are neither associative nor commutative. However, when studying behavioural properties of systems (after the relabelling of transitions in the model selected) the commutativity and associativity of the two operators may be recovered if of importance. Our choice is only needed to have a low level model that contains as much information as necessary to recover many distinct semantic models.

The late proved transition system of π-calculus is defined below. There is an early version as well (see [108]).

Definition 1.3.2 (Late proved transition system) *The* late proved transition system *of π-calculus is the triple* PTS $= (\mathcal{P}, \Theta, \longrightarrow)$. *The transition relation is defined by the axiom and rules in Tab. 1.4 (where the symmetric rules for communications are omitted). Hereafter, we write $[P]_{pts}$ for $(\mathcal{P}, \Theta, \longrightarrow, P)$ (the portion of* PTS *with P as initial state).*

Our late proved transition system of π-calculus only differs from the standard one in the rules for parallel composition and communication. Rule Par_0 (Par_1) adds to the label a tag $||_0$ ($||_1$) to record that the left (right) component is moving. The rules Com_0 and $Close_0$ have in their conclusion a pair instead of a τ to record the components which interacted. Their symmetric version Com_1 and $Close_1$ should be obvious.

The following proposition shows how standard and proved operational semantics of π-calculus are related. Its proof is similar to the one of Proposition 1.2.2, i.e. it is by induction on the proved and standard rules of operational semantics. Note that the proposition holds of late and early semantics.

Proposition 1.3.3 *Let $\longrightarrow$ and $\longrightarrow_P$ denote the standard and the proved transition relation, respectively. Then,*

1. $P \xrightarrow{\overline{x}y} P' \;\Leftrightarrow\; \exists \vartheta \,.\, P \xrightarrow{\vartheta \overline{x}y}_P P'$
2. $P \xrightarrow{\overline{x}(y)} P' \;\Leftrightarrow\; \exists \vartheta \,.\, P \xrightarrow{\vartheta \overline{x}(y)}_P P'$
3. $P \xrightarrow{x(y)} P' \;\Leftrightarrow\; \exists \vartheta \,.\, P \xrightarrow{\vartheta x(y)}_P P'$
4. $P \xrightarrow{\tau} P' \;\Leftrightarrow\; \exists \theta \,.\, P \xrightarrow{\theta}_P P' \wedge \ell(\theta) = \tau$

The above proposition says that if we relabel any computation of the proved transition system with ℓ, we obtain the standard transition system of π-calculus.

$$Act : \mu.P \xrightarrow{\mu} P$$

$$Par_0 : \frac{P \xrightarrow{\theta} P'}{P|Q \xrightarrow{||_0\theta} P'|Q}, bn(\ell(\theta)) \cap fn(Q) = \emptyset \qquad Sum_0 : \frac{P \xrightarrow{\theta} P'}{P + Q \xrightarrow{+_0\theta} P'}$$

$$Par_1 : \frac{P \xrightarrow{\theta} P'}{Q|P \xrightarrow{||_1\theta} Q|P'}, bn(\ell(\theta)) \cap fn(Q) = \emptyset \qquad Sum_1 : \frac{P \xrightarrow{\theta} P'}{Q + P \xrightarrow{+_1\theta} P'}$$

$$Res : \frac{P \xrightarrow{\theta} P'}{(\nu x)P \xrightarrow{(\nu x)\theta} (\nu x)P'}, x \notin n(\ell(\theta)) \qquad Open : \frac{P \xrightarrow{\vartheta\overline{x}y} P'}{(\nu y)P \xrightarrow{\vartheta\overline{x}(y)} P'}, x \neq y$$

$$Com_0 : \frac{P \xrightarrow{\vartheta\overline{x}y} P', Q \xrightarrow{\vartheta' x(z)} Q'}{P|Q \xrightarrow{\langle ||_0\vartheta\overline{x}y, ||_1\vartheta' x(z)\rangle} P'|Q'y/z} \qquad Ide : \frac{P\{\tilde{y}/\tilde{x}\} \xrightarrow{\theta} P'}{Q(\tilde{y}) \xrightarrow{\theta} P'}, Q(\tilde{x}) = P$$

$$Close_0 : \frac{P \xrightarrow{\vartheta\overline{x}(y)} P', Q \xrightarrow{\vartheta' x(y)} Q'}{P|Q \xrightarrow{\langle ||_0\vartheta\overline{x}(y), ||_1\vartheta' x(y)\rangle} (\nu y)(P'|Q')}$$

Table 1.4: Late proved transition system of π-calculus.

1.4 The Spi-Calculus

Syntax. In this section we briefly recall, often also literally, the spi-calculus [8, 7], in its polyadic version from [1]. This calculus is an extension of the π-calculus and has been introduced for the description and the analysis of cryptographic protocols.

Definition 1.4.1 *Let $\mathcal{N}$ be an infinite set of* names*, ranged over by n, p, q, r, and $\mathcal{V}$ an infinite set of* variables*, ranged over by x, y, w, z. Then the* terms *$\mathcal{T}$ are built according to the syntax:*

$$
\begin{array}{ll}
L, M, N ::= & \textit{terms} \\
\quad n & \quad \textit{name} \\
\quad x & \quad \textit{variable} \\
\quad (M, N) & \quad \textit{pair} \\
\quad 0 & \quad \textit{zero} \\
\quad suc(M) & \quad \textit{successor} \\
\quad \{M_1, \ldots, M_k\}_N & \quad \textit{shared} - \textit{key encryption}\ (k \geq 0)
\end{array}
$$

The set of Processes*, denoted by $P, P_1, P_2, Q, R, \dots \in \mathcal{P}$ is defined according to the syntax:*

$$
\begin{array}{ll}
P, Q, R ::= & \textit{processes} \\
\quad \mathbf{0} & \quad \textit{nil} \\
\quad \overline{M}\langle N_1, \ldots, N_k\rangle.P & \quad \textit{output} \\
\quad M(x_1, \ldots, x_k).P & \quad \textit{input} \\
\quad P|P & \quad \textit{parallel composition} \\
\quad (\nu n)P & \quad \textit{restriction} \\
\quad [M\ is\ N]P & \quad \textit{matching} \\
\quad !P & \quad \textit{replication} \\
\quad let\ (x, y) = M\ in\ P & \quad \textit{pair splitting} \\
\quad case\ M\ of\ 0 : P\ suc(x) : Q & \quad \textit{integer case} \\
\quad case\ L\ of\ \{x_1, \ldots, x_k\}_N\ in\ P & \quad \textit{shared} - \textit{key decryption}\ (k \geq 0)
\end{array}
$$

The name n and the variables x, y, $x_1, \ldots, x_k$ are bound in these processes. Hereafter, the trailing $\mathbf{0}$ will be omitted. We often write $\tilde{\ }$ to denote tuples of objects, for instance $\tilde{n}$ for the vector $(n_1, \ldots, n_r)$. Also notations are extended componentwise, e.g. $(\nu\tilde{n})$ stands for $(\nu n_1), \ldots, (\nu n_r)$.

In the standard π-calculus, of which the spi-calculus is an extension, names are the only terms. In the spi-calculus, names are distinguished from variables and constructs for pairing and numbers are added. Moreover, in order to represent encrypted message a further encryption clause is added. The term $\{M_1, \ldots, M_k\}_N$ represents the ciphertext obtained by encrypting $M_1, \ldots, M_k$ under the key N, using a shared-key cryptosystem such as DES [121].

As far as the process constructs are concerned, most of them are the same of π-calculus. The new constructs deal with composed terms:

- The process $let\ (x, y) = M\ in\ P$ behaves as $P[M_0/x, M_1/y]$ if $M = (M_0, M_1)$ and it is stuck if M is not a pair;

- The process $case\ M\ of\ 0 : P\ suc(x) : Q$ behaves as P if M is 0, as $Q[N/x]$ if $M = suc(N)$ and it is stuck otherwise;

- The process $case\ L\ of\ \{x_1, \ldots, x_k\}_N\ in\ P$ attempts to decrypt L with the key N; if L has the form $\{M_1, \ldots, M_k\}_N$, then the process behaves as $P[M_1/x_1, \ldots, M_k/x_k]$, and otherwise is stuck.

Subjects of actions are defined as arbitrary terms. The only useful cases are when they are just atomic terms, i.e. names or variables to be instantiated to names (see [8, 7]).

Implicit assumptions on cryptography are that:

- To decrypt an encrypted message is necessary to know the corresponding key: it is the only way to do it;

- an encrypted message does not reveal the key used to encrypt it;

- the decryption algorithm can detect whether a ciphertext was encrypted with the expected key.

We write $fn(M)$ and $fn(P)$ for the sets of names free in term M and process P, respectively, and $fv(M)$ and $fv(P)$ for the sets of variables free in term M and process P, respectively.

A process P is closed if $fv(P) = \emptyset$.

Congruence. Below we shall need the *structural congruence* $\equiv$ on processes, defined to be the least congruence satisfying:

- if P and Q are α-equivalent ($P =_\alpha Q$) then $P \equiv Q$; e.g.:
  $(\nu n)P \equiv (\nu m)(P[m/n])$ if $m \notin fn((\nu n)P)$, and
  $M(x_1, \ldots, x_i, \ldots, x_k)P \equiv M(x_1, \ldots, z, \ldots, x_k)(P[z/x_i])$ if
  $z \notin fv(M(x_1, \ldots, x_i, \ldots, x_k)P)$;
  $case\ L\ of\ \{x_1, \ldots, x_i, \ldots, x_k\}_N\ in\ P \equiv$
  $case\ L\ of\ \{z_1, \ldots, z_i, \ldots, z_k\}_N\ in\ (P\{z_i/y_i\})$ if
  $z_i \notin fv(case\ L\ of\ \{x_1, \ldots, x_i, \ldots, x_k\}_N\ in\ P)$;

- $(\mathcal{P}/_\equiv, |, \mathbf{0})$ is a commutative monoid;

- $(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$, if $n \neq m$,
  $(\nu n)(P_1|P_2) \equiv (\nu n)P_1|P_2$ if $n \notin fn(P_2)$, and
  $(\nu n)P \equiv P$ if $n \notin fn(P)$.

Semantics. The standard semantics for the spi-calculus is a late semantics, based on a commitment relation, that depends on the *abstraction* and *concretion* constructs:

- An abstraction has the form $(x_1, \ldots, x_k)P$, where $k \geq 0$ and $x_1, \ldots, x_k$ are variables bound in the process P.
- A concretion has the form $(\nu m_1, \ldots, \nu m_r)\langle M_1, \ldots, M_k\rangle P$, where $k, r \geq 0$ and $M_1, \ldots, M_k$ are terms, P is a process and the names $m_1, \ldots, m_r$ are bound in $M_1, \ldots, M_k$ and P.

An *agent* A or B is an abstraction, a concretion or a process.

The restrictions and the compositions are extended to abstractions and concretions in the following way:

$$(\nu m)(x_1, \ldots, x_k)P = (x_1, \ldots, x_k)(\nu m)P$$

$$R \mid (x_1, \ldots, x_k)P = (x_1, \ldots, x_k)(R \mid P)$$

$$(\nu m)(\nu n_1, \ldots, n_r)\langle \tilde{M}\rangle Q = (\nu m, n_1, \ldots, n_r)\langle \tilde{M}\rangle Q$$

$$R \mid (\nu n_1, \ldots, n_r)\langle \tilde{M}\rangle Q = (\nu n_1, \ldots, n_r)\langle \tilde{M}\rangle (R \mid Q),$$

assuming that $x_1, \ldots, x_k \notin fv(R)$, that $m \notin \{n_1, \ldots, n_r\}$ and $\{n_1, \ldots, n_r\} \cap fn(R) = \emptyset$.
If F is the abstraction $(x_1, \ldots, x_k)P$ and C the concretion $(\nu n_1, \ldots, n_r)\langle M_1, \ldots, M_k\rangle Q$ and $\{n_1, \ldots, n_r\} \cap fn(P) = \emptyset$, the interactions $F@C$ and $C@F$ are:

$$F@C = (\nu n_1), \ldots, (\nu n_r)(P[M_1/x_1, \ldots, M_k/x_k] \mid Q)$$

$$C@F = (\nu n_1), \ldots, (\nu n_r)(Q \mid P[M_1/x_1, \ldots, M_k/x_k])$$

Reduction relation. The *reduction relation* $>$ is the least relation on closed processes that satisfies the following axioms:

Red Repl
$!P > P \mid !P$

Red Match
$[M \textit{ is } M]P > P$;

Red Split
$\textit{let } (x, y) = (M, N) \textit{ in } P > P[M/x, N/y]$

Red Zero
$\textit{case } 0 \textit{ of } 0 : P \ \textit{suc}(x) : Q > P$

Comm Out

$$\frac{}{\overline{m}\langle M_1,\ldots,M_k\rangle.P \xrightarrow{\overline{m}} (\nu)\langle M_1,\ldots,M_k\rangle P}$$

Comm In

$$\frac{}{m(x_1,\ldots,x_k).P \xrightarrow{m} (x_1,\ldots,x_k)P}$$

Comm Inter 1

$$\frac{P \xrightarrow{m} F \quad Q \xrightarrow{\overline{m}} C}{P|Q \xrightarrow{\tau} F@C}$$

Comm Inter 2

$$\frac{P \xrightarrow{\overline{m}} C \quad Q \xrightarrow{m} F}{P|Q \xrightarrow{\tau} C@F}$$

Comm Par 1

$$\frac{P \xrightarrow{\mu} A}{P|Q \xrightarrow{\mu} A|Q}$$

Comm Par 2

$$\frac{Q \xrightarrow{\mu} A}{P|Q \xrightarrow{\mu} P|A}$$

Comm Res

$$\frac{P \xrightarrow{\mu} A \quad \mu \notin \{m,\overline{m}\}}{(\nu m)P \xrightarrow{\mu} (\nu m)A}$$

Comm Red

$$\frac{P > Q \quad Q \xrightarrow{\mu} A}{P \xrightarrow{\mu} A}$$

Comm Struct

$$\frac{P \equiv Q \quad Q \xrightarrow{\mu} A \quad A \equiv A'}{Q \xrightarrow{\mu} A'}$$

Table 1.5: The commitment relation.

Red Suc

$\quad$ *case* $suc(M)$ *of* $0 : P\ suc(x) : Q > Q[M/x]$

Red Decrypt

$\quad$ *case* $\{M_1,\ldots,M_k\}_N$ *of* $\{x_1,\ldots,x_k\}_N$ *in* $P > P[M_1/x_1,\ldots,M_k/x_k]$.

Commitment relation. An *action* is a name m (representing input) or a co-name $\overline{m}$ (representing output) or a distinguished *silent action* τ. Note that actions record only the subject of the input or the output action. The *commitment relation* is written $P \xrightarrow{\alpha} A$, where P is a closed process, α is an action, and A is a closed agent. The rules are reported in Tab. 1.5.

Testing Equivalence We give here the definition of testing equivalence [42, 24] directly on the spi-calculus.

A *barb* is a name m (representing input) or a co-name $\overline{m}$ (representing output). A *test* is a pair (Q, β) consisting of a closed process and a *barb* β. We say that P *passes* a test (Q, β) if and only if

$$(P|Q) \xrightarrow{\tau} Q_1 \ldots \xrightarrow{\tau} Q_n \xrightarrow{\beta} A$$

for some $n \geq 0$, some processes $Q_1, \ldots, Q_n$ and some agent A. We obtain a *testing preorder* $\sqsubseteq$ and a *testing equivalence* $\simeq$ on closed processes:

- $P \sqsubseteq P' \Leftrightarrow \forall (Q, \beta),$ if P passes (Q, β) then P' passes (Q, β)
- $P \simeq P' \Leftrightarrow P \sqsubseteq P'$ and $P' \sqsubseteq P$

A *strict barbed simulation* is a binary relation $\mathcal{S}$ on closed processes such that $P\mathcal{S}P'$ implies:

1. for every barb β, if $P \xrightarrow{\beta} A$ for some A, then $P' \xrightarrow{\beta} A$ for some A',
2. for every P_1, if $P \xrightarrow{\tau} P_1$, then there exists P_1' such that $P' \xrightarrow{\tau} P_1'$ and $P_1 \mathcal{S} P_1'$.

A *strict barbed bisimulation* is a relation $\mathcal{S}$ such that both $\mathcal{S}$ and $\mathcal{S}^{-1}$ are strict barbed equivalences.

Lemma 1.4.2 *If for every closed process Q there exists a strict barbed bisimulation $\mathcal{S}$ such that $(P|Q)\mathcal{S}(P'|Q)$, then $P \simeq P'$.*

1.5 Abadi's Type System for Secrecy

In [1], Abadi studies the secrecy of channels and of encrypted messages in the spi-calculus. We recall, sometimes literally, from there. Type soundness guarantees that there is no leaking of secret information. The notion of leaking is formalized in terms of testing equivalence [42, 24] (see Section 1.4 for a review): roughly, a process $P(x)$ does not leak the input x if a second process Q cannot distinguish running in parallel with $P(M)$ from running in parallel with $P(M')$, for every M and M'.

Principles for secrecy There are three classes or types of data:

- *PUB*: data which can be communicated to anyone;
- *SEC*: data which should not be leaked;
- *ANY*: arbitrary data.

Levels are partially ordered: type *ANY* subsumes (see the rule Level Subs) the other two types. For this reason, terms composed with sub-terms both of level *PUB* and *SEC* are typed as *ANY* (see the rules Level Pair, Level Suc, Level Encryption). The classification *SEC* is given only to data created fresh within the system to protect. Moreover, since a piece of data of level *ANY* could be of level *SEC*, it should not be leaked. At the same time, since it could be a piece of data of level *PUB*, it cannot be used as a secret. For instance it cannot be used as a key. Finally natural numbers are of level *PUB* (see the rule Level Zero). The type system mainly follows these principles:

- The data encrypted by a public key has the same classification level as the data (see the rule Level Encryption *Public*), whereas the data encrypted with a secret key are made public (see the rule Level Encryption *Secret*);
- only public data can be sent and received on public channels (see the rules Level Output *Public* and Level Input *Public*), whereas all kinds of data may be sent and received on secret channels (see the rules Level Output *Secret* and Level Input *Secret*).

Furthermore, the disciplined use of data of level *ANY* imposes the following rules:

1. terms of level *ANY* cannot be used as a keys for encryption in the rule (Level Encryption);
2. terms of level *ANY* cannot be used as subjects in (Level Output) and (Level Input) rules;
3. terms of level *ANY* cannot be compared in the rule (Level Match);

4. terms of level $\textit{ANY}$ cannot be split in the rule (Level Pair Splitting);

5. terms of level $\textit{ANY}$ cannot be examined in the rule (Level Integer case);

6. terms encrypted with a key of level $\textit{ANY}$ cannot be decrypted in the rule (Level Decryption).

The last five rules have been introduced to restrict *implicit flow* of information, i.e. the possibility that an attacker could deduce something about a certain parameter from patterns of communications, even if it is not explicitly involved in the communication. The rules forbid any comparison between sensitive data, whose result could be revealed from the kind of actions that follow the comparison

How to handle the different fields of messages depends on their implicit context. To avoid it, a standard *format* is adopted.

It is used both for all messages M encrypted with a secret key K (see the rule Level Encryption *Secret*) and for all messages M passed on a secret channel (see the rules Level Output *Secret* and Level Input *Secret*), i.e.:

- $M = (M_1 : \textit{SEC}, M_2 : \textit{ANY}, M_3 : \textit{PUB})$;

- $\{M\}_K = \{M_1 : \textit{SEC}, M_2 : \textit{ANY}, M_3 : \textit{PUB}, n\}_K$, where n is a *confounder*.

A *confounder* is a fresh value, whose aim is to guarantee that each encryption is different. Since every encrypted message includes a new confounder n, then the same ciphertext cannot be generated more than once. The typing rule (Level *Name*) check that n is used at most once.

Levels are assigned to names and variables when declared inside a binding construct (*restriction*, *input*, *pair splitting*, *integer case*, *decryption*) of processes. In the following we will refer to these *type annotated* processes with $\mathcal{P}r^t$.

The Type System Expressions are identified up to renaming of bound names and variables.

- An environment is a list of distinct names and variables with associated levels or types, where

- $dom(E)$ stands for *domain* of E;

- $x : T$ stands for a variable x with type T;

- $n : T :: \{M_1, \ldots, M_k, n\}_N$ stands for the name n with type T and associated term $\{M_1, \ldots, M_k, n\}_N$, meaning that n can be used as a confounder only in the term $\{M_1, \ldots, M_k, n\}_N$.

Type judgements are of the following kinds:

- $\vdash E : \diamond$ means that the environment E is well formed;

$$(Env-E) \quad \frac{}{\vdash \emptyset : \diamond}$$

$$(Env-V) \quad \frac{\vdash E : \diamond \quad x \notin dom(E)}{\vdash E, x : T : \diamond}$$

$$(Env-N) \quad \frac{\begin{array}{c}\vdash E : \diamond \quad n \notin dom(E) \\ \forall i \in [0,k] : E \vdash M_i : T \ \ E \vdash N : R\end{array}}{\vdash E, n : T :: \{M_1, .., M_k, n\}_N : \diamond}$$

Table 1.6: Rules for environments.

- $E \vdash M : T$ means that the term M is of class T in E;
- $E \vdash P : ok$ means that P typechecks in E.

Types rules are extended to abstractions and concretions in the expected way. Let $E \vdash A : ok^*$ be the general judgement for agent A. The rules for environment are reported in Tab. 1.6, for terms in Tab. 1.7 and for processes in Tab. 1.8. In Tab. 1.8, we use $\mathbf{T}$ to indicate $\{\textsc{pub}, \textsc{sec}\}$.

Correctness Well-formedness is preserved by the operational semantics.

Theorem 1.5.1 (Subject Reduction)

1. *If* $E \vdash P : ok$,
2. $E \vdash \sigma$, *i.e. if for each* x, $\sigma(x)$ *is a closed term such that* $fn(\sigma(x)) \subseteq dom(E)$ *for every* $x \in dom(E)$;
3. *all variables in* $dom(E)$ *are of level* ANY, *then*

1. *if* $P\sigma > Q'$, *then there exists a process* Q *such that*
   - $Q' = Q\sigma$,
   - $E \vdash Q : ok$,
   - $P\sigma' > Q\sigma'$, *whenever* $E \vdash \sigma'$.
2. *if* $P\sigma \xrightarrow{\alpha} A$, *then there exists an agent* A *such that*
   - $A' = A\sigma$,

$$(Level\ Subs)\qquad \frac{E \vdash M : T \qquad T \leq R}{E \vdash M : R}$$

$$(Level\ Var)\qquad \frac{\vdash E : \diamond \quad x : T \in dom(E)}{E \vdash x : T}$$

$$(Level\ Name)\qquad \frac{\vdash E : \diamond \quad n : T :: \{M_1, .., M_k, n\}_N \in dom(E)}{E \vdash n : T}$$

$$(Level\ Zero)\qquad \frac{\vdash E : \diamond}{E \vdash 0 : \textit{PUB}}$$

$$(Level\ Suc)\qquad \frac{E \vdash M : T}{E \vdash suc(M) : T}$$

$$(Level\ Pair)\qquad \frac{E \vdash M : T,\ E \vdash N : T}{E \vdash (M, N) : T}$$

$$(Level\ Enc - Pub)\qquad \frac{\forall i \in [1, k] :\ E \vdash M_i : T,\ E \vdash N : \textit{PUB}}{E \vdash \{M_1, \ldots, M_k\}_N : T}$$
$$k = 0 \Rightarrow T = \textit{PUB}$$

$$(Level\ Enc - Sec)\qquad \frac{\begin{array}{c} E \vdash M_1 : \textit{SEC},\ E \vdash M_2 : \textit{ANY},\ E \vdash M_3 : \textit{PUB} \\ E \vdash N : \textit{SEC} \quad n : T :: \{M_1, M_2, M_3, n\}_N \text{ in } E \end{array}}{E \vdash \{M_1, M_2, M_3, n\}_N : Pub}$$

Table 1.7: Rules for terms.

$$(Level\ Out{-}Pub)\quad \frac{\begin{array}{c}E \vdash M : \textsc{pub}\\ \forall i \in [1,k] : E \vdash M_i : \textsc{pub} \;\; E \vdash P : ok\end{array}}{E \vdash \overline{M}\langle M_1, \ldots, M_k\rangle.P : ok}$$

$$(Level\ Out{-}Sec)\quad \frac{\begin{array}{c}E \vdash M_1 : \textsc{sec},\ E \vdash M_2 : \textsc{any},\ E \vdash M_3 : \textsc{pub}\\ E \vdash M : \textsc{sec} \;\; E \vdash P : ok\end{array}}{E \vdash \overline{M}\langle M_1, M_2, M_3\rangle.P : ok}$$

$$(Level\ In{-}Pub)\quad \frac{\begin{array}{c}E \vdash M : \textsc{pub}\\ E, x_1 : \textsc{pub}, \ldots, x_k : \textsc{pub} \vdash P : ok\end{array}}{E \vdash M(x_1, \ldots, x_k).P : ok}$$

$$(Level\ In{-}Sec)\quad \frac{\begin{array}{c}E \vdash M : \textsc{sec}\\ E, x_1 : \textsc{sec}, x_2 : \textsc{any}, x_3 : \textsc{pub} \vdash P : ok\end{array}}{E \vdash M(x_1, x_2, x_3).P : ok}$$

$$(Level\ Nil)\quad \frac{\vdash E : \diamond}{E \vdash \mathbf{0} : ok} \qquad (Level\ Par)\frac{E \vdash P_1 : ok \quad E \vdash P_2 : ok}{E \vdash P_1 | P_2 : ok}$$

$$(Level\ Repl)\quad \frac{E \vdash P : ok}{E \vdash !P : ok} \qquad (Level\ Res)\quad \frac{E, n : T :: L \vdash P : ok}{E \vdash (\nu n : T)P : ok}$$

$$(Level\ Match)\quad \frac{E \vdash M : T,\ E \vdash N : R, \ \ \vdash P : ok}{\vdash [M is N]P : ok},\ T, R \in \mathbf{T}$$

$$(Level\ Split)\quad \frac{E \vdash M : T,\ E, x : T, y : T \vdash P : ok}{E \vdash let\ (x, y) = M\ in\ P : ok},\ T \in \mathbf{T}$$

$$(Level\ Case)\quad \frac{E \vdash M : T,\ E \vdash P : ok,\ E, x : T \vdash Q : ok}{E \vdash case\ M\ of\ 0 : P\ suc(x) : Q : ok},\ T \in \mathbf{T}$$

$$(Level\ Dec{-}Pub)\quad \frac{\begin{array}{c}E \vdash L : T,\ E \vdash N : \textsc{pub}\\ E, x_1 : T, \ldots, x_k : T \vdash P : ok\end{array}}{E \vdash case\ L\ of\ \{x_1, \ldots, x_k\}_N\ in\ P : ok},\ T \in \mathbf{T}$$

$$(Level\ Dec{-}Sec)\quad \frac{\begin{array}{c}E \vdash L : T,\ E \vdash N : \textsc{sec}\\ E, x_1 : \textsc{sec}, x_2 : \textsc{any}, x_3 : \textsc{pub}, x_4 : \textsc{any} \vdash P : ok\end{array}}{E \vdash case\ L\ of\ \{x_1, x_2, x_3, x_4\}_N\ in\ P : ok},\ T \in \mathbf{T}$$

Table 1.8: Rules for processes.

- $E \vdash A : ok^*$,
- $P\sigma' > A\sigma'$, *whenever* $E \vdash \sigma'$.

Main Theorem If a process typechecks, then it does not reveal the values of parameters of level ANY. More precisely:

Theorem 1.5.2 *Given an environment E, suppose that only variables of level* ANY *and names of level* PUB *are in $dom(E)$. Suppose further that $E \vdash \sigma$ and $E \vdash \sigma'$. If $E \vdash P : ok$, then $P\sigma \simeq P\sigma'$.*

This means that an observer cannot detect the difference between $P\sigma$ and $P\sigma'$.

Part I

Static Techniques for Security

Chapter 2

Control Flow Analysis for the π-Calculus

In this chapter, we present a Control Flow Analysis for the π-calculus that statically predicts how names will be bound to actual channels at run-time. More precisely, it establishes a super-set of the set of channels to which a given name may be bound and of the set of channels that may be sent along a given channel. These super-sets give rise to *solutions*. The only extensions made to the standard syntax of the calculus consist in assigning explicit "channels" χ to the names occurring within restrictions, and assigning explicit "binders" β to the names occurring within input prefixes. Annotations do not affect the semantics and can be done mechanically, e.g. assigning different markers for different objects.

We formulate the Control Flow Analysis as a specification of the correctness of a candidate solution. We show that best solutions always exist and we establish semantic correctness in the form of a subject-reduction result. We then present a procedure that generates solutions by inducing on the structure of processes, and operates in $O(N^5)$ time with respect to the size N of the process under consideration.

We apply our analysis for statically checking two simple security properties. The first property considers channels as divided into "secret" and "public" ones. Then, the dynamic security requirement is that secret information may only be communicated over secret channels; in other words a process has no leaks of secret information. This notion is similar to the secrecy notion a là Dolev and Yao, as expressed by Abadi for the spi-calculus in [4]. With simple checks on a solution, we obtain a static test (called *confinement*) for a given process having no leaks, and we prove it safe with respect to the dynamic notion. The second property presented in Section 2.5 is a variant of the no read-up/no write-down property of Bell and LaPadula [14]. Processes are given levels of security clearance, and the dynamic property demands that those at high level never send information to those at low level, while communications in any other direction is permitted. A little extension to our machinery is sufficient to define a static check (called *discreetness*) for when a process respects the classification hierarchy, and to prove it safe with respect to the dynamic notion. Finally, we briefly discuss some possible extensions.

2.1 Instrumenting the π-calculus

The formulation of our analysis only requires a minor extension to the syntax of the π-calculus, namely annotating the binding occurrences of names within restrictions with "channels" χ, and the binding occurrences of names within input prefixes with "binders" β. A channel is intended here as an abstract value indicating a real channel. These syntactic extensions are needed because of the α-conversion allowed by the structural congruence. These annotations will by no means affect the dynamic semantics of the π-calculus; however, they heavily influence the way in which our static analysis of Section 2.2 operates. From the point of view of the analysis, annotations place all the names of a process in a finite set of equivalence classes. Through them, the analysis computes (a super-set of) the actual links that a name can denote.

Syntax.

Definition 2.1.1 *Let $\mathcal{B}$ be a non-empty set of* binders *ranged over by $\beta, \beta', \cdots$; and let $\mathcal{C}$ be a non-empty set of* channels *ranged over by $\chi, \chi', \cdots$ such that $\mathcal{B} \cap \mathcal{C} = \emptyset$; moreover call $\mathcal{B} \cup \mathcal{C}$ the set of* markers. *Then* (annotated) processes, *denoted by $P, P_1, P_2, Q, R, \cdots \in Proc$ are built as in Def. 1.2.1 (with replication construct), where the (annotated) input prefix $x(y^\beta)$ replaces $x(y)$ and the (annotated) restriction $(\nu x^\chi)P$ replaces $(\nu x)P$ and where the replication construct is chosen.*

Semantics. We consider the *early* operational semantics defined in SOS style for the π-calculus, because it is emerging as a standard for transitional semantics and appears to be more suitable for the security issues studied in the next sections. We follow [89], in particular for the distinction between free and bound input.

The labels μ of transitions are τ for silent actions, xy for free input, $\overline{x}y$ for free output, $x(y^\chi)$ for bound input and $\overline{x}(y^\chi)$ for bound output. The notion of free names $fn(\mu)$, bound names $bn(\mu)$, and names $n(\mu) = fn(\mu) \cup bn(\mu)$ are extended in the following obvious way. The same hold for the functions sbj and obj, that return respectively, the subject x and the object y of input and output actions, i.e. the channel x on which y is transmitted.

Kind	μ	$fn(\mu)$	$bn(\mu)$	$sbj(\mu)$	$obj(\mu)$
Silent move	τ	$\emptyset$	$\emptyset$		
Free input and output	$xy, \overline{x}y$	$\{x, y\}$	$\emptyset$	$\{x\}$	$\{y\}$
Bound input and output	$x(y^\chi), \overline{x}(y^\chi)$	$\{x\}$	$\{y\}$	$\{x\}$	$\{y\}$

$Tau : \tau.P \xrightarrow{\tau} P$	$Out : \overline{x}y.P \xrightarrow{\overline{x}y} P$
$FreeIn : x(y^\beta).P \xrightarrow{xw} P\{w/y\}$	$BoundIn : x(y^\beta).P \xrightarrow{x(y^\chi)} P$
$Par : \dfrac{P_1 \xrightarrow{\mu} Q_1}{P_1\|P_2 \xrightarrow{\mu} Q_1\|P_2}, bn(\mu) \cap fn(P_2) = \emptyset$	$Sum : \dfrac{P_1 \xrightarrow{\mu} Q_1}{P_1 + P_2 \xrightarrow{\mu} Q_1}$
$Res : \dfrac{P \xrightarrow{\mu} Q}{(\nu x^\chi)P \xrightarrow{\mu} (\nu x^\chi)Q}, x \notin n(\mu)$	$Open : \dfrac{P \xrightarrow{\overline{x}y} Q}{(\nu y^\chi)P \xrightarrow{\overline{x}(y^\chi)} Q}, y \neq x$
$Close : \dfrac{P_1 \xrightarrow{\overline{x}(y^\chi)} Q_1,\ P_2 \xrightarrow{x(y^\chi)} Q_2}{P_1\|P_2 \xrightarrow{\tau} (\nu y^\chi)(Q_1\|Q_2)}$	$Com : \dfrac{P_1 \xrightarrow{\overline{x}y} Q_1,\ P_2 \xrightarrow{xy} Q_2}{P_1\|P_2 \xrightarrow{\tau} Q_1\|Q_2}$
$Var : \dfrac{P' \equiv P \xrightarrow{\mu} Q \equiv Q'}{P' \xrightarrow{\mu} Q'}$	$Match : \dfrac{P \xrightarrow{\mu} Q}{[x = x]P \xrightarrow{\mu} Q}$

Table 2.1: Early annotated transition system for the π-calculus.

Congruence. The *structural congruence* $\equiv$ on processes is the same of that of the standard π-calculus, apart from the following rule, that permits to exchange restrictions only when restricted names are different, because otherwise $(\nu x^\chi)P \equiv (\nu x^{\chi'})P$.

- $(\nu x^\chi)(\nu y^{\chi'})P \equiv (\nu y^{\chi'})(\nu x^\chi)P$, if $x \neq y$.

Note that α-conversions do not affect markers.

Table 2.1 shows the (annotated) *early* transition system of the π-calculus defined in SOS style.

We now present a straightforward fact that will be repeatedly used in the proofs later on.

Fact 2.1.2 *If $P \xrightarrow{\mu} Q$ then*

(1) If $\mu = \tau$ then $fn(P) \supseteq fn(Q)$.

(2) If $\mu = \overline{x}y$ then $fn(P) \supseteq \{x, y\} \cup fn(Q)$.

(3) If $\mu = \overline{x}(y^\chi), xy, x(y^\chi)$ then $fn(P) \supseteq \{x\} \cup (fn(Q) \setminus \{y\})$.

Another possible treatment of *Match* A different treatment of matching is possible and is presented in [18]. There, the structural congruence law $[x = x]P \equiv P$ is assumed and the transitional rule *Match* is removed from Tab. 2.1 (see also Section 1.2). In that case an accurate handling of free names is required, otherwise applying the congruence rule from right to left may introduce new free names *ad libitum*. The need of handling similar kinds of low level details is a recurrent problem in congruence-based semantics, and in [18] one of the techniques needed to deal with them is presented.

This amounts to introduce the *free interesting names* $\mathit{fin}(P)$ of a process P, i.e. all the free names, apart from those occurring in $[- = -]$. The formal definition is as follows:

$$\begin{gathered}
\mathit{fin}(\mathbf{0}) = \emptyset \\
\mathit{fin}(\tau.P) = \mathit{fin}([x = y]P) = \mathit{fin}(!P) = \mathit{fin}(P) \\
\mathit{fin}(\overline{x}y.P) = \mathit{fin}(P) \cup \{x, y\} \\
\mathit{fin}(x(y^\beta).P) = (\mathit{fin}(P) \setminus \{y\}) \cup \{x\} \\
\mathit{fin}(P_1 + P_2) = \mathit{fin}(P_1|P_2) = \mathit{fin}(P_1) \cup \mathit{fin}(P_2) \\
\mathit{fin}((\nu x^\chi)P) = \mathit{fin}(P) \setminus \{x\}
\end{gathered}$$

So $\mathit{fin}([x = y]P) = \mathit{fin}(P)$, whereas $\mathit{fn}([x = y]P) = \mathit{fn}(P) \cup \{x, y\}$.

The introduction of free interesting names leads to a couple of auxiliary results. The first item is just an analogue of an obvious property that holds for the free names. The second item is more interesting because there is no analogue for the set of free names: clearly $[x = x]\mathbf{0} \equiv \mathbf{0}$, while $\mathit{fn}([x = x]\mathbf{0}) \neq \mathit{fn}(\mathbf{0})$.

Lemma 2.1.3 *We have the following:*

- $\mathit{fin}(P\{w/y\}) = \begin{cases} (\mathit{fin}(P) \setminus \{y\}) \cup \{w\} & \text{if } y \in \mathit{fin}(P) \\ \mathit{fin}(P) & \text{if } y \notin \mathit{fin}(P) \end{cases}$

- *If* $P \equiv Q$ *then* $\mathit{fin}(P) = \mathit{fin}(Q)$.

Moreover, there is a result that ensures that the set of free interesting names may grow when performing bound outputs or inputs.

Lemma 2.1.4 *We have the following:*

- *If* $P \xrightarrow{\tau} Q$ *then* $\mathit{fin}(P) \supseteq \mathit{fin}(Q)$;

- *If* $P \xrightarrow{\overline{x}y} Q$ *then* $\mathit{fin}(P) \supseteq \{x, y\} \cup \mathit{fin}(Q)$;

- *If* $P \xrightarrow{\overline{x}(y^\chi)} Q$ *then* $\mathit{fin}(P) \supseteq \{x\} \cup (\mathit{fin}(Q) \setminus \{y\})$;

- *If* $P \xrightarrow{xy} Q$, *then* $\mathit{fin}(P) \supseteq \{x\} \cup (\mathit{fin}(Q) \setminus \{y\})$.
- *If* $P \xrightarrow{x(y^\chi)} Q$, *then* $\mathit{fin}(P) \supseteq \{x\} \cup (\mathit{fin}(Q) \setminus \{y\})$.

If *fin* were adopted, Lemma 2.1.4 would replace Fact 2.1.2 and all the following assumptions made on $me[\mathit{fn}(P)]$ would be replaced by the corresponding assumptions on $me[\mathit{fin}(P)]$. A part from this slight modifications, all the following results would still hold.

2.2 Control Flow Analysis

The result of analysing a process P is a pair (ρ, κ). The first component, ρ, is an abstract environment which gives information about the set of channels to which names can be bound; the second component, κ, is an abstract channel environment which gives information about the set of channels that can flow over given channels.

One way to view the pair (ρ, κ) is as a record of the actual communications taking place during execution. Whenever a value a_{val} is output on some channel b_{chan}, as in $\overline{b_{chan}} a_{val}$, it must be duly recorded in the κ component, intuitively by ensuring that $a_{val} \in \kappa(b_{chan})$. Similarly, whenever a variable c_{var} inputs a value on some channel b_{chan}, as in $b_{chan}(c_{var})$, this must also be duly recorded in the ρ component, intuitively by ensuring that $a_{val} \in \rho(c_{var})$ for all $a_{val} \in \kappa(b_{chan})$.

We now make this more precise (also paying attention to an additional marker environment me for associating names with markers).

2.2.1 Validation

To *validate* the correctness of a proposed solution (ρ, κ) we state a set of clauses operating upon judgments of the form:

$$(\rho, \kappa) \models_{me} P$$

The purpose of me, ρ and κ is clarified by:

- $me : \mathcal{N} \to (\mathcal{B} \cup \mathcal{C})$ is the *marker environment* that maps a name (in particular the free names of a process) to the appropriate channel or binder used when the name was introduced; so $me(x)$ will be the marker (in $\mathcal{B}$ or $\mathcal{C}$) where the current name x is bound.

- $\rho : \mathcal{B} \to \wp(\mathcal{C})$ is the *abstract environment* that maps a binder to the set of channels that it can be bound to; more precisely, $\rho(\beta)$ must include the set of channels that β could evaluate to. We shall allow to regard the abstract environment as a function $\rho : (\mathcal{B} \cup \mathcal{C}) \to \wp(\mathcal{C})$ by setting $\forall \chi : \rho(\chi) = \{\chi\}$.

  We write $\perp$ for the function that maps everything to $\emptyset$. However, we continue to assume that $\forall \chi \in \mathcal{C} : \perp(\chi) = \{\chi\}$

- $\kappa : \mathcal{C} \to \wp(\mathcal{C})$ is the *abstract channel environment* that maps a channel to the set of channels that can be communicated over it; more precisely, $\kappa(\chi)$ must include the set of channels that can be communicated over the channel χ.

  Also here we write $\perp$ for the function that maps everything to $\emptyset$.

Note that we use a marker environment, because the identity of names is not preserved under α-conversions (see the rule Var). Indeed, it would not suffice to "α-rename the program apart" because this property is not preserved under reduction, in particular when scope extrusion is required. For example, the process $(\nu a^{\chi'})(a(y^{\beta}).a(z^{\beta'}).\overline{y}z \mid !(\nu x^{\chi})\overline{a}x)$ performs a first communication, then α-converts the name x to perform a second communication and becomes $(\nu a^{\chi'})(\nu x^{\chi})(\nu w^{\chi})(\overline{x}w \mid !(\nu x^{\chi})\overline{a}x)$.

A further comment on annotations may clarify their subsequent use. A typical schema for annotating the occurrences of restricted names and of objects of inputs in a process P is to keep all the χ's and the β's distinct; also, the marker environment me should map the free names of P to fresh channels. Note that annotating a process in this way is merely mechanical and involves no knowledge about its behaviour.

The detailed definition of our Control Flow Analysis is given by the Flow Logic in Tab. 2.2, where we often write $me[x \mapsto \eta]$ to indicate that the me is updated with the new association of the name x with the marker η, overwriting a possible previous association. All the rules dealing with a compound process require that the components are validated, apart from the one for matching. Moreover, the second conjunct of the rule for output requires that the set of channels that can be communicated along each element of $\rho(me(x))$ includes the channels to which y can evaluate. Symmetrically, the rule for input demands that the set of the channels that can pass along x is included in the set of channels to which y can evaluate. In the clause for restriction, we can simply update the marker environment as $me[x \mapsto \chi]$ because $\rho(\chi) = \{\chi\}$ by definition. The condition for matching says that P needs to be validated if there is at least one channel to which both x and y can evaluate, in which case we analyze the continuation.

Example 2.2.1 Consider the following process

$$P = a(x^{\beta_0}).(\nu b^{\chi_0})(\nu c^{\chi_1})((\overline{b}a.\overline{x}x.b(x^{\beta_1}).\overline{x}c \ + \ \overline{b}d.\overline{a}c) \mid b(x^{\beta_2}).\overline{b}x) \mid d(x^{\beta_3}),$$

the marker environment me such that $me(a) = \chi_2$ and $me(d) = \chi_3$, and the pair (ρ, κ) defined as follows, where $i \in \{0, 1, 2, 3, 4\}$

$$\rho(\beta_i) = \begin{cases} \{\chi_0, \chi_1, \chi_2, \chi_3, \chi_4\} & \text{if } i = 1, 2 \\ \{\chi_1, \chi_2, \chi_3, \chi_4\} & \text{if } i = 0, 3 \end{cases} \quad \kappa(\chi_i) = \begin{cases} \{\chi_0, \chi_1, \chi_2, \chi_3, \chi_4\} & \text{if } i = 0 \\ \{\chi_1, \chi_2, \chi_3, \chi_4\} & \text{if } i \geq 1 \end{cases}$$

A simple check shows that $(\rho, \kappa) \ \models_{me} P$. The objects x of the inputs on channels a and d are kept distinct for the analysis, because they are put by the annotations β_0 and β_3 in two different equivalence classes (but this does not influence the dynamic semantics). The reader may have noticed that (ρ, κ) above is not the least solution,

(ρ, κ)	$\models_{me} \mathbf{0}$	iff	$true$
(ρ, κ)	$\models_{me} \tau.P$	iff	$(\rho, \kappa) \models_{me} P$
(ρ, κ)	$\models_{me} \overline{x}y.P$	iff	$(\rho, \kappa) \models_{me} P \wedge$ $\forall \chi \in \rho(me(x)) : \rho(me(y)) \subseteq \kappa(\chi)$
(ρ, κ)	$\models_{me} x(y^{\beta}).P$	iff	$(\rho, \kappa) \models_{me[y \mapsto \beta]} P \wedge$ $\forall \chi \in \rho(me(x)) : \kappa(\chi) \subseteq \rho(\beta)$
(ρ, κ)	$\models_{me} P_1 + P_2$	iff	$(\rho, \kappa) \models_{me} P_1 \wedge (\rho, \kappa) \models_{me} P_2$
(ρ, κ)	$\models_{me} P_1 \mid P_2$	iff	$(\rho, \kappa) \models_{me} P_1 \wedge (\rho, \kappa) \models_{me} P_2$
(ρ, κ)	$\models_{me} (\nu x^{\chi})P$	iff	$(\rho, \kappa) \models_{me[x \mapsto \chi]} P$
(ρ, κ)	$\models_{me} [x = y]P$	iff	$(\rho(me(x)) \cap \rho(me(y)) \neq \emptyset$ $\Rightarrow (\rho, \kappa) \models_{me} P$
(ρ, κ)	$\models_{me} !P$	iff	$(\rho, \kappa) \models_{me} P$

Table 2.2: Control flow analysis for the π-calculus

e.g. because of the presence of χ_4. This kind of useless channels may occur in solutions, although they do appear neither in annotations nor in the image of a marker environment – nor they will occur in the solutions constructed according to Section 2.3 (see also Theorem 2.2.4). ∎

The formulation of our Control Flow Analysis, called 0-CFA, borrows from standard ideas for functional languages. In particular, this means that it is insensitive to flow and context [97], so terms can be re-arranged without affecting the acceptability of a candidate solution; in effect, restrictions can be lifted to the top-level, or to the nearest enclosing !, and prefixing of actions can be replaced by their parallel composition. While more complex flow analyses can be devised, these are not necessary for the applications to security studied here.

Possible optimizations Our analysis can be refined by using more demanding clauses. Clauses can, for instance, subordinate the analysis of the continuation of a process, to the effective *reachability* of that point of the program.

2.2.2 Existence of solutions

So far we have only considered a procedure for validating whether or not a proposed solution (ρ, κ) is in fact acceptable. We now show that there always exists a least choice of (ρ, κ) that is acceptable in the manner of Tab. 2.2.

Definition 2.2.2 *The set of proposed solutions can be partially ordered by setting* $(\rho, \kappa) \sqsubseteq (\rho', \kappa')$ *iff* $\forall \beta \in \mathcal{B} : \rho(\beta) \subseteq \rho'(\beta)$ *and* $\forall \chi \in \mathcal{C} : \kappa(\chi) \subseteq \kappa'(\chi)$.

It is immediate that this suffices for making the set of proposed solutions into a complete lattice; using standard notation we write $(\rho, \kappa) \sqcup (\rho', \kappa')$ for the binary least upper bound (defined pointwise), $\sqcap \mathcal{I}$ for the greatest lower bound of a set $\mathcal{I}$ of proposed solutions (also defined pointwise), and $(\bot, \bot)$ for the least element.

Definition 2.2.3 *A set $\mathcal{I}$ of proposed solutions is a* Moore family *if and only if it contains $\sqcap \mathcal{J}$ for all $\mathcal{J} \subseteq \mathcal{I}$ (in particular $\mathcal{J} = \emptyset$ and $\mathcal{J} = \mathcal{I}$).*

This concept plays an important rôle in the theory of Abstract Interpretation [36]; in other branches of Computer Science it is sometimes called the model intersection property. When $\mathcal{I}$ is a Moore family it contains a greatest element $(\sqcap \emptyset)$ as well as a least element $(\sqcap \mathcal{I})$. The following theorem then guarantees that there always is a least solution to the specification in Tab. 2.2 (just take $(\overline{\rho}, \overline{\kappa}) = (\bot, \bot)$ in the statement below).

Theorem 2.2.4 *For all me, P and $(\overline{\rho}, \overline{\kappa})$ the set*

$$\{(\rho, \kappa) \mid (\rho, \kappa) \models_{me} P \wedge (\rho, \kappa) \sqsupseteq (\overline{\rho}, \overline{\kappa})\}$$

is a Moore family.

Proof. We proceed by structural induction on P (since Tab. 2.2 is defined by structural induction on P). Let

$$\mathcal{J} \subseteq \{(\rho, \kappa) \mid (\rho, \kappa) \models_{me} P \wedge (\rho, \kappa) \sqsupseteq (\overline{\rho}, \overline{\kappa})\}$$

and let J and (ρ_j, κ_j) be given such that $\mathcal{J} = \{(\rho_j, \kappa_j) \mid j \in J\}$. Next define

$$(\rho', \kappa') = \sqcap \mathcal{J} = \sqcap \{(\rho_j, \kappa_j) \mid j \in J\}$$

and recall that the greatest lower bound is defined pointwise and hence that $(\rho', \kappa') \sqsupseteq (\overline{\rho}, \overline{\kappa})$. It remains to check that $(\rho', \kappa') \models_{me} P$. For this we proceed by cases on P making use of the induction hypothesis. Most cases are straightforward and here we only consider two of the more interesting ones.

The case $x(y^\beta).P$. Since $\forall j \in J : (\rho_j, \kappa_j) \models_{me} x(y^\beta).P$ we have

$$\forall j \in J : (\rho_j, \kappa_j) \models_{me[y \mapsto \beta]} P \qquad \text{and} \qquad \forall j \in J : \forall \chi \in \rho_j(me(x)) : \kappa_j(\chi) \subseteq \rho_j(\beta)$$

Using the induction hypothesis and the fact that ρ' is obtained in a pointwise manner, we then obtain

$$(\rho', \kappa') \models_{me[y \mapsto \beta]} P \qquad \text{and} \qquad \forall \chi \in \rho'(me(x)) : \kappa'(\chi) \subseteq \rho'(\beta)$$

thus establishing the desired $(\rho', \kappa') \models_{me} x(y^\beta).P$.

The case $[x = y]P$. Since $\forall j \in J : (\rho_j, \kappa_j) \models_{me} [x = y]P$ we have

$$\forall j \in J : (\rho_j(me(x)) \cap \rho_j(me(y)) \neq \emptyset \Rightarrow (\rho_j, \kappa_j) \models_{me} P$$

Using the induction hypothesis and the fact that ρ' is obtained in a pointwise manner, we then obtain

$$(\rho'(me(x)) \cap \rho'(me(y)) \neq \emptyset \Rightarrow (\rho', \kappa') \models_{me} P$$

thus establishing the desired $(\rho', \kappa') \models_{me} [x = y]P$. ■

2.2.3 Correctness

We state now some auxiliary results that will allow us to establish the semantic correctness of our analysis; they are all independent of the operational semantics and only rely on Tab. 2.2.

Lemma 2.2.5 *Assume that* $\forall x \in fn(P) : me_1(x) = me_2(x)$*; then* $(\rho, \kappa) \models_{me_1} P$ *if and only if* $(\rho, \kappa) \models_{me_2} P$.

Proof. A straightforward structural induction on P. ■

Lemma 2.2.6 *Assume that* $me(y) = me(z)$*; then* $(\rho, \kappa) \models_{me} P$ *if and only if* $(\rho, \kappa) \models_{me} P\{z/y\}$.

Proof. The proof is by induction on the size of P. Most cases are straightforward using the fact that $\forall x : me(x) = me(x\{z/y\})$. This leaves us with the cases where the marker environment is modified and here we consider only a typical case.

The case $P = u(v^\beta).Q$. If $v = y$ the result follows from the above remarks so assume that $v \neq y$. Let w be a fresh name, i.e. let $w \notin fn(Q) \cup \{z, y\}$, in case $z = v$, and let $w = v$ in case $z \neq v$; in both cases $me[w \mapsto \beta](z) = me[w \mapsto \beta](y)$. Then

$$P\{z/y\} = u\{z/y\}(w^\beta).(Q\{w/v\}\{z/y\})$$

and it follows that

$$(\rho, \kappa) \models_{me} P\{z/y\}$$

amounts to

$$(\rho, \kappa) \models_{me[w\mapsto\beta]} Q\{w/v\}\{z/y\} \quad \text{and} \quad \forall \chi \in \rho(me(u\{z/y\})) : \kappa(\chi) \subseteq \rho(\beta)$$

and by the induction hypothesis this amounts to

$$(\rho, \kappa) \models_{me[w\mapsto\beta]} Q\{w/v\} \quad \text{and} \quad \forall \chi \in \rho(me(u)) : \kappa(\chi) \subseteq \rho(\beta)$$

which by Lemma 2.2.5 amounts to

$$(\rho, \kappa) \models_{me[w\mapsto\beta, v\mapsto\beta]} Q\{w/v\} \quad \text{and} \quad \forall \chi \in \rho(me(u)) : \kappa(\chi) \subseteq \rho(\beta)$$

so that by the induction hypothesis this amounts to

$$(\rho, \kappa) \models_{me[w\mapsto\beta, v\mapsto\beta]} Q \quad \text{and} \quad \forall \chi \in \rho(me(u)) : \kappa(\chi) \subseteq \rho(\beta)$$

which by Lemma 2.2.5 amounts to

$$(\rho, \kappa) \models_{me[v\mapsto\beta]} Q \quad \text{and} \quad \forall \chi \in \rho(me(u)) : \kappa(\chi) \subseteq \rho(\beta)$$

which again amounts to

$$(\rho, \kappa) \models_{me} P$$

as was to be shown. ■

Corollary 2.2.7 *Assume that* $z \notin fn(P)$ *and* $\eta \in \mathcal{B} \cup \mathcal{C}$*; then* $(\rho, \kappa) \models_{me[y\mapsto\eta]} P$ *if and only if* $(\rho, \kappa) \models_{me[z\mapsto\eta]} P\{z/y\}$.

Proof. By Lemma 2.2.6

$$(\rho, \kappa) \models_{me[y \mapsto \eta, z \mapsto \eta]} P \quad \text{iff} \quad (\rho, \kappa) \models_{me[y \mapsto \eta, z \mapsto \eta]} P\{z/y\}$$

and by Lemma 2.2.5 and $z \notin fn(P)$

$$(\rho, \kappa) \models_{me[y \mapsto \eta]} P \quad \text{iff} \quad (\rho, \kappa) \models_{me[z \mapsto \eta]} P\{z/y\}$$

as was to be shown. ■

Lemma 2.2.8 *Assume that $P \equiv Q$; then $(\rho, \kappa) \models_{me} P$ iff $(\rho, \kappa) \models_{me} Q$.*

Proof. The proof is by induction on the construction of $P \equiv Q$ and here we only consider the two harder cases.

The case of α-equivalence. Consider the subcase $(\nu x^{\chi})P \equiv (\nu y^{\chi})(P\{y/x\})$ where $y \notin fn((\nu x^{\chi})P)$. We calculate that

$$(\rho, \kappa) \models_{me} (\nu x^{\chi})P$$

is equivalent to

$$(\rho, \kappa) \models_{me[x \mapsto \chi]} P$$

which by Corollary 2.2.7 is equivalent to

$$(\rho, \kappa) \models_{me[y \mapsto \chi]} P\{y/x\}$$

(since either $y \notin fn(P)$ or $y = x$) which is equivalent to

$$(\rho, \kappa) \models_{me} (\nu y^{\chi})(P\{y/x\})$$

as was to be shown. The other subcase is similar.

The cases $(\nu x^{\chi})(P_1|P_2) \equiv (\nu x^{\chi})P_1|P_2$ (if $x \notin fn(P_2)$) and $(\nu x^{\chi})P \equiv P$ (if $x \notin fn(P)$) are easy consequences of Lemma 2.2.5.
The case $(\nu x^{\chi})(\nu y^{\chi'})P \equiv (\nu y^{\chi'})(\nu x^{\chi})P$ (if $x \neq y$) is straightforward. ■

Lemma 2.2.9 *Assume that $(\rho, \kappa) \models_{me} P$ and $me(w) \in \rho(me(z))$; then $(\rho, \kappa) \models_{me} P\{w/z\}$.*

Proof. The proof is by structural induction on P. Most cases are straightforward using the following fact:

$$\forall x : \rho(me(x\{w/z\})) \subseteq \rho(me(x))$$

Here we only consider the two harder cases.

The case $P = u(v^\beta).Q$. By Lemma 2.2.8 (since α-equivalence is part of the structural congruence) we may without loss of generality assume that v is neither w nor z. Then we may calculate that

$$(\rho, \kappa) \ \models_{me} u(v^\beta).Q$$

amounts to

$$(\rho, \kappa) \ \models_{me[v \mapsto \beta]} Q \qquad \text{and} \qquad \forall \chi \in \rho(me(u)) : \kappa(\chi) \subseteq \rho(\beta)$$

which, by the induction hypothesis and the fact stated above, imply that

$$(\rho, \kappa) \ \models_{me[v \mapsto \beta]} Q\{w/z\} \qquad \text{and} \qquad \forall \chi \in \rho(me(u\{w/z\})) : \kappa(\chi) \subseteq \rho(\beta)$$

which is equivalent to the required

$$(\rho, \kappa) \ \models_{me} (u(v^\beta).Q)\{w/z\}.$$

The case $P = [x = y]Q$. Our assumption $(\rho, \kappa) \ \models_{me} P$ amounts to

$$(\rho(me(x)) \cap \rho(me(y)) \neq \emptyset \Rightarrow (\rho, \kappa) \ \models_{me} Q$$

and our goal is to show

$$\rho(me(x\{w/z\})) \cap \rho(me(y\{w/z\})) \neq \emptyset \Rightarrow (\rho, \kappa) \ \models_{me} Q\{w/z\}$$

as this amounts to $(\rho, \kappa) \ \models_{me} P\{w/z\}$. By the induction hypothesis it suffices to show that

$$\rho(me(x\{w/z\})) \cap \rho(me(y\{w/z\})) \neq \emptyset \Rightarrow \rho(me(x)) \cap \rho(me(y)) \neq \emptyset$$

that is immediate using the fact stated at the beginning of the proof. ■

Subject reduction. To establish the semantic correctness of our analysis we rely on the definition of the early semantics in Tab. 2.1 as well as on the analysis in Tab. 2.2. The subject reduction result below applies to *all* the solutions of the analysis, and hence in particular to the least. The operational semantics only rewrites processes at "top level" where it is natural to demand that all free names are bound to channels (rather than to binders); this is formalised by the condition $me[fn(-)] \subseteq \mathcal{C}$. In the statement below, we write $(\rho, \kappa) \ \models^{\mathcal{C}}_{me} P$ as a shorthand for $(\rho, \kappa) \ \models_{me} P \wedge me[fn(P)] \subseteq \mathcal{C}$.

Theorem 2.2.10 *If* $(\rho, \kappa) \ \models^{\mathcal{C}}_{me} P$ *and* $P \xrightarrow{\mu} Q$ *we have:*

(1) *if* $\mu = \tau$ *then* $(\rho, \kappa) \models^{\mathcal{C}}_{me} Q$;

(2a) *if* $\mu = \overline{x}y$ *then* $(\rho, \kappa) \models^{\mathcal{C}}_{me} Q$, *and* $me(y) \in \kappa(me(x))$;

(2b) *if* $\mu = \overline{x}(y^{\chi})$ *then* $(\rho, \kappa) \models^{\mathcal{C}}_{me[y \mapsto \chi]} Q$, *and* $\chi \in \kappa(me(x))$;

(3a) *if* $\mu = xy$ *and* $me(y) \in \kappa(me(x))$ *then* $(\rho, \kappa) \models^{\mathcal{C}}_{me} Q$;

(3b) *if* $\mu = x(y^{\chi})$ *and* $\chi \in \kappa(me(x))$ *then* $(\rho, \kappa) \models^{\mathcal{C}}_{me[y \mapsto \chi]} Q$.

Proof. The proof is by induction on the construction of $P \xrightarrow{\mu} Q$ and with subcases depending on whether case *(1)*, *(2a)*, *(2b)*, *(3a)* or *(3b)* applies. Throughout assume that $me[fn(P)] \subseteq \mathcal{C}$, $(\rho, \kappa) \models_{me} P$ (i.e. $(\rho, \kappa) \models^{\mathcal{C}}_{me} P$) and $P \xrightarrow{\mu} Q$.

The case *(1)*. That $me[fn(Q)] \subseteq \mathcal{C}$ is immediate from Fact 2.1.2. It remains to show that $(\rho, \kappa) \models_{me} Q$. In the case of *Tau* this is immediate; and clearly the axioms *Out*, *FreeIn* and *BoundIn* do not apply. Thanks to Lemma 2.2.8 and the induction hypothesis *(1)*, the property is preserved by the rules *Var*, *Par*, *Sum*, *Res* and *Match*; and clearly the rule *Open* does not apply.

For suitable Q_1 and Q_2 such that $Q = Q_1|Q_2$, in the case of rule *Close* the induction hypothesis *(2b)* ensures that

$$(\rho, \kappa) \models_{me[y \mapsto \chi]} Q_1 \quad \wedge \quad \chi \in \kappa(me(x))$$

and the induction hypothesis *(3b)* ensures that

$$(\rho, \kappa) \models_{me[y \mapsto \chi]} Q_2$$

thereby establishing the desired

$$(\rho, \kappa) \models_{me} (\nu y^{\chi})(Q_1|Q_2).$$

The case of rule *Com* is similar but uses the induction hypotheses *(2a)* and *(3a)*.

The case *(2a)*. That $me[fn(Q)] \subseteq \mathcal{C}$ is immediate from Fact 2.1.2; therefore $\rho(me(x)) = \{me(x)\} \subseteq \mathcal{C}$ and $\rho(me(y)) = \{me(y)\} \subseteq \mathcal{C}$. It remains to show that $(\rho, \kappa) \models_{me} Q$ and that $me(y) \in \kappa(me(x))$. In the case of *Out* it follows from Table 2.2 that $(\rho, \kappa) \models_{me} Q$ and $\forall \chi \in \rho(me(x)) : \rho(me(y)) \subseteq \kappa(\chi)$ which amounts to $me(y) \in \kappa(me(x))$; the axioms *Tau*, *FreeIn* and *BoundIn* do not apply. Thanks to Lemma 2.2.8 and the induction hypothesis *(2a)* the property is preserved by the rules *Var*, *Par*, *Sum*, *Res* and *Match*; clearly the rules *Open*, *Close* and *Com* do not apply.

The case *(2b)*. That $(me[y \mapsto \chi])[fn(Q)] \subseteq \mathcal{C}$ and $\lambda = \chi$ is immediate from Fact 2.1.2; therefrom also $\rho(me(x)) = \{me(x)\} \subseteq \mathcal{C}$. It remains to show that

$(\rho, \kappa) \models_{me[y \mapsto \chi]} Q$ and that $\chi \in \kappa(me(x))$. None of the axioms *Tau*, *Out*, *FreeIn* and *BoundIn* nor any of the rules *Close* or *Com* can apply. Thanks to Lemma 2.2.8 and the induction hypothesis *(2b)* the property is preserved by the rules *Var*, *Par*, *Sum*, *Res* and *Match*. In the case of rule *Open* the induction hypothesis *(2a)* and $y \neq x$ ensure that $(\rho, \kappa) \models_{me[y \mapsto \chi]} Q$ and $(me[y \mapsto \chi])(y) \in \kappa((me[y \mapsto \chi])(x))$ which establish the desired result.

The case *(3a)*. Here we also assume that $me(y) \in \kappa(me(x)) \subseteq \mathcal{C}$ so that $me(y) \in \mathcal{C}$. Then $me[fn(Q)] \subseteq \mathcal{C}$ is immediate from Fact 2.1.2 that also implies $\rho(me(x)) = \{me(x)\} \subseteq \mathcal{C}$; it remains to show that $(\rho, \kappa) \models_{me} Q$.

In the case of *FreeIn*, P is on the form $x(w^\beta).R$ and Q is $R\{y/w\}$. By Lemma 2.2.8 (since α-equivalence is part of the structural congruence) we may use rule *Var* to guarantee that $w \neq y$. From $(\rho, \kappa) \models_{me} x(w^\beta).R$ we have that $(\rho, \kappa) \models_{me[w \mapsto \beta]} R$ and $\kappa(me(x)) \subseteq \rho(\beta)$. It follows that $me(y) \in \rho(\beta)$, which may be rewritten as $(me[w \mapsto \beta])(y) \in \rho((me[w \mapsto \beta])(w))$. From Lemma 2.2.9 we then get $(\rho, \kappa) \models_{me[w \mapsto \beta]} R\{y/w\}$ and the desired $(\rho, \kappa) \models_{me} R\{y/w\}$ follows, by Lemma 2.2.5.

Neither the axioms *Tau*, *Out* and *BoundIn*, nor the rules *Open*, *Close* or *Com* are applicable. Thanks to Lemma 2.2.8 and induction hypothesis *(3a)* the property is preserved by the rules *Var*, *Par*, *Sum*, *Res* and *Match*.

The case *(3b)*. Here we also assume that $\chi \in \kappa(me(x)) \subseteq \mathcal{C}$. Then $(me[y \mapsto \chi])[fn(Q)] \subseteq \mathcal{C}$ is immediate from Fact 2.1.2 that implies $\rho(me(x)) = \{me(x)\} \subseteq \mathcal{C}$ and it remains to show that $(\rho, \kappa) \models_{me[y \mapsto \chi]} Q$.

The axiom *BoundIn* applies in this case so P has the form $x(y^\beta).Q$. Let now $w \notin \{y\} \cup fn(Q)$ and note that $x(w^\beta).R \equiv x(y^\beta).Q$ for $R = Q\{w/y\}$. Since $Q \equiv R\{y/w\}$ it follows from Lemma 2.2.8 that it suffices to show that $(\rho, \kappa) \models_{me[y \mapsto \chi]} R\{y/w\}$. From $(\rho, \kappa) \models_{me} P$ and Lemma 2.2.8 it follows that $(\rho, \kappa) \models_{me} x(w^\beta).R$ and hence that $(\rho, \kappa) \models_{me[w \mapsto \beta]} R$ and $\kappa(me(x)) \subseteq \rho(\beta)$. Using Lemma 2.2.5 it now follows that $(\rho, \kappa) \models_{me[w \mapsto \beta][y \mapsto \chi]} R$. As in the previous case we have that $(me[w \mapsto \beta][y \mapsto \chi])(y) \in \rho((me[w \mapsto \beta][y \mapsto \chi])(w))$ and Lemma 2.2.9 gives $(\rho, \kappa) \models_{me[w \mapsto \beta][y \mapsto \chi]} R\{y/w\}$. The desired $(\rho, \kappa) \models_{me[y \mapsto \chi]} R\{y/w\}$ follows by Lemma 2.2.5.

Neither the axioms *Tau*, *Out* and *FreeIn*, nor the rules *Open*, *Close* or *Com* are applicable. Thanks to Lemma 2.2.8 and induction hypothesis *(3b)* the property is preserved by the rules *Var*, *Par*, *Sum*, *Res* and *Match*. ∎

Most of the inclusions occurring in the above items mildly constrain the environment where the process under validation operates. If one only considers closed systems, which can only perform τ moves, the inclusions become useless, as all names are bound; in essence we retain only item *(1)*.

$$
\begin{array}{lcl}
\mathcal{G}_C[\![\mathbf{0}]\!]_{me} & = & \emptyset \\
\mathcal{G}_C[\![\tau.P]\!]_{me} & = & \mathcal{G}_C[\![P]\!]_{me} \\
\mathcal{G}_C[\![\overline{x}y.P]\!]_{me} & = & \mathcal{G}_C[\![P]\!]_{me} \cup \\
 & & \{(C \cup \{\{\chi\} \subseteq r(me(x))\}) \Rightarrow r(me(y)) \subseteq k(\chi) \mid \chi \in \mathcal{U}_\star \cap \mathcal{C}\} \\
\mathcal{G}_C[\![x(y^\beta).P]\!]_{me} & = & \mathcal{G}_C[\![P]\!]_{me[y \mapsto \beta]} \cup \\
 & & \{(C \cup \{\{\chi\} \subseteq r(me(x))\}) \Rightarrow k(\chi) \subseteq r(\beta) \mid \chi \in \mathcal{U}_\star \cap \mathcal{C}\} \\
\mathcal{G}_C[\![P_1 + P_2]\!]_{me} & = & \mathcal{G}_C[\![P_1]\!]_{me} \cup \mathcal{G}_C[\![P_2]\!]_{me} \\
\mathcal{G}_C[\![P_1 | P_2]\!]_{me} & = & \mathcal{G}_C[\![P_1]\!]_{me} \cup \mathcal{G}_C[\![P_2]\!]_{me} \\
\mathcal{G}_C[\![(\nu x^\chi)P]\!]_{me} & = & \mathcal{G}_C[\![P]\!]_{me[x \mapsto \chi]} \\
\mathcal{G}_C[\![[x = y]P]\!]_{me} & = & \mathcal{G}_{C \cup \{r(me(x)) \cap r(me(y)) \neq \emptyset\}}[\![P]\!]_{me} \\
\mathcal{G}_C[\![!P]\!]_{me} & = & \mathcal{G}_C[\![P]\!]_{me}
\end{array}
$$

Table 2.3: Constraint generation for the π-calculus.

2.3 Construction of Solutions

There is also a constructive procedure for obtaining the least solution which operates in $O(N^5)$ time in the size of processes (see [61, 97]). To describe it we shall concentrate on a process, $P_\star$, of interest as well as a marker environment, $me_\star$, giving the binders and channels of free names in the process. To obtain a finite algorithm we shall prefer to restrict our attention to a finite universe, $\mathcal{U}_\star$, containing all the relevant binders and channels and the set $me[fn(P_\star)] =_{\text{def}} \{me(x) \mid x \in fn(P_\star)\}$. We shall say that the size, N, of $P_\star$ is the maximum of the number of symbols in $P_\star$ and the number of elements in $\mathcal{U}_\star$. In case we take an annotation which has all markers different from each other and disjoint from the image of the marker environment, N linearly depends on the size of $P_\star$, only. Validating a solution $(\rho, \kappa) \models_{me} P$ amounts to checking a number of individual constraints. We now define a function $\mathcal{G}_C[\![P]\!]_{me}$ for explicitly extracting the set of constraints to be checked. In doing so, we shall make it clear that we are extracting the constraints in the form of syntax. Hence replace ρ of Tab. 2.2 by r and κ by k.

Definition 2.3.1 *We now define two classes of constraints.*

- *An* individual constraint *is on one of the forms $n_1 \subseteq n_2$, $\{\chi\} \subseteq n_0$, or $n_1 \cap n_2 \neq \emptyset$, where each n_i will be on one of the three forms $r(\beta)$, $r(\chi)$ or $k(\chi)$.*

- *A* composite constraint *is on the form* $\{ic_1, \cdots, ic_m\} \Rightarrow ic$ *where* $m \geq 0$*, all* ic_i *(on the left-hand side) are individual constraints on either form* $\{\chi\} \subseteq n_0$ *or* $n_1 \cap n_2 \neq \emptyset$*, and* ic *(on the right-hand side) is an individual constraint on the form* $n_1 \subseteq r(\beta)$ *or* $n_1 \subseteq k(\chi)$*.*

The details of $\mathcal{G}_C[\![P]\!]_{me}$ are given in Tab. 2.3. In the clause for $[x = y]P$ we cannot decide whether or not $r(me(x)) \cap r(me(y)) \neq \emptyset$ and hence we ensure that the constraints subsequently generated will in fact explicitly test this. Technically, this is achieved by means of the subscript C to the function $\mathcal{G}_C[\![P]\!]_{me}$: each constraint generated will be conditional on all of the constraints in the (initially empty) set C. Similarly, in the clauses for $\overline{x}y.P$ and $x(y^\beta).P$ we use the assumptions about the universe $\mathcal{U}_\star$, in particular that $me[fn(P_\star)] \subseteq \mathcal{U}_\star$, to generate a sufficiently large set of constraints that then explicitly test for $\{\chi\} \subseteq r(me(x))$.

The call $\mathcal{G}_\emptyset[\![P_\star]\!]_{me_\star}$ will give rise to at most $O(N)$ recursive calls, each call directly responsible for generating at most $O(N)$ constraints (see the clauses for input and output). The set of constraints C occurring as index to the recursive calls $\mathcal{G}_C[\![P]\!]_{me}$ can have size at most $O(N)$ (see the clause for matching). Hence at most $O(N^2)$ constraints of size $O(N)$ will be produced.

The semantics of the constraints. To relate Tables 2.2 and 2.3 we need to interpret a set of constraints with respect to a proposed solution (ρ, κ). First we define an evaluation function for left-hand sides:

Definition 2.3.2 *We define* $\mathcal{V}_{(\rho,\kappa)}[\![nn]\!]$ *as follows:*

$$\mathcal{V}_{(\rho,\kappa)}[\![nn]\!] = \begin{cases} \rho(\beta) & \text{if } nn = r(\beta) \\ \{\chi\} & \text{if } nn = r(\chi) \\ \kappa(\chi) & \text{if } nn = k(\chi) \end{cases}$$

Next we define a satisfaction relation.

Definition 2.3.3 *We define*

$$\begin{array}{lll} (\rho,\kappa) \ \mathsf{sat}\ nn \subseteq n & \mathsf{iff} & \mathcal{V}_{(\rho,\kappa)}[\![nn]\!] \subseteq \mathcal{V}_{(\rho,\kappa)}[\![n]\!] \\ (\rho,\kappa) \ \mathsf{sat}\ \{\chi\} \subseteq n & \mathsf{iff} & \{\chi\} \subseteq \mathcal{V}_{(\rho,\kappa)}[\![n]\!] \\ (\rho,\kappa) \ \mathsf{sat}\ nn \neq \emptyset & \mathsf{iff} & \mathcal{V}_{(\rho,\kappa)}[\![nn]\!] \neq \emptyset \end{array}$$

and also

$$(\rho,\kappa) \ \mathsf{sat}\ \{ic_1, \cdots, ic_m\} \Rightarrow ic \quad \mathsf{iff} \quad (\textstyle\bigwedge_{i=1}^m (\rho,\kappa)\,\mathsf{sat}\,ic_i) \Rightarrow ((\rho,\kappa)\,\mathsf{sat}\,ic)$$

and finally

$$(\rho,\kappa) \ \mathsf{SAT}\ C \quad \mathsf{iff} \quad \forall cc \in C : (\rho,\kappa) \ \mathsf{sat}\ cc$$

The relationship between Tables 2.2 and 2.3 is now given by the following result.

Lemma 2.3.4 $((\rho,\kappa)\ \mathsf{SAT}\ \mathcal{G}_C[\![P]\!]_{me})$ *iff* $(((\rho,\kappa)\,\mathsf{SAT}\,C) \Rightarrow ((\rho,\kappa)\ \models_{me} P))$

Proof. We proceed by structural induction on P. Most cases are straightforward and here we only consider two of the more interesting ones.

The case $x(y^\beta).P$. We have that (ρ,κ) SAT $\mathcal{G}_C[\![x(y^\beta).P]\!]_{me}$ is equivalent to

$$\begin{array}{l}(\rho,\kappa)\ \mathsf{SAT}\ \mathcal{G}_C[\![P]\!]_{me[y\mapsto\beta]}\\ (\rho,\kappa)\ \mathsf{SAT}\ \{(C \cup \{\{\chi\} \subseteq r(me(x))\}) \Rightarrow k(\chi) \subseteq r(\beta) \mid \chi \in \mathcal{U}_\star \cap \mathcal{C}\}\end{array}$$

Using the induction hypothesis this is equivalent to

$$\begin{array}{lcl}((\rho,\kappa)\ \mathsf{SAT}\ C) & \Rightarrow & (\rho,\kappa)\ \models_{me[y\mapsto\beta]} P\\ ((\rho,\kappa)\ \mathsf{SAT}\ C) & \Rightarrow & (\forall\chi \in \mathcal{U}_\star \cap \mathcal{C} : \chi \in \rho(me(x)) \Rightarrow \kappa(\chi) \subseteq \rho(\beta))\end{array}$$

and given the assumptions about the universe $\mathcal{U}_\star$ this is equivalent to the desired $((\rho,\kappa)\ \mathsf{SAT}\ C)\ \Rightarrow\ (\rho,\kappa)\ \models_{me} x(y^\beta).P$.

The case $[x = y]P$. We have that (ρ,κ) SAT $\mathcal{G}_C[\![[x = y]P]\!]_{me}$ is equivalent to

$$(\rho(me(x)) \cap \rho(me(y))) \neq \emptyset \Rightarrow\ ((\rho,\kappa)\ \mathsf{SAT}\ \mathcal{G}_{C\cup\{r(me(x))\cap r(me(y))\neq\emptyset\}}[\![P]\!]_{me})$$

Using the induction hypothesis this is equivalent to

$$(\rho(me(x)) \cap \rho(me(y)) \neq \emptyset \wedge (\rho,\kappa)\,\mathsf{SAT}\,C)\ \Rightarrow\ ((\rho,\kappa)\ \models_{me} P)$$

which may be rewritten as

$$(\rho,\kappa)\,\mathsf{SAT}\,C \wedge (\rho(me(x)) \cap \rho(me(y)) \neq \emptyset))\ \Rightarrow\ ((\rho,\kappa)\ \models_{me} P)$$

that is equivalent to the desired $((\rho,\kappa)\ \mathsf{SAT}\ C)\ \Rightarrow\ (\rho,\kappa)\ \models_{me} [x = y]P$. ■

Solving the constraints. It now remains to demonstrate how to solve a set of constraints as might have been generated above. For our purposes, merely obtaining an algorithm running in $O(N^5)$ time, it suffices to perform a simple iterative procedure upon a function f_C associated with the set of constraints C.

Definition 2.3.5 *Given a set C of constraints, the function f_C maps a proposed solution into another one as follows:*

$$f_C(\rho,\kappa) = (\rho',\kappa')$$

where

$$\rho'(\beta) = \bigcup_{(\{ic_1,\cdots,ic_m\}\Rightarrow nn\subseteq r(\beta))\in C} \left\{\begin{array}{ll}\mathcal{V}_{(\rho,\kappa)}[\![nn]\!] & \textit{if } \bigwedge_{i=1}^m (\rho,\kappa)\ \mathsf{sat}\ ic_i\\ \emptyset & \textit{otherwise}\end{array}\right.$$

and

$$\kappa'(\chi) = \bigcup_{(\{ic_1,\cdots,ic_m\}\Rightarrow nn\subseteq k(\chi))\in C} \begin{cases} \mathcal{V}_{(\rho,\kappa)}[\![nn]\!] & \textit{if } \bigwedge_{i=1}^{m}(\rho,\kappa) \text{ sat } ic_i \\ \emptyset & \textit{otherwise} \end{cases}$$

A related function $\hat{f}_C$ is given by

$$\hat{f}_C(\rho,\kappa) = (\rho,\kappa) \sqcup f_C(\rho,\kappa)$$

The relationship between the two functions, f_C and $\hat{f}_C$, and satisfaction, $\mathsf{SAT}\ C$, of the set of constraints is given by the following result.

Lemma 2.3.6 *(ρ,κ) SAT C iff $f_C(\rho,\kappa) \sqsubseteq (\rho,\kappa)$ iff $\hat{f}_C(\rho,\kappa) = (\rho,\kappa)$ whenever C is a set of composite constraints.*

Proof. The second "iff" is immediate so consider the first. A composite constraint in C has the form $\{ic_1,\cdots,ic_m\} \Rightarrow nn \subseteq n$; there are two possibilities for n and here we just consider the case where n is $r(\beta)$. Then

$$(\rho,\kappa) \text{ sat } (\{ic_1,\cdots,ic_m\} \ \Rightarrow\ nn \subseteq r(\beta))$$

is equivalent to

$$(\bigwedge_{i=1}^{m}(\rho,\kappa) \text{ sat } ic_i) \ \Rightarrow\ \mathcal{V}_{(\rho,\kappa)}[\![nn]\!] \subseteq \rho(\beta)$$

which can be rewritten as

$$\left(\begin{cases} \mathcal{V}_{(\rho,\kappa)}[\![nn]\!] & \text{if } \bigwedge_{i=1}^{m}(\rho,\kappa) \text{ sat } ic_i \\ \emptyset & \text{otherwise} \end{cases}\right) \subseteq \rho(\beta)$$

It follows that

$$(\rho,\kappa) \ \mathsf{SAT}\ \{(\{ic_1,\cdots,ic_m\} \Rightarrow nn \subseteq r(\beta')) \in C \mid \beta' = \beta\}$$

is equivalent to

$$\bigcup_{(\{ic_1,\cdots,ic_m\}\Rightarrow nn\subseteq r(\beta'))\in C\wedge\beta'=\beta} \left(\begin{cases} \mathcal{V}_{(\rho,\kappa)}[\![nn]\!] & \text{if } \bigwedge_{i=1}^{m}(\rho,\kappa) \text{ sat } ic_i \\ \emptyset & \text{otherwise} \end{cases}\right) \subseteq \rho(\beta)$$

and the desired result easily follows. ∎

To prepare for the iteration we need the following standard result.

Lemma 2.3.7 *f_C is monotone and $\hat{f}_C$ is monotone and extensive whenever C is a set of composite constraints.*

Proof. It is immediate that $\hat{f}_C$ is extensive (meaning that $\forall(\rho,\kappa) : (\rho,\kappa) \sqsubseteq \hat{f}_C(\rho,\kappa)$) and that it is monotone (meaning that $\forall(\rho,\kappa) \sqsubseteq (\rho',\kappa') : \hat{f}_C(\rho,\kappa) \sqsubseteq \hat{f}_C(\rho',\kappa')$) if f_C is. To see that f_C is monotone let $(\rho,\kappa) \sqsubseteq (\rho',\kappa')$ and consider a composite constraint $\{ic_1,\cdots,ic_m\} \Rightarrow nn \subseteq n$ in C; there are two possibilities for n and here we just consider the case where n is $r(\beta)$. It is immediate that $\mathcal{V}_{(\rho,\kappa)}[\![nn]\!] \subseteq \mathcal{V}_{(\rho',\kappa')}[\![nn]\!]$ and we also have $(\rho,\kappa)\,\mathsf{sat}\,ic_i \;\Rightarrow\; (\rho',\kappa')\,\mathsf{sat}\,ic_i$ because ic_i is either of the form $nn' \neq \emptyset$ or $\{\chi\} \subseteq n'$ (and *not* of the form $nn' \subseteq n'$). This establishes the desired result. ■

Setting $C = \mathcal{G}_\emptyset[\![P_\star]\!]_{me_\star}$ it now follows that the least (ρ,κ) (above a given $(\overline{\rho},\overline{\kappa})$, typically $(\perp,\perp)$) that satisfies $(\rho,\kappa) \models_{me_\star} P_\star$ equals the least fixed point of $\hat{f}_C$ (above $(\overline{\rho},\overline{\kappa})$). Because of our assumptions about the universe, $\mathcal{U}_\star$, the set C is finite and $\hat{f}_C$ operates over a complete lattice of finite size. Standard fixed point theory then ensures that the desired (ρ,κ) is obtained as $\hat{f}_C^j(\overline{\rho},\overline{\kappa})$ upon stabilisation, i.e. for the least j such that $\hat{f}_C^j(\overline{\rho},\overline{\kappa}) = \hat{f}_C^{j+1}(\overline{\rho},\overline{\kappa})$ where it is certain that such j exists.We may summarise the development as follows.

Theorem 2.3.8 *Writing $C = \mathcal{G}_\emptyset[\![P_\star]\!]_{me_\star}$ there is a least natural number j such that $\hat{f}_C^j(\overline{\rho},\overline{\kappa}) = \hat{f}_C^{j+1}(\overline{\rho},\overline{\kappa})$; it satisfies that*

$$\hat{f}_C^j(\overline{\rho},\overline{\kappa}) \;\models_{me_\star} P_\star$$

and that $\hat{f}_C^j(\overline{\rho},\overline{\kappa}) = \sqcap\{(\rho,\kappa) \mid (\rho,\kappa) \;\models_{me_\star} P_\star \wedge (\rho,\kappa) \sqsupseteq (\overline{\rho},\overline{\kappa})\}$.

The tuple (ρ,κ) contains $O(N)$ elements taking values in a set of size $O(N)$. Hence at most $O(N^2)$ iterations of the function will be needed. Each iteration can be performed by considering each of the $O(N^2)$ constraints of form $\{ic_1,\cdots,ic_m\} \Rightarrow nn \subseteq n$, calculating $ic_1,\cdots,ic_m$ and nn in time $O(N)$ and then updating the entry for n accordingly. Hence the desired solution can be obtained in time $O(N^5)$ and space $O(N^3)$.

Example 2.3.9 As an example suppose that (ρ_1,κ_1) is the least solution for P_1 and that (ρ_2,κ_2) is the least solution for P_2 (w.r.t. the same me). It follows that there also is a least solution (ρ,κ) for $P_1|P_2$ and since this is a solution for P_1 and for P_2 we have

$$(\rho_1,\kappa_1) \sqcup (\rho_2,\kappa_2) \sqsubseteq (\rho,\kappa)$$

We therefore know that (ρ,κ) equals

$$\hat{f}_C^j((\rho_1,\kappa_1) \sqcup (\rho_2,\kappa_2))$$

upon stabilisation.

To see that j cannot always be taken to be 0 consider the two processes $P_1 = x(y^{\beta_0}).(\nu x^{\chi_1})\overline{x}x$ and $P_2 = (\nu x^{\chi_1})(\nu z^{\chi_2})\overline{x}z$, the marker environment $me(x) = \chi_1$, and the following definitions of ρ_i and κ_i

	$\kappa_i(\chi_1)$	$\rho_i(\beta_0)$	$\cdots$
$i=1$	$\{\chi_1\}$	$\{\chi_1\}$	$\cdots$
$i=2$	$\{\chi_2\}$	$\emptyset$	$\cdots$

Setting $(\rho', \kappa') = (\rho_1, \kappa_1) \sqcup (\rho_2, \kappa_2)$ we have $\kappa'(\chi_1) = \{\chi_1, \chi_2\}$ and $\rho'(\beta_0) = \{\chi_1\}$ and hence the constraint $\kappa'(\chi_1) \subseteq \rho'(\beta_0)$ fails. ■

2.4 Preventing Leakage

A common approach to system security is based on putting objects and subjects into security classes and preventing information from flowing from higher levels to lower ones. We demonstrate that Control Flow Analysis helps in statically obtaining information relevant from a security point of view. We consider in this section a static property ensuring that a channel, devised to be secret to a process P, is never communicated to an external user. In the next section we statically check another property that concerns how information flows between processes at different levels.

The π-calculus does not distinguish between data, that may be secret, and channels that can be seen as capabilities of accessing data. More refined calculi, notably the spi-calculus [8, 7] that we consider in Chapter 3, make such a distinction and permit a finer description of security properties. We study first a pure calculus of computations, without encryption and decryption primitives, in order to concentrate on the applicability of the Control Flow Analysis to security issues. In fact, the scoping rules of the π-calculus are sufficient for a careful use of channels, because processes can generate and pass new names, making the channels they denote available for communication. In a sense, learning the name of a channel amounts to possessing the capability to communicate on it, and restriction governs the visibility of names.

A process P could have the security requirement of keeping secret (some of) its channels, i.e. not to communicate them to the external environment. A process matches this requirement if it never performs an extrusion of a secret channel, as made precise below. In the following, we assume that $\mathcal{S}$ is a set of *secret* channels given by a designated authority, e.g. the designer or the user of a process P, who implicitly introduces also the set $\mathcal{P}$ of *public* channels as the complement of $\mathcal{S}$. We have only two classification levels. An immediate extension consists of defining a hierarchy of classification levels associated with channels. A priori, binders are neither public nor secret; the actual solution of the analysis establishes which kind of channels can be bound to each binder.

Definition 2.4.1 *The pair P, me is* admissible *for the set $\mathcal{S} \subseteq \mathcal{C}$ of* secret *channels, if and only if $me[fn(P)] \subseteq \mathcal{C}$ and $me[fn(P)] \cap \mathcal{S} = \emptyset$. Then, the set of* public *channels is $\mathcal{P} = \mathcal{C} \setminus \mathcal{S}$.*

Note that the condition of admissibility is equivalent to $me[fn(P)] \subseteq \mathcal{P}$, i.e. all free names are public channels.

A dynamic notion. Now, we would like to characterize a process P that never discloses its secrets. We describe this property by saying that P *has no leaks.*

Intuitively, P enjoys this property if neither it nor any of its derivatives can perform an extrusion of a name bound to a secret channel. For this to make sense it is important to assume that the environment (or an external user) cannot guess any secret channel. We formalise this by a constrained notion of computation called *censored*. Essentially, a computation is censored if no name y, with $me(y) \in \mathcal{S}$, can be read from the environment through an input.

Definition 2.4.2 *Given $P, me, \mathcal{S}$ a* censored step $(P, me) \stackrel{\mu}{\Longrightarrow} (Q, me')$ *is defined whenever the following conditions hold:*

1. $P \stackrel{\mu}{\longrightarrow} Q$
2. (a) *if* $\mu = xy$, *then* $me(y) \in \mathcal{P}$
   (b) *if* $\mu = x(y^\chi)$, *then* $\chi \in \mathcal{P}$

where $me' = \begin{cases} me & \text{if } \mu = \tau, \overline{x}y, xy \\ me[y \mapsto \chi] & \text{if } \mu = \overline{x}(y^\chi), x(y^\chi) \end{cases}$

A censored computation $(P, me) \Longrightarrow^* (Q, me') \stackrel{\mu}{\Longrightarrow} (R, me'')$ *is made of censored steps, whose labels are all immaterial, apart from the last one.*

The following proposition shows that admissibility is preserved under censored computations, provided that there is no extrusion of a name marked secret. It also reveals the rôle played by the second condition of a censored step.

Lemma 2.4.3 *Let P, me be admissible for $\mathcal{S}$ and $(P, me) \stackrel{\mu}{\Longrightarrow} (Q, me')$ be such that $\mu = \overline{x}(y^\chi)$ implies $\chi \in \mathcal{P}$; then Q, me' are admissible for $\mathcal{S}$.*

Proof. The proof is by cases on μ. If $\mu = \tau$ or $\mu = \overline{x}y$ then $me' = me$; from the hypotheses and Fact 2.1.2 we have that $me'[fn(Q)] \subseteq \mathcal{P}$. If $\mu = \overline{x}(y^\chi)$ we have that $me' = me[y \mapsto \chi]$; from the hypotheses and Fact 2.1.2 we get that $me'[fn(Q)] \subseteq \mathcal{P}$. If $\mu = xy$, $me(y) \in \mathcal{P}$ (by censored step) and $me' = me$, therefore the hypotheses and Fact 2.1.2 ensure that $me'[fn(Q)] \subseteq \mathcal{P}$. If $\mu = x(y^\chi)$ we have that $\chi \in \mathcal{P}$ (by censored step) and that $me' = me[y \mapsto \chi]$ so that the hypotheses together with Fact 2.1.2 ensure that $me'[fn(Q)] \subseteq \mathcal{P}$. ■

We are ready to define our dynamic notion of security for some P, me that are admissible for the set $\mathcal{S}$ of secret channels.

Definition 2.4.4 *The process P has* no leaks *with respect to $\mathcal{S}, me$ if and only if $(P, me) \Longrightarrow^* (Q, me')$ implies that there is no pair (R, me'') such that $(Q, me') \stackrel{\overline{x}(y^\chi)}{\Longrightarrow} (R, me'')$ with $\chi \in \mathcal{S}$.*

Of course when P is stuck, it has no leaks. Note that if P has no leaks, with respect to $\mathcal{S}, me$, and P, me are admissible for $\mathcal{S}$, then for all Q such that $(P, me) \Longrightarrow^* (Q, me')$, Q, me' are admissible for $\mathcal{S}$, due to Lemma 2.4.3.

The above definition exploits the π-calculus semantics and, in particular, the extrusion mechanism to dynamically track the leaking of names, associated to secret channels. In fact, we chose to associate secret channels only to restricted names. We could have chosen differently, relaxing this hypothesis and allowing free names to have secret channels associated as well. In this case, we would have instrumented the semantics to more precisely record the markers, during the execution of actions, in the form of a marker environment me, to recall the associated channel for each name, as done in Chapter 3.

A static notion. The notion of no leaks above is dynamic. We now introduce a static notion, in order to predict at compile time if a process has no leaks. It is called *confinement* [1] and we prove that it is a sufficient condition for a process to have no leaks.

Definition 2.4.5 *Let P, me be admissible for a given $\mathcal{S}$. A process P is* confined *with respect to $\mathcal{S}, me$ if and only if there exists (ρ, κ) such that*

(a) $(\rho, \kappa) \models_{me} P$ *and*

(b) $\forall \chi \in \mathcal{P} : \kappa(\chi) = \mathcal{P}$.

Hereafter, we will say that P is confined via the confining *solution* (ρ, κ).

Note that if a process P is confined, by admissibility we also have that $me[fn(P)] \subseteq \mathcal{P}$. Intuitively, condition (b) above implies that only public information can be transmitted along a public channel, i.e. $\bigcup_{\chi \in \mathcal{P}} \kappa(\chi) \subseteq \mathcal{P}$; conversely, any channel, be it secret or public, can pass along a secret channel. We now show that confinement is preserved by censored computations.

Lemma 2.4.6 (Subject reduction for confinement)
Let P be confined (w.r.t. $\mathcal{S}, me$) and $(P, me) \stackrel{\mu}{\Longrightarrow} (Q, me')$; then Q is confined (w.r.t. $\mathcal{S}, me'$).

Proof. The proof is by cases on μ and considers the confining solution (ρ, κ) for P. If $\mu = \tau$ or $\mu = \overline{x}y$ we have that $me' = me$; it follows from Theorem 2.2.10 that $(\rho, \kappa) \models_{me'} Q$ and from Lemma 2.4.3 that Q, me' is admissible for $\mathcal{S}$. If $\mu = \overline{x}(y^\chi)$ we have $me' = me[y \mapsto \chi]$ and $x \in fn(P)$ (by Fact 2.1.2); it follows from Theorem 2.2.10 and confinement that $(\rho, \kappa) \models_{me'} Q$ and that $\chi \in \kappa(me(x)) = \mathcal{P}$; then, by Lemma 2.4.3 it follows that Q, me is admissible for $\mathcal{S}$. If $\mu = xy$ or $\mu = x(y^\chi)$ we have $x \in fn(P)$ (because of Fact 2.1.2). In the first subcase $me' = me$

[1] In the literature on security, confinement is also used with different meanings.

and $me(y) \in \mathcal{P} = \kappa(me(x))$ (by censored step and confinement); Theorem 2.2.10 guarantees that $(\rho, \kappa) \models_{me'} Q$ and Lemma 2.4.3 that Q, me' is admissible for $\mathcal{S}$. In the second subcase $me' = me[y \mapsto \chi]$, $\chi \in \mathcal{P} = \kappa(me(x))$ (by censored step and confinement); Theorem 2.2.10 guarantees that $(\rho, \kappa) \models_{me'} Q$ and Lemma 2.4.3 that Q, me' is admissible for $\mathcal{S}$. ■

We are finally ready to show that confinement is sufficient to guarantee that P has no leaks.

Theorem 2.4.7 *If P is confined (with respect to $\mathcal{S}, me$) then P has no leaks (with respect to $\mathcal{S}, me$).*

Proof. By Lemma 2.4.6 it is enough to prove that $(P, me) \stackrel{\overline{x}(y^\chi)}{\Longrightarrow} (Q, me')$ implies $\chi \in \mathcal{P}$. Since $x \in fn(P)$ by Fact 2.1.2, we have that $me(x) \in \mathcal{P}$ and $\kappa(me(x)) = \mathcal{P}$ (by confinement). By Theorem 2.2.10, $\chi \in \kappa(me(x))$ and hence $\chi \in \mathcal{P}$ as was to be shown. ■

Example 2.4.8 We consider here an abstract version of the Wide Mouthed Frog protocol, adapted from [7]. Intuitively, two principals A and B wish to communicate by classical shared-key cryptography, and the first forwards a secret key, say c_{AB}, to the second one, with the help of a server. The server shares the secret keys c_{AS} and c_{SB} with A and B, respectively. So, the principal A passes its key c_{AB}, encrypted with c_{AS}, to the server. The server decrypts the message, encrypts it with c_{SB} and passes it to B. Afterwards A sends the restricted message M, encrypted with c_{AB}, to B.

Here principals will be represented by processes and keys by secret channels, because the π-calculus has no cryptographic primitives. The specification follows:

- $A = (\nu M^{\chi_M})(\nu c_{AB}^{\chi_{AB}})\overline{c_{AS}}c_{AB}.\overline{c_{AB}}M$
- $S = c_{AS}(x^{\beta_x}).\overline{c_{SB}}x$
- $B = c_{SB}(y^{\beta_y}).y(z^{\beta_z})$
- $P = (\nu c_{AS}^{\chi_{AS}})(\nu c_{SB}^{\chi_{SB}})(A|S|B)$

The following solution is the least one:

$$\rho(\beta) = \begin{cases} \{\chi_{AB}\} & \text{if } \beta = \beta_x, \beta_y \\ \{\chi_M\} & \text{if } \beta = \beta_z \end{cases} \qquad \kappa(\chi) = \begin{cases} \{\chi_M\} & \text{if } \chi = \chi_{AB} \\ \{\chi_{AB}\} & \text{if } \chi = \chi_{AS}, \chi_{SB} \\ \emptyset & \text{if } \chi = \chi_M \end{cases}$$

If we take $\mathcal{S} = \{\chi_{AB}, \chi_{AS}, \chi_{SB}, \chi_M\}$, for all choices of $\mathcal{P}$ there is a solution confining P (with respect to $\mathcal{S}$). Thus, secrecy of M is guaranteed.

Suppose to have a new name c_{AC}, with $me(c_{AC}) = \chi_c \in \mathcal{P}$. Let A be extended as follows

$$A = (\nu M^{\chi_M})(\nu c_{AB}^{\chi_{AB}})\overline{c_{AS}}c_{AB}.\overline{c_{AS}}M.\overline{c_{AC}}c_{AB}.$$

Intuitively, A sends c_{AB} along c_{AC} after the completion of the protocol. For all solutions, it turns out that $\kappa(\chi_{c_{AC}}) \supseteq \{\chi_{AB}\} \not\subseteq \mathcal{P}$. Therefore, there is no solution confining P, with respect to $\mathcal{S}$. Indeed, the analysis reveals the extrusion of c_{AB}. With our specification in the π-calculus, this action does not affect the secrecy of M, yet it signals a potential problem, as an extrusion represents the publication of the corresponding secret key. Indeed, if we take the *spi*-calculus where encrypted messages are transmitted as cleartext, an enemy may intercept M encrypted, store it and then decrypt it after c_{AB} has been extruded (this violates the so-called forward-secrecy property, see e.g. [2]). □

■

It is immediate to see that confinement is not a necessary condition for P having no leaks. For instance the process $(\nu x^{\chi})\overline{x}y.\overline{y}x$ has no leaks but it is not confined with respect to $\mathcal{S} = \{\chi\}$ and $me(y) \neq \chi$. Indeed not all deadlocks can be detected statically. So the extrusion of the name x along channel y is considered a possible violation of secrecy.

Checking the static condition. One approach to checking confinement of a process P with respect to $\mathcal{S}, me$ is to use the polynomial time algorithm devised in the previous section. First we check that P, me (or $P_\star, me_\star$) is indeed admissible for the $\mathcal{S}$ given. Next we construct sets of constraints C_a and C_b corresponding closely to the two conditions in Def. 2.4.5. For C_a we may simply use $\mathcal{G}_\emptyset[\![P]\!]_{me}$. For C_b we generate constraints for half of the equality, namely for the inclusion "$\supseteq$". To be specific, write $\mathcal{P}_\star = \mathcal{P} \cap \mathcal{U}_\star$ and let C_b consist of the constraints $\mathcal{P}_\star \subseteq k(\chi)$ for all $\chi \in \mathcal{P}_\star$. (These constraints can be expanded into a form allowed by Def. 2.3.1 unlike what would be the case if we let C_b take care of the entire equality.) We then solve the set of constraints in polynomial time so as to get the least solution. Then P is confined if and only if the least solution has a κ component equal to the one displayed in Def. 2.4.5. This approach to determining whether or not P is confined (w.r.t. to $\mathcal{S}, me$) clearly operates in polynomial time.

René Rydhof Hansen has constructed a web-based system for checking confinement, based on the algorithm reported here. It is available from the web-page `http://www.daimi.au.dk/~fn/FlowLogic.html`

2.5 Multi-level Security

Another way of enforcing security is by defining a hierarchy of levels for processes. The security requirement is that a process classified at a high level cannot write any value to a process at low level, while the converse is allowed; symmetrically a process at low level cannot read data from one of a high level. Sometimes this non-interference property is referred to as *no read-up/no write-down* [56]. These requirements amount to the simple security property that is part of the multi-level access control policy of [14, 51].

To study the no read-up/no write-down property we need an extension to the syntax of the π-calculus to assign security levels to processes. Accordingly, the operational semantics requires a little extension. Moreover, we will adapt the static analysis as well, to take care of these levels.

In the following we will only introduce the necessary extensions to the setting of the previous sections.

Syntax. In order to simplify our presentation, we make here two assumptions, that are common in the literature (see e.g. [128]). First we consider to have only two levels of security clearance: L for low and H for high. So, we introduce the set $\mathcal{L} = \{L, H\}$. The case with a hierarchy of levels is studied in [19]. Secondly, we assume to have systems made of processes in parallel, and only these top-level components are labelled by clearance levels.

The new syntax, defined below, imports the one in Def. 1.2.1 for processes.

Definition 2.5.1 Systems*, denoted by* $S, S', S_1, S_2, \dots \in Sys$ *are built from processes according to the syntax*

$$S ::= \langle P \rangle^l \mid (\nu x^\chi)S \mid S|S \mid !S$$

where $\langle P \rangle^l$ *expresses that* P *has level* $l \in \mathcal{L}$*.*

The definition of free names is trivially extended by letting

$$\begin{array}{ll} fn(\langle P \rangle^l) = fn(P) & fn((\nu x^\chi)S) = fn(S) \setminus \{x\} \\ fn(S_1|S_2) = fn(S_1) \cup fn(S_2) & fn(!S) = fn(S). \end{array}$$

Semantics. The structural congruence now takes care of systems, i.e.:

- $S \equiv S'$ if S and S' are α-equivalent;
- $\langle \mathbf{0} \rangle^H \equiv \langle \mathbf{0} \rangle^L$, and $(Sys/_\equiv, |, \langle \mathbf{0} \rangle^H)$ is a commutative monoid;
- $(\nu x^\chi)(\nu y^{\chi'})S \equiv (\nu y^{\chi'})(\nu x^\chi)S$, if $x \neq y$,
  $(\nu x^\chi)(S_1|S_2) \equiv (\nu x^\chi)S_1|S_2$ if $x \notin fn(S_2)$, and
  $(\nu x^\chi)S \equiv S$ if $x \notin fn(S)$;
- $!S \equiv S|!S$.

The operational semantics is defined in Tab. 2.4, and uses the transition relation defined in Tab. 2.1. The transitions have the form $S \xrightarrow[l]{\mu} S'$, with $l \in \mathcal{L} \cup \{\epsilon\}$.

$$Lev: \frac{P \xrightarrow{\mu} Q}{\langle P\rangle^l \xrightarrow[l]{\mu} \langle Q\rangle^l} \qquad VarS: \frac{S_1' \equiv S_1,\ S_1 \xrightarrow[l]{\mu} S_2,\ S_2 \equiv S_2'}{S_1' \xrightarrow[l]{\mu} S_2'}$$

$$ParS: \frac{S_1 \xrightarrow[l]{\mu} S_1'}{S_1|S_2 \xrightarrow[l]{\mu} S_1'|S_2}, bn(\mu) \cap fn(S_2) = \emptyset \qquad ResS: \frac{S \xrightarrow[l]{\mu} S'}{(\nu x^\chi)S \xrightarrow[l]{\mu} (\nu x^\chi)S'}, x \notin n(\mu)$$

$$CloseS: \frac{S_1 \xrightarrow[l_1]{\overline{x}(y^\chi)} S_1',\ S_2 \xrightarrow[l_2]{x(y^\chi)} S_2'}{S_1|S_2 \xrightarrow[\epsilon]{\tau} (\nu y^\chi)(S_1'|S_2')} \qquad OpenS: \frac{S \xrightarrow[l]{\overline{x}y} S'}{(\nu y^\chi)S \xrightarrow[l]{\overline{x}(y^\chi)} S'}, y \neq x$$

$$ComS: \frac{S_1 \xrightarrow[l_1]{\overline{x}y} S_1',\ S_2 \xrightarrow[l_2]{xy} S_2}{S_1|S_2 \xrightarrow[\epsilon]{\tau} S_1'|S_2'}$$

Table 2.4: Early transition system for the π-calculus with security levels.

When different from ϵ, the additional level label l records the clearance of the system performing the transition. When a communication is derived the levels of the sender S_o and of the receiver S_i are discarded, leading to a transition of the form $S_o|S_i \xrightarrow[\epsilon]{\tau} S'$ (see the rules *ComS* and *CloseS* in Tab. 2.4).

Analysis. The solution of our new analysis is a triple (ρ, κ, σ) and the Flow Logic judgements are on either form

$$(\rho, \kappa, \sigma) \models_{me} S \quad \text{or} \quad (\rho, \kappa, \sigma) \models^l_{me} P.$$

The rôle of ρ, κ, me is exactly as in the previous sections. The purpose of the new entries l and $\sigma = \langle \sigma_{in}, \sigma_{out}\rangle$, the *abstract communication structure*, is the following:

- $l \in \mathcal{L}$ keeps track of the current security level of the process under validation, and is not needed for systems.

- $\sigma_{in}, \sigma_{out} : \mathcal{L} \to (\mathcal{C} \to \wp(\mathcal{C}))$ give the set of the channels that can be bound to the possible objects of an input and an output action respectively, performed by the sub-processes labelled by $l \in \mathcal{L}$, on a given channel $\chi \in \mathcal{C}$.

The analysis in Tab. 2.5 extends the one in Tab. 2.2, to deal with the levels of security clearance, by imposing two further conditions while checking an input and an output prefix. The channels that can be bound to the object of an input (resp. an output) action along channel χ must be included in $\sigma_{in}(l)(\chi)$ (resp. $\sigma_{out}(l)(\chi)$), where l is the current security level. This is determined by, and is the only task of the clause for the analysis of the system $\langle P \rangle^l$. The rules for the other systems are just as the rules for the corresponding processes. The technical results of Section 3 concerning the set of solutions forming a Moore family, as well as the computation of solutions in $\mathcal{O}(N^5)$ time and $\mathcal{O}(N^3)$ space via the generation of constraints are easily adapted to the setting at hand. We therefore dispense with the details.

Example 2.5.2 Consider the following system S. It consists of the processes R and Q of low level, and of P of high level.

- $R = \overline{a}b.\overline{a}b.\overline{b}c$
- $Q = a(x^{\beta_x}).\overline{x}x$
- $P = a(y^{\beta_y}).y(z^{\beta_z}).([y = z]\overline{y}a \ + \ y(w^{\beta_w}))$
- $S = !(\langle R \rangle^L \mid \langle Q \rangle^L \mid \langle P \rangle^H)$

where the marker environment me is such that $me(fv) = \chi_{fv}$ for all the free names $fv \in \{a, b, c\}$. The triple (ρ, κ, σ) is defined as follows, where the bound names are $bv \in \{x, y, z, w\}$:

$$\rho(\beta_{bv}) = \begin{cases} \{\chi_b\} & \text{if } bv = x, y \\ \{\chi_a, \chi_b, \chi_c\} & \text{if } bv = z, w \end{cases} \qquad \kappa(\chi_{fv}) = \begin{cases} \{\chi_b\} & \text{if } fv = a \\ \{\chi_a, \chi_b, \chi_c\} & \text{if } fv = b \\ \emptyset & \text{if } fv = c \end{cases}$$

$$\begin{array}{ll} \sigma_{in}(L)(\chi_a) = \{\chi_b\} & \sigma_{in}(H)(\chi_a) = \{\chi_b\} \\ \sigma_{in}(L)(\chi_b) = \emptyset & \sigma_{in}(H)(\chi_b) = \{\chi_a, \chi_b, \chi_c\} \\ \sigma_{in}(L)(\chi_c) = \emptyset & \sigma_{in}(H)(\chi_c) = \emptyset \end{array}$$

$$\begin{array}{ll} \sigma_{out}(L)(\chi_a) = \{\chi_b\} & \sigma_{out}(H)(\chi_a) = \emptyset \\ \sigma_{out}(L)(\chi_b) = \{\chi_b, \chi_c\} & \sigma_{out}(H)(\chi_b) = \{\chi_a\} \\ \sigma_{out}(L)(\chi_c) = \emptyset & \sigma_{out}(H)(\chi_c) = \emptyset \end{array}$$

Recall that $\rho(\chi) = \{\chi\}$. A simple check shows that $(\rho, \kappa, \sigma) \models_{me} S$. ■

The semantic soundness of our new analysis is stated by the new subject reduction theorem, which is almost identical to Theorem 2.2.10. The only differences are the new judgements $(\rho, \kappa, \sigma) \models_{me} S$ and the obvious conditions about the new components σ and l. Its proof is a straightforward adaptation of the one of Theorem 2.2.10, so we omit it. We also omit the easy extensions of the relevant lemmata in Section 2.2. As before, $(\rho, \kappa, \sigma) \models^{\mathcal{C}}_{me} S$ stands for $(\rho, \kappa, \sigma) \models^{\mathcal{C}}_{me} S \wedge me[fn(S)] \subseteq \mathcal{C}$.

$$
\begin{array}{lll}
(\rho,\kappa,\sigma) \models^{l}_{me} \mathbf{0} & \text{iff} & true \\
(\rho,\kappa,\sigma) \models^{l}_{me} \tau.P & \text{iff} & (\rho,\kappa,\sigma) \models^{l}_{me} P \\
(\rho,\kappa,\sigma) \models^{l}_{me} \overline{x}y.P & \text{iff} & (\rho,\kappa,\sigma) \models^{l}_{me} P \wedge \\
 & & \forall \chi \in \rho(me(x)) : \left(\begin{array}{l} \rho(me(y)) \subseteq \kappa(\chi) \wedge \\ \rho(me(y)) \subseteq \sigma_{out}(l)(\chi)) \end{array} \right) \\
(\rho,\kappa,\sigma) \models^{l}_{me} x(y^{\beta}).P & \text{iff} & (\rho,\kappa,\sigma) \models^{l}_{me[y\mapsto\beta]} P \wedge \\
 & & \forall \chi \in \rho(me(x)) : \left(\begin{array}{l} \kappa(\chi) \subseteq \rho(\beta) \wedge \\ \kappa(\chi) \subseteq \sigma_{in}(l)(\chi) \end{array} \right) \\
(\rho,\kappa,\sigma) \models^{l}_{me} P_1 + P_2 & \text{iff} & (\rho,\kappa,\sigma) \models^{l}_{me} P_1 \wedge (\rho,\kappa,\sigma) \models^{l}_{me} P_2 \\
(\rho,\kappa,\sigma) \models^{l}_{me} P_1 | P_2 & \text{iff} & (\rho,\kappa,\sigma) \models^{l}_{me} P_1 \wedge (\rho,\kappa,\sigma) \models^{l}_{me} P_2 \\
(\rho,\kappa,\sigma) \models^{l}_{me} (\nu x^{\chi})P & \text{iff} & (\rho,\kappa,\sigma) \models^{l}_{me[x\mapsto\chi]} P \\
(\rho,\kappa,\sigma) \models^{l}_{me} [x=y]P & \text{iff} & (\rho(me(x)) \cap \rho(me(y)) \neq \emptyset \\
 & & \Rightarrow (\rho,\kappa,\sigma) \models^{l}_{me} P \\
(\rho,\kappa,\sigma) \models^{l}_{me} !P & \text{iff} & (\rho,\kappa,\sigma) \models^{l}_{me} P \\
\hline
(\rho,\kappa,\sigma) \models_{me} \langle P \rangle^{l} & \text{iff} & (\rho,\kappa,\sigma) \models^{l}_{me} P \\
(\rho,\kappa,\sigma) \models_{me} (\nu x^{\chi})S & \text{iff} & (\rho,\kappa,\sigma) \models_{me[x\mapsto\chi]} S \\
(\rho,\kappa,\sigma) \models_{me} S_1 | S_2 & \text{iff} & (\rho,\kappa,\sigma) \models_{me} S_1 \wedge (\rho,\kappa,\sigma) \models_{me} S_2 \\
(\rho,\kappa,\sigma) \models_{me} !S & \text{iff} & (\rho,\kappa,\sigma) \models_{me} S
\end{array}
$$

Table 2.5: Control flow analysis for the π-calculus with security levels.

Theorem 2.5.3 *If* $(\rho,\kappa,\sigma) \models^{\mathcal{C}}_{me} S$ *and* $S \xrightarrow[l]{\mu} S'$ *we have:*

(1) if $\mu = \tau$ *then* $(\rho,\kappa,\sigma) \models^{\mathcal{C}}_{me} S'$*;*

(2a) if $\mu = \overline{x}y$ *then*
$(\rho,\kappa,\sigma) \models^{\mathcal{C}}_{me} S'$*,* $me(y) \in \kappa(me(x))$*, and* $me(y) \in \sigma_{out}(l)(me(x))$*;*

(2b) if $\mu = \overline{x}(y^\chi)$ *then*
$(\rho,\kappa,\sigma) \models^{\mathcal{C}}_{me[y\mapsto\chi]} S'$*,* $\chi \in \kappa(me(x))$ *and* $\chi \in \sigma_{out}(l)(me(x))$*;*

(3a) if $\mu = xy$ *and* $me(y) \in \kappa(me(x))$ *then*
$(\rho,\kappa,\sigma) \models^{\mathcal{C}}_{me} S'$*, and* $me(y) \in \sigma_{in}(l)(me(x))$*;*

(3b) if $\mu = x(y^\chi)$ *and* $\chi \in \kappa(me(x))$ *then*
$(\rho,\kappa,\sigma) \models^{\mathcal{C}}_{me[y\mapsto\chi]} S'$*, and* $\chi \in \sigma_{in}(l)(me(x))$*,*

with $l \in \{L,H\}$ *in all cases, except for the case (1) where* $l \in \{L,H\} \cup \{\epsilon\}$*.*

A dynamic notion. We now introduce the dynamic version of the no read-up/no write-down property. As we did with censored steps, we impose a restriction on the channels that can be read by a process P with a given clearance l. We assume that the environment is always willing to listen to P, but it can select which information is to be transmitted to P. To formalize the intentions of the environment, we use a function

$$\varsigma : \mathcal{L} \to (\mathcal{C} \to \wp(\mathcal{C}))$$

that maps a label $l \in \mathcal{L}$ and a channel $\chi \in \mathcal{C}$ to the set of channels that the environment considers secure to communicate to $\langle P\rangle^l$.

Definition 2.5.4 *Given* S, me, ς*, a* granted step *is* $(S,me) \overset{\mu}{\underset{l}{\longmapsto}} (S',me')$*, and is defined whenever*

1. $S \xrightarrow[l]{\mu} S'$*, and*
2. *(a) if* $\mu = xy$*, then* $me(y) \in \varsigma(l)(me(x))$
   (b) if $\mu = x(y^\chi)$*, then* $\chi \in \varsigma(l)(me(x))$

where $me' = \begin{cases} me & \text{if } \mu = \tau, \overline{x}y, xy \\ me[y\mapsto\chi] & \text{if } \mu = \overline{x}(y^\chi), x(y^\chi) \end{cases}$

A granted computation $(S,me) \longmapsto^* (S',me') \overset{\mu}{\underset{l}{\longmapsto}} (S'',me'')$ *is made of granted steps.*

The definition of our version of the no read-up/no write-down property follows.

Definition 2.5.5 *A system S is* no read-up/no write-down *(*nru/nwd *for short) with respect to ς, me if and only if:*

whenever $(S, me) \longmapsto^* (S', me') \overset{\tau}{\underset{\epsilon}{\longmapsto}} (S'', me'')$, *where the last granted step is a communication (between S_o and S_i) that has been deduced with either*

(a) the rule ComS, using the premises $S_o \xrightarrow[l_o]{\overline{x}y} S'_o$ *and* $S_i \xrightarrow[l_i]{xy} S'_i$, *or*

(b) the rule CloseS, using the premises $S_o \xrightarrow[l_o]{\overline{x}(y^\chi)} S'_o$ *and* $S_i \xrightarrow[l_i]{x(y^\chi)} S'_i$,

then $l_o = H$ implies $l_i = H$.

A static notion. We define now a static property that guarantees that a process is nru/nwd. Besides finding a solution (ρ, κ, σ) for a process P, we require that the channels that can pass along a given channel χ include those that can be read and sent along χ, recorded by the abstract communication structure σ (condition *2(a)* below). More interesting is item *(b)* of the same condition, where the channels read along χ should include those that the environment is willing to supply, expressed by ς. The last condition is the key one. It requires that the set of channels that can be communicated along the channel χ, cannot be associated with objects of ouput actions of processes with high level H and, at the same time, with objects of input actions of processes with low level L.

Definition 2.5.6 *Let S, me be such that $me[fn(S)] \subseteq \mathcal{C}$. Then S is* discreet *(w.r.t. ς, me) if and only if there exists (ρ, κ, σ) such that*

1. $(\rho, \kappa, \sigma) \models_{me} S$
2. $\forall l \in \{L, H\}, \chi \in \mathcal{C}:$ $\begin{array}{l}(a)\ \kappa(\chi) \supseteq \sigma_{in}(l)(\chi) \cup \sigma_{out}(l)(\chi) \\ (b)\ \sigma_{in}(l)(\chi) \supseteq \varsigma(l)(\chi)\end{array}$
3. $\forall \chi \in \mathcal{C} : \sigma_{out}(H)(\chi) \cap \sigma_{in}(L)(\chi) = \emptyset$.

Below, we show that the property of being discreet is preserved under granted steps.

Lemma 2.5.7 (Subject reduction for discreetness)

If S is discreet with respect to ς, me, and $(S, me) \overset{\mu}{\underset{l}{\longmapsto}} (S', me')$, *then S' is discreet with respect to ς, me'.*

Proof. Theorem 2.5.3 suffices to prove both $me[fn(S')] \subseteq \mathcal{C}$ and $(\rho, \kappa, \sigma) \models_{me} S'$. The proof of the second and third items of discreetness is immediate, because the solution does not change. The only delicate point for the application of Theorem 2.5.3 is when the granted step is an input, while in the other cases the proof is trivial. For input, consider first the case $\mu = xy$. It suffices to show that $me(y) \in \kappa(me(x))$. Conditions *(2a)* and *(2b)* of Def. 2.5.6 guarantee that $\kappa(me(x)) \supseteq \sigma_{in}(l)(me(x)) \supseteq \varsigma(l)(me(x))$. Furthermore, $me(y) \in \varsigma(l)(me(x))$ because the step is granted. The case when $\mu = x(y^\chi)$ is similar. ■

Theorem 2.5.8
If S is discreet (w.r.t. ς, me), then S is nru/nwd (w.r.t. ς, me).
Proof. Since our computations are granted, by Lemma 2.5.7 it is enough to check that, if $(S, me) \overset{\tau}{\underset{\epsilon}{\longmapsto}} (S', me')$, then S' is nru/nwd, with τ being a communication between S_o and S_i, defined as in Def. 2.5.5. Without loss of generality we may assume that all restrictions in S occur outermost or inside some $\langle \cdots \rangle$. Assume, per absurdum, that $l_o = H$ and $l_i = L$. Then there exists $\langle P_o \rangle^H$ and $\langle P_i \rangle^L$ such that either

(a) $\langle P_o \rangle^H \xrightarrow[H]{\overline{x}y} \langle P'_o \rangle^H$ and $\langle P_i \rangle^L \xrightarrow[L]{xy} \langle P'_i \rangle^L$, or

(b) $\langle P_o \rangle^H \xrightarrow[H]{\overline{x}(y^\chi)} \langle P'_o \rangle^H$ and $\langle P_i \rangle^L \xrightarrow[L]{x(y^\chi)} \langle P'_i \rangle^L$,

given the assumptions about S_o and S_i. Given that $(\rho, \kappa, \sigma) \models_{me} S$ we obtain that $(\rho, \kappa, \sigma) \models_{\overline{me}} \langle P_o \rangle^H$ and $(\rho, \kappa, \sigma) \models_{\overline{me}} \langle P_i \rangle^L$, for some suitable $\overline{me}$ taking care of the restrictions mentioned above. The analysis and Theorem 2.5.3 tell us that

(a) $\overline{me}(y) \in \sigma_{out}(H)(\overline{me}(x))$ and $\overline{me}(y) \in \sigma_{in}(L)(\overline{me}(x))$, or

(b) $\chi \in \sigma_{out}(H)(\overline{me}(x))$ and $\chi \in \sigma_{in}(L)(\overline{me}(x))$.

This contradicts item 3 of Def. 2.5.6, that demands the intersection of the two sets to be empty. ■

Example 2.5.9 It is easy to prove that the process S validated in Example 2.5.2 is discreet. In particular the following two conditions hold.

- $\sigma_{out}(H)(\chi_a) \cap \sigma_{in}(L)(\chi_a) = \emptyset \cap \{\chi_b\} = \emptyset$.
- $\sigma_{out}(H)(\chi_b) \cap \sigma_{in}(L)(\chi_b) = \{\chi_a\} \cap \emptyset = \emptyset$.

■

As was the case for confinement, the polynomial time algorithm of Section 2.3 can be used to perform a polynomial time check of whether or not a process P is discreet with respect to given ς, me.

René Rydhof Hansen has constructed a web-based system for checking discreetness, based on the algorithm reported here. It is available from the web-page
`http://www.daimi.au.dk/~fn/FlowLogic.html`

A lattice of security levels A possible extension to the nru/nwd property might consider to have a lattice of clearance levels (see Chapter 5 of [51]).

It is quite immediate, in this case, extending our analysis in Section 2.4 to statically check that a high level information is never transmitted on a channel with a lower level of security classification Let $\mathbf{L}$ be the lattice of security levels. Each process P has level $l \in \mathbf{L}$. In this case Def. 2.5.5 becomes as follows.

Definition 2.5.10 *A system S is* no read-up/no write-down *(*nru/nwd *for short) with respect to ς, me if and only if:*
whenever $(S, me) \longmapsto^* (S', me') \overset{\tau}{\underset{\epsilon}{\longmapsto}} (S'', me'')$, *where the last granted step is a communication (between S_o and S_i) that has been deduced with either*

(a) the rule ComS, using the premises $S_o \xrightarrow[l_o]{\overline{x}y} S'_o$ *and* $S_i \xrightarrow[l_i]{xy} S'_i$, *or*

(b) the rule CloseS, using the premises $S_o \xrightarrow[l_o]{\overline{x}(y^\chi)} S'_o$ *and* $S_i \xrightarrow[l_i]{x(y^\chi)} S'_i$,

then $l_o \supseteq l_i$.

Def. 2.5.6 becomes instead as follows.

Definition 2.5.11 *Let S, me be such that $me[fn(S)] \subseteq \mathcal{C}$. Then S is* discreet *(w.r.t. ς, me) if and only if there exists (ρ, κ, σ) such that*

1. $(\rho, \kappa, \sigma) \models_{me} S$
2. $\forall \chi \in \mathcal{C}$ *and* $\forall l \in \mathbf{L}$: $\begin{array}{l} (a)\ \kappa(\chi) \supseteq \sigma_{in}(l)(\chi) \cup \sigma_{out}(l)(\chi) \\ (b)\ \sigma_{in}(l)(\chi) \supseteq \varsigma(l)(\chi) \end{array}$
3. $\forall l \subseteq l' \in \mathbf{L}$ *and* $\forall \chi \in \mathcal{C} : \sigma_{out}(l')(\chi) \cap \sigma_{in}(l)(\chi) = \emptyset$.

2.6 Some Possible Extensions

A more general notion of the no read-up/no write-down property assigns levels of confidentiality also to the exchanged data (i.e. the objects of input and output actions). Processes with low level clearance are then not allowed to access (i.e. they can neither send nor receive) highly classified data. In this case, our previous analysis should be extended. Note in passing that (an extension of) our first security property is not enough for getting the full version of the nru/nwd property; indeed, processes with different clearances are allowed to send objects with different classification along the same channel, while this is forbidden in the case considered in Section 2.4.

A further possible extension of our analysis concerns the static control of the *directionality* of channels [103], stating if a channel is used only for input or only for output or in the remaining combinations. Besides an obvious use for code optimization, directionality can be applied to check access rights of processes on the usage of channels.

The corresponding analysis is a variant of that in Section 2.4, , in which the rules are exactly the same, but where the interpretation of the label l changes: each different process is annotated with a distinguished label that acts as a static identifier, rather than as a security clearance level.

Definition 2.6.1

1. *A process* $\langle P \rangle^l$ *uses the channel* χ for input only *if*
   - $(\rho, \kappa, \sigma) \models_{me} \langle P \rangle^l$
   - $\sigma_{in}(l)(\chi) \neq \emptyset$ *and*
   - $\sigma_{out}(l)(\chi) = \emptyset$.
2. *A process* P *uses the channel* χ for output only *if*
   - $(\rho, \kappa, \sigma) \models_{me} \langle P \rangle^l$
   - $\sigma_{in}(l)(\chi) = \emptyset$ *and*
   - $\sigma_{out}(l)(\chi) \neq \emptyset$.
3. *A process* P *uses the channel* χ for input and output *if*
   - $(\rho, \kappa, \sigma) \models_{me} \langle P \rangle^l$
   - $\sigma_{in}(l)(\chi) \neq \emptyset$ *and*
   - $\sigma_{out}(l)(\chi) \neq \emptyset$.
4. *A process* P *uses the channel* χ neither for input nor for output *if*
   - $(\rho, \kappa, \sigma) \models_{me} \langle P \rangle^l$
   - $\sigma_{in}(l)(\chi) = \emptyset$ *and*
   - $\sigma_{out}(l)(\chi) = \emptyset$.

It is not difficult to prove, for instance, that if $me(a) = \chi$ and $\langle P \rangle^l$ uses χ only for input, then neither P nor any of its derivatives could output on a.

Also, we would like to statically check that processes migrating from site to site correctly use their capabilities and never harm the sites where they are hosted [60, 30, 40, 111].

Finally, it is interesting to formulate even stronger Control Flow Analyses based on what has been done for functional languages and to compare the precision obtained for the π-calculus with that for the λ-calculus. This might take the form of translating the λ-calculus into the π-calculus and comparing the analysis of λ-terms to the translated terms.

Chapter 3

Control Flow Analysis for the Spi-Calculus

In this chapter, we present a Control Flow Analysis for the spi-calculus, that extends the Control Flow for the π-calculus, seen in the previous chapter.

The data that can be exchanged in communications have a richer structure than in the π-calculus. Messages can be, for instance, pairs or data encrypted under some key. Therefore, the analysis should verify the way terms are manipulated. This implies another extension to the syntax, i.e. assigning explicit "labels" to the occurrences of terms, so as to give names to the program points of interest. Annotations do not affect the semantics and can be done mechanically, e.g. assigning different markers and labels for different objects.

Apart from this difference, the development of the analysis follows the same pattern seen in Chapter 2. Control Flow Analysis is formulated as a specification of the correctness of a candidate solution. Best solutions always exist and the semantic correctness of solutions is established in the form of a subject-reduction result. At present, we have not yet derived a procedure that generates solutions.

We apply our analysis for statically checking security properties. We first extend the notion of *no leaking* and of *confinement* seen for the π-calculus.

The first notion says that a process is *careful* if it does not perform any outputs of secret terms on names associated with public channels. Our notion is similar to the secrecy a là Dolev and Yao, as expressed by Abadi in [4]. The static test amounts to checking that the data that may be sent along public channels can only be made of public components.

We use the name carefulness, because the notion of no leaking has been previously used by Abadi, in [1], with a different, even though related, meaning. In that setting, the notion of leaking is formalized in terms of testing equivalence [42, 24] (see Section 1.5 for a review). Roughly, a process $P(x)$ does not leak the input x if a second process Q cannot distinguish running in parallel with $P(M)$ from running in parallel with $P(M')$, for every message M and M'. There, type soundness guarantees that there is no leaking of secret information. The two approaches are compared here in a simplified case, giving the result that if a process P is confined (according to our rules), then it typechecks (according to the Abadi's typing rules) and therefore P does not leak its secret inputs. Moreover, if a process P typechecks, then it is possible to build a Control Flow Analysis solution that makes P confined.

Finally, we consider *message authentication*, i.e. the property requiring that the receiver of a message can ascertain its origin. Handling message authentication at compile-time presents many difficulties, because Control Flow Analysis cannot predict the exact sequence of message exchanges. We then limited our investigation to a weak form of message authentication, called *authorship*, that states when a message is made of packets of information generated by a selected process, that is therefore authenticated.

This chapter is based on ideas developed jointly with Pierpaolo Degano, Flemming Nielson and Hanne Riis Nielson.

3.1 Instrumenting the Spi-Calculus

Syntax. The formulation of the Control Flow Analysis for the spi-calculus [8, 7] (see Section 1.4 for a review), requires a minor extension to the standard syntax of the calculus (we refer to the polyadic version of [1]), namely assigning "labels" to the occurrences of terms, "channels" to the binding occurrences of names within restrictions and "binders" to the binding occurrences of variables. As will emerge later, this is because of the α-conversion allowed by the structural congruence, and the syntactic extension will allow to compute a super-set of the actual values that terms, channels and variables may assume at run-time. These annotations will by no means affect the dynamic semantics of the spi-calculus; however, they heavily influence the way in which our static analysis of Section 3.2 operates. The syntax is based on Def. 1.4.1. There is the further syntax category of expressions, made by labelled terms.

Definition 3.1.1 *Let $\mathcal{L}$ be a non-empty set of* labels *ranged over by l, l', l_i, let $\mathcal{B}$ be a non-empty set of* binders *ranged over by $\beta, \beta', \cdots$; and let $\mathcal{C}$ be a non-empty set of* channels *ranged over by $\chi, \chi', \cdots$; moreover let $\mathcal{B} \cup \mathcal{C}$ be the set of* markers. *Then labelled entities, called* expressions, *$E, T, V \in \mathcal{E}$, and similarly unlabelled entities, called* terms, *$M, N \in \mathcal{M}$, are built according to the following syntax:*

$$
\begin{array}{ll}
M, N ::= & \textit{terms} \\
\quad n & \quad \textit{name} \\
\quad x & \quad \textit{variable} \\
\quad (E, E') & \quad \textit{pair} \\
\quad 0 & \quad \textit{zero} \\
\quad suc(E) & \quad \textit{successor} \\
\quad \{E_1, \ldots, E_k\}_V & \quad \textit{shared-key encryption } (k \geq 0)
\end{array}
$$

E, T, V ::= *expressions*
M^l

The set of cfa-annotated Processes, *denoted by* $P, P_1, P_2, Q, R, \dots \in \mathcal{P}r^{cfa}$ *is defined according to the syntax*

P, Q, R ::=	*processes*
$\mathbf{0}$	*nil*
$\overline{E}\langle V_1, \dots, V_k\rangle.P$	*output*
$E(x_1^{\beta_1}, \dots, x_k^{\beta_k}).P$	*input*
$P \vert P$	*parallel composition*
$(\nu n^{\chi})P$	*restriction*
$[E \ is \ V]P$	*matching*
$!P$	*replication*
$let \ (x^{\beta_x}, y^{\beta_y}) = E \ in \ P$	*pair splitting*
$case \ E \ of \ 0 : P \ suc(x^{\beta}) : Q$	*integer case*
$case \ E \ of \ \{x_1^{\beta_1}, \dots, x_k^{\beta_k}\}_V \ in \ P$	*shared−key decryption* $(k \geq 0)$

The (labelled) name n *and the (labelled) variables* $x,\ y,\ x_1, \dots, x_k$ *are bound in these processes. We often omit the trailing* $\mathbf{0}$ *and write* $\tilde{\cdot}$ *to denote tuples of objects.*

Let *constr* be the generic kind of constructors,

$$constr := suc \mid pair \mid enc,$$

acting on single terms, pairs of terms and k-tuple of terms.
Substitution of expressions, $E[T/x]$, is defined as the usual, e.g. $x^{l_x}[M^l/x] = M^{l_x}$, and also the notions of free names and variables are standard.

We say that $\lfloor M^l \rfloor = \lfloor M'^{l'} \rfloor$ when the two labelled terms have the same underlying expression.

Definition 3.1.2 $\lfloor M_0^{l_0} \rfloor = \lfloor M_1^{l_1} \rfloor$ *is defined as follows:*

- $\lfloor n_0^{l_0} \rfloor = \lfloor n_1^{l_1} \rfloor$ *if* $n_0 = n_1$;
- $\lfloor x_0^{l_0} \rfloor = \lfloor x_1^{l_1} \rfloor$ *if* $x_0 = x_1$;
- $\lfloor 0^{l_0} \rfloor = \lfloor 0^{l_1} \rfloor$;
- $\lfloor suc(M_0^{l_0})^l \rfloor = \lfloor suc(M_1^{l_1})^{l'} \rfloor$ *if* $\lfloor M_0^{l_0} \rfloor = \lfloor M_1^{l_1} \rfloor$;
- $\lfloor (M_0^{l_0}, M_1^{l_1})^l \rfloor = \lfloor (M_2^{l_2}, M_3^{l_3})^{l'} \rfloor$ *if*
  $\lfloor M_0^{l_0} \rfloor = \lfloor M_2^{l_2} \rfloor$ *and* $\lfloor M_1^{l_1} \rfloor = \lfloor M_3^{l_3} \rfloor$;

- $\lfloor \{M_1^{l_1}, \ldots, M_k^{l_k}\}^l_{M_0^{l_0}} \rfloor = \lfloor \{N_1^{l'_1}, \ldots, N_k^{l'_k}\}^{l'}_{N_0^{l'_0}} \rfloor$ *if*
  $\forall i = 0, \ldots, k : \lfloor M_i^{l_i} \rfloor = \lfloor N_i^{l'_i} \rfloor$;

Recall that subjects of actions are usually just atomic terms, i.e. names or variables to be instantiated to names (see Section 1.4).

Congruence. The *structural congruence* $\equiv$ on processes is the same of that of the standard spi-calculus, apart from the following rule, that permits to exchange restrictions only when restricted names are different, because otherwise $(\nu n^{\chi})P \equiv (\nu n^{\chi'})P$.

- $(\nu n^{\chi})(\nu m^{\chi'})P \equiv (\nu m^{\chi'})(\nu n^{\chi})P$, if $n \neq m$.

In the case of α-conversions the markers are not affected *at all*.

Semantics. Annotations will by no means affect the dynamic semantics of the spi-calculus. They only serve as place-holders for our CFA and would not have been needed for a calculus where α-renaming cannot be done freely.

The only difference is that semantics has to to carry the annotations. In particular:

- An abstraction becomes form $F = (x_1^{\beta_1}, \ldots, x_k^{\beta_k})P$.
- A concretion becomes form $C = (\nu m_1^{\chi_1}, \ldots, \nu m_r^{\chi_r})\langle E_1, \ldots, E_k\rangle P$.

Provided that $\{n_1, \ldots, n_r\} \cap fn(P) = \emptyset$, the interactions $F@C$ and $C@F$ are then:

$$F@C = (\nu n_1^{\chi_1}), \ldots, (\nu n_r^{\chi_r})(P[E_1/x_1, \ldots, E_k/x_k] \mid Q)$$
$$C@F = (\nu n_1^{\chi_1}), \ldots, (\nu n_r^{\chi_r})(Q \mid P[E_1/x_1, \ldots, E_k/x_k])$$

Reduction relation. The *reduction* rules mainly differ from the standard spi-calculus ones in the treatment of expressions syntactically equal, but with possibly different labels.

Red Repl
$!P > P \mid !P$

Red Match
$[M^l is M'^{l'}]P > P$ if $\lfloor M^l \rfloor = \lfloor M'^{l'} \rfloor$

Red Split
$let\ (x^{\beta_x}, y^{\beta_y}) = (E, E')\ in\ P > P[E/x, E'/y]$

Red Zero
$case\ 0\ of\ 0 : P\ suc(x^{\beta}) : Q > P$

Red Suc

$case\ suc(E)\ of\ 0 : P\ suc(x^{\beta}) : Q > Q[E/x]$

Red Decrypt

$case\ \{E_1, \ldots, E_k\}_{N^l}\ of\ \{x_1^{\beta_1}, \ldots, x_k^{\beta_k}\}_{N'^{l'}}\ in\ P > P[E_1/x_1, \ldots, E_k/x_k]$ if $\lfloor N^l \rfloor = \lfloor N'^{l'} \rfloor$.

Commitment relation. The commitment relation arrow (see Tab. 1.5) is enriched with a string L made of labels l or made of binders β. They record the labels of the objects sent by an output when deduced with a *Out* rule; symmetrically, for the binders of the objects of inputs. Both kinds of strings appear in the premises of the *Inter* rules: their arities must match and then they are discarded. The rules are reported in Tab. 3.1.

It is immediate to see that one can recover the standard semantics from our instrumented semantics.

Note that

- If $P \xrightarrow[l'_1,\ldots,l'_{k'}]{\overline{m}} (\nu m_1^{\chi_1}, \ldots, m_r^{\chi_r})\langle M_1^{l_1}, \ldots, M_k^{l_k}\rangle Q$ then $k = k'$ and $\forall i = 1, \ldots, k :\ l'_i = l_i$.

- If $P \xrightarrow[\beta'_1,\ldots,\beta'_{k'}]{m} (x_1^{\beta_1}, \ldots, x_k^{\beta_k})Q$ then $k = k'$ and $\forall i = 1, \ldots, k :\ \beta'_i = \beta_i$.

Comm Out

$$\frac{}{\overline{m}\langle M_1^{l_1},\ldots,M_k^{l_k}\rangle.P \xrightarrow[l_1,\ldots,l_k]{\overline{m}} (\nu)\langle M_1^{l_1},\ldots,M_k^{l_k}\rangle P}$$

Comm In

$$\frac{}{m(x_1^{\beta_1},\ldots,x_k^{\beta_k}).P \xrightarrow[\beta_1,\ldots,\beta_k]{m} (x_1^{\beta_1},\ldots,x_k^{\beta_1})P}$$

Comm Inter 1

$$\frac{P \xrightarrow[L_P]{m} F \quad Q \xrightarrow[L_Q]{\overline{m}} C}{P|Q \xrightarrow{\tau} F@C},\ |L_P| = |L_P|$$

Comm Inter 2

$$\frac{P \xrightarrow[L_P]{\overline{m}} C \quad Q \xrightarrow[L_Q]{m} F}{P|Q \xrightarrow{\tau} C@F},\ |L_P| = |L_Q|$$

Comm Par 1

$$\frac{P \xrightarrow[L]{\alpha} A}{P|Q \xrightarrow[L]{\alpha} A|Q}$$

Comm Par 2

$$\frac{Q \xrightarrow[L]{\alpha} A}{P|Q \xrightarrow[L]{\alpha} P|A}$$

Comm Res

$$\frac{P \xrightarrow[L]{\alpha} A \quad \alpha \notin \{m,\overline{m}\}}{(\nu m^{\chi})P \xrightarrow[L]{\alpha} (\nu m^{\chi})A}$$

Comm Red

$$\frac{P > Q \quad P \xrightarrow[L]{\alpha} A}{P \xrightarrow[L]{\alpha} A}$$

Comm Struct

$$\frac{P \equiv Q \quad Q \xrightarrow[L]{\alpha} A \quad A \equiv A'}{P \xrightarrow[L]{\alpha} A'}$$

Table 3.1: The instrumented commitment relation.

3.2 Control Flow Analysis

We assume to analyse only closed processes, i.e. processes without free variables.

The result of our analysis for a process P (with respect to an additional marker environment me for associating names and variables with markers) is a triple (ρ, κ, ζ): the abstract environment ρ gives information about which abstract values can be bound to variables, while the abstract channel environment κ gives information about which abstract values can be communicated over channels. Finally, the abstract cache ζ associates abstract values with each labelled term. We now make the above more precise.

3.2.1 Validation

To *validate* the correctness of a proposed solution (ρ, κ, ζ) we state a set of clauses operating upon Flow Logic judgments on the forms:

$$(\rho, \kappa, \zeta) \ \models_{me} P \quad \text{and} \quad (\rho, \kappa, \zeta) \ \models_{me} M$$

Let $Val = \mathcal{C} \cup \{0\} \cup (\{suc\} \times Val) \cup (\{pair\} \times Val \times Val) \cup (\{enc\} \times Val^{+})$, ranged over by $w, v, \ldots$ be the set of *abstract values* also described by the syntax

$$w, v ::= \chi \mid 0 \mid suc(w) \mid pair(w, w) \mid enc\{w_1, \ldots, w_k\}_{w_0},$$

where, for the sake of simplicity, we use the following shorthands:

- $suc(w)$ stands for (suc, w);
- $pair(w, w')$ stands for $(pair, (w, w'))$;
- $enc\{w_1, \ldots, w_k\}_{w_0}$ stands for $(enc, (w_1, \ldots, w_k, w_0))$.

We extend this notation to sets $W \subseteq \widehat{Val}$, where $\widehat{Val} = \wp(Val)$:

- $\textsc{suc}(W)$ that stands for $\{suc(w) | w \in W\}$;
- $\textsc{pair}(W, W')$ that stands for $\{pair(w, w') | w \in W, w' \in W'\}$;
- $\textsc{enc}\{W_1, \ldots, W_k\}_{W_0}$ that stands for $\{enc\{w_1, \ldots, w_k\}_{w_0} | \forall i = 0, \ldots, k : \ w_i \in W_i\}$.

Let $\textsc{constr}$ be the generic kind of constructions on sets of abstract values,

$$\textsc{constr} := \textsc{suc} \mid \textsc{pair} \mid \textsc{enc},$$

acting on single sets, pairs of sets and k-tuple of sets. The purpose of me, ρ, κ and ζ is clarified by:

$(\rho, \kappa, \zeta) \models_{me} n^l$	iff	$\{me(n)\} \subseteq \zeta(l)$
$(\rho, \kappa, \zeta) \models_{me} x^l$	iff	$\rho(me(x)) \subseteq \zeta(l)$
$(\rho, \kappa, \zeta) \models_{me} (M^{l_1}, N^{l_2})^l$	iff	$(\rho, \kappa, \zeta) \models_{me} M^{l_1} \wedge (\rho, \kappa, \zeta) \models_{me} N^{l_2} \wedge$ $\textsc{pair}(\zeta(l_1), \zeta(l_2)) \subseteq \zeta(l)$
$(\rho, \kappa, \zeta) \models_{me} 0^l$	iff	$\{0\} \subseteq \zeta(l)$
$(\rho, \kappa, \zeta) \models_{me} suc(M^{l_M})^l$	iff	$(\rho, \kappa, \zeta) \models_{me} M^{l_M} \wedge$ $\textsc{suc}(\zeta(l_M)) \in \zeta(l)$
$(\rho, \kappa, \zeta) \models_{me} \{M_1^{l_1}, \ldots, M_k^{l_k}\}^l_{N^{l_0}}$	iff	$\forall i = 1, .., k : (\rho, \kappa, \zeta) \models_{me} M^{l_i} \wedge$ $(\rho, \kappa, \zeta) \models_{me} N^{l_0} \wedge$ $\textsc{enc}\{\zeta(l_1), \ldots, \zeta(l_k)\}_{\zeta(l_0)} \subseteq \zeta(l)$

Table 3.2: Control flow analysis for the spi-calculus expressions.

- $me : \mathcal{N} \cup \mathcal{V} \to \mathcal{B} \cup \mathcal{C}$ is the *marker environment* that associates names and variables with the appropriate channel or binder;

- $\rho : \mathcal{B} \to \widehat{Val}$ is the *abstract environment* that associates binders with the set of abstract values that they can be bound to; more precisely, $\rho(\beta)$ must include the set of abstract values that β could evaluate to.

- $\kappa : \mathcal{C} \to \widehat{Val}$ is the *abstract channel environment* that associates channels with the set of abstract values that can be communicated over them; more precisely, $\kappa(\chi)$ must include the set of abstract values that can be communicated over the channel χ.

- $\zeta : \mathcal{L} \to \widehat{Val}$ is the *abstract cache* that associates abstract values with each labelled term; more precisely $\zeta(l)$ must include the set of the possible actual values of the term labelled l.

The analysis on terms is in Tab. 3.2 and the analysis on processes is in Tab. 3.3. All the rules for validating a compound term or process require that the components are validated. The rules for an expression M^l demand that $\zeta(l)$ contains all the abstract values associated with its components. Moreover, the rule for output requires that the $k-$tuple of sets of abstract values associated with each component of the object can be passed on each channel associated with the subject.

Symmetrically, the rule for input requires that for each $k-$tuple of abstract values passing along the subject, the $k-$tuple of sets of this is contained, pointwise, in the

$k-$tuple of sets to which $\tilde{x}$ can be evaluated to. The condition for matching says that P needs to be validated only if the abstract values of M and of N have a non empty intersection.

Finally, the last three rules ask that for each abstract value associated with the expression to decompose, its sub-components are contained, pointwise, in $\tilde{x}$. Furthermore, we need to extend the analysis to the concretion and abstraction constructs as follows:

Concretion $(\rho,\kappa,\zeta) \models_{me} (\nu\tilde{m}^{\tilde{\chi}})\langle M_1^{l_1},\ldots,M_k^{l_k}\rangle P$	iff	$(\forall i = 1,\ldots,k :$ $(\rho,\kappa,\zeta) \models_{me[\tilde{m}\mapsto\tilde{\chi}]} M_i^{l_i}) \wedge$ $(\rho,\kappa,\zeta) \models_{me[\tilde{m}\mapsto\tilde{\chi}]} P$
Abstraction $(\rho,\kappa,\zeta) \models_{me} (\tilde{x}^{\tilde{\beta}})P$	iff	$(\rho,\kappa,\zeta) \models_{me[\tilde{x}\mapsto\tilde{\beta}]} P$

Example 3.2.1 Consider the following process

$$A = (\nu K_{AB}^{\chi_{AB}})\overline{c_{AS}^{l'}}\langle\{K_{AB}^{l_1}\}_{K_{AS}^{l_3}}^{l_2}\rangle,$$

where the initial marker environment me_I is such that $me_I(c_{AS}) = \chi_A$ and $me_I(K_{AS}) = \chi_{AS}$. Then the minimal solution is given by the triple (ρ,κ,ζ) such that $\rho = \emptyset$ and

$$\kappa(\chi) = \begin{cases} \{\{\chi_{AB}\}_{\chi_{AS}}\} & \text{if } \chi = \chi_A \\ \emptyset & \text{otherwise} \end{cases}$$

$$\zeta(l) = \begin{cases} \{\chi_{AS}\} & \text{if } l = l' \\ \{\chi_{AB}\} & \text{if } l = l_1 \\ \{\{\chi_{AB}\}_{\chi_{AS}}\} & \text{if } l = l_2 \\ \{\chi_{AS}\} & \text{if } l = l_3 \end{cases}$$

A simple check shows that $(\rho,\kappa,\zeta) \models_{me} A$. ∎

Possible optimizations There are several ways to refine our analysis, via more demanding clauses. They can consider, in deeper detail, the problem of *reachability*, i.e. checking whether a process can reach a certain point of the program. For instance, the clause for *integer case* can be refined as follows, that checks P or Q only if they can be reached.

$(\rho,\kappa,\zeta) \models_{me}$	$\mathit{case}\ M^l\ \mathit{of}$ $0^{l'} : P\ \mathit{suc}(x^\beta) : Q$	iff	$(\rho,\kappa,\zeta) \models_{me} M^l \wedge$ $(\zeta(l)\cap\{0\} \neq \emptyset \Rightarrow (\rho,\kappa,\zeta) \models_{me} P) \wedge$ $(\zeta(l)\cap\{\mathit{suc}(w) \mid w \in \widehat{Val}\} \neq \emptyset \Rightarrow$ $(\rho,\kappa,\zeta) \models_{me[x\mapsto\beta]} Q \wedge$ $\forall\ \mathit{suc}(w) \in \zeta(l) : \{w\} \subseteq \rho(\beta))$

$(\rho, \kappa, \zeta) \models_{me} \mathbf{0}$	iff	$true$
$(\rho, \kappa, \zeta) \models_{me} \overline{M^l}\langle N_1^{l_1}, \dots, N_k^{l_k}\rangle.P$	iff	$(\rho, \kappa, \zeta) \models_{me} M^l \wedge$ $\forall i = 1,..,k :\ (\rho, \kappa, \zeta) \models_{me} N_i^{l_i} \wedge$ $(\rho, \kappa, \zeta) \models_{me} P \wedge \forall \chi \in \zeta(l) :$ $\zeta(l_1) \times \dots \times \zeta(l_k) \subseteq \kappa(\chi)$
$(\rho, \kappa, \zeta) \models_{me} M^l(x_1^{\beta_1}, \dots, x_k^{\beta_k}).P$	iff	$(\rho, \kappa, \zeta) \models_{me} M^l \wedge$ $(\rho, \kappa, \zeta) \models_{me[\tilde{x} \mapsto \tilde{\beta}]} P \wedge \forall \chi \in \zeta(l) :$ $\forall (w_1, \dots, w_{k'}) \in \kappa(\chi) :$ if $k = k'$ then $\forall i = 1, \dots, k :\ \{w_i\} \subseteq \rho(\beta_i)$
$(\rho, \kappa, \zeta) \models_{me} P_1 \mid P_2$	iff	$(\rho, \kappa, \zeta) \models_{me} P_1 \wedge (\rho, \kappa, \zeta) \models_{me} P_2$
$(\rho, \kappa, \zeta) \models_{me} (\nu n^{\chi})P$	iff	$(\rho, \kappa, \zeta) \models_{me[n \mapsto \chi]} P$
$(\rho, \kappa, \zeta) \models_{me} !P$	iff	$(\rho, \kappa, \zeta) \models_{me} P$
$(\rho, \kappa, \zeta) \models_{me} [M^l\ is\ N^{l'}]P$	iff	$(\rho, \kappa, \zeta) \models_{me} M^l \wedge (\rho, \kappa, \zeta) \models_{me} N^{l'} \wedge$ $(\rho, \kappa, \zeta) \models_{me} P$
$(\rho, \kappa, \zeta) \models_{me}$ $let\ (x^{\beta_x}, y^{\beta_y}) = M^l$ $in\ P$	iff	$(\rho, \kappa, \zeta) \models_{me[x \mapsto \beta_x, y \mapsto \beta_y]} P \wedge$ $(\rho, \kappa, \zeta) \models_{me} M^l \wedge \forall\ pair(v, w) \in \zeta(l) :$ $\{v\} \subseteq \rho(\beta_x) \wedge \{w\} \subseteq \rho(\beta_y)$
$(\rho, \kappa, \zeta) \models_{me}$ $case\ M^l\ of$ $0 : P\ suc(x^{\beta}) : Q$	iff	$(\rho, \kappa, \zeta) \models_{me} M^l \wedge (\rho, \kappa, \zeta) \models_{me} P \wedge$ $(\rho, \kappa, \zeta) \models_{me[x \mapsto \beta]} Q \wedge \forall\ suc(w) \in \zeta(l) :$ $\{w\} \subseteq \rho(\beta)$
$(\rho, \kappa, \zeta) \models_{me}$ $case\ M^l\ of$ $\{x_1^{\beta_1}, \dots, x_k^{\beta_k}\}_{N^{l_0}}$ $in\ P$	iff	$(\rho, \kappa, \zeta) \models_{me} M^l \wedge (\rho, \kappa, \zeta) \models_{me[\tilde{x} \mapsto \tilde{\beta}]} P \wedge$ $(\rho, \kappa, \zeta) \models_{me} N^{l_0} \wedge$ $\forall\ enc\{w_1, \dots, w_m\}_{w_0} \in \zeta(l) :$ if $m = k \wedge w_0 \in \zeta(l_0)$ then $\forall i = 1, \dots, k :\ \{w_i\} \subseteq \rho(\beta_i)$

Table 3.3: Control flow analysis for the spi-calculus processes.

3.2.2 Existence of solutions

So far we have only considered a procedure for validating whether or not a proposed solution (ρ, κ, ζ) is in fact acceptable. Now, we show that there always exists a least choice of (ρ, κ, ζ) that is acceptable in the manner of Tab. 3.3.

Definition 3.2.2 *The set of proposed solutions can be partially ordered by setting* $(\rho, \kappa, \zeta) \sqsubseteq (\rho', \kappa', \zeta')$ *iff* $\forall \beta \in \mathcal{B} : \rho(\beta) \subseteq \rho'(\beta)$, $\forall \chi \in \mathcal{C} : \kappa(\chi) \subseteq \kappa'(\chi)$ *and* $\forall l \in \mathcal{L} : \zeta(l) \subseteq \zeta'(l)$.

It is immediate that this suffices for making the set of proposed solutions into a complete lattice; using standard notation we write $(\rho, \kappa, \zeta) \sqcup (\rho', \kappa', \zeta')$ for the binary least upper bound (defined pointwise), $\sqcap \mathcal{I}$ for the greatest lower bound of a set $\mathcal{I}$ of proposed solutions (also defined pointwise), and $(\bot, \bot, \bot)$ for the least element.

The following theorem then guarantees that there always is a least solution to the specification in Tab. 3.3 (just take $(\overline{\rho}, \overline{\kappa}, \overline{\zeta}) = (\bot, \bot, \bot)$ in the statement below). Its statement is structured: the first part is defined on terms M^l and the second part, relying on the first one, on processes P. The same structure have most of the statements of Section 3.2.3.

Theorem 3.2.3

1. *For all me,* M^l *and* $(\overline{\rho}, \overline{\kappa}, \overline{\zeta})$ *the set*

$$\{(\rho, \kappa, \zeta) \mid (\rho, \kappa, \zeta) \models_{me} M^l \wedge (\rho, \kappa, \zeta) \sqsupseteq (\overline{\rho}, \overline{\kappa}, \overline{\zeta})\}$$

   is a Moore family.

2. *For all me,* P *and* $(\overline{\rho}, \overline{\kappa}, \overline{\zeta})$ *the set*

$$\{(\rho, \kappa, \zeta) \mid (\rho, \kappa, \zeta) \models_{me} P \wedge (\rho, \kappa, \zeta) \sqsupseteq (\overline{\rho}, \overline{\kappa}, \overline{\zeta})\}$$

   is a Moore family.

Proof. We proceed by structural induction on M^l and on P. Let D stand for both M^l and P and let

$$\mathcal{J} \subseteq \{(\rho, \kappa, \zeta) \mid (\rho, \kappa, \zeta) \models_{me} D \wedge (\rho, \kappa, \zeta) \sqsupseteq (\overline{\rho}, \overline{\kappa}, \overline{\zeta})\}$$

and let J and $(\rho_j, \kappa_j, \zeta_j)$ be given such that $\mathcal{J} = \{(\rho_j, \kappa_j, \zeta_j) \mid j \in J\}$. Next define

$$(\rho', \kappa', \zeta') = \sqcap \mathcal{J} = \sqcap \{(\rho_j, \kappa_j, \zeta_j) \mid j \in J\}.$$

Recall that the greatest lower bound is defined pointwise and hence that $(\rho', \kappa', \zeta') \sqsupseteq (\overline{\rho}, \overline{\kappa}, \overline{\zeta})$. It remains to check that $(\rho', \kappa', \zeta') \models_{me} M^l$ and that $(\rho', \kappa', \zeta') \models_{me} P$. For both, we proceed by cases, making use of the induction hypothesis. Most cases are straightforward and here we only consider the more interesting ones.

1. **The case** n^l. Since $\forall j \in J : (\rho_j, \kappa_j, \zeta_j) \models_{me} n^l$ we have

$$\forall j \in J : \{me(n)\} \subseteq \zeta_j(l).$$

Therefore $\{me(n)\} \subseteq \zeta'(l)$, thus establishing the desired $(\rho', \kappa', \zeta') \models_{me} n^l$.

The case $(M_0^{l_0}, M_1^{l_1})^l$. Since $\forall j \in J : (\rho_j, \kappa_j, \zeta_j) \models_{me} (M_0^{l_0}, M_1^{l_1})^l$ we have that $\forall j \in J$:

$$(\rho_j, \kappa_j, \zeta_j) \models_{me} M_0^{l_0} \;\wedge\; (\rho_j, \kappa_j, \zeta_j) \models_{me} M_1^{l_1} \;\wedge\; \textsc{pair}(\zeta_j(l_0), \zeta_j(l_1)) \subseteq \zeta_j(l).$$

Using the induction hypothesis and that ρ' is obtained in a componentwise manner, we then obtain

$$(\rho', \kappa', \zeta') \models_{me} M_0^{l_0} \wedge (\rho', \kappa', \zeta') \models_{me} M_1^{l_1}.$$

Then, since $\textsc{pair}(\zeta'(l_0), \zeta'(l_1)) \subseteq \zeta'(l)$, we establish the desired $(\rho, \kappa, \zeta) \models_{me} (M_0^{l_0}, M_1^{l_1})^l$.

2. **The case** $M^l(x_1^{\beta_1}, \ldots, x_k^{\beta_k}).P$.
Since $\forall j \in J : (\rho_j, \kappa_j, \zeta_j) \models_{me} M^l(x_1^{\beta_1}, \ldots, x_k^{\beta_k})$ we have that

$$(\rho_j, \kappa_j, \zeta_j) \models_{me} M^l \quad\wedge\quad (\rho_j, \kappa_j, \zeta_j) \models_{me[\tilde{x} \mapsto \tilde{\beta}]} P \text{ and}$$

$$\forall \chi \in \zeta_j(l) \forall (w_1, \ldots, w_{k'}) \in \kappa_j(\chi) : \text{ if } k = k' \text{ then } \forall i = 1, .., k : \{w_i\} \subseteq \rho_j(\beta_i)$$

Using the induction hypothesis and that ρ' is obtained in a componentwise manner, we then obtain

$$(\rho', \kappa', \zeta') \models_{me} M^l \quad\wedge\quad (\rho', \kappa', \zeta') \models_{me[\tilde{x} \mapsto \tilde{\beta}]} P \text{ and}$$

$$\forall \chi \in \zeta'(l) \forall (w_1, \ldots, w_{k'}) \in \kappa'(\chi) : \text{ if } k = k' \text{ then } \forall i = 1, \ldots, k : \{w_i\} \subseteq \rho'(\beta_i)$$

thus establishing the desired $(\rho', \kappa', \zeta') \models_{me} M^l(x_1^{\beta_1}, \ldots, x_k^{\beta_k})$.

■

3.2.3 Correctness

We state now some auxiliary results that will allow us to establish the semantic correctness of our analysis. The former three concern the analysis of agents. They can be proved by applying the analysis of abstractions and concretions and the structural congruence rules in Subsection 3.2.1. The other ones concern the analysis of both terms and processes and can be proved by structural induction.

Lemma 3.2.4 $(\rho, \kappa, \zeta) \models_{me} A|P$ *iff* $((\rho, \kappa, \zeta) \models_{me} A \;\wedge\; (\rho, \kappa, \zeta) \models_{me} P)$*, whenever* A *is an abstraction or a concretion.*

Lemma 3.2.5 *$(\rho, \kappa, \zeta) \models_{me} (\nu \tilde{m}^{\tilde{\chi}})A$ iff $(\rho, \kappa, \zeta) \models_{me[\tilde{m} \mapsto \tilde{\chi}]} A$, whenever A is an abstraction or a concretion.*

Lemma 3.2.6 *Let $F = (x_1^{\beta_1}, \ldots, x_k^{\beta_k})Q$ and $C = (\nu \tilde{n}^{\tilde{\chi}})\langle M_1^{l_1}, \ldots, M_k^{l_k}\rangle P$. If there exists m such that*

- *$(\rho, \kappa, \zeta) \models_{me} C$, with $\emptyset \neq \zeta(l_1) \times \ldots \times \zeta(l_k) \subseteq \kappa(me(m))$,*
- *$(\rho, \kappa, \zeta) \models_{me} F$, and $\forall (w_1, \ldots, w_k) \in \kappa(me(m)) : \forall i : \ \{w_i\} \subseteq \rho(\beta_i)$*

then $(\rho, \kappa, \zeta) \models_{me} F@C$ and $(\rho, \kappa, \zeta) \models_{me} C@F$.

Lemma 3.2.7 *If*

(a) $\lfloor M^l \rfloor = \lfloor M'^{l'} \rfloor$;

(b) $fv(M^l) = \emptyset = fv(M'^{l'})$;

(c) $(\rho, \kappa, \zeta) \models_{me} M^l$ and $(\rho, \kappa, \zeta) \models_{me} M'^{l'}$,

then $\zeta(l) \cap \zeta(l') \neq \emptyset$.

Proof. The proof is by structural induction. We just illustrate the case $M = (M_1^{l_{11}}, N_1^{l_{12}})^l$ and $M'^{l'} = (M_2^{l_{21}}, N_2^{l_{22}})^{l'}$, the other cases are analogous. Our hypotheses are that

(a) $\lfloor (M_1^{l_{11}}, N_1^{l_{12}})^l \rfloor = \lfloor (M_2^{l_{21}}, N_2^{l_{22}})^{l'} \rfloor$;

(b) $fv((M_1^{l_{11}}, N_1^{l_{12}})^l) = \emptyset$;

(c) $(\rho, \kappa, \zeta) \models_{me} (M_1^{l_{11}}, N_1^{l_{12}})^l$ and $(\rho, \kappa, \zeta) \models_{me} (M_2^{l_{21}}, N_2^{l_{22}})^{l'}$,

From (b), we have that $fv(M_1), fv(N_1) = \emptyset$. From $(\rho, \kappa, \zeta) \models_{me} (M_1^{l_{11}}, N_1^{l_{12}})^l$, we have that $\text{PAIR}(\zeta(l_{11}), \zeta(l_{12})) \subseteq \zeta(l)$ and from $(\rho, \kappa, \zeta) \models_{me} (M_2^{l_{21}}, N_2^{l_{22}})^{l'}$, we have that $\text{PAIR}(\zeta(l_{21}), \zeta(l_{22})) \subseteq \zeta(l')$. By inductive hypothesis, we have that $\zeta(l_{11}) \cap \zeta(l_{21}) \neq \emptyset$ and $\zeta(l_{21}) \cap \zeta(l_{22}) \neq \emptyset$ and finally $\zeta(l) \cap \zeta(l') \neq \emptyset$. ∎

Corollary 3.2.8 *If $fv(M^l) = \emptyset$, $(\rho, \kappa, \zeta) \models_{me} M^l$ then $\zeta(l) \neq \emptyset$.*

Lemma 3.2.9
For all $z \in fn(M^l) \cup fv(M^l)$ and respectively, for all $z \in fn(P) \cup fv(P)$ such that $me_1(z) = me_2(z)$:

1. *$(\rho, \kappa, \zeta) \models_{me_1} M^l$ if and only if $(\rho, \kappa, \zeta) \models_{me_2} M^l$.*
2. *$(\rho, \kappa, \zeta) \models_{me_1} P$ if and only if $(\rho, \kappa, \zeta) \models_{me_2} P$.*

Proof. A straightforward structural induction on M and P, respectively. ■

Lemma 3.2.10

1. *Assume that* $me(m) = me(n)$ *for* $m, n \in \mathcal{N}$*; then*
   (a) $(\rho, \kappa, \zeta) \models_{me} M^l$ *if and only if* $(\rho, \kappa, \zeta) \models_{me} M^l[m/n]$*;*
   (b) $(\rho, \kappa, \zeta) \models_{me} P$ *if and only if* $(\rho, \kappa, \zeta) \models_{me} P[m/n]$*.*
2. *Assume that* $me(x) = me(y)$ *for* $x, y \in \mathcal{V}$*; then*
   (a) $(\rho, \kappa, \zeta) \models_{me} M^l$ *if and only if* $(\rho, \kappa, \zeta) \models_{me} M^l[x/y]$*;*
   (b) $(\rho, \kappa, \zeta) \models_{me} P$ *if and only if* $(\rho, \kappa, \zeta) \models_{me} P[x/y]$*.*

Proof. By structural induction.

■

Corollary 3.2.11

1. *Assume that* $z \notin fv(M^l)$ *and, respectively,* $z \notin fv(P)$ *and* $\beta \in \mathcal{B}$*; then* $(\rho, \kappa, \zeta) \models_{me[y \mapsto \beta]} M^l$ *iff* $(\rho, \kappa, \zeta) \models_{me[z \mapsto \beta]} M^l[z/y]$ *and, respectively,* $(\rho, \kappa, \zeta) \models_{me[y \mapsto \beta]} P$ *iff* $(\rho, \kappa, \zeta) \models_{me[z \mapsto \beta]} P[z/y]$*.*
2. *Assume that* $m \notin fn(M^l)$ *and, respectively,* $m \notin fn(P)$ *and* $\chi \in \mathcal{C}$*; then* $(\rho, \kappa, \zeta) \models_{me[n \mapsto \chi]} M^l$ *iff* $(\rho, \kappa, \zeta) \models_{me[m \mapsto \chi]} M^l[m/n]$ *and, respectively,* $(\rho, \kappa, \zeta) \models_{me[n \mapsto \chi]} P$ *iff* $(\rho, \kappa, \zeta) \models_{me[m \mapsto \chi]} P[m/n]$*.*

Lemma 3.2.12
Assume that $P \equiv Q$ *then* $(\rho, \kappa, \zeta) \models_{me} P$ *iff* $(\rho, \kappa, \zeta) \models_{me} Q$*.*

Proof. The proof is by induction on the construction of $P \equiv Q$ and here we only consider the two harder cases.

The case of α-equivalence. Consider the sub-case $(\nu n^\chi)P \equiv (\nu m^\chi)(P[m/n])$ where $m \notin fn((\nu n^\chi)P)$. We calculate that

$$(\rho, \kappa, \zeta) \models_{me} (\nu n^\chi)P$$

is equivalent to

$$(\rho, \kappa, \zeta) \models_{me[n \mapsto \chi]} P$$

which by Corollary 3.2.11 is equivalent to

$$(\rho, \kappa, \zeta) \models_{me[m \mapsto \chi]} P[m/n]$$

(since either $m \notin fn(P)$ or $m = n$) which is equivalent to

$$(\rho, \kappa, \zeta) \models_{me} (\nu m^{\chi})(P[m/n])$$

as was to be shown.
The other sub-cases are similar.

The cases $(\nu n^{\chi})(P_1|P_2) \equiv (\nu n^{\chi})P_1|P_2$ (if $n \notin fn(P_2)$) and $(\nu n^{\chi})P \equiv P$ (if $n \notin fn(P)$) are easy consequences of Lemma 3.2.9.
The case $(\nu n^{\chi})(\nu m^{\chi'})P \equiv (\nu m^{\chi'})(\nu n^{\chi})P$ (if $n \neq m$) is straightforward. ∎

Lemma 3.2.13 (Substitution lemma)

1. *If* $(\rho, \kappa, \zeta) \models_{me[x \mapsto \beta]} \hat{M}^{\hat{l}}$ *and* $(\rho, \kappa, \zeta) \models_{me} M^l$, *with* $\zeta(l) \subseteq \rho(\beta)$ *then* $(\rho, \kappa, \zeta) \models_{me} \hat{M}^{\hat{l}}[M^l/x^{l_x}]$.
2. *If* $(\rho, \kappa, \zeta) \models_{me[x \mapsto \beta]} P$ *and* $(\rho, \kappa, \zeta) \models_{me} M^l$, *with* $\zeta(l) \subseteq \rho(\beta)$ *then* $(\rho, \kappa, \zeta) \models_{me} P[M^l/x^{l_x}]$.

Proof. The proof is by structural induction. We just illustrate the more interesting cases.

1. **The case** y. If $x \neq y$, trivial. Otherwise, if $x = y$, let us consider the sub-cases depending on the structure of M. If $M^l = n^l$, we need to prove that $(\rho, \kappa, \zeta) \models_{me} x^{l_x}[n^l/x^{l_x}]$, i.e. $(\rho, \kappa, \zeta) \models_{me} n^{l_x}$. This holds if and only if $\{me(n)\} \subseteq \zeta(l_x)$. Since n^l is validated, we know that $\{me(n)\} \subseteq \zeta(l)$. By hypothesis, $\zeta(l) \subseteq \rho(\beta)$ and $\rho(\beta) \subseteq \zeta(l_x)$, therefrom $\{me(n)\} \subseteq \zeta(l_x)$.
   If $M^l = z^l$ the proof is analogous and for 0 is trivial.
   For composed term, let M^l be $constr(M_1, \dots, M_k)$. We will use the following parametric clause for validating M^l:

   $$(\rho, \kappa, \zeta) \models_{me} M^l \text{ iff } \begin{array}{l} \forall i = 1, \dots, k : (\rho, \kappa, \zeta) \models_{me} M_i^{l_i} \wedge \\ \textsc{constr}(\zeta(l_1), \dots, \zeta(l_k)) \subseteq \zeta(l) \end{array}$$

   In this case, we need to prove that $(\rho, \kappa, \zeta) \models_{me} x^{l_x}[M^l/x^{l_x}]$, that is $(\rho, \kappa, \zeta) \models_{me} M^{l_x}$. The condition $\forall M_i^{l_i} : (\rho, \kappa, \zeta) \models_{me} M_i^{l_i}$ holds independently from the top-level label. We are left to prove that $\textsc{constr}(\zeta(l_1), \dots, \zeta(l_k)) \subseteq \zeta(l_x)$. Since M^l is validated, we know that $\textsc{constr}(\zeta(l_1), \dots, \zeta(l_k)) \subseteq \zeta(l)$. By hypothesis, $\zeta(l) \subseteq \rho(\beta)$ and $\rho(\beta) \subseteq \zeta(l_x)$, from which follows that $\textsc{constr}(\zeta(l_1), \dots, \zeta(l_k)) \subseteq \zeta(l_x)$.

2. **The case** $M_1^{l_1}(x_1^{\beta_1}, \dots, x_k^{\beta_k}).P$.
   Without loss of generality, we can assume that $x_i \notin fv(M^l)$. We need to prove that $(\rho, \kappa, \zeta) \models_{me} (M_1^{l_1}(x_1^{\beta_1}, \dots, x_k^{\beta_k}).P)[M^l/x^{l_x}]$, that is:

(a) $(\rho, \kappa, \zeta) \models_{me} M_1^{l_1}[M^l/x^{l_x}]$,

(b) $(\rho, \kappa, \zeta) \models_{me[\tilde{x}\mapsto\tilde{\beta}]} P[M^l/x^{l_x}]$, and

(c) $\forall \chi \in \zeta(l_1)$:
$\forall (w_1, \ldots, w_{k'}) \in \kappa(\chi)$: if $k = k'$ then $\forall i = 1, \ldots, k$: $\{w_i\} \subseteq \rho(\beta_i)$.

Now, (a) and (c) hold, because of the corresponding conditions on the process $M_1^{l_1}(x_1^{\beta_1}, \ldots, x_k^{\beta_k}).P$ and by induction hypothesis. If $x \notin \{x_1, \ldots, x_k\}$, then by (b) holds, whereas if $x \in \{x_1, \ldots, x_k\}$, then $M^l \in \mathcal{V}$ and the thesis follows from Lemma 3.2.9.

■

Subject reduction. To establish the semantic correctness of our analysis we rely on the definition of commitment semantics seen in the previous section. A subject-reduction result is established both for the reduction and the commitment relations. The subject reduction results below apply to *all* the solutions of the analysis, and hence in particular to the least.
Recall that we assumed that processes under analysis are closed and that annotations do not affect reductions.

Theorem 3.2.14 (Subject Reduction for $>$)
Let P a closed process. If $(\rho, \kappa, \zeta) \models_{me} P$ and $P > Q$ then $(\rho, \kappa, \zeta) \models_{me} Q$.

Proof. Repl Trivial.

Match By Lemma 3.2.7, $\zeta(l) \cap \zeta(l') \neq \emptyset$. This implies that $(\rho, \kappa, \zeta) \models_{me} Q$.

Split $(\rho, \kappa, \zeta) \models_{me}$ *let* $(x^{\beta_x}, y^{\beta_y}) = (M^{l_M}, N^{l_N})^l$ *in* P if and only if

(a) $(\rho, \kappa, \zeta) \models_{me[x\mapsto\beta_x, y\mapsto\beta_y]} P$,

(b) $(\rho, \kappa, \zeta) \models_{me} M^{l_M}, N^{l_N}$,

(c) $\textsc{pair}(\zeta(l_M), \zeta(l_N)) \subseteq \zeta(l)$ and

(d) $\forall\, pair(v, w) \in \zeta(l)$: $\{v\} \subseteq \rho(\beta_x) \wedge \{w\} \subseteq \rho(\beta_y)$.

To prove $(\rho, \kappa, \zeta) \models_{me} P[M^{l_M}/x, N^{l_N}/y]$, we need to use twice Lemma 3.2.13, i.e. we show each time, that its three hypotheses are fulfilled. In both cases, the first two hypotheses hold because of (a) and (b). We need to prove the third one, i.e. that $\zeta(l_M) \subseteq \rho(\beta_x)$ and, respectively, $\zeta(l_N) \subseteq \rho(\beta_y)$. Now, by Corollary 3.2.8, both $\zeta(l_M) \neq \emptyset$ and $\zeta(l_M) \neq \emptyset$. By (d), for all $pair(v, w) \in \zeta(l)$, and in particular, by (c), for all $pair(v, w) \in \textsc{pair}(\zeta(l_M), \zeta(l_N))$ we have that $\{v\} \subseteq \rho(\beta_x) \wedge \{w\} \subseteq \rho(\beta_y)$, hence the third condition is proved.

Zero Trivial.

Suc Analogous to **Split**.

Decrypt $(\rho, \kappa, \zeta) \models_{me} case \; \{M_1, \ldots, M_k\}^l_{M_0^{l_0}} \; of \; \{x_1^{\beta_1}, \ldots, x_k^{\beta_k}\}_{M_0^{\prime l_0'}} \; in \; P$ iff

(a) $(\rho, \kappa, \zeta) \models_{me[\tilde{x} \mapsto \tilde{\beta}]} P$,

(b) $(\rho, \kappa, \zeta) \models_{me} M_i^{l_i}$,

(c) $\text{ENC}\{\zeta(l_1), \ldots, \zeta(l_k)\}_{\zeta(l_0)} \subseteq \zeta(l)$ and

(d) $\forall \; enc\{w_1, \ldots, w_k\}_{w_0} \in \zeta(l): \; w_0 \in \zeta(l_0) \; \forall i = 1, \ldots, k: \; \{w_i\} \subseteq \rho(\beta_i)$.

To prove $(\rho, \kappa, \zeta) \models_{me} P[M_1^{l_1}/x_1, \ldots, M_k^{l_k}/x_k]$, we need to use Lemma 3.2.13 k times, i.e. we show that its three hypotheses are fulfilled. The first two hold because of (a) and (b). We need to prove the third one, i.e. that $\forall i : 1, \ldots, k : \zeta(l_i) \subseteq \rho(\beta_i)$. Now, by Corollary 3.2.8, $\forall i : 1, \ldots, k : \zeta(l_i) \neq \emptyset$. By (d), for all $enc\{w_1, \ldots, w_k\}_{w_0} \in \zeta(l)$, and in particular, by (c), for all $enc\{w_1, \ldots, w_k\}_{w_0} \in \text{ENC}\{\zeta(l_1), \ldots, \zeta(l_N)\}_{\zeta(l_1)}$ we have that $\forall i = 1, \ldots, k : w_i \subseteq \rho(\beta_i)$, hence the third condition is proved.

■

Theorem 3.2.15 (Subject Reduction for $\xrightarrow[L]{\alpha}$)

Let P a closed process.

(1) If $(\rho, \kappa, \zeta) \models_{me} P$ and $P \xrightarrow{\tau} Q$ then $(\rho, \kappa, \zeta) \models_{me} Q$.

(2) If $(\rho, \kappa, \zeta) \models_{me} P$ and $P \xrightarrow[l_1, \ldots, l_k]{\overline{m}} A$ then $(\rho, \kappa, \zeta) \models_{me} A$ and

$\emptyset \neq \zeta(l_1) \times \ldots \times \zeta(l_k) \subseteq \kappa(me(m)) \;\; [cond \; 1]$

(3) If $(\rho, \kappa, \zeta) \models_{me} P$ and $P \xrightarrow[\beta_1, \ldots, \beta_k]{m} A$ then $(\rho, \kappa, \zeta) \models_{me} A$, and

$(\forall (w_1, \ldots, w_k') \in \kappa(me(m)) : \;\; if \; k = k' \; then$
$\forall i = 1, \ldots, k : \; \{w_i\} \subseteq \rho(\beta_i)) \;\; [cond \; 2]$

Proof. The proof is by induction on the construction of $P \xrightarrow[L]{\alpha} A$.

Comm Out

$$\frac{}{\overline{m^l}\langle M_1^{l_1}, \ldots, M_k^{l_k}\rangle.P' \xrightarrow[l_1, \ldots, l_k]{\overline{m}} (\nu)\langle M_1^{l_1}, \ldots, M_k^{l_k}\rangle P'}$$

Then $P = \overline{m^l}\langle M_1^{l_1}, \ldots, M_k^{l_k}\rangle.P'$ and $A = (\nu)\langle M_1^{l_1}, \ldots, M_k^{l_k}\rangle P'$.
(1) and (3) hold vacuously.
(2) Since $(\rho, \kappa, \zeta) \models_{me} P$ then

(a) $(\rho, \kappa, \zeta) \models_{me} m^l$ and for $i = 1, \ldots, k : (\rho, \kappa, \zeta) \models_{me} M_i^{l_i}$,

(b) $(\rho, \kappa, \zeta) \models_{me} P'$ and

(c) $\forall \chi \in \zeta(l) : \zeta(l_1) \times \ldots \times \zeta(l_k) \subseteq \kappa(\chi)$.

The first two conditions suffice to prove that $(\rho, \kappa, \zeta) \models_{me} A$. The third one together with the fact that $\{me(m)\} \subseteq \zeta(l)$, implies that the condition [*cond* 1], i.e. $\zeta(l_1) \times \ldots \times \zeta(l_k) \subseteq \kappa(me(m))$ holds. Recall that, by Corollary 3.2.8, $\zeta(l_i) \neq \emptyset$.

Comm In

$$\frac{}{m^l(x_1^{\beta_1}, \ldots, x_k^{\beta_k}).P' \xrightarrow[\beta_1,\ldots,\beta_k]{m} (x_1, \ldots, x_k)P'}$$

Then $P = m^l(x_1, \ldots, x_k).P'$ and $A = (x_1, \ldots, x_k)P'$.
(1) and (2) hold vacuously.
(3) Since $(\rho, \kappa, \zeta) \models_{me} P$ then

(a) $(\rho, \kappa, \zeta) \models_{me} m^l$,

(b) $(\rho, \kappa, \zeta) \models_{me[\tilde{x} \mapsto \tilde{\beta}]} P'$ and

(c) $\forall \chi \in \zeta(l)\ \forall (w_1, \ldots, w'_k) \in \kappa(\chi)$: if $k = k'$ then for $i = 1, \ldots, k : \{w_i\} \subseteq \rho(\beta_i)$.

The first two conditions suffice to prove that $(\rho, \kappa, \zeta) \models_{me} A$. The third one together with the fact that $\{me(m)\} \subseteq \zeta(l)$, implies that the condition [*cond* 2], i.e. $(\forall (w_1, \ldots, w'_k) \in \kappa(me(m)) :$ if $k = k'$ then $\forall i = 1, \ldots, k : \{w_i\} \subseteq \rho(\beta_i))$ holds.

Comm Par$_1$

$$\frac{P \xrightarrow[L]{\alpha} A}{P|Q \xrightarrow[L]{\alpha} A|Q}$$

By Lemma 3.2.13, $(\rho, \kappa, \zeta) \models_{me} A|Q$ if and only if $(\rho, \kappa, \zeta) \models_{me} A$ (that holds by induction hypothesis) and $(\rho, \kappa, \zeta) \models_{me} Q$ (that holds because $(\rho, \kappa, \zeta) \models_{me} P|Q$). If $\alpha = \tau$ no check is needed. Otherwise, either [*cond* 1] or [*cond* 2] still hold.

Comm Par$_2$ analogous to the previous one.

Comm Res

$$\frac{P \xrightarrow[L]{\alpha} A \quad \alpha \notin m, \overline{m}}{(\nu m^{\chi})P \xrightarrow[L]{\alpha} (\nu m^{\chi})A}$$

Whichever action α is, $(\rho, \kappa, \zeta) \models_{me} (\nu m^{\chi})P$ holds, by Lemma 3.2.9, if and only if $(\rho, \kappa, \zeta) \models_{me[m \mapsto \chi]} A$, that, in turn, holds by induction hypothesis. To prove our claim, we have three exhaustive cases, depending on what α is.

[$\alpha = \tau$] trivial.

[$\alpha = \overline{n}$] [*cond* 2], i.e. $\zeta(l_1) \times \ldots \times \zeta(l_k) \subseteq \kappa(me'(n))$, with $me' = me[m \mapsto \chi]$, holds because $\zeta(l_1) \times \ldots \times \zeta(l_k) \subseteq \kappa(me(n))$ and because the two marker environments coincide on the value n ($\alpha \neq m$).

[$\alpha = n$] analogous.

Comm Red

$$\frac{P > Q \quad Q \xrightarrow[L]{\alpha} A}{P \xrightarrow[L]{\alpha} A}$$

By Theorem 3.2.14 and by induction hypothesis.

Comm Inter$_1$

$$\frac{P \xrightarrow[\tilde{\beta}]{m} F \quad Q \xrightarrow[\tilde{\lambda}]{\overline{m}} C}{P|Q \xrightarrow{\tau} F@C}$$

We have to prove $(\rho, \kappa, \zeta) \models_{me} F@C$, that holds by Lemma 3.2.6.

Comm Inter$_2$ Analogous.

■

3.3 Confinement for the Spi-Calculus

We here extend the notion of *no leaking* and of *confinement* seen for the π-calculus, ensuring that data, composed with channels devised to be secret to a certain process P, are never communicated to an external user. This property requires more machinery than in the π-calculus, mainly because terms are structured and because of cryptographic constructs.

The dynamic notion says that a process is *careful* if it does not perform any outputs of secret terms on names associated with public channels. The static test amounts to checking that the data that may be sent along public channels can only be made of public components.

Here, we assume to use the monadic calculus, as far as actions are concerned; the generalization to the polyadic case is immediate.

By technical convenience, we instrument the semantics to more precisely record the markers, during the execution of actions, in the form of a marker environment me. This will obviously be tightly related to the marker environments used in validating a solution (ρ, κ, ζ) for a process P. Therefore the new commitment relation is written

$$me \vdash P \xrightarrow[L]{\alpha} A,$$

where me is a marker environment, that is assumed to bind all the free names of the closed process P, α is an action, L is a list of labels or binders and A is a closed agent. The new rules are in Tab. 3.4, where the rule *Comm Par 2* has been omitted.

This implies to carry on this modification to all our previous results, in particular to the subject reduction ones. Naturally, the standard semantics can be recovered by discarding me, while there always exists a marker environment me such that if a transition $P \xrightarrow[L]{\alpha} A$ is deducible in the standard semantics, then $me \vdash P \xrightarrow[L]{\alpha} A$ is deducible in the modified semantics. Moreover, for technical reasons we will make use of the following straightforward fact. It states that enlarging (*not* updating) the marker environment does not affect the transitions of a process P:

Lemma 3.3.1

$$\frac{me \supseteq me' \quad me' \vdash P \xrightarrow[L]{\alpha} A}{me \vdash P \xrightarrow[L]{\alpha} A}.$$

As in Chapter 2, we assume that channels $\mathcal{C}$ are partitioned in public, $\mathcal{P}$, and secret, $\mathcal{S}$. We also assume that the channels associated with free names of processes are all public; it follows that the secret channels either do not occur at all or are annotations on names which are restricted within the process. Of course, numbers are public.

Then, this partition is used as a basis to partition both expressions and abstract values. They will be divided in the two kinds *secret* and *public*, expressing the fact

Comm Out

$$\frac{}{me \vdash \overline{m}\langle M_1^{l_1}, \ldots, M_k^{l_k}\rangle.P \xrightarrow[l_1,\ldots,l_k]{\overline{m}} (\nu)\langle M_1^{l_1}, \ldots, M_k^{l_k}\rangle P}$$

where $fn(\overline{m}\langle M_1^{l_1}, \ldots, M_k^{l_k}\rangle.P) \subseteq dom(me)$

Comm In

$$\frac{}{me \vdash m(x_1^{\beta_1}, \ldots, x_k^{\beta_k}).P \xrightarrow[\beta_1,\ldots,\beta_k]{m} (x_1^{\beta_1}, \ldots, x_k^{\beta_1})P}$$

where $fn(m(x_1^{\beta_1}, \ldots, x_k^{\beta_k}).P) \subseteq dom(me)$

Comm Inter 1

$$\frac{me \vdash P \xrightarrow[L_P]{m} F \quad me \vdash Q \xrightarrow[L_Q]{\overline{m}} C}{me \vdash P|Q \xrightarrow{\tau} F@C},$$

$|L_P| = |L_Q|$

Comm Inter 2

$$\frac{me \vdash P \xrightarrow[L_P]{\overline{m}} C \quad me \vdash Q \xrightarrow[L_Q]{m} F}{me \vdash P|Q \xrightarrow{\tau} C@F},$$

$|L_P| = |L_Q|$

Comm Par 1

$$\frac{me \vdash P \xrightarrow[L]{\alpha} A}{me \vdash P|Q \xrightarrow[L]{\alpha} A|Q}$$

Comm Res

$$\frac{me[m \mapsto \chi] \vdash P \xrightarrow[L]{\alpha} A \quad \alpha \notin \{m, \overline{m}\}}{me \vdash (\nu m^\chi)P \xrightarrow[L]{\alpha} (\nu m^\chi)A}$$

$$Struct \quad \frac{P \equiv Q \quad me \vdash Q \xrightarrow[L]{\alpha} A \quad A \equiv A'}{me \vdash P \xrightarrow[L]{\alpha} A'}$$

Comm Red

$$\frac{P > Q \quad me \vdash P \xrightarrow[L]{\alpha} A}{me \vdash P \xrightarrow[L]{\alpha} A}$$

Table 3.4: The instrumented commitment relation.

that one wishes to maintain the confidentiality of data of the first kind. We use the word "kind" opposed to type, to stress that we are not dealing with Type Systems.

Abstract values and expressions are given a kind by the following two operators, the first acting statically (at analysis time), the second dynamically (at execution time). Both follow the same pattern. The intuition behind them is that a single "drop" of *secret* turns to *secret* the kind of a whole abstract value or of a expression. Of course there is the exception for encrypted data: what is encrypted with a key of kind *secret* has kind *public*, even if it contains components of kind *secret*.

The operator *Skind*, (S stands for static) given an abstract value w returns *secret*, if one of the components of w is a secret channel, and returns *public* otherwise. It represents, in a sense, the abstract type of w. Its definition follows, and is reused in that of the dynamic operator.

Definition 3.3.2 *The* static *operator* $Skind : Val \to \{secret, public\}$ *is defined as follows:*

- $Skind(\chi) = \begin{cases} secret & \text{if } \chi \in \mathcal{S} \\ public & \text{if } \chi \in \mathcal{P} \end{cases}$
- $Skind(0) = public$;
- $Skind(pair(w, w')) =$
$\begin{cases} secret & \text{if } (Skind(w) = secret \vee Skind(w') = secret) \\ public & \text{otherwise;} \end{cases}$
- $Skind(suc(w)) = Skind(w)$;
- $Skind(enc\{w_1, \ldots, w_k\}_{w_0}) =$
$\begin{cases} Skind(\{w_1, \ldots, w_k\}) & \text{if } Skind(w_0) = public \wedge k \neq 0 \\ public & \text{otherwise,} \end{cases}$

where, by abuse of notation,
$Skind(W) = \begin{cases} secret & \text{if } \exists w \in W : Skind(w) = secret \\ public & \text{otherwise.} \end{cases}$

The operator *Dkind* (D stands for dynamic), given an expression E returns *secret*, if one of the name composing E is bound by me to a secret channel, and returns *public* otherwise. It exploits an auxiliary function to extract the abstract values of a closed expression E with free names bound by a given marker environment me.

Definition 3.3.3 *Given a marker environment me, the* dynamic *operator Dkind maps closed expressions into* $\{secret, public\}$ *and is defined as*

$$Dkind(me, E) = Skind(f(me, E))$$

where the translation *function f is defined as:*

- $f(me, 0^l) = \{0\}$;
- $f(me, (E, E')^l) = \textsc{pair}(f(me, E), f(me, E'))$;

- $f(me, n^l) = \{me(n)\}$; • $f(me, suc(E)^l) = \textsc{suc}(f(me, E))$;
- $f(me, \{E_1, \ldots, E_k\}^l_{E_0}) = \textsc{enc}\{f(me, E_1), \ldots, f(me, E_k)\}_{f(me,E_0)}$.

It is immediate to see that if an expression is considered to be secret, i.e. has kind *secret*, then so is also the set of its corresponding abstract values. Instead, if $Dkind(me, M^l) = public$ then $Skind(\zeta(l))$ is not necessarily *public*.

Lemma 3.3.4 *If* $(\rho, \kappa, \zeta) \models_{me} M^l$ *and* $Dkind(me, M^l) = secret$ *then* $Skind(\zeta(l)) = secret$.

Proof. By structural induction on M.
The case n^l. Since $(\rho, \kappa, \zeta) \models_{me} n^l$, then $\{me(n)\} \in \zeta(l)$ and by Def. 3.3.2 $me(n) = \chi \in \mathcal{S}$ then by Def. 3.3.2, $Skind(\zeta(l)) = secret$.
The case $(M_0^{l_0}, M_1^{l_1})^l$. By induction hypothesis, $Skind(\zeta(l_0)) = secret$ or $Skind(\zeta(l_1)) = secret$. Moreover, since $(\rho, \kappa, \zeta) \models_{me} (M_0^{l_0}, M_1^{l_1})^l$, we have that $\textsc{pair}(\zeta(l_0), \zeta(l_1)) \subseteq \zeta(l)$, from which the thesis follows, by Def. 3.3.2.
The other cases are analogous. ∎

Dynamic notion We are ready to introduce our notion of secrecy. We say that a process is *careful* if it does not perform outputs of confidential expressions on names associated with public channels. We will use hereafter me_I to denote the initial marker environment for a process P, such that $me_I[fn(P)] \subseteq \mathcal{P}$.

Definition 3.3.5 *A process P is* careful *w.r.t.* $(\mathcal{S}, me_I)$ *iff whenever* $me_I \vdash P \rightarrow^* P' \xrightarrow{\tau} P''$, *with the last step deduced with the premise*

$$me \vdash R \xrightarrow[l]{\overline{m}} (\nu)\langle M^l \rangle R', \quad \textit{with } me(m) \in \mathcal{P},$$

it is always the case that $Dkind(me, M^l) = public$.

Static notion We now define the confinement property for the spi-calculus. It predicts at compile time that a process is careful. Intuitively, the notion requires that only abstract values of type *public* can be transmitted along a public channel: for each public channel χ, the abstract values contained in its channel environment, i.e. $\kappa(\chi)$, all have kind *public*.

Definition 3.3.6 *A system P is* confined *w.r.t.* $(\mathcal{S}, me_I)$ *iff there exists* (ρ, κ, ζ) *such that*

- $(\rho, \kappa, \zeta) \models_{me_I} P$;
- $\forall \chi \in \mathcal{P} : \quad Skind(\kappa(\chi)) = public$.

The subject reduction theorem extends trivially to confined processes thereby paving the way for showing that the static notion implies the dynamic one.

Corollary 3.3.7 (Subject reduction for confinement)
Let P be confined and $me_I \vdash P \xrightarrow{\tau} P'$; then P' is confined.

Theorem 3.3.8 *If P is confined (with respect to $\mathcal{S}, me_I$) then P is careful.*

Proof. By Corollary 3.3.7, it suffices to consider a computation made of a single transition, i.e.

$$me_I \vdash P \xrightarrow{\tau} P'.$$

Suppose P is confined (with respect to $\mathcal{S}, me_I$), but not careful, i.e. that in a computation ξ, a communication has been deduced with the premise

$$me \vdash R \xrightarrow[l]{\overline{m}} (\nu)\langle M^l \rangle R',$$

with $Dkind(me, M^l) = secret$ and $me(m) \in \mathcal{P}$. Then, by hypothesis and by Lemma 3.3.4, $Skind(\zeta(l)) = secret$. Moreover, since P is confined, $(\rho, \kappa, \zeta) \models_{me} P$ and also $(\rho, \kappa, \zeta) \models_{me} R$. To check R, the analysis guarantees that there exists me' such that $me(m) = me'(m)$ and $(\rho, \kappa, \zeta) \models_{me'} \overline{m}^{l_m} \langle M^l \rangle .R'$, that implies that

$$\forall \chi \in \zeta(l_m) : \zeta(l) \subseteq \kappa(\chi).$$

Since $Skind(\zeta(l)) = secret$ and $\chi = me(m) \in \zeta(l_m)$ $(me(m) \in \mathcal{P})$, we have that there exists $\chi \in \mathcal{P}$ such that $Skind(\kappa(\chi)) = secret$ against the hypothesis of confinement. ■

Example 3.3.9 We consider here the cryptographic version of the Wide Mouthed Frog key exchange protocol considered in Example 2.4.8. It is an adaptation of the simplified version analyzed in [7]. Again, the two processes A and B share keys K_{AS} and K_{BS} with a trusted server S. In order to establish a secure channel with B, A sends a fresh key K_{AB} encrypted with K_{AS} to the server S. Then, the server decrypts the key and forwards it to B, this time encrypted with K_{BS}. Now B has the key K_{AB} and A can send a fresh message M (for simplicity, a name, but it could also be an expression composed with secret restricted names, as well) encrypted with K_{AB} to B. The analysis should guarantee that M is kept secret.

The protocol is composed of the following three messages:

$$\begin{array}{lll} \text{Message 1} & A \to S : & \{K_{AB}\}_{K_{AS}} \\ \text{Message 2} & S \to B : & \{K_{AB}\}_{K_{BS}} \\ \text{Message 3} & A \to B : & \{M\}_{K_{AB}} \end{array}$$

and its specification is:

$$
\begin{array}{rcl}
P & = & (\nu K_{AS}^{\chi_{k1}})(\nu K_{BS}^{\chi_{k2}})(\ (A|B)\ |S) \\
A & = & (\nu K_{AB}^{\chi_{k3}})(\nu M^{\chi_M})(\overline{c_{AS}^{l_{c1}}}\langle\{K_{AB}^{l_{k3}}\}_{K_{AS}^{l_{k1}}}\rangle.\overline{c_{AB}^{l_{c3}}}\langle\{M^{l_M}\}_{K_{AB}^{l_{k3}}}\rangle) \\
S & = & c_{AS}^{l_{c1}}(x^{\beta_x}).case\ x^{l_x}\ of\ \{s^{\beta_s}\}_{K_{AS}^{l_{k1}}}\ in\ \overline{c_{BS}}\langle\{s^{l_s}\}_{K_{BS}^{l_{k2}}}\rangle \\
B & = & c_{BS}^{l_{c2}}(w^{\beta_w}).case\ w^{l_w}\ of\ \{y^{\beta_y}\}_{K_{BS}^{l_{k2}}}\ in\ c_{AB}^{l_{c3}}(z^{\beta_z}). \\
 & & case\ z^{l_z}\ of\ \{r^{\beta_r}\}_{y^{l_y}}\ in\ B'(r^{l_r})
\end{array}
$$

Suppose that the marker environment is such that $me_I(c_{AS}) = \chi_{c1}$, $me_I(c_{SB}) = \chi_{c2}$, and $me_I(c_{AB}) = \chi_{c3}$. The following triple (ρ, κ, ζ) is a solution for P:

$$
\rho(\beta) = \begin{cases} \{enc\{\chi_{k3}\}_{\chi_{k1}}\} & \text{if } \beta = \beta_x \\ \{\chi_{k3}\} & \text{if } \beta = \beta_s, \beta_y \\ \{enc\{\chi_{k3}\}_{\chi_{k2}}\} & \text{if } \beta = \beta_w \\ \{enc\{\chi_M\}_{\chi_{k3}}\} & \text{if } \beta = \beta_z \\ \{\chi_M\} & \text{if } \beta = \beta_r \end{cases}
\qquad
\kappa(\chi) = \begin{cases} \{enc\{\chi_{k3}\}_{\chi_{k1}}\} & \text{if } \chi = \chi_{c1} \\ \{enc\{\chi_{k3}\}_{\chi_{k2}}\} & \text{if } \chi = \chi_{c2} \\ \{enc\{\chi_M\}_{\chi_{k3}}\} & \text{if } \chi = \chi_{c3} \\ \emptyset & \text{otherwise} \end{cases}
$$

and for $i = 1, 2, 3 : \zeta(l_{ci}) = \{\chi_{ci}\}$, $\zeta(l_{ki}) = \{\chi_{ki}\}$, $\zeta(l_M) = \{\chi_M\}$ and for $bv \in \{x, s, w, y, z, r\} : \zeta(l_{bv}) = \{\chi_{bv}\}$.

If we take $\mathcal{S} = \{\chi_{k1}, \chi_{k2}, \chi_{k3}, \chi_M\}$ and $\mathcal{P} = \{\chi_{c1}, \chi_{c2}, \chi_{c3}\}$ then P is confined, hence the secrecy of M is guaranteed. ■

3.4 Comparison with Abadi's type system

In [1], Abadi studies a different notion of secrecy in the spi-calculus, statically ensured by a type system (see Section 1.5 for a review). Type soundness guarantees that there is no leaking of secret information. As for the disciplined use of channels, Abadi's and our aims are very close, as well as for the assumptions made (except that Abadi has a further notion of *Any*, besides those of *Public* and *Secret*). This section compares Abadi's notion of secrecy with our notion of confinement.

First, we reformulate the assumptions of Abadi in our setting, occasionally making some technical adjustments to our analysis. For simplicity, we will not deal with confounders, as they require deeper changes to our CFA.

1. We assume that both data encrypted with secret keys and messages passed on secret channels have the format $M = (M_1 : \textsc{sec}, M_2 : \textsc{any}, M_3 : \textsc{pub})$.

2. The comparison with the notion of well-formed environment is made straightforward by assuming that all the names and variables are distinct. Hereafter, we then assume that processes contain only terms in this form, obtainable via α-conversions.

3. To deal with type *Any*, we introduce the kind *any*, besides the kind secret *sec* and public *pub*.

4. We also need to simplify the *match* clause, by assuming to validate P in any case (see the clause below). This is because, otherwise, the clause would validate the same processes that Abadi's rule (Level *Match*) validates.

$$\boxed{\begin{array}{llll} (\rho,\kappa,\zeta,\delta) \models_{me} [M^l is N^{l'}]P & \text{iff} & (\rho,\kappa,\zeta,\delta) & \models_{me} M^l \wedge \\ & & (\rho,\kappa,\zeta,\delta) & \models_{me} N^{l'} \wedge \\ & & (\rho,\kappa,\zeta,\delta) & \models_{me} P \end{array}}$$

5. In order to discipline the usage of terms with kind *Any*, as in items $1,\ldots,6$ at the beginning of Section 1.5, we add a further component to solutions. New judgements are on the forms

$$(\rho,\kappa,\zeta,\delta) \models_{me} M^l \text{ and } (\rho,\kappa,\zeta,\delta) \models_{me} P, \text{ where:}$$

$\delta \subseteq Val$ is the *abstract collection* and contains sets of abstract, whose kind should be not of kind *any*.

The old clauses of Tables 3.2 and 3.3 are enriched with a further condition on the new component, asking to collect the abstract values corresponding to sensitive data. Table 3.5 report these additions, only.

6. The Type System gives a type to each term. In our setting, the *type* of a term is essentially its kind. To this aim, we define the operators *SKIND* and *DKIND* in the style of Definitions 3.3.2 and 3.3.3. The kind of a term M^l is forced to be the same of the kind K of its abstract values $\zeta(l)$. In a sense, K is a sort of abstract kind of M^l.

These definitions match type rules for terms. Once had a type for each term, we can construct a type system as Abadi does (see Section 1.5).

7. Finally, Abadi deals with free variables - with type *Any* - whereas we only consider closed processes. Our Control Flow Analysis is not able to associate a set of channels to (the marker of) a free variable x in a process P, unless an explicit *binding* construct is introduced. Nevertheless, for our aim, only the kind of the data associated with x, that should be of type *any*, would be of interest. This fact allows us to treat x as a bound variable. We introduce a special and distinguished binder β_* associated, by definition, with the distinct channel $* \notin \mathcal{C}$. Channels are now $\mathcal{C}^* = \mathcal{C} \cup \{*\}$. For technical convenience, we recall this association, writing the process in which x occurs as $P(x^{\beta_*})$. We also assume that all our marker environments, in particular the initial one, map each free variable x to β^*, with $\rho(\beta^*) = \{*\}$. Therefore, binders are now $\mathcal{B}^* = \mathcal{B} \cup \{\beta^*\}$. Intuitively, $*$ stands for all possible values and will give rise to the kind *any*. In a sense, we are using a form of abstraction with respect to

$(\rho, \kappa, \zeta, \delta)$	$\models_{me} \{M_1^{l_1}, \ldots, M_k^{l_k}\}^l_{N^{l_0}}$	iff	$\ldots \wedge \zeta(l_0) \subseteq \delta$
$(\rho, \kappa, \zeta, \delta)$	$\models_{me} \overline{M^l}\langle N_1^{l_1}, \ldots, N_k^{l_k}\rangle.P$	iff	$\ldots \wedge \zeta(l) \subseteq \delta$
$(\rho, \kappa, \zeta, \delta)$	$\models_{me} M^l(x_1^{\beta_1}, \ldots, x_k^{\beta_k}).P$	iff	$\ldots \wedge \zeta(l) \subseteq \delta$
$(\rho, \kappa, \zeta, \delta)$	$\models_{me} [M^l is N^{l'}]P$	iff	$\ldots \wedge \zeta(l) \cup \zeta(l') \subseteq \delta$
$(\rho, \kappa, \zeta, \delta)$	$\models_{me}$ *let* $(x^{\beta_x}, y^{\beta_y}) = M^l$ *in* P	iff	$\ldots \wedge \textsc{pair}(\zeta(l), \textsc{pair}(\rho(\beta_x), \rho(\beta_y))) \subseteq \delta$
$(\rho, \kappa, \zeta, \delta)$	$\models_{me}$ *case* M^l *of* $0 : P\ suc(x^\beta) : Q$	iff	$\ldots \wedge \textsc{pair}(\zeta(l), \rho(\beta)) \subseteq \delta$
$(\rho, \kappa, \zeta, \delta)$	$\models_{me}$ *case* L^l *of* $\{x_1^{\beta_1}, \ldots, x_k^{\beta_k}\}_{N^{l_0}}$ *in* P	iff	$\ldots \wedge \textsc{pair}(\zeta(l), \textsc{enc}\{\rho(\beta_1), \ldots, \rho(\beta_k)\}_{\zeta(l_0)}) \subseteq \delta$

Table 3.5: Extended CFA for encryption and processes.

the type of free variables, directly inside solutions. We also assume that free variables cannot be used as keys or as subjects of actions, as Abadi does.

The functions *SKIND* and *DKIND* refine the ones of *Skind* and *Dkind*, adding the kind *any* and stating that *sec* and *pub* are subsumed by *any*. Subsumption is made explicit via the least upper bound *lub*: if the components of a term (or of an abstract value) have different kinds, the resulting type is *any* (note that we use here *sec* and *pub*, while the kinds in Section 3.3 are *secret* and *public*).

Definition 3.4.1 *The* static *operator*
$\textit{SKIND} : \widehat{Val} \rightarrow \{sec, pub, any\}$ *is defined as follows:*

- $\textit{SKIND}(\chi) = \begin{cases} sec & \text{if } \chi \in \mathcal{S} \\ pub & \text{otherwise;} \end{cases}$
- $\textit{SKIND}(*) = any;$
- $\textit{SKIND}(0) = pub;$
- $\textit{SKIND}(pair(w, w')) = lub(\textit{SKIND}(w), \textit{SKIND}(w'));$
- $\textit{SKIND}(suc(w)) = \textit{SKIND}(w);$

- $\textit{SKIND}(enc\{w_1, \ldots, w_k\}_{w_0}) =$
  $\begin{cases} \textit{SKIND}(\{w_1, \ldots, w_k\}) & \textit{if } \textit{SKIND}(w_0) = pub \ \wedge \ k \neq 0 \\ pub & \textit{otherwise,} \end{cases}$

  where, by abuse of notation,
  $\textit{SKIND}(W) = lub\{\textit{SKIND}(w_i) \mid w_i \in W\}$.

Definition 3.4.2 *Given a marker environment me, the* dynamic *operator* $\textit{DKIND}$: $me \times \mathcal{E} \to \{sec, pub, any\}$ *is defined as* $\textit{DKIND}(me, E) = \textit{SKIND}(f(me, E))$.

Note that $Skind(w) = public$ if and only if $\textit{SKIND}(w) = pub$, while kind *secret* is partitioned, in a sense, in *sec* and *any*. Similarly, $Dkind(me, E) = public$ if and only if $\textit{DKIND}(me, E) = pub$.

In the following, we will use a mapping Θ that transforms a kind K in one of the corresponding Abadi's type, in the obvious way, i.e.:
$\Theta : \{pub, sec, any\} \to \{\textit{PUB}, \textit{SEC}, \textit{ANY}\}$ with:

- $\Theta(pub) = \textit{PUB}$;
- $\Theta(sec) = \textit{SEC}$;
- $\Theta(any) = \textit{ANY}$.

We now impose a reasonable restriction to our solution: a term M^l and its abstract cache $\zeta(l)$ should have the same kind. Only in this case, we can consider the kind of $\zeta(l)$ as the *abstract* kind of M^l. Moreover, we require that binders are always mapped to a non-empty set, in order to give a kind also to terms with variables inside (note that this last condition may not be enjoyed in special cases by minimal solutions). We call these solutions homogeneous.

Definition 3.4.3 *Given the process P, the solution $(\rho, \kappa, \zeta, \delta)$ is* homogeneous *with respect to me binding all free names and variables of P if*

- $(\rho, \kappa, \zeta, \delta) \models_{me} P$;
- $\forall \beta \in dom(\rho) : \rho(\beta) \neq \emptyset$;
- $\forall M^l$ *occurring in* $P : \textit{DKIND}(me, M^l) = \textit{SKIND}(\zeta(l))$ *such that* $(\rho, \kappa, \zeta, \delta) \models_{me} M^l$, *where the function* $\textit{DKIND}$ *in Def. 3.4.2 is extended to open expressions by stipulating that* $\forall x^l : \textit{DKIND}(me, x) = \textit{SKIND}(\rho(me(x)))$.

Note that we extend $\textit{DKIND}$ to open expressions, in order to give a kind also to expressions with variables inside. The only way to give kind to a variable x is through the analysis, i.e. by defining $\textit{DKIND}(me, x) = \textit{SKIND}(\rho(me(x)))$. This is possible only if the abstract environment of binders is always not empty. Otherwise, $\textit{SKIND}$ could not be always defined.

The extension of $\textit{DKIND}$ to open expressions implies that $\textit{DKIND}(me, x^l) = \textit{SKIND}(\zeta(l)))$ for all variables x. Also if $\lfloor M^l \rfloor = \lfloor M^{l'} \rfloor$, then $\textit{SKIND}(\zeta(l)) = \textit{SKIND}(\zeta(l'))$, i.e. different occurrences of the same expression do have the same kind. This shows that the kind of a expression does *not* depend on its label, but only on the channels bound to its components.

Now, we can state the main theorem of this section. The process P (with type annotations driven by CFA solutions) typechecks, if it is confined via a homogeneous solution (condition (a) below), formats are respected (condition (b) and (c) below) and expressions of kind *any* are used where allowed (condition (d) below). To be more precise, if the above conditions hold for a cfa-annotated process P, there is a corresponding type annotated process (see Section 1.5) that typechecks. Let *cod* represent the function codomain.

Theorem 3.4.4 *Let $P \in \mathcal{P}r^{cfa}$ be a process confined via $(\rho, \kappa, \zeta, \delta)$, with respect to $\mathcal{S}, me_I$, if*

(a) $(\rho, \kappa, \zeta, \delta)$ is homogeneous and me_I is s.t. $me_I(fn(P)) \subseteq \mathcal{P}$ and $me_I(fv(P)) = \{\beta^*\}$, where $fn(P)$ and $fv(P)$ are the initial free names and variables of P;

(b) $\forall enc\{w_1, \ldots, w_k\}_{w_0} \in cod(\rho) \cup cod(\kappa) \cup cod(\zeta)$: if $\textit{SKIND}(w_0) = sec$ then $k = 3$ and

- $\textit{SKIND}(w_1) = sec$;
- $\textit{SKIND}(w_2) = any$;
- $\textit{SKIND}(w_3) = pub$;

(c) $\forall \chi \in \mathcal{S}$, $\forall (w_1, \ldots, w_k) \in \kappa(\chi)$: then $k = 3$ and

- $\textit{SKIND}(w_1) = sec$;
- $\textit{SKIND}(w_2) = any$;
- $\textit{SKIND}(w_3) = pub$;

(d) $\forall w \in \delta : \textit{SKIND}(w) \in \{pub,\ sec\}$.

then there exists E and a type annotated process $a(P) \in \mathcal{P}r^t$, constructed from P, such that

- $E \ \vdash a(P) : ok$,

Proof. Let *env* be the function that maps marker environments *me* in environments E, in the following way:

$$E = env(me) = \{z \mapsto \Theta(\textit{DKIND}(me, z^l)) \mid z \in \mathcal{N} \cup \mathcal{V}\},$$

and, in particular let E_I be the initial environment, depending on me_I:

$$E_I = \{n^l \mapsto \textit{PUB} \mid n \in \mathit{fn}(P)\} \cup \{x^l \mapsto \textit{ANY} \mid x \in \mathit{fv}(P)\},$$

because, by (a), $me_I(\mathit{fn}(P)) \subseteq \mathcal{P}$ and $me_I(\mathit{fn}(P)) = \{\beta^*\}$.
Let $a(P)$ be the type annotated process, constructed from P, as follows

(i) $\forall n^l \in \mathit{fn}(P) : n$ replaces n^l;

(ii) $\forall x^l \in \mathit{fn}(P) : x$ replaces x^l;

(iii) $\forall (\nu n^\chi) \in P : (\nu n : T)$ replaces (νn^χ), where $T = \begin{cases} \textit{SEC} & \text{if } \chi \in \mathcal{S} \\ \textit{PUB} & \text{otherwise;} \end{cases}$

(iv) $\forall x^\beta \in P : x : T$ replaces x^β, where $T = \begin{cases} \textit{SEC} & \text{if } \textit{SKIND}(\rho(\beta)) = \mathit{sec} \\ \textit{PUB} & \text{otherwise;} \end{cases}$

The proof is by joint induction on the structure of processes and terms.

(1) Since all the names and variables in P are distinct, the initial environment E_I and all the other environments $E \supseteq E_I$, constructed upon E_I, are such that $\vdash E : \diamond$, i.e. they are well formed (see also the note in the cases of input and restriction, below).

(2) We prove that $E \vdash M : T$, under the induction hypothesis that $(\rho, \kappa, \zeta, \delta) \models_{me} M^l$ (recall that $E = \mathit{env}(me)$). We will see only two cases: the others are analogously proved.

The case z^l, with $z \in \mathcal{N} \cup \mathcal{V}$. If $z^l \in \mathit{fn}(P) \cup \mathit{fv}(P)$ then trivial, because $E \supseteq E_I$. Otherwise, in E, the association between z and T is made by passing through a binding construct, like an input or a restriction. If $z = n$ and $(\nu n : T)$ appears in $a(P)$, in the place of (νn^χ), then $E \vdash n : T$, where, by (iii), $T = \Theta(K)$, with $K = \textit{DKIND}(me, n^l) = \textit{SKIND}(\chi)$, by (a).
Similarly for $z = x$, by using (iv).

The case $M^l = (M_1^{l_1}, M_2^{l_2})^l$. By induction hypothesis, $\forall i = 1, 2 :$
$E \vdash M_i^{l_i} : \Theta(K_i)$, with $K_i = \textit{DKIND}(me, M_i^{l_i}) = \textit{SKIND}(\zeta(l_i))$, because the solution $(\rho, \kappa, \zeta, \delta)$ is homogeneous. Furthermore
$\mathrm{PAIR}(\zeta(l_1), \zeta(l_2)) \subseteq \zeta(l)$. Again, since $(\rho, \kappa, \zeta, \delta)$ is homogeneous, we have that $\textit{SKIND}(me, M^l) = \textit{SKIND}(\zeta(l)) = \mathit{lub}(K_1, K_2)$, from which the thesis follows.

The case $M = \{M_1^{l_1}, \ldots, M_k^{l_k}\}^l_{M_0^{l_0}}$. As above, by induction hypothesis, $\forall i = 0, \ldots, k : E \vdash M_i : \Theta(K_i)$, with $K_i = \textit{DKIND}(me, M_i^{l_i}) = \textit{SKIND}(\zeta(l_i))$, by$(a)$. Now, by hypothesis, $K_0 \neq \mathit{any}$, because $\zeta(l_0) \subseteq \delta$, consequently we have two cases. If $\textit{DKIND}(me, M_0^{l_0}) = \mathit{pub}$ and then, by definition, $E \vdash M_0 : \textit{PUB}$, we obtain that $K = \textit{DKIND}(me, M^l)$ that is equal to $K = \textit{DKIND}(me, \{M_1^{l_1}, \ldots, M_k^{l_k}\})$. Therefore $E \vdash M : \Theta(K)$, which is the conclusion of the type rule (Level encryption *Public*). Otherwise, from $E \vdash M_0 : \textit{SEC}$, we symmetrically obtain the type rule (Level encryption *Secret*), with the right format, because of condition (b).

(3) We prove that $E \vdash P : ok$, under the induction hypothesis that $(\rho, \kappa, \zeta, \delta) \models_{me} P$ (recall that $E = env(me)$).

The cases 0, $!P'$, $P'|Q$ are trivial by induction hypothesis.

The case $\overline{M^l}\langle N_1^{l_1}, \dots, N_k^{l_k}\rangle.P'$. By induction hypothesis, we have that:

- $E \vdash M : \Theta(K)$;
- $\forall i = 1, \dots, k : E \vdash N_i : \Theta(K_i)$;
- $E \vdash P' : ok$,

with $K = \textit{DKIND}(me, M^l)$ and $K_i = \textit{DKIND}(me, M_i^{l_i})$ for $i = 1, \dots, k$. Moreover the analysis also guarantees that

$\forall \chi \in \zeta(l) : \zeta(l_1) \times \dots \times \zeta(l_k) \subseteq \kappa(\chi) \quad (\bullet)$.

There are only two cases here, depending on K. In fact, condition (d), guarantees that K can only be either *pub* or *sec*.

If $K = pub$, then, by homogeneity, $\textit{SKIND}(\zeta(l)) = pub$ and therefore $\forall \chi \in \zeta(l) : \chi \in \mathcal{P}$. Then, by confinement, $\forall \chi \in \zeta(l) : Skind(\kappa(\chi)) = public$, and consequently $\textit{SKIND}(\kappa(\chi)) = pub$. Therefore, by condition $(\bullet)$, we have that $\textit{SKIND}(\{\zeta(l_1), \dots, \zeta(l_k)\}) = pub$, and, in turn, $\textit{DKIND}(me, \{N_1^{l_1}, \dots, N_k^{l_k}\}) = pub$, leading us to the rule (Level Output *Public*).
Symmetrically for (Level Output *Secret*), with the right format, because of condition (c).

The case $M^l(x_1^{\beta_1}, \dots, x_k^{\beta_k}).P'$. By induction hypothesis, we have that:

- $E \vdash M : \Theta(K)$;
- $E, x_1 : \Theta(K_1), \dots, x_k : \Theta(K_k) \vdash P' : ok$,

with $K = \textit{DKIND}(me, M^l)$ and $K_i = \textit{DKIND}(me, x_i^{\beta_i})$ for all $i = 1, ..k$. Moreover, the following condition holds:

$\forall \chi \in \zeta(l),\ \forall (w_1, \dots, w_k) \in \kappa(\chi),\ \forall i = 1, \dots, k : \{w_i\} \subseteq \rho(\beta_i)$.

Again, we have only two cases here, depending on K.

If $K = pub$, then $\forall i = 1, \dots, k : \textit{SKIND}(w_i) = pub$. Therefore $\textit{SKIND}(\rho(\beta_i))$ is equal to $K_i = pub$, for all $i = 1, \dots, k$, leading us to the rule (Level Input *Public*).
Symmetrically for (Level Input *Secret*), with the right format, because of condition (c).

Note that in the case of input the analysis updates the current marker environment with the associations $[x_i \mapsto \beta_i]$. Consequently, the current environment $E = env(me)$ is updated with the corresponding associations, i.e. with $[x_i \mapsto \Theta(\textit{DKIND}(me, x_i))]$, where $\textit{DKIND}(me, x_i) = \textit{SKIND}(\rho(\beta_i))$.

The case $(\nu n^\chi)P'$. By induction hypothesis we have that: $E, n : T \ \vdash P' : ok$ and therefore $E \ \vdash (\nu n : T)P' : ok$.

Also in the case of restrictions the analysis updates the current marker environment with the association $[n \mapsto \chi]$, Consequently the current environment $E = env(me)$ is updated with the corresponding association i.e. with $[n \mapsto \Theta(me, \textit{DKIND}(n))]$, where $\textit{DKIND}(me, n) = \textit{SKIND}(\chi)$.

The cases $[M^{l_M} \ is \ N^{l_N}]P'$,
$let \ (x^{\beta_x}, y^{\beta_y}) = M^l \ inP'$,
$case \ M^l \ of \ 0 : P' \ suc(x^\beta) : Q$,
$case \ L^l \ of \ \{x_1^{\beta_1}, \ldots, x_k^{\beta_k}\}_{N^{l_N}} \ in \ P'$

are proved in an analogous way, specially using the hypothesis (d) on δ, to check that sensitive data are not of kind *any*.

■

The above theorem shows that our analysis leaves it to a next and separate step to impose demands on when programs are well-formed.

Due to the above theorem, and disregarding confounders, we can reuse the main theorem of [1] to ensure that a confined process never leaks the values of parameters of type *ANY*. More precisely, it says that if only variables of level *ANY* and names of level *PUB* are in the domain of the environment E, if σ and σ' are two substitutions of values for the variables in E, and if P typechecks, then $P\sigma$ and $P\sigma'$ are testing equivalent. An observer cannot distinguish $P\sigma$ and $P\sigma'$ and therefore cannot detect the difference in the values for the variables. In our setting it reads as follows. Note that we assume that each cyphertext is produced only once, for instance using a history-dependent encryption.

Corollary 3.4.5 *If the hypotheses of Theorem 3.4.4 hold, cyphertexts are assumed to be all distinct and if furthermore* $E \vdash \sigma$ *and* $E \vdash \sigma'$*, then* $P\sigma \simeq P\sigma'$*, where* $\simeq$ *is the testing equivalence.*

Finally if a process P typechecks then P is confined, via a solution $(\rho, \kappa, \zeta, \delta)$ homogeneous and that respects kinds and formats.

Usually, channels have a fixed arity, stating the number of messages that can simultaneously flow on it. We assume that processes are made with channels in this form. Nonetheless, to simplify the proof, we further assume that the arity of public channels is fixed to a particular r, while the arity of secret ones is fixed to 3, because of formats.

Let $\mathcal{P}$ be $\{\chi_p\} \cup \{0\}$ and $\mathcal{S}$ be $\{\chi_s\}$. Furthermore let $\widehat{Val^*}$ be $\wp(Val \cup \{*\})$, $\widehat{PVal} = \{w \in \widehat{Val} \mid \textsc{skind}(w) = pub\}$ and $\widehat{SVal} = \{w \in \widehat{Val} \mid \textsc{skind}(w) = sec\}$.

Theorem 3.4.6 *Given a process $P \in \mathcal{P}r^t$ and an environment E_I such that only names of type* PUB *and only variables of type* ANY *are in $dom(E_I)$, if $E_I \vdash P : ok$, then there exist $(\rho, \kappa, \zeta, \delta)$, me and a unique process $b(P) \in \mathcal{P}r^{cfa}$, annotated with markers and labels constructed from P, such that*

(a) $(\rho, \kappa, \zeta, \delta)$ *is a homogeneous solution for* $b(P)$*;*

(b) $\forall enc\{w_1, \ldots, w_k\}_{w_0} \in cod(\rho, \kappa, \zeta)$: *if*

1. $\textsc{skind}(w_0) = sec$ *then* $k = 3$ *and*
   - $\textsc{skind}(w_1) = sec$;
   - $\textsc{skind}(w_2) = any$;
   - $\textsc{skind}(w_3) = pub$;
2. $\forall enc\{w_1, \ldots, w_k\}_{w_0} \in cod(\rho) \cup cod(\kappa) \cup cod(\zeta)$: *if* $\textsc{skind}(w_0) = pub$ *then* $\forall i = 1, \ldots, k$: $\textsc{skind}(w_i) = pub$;

(c)

1. $\forall \chi \in \mathcal{S}$, $\forall (w_1, \ldots, w_k) \in \kappa(\chi)$: *then* $k = 3$ *and*
   - $\textsc{skind}(w_1) = sec$;
   - $\textsc{skind}(w_2) = any$;
   - $\textsc{skind}(w_3) = pub$;
2. $\forall \chi \in \mathcal{P}$, $\forall (w_1, \ldots, w_k) \in \kappa(\chi)$: *then* $k = r$ *and* $\forall i = 1, \ldots, r : \textsc{skind}(w_i) = pub$;

(d) $\forall w \in \delta : \textsc{skind}(w) \in \{pub,\ sec\}$.

Proof. We construct $(\rho, \kappa, \zeta, \delta)$, me and $b(P)$ as follows. We assume to use different binders in $\mathcal{B}$ for different variables and different labels for different terms. We construct below in a single step a marker environment that contains an association for every name and variable of P. This is possible, because we assumed to have all names and variables distinct.

(i) $\forall n \in fn(P)$: n^l replaces n in $b(P)$, where $fn(P)$ are the initial free names of P, and:

- $me(n) = \{\chi_p\}$;
- $\zeta(l) = \{\chi_p\}$.

(ii) $\forall x \in fv(P)$: x^l replaces x in $b(P)$, where $fv(P)$ are the initial free variables of P, and:

- $me(x) = \{\beta^*\}$;

- $\rho(\beta^*) = \{*\}$;
- $\zeta(l) = \{*\}$.

(iii) $\forall\ (\nu n : T)$ occurring in P: (νn^{χ_T}) replaces $(\nu n : T)$ in $b(P)$, with $T = \begin{cases} p & \text{if } T = \textsc{pub} \\ s & \text{if } T = \textsc{sec} \end{cases}$ and:

- $me(n) = \zeta(l) = \{\chi_T\}$.

(iv) $\forall x : T$ occurring in P: x^β replaces $x : T$ in $b(P)$ and:

- $me(x) = \beta$;
- $\rho(\beta) = \begin{cases} \widehat{SVal} & \text{if } T = \textsc{sec} \\ \widehat{Val^*} & \text{if } T = \textsc{any} \\ \widehat{PVal} & \text{if } T = \textsc{pub} \end{cases}$;
- $\zeta(l) = \rho(\beta)$.

(v) $\forall M \notin \mathcal{N} \cup \mathcal{V}$ occurring in P, M^l replaces M in $b(P)$ and $\zeta(l)$ is defined by induction on the shape of M, under the assumption that $\lfloor M^l \rfloor = \lfloor M^{l'} \rfloor$ implies that $\zeta(l) = \zeta(l')$ (recall that all the names and variables are distinct):

- $M^l = (M_1^{l_1}, M_2^{l_2})$ then $\zeta(l) = \textsc{pair}(\zeta(l_1), \zeta(l_2))$;
- $M = 0^l$ then $\zeta(l) = \{0\}$;
- $M^l = suc(M'^{l'})$ then $\zeta(l) = \textsc{suc}(\zeta(l'))$;
- $M^l = \{M_1^{l_1}, \ldots, M_k^{l_k}\}_{M_0^{l_0}}$ then $\zeta(l) = \textsc{enc}\{\zeta_1(l_1), \ldots, \zeta_k(l_k)\}_{\zeta_0(l_0)}$;

(vi) $\kappa(\chi_s) = \widehat{SVal} \times Val^* \times \widehat{PVal}$;

(vii) $\kappa(\chi_p) = \kappa(0) = \widehat{PVal}^r$;

(viii) $\kappa(*) = \emptyset$;

(ix) $\delta = \widehat{SVal} \cup \widehat{PVal}$.

Items $(b), (c), (d)$ hold by construction. Since also homogeneity holds by construction, we are left to prove that $(\rho, \kappa, \zeta, \delta)$ is actually a solution.

We proceed by joint induction on terms and on processes.

(1) We prove that if $E \vdash M : T$ then $(\rho, \kappa, \zeta, \delta) \models_{me} M^l$.

The case n^l. Two cases are possible. If $E \vdash n : \textsc{pub}$, then $me(n) = \{\chi_p\} = \zeta(l)$, by construction: i.e. by item (i) if n is a free name of the initial process P, by item (iii) otherwise.
Similarly if $E \vdash n : \textsc{sec}$ (only by item (iii)).

The case x^l. Three cases are possible. The argument is the same of above, looking to the items (ii) and (iv).

The case $(M_1^{l_1}, M_2^{l_2})^l$. Since, by induction hypothesis, $(\rho,\kappa,\zeta,\delta) \models_{me} M_1^{l_1}$ and $(\rho,\kappa,\zeta,\delta) \models_{me} M_2^{l_2}$ and, by construction $\zeta(l) = \textsc{pair}(\zeta(l_1),\zeta(l_2))$, then the thesis follows.

The case $\{M_1^{l_1},\ldots,M_k^{l_k}\}^l_{M_0^{l_0}}$.

Since, by induction hypothesis, $\forall i = 0,\ldots,k : (\rho,\kappa,\zeta,\delta) \models_{me} M_i^{l_i}$ and, by construction $\zeta(l) = \textsc{enc}\{\zeta(l_1),\ldots,\zeta(l_k\}_{\zeta(l_0)}$, then the thesis follows.

The other cases are analogous.

(2) We prove that if $E \vdash P : ok$ then $(\rho,\kappa,\zeta,\delta) \models_{me} P$.

The cases $\mathbf{0}$, $!P'$, $P'|Q$ are trivial by induction hypothesis.

The case $\overline{M^l}\langle N_1^{l_1},\ldots,N_k^{l_k}\rangle.P'$. By induction hypothesis and by (1) we have that:

- $(\rho,\kappa,\zeta,\delta) \models_{me} M^l$;
- $\forall i = 1,\ldots,k : (\rho,\kappa,\zeta,\delta) \models_{me} N_i^{l_i}$;
- $(\rho,\kappa,\zeta,\delta) \models_{me} P'$,

We are left to prove that

$$\forall \chi \in \zeta(l) : \zeta(l_1) \times \ldots \times \zeta(l_k) \subseteq \kappa(\chi). \quad (\bullet)$$

There are only two cases here, depending on the type of M.

If $E \vdash M : \textit{PUB}$ then $k = r$ and $E \vdash N_i : \textit{PUB}$ for all $i = 1,\ldots,r$ and, consequently, $\textit{DKIND}(me, M^l) = pub$ and $\textit{DKIND}(me, N_i^{l_i}) = \textit{SKIND}(\zeta(l_i)) = pub$. Then $\forall \chi \in \zeta(l) : \chi = \chi_p$ or $\chi = 0$. By construction, $\kappa(\chi_p) = \widehat{PVal}^r$ and $\kappa(0) = \widehat{PVal}^r$, hence $(\bullet)$.
If $E \vdash M : \textit{SEC}$ the proof is analogous (recall that in this case $k = 3$).

The case $M^{l_1}(x_1^{\beta_1},\ldots,x_k^{\beta_k}).P'$.

By induction hypothesis and by (1) we have that:

- $(\rho,\kappa,\zeta,\delta) \models_{me} M^l$;
- $(\rho,\kappa,\zeta,\delta) \models_{me} P'$,

Note that $me[x_1 \mapsto \beta_1,\ldots,x_k \mapsto \beta_k](y) = me(y)$, by construction. We are left to prove that

$$\forall \chi \in \zeta(l),\ \forall (w_1,\ldots,w_k) \in \kappa(\chi),\ \forall i = 1,\ldots,k : \{w_i\} \subseteq \rho(\beta). \quad (\bullet\,\bullet)$$

Again, we have only two cases here, depending on the type of M. If $E \vdash M : \textsc{pub}$ then $k = r$ and $E \vdash x_i : \textsc{pub}$ for all $i = 1, \ldots, r$ and, consequently, $\textsc{dkind}(me, M^l) = pub$ and $\textsc{dkind}(me, x_i^{\beta_i}) = \textsc{skind}(\rho(\beta_i)) = pub$. Then $\forall \chi \in \zeta(l) : \chi = \chi_p$ or $\chi = 0$. By construction, $\kappa(\chi_p) = (\widehat{PVal})^r$ and $\kappa(\chi_p) = \widehat{PVal}^r$. Moreover, $\forall i = 1, \ldots, r : \rho(\beta_i) = \widehat{PVal}$. Therefore ($\bullet\,\bullet$) holds.
If $E \vdash M : \textsc{sec}$ the proof is analogous (recall that in this case $k = 3$).

The other cases are analogous.

■

Example 3.4.7 Consider the simple cryptographic protocol discussed in [7], disregarding formats for brevity. Two processes A and B share the key K_{AB} and A uses it to encrypt a message x that sends to B.

The protocol is composed of the following message:

$$\text{Message 1} \quad A \to B \;:\; \{x\}_{K_{AB}}$$

Its specification is:

$$\begin{array}{rcl} P(x^{\beta_*}) & = & (\nu K_{AB}^{\chi_k})(A(x^{\beta_*})|B) \\ A(x^{\beta_*}) & = & (\overline{c_{AB}^{l_c}}\langle\{x^{\beta_*}\}_{K_{AB}^{l_k}}\rangle \\ B & = & c_{AB}^{l_c}(y^{\beta_y}).case\ y^{l_y}\ of\ \{z^{\beta_z}\}_{K_{AB}^{l_k}}\ in\ \mathbf{0}. \end{array}$$

Let $me_I(c_{AB}) = \chi_c$, $\mathcal{S} = \{\chi_k\}$ and $\mathcal{P} = \{\chi_c\}$. The following (ρ, κ, ζ) is the least solution for $P(x^{\beta_*})$.

$$\rho(\beta) = \begin{cases} \{*\} & \text{if } \beta = \beta_*, \beta_y \\ \{enc\{*\}_{\chi_k}\} & \text{if } \beta = \beta_y \end{cases} \qquad \kappa(\chi) = \begin{cases} \{enc\{*\}_{\chi_k}\} & \text{if } \chi = \chi_c \\ \emptyset & \text{o.w.} \end{cases}$$

$$\zeta(l) = \begin{cases} \{\chi_i\} & \text{if } l = l_i,\ \ i = c, k \\ \{*\} & \text{if } l = l_*, l_z \\ \{enc\{*\}_{\chi_k}\} & \text{if } l = l_y \end{cases} \qquad \delta = \{\chi_x\}.$$

This solution confines $P(x^{\beta_*})$ and matches all the requirements in Theorem 3.4.4 (except for (b) on formats). Hence, the type annotated process $a(P(x^{\beta_*}))$ built from $P(x^{\beta_*})$ and the solution is such that $E\ \vdash a(P(x^{\beta_*})) : ok$.

■

3.5 Authorship

Syntax We show here how our static analysis can be used to guarantee a limited form of message authentication, that we call *authorship*. This property states when a message is only constructed from names generated by a single selected process (i.e. by some names restricted in it), that is therefore authenticated.

Here, we restrict ourselves to (monadic) systems S made of a finite number of processes P_i in parallel, that is a common assumption in the literature on security (e.g. [128]). These systems could be surrounded by some restrictions to account for the variables that are common to all the components. Also, for simplicity, we do not allow replication ! to occur free in any P_i, (we call them *bang-free*); removing this limitation requires a finer analysis than the one presented here. Furthermore, it is convenient to *assume* that distinct binding occurrences of names and variables are labelled with *different markers*. Note that this assumption is preserved by the operational semantics.

By using the structural congruence $\equiv$ it is possible to move all the names $\tilde{p}$ common to more than one of the processes P_i in the system S in front of S itself and the all names $\tilde{n_i}$, belonging to P_i only in front of it. In the following definitions, we assume to have processes always in this form.

Definition 3.5.1 Systems*, denoted by* $S, S', S_1, S_2, \cdots$ *are on the form*

$$S = (\nu\tilde{p}^{\tilde{\chi}})(P_1 \mid P_2 \mid \ \ldots \mid P_n),$$

where each component P_i *is bang-free.*

In order to describe the desired property, a couple of operators will help. The first, called *Schan* (S stands for static), applied to an abstract value w, returns the channels of its components. The other, *Dchan*, applied to a marker environment and an expression E, returns the channels corresponding to the names composing E.

Definition 3.5.2 (*Schan* operator)
The static channel operator $Schan : Val \to \wp(\mathcal{C})$ *is defined as follows.*

- $Schan(\chi) = \{\chi\}$; • $Schan(enc\{w_1, \ldots, w_k\}_{w_0}) = \bigcup_{i=0}^{k} Schan(w_i)$;
- $Schan(pair(w_1, w_2)) = Schan(w_1) \cup Schan(w_2)$;
- $Schan(0) = \emptyset$; • $Schan(suc(w)) = Schan(w)$;

The operator *Dchan* (D stands for dynamic) reuses the definition of *Schan* and the translation function f, introduced in Def. 3.3.3.

Definition 3.5.3 *Given me, the* dynamic channel operator *Dchan maps a closed expression to* $\wp(\mathcal{C})$*, and is defined as* $Dchan(me, E) = Schan(f(me, E))$.

The following auxiliary lemma proves that both the above operators *Schan* and *Dchan* are monotone and that the channels of a term M contain the channels of its components.

Lemma 3.5.4

(a) *Let I and J be two sets. If $I \subseteq J$, then $Schan(I) \subseteq Schan(J)$ and $Dchan(me, I) \subseteq Dchan(me, J)$.*

(b) *If $(\rho, \kappa, \zeta) \models_{me} M^l$, then $\forall \hat{n}^{\hat{l}} \in comp(M^l)$: $Schan(\zeta(l)) \supseteq Schan(\zeta(\hat{l}))$.*

Proof. The first point follows by Def. 3.5.2 and Def. 3.5.3, while the second one follows by straightforward structural induction on M^l. ■

Dynamic notion The following notion describes when a certain term M^l binds a variable x^β in a computation, i.e. during a communication in which M^l is sent by a process and received by another along x^β.

Definition 3.5.5
An expression M^l binds *x^{β_x} in a computation $me \vdash S_0 \xrightarrow{\tau} S_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S_n$ if there exists $me \vdash S_i \xrightarrow{\tau} S_{i+1}$ $(0 \leq i < n)$ deduced with the premises*

$$me' \vdash \overline{m^{l_m}}\langle M^l \rangle.Q \xrightarrow[l]{\overline{m}} (\nu)\langle M^l \rangle Q \quad and \quad me'' \vdash m^{l'_m}(x^{\beta_x}).R \xrightarrow[\beta_x]{m} (x^{\beta_x})R.$$

Note that here M^l contains no variables. Furthermore there is at most one transition binding x^{β_x} to M^l, because we assumed to have all the annotations distinct, and so there exists a unique β_x in S_0, therefore in all S_j $(0 \leq j < n)$.

The dynamic notion for authorship says that P is the author of a certain message M^l received on x^{β_x}, if all its components are associated with channels of P. To be more precise M^l is the author of every single component of M^l.

Definition 3.5.6
Let M^l bind x^{β_x} in a computation $me \vdash S_0 \xrightarrow{\tau} S_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S_n$, then $P_i = (\nu\tilde{n}^{\tilde{\chi}})P'_i$, occurring in the initial system S_0, is the author *of M^l, if and only if $Dchan(me[\tilde{n} \mapsto \tilde{\chi}], M^l) \subseteq \{\tilde{\chi}\}$.*

Static notion The static notion for authorship requires that $\rho(\beta_x)$ is made with channels of P, only.

Definition 3.5.7 *The variable x^{β_x} is* booked for *$P_i = (\nu\tilde{n}^{\tilde{\chi}})P'_i$ occurring in S_0 if and only there exists (ρ, κ, ζ) such that*

- $(\rho, \kappa, \zeta) \models_{me} S_0$;
- $Schan(\rho(\beta_x)) \subseteq \{\tilde{\chi}\}$.

Again, the static notion implies the dynamic one.

Theorem 3.5.8 *If x^{β_x} is booked for $P_i = (\nu \tilde{n}^{\tilde{\chi}})P'_i$, occurring in S_0, then, for all computations $me \vdash S_0 \xrightarrow{\tau} S_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} S_n$ in which an expression M^l binds x^{β_x}, P_i is the author of M^l.*

Proof. Suppose that x^{β_x} is booked for P_i, that M^l binds x^{β_x} because the transition $me \vdash S_i \xrightarrow{\tau} S_{i+1}$ of the computation above has been deduced with the premises

$$me' \vdash \overline{m^{l_m}}\langle M^l\rangle.Q \xrightarrow[l]{\overline{m}} (\nu)\langle M^l\rangle Q \quad \text{and} \quad me'' \vdash m^{l'_m}(x^{\beta_x}).R \xrightarrow[\beta_x]{m} (x^{\beta_x})R,$$

and that, per absurdum, $Dchan(me', comp(M^l)) \not\subseteq \{\tilde{\chi}\}$, i.e. assume that there exists $\hat{n}^{\hat{l}} \in comp(M^l) : Dchan(me', \hat{n}^{\hat{l}}) = \{\hat{\chi}\} \not\subseteq \{\tilde{\chi}\}$, namely $me'(\hat{n}^{\hat{l}}) = \hat{\chi}$. Since x^{β_x} is booked for P_i, $(\rho, \kappa, \zeta) \models_{me} S_0$ and, by Theorem 3.2.15, also $(\rho, \kappa, \zeta) \models_{me} S_i$. To check S_i, the analysis guarantees that $(\rho, \kappa, \zeta) \models_{me'} \overline{m^{l_m}}\langle M^l\rangle.Q$, that implies that

$$\forall \chi \in \zeta(l_m) : \zeta(l) \subseteq \kappa(\chi). \qquad (*)$$

Also, $(\rho, \kappa, \zeta) \models_{me'} M^l$ and $(\rho, \kappa, \zeta) \models_{me'} \hat{n}^{\hat{l}}$. Therefore $\{\hat{\chi}\} \subseteq \zeta(\hat{l})$ and $\hat{\chi} \in Schan(\zeta(\hat{l}))$. By Lemma 3.5.4(b), also $\hat{\chi} \in Schan(\zeta(l))$.

The analysis for S_i also guarantees that $(\rho, \kappa, \zeta) \models_{me''} m^{l'_m}(x^{\beta_x}).R$, that implies

$$\forall \chi \in \zeta(l'_m),\ \forall w \in \kappa(\chi) : \{w\} \subseteq \rho(\beta_x). \qquad (**)$$

Now, for the communication to be deducible, $\lfloor M^{l_m} \rfloor = \lfloor M^{l'_m} \rfloor$, and there exists, by Lemma 3.3.1 me such that $me \supseteq me'$ and $me \supseteq me''$, where m is bound to the same channel, then by Lemma 3.2.7, $\zeta(l_m) \cap \zeta(l_{m'}) = Z \neq \emptyset$. From (*) and (**) one has that $\forall \chi \in Z$: both $\zeta(l) \subseteq \kappa(\chi)$ and $\forall w \in \kappa(\chi) : \{w\} \subseteq \rho(\beta_x)$ and therefore $\zeta(l) \subseteq \rho(\beta_x)$. Furthermore, by Lemma 3.5.4(a), $Schan(\zeta(l)) \subseteq Schan(\rho(\beta_x))$. Since $\hat{n}^{\hat{l}} \in comp(M^l)$, by Lemma 3.5.4, and by hypothesis, we have the following

$$\{\hat{\chi}\} \subseteq Schan(\zeta(\hat{l})) \subseteq Schan(\zeta(l)) \subseteq Schan(\rho(\beta_x)) \subseteq \{\tilde{\chi}\}.$$

Consequently, $\hat{\chi} \in \{\tilde{\chi}\}$, against the hypothesis. ∎

The notion seen before is quite strong. Actually, our analysis is sophisticated enough to check other kinds of authorship. For example, one may have a weaker notion of being booked: a process may be willing to receive messages from more than one process, as considered above, e.g. from a set $\{P_j \mid j \in J\}$. Once changed in the obvious way the dynamic notion, the only change to the static one is in its second item, that becomes $Schan(\rho(\beta_x)) \subseteq \bigcup_{j \in J}\{\tilde{\chi}_j\}$.

Further notions may require to ascertain the author of some selected components of a message, for instance, the author of the key K used to encrypt the message M. The static requirement amounts to:

$$\forall enc\{w_1, \ldots, w_k\}_{w_0} \subseteq \rho(\beta_x) : Schan(w_0) \in \{\tilde{\chi}\}.$$

Note that in this case, we do not make sure that the message was encrypted by the selected process: any other process could do, provided that it received the key, being in this way delegated to use it.

Moreover, one may wish to ascertain who generates the message M encrypted with the key K. In this case we have to introduce a new type of binding, in which the bound variable is inside a *case* L *of* $\{x^\beta\}_N$ *in* P.

Example 3.5.9 We consider again the key-exchange protocol in Example 3.3.9, whose specification we recall below:

$$\begin{array}{rcl}
P & = & (\nu K_{AS}^{\chi_{k1}})(\nu K_{BS}^{\chi_{k2}})(\ (A|B)\ |S) \\
A & = & (\nu K_{AB}^{\chi_{k3}})(\nu M^{\chi_M})(\overline{c_{AS}^{l_{c1}}}\langle\{K_{AB}^{l_{k3}}\}_{K_{AS}^{l_{k1}}}\rangle.\overline{c_{AB}^{l_{c3}}}\langle\{M^{l_M}\}_{K_{AB}^{l_{k3}}}\rangle) \\
S & = & c_{AS}^{l_{c1}}(x^{\beta_x}).case\ x^{l_x}\ \ of\ \ \{s^{\beta_s}\}_{K_{AS}^{l_{k1}}}\ in\ \overline{c_{BS}}\langle\{s^{l_s}\}_{K_{BS}^{l_{k2}}}\rangle \\
B & = & c_{BS}^{l_{c2}}(w^{\beta_w}).case\ w^{l_w}\ \ of\ \ \{y^{\beta_y}\}_{K_{BS}^{l_{k2}}}\ in\ c_{AB}^{l_{c3}}(z^{\beta_z}). \\
 & & case\ z^{l_z}\ \ of\ \ \{r^{\beta_r}\}_{y^{l_y}}\ in\ B'(r^{l_r})
\end{array}$$

It is easy to prove that the variable z^{β_z} is booked for A and therefore that the message $\{M\}_{K_{AB}}$ that B will receive is authentic from A. In fact, $Schan(\rho(\beta_z)) \subseteq \{\tilde{\chi}_A\}$, where $\{\tilde{\chi}_A\} = \{\chi_{k3}, \chi_M\}$.

Similarly, one can also prove at static time that the message K_{AB} has been encrypted with the key K_{BS} of S, when received on w, because $\{enc\{\chi_{K3}\}_{\chi_{K2}}\} \subseteq \rho(\beta_w)$ and $Schan(w_0) \in \{\tilde{\chi}_S\}$, where $\{\tilde{\chi}_S\} = \{\chi_{k1}, \chi_{k2}\}$.

■

We now briefly suggest how to remove the restriction that a process P in a system S contains no replication !. Following the notation used in [19], systems may be written as

$$S = (\nu\tilde{p}^{\tilde{l},\tilde{\chi}})([P_1]^{t_1}\ |\ [P_2]^{t_2}\ |\ \ldots |\ [P_n]^{t_n}),$$

where t_i are tags ($t_i \neq t_j$ for $i \neq j$). Of course $[!P]^t > [P|!P]^t$, so the identity of the processes is never lost. However, one has to redefine portions of the semantics and of the analysis to take care of the enhanced syntax for systems.

Chapter 4

Related Work

Static analysis of programs aims at developing efficient techniques that may be used to obtain approximate answers about how a program executes without actually executing it. Traditionally this has been a main component in the construction of efficient implementations of programming languages. There is a vast literature on static program analysis and we here survey some of the more important approaches [97].

Control Flow Analysis (e.g. [120]) deals with the problem of dynamic dispatch where it is *not* apparent which function (or method) is actually being invoked and where an analysis is needed in order to determine this information. This is a problem that appears all the time in higher-order functional languages and many developments of Control Flow Analysis have indeed been performed for functional languages [120], but also object-oriented languages [102] and languages with concurrency [61] can be addressed.

Control Flow Analysis naturally leads to a general treatment of semantic correctness (in the form of a subject-reduction result) and to the existence of best solutions. On this basis, it recently has been used for validating security and safety issues for concurrent and mobile systems.

Besides our work on π-calculus and spi-calculus, it is worthwhile mentioning the Control Flow Analysis [98] defined for the Mobile Ambient Calculus ([30],[31]), that is a calculus of computation that allows active processes to move between sites; the sites correspond to administrative domains and are modelled using a notion of ambients. In [98] the Control Flow Analysis is developed, adapting the techniques used for the π-calculus and for the spi-calculus. Two accompanying papers, [67, 100], show how to perform a rational reconstruction of the analysis in the framework of Abstract Interpretation as well as how to specify stronger analyses using powerful counting analyses (motivated by state-of-the-art techniques for pointer analysis in imperative programming languages). The analysis is quite similar to ours: the judgements in this case are on the form

$$(I, H) \models^{l}_{me} P$$

where I gives information about what ambients can turn up inside what other ambients and H gives information about the markers associated with classes of ambients; as before me is the marker environment and l designates one of the contexts in which P can appear. The specification of the analysis is somewhat more complex than for the π-calculus because the notion of "context" is dynamically evolving. Again, least solutions always exist and the semantic correctness of solutions is established in the form of a subject-reduction result with respect to an operational semantics. The analysis is then used to validate an aspect of a process being a firewall. One ingredient, established in [30], is that a firewall should allow an agent to enter when it possesses the appropriate passwords. This takes the form of a small probe that is sent out from the firewall, that enters the agent and that then guides the agent inside. Another ingredient, established in [98], is to ensure that the firewall is *protective* in the sense in the sense that it is successful in denying entry to agents not knowing the required passwords. Since checking the absence of explicit occurrences of sensitive passwords in would-be attackers is a simple linear-time scan, this presents a useful approach to screening a system against attackers. However, to establish this property it would seem that one has to consider an infinity of attackers and that therefore automatic techniques would not apply. The analysis is used to show that among the infinitely many attackers it is possible to identify one (called the *hardest attacker*) that is the hardest (*hardest attacker*) to protect against as any other attacker. If an analysis of the firewall operating in parallel with this hardest attacker shows that the attacker cannot enter, then indeed the firewall is protective in all possible execution contexts. Overall, the development yields a polynomial-time procedure for rejecting a class of non-protective firewalls, where the linear-time scan does not suffice for banning illegal access to the internals of the non-protective firewall.

A more widely used alternative for calculi of computation is based on Type Systems: security requirements are seen as static information about the objects of a system [1, 129, 40, 31]. Security classes have been formalized as types and the control of flow is based on type checking. Palsberg and Ørbæk [101] use type system for reasoning about trustworthiness of data in the λ-calculus context, and permit to freely pass from trusted annotations to detrusted ones and vice versa. Heintze and Riecke [70] refine Denning's analysis and study a non-interference property on the SLam Calculus (Secure λ-calculus). Well-typed processes are such that values of low-levels expressions are independent of the values of high-level parameters. Volpano, Smith and Irvine develop a type system to ensure secure information flow in a sequential imperative language [129]; this work was later extended by the first two authors in a concurrent, shared-memory based setting [128]. They also address, in a more general way, security in mobile languages in [126]. Furthermore, in [127], they study, still in a concurrent framework, a probabilistic form of non-interference. Probabilistic properties are also analysed by Sabelfeld and Sands [113], that show how probabilistic security can be formalized by using probabilistic powerdomain semantics. Degrees of security in their information flow model is based on partial equivalence relations (pers).

Some kinds of program analysis incorporate aspects of program dependency, that consequently results interesting to analyse in typed calculi for secure information flow [5], such as SLam or those defined by Volpano and Smith.

Type Systems also allow a clear statement of semantic correctness (sometimes called type soundness) and allow us to study whether or not best solutions exist (in the form of principal types). In the Type Systems' setting, the work closest to our proposal in Section 2.4 is that by Abadi [1], as we have seen in the previous sections, even though our notion is more similar to the one a là Dolev and Yao, as expressed by Abadi in [4].

Secrecy has been also studied and proved by Steve Schneider [116] for CSP [28], by using rank functions.

Another approach is that of Hankin and Malacaria [81], that propose a game semantics-based program analysis that can be applied to security. Hankin and Malacaria are not completely convinced that security can be always treated as type information: if a variable is of type integer, independently from the context it occurs, the security value of a variable is often contextual. The use of information flow, able to consider temporal aspects, and therefore contextual aspects, is proposed instead and generalized to higher order languages.

An alternative strand, based on Abstract Interpretation [36], is used by Venet [124, 125] to analyse processes in a fragment of the π-calculus, with particular attention to communication topology and therefore to the usage of channels. Venet is able to statically determine all the potential links between the agents of a π-calculus process.

These approaches are not so dissimilar as might appear at first sight. In fact, Control Flow Analyses can be expressed in the Constraint Based Formulation used here, in the terminology of Type Systems and using Abstract Interpretation. We refer to [35, 97] and their references for further information about the relationship between these approaches.

Other compile-time techniques have been proposed by He and Gligor [68] to guarantee trusted computations involving channels; by Banâtre, Bryce and Le Métayer [13] to detect violations of secure information flow in sequential non-deterministic programs; and by Myers and Liskov [91] to control information flow in a setting where de-classification of data is also possible.

Other interesting papers in this area are [104, 119, 60, 37, 10, 40, 111]. Particularly relevant are Hennessy and Riely's papers [111, 112] that give a type system for Dπ, a variant of the π-calculus with explicit sites that harbour mobile processes. A well-typed process correctly uses its capabilities and never compromises the integrity of well-behaved or "good" locations. Recently also Cardelli and Gordon [30, 31] propose a type system for the Mobile Ambient calculus ensuring that a well-typed mobile computation cannot cause certain kinds of run-time faults, even when capabilities can be exchanged between processes. In the same setting, jointly with Ghelli, they define mobility types.

The idea of static analysis for security has been followed also in the Java world, for example in the Java Bytecode Verifier [74] and in the techniques of proof-carrying code [92].

More recent works is devoted to study the foundations of secure mobile languages (see e.g. the papers on the spi-calculus [8, 7], on the ambient-calculus [98] , KLAIM [41], Join [6]).

The dynamic point of view has been adopted by a certain number of information flow models [64, 83, 84, 56] (to cite only a few). Here, only the external observable behaviour is the object of the analysis. A classical example is the non-interference model of [64, 65], where the actions of the group of high level users have no effect on what the group of low level users can observe. McCullough [83, 84] extends this definition to cope with non determinism. Focardi and Gorrieri [56] use a process algebras setting, by using SPA (Security Process Algebra), an extension of CCS [85]. Our property of no read-up/no write-down could be considered a variant of the non-interference property, studied by most of the authors mentioned above.

Part II

A Dynamic Approach to Security

Chapter 5

Authentication in the π-calculus

Efficiency considerations suggest implementations which provide each sequential process in the system with its own *local* environment. In the π-calculus [88, 86], this amounts to saying that each process has its own space of private names. Some of these names may be communicated to another process and so they become shared by different local environments.

In this chapter, we give a new structural operational semantics to the π-calculus which considers names as being *localized* to their owners. In other words, each sequential process has its local space of names and a local name manager that generates a fresh name, whenever necessary. We use here the adjective *local* in the sense of Algol-like languages, with no connection with the physical localities where processes run. Therefore, our description is less detailed than those which enable their users to explicitly handle the allocation of processes on processors (see e.g. [58, 30, 71, 10]). Indeed here actual distribution is completely transparent to users and only logical separation of processes, induced by parallel composition, is considered.

In our proposal, when a name is exported, it is equipped with the information needed to point back to the local environment where it has been installed as fresh, in the style of static chains that are used to solve non local references in sequential languages with nested environments [107]. More precisely, while deducing a communication (or an extrusion), the name exported is inductively enriched with the path, called *relative address*, from the receiving process to the one that generated the name (*not* to the sender). The reference to the generator of a name is maintained during all the communications by an operator of *address composition*, which updates the relative addresses of names. Names generated by different environments are thus certainly kept distinct. Note that update of names is unnecessary if every process has a unique identifier, e.g. the Internet addresses of [9]. But this entails having a *centralized, global* authority that assigns these unique identifiers every time a new component is added to the system. We can think of our proposal as implementing a *lazy* addressing, while the one that statically associates a unique identifier with each names can be seen as an *eager* addressing. The eager version could result more efficient at run-time, because it requires no dynamic calculation of addresses during communications. However, this gain is lost and our lazy addressing seems more suited than the eager one when some checks are to be made while names are communicated along channels, e.g. for enforcing security properties, like authentication.

We characterize our address composition algebraically. Our proposal makes it useless the global, therefore inefficient, check that a name involved in a transition captures names already in use. Also, α-conversions are no longer necessary to enforce disjointness of local environments (cf. the semantic definitions of the calculi for mobile processes). In this respect, Ambler's work [11] is quite similar to ours, although it is not easy to equip sequential processes with their local environments as in our proposal. This is because [11] uses nesting of binders à la de Bruijn as basic mechanism for naming, while we use parallel composition.

Our transition system for the π-calculus is more concrete and detailed than the original one, because it is nearer to an implementation. However, the two are strongly related. Our proposal conservatively extends the standard semantics of the π-calculus with local environments.

We also study how to deal with the names of a process, when its code is sent over a network: our theory scales up smoothly to higher order π-calculus [114].

As a consequence, our model can easily be transferred to real programming languages, such as *Facile* [62, 123] or *CML* [87, 110].

A possible benefit of our approach, as we will see also in Chapter 6, concerns security in mobile agent environments. The semantics here introduced is used to study authentication, by exploiting the key feature that the reference to the generator of a name is maintained during all the communications. This is made possible by the operator of *address composition*, a mechanism *not* available to the users. Rather it is used by the abstract machine of the π-calculus only, defined by its semantics. Therefore, the relative address of a message uniquely identifies its originator, even after the message has been maliciously intercepted and forwarded.

As a matter of fact, the semantics itself provides us with a built-in primitive that enables the receiver of a message to ascertain its origin.

The notion of message authentication seen in Chapter 3 relies on the similar idea that a message M is authentic of a process P, only if each component of M has been declared as new in P.

We write the primitive for authentication as $[x \stackrel{@}{=} A]P$, where x is a name and A is a process. Intuitively, the execution of the process P starts only when the check $[x \stackrel{@}{=} A]$ succeeds, and this happens if and only if x has been originated by A. [1] In other words, the interpreter of the calculus (i.e. the semantic rules, *not* the user!) checks whether the local environment to which x belongs is the one of A.

[1] For the sake of readability we write here the operands in a simplified form; in the full notation, the name x is equipped with a relative address, that is checked for equality against the address of A (for details, see below).

A communication protocol that guarantees authentication by construction is now easy to define, by using the plain handshaking communication of the π-calculus and the primitive sketched above. Suppose that a process A sends a (fresh) message M to B along a public channel c. In the π-calculus syntax they have the following form:

$$A = (\nu M)\overline{c}M.A'$$

and

$$B = c(x).[x \stackrel{@}{=} A]B'(x),$$

where the operator (νM) declares M to be a fresh name, different from all the others in the whole system; the input $c(x)$ is that paired in a communication with an output $\overline{c}z$ has the effect of binding x to z within its residual yielding $[z \stackrel{@}{=} A]B'(z)$. If we put A, B and also an intruder E in parallel, $A \mid B \mid E$, the effect is to guarantee that the residual of B is indeed $B'(M)$. Note that the intruder might as well intercept the message M originated by A, but it cannot modify any of its parts, because relative addresses are manipulated by the semantics only. Also, there is no need for c to be a channel private to A and B. This example is worked out in detail in Sect. 5.7 and any intrusion shown to fail.

Our solution assumes that the implementation of the communication primitive has a reliable mechanism to control and manage relative addresses. In some real cases this is possible, e.g., if the network management system filters every access of a user to the network. This may be not the case in many other situations.

In Chapter 6, we consider the spi-calculus, to which the mechanism of handling names locally carries over easily.

5.1 Handling names

In the following, we use the π-calculus introduced in Section 1.2, in the version with the constant definition.

Consider for a while the binary parallel composition as the main operator of the π-calculus, stipulating now that it is neither associative nor commutative. Then, build abstract syntax trees of processes as binary trees whose nodes are $\mid$ operators and whose leaves are the sequential components of the whole process. Call them *trees of (sequential) processes* (see Fig. 5.1). Assume that their left (respectively right) branches denote the left (respectively right) component of parallel compositions, and label their arcs with tag $||_0$ (respectively $||_1$). Therefore, any sequential component of a process is uniquely identified by a string ϑ over $\{||_0, ||_1\}^*$. The string corresponds to a path from the root, the top-level $\mid$ of the whole process, to a leaf. Intuitively, ϑ is the address of the sequential component relative to the root of the binary tree. In other words, the sequential component is reachable via ϑ from the root. We can also consider trees, with an n-adic parallel operator, as well: just take n tags $||_0, ||_1, \ldots, ||_{n-1}$. Indeed, our results are insensitive to the arity of the parallel operator. Through a set of *SOS* rules, we accumulate strings of tags which are used to specify a distributed name manager that handles names locally to sequential processes. In this way, the space of names of a whole process is partitioned into local environments associated with its sequential sub-processes. Of course, a distributed environment rules out the equations that manage restrictions globally, e.g. $(\nu\, x)(P \mid Q) \equiv_P (\nu\, x)P \mid Q$ if $x \notin fn(Q)$ becomes useless.

In a process, bound names may become free through either input actions or extrusions. When a bound name becomes free, an expensive α-conversion may be needed to prevent clashes of free names. When such a clash occurs, a fresh name replaces the one that becomes free (see [88] for a detailed discussion), and this requires an extensive search for names in the whole system.

To avoid global management of names, we have to solve two problems. Names have to be generated *locally* and to be brand-new in that local environment. Furthermore, when a name is exported to other local environments via communications or extrusions, we must guarantee that it there are *no clashes* involving the *free names* around.

First, we introduce a new indexed set of *localized names* (for the sake of simplicity, integers), and we associate a counter with each sequential process. When needed, the first name not in use is taken and the counter is increased. If the firing of a prefix enables new sequential processes, the counter is distributed to all of them. Clearly, this mechanism guarantees that a newly generated name is unique within its generator and does not have clashes with other names therein.

The second problem arises when two different sequential processes, say G and R, have generated two syntactically equal names, say n. However they are semantically distinct. Suppose now that G sends its n to R. It is necessary to distinguish between the two different instances of n. Compare this situation with that of names with limited scopes in Algol-like languages. In the same program it is possible to define the same identifier more than once to denote distinct objects. The mechanism of static chains dynamically solves the problem, keeping track of the local environment in which declarations are made. In our case, in the local environment of R, the name generated by G, will be enriched with the *address* of G *relative* to R, which points back from R to the local environment of G. The relative address can be decomposed into two parts according to the minimal common predecessor P of G and R in the tree. A relative address is a string $\vartheta \bullet \vartheta' \in \{||_0, ||_1\}^* \bullet \{||_0, ||_1\}^*$, where ϑ represents the path from P to R, and ϑ' the path from P to G. Consider Fig. 5.1, and let G be P_3 and R be P_1. The address of P_3 relative to P_1 is $||_0||_1 \bullet ||_1||_1||_0$ (read the path upwards from P_1 to the root and reverse, then downwards to P_3). We will inductively build relative addresses while deducing transitions. We use the transition system of the π-calculus introduced in [48]. It suffices to record the application of inference rules involving the $\mid$ in the label of a deduced transition.

A slightly more complex situation arises when a process receives a name and sends it to another process. The name must arrive at the new receiver with the address of the generator (*not* of the sender) relative to the new receiver. This is done by composing relative addresses. Consider again Fig. 5.1, where P_1 sends to P_2 a name that was generated by P_3 (i.e. with relative address $||_0||_1\bullet||_1||_1||_0$). The rules for communication provide us with the address of P_1 relative to P_2, i.e. $||_1||_0\bullet||_0||_1$. The composition of the two relative addresses, written $||_1||_0\bullet||_0||_1 \star ||_0||_1\bullet||_1||_1||_0$, will result in $||_0\bullet||_1||_0$, where $\star$ is the address composition defined in the next section.

Relative addresses and their distributed management are worthless if each process gets a unique identifier by some centralized authority. We prefer not to have such a global name generator and our proposal makes it useless.

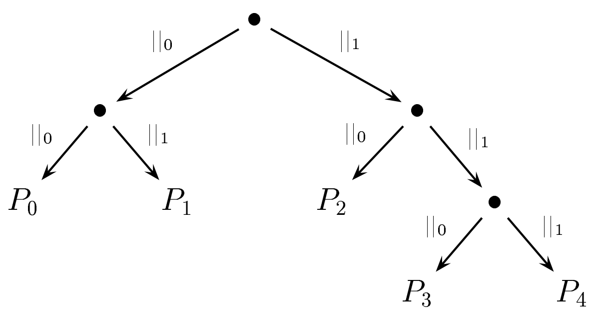

Figure 5.1: The tree of (sequential) processes of $(P_0|P_1)|(P_2|(P_3|P_4))$.

The relative addresses that we use to handle names locally are defined according to the abstract syntax of systems considering the parallel composition as the main operator. Here we only consider binary trees as the logical architecture, and argue that the extension to n-adic trees is immediate. Of course, taking an n-adic parallel operator affects the *SOS* rules. Only slight and technical adjustments are needed to Tab. 5.1, once the rules involving the new parallel operator have replaced the corresponding rules in Tab. 2.1. Also, there is no difficulty in reflecting a different addressing mechanism when a more basic description of the topology of the network is available, as well as the actual placement of processes on processors. It suffices to take the space of relative addresses, quotient the relation induced by the topology and by the allocation. This relation will consider the logical access paths to the processes running on the same processor to be equivalent. Then, the address composition can be left unchanged and the quotient made after the needed calculations.

An example

To make it more intuitive, we instantiate the example above to a simple logical network of a typical educational department (see Fig. 5.2). Through suitable routines, processes P_i can access their own data-bases, which store information about didactic and administrative matters. In our simplified description, we first divide the departmental network into a didactic and an administrative sub-network identified by tags "did" and "adm", respectively. The didactic branch is further decomposed into a server for teachers (P_0, reachable via "teach"), and one for students (P_1, reachable via "stud"). The administrative staff deal with financial issues and didactic matters. The corresponding sub-networks are identified by "financial" (leading to P_2) and "did". Note that the second "did" is different from the first one, because it occurs within the sub-network for an administration identified by "adm". Finally, within the administration, didactic matters are subdivided into information on exams and lectures (stored in nodes P_3 and P_4, respectively). As an example of a relative address, consider did.teach$\bullet$adm.did.exams, which is P_3's address relative to P_0, the server for teachers (we separate branch identifiers by dots ".").

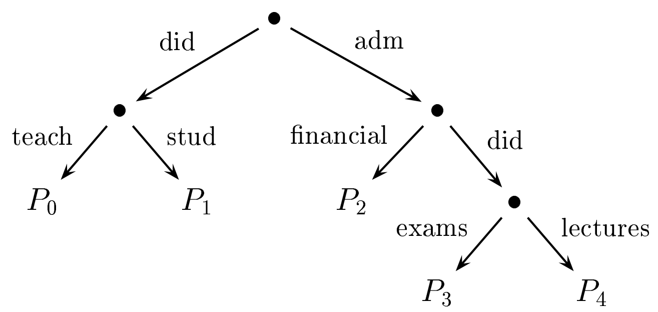

Figure 5.2: The departmental tree of $(P_0|P_1)|(P_2|(P_3|P_4))$.

We consider a simple specification (in polyadic *pi*-calculus) of the flow of information in the departmental network. In Section 5.4 we shall exemplify our way of handling names in the style of Internet, and in Section 5.6 the case when processes can be communicated.

Teachers and students can exchange information along two links, about the programs and the timetable of a course. This piece of information is identified by the global name *prog&timetab*; and likewise for lecture notes and exams dates (*notes&dates*). The teacher's link, *t_chan*, is global. The student's link, $\bullet s\_chan$ (shorthand for $\epsilon\bullet\epsilon\, s\_chan$), has the empty relative address, because it belongs to his/her environment.

A possible specification of the network in the π-calculus follows. Hereafter, names are written in italics, and relative addresses and constants in roman.

$$\begin{aligned}
&\text{Teacher} = t_chan(req, y).\\
&\quad [req = prog\&timetab]\overline{y}\langle \bullet myprog, \text{did.teach}\bullet\text{adm.did.lectures}\, mytimetab\rangle.\text{Teacher}\\
&\quad +[req = notes\&dates]\overline{y}\langle \bullet mynotes, \text{did.teach}\bullet\text{adm.did.exams}\, mydates\rangle.\text{Teacher}\\
&\quad +\text{Work}
\end{aligned}$$

$$\text{Stud} = \overline{t\_chan}\langle prog\&timetab, \bullet s\_chan\rangle.\bullet s\_chan(a, b).\overline{print}\langle a, b\rangle.\text{Stud}$$

$$
\begin{array}{c}
(\mathrm{Teacher}|\mathrm{Stud})|P \\
\downarrow \tau \\
\overline{\mathrm{teach}{\bullet}\mathrm{stud}\mathit{s\_chan}}\langle {\bullet}\mathit{myprog}, \mathrm{did.teach}{\bullet}\mathrm{adm.did.lectures}\,\mathit{mytimetab}\rangle.\mathrm{Teacher}\ | \\
{\bullet}\mathit{s\_chan}(a,b).\overline{\mathit{print}}\langle a,b\rangle.\mathrm{Stud}|P \\
\downarrow \tau \\
\mathrm{Teacher}\ |\ \overline{\mathit{print}}\langle \mathrm{stud}{\bullet}\mathrm{teach}\,\mathit{myprog}, \mathrm{did.stud}{\bullet}\mathrm{adm.did.lectures}\,\mathit{mytimetab}\rangle.\mathrm{Stud}|P \\
\downarrow \overline{\mathit{print}}\langle \mathrm{stud}{\bullet}\mathrm{teach}\,\mathit{myprog}, \mathrm{did.stud}{\bullet}\mathrm{adm.did.lectures}\,\mathit{mytimetab}\rangle \\
(\mathrm{Teacher}|\mathrm{Stud})|P
\end{array}
$$

Figure 5.3: A computation of (Teacher|Stud)$|P$, where $P = (P_2|(P_3|P_4))$.

$$
\begin{aligned}
&+\overline{\mathit{t\_chan}}\langle \mathit{notes\&dates}, {\bullet}\mathit{s\_chan}\rangle.{\bullet}\mathit{s\_chan}(a,b).\overline{\mathit{print}}\langle a,b\rangle.\mathrm{Stud} \\
&+\mathrm{Study}.
\end{aligned}
$$

A teacher either works (Work) or waits for a request on channel t_chan from a student who also communicates the link on which he/she is waiting for the answer ($t\_chan(req, y)$ is an input of two arguments. Then, the teacher answers driven by a matching, and resumes ($\overline{y}\langle z, w\rangle$ is a two-place output). The pointer to the timetable of lectures, *mytimetab*, is prefixed with the address of P_4 relative to P_0. The student waits for the information requested and then prints it.

We see now two interactions between a teacher and a student, followed by a print action are depicted in Fig. 5.3.

In the first communication the student asks the teacher for the course program and timetable. The effect is to bind in Teacher the placeholder y to the link s_chan defined at the address teach$\bullet$stud, and the name req to $prog\&timetab$ (recall that the latter is global). This enables the first output, because the matching $[req = prog\&timetab]$ succeeds. The teacher then uses the student's link, known as ${\bullet}s\_chan$ in P_1, and teach$\bullet$stud s_chan in P_0, to output his/her private name $\bullet\, myprog$ and the name *mytimetab* at address did.teach$\bullet$adm.did.lectures. While communicating the two names, the address composition transforms these relative addresses to stud$\bullet$teach and did.stud$\bullet$adm.did.lectures, respectively. The new addresses are then used to print the information required in the last step of the computation.

The similarity of relative addresses with static chains used in Algol-like languages can now be better explained. A relative address qualifying a name (e.g. stud$\bullet$teach) is a pointer to the local environment where that name (e.g. $\bullet\, myprog$) is defined. Our operational semantics takes care of installing new names when necessary and of updating relative addresses when names are made available as non-local references, via our address composition $\star$.

As an example of how the generation of new names looks like, consider the case in

$$
\begin{array}{c}
(\mathrm{Teacher}|\mathrm{Stud})|P \\
\downarrow \tau \\
((\nu\, thehw)\overline{\mathrm{teach}{\bullet}\mathrm{stud}s\_chan}\langle thehw, {\bullet}dueto\rangle.\mathrm{Teacher} \mid s\_chan(a,b).\overline{do}\langle a,b\rangle.\mathrm{Stud})|P \\
\downarrow \tau \\
(\nu\, \mathrm{teach}{\bullet}hw1)(\mathrm{Teacher} \mid \overline{do}\langle \mathrm{stud}{\bullet}\mathrm{teach}\, hw1, \mathrm{stud}{\bullet}\mathrm{teach}\, dueto\rangle.\mathrm{Stud})|P \\
\downarrow \overline{do}(\mathrm{stud}{\bullet}\mathrm{teach}\, hw1, \mathrm{stud}{\bullet}\mathrm{teach}\, dueto) \\
(\mathrm{Teacher}|\mathrm{Stud})|P \\
\downarrow \tau \\
((\nu\, thehw)\overline{\mathrm{teach}{\bullet}\mathrm{stud}s\_chan}\langle thehw, {\bullet}dueto\rangle.\mathrm{Teacher} \mid s\_chan(a,b).\overline{do}\langle a,b\rangle.\mathrm{Stud})|P \\
\downarrow \tau \\
(\nu\, \mathrm{teach}{\bullet}hw2)(\mathrm{Teacher} \mid \overline{do}\langle \mathrm{stud}{\bullet}\mathrm{teach}\, hw2, \mathrm{stud}{\bullet}\mathrm{teach}\, dueto\rangle.\mathrm{Stud})|P \\
\downarrow \overline{do}(\mathrm{stud}{\bullet}\mathrm{teach}\, hw2, \mathrm{stud}{\bullet}\mathrm{teach}\, dueto) \\
(\mathrm{Teacher}|\mathrm{Stud})|P
\end{array}
$$

Figure 5.4: A computation of (Teacher|Stud)$|P$, where $P = (P_2|(P_3|P_4))$.

which the student asks for the assignment of two homeworks, on the global channel t_chan. Stud is extended with the summand

$$\overline{t\_chan}\langle hw, {\bullet}s\_chan\rangle.{\bullet}s\_chan(a,b).\overline{do}\langle a,b\rangle.\mathrm{Stud}$$

and Teacher with the following summand, where $thehw$ stands for the actual homework to be presented within deadline ${\bullet}dueto$.

$$[req = hw](\nu\ \ thehw)\overline{y}\langle thehw, {\bullet}dueto\rangle.\mathrm{Teacher}.$$

A computation involving the new summands is in Fig. 5.4. First the student asks for an homework. Then the teacher generates a new homework ${\bullet}hw1$ (counters are initialized to 1) and sends it to the student. Note that the bound name $thehw$ has been replaced by the actual name ${\bullet}hw1$, local to the teacher's environment. The application of the rule *Close* exports ${\bullet}hw1$ to Stud and makes it local to (Teacher|Stud). The student solves the exercise via the *do* operation, within stud${\bullet}$teach *dueto*. The next interaction forces the generation of another homework ${\bullet}hw2$, where 2 is the actual value of the counter, increased by the previous extrusion. The usage and management of counters is formalized in Section 5.3.

5.2 Address composition

Here we introduce relative addresses and show how these are updated when names are exported.

Definition 5.2.1 (relative addresses) *Let $\vartheta, \vartheta', \vartheta_i, \ldots \in \{||_0, ||_1\}^*$, and let ϵ be the empty string. Then, the set of* relative addresses *is*

$$\mathcal{A} = \{\vartheta_0 \bullet \vartheta_1 \mid \vartheta_i = ||_j \vartheta'_i \ \textit{ implies } \vartheta_{i\oplus 1} = ||_{j\oplus 1}\vartheta'_{i\oplus 1}, i, j = 0, 1\}$$

where $\oplus$ is sum modulo 2. We will sometimes omit ϵ in relative addresses.

As we said in the previous section, we use relative addresses to encode paths between pairs of nodes of binary trees of processes, like the one in Fig. 5.1. Note that the condition $||_0\vartheta'_0 \bullet ||_1\vartheta'_1$ (and $||_1\vartheta'_0 \bullet ||_0\vartheta'_1$) makes it explicit that the two components of the relative address describe the two distinct paths going out from the *same* node in a binary tree. This node is the minimal common predecessor of the generator of a name and its user. Also $\epsilon \bullet \vartheta$, $\vartheta \bullet \epsilon$ and $\epsilon \bullet \epsilon$ are relative addresses, meaning that one of the two nodes is a predecessor of the other or that they coincide. Actually, there are only three relative positions of the minimal common predecessors of the possible pairs of the processes that generate (G), receive (R) and send (S) a name in an abstract syntax tree, up to symmetries. These three cases are depicted in Fig. 5.5. Note that degenerative cases are obtained when some nodes coincide, and symmetries do not alter relative addresses. As a matter of fact, relative addresses determine minimal paths between nodes, as stated below, where ϑ^R means reversing the oriented path ϑ.

Proposition 5.2.2 *Given two nodes P and P' in a tree, if $\vartheta \bullet \vartheta'$ is the address of P relative to P', then $\vartheta^R\vartheta'$ is the minimal path from P' to P.*
Proof. By definition of relative addresses, ϑ and ϑ' have no part in common. Furthermore, no cycles are possible in a tree. ■

We now define address composition. It is partial, but we will make sure later that it is defined whenever used (see Theorem 5.3.7). Again, Fig. 5.5 depicts the cases in which we will use the address composition $\star$, that discards the dashed paths.

Definition 5.2.3 (address composition) *Address composition $\star : (\mathcal{A} \times \mathcal{A}) \rightharpoonup \mathcal{A}$ is defined by the following three exhaustive cases:*

1. $\vartheta_0 \bullet \vartheta \star \vartheta_2\vartheta \bullet \vartheta_3 = \vartheta_2\vartheta_0 \bullet \vartheta_3$ *with* $\vartheta_2 \neq \epsilon$
2. $\vartheta_0 \bullet \vartheta_1\vartheta \star \vartheta \bullet \vartheta_3 = \vartheta_0 \bullet \vartheta_1\vartheta_3$ *with* $\vartheta_1 \neq \epsilon$
3. $\vartheta'\vartheta_0 \bullet \vartheta \star \vartheta \bullet \vartheta'\vartheta_3 = \vartheta_0 \bullet \vartheta_3$

Below we prove a few properties of $\star$ which will be useful later. They state that $\langle \mathcal{A} \times \mathcal{A}, \star \rangle$ would be a group, if $\star$ were total. This is quite a natural property of address compositions: given a space of addresses and an interconnection topology, the address composition must always connect two sites in both ways, provided that there is a path between them. The first proposition says that $\star$ has a neutral element and an inverse on $\mathcal{A} \times \mathcal{A}$. Its proof is immediate from Def. 5.2.3.

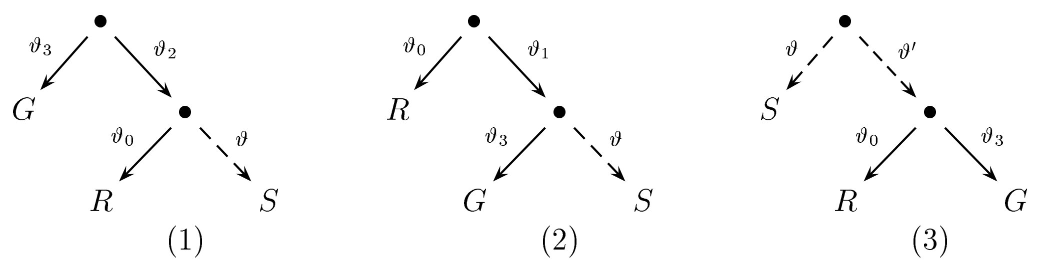

Figure 5.5: The three possible placements of the generator (G), the sender (S), and the receiver (R) of a name.

Proposition 5.2.4 $\forall\, \vartheta_i{\bullet}\vartheta_j \in \mathcal{A}$ *we have that*

1. $\epsilon{\bullet}\epsilon \star \vartheta_i{\bullet}\vartheta_j = \vartheta_i{\bullet}\vartheta_j \star \epsilon{\bullet}\epsilon = \vartheta_i{\bullet}\vartheta_j$, *i.e.* $\epsilon{\bullet}\epsilon$ *is the* neutral element *of* $\star$*;*

2. $\vartheta_j{\bullet}\vartheta_i \star \vartheta_i{\bullet}\vartheta_j = \epsilon{\bullet}\epsilon$, *i.e. the* inverse *of* $\vartheta_i{\bullet}\vartheta_j$ *is equal to* $\vartheta_j{\bullet}\vartheta_i$

Note that $\epsilon{\bullet}\epsilon$ is the *unique* neutral element of $\star$. We prove that $\star$ is also associative.

Proposition 5.2.5 *Whenever defined,* $\star$ *is associative, i.e.*

$$(\vartheta_0{\bullet}\vartheta_1 \star \vartheta_2{\bullet}\vartheta_3) \star \vartheta_4{\bullet}\vartheta_5 = \vartheta_0{\bullet}\vartheta_1 \star (\vartheta_2{\bullet}\vartheta_3 \star \vartheta_4{\bullet}\vartheta_5).$$

Proof. The proof is by case analysis according to the items in Def. 5.2.3. Three cases are possible: *i)* $\vartheta_2 = \vartheta\vartheta_1$, *ii)* $\vartheta_1 = \vartheta\vartheta_2$, and *iii)* $\vartheta_2 = \vartheta_1$.
Consider case *i*). We have $(\vartheta_0{\bullet}\vartheta_1 \star \vartheta\vartheta_1{\bullet}\vartheta_3) \star \vartheta_4{\bullet}\vartheta_5 = \vartheta\vartheta_0{\bullet}\vartheta_3 \star \vartheta_4{\bullet}\vartheta_5$. We now need some hypotheses on ϑ_3 and ϑ_4. We still have three cases: *i'*) $\vartheta_4 = \vartheta'\vartheta_3$, *ii'*) $\vartheta_3 = \vartheta'\vartheta_4$, both with $\vartheta' \neq \epsilon$ and *iii'*) $\vartheta_4 = \vartheta_3$.
Consider case *ii'*), in which, $\vartheta\vartheta_0{\bullet}\vartheta'\vartheta_4 \star \vartheta_4{\bullet}\vartheta_5 = \vartheta\vartheta_0{\bullet}\vartheta'\vartheta_5$. We now prove that the right hand side of the equation in the statement of the theorem yields the same result.

$$\vartheta_0{\bullet}\vartheta_1 \star (\vartheta\vartheta_1{\bullet}\vartheta'\vartheta_4 \star \vartheta_4{\bullet}\vartheta_5) = \vartheta_0{\bullet}\vartheta_1 \star \vartheta\vartheta_1{\bullet}\vartheta'\vartheta_5 = \vartheta\vartheta_0{\bullet}\vartheta'\vartheta_5$$

The other combinations of hypotheses are similar and require only mechanical applications of the definition of $\star$. ∎

The preceding properties ensure that the identity of names is not lost during exportation (a name always encodes a pointer to its generator). More formally, sending a name from a process S to R and back is an involution, i.e. if $\vartheta_0{\bullet}\vartheta_1 \star \vartheta_2{\bullet}\vartheta_3$ is defined

$$\vartheta_1{\bullet}\vartheta_0 \star (\vartheta_0{\bullet}\vartheta_1 \star \vartheta_2{\bullet}\vartheta_3) = \vartheta_2{\bullet}\vartheta_3.$$

There is a categorical interpretation of the structure seen above (for terminology and definitions, see [80]). In fact every binary tree can be made into a category, where the objects are the nodes and the arrows are the paths in the tree (in our case the minimal ones). In this way the correctness of the address composition $\star$ can be immediately shown.

Proposition 5.2.6 *A binary tree T gives rise to the category* **Bintree**. *Its* objects *are the nodes of T and for all nodes P, P' there is a (single)* morphism *from P to P' defined as the address of P relative to P'. The morphism* composition *is $\star$; the (unique)* identity *morphism is $\epsilon \bullet \epsilon$.*
Furthermore, **Bintree** *is a groupoid and a total preorder.*

Proof. By Def. 5.2.1 and Proposition 5.2.2, relative addresses encode the minimal path between two nodes, that is unique in a tree. Hence there is a single morphism between any pair of nodes P, P' (below we write it as $P \leq P'$). The composition is associative, by Proposition 5.2.5. By Proposition 5.2.4.1, for every object P, $\epsilon \bullet \epsilon$ is the unique identity function.
Furthermore **Bintree** is a groupoid, because, by Proposition 5.2.4.2, every morphism is an isomorphism, having only one inverse. Finally, **Bintree** is a total preorder, because, for every P, $P \leq P$ is the identity and if $P \leq P'$ and $P' \leq P''$ then $P \leq P''$, by definition of $\star$. ■

Proposition 5.2.6 enables us to prove the following theorem. It says that, when a node S sends a name generated in G to a node R, address composition $\star$ correctly computes the address of R relative to G. In fact, it computes the minimal path between the two.

Theorem 5.2.7 (correctness of $\star$) *Let G be the generator of a name, and S be the process which receives the name and sends it to the process R. Moreover, let $\vartheta_S \bullet \vartheta_G$ and $\vartheta_R \bullet \vartheta'_S$ be the addresses of G relative to S, and of S relative to R, respectively. Then $\vartheta_R \bullet \vartheta'_S \star \vartheta_S \bullet \vartheta_G$ is the address of G relative to R.*

5.3 Operational semantics

We define a late operational semantics of the π-calculus, based on the late semantics introduced in Section 1.2 (with constant definition as recursive construct), which creates and handles names locally, according to the discussion in the previous sections. This semantics relies also on some features of the proved π-calculus (see Section 1.3).

Our results can be carried over the early version, by adapting the ideas of [33]. A few auxiliary definitions and results are helpful. We start with the set of localized names, or simply names. Localized names are prefixed by relative addresses pointing back to their generators, and they are built upon the abstract syntax of processes.

We implicitly assume a maximal degree of distribution: each sequential process is associated with its virtual processor, uniquely identified by nesting of parallel composition. The actual placement of many virtual processors on a physical one does not affect our management of names. Also, it occurs at a different level of abstraction than actual allocation of processes.

Definition 5.3.1 *Let $\mathcal{N}' = \mathcal{A} \cdot \mathbf{Z} \cup \mathcal{N} \cup \{\tau\}$ be the set of* localized names*, ranged over by $r, s, u, \ldots$, possibly indexed, where $\mathbf{Z}$ is the set of non-zero integer numbers, and "$\cdot$" is the operator of language concatenation.*
The new syntax of the π-calculus is obtained from the one in Def. 1.2.1, with r, s in place of x, y. In particular, π can either be $r(s)$ or $\overline{r}s$ or τ and a restricted name can be s.
Furthermore, we only assume on the new processes the least congruence induced by the monoidal laws for $+$, and the laws for matching and constant invocation.

The new processes with localized names differ from those in $\mathcal{P}$, because they are *not* α-convertible, and consequently variants of transitions involving them are *banned*. Also, parallel composition is *neither* associative, *nor* commutative and *no* congruence rule for restriction holds any longer. We still keep standard names in $\mathcal{N}$ and we do not prefix them with relative addresses. Of course, an initial pass could make them localized to the root of the tree of processes. We wish to avoid this simple pre-processing, because it is global.

We now start considering the problem of sending names. This requires the composition via $\star$ of the address of the name itself with the address of the sender relative to the receiver, as intuitively discussed in Section 5.1. First, we lift composition of addresses to exportation of names. Hereafter we write $\vartheta_0 \bullet \vartheta_1 \star \vartheta_2 \bullet \vartheta_3$ only if the composition is defined.

Definition 5.3.2 *Let $r = \vartheta_r \bullet \vartheta'_r n$ (resp. a) be a name. A name r exported to the relative address $\vartheta \bullet \vartheta'$ is $\vartheta \bullet \vartheta' \star r = (\vartheta \bullet \vartheta' \star \vartheta_r \bullet \vartheta'_r)n$ (resp. $\vartheta \bullet \vartheta' \star a = a$).*

Note that $\vartheta \bullet \vartheta'$ is the address of the sender relative to the receiver. Recall that names also encode a pointer to their generator (above, $\vartheta_r \bullet \vartheta'_r$), unless they belong to $\mathcal{N} \cup \{\tau\}$. This definition makes it explicit that two localized names, syntactically different, that occur in different sequential processes, can denote the *same* name. For instance, both $\bullet s\_chan$ in P_1 and teach$\bullet$studs_chan in P_0 denote the same link between Stud and Teach in the example of Section 5.1 (or rather, using the relative addresses in $\mathcal{A}$, $\bullet s\_chan$ is the same as $||_0 \bullet ||_1 s\_chan$).

The object, subject, names (free and bound) of an action (still ranged over by μ) are as usual. Below, we extend the needed notions to processes with localized names.

Definition 5.3.3 *The* free localized names *of a process are defined by induction in the standard way, except for*

$$fn(P_0|P_1) = \{\bullet||_0 \star fn(P_0)\} \cup \{\bullet||_1 \star fn(P_1)\},$$

where the operation $\star$ applies elementwise to its second argument.
Similarly for the bound localized names *of P, $bn(P)$.*
Finally, $n(P) = fn(P) \cup bn(P)$.

Intuitively, $\bullet||_i \star r$ translates a name r from process P_i into $P_0|P_1$, i.e. one level upwards in the tree of processes. To show how this happens, consider

$$fn(\bullet 1(b).(\overline{a}||_0\bullet 1.\overline{b}\bullet 1 \mid a(x).\overline{||_1\bullet 1}x)) =$$

$$\{\bullet 1\} \cup (\{\bullet||_0 \star fn(\overline{a}||_0\bullet 1.\overline{b}\bullet 1)\} \cup \{\bullet||_1 \star fn(a(x).\overline{||_1\bullet 1}x)\}) \setminus \{b\} =$$
$$\{\bullet 1\} \cup (\{a, \bullet 1, b, \bullet||_0 1\} \cup \{a, \bullet 1\}) \setminus \{b\} = \{\bullet 1, a, \bullet||_0 1\}.$$

The last step but one, determines how the name $||_0\bullet 1$ in $P_0 = \overline{a}||_0\bullet 1.\overline{b}\bullet 1$ is known by the whole process. Since P_0 is reachable from the root of the whole tree via $||_0$, we export $||_0\bullet 1$ to $\bullet||_0$. Similarly for $\bullet 1$ in P_0, known as $\bullet||_0 1$ at the root. Symmetrically for the names in the other parallel component.

To generate new names locally, we enrich processes with a counter, implemented as a family of operators $n \Rightarrow$ (with $n \geq 1$). The intuitive meaning of $n \Rightarrow P$ is that P has generated $n-1$ new names and the next one will be n. Our description of the name manager uses a family of operators $n \Rightarrow$ which extends the syntax of the language. These operators are quite similar to those used to define the causal [38] and locational semantics of CCS [27] and of the π-calculus [115, 25]. We can thus use the technique presented in [25] to encode our name handler into the pure π-calculus.

Definition 5.3.4 *Let P be a process and n be a non-zero natural number. The* extended processes *(denoted by $t, t_i, t' \ldots \in \mathcal{T}$), are defined according to the syntax*

$$t ::= n \Rightarrow P \mid \ (\nu s)t \ \mid \ t|t$$

where P is as in Def. 5.3.1. We assume on $\mathcal{T}$ the least congruence $\equiv$ which satisfies the following clauses

- $n \Rightarrow (\nu\, s)P \equiv (\nu\, s)n \Rightarrow P$ *and*
- $n \Rightarrow (P|Q) \equiv (n \Rightarrow P)|(n \Rightarrow Q)$.

The computations of a process $P \in \mathcal{P}$ start from the extended process $1 \Rightarrow P$.

The standard notions on names (free, bound) are carried over extended processes in the obvious way.[2] Note in passing that the operators $n \Rightarrow$ do *not* distribute neither over prefix nor matching nor $+$.

To exemplify the use of operators $n \Rightarrow$ and of the congruence $\equiv$, suppose to have a computation from the extended process $1 \Rightarrow R$ to the extended process $2 \Rightarrow \overline{a}b.(P|Q)$. Since the counter is 2, one new name has been generated. After the step $\overline{a}b$, one reaches the extended process $2 \Rightarrow (P|Q)$ congruent to $2 \Rightarrow P \mid 2 \Rightarrow Q$. When Q evolves to Q' by generating a new name, one gets the extended process $2 \Rightarrow P \mid 3 \Rightarrow Q'$ (see also the computation in Fig. 5.6).

[2] More precisely, $n(t) = n(FC(t))$, where FC is introduced in Def. 5.4.1.

We extend the standard substitution of one name for another, in the style of [33]. By abuse of notation we write $-\vartheta{\bullet}\vartheta' n$ for $\vartheta{\bullet}\vartheta' - n$ in the following items 2 and 7.

Definition 5.3.5 *Let* $\{-/-\}$ *be the standard substitution. Then the* routed substitution $\{\!|\, - \,/\, - \,|\!\}$ *is defined by induction as follows:*

1. $(n \Rightarrow P)\{\!|u'/u|\!\} = n \Rightarrow P\{\!|u'/u|\!\}$

2. $((\nu\ s)t)\{\!|u'/u|\!\} = \begin{cases} (\nu\ -u)(t\{\!|-u/u'|\!\})\{\!|u'/u|\!\} & \text{if } u' = s \\ (\nu\ s)t & \text{if } u = s \\ (\nu\ s)(t\{\!|u'/u|\!\}) & \text{otherwise} \end{cases}$

3. $(t_0|t_1)\{\!|u'/u|\!\} = t_0\{\!|u'_0/u_0|\!\}|t_1\{\!|u'_1/u_1|\!\}$, *where* $u'_i = ||_{i}{\bullet} \star u'$ *and* $u_i = ||_{i}{\bullet} \star u$, *for* $i = 0, 1$

4. $\mathbf{0}\{\!|u'/u|\!\} = \mathbf{0}$

5. $(\tau.P)\{\!|u'/u|\!\} = \tau.P\{\!|u'/u|\!\}$

6. $(\overline{r}s.P)\{\!|u'/u|\!\} = (\overline{r}s)\{u'/u\}.P\{\!|u'/u|\!\}$

7. $(r(s).P)\{\!|u'/u|\!\} = \begin{cases} r\{u'/u\}(-u).P\{\!|-u/u'|\!\}\{\!|u'/u|\!\} & \text{if } u' = s \\ r\{u'/u\}(s).P & \text{if } u = s \\ r\{u'/u\}(s).(P\{\!|u'/u|\!\}) & \text{otherwise} \end{cases}$

8. $(P + Q)\{\!|u'/u|\!\} = P\{\!|u'/u|\!\} + Q\{\!|u'/u|\!\}$

9. $[r = s]P\{\!|u'/u|\!\} = [r\{u'/u\} = s\{u'/u\}]P\{\!|u'/u|\!\}$

10. $A(r_1, \ldots, r_k)\{\!|u'/u|\!\} = A(r_1\{u'/u\}, \ldots, r_k\{u'/u\})$

For the sake of brevity, we consider only the cases $(\nu s)t$ and $t_0|t_1$ in items 2 and 3 above, although one needs also the cases $(\nu s)P$ and $P_0|P_1$. Their definitions are just the same, so we omit them. Essentially, the routed substitution distributes over the counters $n\Rightarrow$ and also over the operators of processes, and then acts as the standard one. While descending within the components of a parallel composition, the routed substitution inductively updates both the localized names u and u'. This amounts to pushing these names one level downwards in the tree of processes (in Def. 5.3.3 they were lifted one level upwards, instead). Two crucial items are 2 and 7, when $u' = s$. We will prove that the substitutions $\{\!|-u/u'|\!\}$ actually are local α-conversions, because the negative name $-u$ is fresh in t and in P (see Proposition 5.3.8). In passing, negative names occur only bound in extended processes. The α-conversions implemented by items 2 and 7 are the only ones occurring in our setting, together with those possibly called by the congruence rule for constant invocation. Routed substitutions will be used in the operational semantics. We will make sure later that, whenever applied, routed substitution is a conservative extension of the standard one to localized names. Also, by inspecting Tab. 5.1, it will be clear that, if $u' \in \mathcal{N}$ then $u \in \mathcal{N}' \setminus \mathcal{N}$.

We will now work out an application of routed substitution. Consider for instance

$$(2 \Rightarrow \overline{x}y.w(z).\overline{z}z \mid 2 \Rightarrow x(z).(\nu\, x)\overline{w}x.(x(y) \mid \overline{x}z))\{\!|\bullet 1/x|\!\}.$$

First, item 3 applies, that distributes the substitution $\{\!|\bullet 1/x|\!\}$ to the two components. The name $\bullet 1$ is enriched with the tag $||_0$ (respectively $||_1$) in the left (respectively right) component needed to point back to the environment where $\bullet 1$ has been created. Indeed, $\bullet 1$ exported to $||_i\bullet$ is $||_i \bullet \star \bullet 1 = ||_i \bullet 1$. Instead, the name x to be substituted is not enriched with a relative address, because $x \in \mathcal{N}$. So, one obtains

$$\begin{aligned}
&(2 \Rightarrow \overline{x}y.w(z).\overline{z}z\{\!|\, ||_0\bullet 1/x|\!\}) \mid (2 \Rightarrow x(z).(\nu\, x)\overline{w}x.(x(y) \mid \overline{x}z)\{\!|\, ||_1\bullet 1/x|\!\}) = \\
&\quad (2 \Rightarrow \overline{||_0\bullet 1}y.w(z).\overline{z}z) \mid (2 \Rightarrow ||_1\bullet 1(z).((\nu\, x)\overline{w}x.(x(y) \mid \overline{x}z))\{\!|\, ||_1\bullet 1/x|\!\}) = \\
&\quad (2 \Rightarrow \overline{||_0\bullet 1}y.w(z).\overline{z}z) \mid (2 \Rightarrow ||_1\bullet 1(z).(\nu\, x)\overline{w}x.(x(y) \mid \overline{x}z))
\end{aligned}$$

Note that, the substitution takes the binders of names into account correctly. In fact, the distribution of $\{\!|\bullet 1/x|\!\}$ stops when the new binder $(\nu\, x)$ is encountered, according to the second case of item 2. This example will be used just after Proposition 5.3.8.

We show now an example where an α-conversion is forced. Suppose to have $(n+1) \Rightarrow (\nu\, a)\ \overline{a}\ \bullet n\{\!|a/\bullet n|\!\}$.[3] Then, the first rule of item 2 applies and exploits the fresh negative name $\bullet - n$, yielding:

$$\begin{aligned}
&(n+1) \Rightarrow (\nu \bullet - n)\ \overline{a}\ \bullet n\{\!|\bullet - n/a|\!\}\{\!|a/\bullet n|\!\} = \\
&(n+1) \Rightarrow (\nu \bullet - n)\ \overline{\bullet - n}\ \bullet n\{\!|a/\bullet n|\!\} = (n+1) \Rightarrow (\nu \bullet - n)\ \overline{\bullet - n}\ a.
\end{aligned}$$

For notational convenience, we introduce two forms of selective substitutions within a sub-process reachable from the whole process via a path ϑ.

Definition 5.3.6 *Let t be an extended process and $\vartheta \in \{||_0, ||_1\}^*$. Then, the* selective routed substitution *$t\{\!|/|\!\}_{@\vartheta}$ is defined by induction as*

- $(t_0|t_1)\{\!|r/s|\!\}_{@\vartheta} = \begin{cases} t_0\{\!|r/s|\!\}_{@\vartheta'}|t_1 & \text{if } \vartheta = ||_0\vartheta' \\ t_0|t_1\{\!|r/s|\!\}_{@\vartheta'} & \text{if } \vartheta = ||_1\vartheta' \end{cases}$
- $t\{\!|r/s|\!\}_{@\epsilon} = t\{\!|r/s|\!\}$

The selective increment *of counters $t\{n+1 \Rightarrow /n \Rightarrow\}_{@\vartheta}$ is defined as follows, by stipulating that $(n+1 \Rightarrow)$ and $(n \Rightarrow)$ are considered as tokens*

- $(t_0|t_1)\{n+1 \Rightarrow /n \Rightarrow\}_{@||_i\vartheta} = t_i\{n+1 \Rightarrow /n \Rightarrow\}_{@\vartheta}$
- $t\{n+1 \Rightarrow /n \Rightarrow\}_{@\epsilon} = t\{n+1 \Rightarrow /n \Rightarrow\}$.

[3] This is an intermediate step in the deduction of the communication of the process $(\nu\, b)(m \Rightarrow\ \overline{b}a|n \Rightarrow b(x)(\nu\, a)\overline{a}x)$, according to Tab. 5.1.

The selective routed substitution first travels downwards a tree of processes to single out the sub-process P' reachable via ϑ. Then, it applies the routed substitution to P'. Note that the names s and r are *not* updated while descending the tree, because when this selective substitution is called (rules Com_i and $Close_i$ in Tab. 5.1), they are already localized in P'. The selective increment simply adds 1 to the counter of the process reachable via ϑ. It will be used in rule *Open*, only. Both $t\{\!| -/- |\!\}_{@\vartheta}$ and $t\{-/-\}_{@\vartheta}$ are partial. It will be evident that they are indeed defined, whenever used in the rules for the operational semantics.

The late transition system for the π-calculus with localized names is in Tab. 5.1. Following the ideas of [45, 48], we encode in the labels of transitions the parallel structure of processes to identify the sequential component that moves. More precisely a label is

$$\mu@\vartheta \quad \text{(or simply } \mu \text{ if } \vartheta = \epsilon \text{ or } \mu = \tau\text{)},$$

where ϑ encodes the nesting of the process which performed the μ action, as far as the parallel structure of the system is concerned. In fact, a tag $||_0$ (respectively $||_1$) is added whenever a rule Par_0 (respectively Par_1) is applied. Rules In and $Open$ generate a new name $\bullet n$ and increment the counter of the sequential component that moves. In the case of In, the new name is distributed to the residual through the routed substitution $\{\!|\bullet n/x|\!\}$ which enriches $\bullet n$ with the correct relative address. As for *Open*, the new name must be distributed to t, the whole process under restriction. Thus, we use the name $\bullet\vartheta n$ localized in t, i.e. $\bullet\vartheta \star \bullet n$. Also, the counter of the sub-process reachable via ϑ is increased by the operation of selective increment, because this sub-process generated a fresh name. The rules for communication *locally* check whether the channel is the same. The receiver then distributes the value read to the sequential component t_i reachable via ϑ_i (which fired the input) by using the information encoded in the @-part of the label of the transition. Indeed, s_1 is s_0 exported to $||_1\vartheta_1\bullet||_0\vartheta_0$ in Com_0; and symmetrically for Com_1. Note that rules $Close_i$ introduce the correct restriction on the name computed by sending the object of the output to the minimal common predecessor of t_0 and t_1 in the tree of processes.

Note that our rules have *no global* conditions on names. In particular, we drop the side conditions of the original operational semantics of the π-calculus on free and bound names. The only rule which applies substitutions to a context larger than a sequential process is *Open*. However, only the operand of ν is affected. This context is the minimum needed to notify the owners of a name which for them is no longer private.

We now derive a transition. Consider the extended process occurring in Fig. 5.6

$$2 \Rightarrow w(z).\overline{z}z \mid 3 \Rightarrow (\nu\, x)\overline{w}x.(x(||_0\bullet - 2) \mid \overline{x}y).$$

We deduce a communication between $w(z)$ and $\overline{w}x$, by using rule $Close_1$.

$$Out: n\Rightarrow \overline{r}s.P \xrightarrow{\overline{r}s} n\Rightarrow P \qquad In: n\Rightarrow r(s).P \xrightarrow{r(\bullet n)} (n+1)\Rightarrow P\{\!|\bullet n/s|\!\}$$

$$Tau: n\Rightarrow \tau.P \xrightarrow{\tau} n\Rightarrow P$$

$$Par_0: \frac{t_0 \xrightarrow{\mu@\vartheta} t_0'}{t_0|t_1 \xrightarrow{\mu@||_0\vartheta} t_0'|t_1} \quad Par_1: \frac{t_1 \xrightarrow{\mu@\vartheta} t_1'}{t_0|t_1 \xrightarrow{\mu@||_1\vartheta} t_0|t_1'} \quad Sum: \frac{n\Rightarrow P \xrightarrow{\mu@\vartheta} t'}{n\Rightarrow (P+Q) \xrightarrow{\mu@\vartheta} t'}$$

$$Res: \frac{t \xrightarrow{\mu@\vartheta} t'}{(\nu u)t \xrightarrow{\mu@\vartheta} (\nu u)t'},\ \mu \neq \tau \ \Rightarrow \begin{cases} u \notin \{r,s\} \\ r = \bullet\vartheta \star sbj(\mu) \\ s = \bullet\vartheta \star obj(\mu) \end{cases}$$

$$Com_0: \frac{t_0 \xrightarrow{\overline{r_0}s_0@\vartheta_0} t_0',\, t_1 \xrightarrow{r_1(\bullet m)@\vartheta_1} t_1'}{t_0|t_1 \xrightarrow{\tau} t_0'|t_1'\{\!|s_1/\bullet m|\!\}_{@\vartheta_1}},\ \begin{cases} s_1 = ||_1\vartheta_1\bullet||_0\vartheta_0 \star s_0 \\ r_1 = ||_1\vartheta_1\bullet||_0\vartheta_0 \star r_0 \end{cases}$$

$$Com_1: \frac{t_0 \xrightarrow{r_1(\bullet m)@\vartheta_1} t_0',\, t_1 \xrightarrow{\overline{r_0}s_0@\vartheta_0} t_1'}{t_0|t_1 \xrightarrow{\tau} t_0'\{\!|s_1/\bullet m|\!\}_{@\vartheta_1}|t_1'},\ \begin{cases} s_1 = ||_0\vartheta_1\bullet||_1\vartheta_0 \star s_0 \\ r_1 = ||_0\vartheta_1\bullet||_1\vartheta_0 \star r_0 \end{cases}$$

$$Open: \frac{t \xrightarrow{\overline{r}s@\vartheta} t'}{(\nu u)t \xrightarrow{\overline{r}(\bullet n)@\vartheta} (t'\{n+1\Rightarrow /n\Rightarrow\}_{@\vartheta})\{\!|\bullet\vartheta n/u|\!\}},\ \begin{cases} u = \bullet\vartheta \star s \\ r \neq s \end{cases}$$

$$Close_0: \frac{t_0 \xrightarrow{\overline{r_0}(s_0)@\vartheta_0} t_0',\, t_1 \xrightarrow{r_1(\bullet m)@\vartheta_1} t_1'}{t_0|t_1 \xrightarrow{\tau} (\nu u)(t_0'|t_1'\{\!|s_1/\bullet m|\!\}_{@\vartheta_1})},\ \begin{cases} s_1 = ||_1\vartheta_1\bullet||_0\vartheta_0 \star s_0 \\ r_1 = ||_1\vartheta_1\bullet||_0\vartheta_0 \star r_0 \\ u = \bullet||_0\vartheta_0 \star s_0 \end{cases}$$

$$Close_1: \frac{t_0 \xrightarrow{r_1(\bullet m)@\vartheta_1} t_0',\, t_1 \xrightarrow{\overline{r_0}(s_0)@\vartheta_0} t_1'}{t_0|t_1 \xrightarrow{\tau} (\nu u)(t_0'\{\!|s_1/\bullet m|\!\}_{@\vartheta_1}|t_1')},\ \begin{cases} s_1 = ||_0\vartheta_1\bullet||_1\vartheta_0 \star s_0 \\ r_1 = ||_0\vartheta_1\bullet||_1\vartheta_0 \star r_0 \\ u = \bullet||_1\vartheta_0 \star s_0 \end{cases}$$

Table 5.1: Late transition system of the π-calculus with localized names.

$$\dfrac{2\Rightarrow w(z).\overline{z}z \xrightarrow{w(\bullet 2)} 3\Rightarrow \overline{\bullet 2}\bullet 2, \quad \dfrac{3\Rightarrow \overline{w}x.(x(||_0\bullet - 2) \,|\, \overline{x}y) \xrightarrow{\overline{w}x} (3\Rightarrow x(||_0\bullet - 2) \,|\, 3\Rightarrow \overline{x}y)}{(\nu\, x)(3\Rightarrow \overline{w}x.(x(||_0\bullet - 2) \,|\, \overline{x}y) \xrightarrow{\overline{w}(\bullet 3)} (4\Rightarrow ||_0\bullet 3(||_0\bullet - 2) \,|\, 4\Rightarrow \overline{||_1\bullet 3}y)}}{2\Rightarrow w(z).\overline{z}z \,|\, 3\Rightarrow (\nu\, x)\overline{w}x.(x(||_0\bullet - 2) \,|\, \overline{x}y) \xrightarrow{\tau} (\nu \bullet||_1 3)(3\Rightarrow \overline{||_0\bullet||_1 3}\, ||_0\bullet||_1 3 \;|\; (4\Rightarrow ||_0\bullet 3(||_0\bullet - 2) \,|\, 4\Rightarrow \overline{||_1\bullet 3}y)}$$

Consider the target of the transition above, in which $||_0\bullet 3$, $||_1\bullet 3$ and $\bullet||_1 3$ occur in different sub-processes but denote the same thing. Thus $(\nu \bullet||_1 3)$ enforces communication between the rightmost processes. Also, $||_0\bullet||_1 3$, the name extruded, is an alias for $\bullet||_1 3$, hence the leftmost process is stuck.

The following proposition ensures that no name in Tab. 5.1 is left unknown and that every call to address composition $\star$ is well defined.

Proposition 5.3.7 *All names in Tab. 5.1 are defined, i.e. every call to address composition $\star$ is defined.*

Proof. When names are in $\mathcal{N}$, the proof is obvious. Otherwise, note that if $t \xrightarrow{\mu @ \vartheta} t'$ and $\vartheta''\bullet\vartheta' m \in n(\mu)$, then either ϑ'' is a suffix of ϑ or vice versa. Indeed, let P be the minimal common predecessor of $\mu@\vartheta$ and of G, the generator of name $\bullet m$. Assume that in the derivation of the transition $t \xrightarrow{\mu @ \vartheta} t'$ P has been reached, i.e. P is a sub-term of t. Then certainly ϑ'' is a suffix of ϑ, because ϑ'' is the path from P to the sub-process of t that performs action μ, i.e. reachable via ϑ. Otherwise, if P has not been yet reached then ϑ is a suffix of ϑ''.
Now it is routine checking that all the calls to $\star$ are well defined (see Def. 5.2.3). ■

We now show that the routed substitution behaves as the standard one, whenever applied. The only difference is that relative addresses are updated and that α-conversions are made explicitly.

Proposition 5.3.8 *Whenever applied in Tab. 5.1, the routed substitution $\{\!| - / - |\!\}$ conservatively extends the standard substitution $\{-/-\}$.*

Proof. Recall that a computation starts from $1 \Rightarrow P$, with $P \in \mathcal{P}$ and that any extended process t reachable from it only has extended sub-processes $n \Rightarrow t'$ with $n \geq 1$. By Proposition 5.3.7, the names occurring in the routed substitutions are all defined. Then, note that a routed substitution preserves the identity of an object referenced to by a localized name $\vartheta\bullet\vartheta' n$ when this is updated to $||_i\bullet \star \vartheta\bullet\vartheta'$, by item 3 of Def. 5.3.5. Thus, it suffices to prove that a routed substitution implements an α-conversion, whenever the standard one would. This occurs whenever $\{\!|u'/u|\!\}_{@\vartheta}$ is applied to t in the conclusion of a rule in Tab. 5.1, and u does not occur free in the sub-process t reachable via ϑ. In these cases, items 2 or 7 of Def. 5.3.5, with $u' = s$, apply. We proceed by induction of the derivation of transitions in these cases.

In: holds vacuously.

Com_i and $Close_i$: $\bullet m$ has been just generated in the sub-process t'_1 reachable via ϑ, hence all negative names $\vartheta\bullet\vartheta' - k$ therein, if any, are such that $k < m$. Therefore $\bullet - m$ is fresh.

$Open$: $\bullet n$ is fresh in $n \Rightarrow P$, i.e. the sub-process of t reachable via ϑ, consequently also $\bullet\vartheta n$ is fresh in t' and in all its sub-processes, because updated by the routed substitution, whenever needed.

■

$$
\begin{array}{c}
1 \Rightarrow a(x).(\overline{x}y.w(z).\overline{z}z \mid x(z).(\nu\, x)\overline{w}x.(x(y) \mid \overline{x}z)) \\
\Big\downarrow {\scriptstyle a(\bullet 1)} \\
2 \Rightarrow \overline{||_0\bullet 1}y.w(z).\overline{z}z \mid 2 \Rightarrow ||_1\bullet 1(z).(\nu\, x)\overline{w}x.(x(y) \mid \overline{x}z) \\
\Big\downarrow {\scriptstyle \tau} \\
2 \Rightarrow w(z).\overline{z}z \mid 3 \Rightarrow (\nu\, x)\overline{w}x.(x(||_0\bullet - 2) \mid \overline{x}y) \\
\Big\downarrow {\scriptstyle \tau} \\
(\nu\, \bullet||_1 3)(3 \Rightarrow \overline{||_0\bullet||_1 3}\, ||_0\bullet||_1 3 \mid (4 \Rightarrow ||_0\bullet 3(||_0\bullet - 2) \mid 4 \Rightarrow \overline{||_1\bullet 3}y))
\end{array}
$$

Figure 5.6: A computation of $1 \Rightarrow a(x).(\overline{x}y.w(z).\overline{z}z \mid x(z).(\nu\, x)\overline{x}v.(x(y) \mid \overline{x}z))$.

An example of how extended processes evolve is in the computation depicted in Fig. 5.6. Its first transition shows the generation of the new name $\bullet 1$. The application of the routed substitution $\{\!|\bullet 1/x|\!\}$ introduced by rule In is in the example after Def. 5.3.5. The counter $2 \Rightarrow$ is distributed to the components of the parallel composition. The second transition is a communication along a channel called $||_0\bullet 1$ by the sender, and called $||_1\bullet 1$ by the receiver. Indeed, $||_0\bullet 1$ is an alias for $||_1\bullet 1$. Also, the counter of the residual of the receiver is increased. In fact, rule In generates a new name $\bullet 2$ that will be replaced with y by substitution $\{\!|y/\bullet 2|\!\}$ in the conclusion of rule Com_0. Also there is an α-conversion, because $(x(y)|\overline{x}\ ||_1\bullet - 2)\{\!|y/\bullet 2|\!\} = x(||_0\bullet - 2)|\overline{x}y$. The derivation of the last transition is the one worked out before Proposition 5.3.7.

Our transition system is a more concrete version of the original one in Tab. 2.1. However, there is a tight connection between them that we will formalize at the end of the next section. Intuitively, our transitions are "routed" variants of the original ones.

5.4 A Global View of Names

The names occurring in the label of $t \xrightarrow{\mu @ \vartheta} t'$ have a local meaning. This enables processes to exchange names only knowing that they run in parallel, with *no* knowledge of the *global* context in which they are plugged. Indeed, if $\mu = \overline{r}(s)$, the actual reference of r has to be computed following the relative address of r, composed with ϑ; similarly for s. In this section we show how to globalize names in the style of Internet addressing, though they are generated and handled locally. We propose to transform a name as it is known at the root of the tree of processes. We thus have a conversion table of names, but *not* a global manager of names.

The logical addresses of Internet form an n-adic tree, such that all the arcs at the same level have distinct names, each identifying a different domain. When a new address is assigned in a domain (local environment), for instance in a local area network, it is unique. Of course the same name can be assigned to hosts of other domains. They will be distinguished when prefixed with the paths to their domains. See [63] for a technique similar to ours that handles names locally in this setting.

Rephrasing the above in our case, a new local name is generated by the local counter of the sequential component that needs it, while uniqueness is ensured by the relative address of the generator with respect to the root of the system. The correctness of this proposal follows from the fact that a name is uniquely identified by a suitable composition of its relative address and of the access path from the root of the whole system to the sub-process that actually uses it.

Our localized names are converted by the following transition relation which uses the relation defined in Tab. 5.1 as an auxiliary arrow. By abuse of notation, we will write $\vartheta_\bullet\vartheta' \star \mu$ for $\vartheta_\bullet\vartheta' \star sbj(\mu)\ \ \vartheta_\bullet\vartheta' \star obj(\mu)$ if $\mu \neq \tau$, and for τ otherwise.[4]

$$\boxed{\text{if } t \xrightarrow{\mu @ \vartheta} t' \text{ then } t \xmapsto{\bullet\vartheta\star\mu} t'}$$

To illustrate the new style of naming, we consider the last step of the first computation described in Example. 5.1. The label of the output transition in $\longrightarrow$ was

$$\mu = \overline{print}\langle \text{stud}_\bullet\text{teach}\, myprog, \text{did.stud}_\bullet\text{adm.did.lectures}\, mytimetab\rangle @\text{did.stud},$$

and an easy calculation shows that the actual label in the output transition in $\mapsto$ is

$$\bullet\text{did.stud} \star \mu = \overline{print}\langle \bullet\text{did.teach}\, myprog, \bullet\text{adm.did.lectures}\, mytimetab\rangle.$$

The names that are object of the output are now displayed with their accessing paths from the root of the system to the processes they belong to.

Now we can show that our transition systems are in bijection with the original one of [89], up to a sort of routed renaming of their transitions. Essentially, a transition

[4] See also Def. 5.6.1. Of course, parentheses and overbars occurring in μ are preserved in $\vartheta_\bullet\vartheta' \star \mu$.

$t \xrightarrow{\mu'@\vartheta} t'$ can be seen as a variant of $P \xrightarrow{\mu} P'$, where μ, P and P' are obtained from μ', t and t' replacing localized names with fresh names in $\mathcal{N}$. Also counters $n\Rightarrow$ are discarded from extended processes by the following auxiliary function FC.

Definition 5.4.1
The auxiliary function FC is the homomorphic extension of $fc(n \Rightarrow P) = P$ to extended processes.

The correspondence between the standard transition for the π-calculus and ours is now easy to state. In its statement, given suitable $A \subseteq \mathcal{N}$ and $U \subseteq (\mathcal{N}' \setminus \mathcal{N})$, we denote by $\{\!|A/U|\!\}$ the simultaneous routed substitution $\{\!|a_1/u_1, \ldots, a_k/u_k|\!\}$ for all $u_i \in U$, with $a_i \neq a_j$ if $i \neq j$.

Theorem 5.4.2

$$P_0 \xrightarrow{\mu}_L P_1 \textit{ if and only if } t_0 \xrightarrow{\mu'@\vartheta} t_1,$$

with $\mu = (\bullet\vartheta \star \mu')\{\!|A/U|\!\}$ *and for* $i = 0, 1,\ P_i =_\alpha FC(t_i\{\!|A/U|\!\})$,

where $U \supseteq (n(t_0, t_1) \cup obj(\mu')) \cap (\mathcal{N}' \setminus \mathcal{N})$ *and* $A \cap fn(t_0, t_1) = \emptyset$

Furthermore, $P_0 \xrightarrow{\mu}_L P_1$ *if and only if* $t_0 \stackrel{\mu''}{\longmapsto} t_1$, *where* $\mu'' = \bullet\vartheta \star \mu'$.

Proof. We obtain a new set of rules for the standard π-calculus, equivalent to that in Tab. 2.1, as follows. Split axiom *act* in three axioms (*tau* for τ, *in* for input and *out* for output); remove the congruence rules for restriction; remove the monoidal laws for $|$ and add the symmetrical rules for this operator. The new rules are in bijection with those in Tab 5.1. Therefore, the deduction of the transitions $Q \xrightarrow{\mu} Q'$ and $t \xrightarrow{\mu@\vartheta'} t'$, with $Q = FC(t)$ are also in bijection, because the counters $n\Rightarrow$ do not affect the deduction.

Only if) Take $t_0 = n\Rightarrow P_0$ and proceed by induction on the derivation of $P_0 \xrightarrow{\mu}_L P_1$.

[*Base step*] Trivial if $\mu = \tau.P$ (rule *tau*) or $\mu = \overline{x}y.P$ (rule *out*). If the axiom is *in*, $P_0 = x(y).P_1$ and

$$n\Rightarrow P_0 \xrightarrow{x(\bullet n)} (n+1)\Rightarrow P_1\{\!|\bullet n/y|\!\}.$$

Since $\{\!|\bullet n/y|\!\}$ calls no α-conversion as $\bullet n$ is free in P_1 (and is the unique name in $\mathcal{N}' \setminus \mathcal{N}$ therein), it suffices to take $A = \{w\}$ for $w \notin n(P_0, P_1)$.

[*Inductive step*] If the last rule applied is *sum* or *res*: immediate by inductive hypothesis as no routed substitution is needed. Similarly for the new rules par_i, because the rules in Tab. 5.1 require no routed substitutions and $\{\!|A/U|\!\}$ takes care of the $||_i$ prefixed to ϑ.

The cases for the new rules for communication are symmetric, so we consider only com_0. The transition $t_0 \stackrel{\mu'@\vartheta}{\longrightarrow} t_1$ reads as

$$n\Rightarrow P_0|P_1 \stackrel{\tau}{\longrightarrow} t'_0|t'_1\{\!|y/\bullet m|\!\}_{@\vartheta_1},$$

and it is deduced because of $n \Rightarrow P_0 \stackrel{\overline{x}y@\vartheta_0}{\longrightarrow} t'_0$ and $n \Rightarrow P_1 \stackrel{x(\bullet n)@\vartheta_1}{\longrightarrow} t'_1$, which by inductive hypothesis correspond to $P_0 \stackrel{\overline{x}y}{\longrightarrow}_L P'_0$ and to $P_1 \stackrel{x(z)}{\longrightarrow}_L P'_1$. If $y \in bn(P'_1)$ the substitution $P'_1\{y/z\}$ may require α-conversions, replacing the bound occurrences of y with, say, $w \notin A \cup fn(t_0, t_1)$. Then

$$FC(t'_1\{\!|y/\bullet m|\!\}_{\vartheta_1})\{\!|A \cup \{w\}/U \cup \{\bullet\vartheta_1 \star \bullet - m\}|\!\} =_\alpha P'_1\{y/z\}.$$

For rule *open*, the transition $t_0 \stackrel{\mu'@\vartheta}{\longrightarrow} t_1$ actually is

$$n\Rightarrow (\nu y)P_0 \stackrel{\overline{x}(\bullet n)@\vartheta}{\longrightarrow} t'_1 = (t_1\{n+1\Rightarrow /n\Rightarrow\}_\vartheta)\{\!|\bullet\vartheta n/u|\!\},$$

and is deduced because of $n\Rightarrow P \stackrel{\overline{x}(y)@\vartheta}{\longrightarrow} t'_1$. Now, $\bullet\vartheta n$ is the only localized name not in $\mathcal{N}$ of t'_1. For some fresh w, thus $FC(t'_1\{\!|w/\bullet\vartheta n|\!\}) =_\alpha P_1$.

Similarly for the new rules $close_i$, because the objects of the bound output $\overline{x}(\bullet n)@\vartheta_{out}$ and of the input $x(\bullet m)@\vartheta_{in}$ are the only localized names not in $\mathcal{N}$ of the processes in the premises, and the routed substitution requires no introduction of negative names.

If) The deduction Δ of $t_0 \stackrel{\mu'@\vartheta}{\longrightarrow} t_1$ induces a corresponding deduction Δ' of

$$1\Rightarrow FC(t_0\{\!|A/U|\!\}) \stackrel{\mu'\{\!|A/U|\!\}@\vartheta}{\longrightarrow} t'_1 = t_1\{\!|A/U|\!\},$$

because counters $n\Rightarrow$ do not affect deductions. t'_1 has at most two localized names not in $\mathcal{N}$, i.e. $n(t'_1) \cap \{\mathcal{N}' \setminus \mathcal{N}\} \subseteq \{\bullet 1, \bullet - 1\} = U'$. Now, for $w, z \notin n(t_0, t'_1) \cup A$, the required P_0 is $FC(t_0\{\!|A \cup \{w, z\}/U \cup U'|\!\}) = FC(t_0\{\!|A/U|\!\})$. Now the transition $P_0 \stackrel{\mu}{\longrightarrow}_L P_1$ can be derived mimicking Δ', where P_1 is $FC(t_1\{\!|A \cup \{w, z\}/U \cup U'|\!\})$ and $\mu = (\bullet\vartheta \star \mu')\{\!|A \cup \{w, z\}/U \cup U'|\!\}$.

The second claim follows immediately, by definition of $\longmapsto$. ■

The new transition relation $t \stackrel{\mu'}{\longmapsto} t'$ permits a global observer to follow the evolution of a system, by abstracting from its internal structure. In the next section we will easily adapt standard bisimulation to our case, relying on this new arrow $\longmapsto$. Note also that all the computations in our new setting are fresh, in the sense of [25]. Roughly speaking, in a fresh computation all names exported by an extrusion or imported by an input have never been used in the preceding transitions.

Finally, note that here the space of logical addresses is binary because the parallel operator of the π-calculus is such. Recall that our address composition requires no modification if trees are n-adic. As a consequence, our proposal is flexible enough to handle n-adic trees of logical addresses. For example, we model the insertion of a domain D' in a given domain D with n sub-domains, by adding a new arc from D to D', labelled with $||_{n+1}$ (see Fig. 5.7). In fact, the rule defining the new transition $\longmapsto$ shows that no modification is needed on the "new" names occurring in D', as well as on the "old" ones before the insertion.

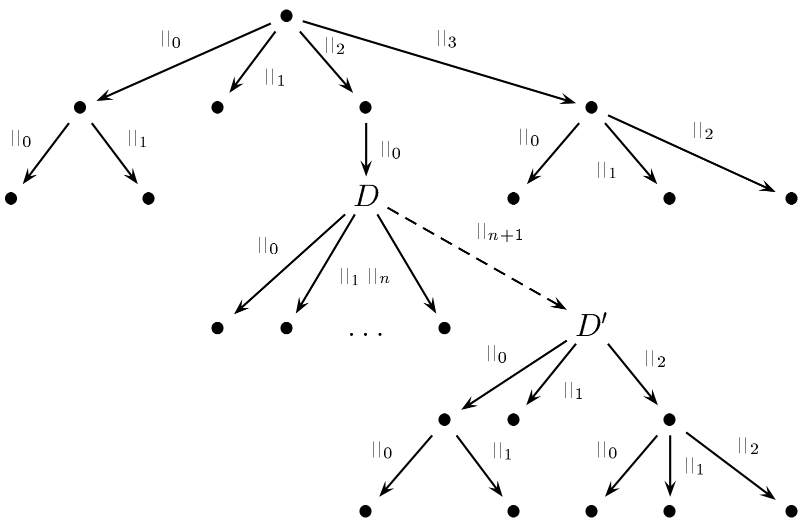

Figure 5.7: Adding a new domain D' as a sub-domain of D in a tree of processes.

5.5 Equivalences

The equivalences on the π-calculus are usually based on bisimulations. Their standard definitions compare the behaviour of a computational step in one system with that of another and check the target systems for bisimilarity. We now accommodate the definition of late bisimulation [89] (see Section 1.2 for a review) to fit our framework.

There is no difference with the standard definition as far as output and silent transitions are concerned. Even in the case of input, we need no change (although the operational rule In calls for a routed substitution). Indeed, the argument of the input is only a placeholder that must be instantiated with any name $w \in \mathcal{N}$, as done in Def. 1.2.3. We only need some care when generating new names through extrusions. In this case, too, we replace the object of the bound output $\vartheta \bullet \vartheta' n$ with a new name $b \in \mathcal{N}$ within the two processes. The derivatives obtained in this way are then checked for bisimilarity. More formally, we have the following definition.

Definition 5.5.1 (name late bisimulation) *A binary relation $\mathcal{R}$ on extended processes is a* name late simulation *if $t_0 \, \mathcal{R} \, t_1$ implies that*

- *If* $t_0 \overset{\mu}{\longmapsto} t_0'$ *and* μ *is* τ, $\overline{x}y$, *then for some* t_1', $t_1 \overset{\mu}{\longmapsto} t_1'$ *and* $t_0' \mathcal{R}\, t_1'$
- *If* $t_0 \overset{x(r)}{\longmapsto} t_0'$ *then for some* t_1', $t_1 \overset{x(s)}{\longmapsto} t_1'$ *and for all* $w \in \mathcal{N}$, $t_0'\{\!|w/r|\!\}\mathcal{R} t_1'\{\!|w/s|\!\}$
- *If* $t_0 \overset{\overline{x}(r)}{\longmapsto} t_0'$ *then for some* t_1', $t_1 \overset{\overline{x}(s)}{\longmapsto} t_1'$ *and for a name* $b \in \mathcal{N}$, $b \notin n(t_0, t_1)$, $t_0'\{\!|b/r|\!\}\mathcal{R} t_1'\{\!|b/s|\!\}$.

The relation $\mathcal{R}$ *is a* name late bisimulation *if both* $\mathcal{R}$ *and* $\mathcal{R}^{-1}$ *are name late simulations.* t_0 *is* name late bisimilar *to* t_1 *(written* $t_0 \approx_N t_1$*) if there exists a name late bisimulation* $\mathcal{R}$ *such that* $t_0 \mathcal{R}\, t_1$.

Note that there is no need for the condition $r, s \notin fn(t_0, t_1)$ in the last two items, because both rules *In* and *Open* generate fresh localized names. We now prove that the name late equivalence, induced by the bisimulation above, coincides with the standard late equivalence.

Theorem 5.5.2

$$P \approx P' \textit{ if and only if } t \approx_N t',$$

with $FC(t) =_\alpha P$ *and* $FC(t') =_\alpha P'$.

Proof. *If*) By the second claim of Theorem 5.4.2, there exists a bijection, say ϕ, between the state space of P and t. Similarly, let ϕ' be the bijection between the state space of P' and t'. Given a late bisimulation $\mathcal{R}$ containing $\langle P, P'\rangle$, let

$$\mathcal{R}_n = \{\langle \phi(P_i), \phi'(P_i')\rangle \mid \langle P_i, P_i'\rangle \in \mathcal{R}\}.$$

To show that $\mathcal{R}_n$ is name late bisimulation we only need to note that the condition in the third item of Def. 5.5.1, i.e. $b \notin n(t_0, t_1)$ amounts to taking a variant of both transitions $t_0 \overset{\overline{x}(r)}{\longmapsto} t_0'$ and $t_1 \overset{\overline{x}(s)}{\longmapsto} t_1'$. This is obvious because b is fresh in both processes under consideration.

Only if) Let $\mathcal{R}_n$ be a localized late bisimulation containing $\langle t, t'\rangle$. Then,

$$\mathcal{R} = \{\langle FC(t_i), FC(t_i')\rangle \mid \langle t_i, t_i'\rangle \in \mathcal{R}_n\}$$

is a late bisimulation by the second claim of Theorem 5.4.2 and because α-convertible processes are congruent. The correspondence in the case of the *Open* transitions relies on the same argument used in the *If* part. ■

We end this section with an example. Consider the extended processes

$$t_0 = 1\Rightarrow\ (\nu\, a)(\nu\, c)\ \overline{x}a.(\overline{c}b \mid c(z).a(y).\overline{y}z) \text{ and } t_0' = 5\Rightarrow\ (\nu\, a)\ \overline{x}a.\tau.a(y)\overline{y}b$$

which are clearly late bisimilar once counters have been erased. The computations of the two processes are depicted in Fig. 5.8.

$$\begin{array}{cc}
t_0 = 1\Rightarrow\ (\nu\, a)(\nu\, c)\overline{x}a.[\overline{c}b \mid c(z).a(y).\overline{y}z] & t'_0 = 5\Rightarrow\ (\nu\, a)\overline{x}a.\tau.a(y).\overline{y}b \\
\Big\downarrow{\scriptstyle \overline{x}(\bullet 1)} & \Big\downarrow{\scriptstyle \overline{x}(\bullet 5)} \\
t_1 = 2\Rightarrow\ (\nu\, c)[\overline{c}b \mid c(z).||_1\bullet 1(y).\overline{y}z] & t'_1 = 6\Rightarrow\ \tau.\bullet 5(y).\overline{y}b \\
\Big\downarrow{\scriptstyle \tau} & \Big\downarrow{\scriptstyle \tau} \\
t_2 = (\nu\, c)[2\Rightarrow\ \mathbf{0} \mid 3\Rightarrow\ ||_1\bullet 1(y).\overline{y}b] & t'_2 = 6\Rightarrow\ \bullet 5(y).\overline{y}b \\
\Big\downarrow{\scriptstyle \bullet 1(\bullet 3)} & \Big\downarrow{\scriptstyle \bullet 5(\bullet 6)} \\
t_3 = (\nu\, c)[2\Rightarrow\ \mathbf{0} \mid 4\Rightarrow\ \bullet\overline{3}b] & t'_3 = 7\Rightarrow\ \bullet\overline{6}b \\
\Big\downarrow{\scriptstyle \bullet\overline{3}b} & \Big\downarrow{\scriptstyle \bullet\overline{6}b} \\
t_4 = (\nu\, c)[2\Rightarrow\ \mathbf{0} \mid 4\Rightarrow\ \mathbf{0}] & t'_4 = 7\Rightarrow\ \mathbf{0}
\end{array}$$

Figure 5.8: The computations of t_0 and t'_0.

To show that t_0 can simulate t'_0 we build a simulation $t_0\mathcal{R}t'_0$. We choose $d \notin n(t_0, t'_0)$, and we show that $t_1\{d/\bullet 1\}\mathcal{R}t'_1\{d/\bullet 5\}$. The processes after the bound outputs are $2\Rightarrow (\nu\, c)(\overline{c}b \mid c(z).d(y).\overline{y}z)$ and $6\Rightarrow\ \tau.d(y).\overline{y}b$ and both perform a τ. Hence, we prove that $t_2\mathcal{R}t'_2$. This holds if $\forall w \in \mathcal{N}, t_3\{w/\bullet 3\}\mathcal{R}t'_3\{w/\bullet 6\}$, i.e. if $(\nu\, c)(2\Rightarrow\ \mathbf{0} \mid 4\Rightarrow\ \overline{w}b)\ \mathcal{R}\ 7\Rightarrow\ \overline{w}b$. Now the counters are immaterial and the simulation is obvious. The relation $\mathcal{R}^{-1}$ is built likewise.

5.6 Higher-Order Mobile Processes

We show the robustness of our approach to the semantics of mobile processes, by taking a late version of the *HOπ*-calculus [114] (see Section 1.2) as a test bed.

When a process P migrates, it exports all its names at the same time. We need to extend the notions on names introduced in Section 5.3 to processes, considered as name containers. For the sake of brevity, we will not give the definitions in detail, but only show what has to be added or changed when the object of an action is a process.

As far as syntax is concerned, we only add process variables, ranged over by V, and we assume that $V \in \mathcal{N}$. Also, we enclose in angled brackets a process object of an output, e.g. $\overline{a}\langle \overline{b}x.(c(z).\overline{z}z|\overline{d}x)\rangle$.

We re-use our example in Section 5.1 to help intuition. Assume that a student asks a teacher for the date of an exam. Suppose also that there is a clash of dates (this teacher is often quite absent minded), and the student asks if the old date od can be changed to the new one nd. The teacher sends a process Update(od,nd) which performs the change to be run by P_3. This change of date is modelled by defining processes P_0 and P_3 as follows, where u_chan is a global channel.

$$P_0 = \text{Teacher} + \overline{u\_chan}\langle \text{Update}(\text{did.teach}\bullet\text{adm.did.exams}\, od, \text{teach}\bullet\text{stud}\, nd)\rangle.P_0,$$

$$P_3 = u\_chan(V).V.P_3 + \text{Other\_exams}.$$

After the interaction between P_0 and P_3, the placeholder V in P_3 is instantiated to the process Update, and P_3 becomes

$$\text{Update}(\bullet\, od, \text{adm.did.exams}\bullet\text{did.stud}\, nd).P_3.$$

As for the semantic definitions, we first lift the address composition of Def. 5.2.3 to the more powerful process address composition, $\star^H$. It re-locates all the names of a process, distributing the address composition for names onto the communicated process P. In other words $\star^H$ composes the address of the sender of P relative to its receiver with the addresses of all the names contained in P. It is inductively defined on the syntax of processes and has localized names as base case, to which $\star$ applies. The only item of the following definition that deserves a comment is the fifth one. Suppose that a process $P = (P_0|P_1)$ has to be exported to some process Q, and that its address relative to Q is $\vartheta\bullet\vartheta'$. When $(P_0|P_1)$ replaces the placeholders in Q, carrying all its names, the address of the *old* instance of P_i in P, relative to the *new* instance of P_i in Q is $\vartheta||_i\bullet\vartheta'||_i$, for $i = 0, 1$. In the last item below, Y_i stands for either a name $r \in \mathcal{N}'$ or a process, possibly with localized names.

Definition 5.6.1 *Process address composition $\star^H$ is defined by*

- $\vartheta\bullet\vartheta' \star^H r = \vartheta\bullet\vartheta' \star r$
- $\vartheta\bullet\vartheta' \star^H \mathbf{0} = \mathbf{0}$
- $\vartheta\bullet\vartheta' \star^H \pi.P = \begin{cases} \vartheta\bullet\vartheta' \star sbj(\pi) \;\; \vartheta\bullet\vartheta' \star^H obj(\pi).\; \vartheta\bullet\vartheta' \star^H P & \textit{if } \pi \neq \tau \\ \tau.\; \vartheta\bullet\vartheta' \star^H P & \textit{otherwise} \end{cases}$
- $\vartheta\bullet\vartheta' \star^H (P_0 + P_1) = \vartheta\bullet\vartheta' \star^H P_0 \; + \; \vartheta\bullet\vartheta' \star^H P_1$
- $\vartheta\bullet\vartheta' \star^H (P_0|P_1) = \vartheta||_0\bullet\vartheta'||_0 \star^H P_0 \;\mid\; \vartheta||_1\bullet\vartheta'||_1 \star^H P_1$
- $\vartheta\bullet\vartheta' \star^H (\nu\, s)P = (\nu\; \vartheta\bullet\vartheta' \star s)\; \vartheta\bullet\vartheta' \star^H P$
- $\vartheta\bullet\vartheta' \star^H [r = s]P = [\vartheta\bullet\vartheta' \star r = \vartheta\bullet\vartheta' \star s]\; \vartheta\bullet\vartheta' \star^H P$
- $\vartheta\bullet\vartheta' \star^H A(Y_1, \ldots, Y_k) = A(\vartheta\bullet\vartheta' \star^H Y_1, \ldots, \vartheta\bullet\vartheta' \star^H Y_1)$

We apply below the process address composition to a simple example, by reporting only the relevant steps.

$$||_1\bullet||_0 \star^H (||_0\bullet 1(V).b(a).V \;\mid\; \overline{||_1\bullet 2}\;\; \langle(\nu\, a)\overline{a}\;\; ||_1\bullet 1\rangle) =$$

$$||_1||_0\bullet||_0||_0 \star^H (||_0\bullet 1(V).b(a).V) \;\;\mid\;\; ||_1||_1\bullet||_0||_1 \star^H (\overline{||_1\bullet 2}\;\; \langle(\nu\, a)\overline{a}\;\; ||_1\bullet 1\rangle) =$$

$$(||_1||_0\bullet||_0||_0 \star ||_0\bullet 1)(V).b(a).V \;\;\mid\;\; ||_1||_1\bullet||_0||_1 \star \overline{||_1\bullet 2}\;\; (||_1||_1\bullet||_0||_1 \star^H \langle(\nu\, a)\overline{a}\;\; ||_1\bullet 1\rangle) =$$

$$
\begin{array}{c}
1 \Rightarrow (d(a).d(e).\overline{c}\langle a(V).b(a).V \mid \overline{e}a\rangle \mid c(V').V') \\
\big\downarrow \scriptstyle d(\bullet 1)@||_0 \\
2 \Rightarrow d(e).\overline{c}\langle ||_0\bullet 1(V).b(a).V \mid \overline{e}\ ||_1\bullet 1\rangle \mid 1 \Rightarrow c(V').V' \\
\big\downarrow \scriptstyle d(\bullet 2)@||_0 \\
3 \Rightarrow \overline{c}\langle ||_0\bullet 1(V).b(a).V \mid \overline{||_1\bullet 2}\ ||_1\bullet 1\rangle \mid 1 \Rightarrow c(V').V' \\
\big\downarrow \scriptstyle \tau \\
3 \Rightarrow \mathbf{0} \mid 2 \Rightarrow (||_1||_0\bullet||_0 1(V).b(a).V \mid \overline{||_1||_1\bullet||_0 2}\ \ ||_1||_1\bullet||_0 1)
\end{array}
$$

Figure 5.9: A computation of $1 \Rightarrow (d(a).d(e).\overline{c}\langle a(V).b(a).V \mid \overline{e}a\rangle \mid c(V').V')$.

$$||_1||_0\bullet||_0 1(V).b(a).V \quad | \quad \overline{||_1||_1\bullet||_0 2}\ \langle(\nu\ a)\overline{a}\ \ ||_1||_1\bullet||_0||_1 \star ||_1\bullet 1\rangle =$$
$$||_1||_0\bullet||_0 1(V).b(a).V \quad | \quad \overline{||_1||_1\bullet||_0 2}\ \langle(\nu\ a)\overline{a}\ \ ||_1||_1\bullet||_0 1\rangle$$

It is straightforward to check that the process address composition homomorphically extends the address composition of names. Thus we can safely extend the results given for names to processes. In particular $\star^H$ is always defined, whenever used in the operational rules.

As sketched above, sending a process corresponds to sending all its names, using the process address composition.

Definition 5.6.2 *A process P exported to the relative address $\vartheta\bullet\vartheta'$ is $\vartheta\bullet\vartheta' \star^H P$.*

Similarly to what we did for $\star^H$, we enrich routed substitution to deal with processes, and we denote it by $\{\!| - / - |\!\}^H$. Its definition is exactly the same as Def. 5.3.5, except that names may now denote processes. For example, item 3 becomes

$$(t_0|t_1)\{\!|V'/V|\!\}^H = t_0\{\!|V_0'/V_0|\!\}^H|t_1\{\!|V_1'/V_1|\!\}^H,\ V_i' = ||_i\bullet \star^H V',\ V_i = ||_i\bullet \star^H V.$$

The definition of $\{\!| - / - |\!\}^H_{@\vartheta}$ is straightforward.

At this point it is easy to define a late transition system of the $HO\pi$-calculus with localized names (see [114] for the details on the standard operational semantics and [48] for its proved version). It is obtained from the one in Tab. 5.1, by replacing substitutions and address composition $\star$ with their higher-order versions. We briefly discuss two rules only, where we consider the relevant names to denote processes, in the case of input. Table 5.2 shows instances of the $HO\text{-}Com_0$ and $HO\text{-}In$ rules. The rule $HO\text{-}In$ still has a $\bullet n$ which, in this case, acts as a place-holder of the process to be received. As usual, $HO\text{-}Com_0$ checks whether the channel is the same and then replaces the placeholder $\bullet m$ with the process V_1, i.e. V_0 updated as needed, within the residual of the process which fired the input. In fact, all the names in process V_0 are correctly updated by routed substitution, via $\star^H$.

Figure 5.9 shows an example of computation.

$$HO\text{-}In : n \Rightarrow r(V).P \xrightarrow{r(\bullet n)} (n+1) \Rightarrow P\{\!|\bullet n/V|\!\}^{H}$$

$$HO\text{-}Com_0 : \frac{t_0 \xrightarrow{\overline{r_0}V_0@\vartheta_0} t'_0, t_1 \xrightarrow{r_1(\bullet m)@\vartheta_1} t'_1}{t_0|t_1 \xrightarrow{\tau} t'_0|t'_1\{\!|V_1/\bullet m|\!\}^{H}_{@\vartheta_1}}, \begin{cases} V_1 = ||_1\vartheta_1\bullet||_0\vartheta_0 \star^{H} V_0 \\ r_1 = ||_1\vartheta_1\bullet||_0\vartheta_0 \star r_0 \end{cases}$$

Table 5.2: Two rules for the higher-order case.

5.7 The Authentication Primitive

The operational semantics seen in Section 5.3 provides us with an authentication primitive that can be used to specify protocols which guarantee authentication "by construction".

Intuitively, our authentication primitive $[x \overset{@}{=} A]$ is akin to the matching operator of the π-calculus. The process $[x \overset{@}{=} A]P$ enables P only if the relative addresses of the two localized names x and A are exactly the same, i.e. only if x has been generated by A. [5] It is now simple to ascertain the origin of a message.

Consider again the example presented at the beginning of the chapter, on which we show in detail how our authentication primitive works. The protocol P is:

$$\begin{aligned} P &= A \mid (B \mid E) \\ A &= (\nu M)\overline{c}M.A' \\ B &= c(x).[x \overset{@}{=} A]B'(x) \end{aligned}$$

Now we show the role that localized names play, and how we guarantee by construction that B' is executed only if the message bound to x has been originated by A, i.e., if the message received on channel c is indeed M. Assume, for simplicity, that the name of A used by B in the authentication check is indeed the relative address $||_1||_0\bullet||_0$ pointing from B to A. As said above, this relative address encodes the "site" hosting A, thus it gives the identity of the process from which B is expecting to receive a message on channel c. In order to analyze the behaviour of the protocol in an hostile environment, we consider a generic enemy E, as powerful as possible. We now consider the following two messages:

$$\begin{aligned} \text{Message } 1 &: A \to B : M \\ \text{Message } 2 &: E \to B : N \end{aligned}$$

[5] We assume that a process knows which are the other processes around, and which are the relative addresses of their local environments. Note also that x is a name while A is a process: we do not distinguish them here for the sake of presentation.

The first message represents the correct exchange of message M from A to B. The second one is an attempt of E to send a different message N to B. The enemy E could actually be of the following form:

$$E = (\nu N)\overline{c}N.E' + \langle \text{other evil actions} \rangle.$$

The names M and N are received by B prefixed by the relative address corresponding to the respective originators.

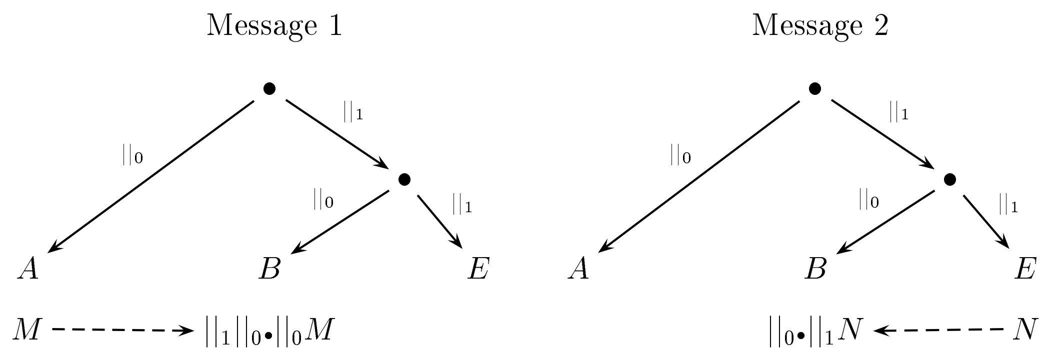

Figure 5.10: The process B detects that the message $||_1||_0\bullet||_0M$ is authentic while the message $||_0\bullet||_1N$ comes from the enemy E.

Fig. 5.10 shows the two message exchanges. In particular we see that M is received by B as $||_1||_0\bullet||_0M$ while N becomes $||_0\bullet||_1N$. It is now immediate to see that only in the first case B will evolve to $B(M)$ while in the latter it will stop. This is so because only $||_1||_0\bullet||_0M$ matches with the address of A. Every attempt of the enemy of introducing new messages on c is filtered out by the authentication primitive.

A further interesting case, that we analyse below, arises when the enemy, masquerading as B (written $E(B)$), intercepts M and forwards it to B. Thanks to address composition, our mechanism accepts the message, as authentic.

$$\begin{array}{lcl} \text{Message 3} & : & A \to E(B) : M \\ \text{Message 4} & : & E \to B : M \end{array}$$

Since E does not modify M we would like to accept the message in B even if it has been forwarded by E. Address composition maintains the identity of messages, so we accept indeed this behaviour. In detail, $E(B)$ receives M as $||_1||_1\bullet||_0M$. When E forwards it to B, the message is composed with the address of E relative to B, $||_0\bullet||_1$, yielding $||_0\bullet||_1 \star ||_1||_1\bullet||_0M$. By applying the rule (1) of Def. 5.2.3 we see that the message is received by B as $||_1||_0\bullet||_0M$ and that it is accepted as authentic.

As we have shown above, our primitive allows processes to obtain authentication in a direct and "secure" way. It thus permits to completely abstract from the underlying mechanisms which are necessary to guarantee authentication. However, if we want to implement it, the interpreter itself of the calculus (or of a language based on it) must control and manage relative addresses in a highly reliable manner. E.g., this is the case for intranets.

Unfortunately, this may be not the case in many other situations. Typically, in a wide-area network setting, the network is completely insecure, i.e., users may have a complete access to the net management system and there is no reliable way of filtering these accesses. In our terminology, a malicious user could change the relative addresses of localized names making them unreliable and our authentication primitive completely useless.

In these situations the "standard" approach to guarantee authentication (and security in general) is to use cryptography. This leads to the spi-calculus [7], that we consider in Chapter 6.

However, our approach can be useful for reasoning on authentication from an abstract point of view and for developing suitable proof techniques for authentication, based on equivalence. The idea, described in a more detailed way in Section 6.3, is to define an equivalence on processes comparing protocols and their corresponding "secure by construction" version (because of the authentication primitive).

Chapter 6

Authentication in the Spi-calculus

We consider here the mechanism of handling names locally, introduced in Chapter 5, for the the spi-calculus. Also in this case, the semantics provides us with a built-in primitive that enables the receiver of a message to ascertain its origin. The primitive for authentication is still $[x \overset{@}{=} A]P$, where x is a name and A is a process and allows the execution of P only when the check $[x \overset{@}{=} A]$ succeeds.

Also in this case, we assume that the implementation of the communication primitive has a reliable mechanism to control and manage relative addresses. Yet, our solution may help checking the correctness of different implementations, e.g. those based on cryptography. This leads to the spi-calculus, over which the mechanism of handling names locally carries easily. In this setting, we can then verify if a cryptography-based protocol ensures authentication, by checking a version of it containing also the primitive $[x \overset{@}{=} A]$: the original formulation and the extended one should exhibit the same behaviour. This check of specifications against implementations, is much in the style of the congruence-based technique proposed in [7]. In Sect. 6.2, we show on an example how an attack is revealed.

6.1 Localized Spi-Calculus

We carry on the localized semantics of π-calculus on the monadic spi-calculus. Since the spi-calculus has the π-calculus as 'core' language, we only give the necessary extensions. The main difference is in the creation of new names, that here is simplified, for the sake of presentation: names are created by a suitable function *Fresh*, able to generate a completely new and fresh name each time it is invoked. Such a function could be implemented, for instance, with a mechanism similar to that used in Chapter 5. Importing the same mechanism of Chapter 5 for ensuring the freshness of names is of course possible and quite easy: it suffices to endow sequential processes with counters. More relevant are the differences between the localized semantics for the π-calculus and that for the spi-calculus. The use of a semantics based on the commitment relation, implies to have a different point of creation of new names.

In the spi-calculus, names are only created by the restrictions including the process involved in an output action and not also by the output action itself, as in the π-calculus. When the rule *Comm Res* is applied to a process P with restriction (νa), sending a term M, that contains a, a new name n is created by *Fresh* and substituted in a "routed" way for a. Note that the new name n is known in all the scope of the restriction

Names created by *Fresh*, distinct from the standard names, can be prefixed with relative addresses. Localized names are therefore defined as follows.

Definition 6.1.1 *Let* $\mathcal{N}' = \mathcal{A} \cdot \mathcal{F} \cup \mathcal{N} \cup \{\tau\}$ *be the set of* localized names*, ranged over by* $r, s, u, \ldots$*, possibly indexed, where* $\mathcal{F}$*, ranged over by* n, m *is the set of* fresh names*,* $\mathcal{N}$*, ranged over by* a, b *is the set of* standard names*, and "$\cdot$" is the operator of language concatenation; we assume that* $\mathcal{F} \cap \mathcal{N} = \mathcal{F} \cap \{\tau\} = \mathcal{N} \cap \{\tau\} = \emptyset$*.*
The new syntax of the spi-calculus is obtained from the one in Def. 1.4.1, with localized names and localized terms, possibly composed with localized names.

Note that variables as standard names and as natural numbers are not prefixed with relative addresses. More precisely:

$$\forall z \in \mathcal{N} \cup \mathbb{N} \cup \mathcal{V} : \ \vartheta {\scriptstyle\bullet} \vartheta' \star z = z$$

The management of localized names extends easily to terms and in particular to the encryption and decryption primitives. The first raises no problems, because $\{M\}_K$ is handled as it were a *new* name, local to the process that encrypts M, say P.

The two localized names M and K are frozen just like they were at encryption time, and their own relative addresses are not changed when the encrypted message is sent through the net; actually, these relative addresses *cannot* be updated. Since $\{M\}_K$ is a new name, its relative address, say $\vartheta_0 {\scriptstyle\bullet} \vartheta_1$, will always point to the process P that made the encryption. Actually M and K are frozen like they were in the process S containing all the restrictions on the names used in the encryption. The process P is a sub-process of S. In this way, when decrypting $\{M\}_K$ the semantic rules recover the correct addresses for M and K by simply composing the actual address of $\{M\}_K$, $\vartheta_0 {\scriptstyle\bullet} \vartheta_1$, with the (frozen) relative addresses of M and K, respectively.

The same management described above is used for the other composed terms, namely for successor and pairs. In the first case, the term M is frozen in $suc(M)$, while the terms M and N are frozen in (M, N).

Address composition is naturally lifted to term address composition $\star^T$. It considers the composed term M as a name, consequently it does not distribute address composition over the sub-terms of M. This corresponds to the fact that the relative addresses of composed terms are frozen and no one can update them, when the term is sent through the net. A term is exported to a relative address, when address composition is applied to it.

Definition 6.1.2 Term address composition $\star^T$ *is defined by induction on terms*

- $\vartheta_\bullet\vartheta' \star^T z = \vartheta_\bullet\vartheta' \star z = z$, *for* $z \in \mathcal{N} \cup \cup \mathcal{V}$;
- $\vartheta_\bullet\vartheta' \star^T 0 = 0$;
- $\vartheta_\bullet\vartheta' \star^T \vartheta_{0\bullet}\vartheta_1 n = \vartheta_\bullet\vartheta' \star \vartheta_{0\bullet}\vartheta_1 n$, *for* $n \in \mathcal{F}$;
- $\vartheta_\bullet\vartheta' \star^T \vartheta_{0\bullet}\vartheta_1 M = \vartheta_\bullet\vartheta' \star \vartheta_{0\bullet}\vartheta_1 M$.

Definition 6.1.3 *A term M* exported *to the relative address $\vartheta_\bullet\vartheta'$ is $\vartheta_\bullet\vartheta' \star^T M$.*

Routed substitution is extended to deal both with terms and with processes in the spi-calculus. Note that routed substitution is distributed to each subterm.

Definition 6.1.4 *Let $[-/-]$ be the standard substitution. Then the* spi routed substitution $\{\!| - / - |\!\}_s$ *is defined by induction as follows on*

terms

1. $a\{\!|u'/u|\!\}_s = z[u'/u]$, *with* $z \in \mathcal{N} \cup \mathcal{V}$;
2. $n\{\!|u'/u|\!\}_s = n[u'/u]$, *with* $n \in \mathcal{F}$;
3. $0\{\!|u'/u|\!\}_s = 0$;
4. $suc(M)\{\!|u'/u|\!\}_s = suc(M\{\!|u'/u|\!\}_s)$;
5. $(M, N)\{\!|u'/u|\!\}_s = (M\{\!|u'/u|\!\}_s, N\{\!|u'/u|\!\}_s)$;
6. $\{M\}_N\{\!|u'/u|\!\}_s = \{M\{\!|u'/u_s|\!\}\}_{N\{\!|u'/u|\!\}_s}$;
7. $(M_1, \ldots, M_k)\{\!|u'/u|\!\}_s = (M_1\{\!|u'/u|\!\}_s, \ldots, M_k\{\!|u'/u|\!\}_s)$;

processes

1. $\mathbf{0}\{\!|u'/u|\!\}_s = \mathbf{0}$;
2. $(\overline{M}\langle N\rangle.P)\{\!|u'/u|\!\}_s = \overline{M}\{\!|u'/u|\!\}_s\langle N\{\!|u'/u|\!\}_s\rangle.P\{\!|u'/u|\!\}_s$
3. $(M(x).P)\{\!|u'/u|\!\}_s = M\{\!|u'/u|\!\}_s(x\{\!|u'/u|\!\}_s).P\{\!|u'/u|\!\}_s$;
4. $(P \mid Q)\{\!|u'/u|\!\}_s = P\{\!|u'/u|\!\}_s \mid Q\{\!|u'/u|\!\}_s$;
5. $((\nu n)P)\{\!|u'/u|\!\}_s = (\nu n\{\!|u'/u|\!\}_s)P\{\!|u'/u|\!\}_s$;
6. $([M\ is\ N]P)\{\!|u'/u|\!\}_s = [M\{\!|u'/u|\!\}_s\ is\ N\{\!|u'/u|\!\}_s]P\{\!|u'/u|\!\}_s$;
7. $(let\ (x, y) = M\ in\ P)\{\!|u'/u|\!\}_s =$
   $let\ (x, y)\{\!|u'/u|\!\}_s = M\{\!|u'/u|\!\}_s\ in\ P\{\!|u'/u|\!\}_s$;
8. $(case\ M\ of\ 0 : P\ suc(x) : Q)\{\!|u'/u|\!\}_s =$
   $case\ M\{\!|u'/u|\!\}_s\ of\ 0 : P\{\!|u'/u|\!\}_s\ suc(x)\{\!|u'/u|\!\}_s : Q\{\!|u'/u|\!\}_s$;

9. $(case\ L\ of\ \{x_1, \ldots, x_k\}_N\ in\ P)\{\!|u'/u|\!\}_s =$
$case\ L\{\!|u'/u|\!\}_s\ of\ \{x_1, \ldots, x_k\}_{N\{\!|u'/u|\!\}_s}\ in\ P\{\!|u'/u|\!\}_s;$

abstractions and concretions

1. $(x)P\{\!|u'/u|\!\}_s = (x\{\!|u'/u|\!\}_s)P\{\!|u'/u|\!\}_s;$
2. $((\nu n)\langle M\rangle P)\{\!|u'/u|\!\}_s = (\nu n\{\!|u'/u|\!\}_s)\langle M\{\!|u'/u|\!\}_s\rangle P\{\!|u'/u|\!\}_s.$

The selective routed substitution and the selective increment are extended in the obvious way.

Localized Congruence. As far as the structural congruence is concerned, we just give the rules for processes and agents, involving localized names, i.e.:

1. $[a]\ (\nu r)T_1|T_2 \equiv (\nu \bullet||_0 r)(T_1|T_2)$ if $\bullet||_0 r \notin fn(T_2)$, and
2. $[b]\ T_1|(\nu r)T_2 \equiv (\nu \bullet||_1 r)(T_1|T_2)$ if $\bullet||_1 r \notin fn(T_1)$.
3. $P \mid (\nu r)\langle M\rangle P' \equiv (\nu \bullet||_1 r)\langle \bullet||_1 \star^T M\rangle(P \mid P')$;
4. $(\nu r)\langle M\rangle P \mid P' \equiv (\nu \bullet||_0 r)\langle \bullet||_1 \star^T M\rangle(P \mid P')$.

assuming, respectively, that $\{\bullet||_1 r\} \cap fn(T) = \emptyset$ and $\{\bullet||_0 r\} \cap fn(T') = \emptyset$.
Note that (3) and (4) may require α-conversion.

Localized Reduction Relation. For the deconstructive rules, we stipulate the following reduction rules:

Red Split
$let\ (x, y) = \vartheta_0 \bullet \vartheta_1 (M, N)\ in\ P >$
$P\{\!|\vartheta_0 \bullet \vartheta_1 \star^T M/x|\!\}\{\!|\vartheta_0 \bullet \vartheta_1 \star^T N/y|\!\};$

Red Suc
$case\ \vartheta_0 \bullet \vartheta_1 suc(M)\ of\ 0 : P\ \vartheta_0 \bullet \vartheta_1 suc(x) : Q > Q\{\!|\vartheta_0 \bullet \vartheta_1 \star^T M/x|\!\};$

Red Decrypt $case\ \vartheta_0 \bullet \vartheta_1 \{M\}_u\ of\ \{y\}_{\vartheta_0 \bullet \vartheta_1 \star^T u}\ in\ P > P\{\!|\vartheta_0 \bullet \vartheta_1 \star^T M/y|\!\}.$

If a process is able to decompose a term, then the updating of single sub-terms can be performed. The intuition for the decryption rule is that we can decrypt a message $\{M\}_u$ only if we use the key $s = \vartheta_0 \bullet \vartheta_1 \star^T u$ which is exactly how the frozen key u should appear to the receiver of the encrypted message. If the key is shared and it is a standard name then $\vartheta_0 \bullet \vartheta_1 \star^T u = u$. When the two names s and u, although starting from different sites, do refer to the same key, the semantic rules decrypt the message and update its relative address by term composing $\vartheta_0 \bullet \vartheta_1$ with M. If the two keys match then the decryption succeeds and the process P behaves as $P\{\!|\vartheta_0 \bullet \vartheta_1 \star^T M/y|\!\}$.

Localized Commitment relation. Also interactions involve localized names. If F_0 is the extended abstraction $(x)P_0$ and C_1 the extended concretion $(\nu\tilde{\gamma})(\nu\tilde{r})\langle M\rangle P_1$ and, respectively, $\{\tilde{\gamma}\} \cup \{\bullet||_1 \star^T \tilde{r}\} \cap fn(P_1) = \emptyset$, and $\{\tilde{\gamma}\} \cup \{\bullet||_0 \star^T \tilde{r}\} \cap fn(P_1) = \emptyset$, the extended interactions $F_0 \hat{@} C_1$ and $C_1 \hat{@} F_0$ are:

$$F_0 \hat{@} C_1 = (\nu\tilde{\gamma})(\nu\bullet||_1 \star \tilde{r})(P_0\{\!|\, ||_0\bullet||_1 \star^T M/x |\!\} \mid P_1)$$

$$C_1 \hat{@} F_0 = (\nu\tilde{\gamma})(\nu\bullet||_0 \star \tilde{r})(P_1 \mid P_0\{\!|\, ||_1\bullet||_0 \star^T M/x |\!\})$$

Finally we present in Tab. 6.1 the extended commitments rules. The *localized commitment relation* is written $P \xrightarrow{\alpha @ \vartheta} A$, where P is a process, α is the subject of the action, ϑ the proof term and A is an agent. In the rule *Comm Res Loc* the function *Fresh* returns a value in $\mathcal{F}$.

Actually, the proved information ϑ on the labels here is only used in the rules for restrictions, to find the sub-process $\hat{P} = P_{@\vartheta}$ performing the action, in which the new name $\bullet n$ is installed as fresh. From $\hat{P}$ we obtain the complete action having α as a subject.

The same proved information is no longer necessary to compute the relative address of the term M in the interactions $F_0 \hat{@} C_1$ and $C_0 \hat{@} F_1$.

In fact, the term M sent by the sub-process $\hat{P}$, appears as $\bullet\vartheta_1 \star^T M$ in the concretion C_1 and must be substituted, in a routed way, to x, at top level, in the abstraction F_0. Actually it appears as $\bullet\vartheta_1 \star^T \hat{M}$, because M is frozen as $\hat{M}$ in the process with restrictions S, containing $\hat{P}$ (see (**) in the example considered in the next section). Therefore in F_0, x will be substituted by $||_0\bullet||_1 \star^T M$.

The computation of the address is done by routed substitution and by rules in items 3 and 4.

For the same reason, the substitution in *Comm Res Loc* is performed at top level.

6.2 Implementing Authentication

We analyze below the cryptographic "implementation" of authentication mechanisms specified in the spi-calculus.

We then observe if a particular authentication protocol is indeed able to "implement" our authentication primitive, i.e., if the cryptographic protocol is as strong in detecting names with an "incorrect" relative address as our secure-by-construction authentication protocol is.

Recall that in the spi-calculus, a composed term, such as an encryption M, are localized, i.e. its relative address, say $\vartheta_0\bullet\vartheta_1$, will always point to the process P that made the encryption. In this way, when decrypting $\{M\}_K$ the semantic rules recover the correct addresses for M and K by simply composing the actual address of $\{M\}_K$, $\vartheta_0\bullet\vartheta_0$, with the (frozen) relative addresses of M and K, respectively.

Comm Out

$$\frac{}{\overline{m}\langle M\rangle.P \xrightarrow{\overline{m}} (\nu)\langle M\rangle P}$$

Comm In

$$\frac{}{m(x).P \xrightarrow{m} (x)P}$$

Comm Inter 1

$$\frac{P_0 \xrightarrow{m@\vartheta_0} F_0 \quad P_1 \xrightarrow{\overline{m}@\vartheta_1} C_1}{P_0|P_1 \xrightarrow{\tau} F_0\hat{@}C_1}$$

Comm Inter 2

$$\frac{P_0 \xrightarrow{\overline{m}@\vartheta_0} C_0 \quad P_1 \xrightarrow{m@\vartheta_1} F_1}{P_0|P_1 \xrightarrow{\tau} C_0\hat{@}F_1}$$

Comm Par 1

$$\frac{P_0 \xrightarrow{\alpha@\vartheta} A_0}{P_0|P_1 \xrightarrow{\alpha@||_0\vartheta} A_0|P_1}$$

Comm Par 2

$$\frac{P_1 \xrightarrow{\alpha@\vartheta} A_1}{P_0|P_1 \xrightarrow{\alpha@||_1\vartheta} P_0|A_1}$$

Comm Res

$$\frac{P \xrightarrow{\alpha@\vartheta} A}{(\nu\gamma)P \xrightarrow{\alpha@\vartheta} (\nu\gamma)A}, \alpha \notin \{\gamma, \overline{\gamma}\}$$

if $P_{@\vartheta} = \overline{m}\langle M\rangle.P'$ then $\gamma, \overline{\gamma} \notin n(M)$

Comm Res Loc

$$\frac{P \xrightarrow{\overline{m}@\vartheta} A}{(\nu\gamma)P \xrightarrow{\overline{m}@\vartheta} (\nu_{\bullet}\vartheta n)A\{\!|_{\bullet}\vartheta n/\gamma|\!\},} \quad n = \mathit{Fresh},\ m \neq \gamma$$

if $P_{@\vartheta} = \overline{m}\langle M\rangle.P'$ then $\gamma, \overline{\gamma} \notin n(M)$

Comm Red

$$\frac{P > P' \quad P' \xrightarrow{\alpha@\vartheta} A'}{P \xrightarrow{\alpha@\vartheta} A'}$$

Comm Struct

$$\frac{P \equiv Q \quad Q \xrightarrow{\alpha@\vartheta} A \quad A \equiv A'}{Q \xrightarrow{\alpha@\vartheta} A'}$$

Table 6.1: The localized commitment relation.

Recall also the following reduction rule:

$$case\ \vartheta_0{\bullet}\vartheta_1\{M\}_u\ \ of\ \ \{y\}_{\vartheta_0\bullet\vartheta_1\star^T u}\ in\ P > P[\vartheta_0{\bullet}\vartheta_1 \star^T M/y].$$

We now show an example of a correct run of the Wide Mouthed Frog key exchange protocol, just considered in Example 3.3.9. The protocol should guarantee that when B receives M, such a message has been indeed originated by A. We use an atomic message M, i.e. a name or a variable, for the sake of simplicity.

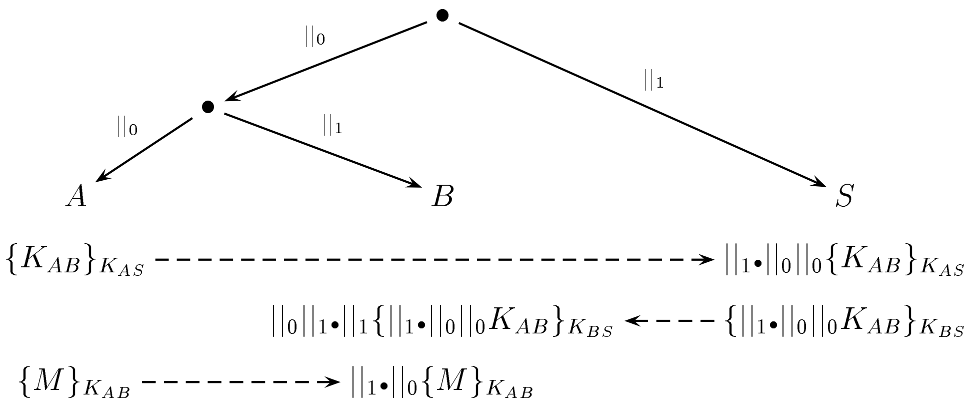

Figure 6.1: A correct execution of the Wide Mouthed Frog protocol

The protocol is composed of the following three messages:

$$\begin{array}{lll} \text{Message 1} & A \rightarrow S & : & \{K_{AB}\}_{K_{AS}} \\ \text{Message 2} & S \rightarrow B & : & \{K_{AB}\}_{K_{BS}} \\ \text{Message 3} & A \rightarrow B & : & \{M\}_{K_{AB}} \end{array}$$

Its specification in our calculus with localized names is:

$$\begin{array}{rcl} P & = & (\nu K_{AS})(\nu K_{BS})(\ (A|B)\ |\ S) \\ A & = & (\nu K_{AB})(\nu M)(\overline{c_{AS}}\langle\{K_{AB}\}_{K_{AS}}\rangle.\overline{c_{AB}}\langle\{M\}_{K_{AB}}\rangle) \\ S & = & c_{AS}(x).case\ x\ of\ \ \{y\}_{K_{AS}}\ in\ \overline{c_{BS}}\langle\{y\}_{K_{BS}}\rangle \\ B & = & c_{BS}(x).case\ x\ of\ \ \{y\}_{K_{BS}}\ in \\ & & c_{AB}(z).case\ z\ of\ \ \{r\}_y\ in\ B'(r) \end{array}$$

Not surprisingly, the specification is in the style of [7]. Indeed the localized name mechanism is not visible at this level, and we are not using yet our authentication primitive. Fig. 6.1 shows how the localized names are handled in a correct execution of the protocol P. Note that K_{AS} and K_{BS} are standard names, belonging to $\mathcal{N}$. Indeed, they are restricted in P but never sent in a output and therefore they never

assume the form of localized names. So the keys have no relative address; of course, no problem would arise if they were localized names and carry a relative address, instead.
For the sake of simplicity, we use the same names K_{AB} and M for the names restricted at the beginning and for the names generated and substituted to them. After the reception of message 2, B decrypts the received message

$$||_0||_1\bullet||_1\{||_1\bullet||_0||_0 K_{AB}\}_{K_{BS}}.\ (**)$$

This means computing

$$\begin{aligned} &\mathit{case}\ ||_0||_1\bullet||_1\{||_1\bullet||_0||_0 K_{AB}\}_{K_{BS}}\ \mathit{of}\ \ \{y\}_{K_{BS}}\ \mathit{in}\ c_{AB}(z). \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathit{case}\ z\ \mathit{of}\ \ \{r\}_y\ \mathit{in}\ B'(r), \end{aligned}$$

where $||_0||_1\bullet||_1 K_{BS} = K_{BS}$ because the key is a standard name in $\mathcal{N}$ (see Def. 5.3.2). The variable y is then set to

$$||_0||_1\bullet||_1 \star ||_1\bullet||_0||_0 K_{AB},$$

that results in $||_1\bullet||_0 K_{AB}$, by the rule (3) of Def. 5.2.3. This is indeed the correct reference to the key K_{AB} generated by A and installed in its local environment. In the last message B receives $||_1\bullet||_0\{M\}_{K_{AB}}$, and succeeds in decrypting it with $||_1\bullet||_0 K_{AB}$, obtaining $||_1\bullet||_0 M$. The addresses of M and of A relative to B are equal, so M is indeed authentic. This characterizes a "correct" execution.

It is well-known that an attack can take place over two sessions of the protocol above. Basically, it occurs when the enemy replays some messages of the first session in the second one. We follow below the formalization of [7], where this attack is analyzed. Note that in the single session illustrated above no problem arises instead, even if an enemy intercepts the message sent by A and then forwards it to B. Indeed the message received is the right one, because no one can alter it.

Now, we show that the attack above is immediately detected by an observer that can compare localized name, e.g. by using our authentication primitive $[x \overset{@}{=} A]$. For the sake of readability, we call A' and B' the two instances of A and B in the second session where A' is trying to send the message M' to B' using the session key K'_{AB}:

$$\begin{array}{lrcl}
\text{Message } 1 & A \to S & : & \{K_{AB}\}_{K_{AS}} \\
\text{Message } 2 & S \to B & : & \{K_{AB}\}_{K_{BS}} \\
\multicolumn{4}{c}{\langle E \text{ eavesdrops Message } 2\rangle} \\
\text{Message } 3 & A \to B & : & \{M\}_{K_{AB}} \\
\multicolumn{4}{c}{\langle E \text{ eavesdrops Message } 3\rangle} \\
\text{Message } 1' & A' \to S & : & \{K'_{AB}\}_{K_{AS}} \\
\text{Message } 2' & E(S) \to B' & : & \{K_{AB}\}_{K_{BS}} \\
\text{Message } 3' & E(A') \to B' & : & \{M\}_{K_{AB}}
\end{array}$$

The enemy eavesdrops the first session and then replays messages 2 and 3 in the second session (messages $2'$ and $3'$). The result is that B' receives a copy of M instead of one of M'.

The specification of the protocol is:

$$P' = (\nu K_{AS})(\nu K_{BS})(\ (A \mid A) \mid (B \mid B)\ ) \mid (S \mid E)$$

Note that A generates both M and K_{AB} as fresh names. So when we consider $A \mid A$, the two A's originate two different messages, say M and M', and two different keys, say K_{AB} and K'_{AB}. We need to modify S so that it can serve more sessions:

$$S = c_{AS}(x).case\ x\ of\ \ \{y\}_{K_{AS}}\ in\ \overline{c_{BS}}\langle\{y\}_{K_{BS}}\rangle.S$$

and we specify the enemy as:

$$E = c_{BS}(x).c_{AB}(y).\overline{c_{BS}}\langle x\rangle.\overline{c_{AB}}\langle y\rangle.$$

Note that the names c_{AB}, c_{AS} and c_{BS} are free, i.e., known to all the processes of P'.

Now we can observe the attack sequence in Fig. 6.2. Note in particular that when B' decrypts $||_1||_1\bullet||_0||_0\{M\}_{K_{AB}}$, it obtains $||_1||_1\bullet||_0||_0M$ which is a message originated from A and not from A', as it should be. Indeed, the address of A' relative to B' is $||_1||_1\bullet||_0||_1$, and the attack is detected.

6.3 Possible Developments

The same mechanism of our authentication primitive, can also be applied in case one is interested in making sure which principal encrypted the message received, under the hypothesis that the same entity also generated the message or has been delegated to send it. This is a somehow weaker form of authentication than the one considered here, which assumes that a key can be used by its trusted owners, only. We make use of this restricted version of authentication in the discussion below.

Note that our authentication primitive does not transform a public channel into a private one. Message authentication is guaranteed over public channels. It neither guarantees that the received messages have not been intercepted nor eavesdropped, i.e., it does not guarantee secrecy. Indeed, our primitive corresponds neither to a private channel in the basic π-calculus, nor to a cryptographic one in the spi-calculus. These appear to be too strong to model authentication alone, as they guarantee also secrecy.

As discussed before, our primitive may be not directly implementable. Indeed, one should have a low-level, highly reliable mechanism to manage localized names, which is quite unrealistic in many cases. Nevertheless, our proposal can help reasoning on authentication and security from an abstract point of view. We briefly

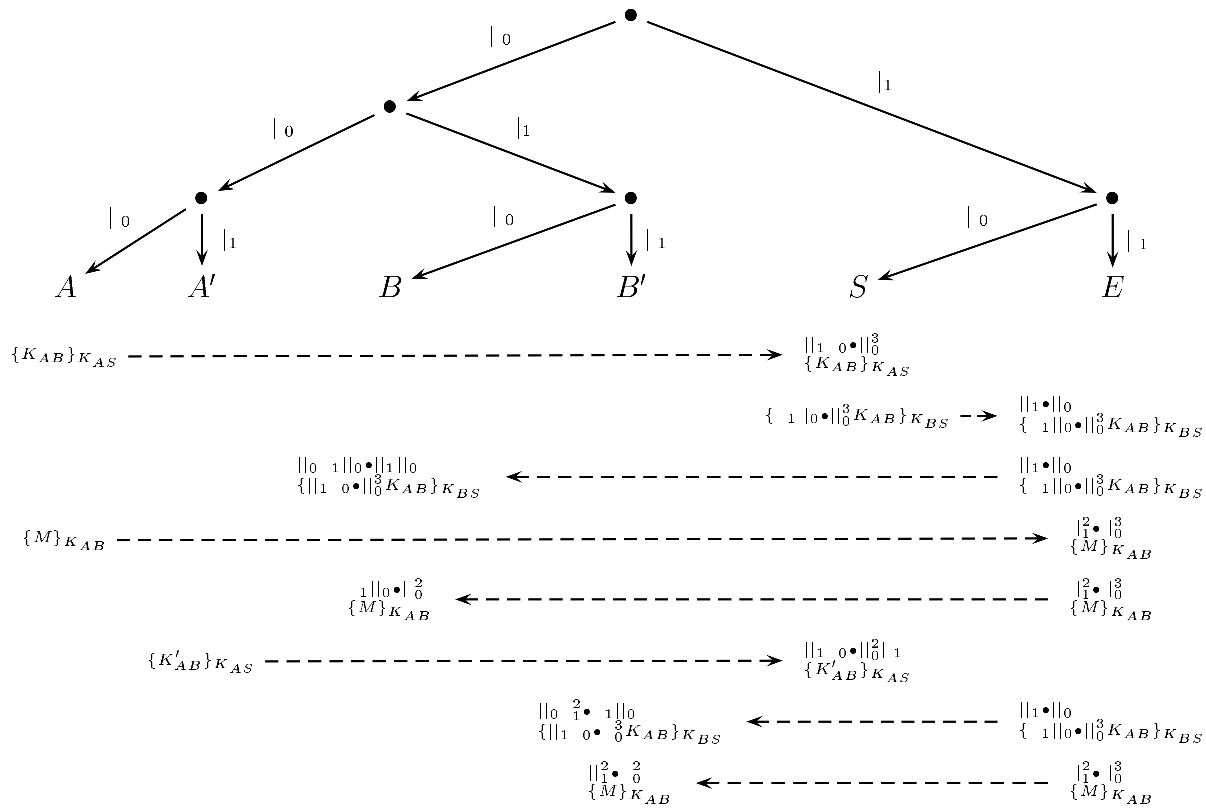

Figure 6.2: An attack on the Wide Mouthed Frog protocol; where $||_i^k$ stands for a sequence of k tags $||_i$.

exemplified this using the spi-calculus. We plan to apply the proof techniques presented in [7] to our model. The idea, just mentioned in the previous chapter, is to define a suitable equivalence $\approx$ on our processes, and then check if a certain protocol guarantees authentication by comparing it with its corresponding "secure by construction" version. This secure version is easily got by adding the authentication primitive at the end of the protocol, i.e., where the message which should be authentic is derived. As an example, the process B in the Wide Mouthed Frog protocol becomes:

$$\begin{aligned} B' &= c_{BS}(x).case\ x\ of\ \ \{y\}_{K_{BS}}\ in \\ &\qquad c_{AB}(z).case\ z\ of\ \ \{w\}_y\ in\ [w \overset{@}{=} A]B''(w) \end{aligned}$$

which makes the protocol secure by construction. So, if we establish that

$$(A|B)|S \approx (A|B')|S,$$

we are guaranteed that also the original protocol $(A|B)|S$ is secure.

Besides the technique mentioned above, our authentication primitive could support proofs of correctness that are split in a sequence of "small" steps. A simple example follows. Consider the transmission of an encrypted message M on a public channel in the spi-calculus:

$$\begin{aligned} P &= (\nu K_{AB})(\ (\nu M)\overline{c}\langle\{M\}_{K_{AB}}\rangle \mid \\ &\qquad c(x).case\ x\ of\ \ \{y\}_{K_{AB}}\ in\ B'(y)\ ), \end{aligned}$$

that enables $B'(y)$ only if $y = M$. Its secure version is:

$$\begin{aligned} P' &= (\nu K_{AB})(\ (\nu M)\overline{c}\langle\{M\}_{K_{AB}}\rangle \mid \\ &\qquad c(x).case\ x\ of\ \ \{y\}_{K_{AB}}\ in\ [y \overset{@}{=} A]B'(y)\ ) \end{aligned}$$

where we assume A to be (the address of) the process $(\nu M)\overline{c}\langle\{M\}_{K_{AB}}\rangle$. Instead of checking P against P', we could prove it is equivalent to

$$\begin{aligned} P'' &= (\nu K_{AB})\ (\nu M)(\overline{c}\langle\{M\}_{K_{AB}}\rangle \mid \\ &\qquad c(x).[x \overset{@}{=} A]case\ x\ of\ \ \{y\}_{K_{AB}}\ in\ B'(y)\ ) \end{aligned}$$

The check $P'' \approx P$ seems easier to prove than $P' \approx P$. Indeed, P'' makes explicit that P uses encryption to guarantee authentication: when B receives a message encrypted with K_{AB}, it knows that the encryption (not the encrypted message!) was originated by A. This constitutes the first step of the equivalence proof where we are still not interested in "what is encrypted". The next step consists of proving the equivalence $P'' \approx P'$, by observing that the only message which is encrypted with K_{AB} is actually originated by A. The steps above rely on the fact that K_{AB} is private to A and B, and exploit the weak notion of authentication mentioned above.

Of course, devising the intermediate processes, like P'' above, could be non trivial, but we are confident that also their generation could be mechanized.

We feel that our authentication primitive works as well, when processes migrate while executing a communication protocol. Our claim is supported by the fact that the localized semantics of the (first order) π-calculus is extended to the Higher Order π-calculus [114] as seen in Chapter 5.

Finally, we feel confident that our proposal scales up, because many real languages for concurrency, like Facile [123], PICT [105], CML [93], are built on top of a core process calculus like the one we use here. Of course a great deal of work is still necessary to make our theoretical proposal applicable in real cases. The relative addresses that we use to handle names locally are defined according to the abstract syntax of systems. Also, there is no difficulty in reflecting a different addressing mechanism when a more basic description of the topology of the network is available, as well as the actual placement of processes on processors. It suffices to take the space of relative addresses, quotient the relation induced by the topology and by the allocation. This relation will consider the logical access paths to the processes running on the same processor to be equivalent. Then, the address composition can be left unchanged and the quotient made after the needed calculations. This suggests a way to define a hierarchy of descriptions, which is increasingly nearer the actual implementation.

Chapter 7

Related Work

Authentication [66, 3] is one of the main issues in security and can have different purposes depending on the specific application considered. We briefly survey some proposals and refer the reader to [32]. For example, *entity authentication* is related to the verification of an entity's claimed identity [57], while *message authentication* should make it possible for the receiver of a message to ascertain its origin [118]. In recent years there have been some formalizations of these different aspects of authentication (see, e.g., [29, 7, 76, 117, 53]). These formalizations are crucial for proofs of authentication properties, that sometimes have been automatized (see, e.g., [55, 90, 75, 73, 52]). A typical approach presented in the literature is the following. First, a protocol is specified in a certain formal model. Then the protocol is shown to enjoy the desired properties, regardless of its operating environment, that can be unreliable, and can even harbour an hostile intruder. By introducing the notion of safe simplifying transformations, Lowe [77] follows, somehow, the opposite direction. These transformations, safe because they preserve insecurities, are used to simplify and verify large protocols like the commercial ones.

The idea of authentication followed in [7] seems slightly stronger than ours: it requires that if A has to send a message to B, then this should be the only message that B accepts at the end of (a successful run of) the protocol. Our approach to authentication relaxes this requirement. As a matter of fact, we only ask that the message received by B has been indeed originated by the process A. We have shown on an example how this feature can be exploited to reveal replay attacks. Further investigation is needed for comparing the two approaches. It would also interesting to inquiry the possibilities offered by static techniques such as Control Flow Analysis, as we begin to do in Chapter 3.

The idea of exploiting locations for the analysis of authentication comes from [54], where however entities are bound to physical addresses of the net. Here we relaxed the rigidity of that mapping, by introducing a sort of "identifiers of sites" represented by relative addresses. As a matter of fact, the actual placement of a process on a site can be recovered by composing our localized names (akin to the environment function of sequential languages) with allocation tables (similar to a store). Alternatively, one could have used absolute names, as those used in Join-Calculus [59]. Actually, [54] models a wider notion of authentication, that we plan to investigate next.

Conclusions

We have used well established theories and techniques to venture in the security jungle of concurrent and distributed systems.

We studied security properties of systems, in the process algebraic framework of the π-calculus and of the spi-calculus, on the common basis of their operational semantics using static and dynamic techniques. This dichotomy corresponds to the structure of the thesis.

In fact, in the first part, we presented a Control Flow Analysis both for the π-calculus and the spi-calculus, statically predicting safe and computable approximations or solutions to the set of values that the objects of processes may assume during the execution. Then, we showed that all solutions are semantically correct and a best solution always exists. Remarkably, for the π-calculus, we offered a constructive and low-polynomial procedure to build solutions.

For each property to verify, we then defined a static test on approximations that guarantees, if passed, that the analysed process will never violate the security requirements at run time. Properties of secrecy, of control of information flow and of message authentication are considered. Moreover, we compared our notion of secrecy with Abadi's, throwing some light on the relationship between the various notions and techniques for secrecy present in the literature.

In the second part, we relied instead on certain forms of enhanced operational semantics, again for both calculi, that considers names and terms as being *localized* to their owners. This semantics offers a built-in mechanism that guarantees (a form of) *message authentication*.

We chose the pure framework of process algebras and a small set of simple security properties to begin our study on security. In this way, we experimented with the validity of our analysis and techniques. Furthermore, even though yet too abstract, the used calculi powerfully describe communications among processes, that are quite delicate and attack-prone activities of processes, from the security point of view.

The light of theory can now be turned on real systems, that surely deserve more attention and whose deeper study may only broaden our perspective on security and develop the potentialities of our research. Several lines of work remain open and can benefit from tighter contacts with the security problems arising in the real world.

Static techniques can be refined to make more accurate analyses: they can be enriched indeed with contextual or data flow information.

These techniques can also exploit other forms of enhanced semantics that can help looking in different and new ways to the same problems. They can consider other aspects connected with security, such as the probability of security violations, the communication topology or the information about causality between actions. The development of efficient tools, both for the analysis and for the implementation of prototypes, is necessary for any further investigation.

Another line to follow is to consider other properties, like those connected with access control and explicit capabilities.

Moreover, we should test the robustness of our approach on other calculi, possibly closer and closer to real languages.

The ideal line underlying our work relies on the convinction that security should directly influence the design of programming languages. This way of thinking is intended to foster the development of new paradigms, where security could play a more significant role.

Bibliography

[1] M. Abadi. Secrecy by typing in security protocols. In *Proceedings of Theoretical Aspects of Computer Software 1997, LNCS 1281*, pages 611–638. Springer-Verlag, 1997. Extended version to appear in *Journal of the ACM.*

[2] M. Abadi. Protection in programming-language translations. In *Proceedings of ICALP'98, LNCS 1443*, pages 868–883. Springer-Verlag, 1998.

[3] M. Abadi. Two facets of authentication. In *Proceedings of* 11^{th} *IEEE Computer Security Foundations Workshop*, pages 25–32. IEEE press, 1998.

[4] M. Abadi. Security protocols and specifications. In *Proceedings of FoSSaCS'99, LNCS 1578*, pages 1–13. Springer-Verlag, 1999.

[5] M. Abadi, A. Banerjee, N. Heintze, and J. G. Riecke. A core calculus of dependency. In *Proceedings of POPL'99*, pages 147–160, New York, N.Y., 1999. ACM Press.

[6] M. Abadi, C. Fournet, and G. Gonthier. Secure implementation of channel abstractions. In *Proceedings of LICS'98.* IEEE, Computer Society Press, 1998.

[7] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols - The Spi calculus. *Information and Computation 148*, 1:1–70, January 1999.

[8] M. Abadi and A.D. Gordon. A calculus for cryptographic protocols - The Spi calculus. In *Proceedings of* 4^{th} *ACM Conference on Communications and Computer Security*, pages 36–47. ACM press, 1997.

[9] G.A. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, 1988.

[10] R.M. Amadio. An asynchronous model of locality, failure, and process mobility. In *Proceedings of COORDINATION'97, LNCS 1282*, pages 374–391. Springer-Verlag, 1997.

[11] J.S. Ambler. A de Bruijn notation for the π-calculus. Technical Report 569, Dept. of Computer Science, Queen Mary & Westfield College, London, 1991.

[12] D. Austry and G. Boudol. Algébre de processus et synchronization. *Theoretical Computer Science*, 30:91–131, 1984.

[13] J. Banâtre, C. Bryce, and Le Métayer. Compile-time detection of information flow in sequential programs. In *Proceedings of* 3^{rd} *European Symposium on Research in Computer Security, LNCS 875*. Springer-Verlag, 1994.

[14] D.E. Bell and L.J. LaPadula. Secure computer systems: Unified exposition and Multics interpretation. Technical Report ESD-TR-75-306, Mitre C., 1976.

[15] J. Bergstra and J. Klop. Process algebra for synchronous communication. *Information and Computation*, 61:109–137, 1984.

[16] C. Bodei, P. Degano, R. Focardi, and C. Priami. Authentication via localized names. In *Proceedings of* 12^{th} *Computer Security Foundations Workshop*, pages 98–110. IEEE CS Press, 1999.

[17] C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Static analysis for the π-calculus with their application to security. To appear in *Information and Computation*. Also available at http://www.di.unipi.it/∼chiara/publ-40/BDNNi00.ps.

[18] C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Control flow analysis for the π-calculus. In *Proceedings of CONCUR'98, LNCS 1466*, pages 84–98. Springer-Verlag, 1998.

[19] C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Static analysis of processes for no read-up and no write-down. In *Proceedings of* 2^{nd} *FoSSaCS'99, LNCS 1578*, pages 120–134. Springer-Verlag, 1999.

[20] C. Bodei, P. Degano, and C. Priami. Names of mobile agents handled locally. To appear in *Theoretical Computer Science*. Also available at http://www.di.unipi.it/∼chiara/publ-40/TCS.ps.

[21] C. Bodei, P. Degano, and C. Priami. Mobile processes with a distributed environment. In *Proceedings of ICALP'96, LNCS 1099*, pages 490–501. Springer-Verlag, 1996.

[22] C. Bodei, P. Degano, and C. Priami. Constructing specific SOS semantics for concurrency via abstract interpretation. In *Proceedings of SAS'98, LNCS 1503*, pages 168–183, 1998.

[23] C. Bodei and C. Priami. True concurrency via abstract interpretation. In *Proceedings of SAS'97, LNCS 1302*, pages 202–216, 1997.

[24] M. Boreale and R. De Nicola. Testing equivalence for mobile processes. *Information and Computation*, 120(2):279–303, August 1995.

[25] M. Boreale and D. Sangiorgi. A fully abstract semantics of causality in the π-calculus. In *Proceedings of STACS'95, LNCS*. Springer Verlag, 1995.

[26] G. Boudol and I. Castellani. A non-interleaving semantics for CCS based on proved transitions. *Fundamenta Informaticae*, XI(4):433–452, 1988.

[27] G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn. A theory of processes with localities. *Theoretical Computer Science*, 114, 1993.

[28] S.D. Brookes, C.A.R. Hoare, and A.D. Roscoe. A theory of communicating sequential processes. *Journal of ACM*, 31(4):560–599, 1984.

[29] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*, pages 18–36, February 1990.

[30] L. Cardelli and A.D. Gordon. Mobile ambients. In *Proceedings of FoSSaCS'98, LNCS 1378*, pages 140–155. Springer-Verlag, 1998.

[31] L. Cardelli and A.D. Gordon. Types for mobile ambients. In *Proceedings of POPL'99*, pages 79–92. ACM Press, 1999.

[32] E.M. Clark and J. Jacob. *A survey of authentication protocol literature*. 1997.

[33] R. Cleaveland and D. Yankelevich. An operational framework for value-passing processes. In *Proceedings of POPL'94*, pages 326–338, 1994.

[34] P. Cousot. Program analysis: the abstract interpretation perspective. *ACM Computing Surveys*, 28A(4), 1996.

[35] P. Cousot. Types as abstract interpretations. In *Proceedings of POPL '97*, pages 316–331. ACM Press, 1997.

[36] P. Cousot and R. Cousot. Systematic Design of Program Analysis Frameworks. In *Proceedings of POPL '79*, pages 269–282. ACM Press, 1979.

[37] M. Dam. Proving trust in systems of second-order processes: (extended abstract). In *Proceedings of HICSS 31*, volume VII, pages 255–264. IEEE Computer Science, 1998.

[38] Ph. Darondeau and P. Degano. Causal trees. In *Proceedings of ICALP'89, LNCS 372*, pages 234–248. Springer-Verlag, 1989.

[39] J.W. de Bakker and W.P. de Roever. A calculus for recursive program schemes. In *Proceedings of ICALP'72*, pages 167–196. North-Holland, 1972.

[40] R. De Nicola, G. Ferrari, and R. Pugliese. Types for access control. To appear in *Theoretical Computer Science*.

[41] R. De Nicola, G. Ferrari, and R. Pugliese. Klaim: a kernel language for agents interactions and mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, 1998.

[42] R. De Nicola and M.C.B. Hennessy. Testing equivalence for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[43] P. Degano, R. De Nicola, and U. Montanari. Partial ordering derivations for CCS. In *Proceedings of FCT, LNCS 199*, pages 520–533. Springer-Verlag, 1985.

[44] P. Degano and U. Montanari. Concurrent histories: A basis for observing distributed systems. *Journal of Computer and System Sciences*, 34(2):411–449, 1987.

[45] P. Degano and C. Priami. Proved trees. In *Proceedings of ICALP'92, LNCS 623*, pages 629–640. Springer-Verlag, 1992.

[46] P. Degano and C. Priami. Enhanced operational semantics. *ACM Computing Surveys*, 28(2):352–354, 1996.

[47] P. Degano and C. Priami. Non interleaving semantics for mobile processes. *Theoretical Computer Science*, 1-2(216), March 1999.

[48] P. Degano and C. Priami. Non interleaving semantics for mobile processes. *Theoretical Computer Science*, 1-2(216), March 1999.

[49] D. E. Denning. A Lattice Model of Secure Information Flow. *Communications of the ACM*, pages 236–243, May 1976.

[50] D. E. Denning and P. J. Denning. Certification of Programs for Secure Information Flow. *Communications of the ACM*, pages 504–513, July 1977.

[51] D.E Denning. *Cryptography and Data Security.* Addison-Wesley, Reading Mass., 1982.

[52] A. Durante, R. Focardi, and R. Gorrieri. CVS: A compiler for the analysis of cryptographic protocols". In *Proceedings of* 12^{th} *Computer Security Foundations Workshop*. IEEE CS Press, 1999.

[53] F. J. T. Fabrega, J. C. Herzog, and J. D. Guttman. Strand spaces: Why is a security protocol correct? In *Proceedings of the 1998 Symposium on Security and Privacy*. IEEE Computer Society Press, May 1998.

[54] R. Focardi. Using entity locations for the analysis of authentication protocols. In *Proceedings of* 6^{th} *Italian Conference on Theoretical Computer Science (ICTCS'98)*, November 1998.

[55] R. Focardi, A. Ghelli, and R. Gorrieri. Using non interference for the analysis of security protocols. In *Proceeding of the DIMACS Workshop on Design and Formal Verification of Security Protocols*, DIMACS Center, Rutgers University, September 1997.

[56] R. Focardi and R. Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering*, 23(9), September 1997.

[57] International Organization for Standardization. Information technology - Security techniques - Entity authentication machanism; Part 1: General model. ISO/IEC 9798-1, Second Edition, September 1991.

[58] C. Fournet and Gonthier. The reflexive chemical abstract machine and the join calculus. In *Proceedings of POPL'96*, 1996.

[59] C. Fournet, G. Gonthier, J-J. Levy, L. Maranget, and D. Remy. A calculus of mobile agents. In *Proceedings of CONCUR'96, LNCS 1119*, pages 406–421. Springer-Verlag, 1996.

[60] C. Fournet, C. Laneve, L. Maranget, and D. Remy. Implicit typing à la ML for the join calculus. In *Proceedings of CONCUR'97, LNCS 1243*, pages 196–212. Springer-Verlag, 1997.

[61] K.L.S. Gasser, F. Nielson, and H.R. Nielson. Systematic realisation of control flow analysis for CML. In *Proceedings of ICFP'97*, pages 38–51. ACM Press, 1997.

[62] A. Giacalone, P. Mishra, and S. Prasad. Operational and algebraic semantics for Facile: A symmetric integration of concurrent and functional programming. In *Proceedings of ICALP'90, LNCS 443*, pages 765–780. Springer-Verlag, 1990.

[63] D. Gilbert and C. Palamidessi. A calculus of process migration. Technical Report CSE Tech. Rep., The Pennsylvania State University, 1998.

[64] J.A. Goguen and J. Meseguer. Security policy and security models. In *Proceedings of the 1982 IEEE Symposium on Research on Security and Privacy*, pages 11–20. IEEE Press, 1982.

[65] J.A. Goguen and J. Meseguer. Unwinding and inference control. In *Proceedings of the 1984 IEEE Symposiom on Security and Privacy*, pages 75–87. IEEE Computer Society, 1984.

[66] D. Gollmann. What do we mean by Entity Authentication. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 46–54. IEEE Computer Society Press, 1996.

[67] R. R. Hansen, J. G. Jensen, F. Nielson, and H. R. Nielson. Abstract interpretation of mobile ambients. In *Proceedings of SAS'99, LNCS 1694*, pages 135–148, 1999.

[68] J. He and V. Gligor. Formal methods and automated tool for timing-channels identification in TCS source code. In *Proceedings of the* 2^{nd} *European Symposium on Research in Computer Security*, 1992.

[69] N. Heintze. Control-flow analysis and type systems. In *Proceedings of SAS'95*, pages 189–206, 1995.

[70] N. Heintze and J.G Riecke. The SLam calculus: Programming with secrecy and integrity. In *Proceedings of POPL'98*, pages 365–377. ACM Press, 1998.

[71] K. Honda and M. Tokoro. A small calculus for concurrent objects. *OOPS Messenger*, 2(2):50–54, 1991.

[72] S. Jagannathan and S. Weeks. A unified treatment of flow analysis in higher-order languages. In ACM, editor, *Proceedings of POPL'95*, pages 393–407, New York, NY, USA, 1995. ACM Press.

[73] R. Kemmerer, C. Meadows, and J. Millen. "Three systems for cryptographic protocol analysis". *J. Cryptology*, 7(2):79–130, 1994.

[74] T. Lindholm and F. Yellin. *The Java Virtual Machine Specification*. Addison-Wesley, 1996.

[75] G. Lowe. Breaking and Fixing the Needham-Schroeder Public-key Protocol using FDR. In *Proceedings of TACAS'96*, pages 146–166, Passau (Germany), March 1996. Springer-Verlag, LNCS 1055.

[76] G. Lowe. A hierarchy of authentication specification. In *Proceedings of the* 10^{th} *Computer Security Foundation Workshop*. IEEE press, 1997.

[77] G. Lowe. Safe simplifying transformations for security protocols. In *Proceedings of the* 12^{th} *Computer Security Foundation Workshop*, pages 32 –43. IEEE press, 1999.

[78] G. Lowe and B.B. Roscoe. Using CSP to detect errors in the TMN protocol. *IEEE Transactions on Software Engineering*, 23(10):659–669, 1997.

[79] P. Lucas. Formal definition of programming languages and systems. In Springer Verlag, editor, *Proceedings of IFIP'71*, 1971.

[80] S. Mac Lane. *Categories for the Working Mathematician*. Springer-Verlag, 1971.

[81] P. Malacaria and C. Hankin. Non-deterministic games and program analysis: An application to security. In *Proceedings of LICS'99*, pages 443–452, 1999.

[82] J. McCarthy. Towards a mathematical science of computation. In C.M. Popplewell, editor, *Information Processing 1962*, pages 21–28, 1963.

[83] D. McCullough. Specifications for multi-level security and hook-up property. In *Proceedings of the 1987 IEEE Symposium on Research on Security and Privacy*. IEEE Press, 1987.

[84] D. McCullough. Noninterference and the composability of security properties. In *Proceedings of the 1988 IEEE Symposium on Research on Security and Privacy*, pages 177–186. IEEE Press, 1988.

[85] R. Milner. *Communication and Concurrency*. Prentice-Hall, London, 1989.

[86] R. Milner. The polyadic π-calculus: a tutorial. Technical Report ECS-LFCS-91-180, University of Edinburgh, 1991.

[87] R. Milner, D. Berry, and D. Turner. A semantics for ML concurrency primitives. In *Proceedings of POPL'92*, 1992.

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

[89] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *TCS*, 114:149–171, 1993.

[90] J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murϕ. In *Proceedings of the 1997 IEEE Symposium on Research in Security and Privacy*, pages 141–153. IEEE Computer Society Press, 1997.

[91] A.C. Myers and B. Liskov. A decentralized model for information flow control. In *Proceedings of the ACM Symposium on Operating System Principles*, 1997.

[92] G. Necula. Proof-carrying code. In *Proceedings of POPL'97*, pages 106–119. ACM Press, 1997.

[93] F. Nielson and H. R. Nielson. From CML to its process algebra. *Theoretical Computer Science*, 155:179–219, 1996.

[94] F. Nielson and H. R. Nielson. Infinitary control flow analysis: a collecting semantics for closure analysis. In *Proceedings of POPL'97*. ACM Press, 1997.

[95] F. Nielson and H. R. Nielson. Flow logics and operational semantics. *Electronic Notes of Theoretical Computer Science*, 10, 1998.

[96] F. Nielson and H. R. Nielson. Interprocedural control flow analysis. In *Proceedings of ESOP'99*, number 1576 in Lecture Notes in Computer Science, pages 20–39. Springer-Verlag, 1999.

[97] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer, 1999.

[98] F. Nielson, H. R. Nielson, R. R. Hansen, and J. G. Jensen. Validating firewalls in mobile ambients. In *Proc. Concur'99, LNCS 1664*, pages 463–477, 1999.

[99] H. R. Nielson and F. Nielson. Flow logics for constraint based analysis. In *Proc. CC'98*, number 1383 in Lecture Notes in Computer Science, pages 109–127. Springer-Verlag, 1998.

[100] H. R. Nielson and F. Nielson. Shape analysis for mobile ambients. In *Proceedings of POPL2000*. ACM Press, 2000.

[101] P. Ørbæk and J. Palsberg. Trust in the λ-calculus. *Journal of Functional Programming*, 7(6):557–591, nov 1997.

[102] J. Palsberg and M.J. Schwartzbach. *Object-Oriented Type Systems.* Wiley, 1994.

[103] D. Park. Concurrency and automata on infinite sequences. In *Proceedings of GI, LNCS 104*, pages 167–183. Springer-Verlag, 1981.

[104] B.C. Pierce and D. Sangiorgi. Typing and sub-typing for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996.

[105] B.C. Pierce and D.N. Turner. PICT: A programming language based on the pi-calculus. In Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. mit, 1999.

[106] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, Denmark, 1981.

[107] T.W. Pratt. *Programming Languages: Design and Implementation.* Prentice Hall, 1984.

[108] C. Priami. *Enhanced Operational Semantics for Concurrency.* PhD thesis, Dipartimento di Informatica, Università di Pisa, March 1996. Available as Tech. Rep. TD-08/96.

[109] W. Reisig. Petri nets: An introduction. In *EATCS Monographs*. Springer-Verlag, 1985.

[110] J. Reppy. *Higher order concurrency.* PhD thesis, Cornell University, TR 92-1285, 1992.

[111] J. Riely and M. Hennessy. A typed language for distributed mobile processes. In *Proceedings of POPL'98*, pages 378–390. ACM Press, 1998.

[112] J. Riely and M. Hennessy. Trust and partial typing in open systems of mobile agents. In *Proceedings of POPL'99*, pages 93–104. ACM Press, 1999.

[113] A. Sabelfeld and D. Sands. A per model of secure information flow in sequential programs. *Lecture Notes in Computer Science*, 1576:40–58, 1999.

[114] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD thesis, University of Edinburgh, 1992.

[115] D. Sangiorgi. Locality and non-interleaving semantics in calculi for mobile processes. In *Proceedings of TACS'94, LNCS 789*. Springer-Verlag, 1994.

[116] S. Schneider. Security properties and CSP. In *Proceedings of the IEEE 1996 Symposium on Research in Security and Privacy*, pages 174–187. IEEE Computer Society Press, 1996.

[117] S. Schneider. Verifying authentication protocols in CSP. *IEEE Transactions on Software Engineering*, 24(9), September 1998.

[118] B. Schneier. *Applied Cryptography.* John Wiley & Sons, Inc., 1996. second edition.

[119] P. Sewell. Global/local subtyping and capability inference for a distributed π-calculus. In *Proceedings of ICALP'98, LNCS 1443*, pages 695–706. Springer-Verlag, 1998.

[120] O. Shivers. Control flow analysis in Scheme. In *Proceedings of PLDI'88*, volume 7(1). ACM SIGPLAN Notices, 1988.

[121] DES. Fed. Inform. Processing Standards. Data encryption standard. Pub.46, National Bureau of Standards, Washington DC, January 1977.

[122] B. Thomsen. Plain CHOCS: a second generation calculus for higher order processes. *Acta Informatica*, 30(1):1–59, 1993.

[123] B. Thomsen, L. Leth, S. Prasad, T.-M. Kuo, A. Kramer, F. Knabe, and A. Giacalone. Facile Antigua Release Programming Guide. Technical Report ECRC-93-20, European Computer-Industry Research Centre, 1993.

[124] A. Venet. Abstract interpretation of the π-calculus. In *Analysis and Verification of Multiple-Agent Languages, LNCS 1192*, pages 51–75. Springer-Verlag, 1997.

[125] A. Venet. Automatic determination of communication topologies in mobile systems. In *Proceedings of SAS'98, LNCS 1503*, pages 152–167. Springer-Verlag, 1998.

[126] D. Volpano and G. Smith. Language Issues in Mobile Program Security. In *Mobile Agent Security, LNCS 1419*, pages 25–43. Springer-Verlag, 1998.

[127] D. Volpano and G. Smith. Probabilistic noninterference in a concurrent language. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, pages 34–43. IEEE, 1998.

[128] D. Volpano and G. Smith. Secure information flow in a multi-threaded imperative language. In *Proceedings of POPL'98*, pages 355–364. ACM Press, 1998.

[129] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4:4–21, 1996.