Trust and Mobility

Marco Carbone

PhD Dissertation

Department of Computer Science
University of Aarhus
Denmark

Trust and Mobility

A Dissertation
Presented to the Faculty of Science
of the University of Aarhus
in Partial Fulfilment of the Requirements for the
PhD Degree

by
Marco Carbone
April 5, 2005

To my Mum and Dad
To my uncle Pietro

Abstract

This PhD dissertation develops formal models for studying large scale systems which make use of the human notion of trust. Mobility is the main formalism for such a purpose.

We shall begin by introducing the main source of inspiration of this dissertation, the SECURE project, funded by EU IST FET with the aim of designing a novel security approach that addresses the challenges brought forward by trust. If successful, this approach will significantly benefit not only future systems but also various emerging mobile computing applications.

The first technical contribution of the dissertation is the study of *trust domains*, a formal model of trust informed by the Global Computing scenario and focusing on the aspects of trust formation, evolution, and propagation. The model is based on a novel notion of trust structures which, building on concepts from trust management and domain theory, feature at the same time a trust and an information partial order.

We then proceed by introducing `ctm`, a *Calculus for Trust Management*. `ctm` is a powerful calculus where it is possible to model the behaviour of trust-based systems. In `ctm` each principal (location) is equipped with a policy, which determines its legal behaviour, and with a protocol, which allows interactions between principals and the flow of information from principals to policies. We elect to formalise policies with a general framework which embeds many existing policy languages including the one introduced with trust domains, and to express protocols in the process algebra style. This yields an expressive calculus very suitable for the global computing scenarios, and provides a formalisation of notions such as trust evolution. For ctm we define barbed equivalences and study their possible applications, and, furthermore, we provide a proof method for such equivalences based on bisimulations.

The second part of the dissertation is about the formal study of an extension of communication introduced together with `ctm`, called *polyadic synchronisation*, i.e. a generalisation of the communication mechanism which allows channel names to be composite. We show that this operator embeds nicely in the whole theory, we suggest that it permits divergence-free encodings of distributed calculi, and we show that a limited form of polyadic synchronisation can be encoded weakly. After showing that matching cannot be derived in π-like calculi, we compare the expressivity of polyadic synchronisation, mixed choice and matching. In particular we show that the degree of synchronisation of a language increases its expressive power by means of a separation result in the style of Palamidessi's result for mixed choice. This part ends with an overview of a possible type system for polyadic synchronisation.

Acknowledgements

Just kidding, no hard feelings
— Frank Darwin Valencia, 2003.

I really would like to hugely thank my supervisor, Mogens Nielsen, who has always believed in me, and never given up encouraging me, even in the most difficult moments of my PhD studies.

Another big thank you goes to Vladimiro Sassone, who has co-supervised me since my "abroad experience" spent in Falmer, at University of Sussex, and still, never stops giving me very useful suggestions.

I'm indebted to Sergio Maffeis, companion of many adventures, who "walked" with me since the undergraduate studies but also co-authored some of the work that can be found in this dissertation: thank you Sergio!

I would like to say thanks to the SECURE project and its people. The fact that the project started in the same time I started my PhD studies has been a wonderful opportunity for new and great ideas. I have to say that I was very lucky for that. Huge thanks to the friends from Cambridge (Ken, Jeane, Andy, Brian, Daniel, David, Nathan), the friends from Glasgow (Waleed, Sotirios, Colin, Paddy and Helen), the friends from Trinity College in Dublin (Vinny, Chen, Christian, Jean-Marc and of course Liz) and the friends from Geneva (Ciran and Giovanna). And of course the ones from Århus, i.e. Mogens and Karl Krukow.

I can never forget the fellow PhD students at BRICS, from the old ones (Claus, Daniele, Frank, Giuseppe, Jiri, Mikkel, Paulo and Emanuela, and Pawel) to the ones I started with (Jesus, Kirill and Sunil), the new ones (Chris, Darok, Gabriel, Gosia, Johann, Saurabh) and all other BRICS and visiting PhD Students. Thank you all, it's been great!

Thanks, for the wonderful moments spent together, to Gianga, Martin and all tha italian speakin' cru in Århus! Aight!

Thanks to all the BRICS/DAIMI employees, in particular Janne, Karen, Lene and Uffe, but also Ingrid, Hanne and the new Ellen.

Well, we are almost at the end. Of course I'm going to say "GRAZIE" to my family who has suffered with me the whole pain of 9 years at university, morally but also economically. Grazie Mamma, Papà e Carmelo. And of course I will never stop thanking my uncle, Pietro. It was him who pushed me into computer stuff, and that's why I started studying Computer Science.

Marco Carbone,
London, April 5, 2005.

List of Publications

1. M. Carbone and S. Maffeis. On the expressive power of polyadic synchronisation in π-calculus. In *EXPRESS '02*, volume 68.2 of *ENTCS*. Elsevier Science Publishers, 2002.

2. W. Wagealla, M. Carbone, C. English, S. Terzis, H. Lowe and P. Nixon. A Formal Model for Trust Lifecycle Management. In Proc. of the *1st International Workshop on Formal Aspect of Security and Trust (FAST)*, Pisa, Italy, 2003.

3. M. Carbone, M. Nielsen, and V. Sassone. A formal model for trust in dynamic networks. In *International Conference on Software Engineering and Formal Methods (SEFM'03)*. IEEE, 2003.

4. M. Carbone and S. Maffeis. On the expressive power of polyadic synchronisation in π-calculus. *Nordic Journal of Computing (NJC)*, 10(2), September 2003.

5. V. Cahill, E. Gray, J.-M. Seigneur, C. Jensen, Y. Chen, B. Shand, N. Dimmock, A. Twigg, J. Bacon, C. English, W. Wagealla, S. Terzis, P. Nixon, G. Serugendo, C. Bryce, M. Carbone, K. Krukow, and M. Nielsen. Using Trust for Secure Collaboration in Uncertain Environments. *IEEE Pervasive Computing Magazine*, 2(3):52–61, 2003.

6. M. Carbone, M. Nielsen, and V. Sassone. Trust in Global Computing. In *ALP Newsletter*, Vol 17, n. 4, November 2004.

7. M. Carbone, M. Nielsen, and V. Sassone. A calculus for trust management. In *Proc. of the FST-TCS '04*, volume 3328 of *LNCS*, pages 161–173. Springer-Verlag, 2004.

Contents

Chapter 1

Introduction

We'll start the war from right here
— Theodore Roosevelt, Jr., Utah Beach, June 6, 1944.

1.1 Overview

This dissertation studies the problem of introducing the notion of trust into models for concurrency and distributed networks like process calculi. In particular, it focuses on the foundations of formal models for trust in large scale environments, capable of underpinning the use of trust-based security mechanisms as an alternative to the traditional ones.

Distributed networks, such as the Internet, have become widely used nowadays, and this has brought forward new issues and challenges in computer science and related fields. It seems clear that this is due to the impressive way the Internet has spread out all over the globe among all populations: it allows to connect any person from anywhere in the world to anyone else in any other place. Besides this, it also provides the biggest database available about anything. The Internet, like any other distributed network, can be seen as a big collection of computers, connected together. This reflects the topology of a global machine which does its computations in a distributed way, e.g. an online purchase where the buyer, the seller and the credit card issuer are involved in some computation. For this reason, a new branch of computer science, which takes care of distributed networks, is known as Global Computing (GC).

A GC system is composed of entities which are autonomous, decentralised, mobile, dynamically configurable, and capable of operating under partial information. Such systems, as e.g. the Internet, become easily very complex, and bring forward once again the need to guarantee security properties. Traditional security mechanisms, however, have severe limitations in this setting, as they are often either too weak to safeguard against the actual risks, or so stringent to impose unacceptable burdens on the effectiveness and flexibility of the infrastructure.

In this dissertation, we investigate the possibility of giving a solution to this new challenge. Our approach is to study the problem from a mathematical point of view by using formal methods, so that we can guarantee certain claims. Nowadays, software production is troubled by verification: unfortunately, verifying certain properties of systems is not always completely achievable, sometimes impossible. The complexity

of a GC system has impressively increased with respect to traditional systems, and this did not contribute to make things better, but actually it made things even worse. Our approach will be using formal methods, and so a formal model for verifying these properties. A consistent part of this document refers to a particular approach to distributed system management, which is *trust management systems*, whereby safety critical decision are made based on trust policies and their deployment in the presence of partial knowledge, have an important role to play in GC.

In this introduction we shall go through some history of the area and introduce the whole topic, starting from the pioneers of concurrency and then go on to trust, and its new applications.

1.1.1 Concurrency

The theoretical aspect of computer science has always evolved alongside the development of technologies. In progress, the need of building new tools for improving the quality of any kind of service, is always accompanied by the need of making theoretical studies in order to ensure consistency with the expectations and avoiding failures of what is built. Consider, for instance, the evolution of computer machines from single to multi threaded, and notice how new challenges have always been introduced in theoretical computer science. In the beginning, scientists were worried about modeling computations of single programs, e.g. operational semantics, denotational semantics and axiomatic semantics. But when parallelism was introduced, scientists realized that there was need for new theories, capable of expressing new features of the new systems.

Concurrency is a young branch of computer science which first appeared in the early '60s. Concurrency is concerned about the fundamental features of systems consisting of multiple computing agents, also known as processes, that interact among each other where, according to [8], a "process" is defined as something that refers to the behaviour of a system.

These notions cover a very large variety of systems which nowadays, most people can easily relate to, due to technological advances such as the Internet, programmable robotic devices and mobile computing. We shortly comment some features that a typical concurrent system may have [111], e.g. message-passing, shared-variables and synchrony, but we also comment reactive systems, timed systems, mobile systems and secure systems.

- *Message-passing* is one of the most common feature in concurrent systems that is a mechanism allowing agents to send messages to other agents (a typical example is email communication);

- *shared variables* refer to systems with a single central store which is shared by all the agents. Note that this may also implement communication among agents, and so message-passing through the store (an example is an Internet newsgroup);

- *synchrony* happens when the agents of the system are put on hold sending a message, constrained to wait for another agent to pick the message (a typical example is when doing a phone call). As an opposite of synchrony, in *asynchrony*,

messages are just left for other agents to be picked eventually (an example would be communication between our email reader and the email server);

- a *reactive system* is any physical system that responds to external stimuli and triggers events that may be perceived by its observer. This definition is wide enough to encompass all physical devices that exhibit some organized behavior, as well as interconnections of such devices into possibly large networks of dynamic and interacting components. One example of such a network is a modern factory in which manufacturing equipment is interconnected and organized around a common goal, e.g. the production of some goods;

- *timed systems* are systems where agents are constraint by temporal requirements (a typical example is a timed-out transaction with online banking);

- *mobile systems* are those systems where agents are mobile and able to change their communication links (an example of this could be agents moving around on the Internet hopping in different locations);

- *secure systems* are those systems where it is possible to guarantee certain properties perceived as secure. These systems usually need some formal reasoning for guaranteeing such properties (an example is again online banking).

The list above is just a subset of the numerous varieties of concurrent systems which are around nowadays. Mind that these systems can be combined together and result into more complicated ones, as for instance the Internet where all the examples listed above are embedded.

Now that we have given a flavour of what concurrent system are, we move into the theoretical aspect of these. The idea is to provide some formal methods for ensuring certain properties of a given system. In other words, we want to use formal methods in order to specify and prove properties about the functioning of a system. A good advantage is when these various features are perpendicular, i.e. completely unrelated, as this allows to study problems separately and so in a simpler way.

Taking inspiration from [111], we list some of the features a good model for concurrent systems should enjoy. A concurrent system must

- be simple, in order to provide a mean for doing clear thinking;

- be expressive, in order to capture interesting aspect of real systems;

- provide techniques, in order to do reasoning about particular properties.

Simplicity is a very important aspect that any model should have. A formal model must be a mean for helping understanding the complexity of a real system. Dealing with something which is not simple may complicate the way problems are approached, and, as a consequence, the way they are understood. Hans Bekiĉ [12], on providing a formal model for concurrent systems, states:

> "Our plan to develop an algebra of processes may be viewed as a high-level approach: we are interested in how to compose complex processes from simpler (still arbitrarily complex) ones".

There is, of course, a trade between simplicity and expressivity: the more expressive a model is, the more things it allows to model. This trade off is one of those important things that must be taken care of when designing an abstraction of a real system. Besides these two aspects, the third one is also very important. A model is meant to support formal reasoning about a system, i.e. it must provide techniques for espressing and then proving properties. Given these three parameter it is clear that it is not a very easy task designing a model for a concurrent system.

We now go through a short analysis of past concurrent models introduced in the last 40 years. Concurrency theory did its first apprearance in the early '60s with Petri nets, conceived by Carl Adam Petri starting from his dissertation [94]. Petri nets are a very simple yet effective model for expressing concurrent flow of information from agents to other agents. In those years, the introduction of Petri nets raised the question how to give semantics to programs containing a parallel operator.

Between 1973 and 1980, Robin Milner introduced a new calculus called CCS (Calculus for Communicating Systems) modeling interacting processes based on pure synchronisation. CCS [82] is a very simple, yet powerful model based on channels represented by names. Each process has the capability of running different children in parallel and enjoys a sequential operator (called prefixing) and recursion. The success of CCS lays on the development of semantics of concurrency, which went from the traditional domain-theoretic approach to a new operational view of processes equality based on the notion of bisimulation.

Almost at the same time, Tony Hoare developed CSP (Communicating Sequential Processes). CSP [61] is a simple and elegant language for describing parallel computations and their interactions. It evolved from a formal notation used to discuss communicating independent entities into a formal language for describing parallel systems, simulating them, and reasoning about them. CSP's strength is in its support for defining parallelism: definitions of processes and communcation buffers. The fundamental notion of a guarded command was introduced by CSP, and serves as a powerful model for regulating and synchronizing concurrent processes. All of the facilities in CSP were carefully chosen to permit formal proofs about deadlock-freedom and other properties of CSP models.

We also mention ACP (Algebra of Communicating Processes), a work initiated by Bergstra and Klop [15] and then taken over by Baeten [9], which had a very important impact on concurrency theory. In general, a process algebra (including CCS and CSP) is an algebraic approach to the study of concurrent processes. Its tools are algebraical languages for the specification of processes and the formulation of statements about them, together with calculi for the verification of these statements. ACP was important because it was the first (successful) tentative of seeing all previous models within a general picture.

CCS together with Hoare's CSP and ACP, constituted a new mathematical approach to non-sequential interactive computing.

Later on, in the beginning of the '90s, the π-calculus [84] has been introduced as a new and fundamental way of thinking about mobile interactive processes. This calculus, born as a direct continuation of CCS, comes from an idea of Uffe Engberg and Mogens Nielsen (ECCS) who introduced the notion of name passing (mobility) in [45]. The π-calculus enjoys an operator for creating local names which combined with name passing results into a very powerful computational mechanism. None of

the predecessors of π-calculus dealt with mobility among processes, and this is done in very simple way, i.e. by merging many notions into a single one. As Robin Milner states in [83], "the main contribution of the π-calculus is its identification of many apparently different things, e.g. labels, channels, pointers, variables, etc., as simply one single thing: names".

The π calculus has been widely studied in the recent years, but it still needs further exploration. Together with the study of its theoretical properties, many extensions and possible alternative have been proposed. One main problem with the upcoming need of modeling distributed networks is the notion of location. The π-calculus, as explained above, enjoys a flat structure of names without making distinction among different things. Recently, an extension of it to locations has been proposed by Hennessy and Riely [60] and Amadio, Boudol and Lhousianne [6] known as Distributed π-calculus or just *D*-π. This calculus is an extension of π where there are locations which contain π-calculus processes. These processes are enhanced with an operator for migrating to other locations. Unfortunately when coming to locations there is, somehow, the need of making such a distinction with other names. In this dissertation we show that, actually adding locations to the π-calculus leads to an improvement of expressivity with respect to certain conditions.

Another alternative to distributed π-calculus is the Ambient Calculus, introduced by Cardelli and Gordon in [35] in the late '90s. The Ambient Calculus is a powerful and strong model for mobile agents based only on the notion of location. Contrary of the π-calculus, computations are not based on communication among agents[1] but on pure migrations of nested locations into other locations. We shall see (informally) in the last part of this dissertation, that the ambient calculus can be seen as an infinite extension of D-π. Although, the Ambient Calculus may seem more powerful than the π-calculus, it has not been as successful. Unfortunately, it lacks that sort of mixture between simplicity and elegance that features π-calculus. There is a lot of theory developed on it, but it has always led to technical problems, especially when defining techniques for proving equivalences of processes.

One of the latest models, which also allow to embed locations are Milner's bigraphical reactive system (BRS) [68, 67]. They involve bigraphs, a categorical structure in which the nesting of nodes represents locality. BRSs aim to provide a uniform way to model spatially distributed systems that both compute and communicate. BRSs can also be seen as an abstract model for process calculi.

We must mention the spi calculus [2], a calculus for cryptographic protocols. The spi calculus is an extension of the π-calculus designed for describing and analyzing cryptographic protocols, in particular for studying authentication protocols. Whereas the π-calculus suffices for some abstract protocols, the spi calculus enables to consider cryptographic issues in more detail. Again, in this dissertation, we shall briefly see how to encode some spi calculus protocols in a minimal extension of π.

We conclude this short overview of calculi, by mentioning the fusion calculus, the join calculus and Klaim. The fusion calculus [93] contains the polyadic pi-calculus as a proper subcalculus and thus inherits all its expressive power. The gain is that "fusion contains actions akin to updating a shared state, and a scoping construct for

[1] Some versions of the Ambient Calculus enjoy a local communication mechanism which has been shown to be redundant in [122].

bounding their effects". The authors claim that they achieve these improvements by simplifying the pi-calculus rather than adding features to it. The fusion calculus has only one binding operator whereas the pi-calculus has two (input and restriction). The join calculus [48] is still π-like, but it makes use of join patterns when processes synchronise and communicate. Klaim [89] is instead a kind of shared variables calculus where there are multiple tuple spaces processes that can be moved from one computing environment to another.

All models mentioned above provide excellent techniques for reasoning about the modeled systems. As said, the most common techniques for reasoning about processes have been equivalences. Certainly bisimulation is the most used, as it is also very intuitive. Nevertheless, there have been many other equivalences introduced in the past, like trace equivalence, testing equivalence [90] and many others.

But equivalences are not the only techniques that can be found. There are very common static techniques where a lot of research has been done. In particular, we must mention types for process calculi, especially for π-calculus, from '96 [96] to new and very interesting linear types [13, 14, 121, 64]. In this dissertation, there will be a short treatment of types for the extension of π we have mentioned above. Moreover, there are also other kind of static techniques for defining properties of system, e.g. logic and control flow analysis. Hennessy and Milner have developed a very interesting modal logic, known as Hennssy-Milner logic [58], which characterises the bimulation equivalence for π-calculus, i.e. whenever two processes are bisimilar then they satisfy the same set of formula and vice versa. After this, many other logics have been studied, aiming at different properties for different calculi, e.g. a logic for π-calculus [85] and one for the Ambient Calculus [27, 28]. We conclude with control flow analysis and abstract interpretation. We must definitely cite the work in [18] where information flow properties are studied, and [19] concerning non-interference. Also in [114] and in [91, 74] the authors develop abstract interpretation techniques respectively for π-calculus and Ambient Calculus.

1.1.2 A new approach: Trust

In the previous section we have treated concurrent systems in general, and then listed some important model that have been milestones. In this dissertation, the aim is to write about a particular kind of concurrent systems, and then provide a model for reasoning about it. We aim at studying concurrent systems where the notion of human trust is used for establishing interactions between computational entities.

What is Trust?

The variety of common terms shows that there is no precise definition and hints at the range of views of trust. Sociologist Diego Gambetta [52] introduces trust as 'a particular level of the subjective probability with which an agent assesses that another agent or a group of agents will perform a particular action, both before he can monitor such action (or independently of his capacity ever to be able to monitor it) and in a context in which it affects his own action. Social psychologist Morton Deutsch's considers trust when faced with an ambiguous path with beneficial or harmful results depending on another person. He identifies various types of trust, from trust as the fallback

when no other option is available to trust as confidence that the desired outcome will be reached. Deutsch suggests that people take trusting actions when possible benefits outweigh the likelihood of being let down. This implies that risk analysis forms an important part of the trust decision. Due to these and other views, Stephen Marsh [77] reasons that it might prove more suitable to model trust's behavior rather than trust itself, removing the need to adhere to specific definitions. An important observation from all these sources is that trust one individual's opinion of another is a subjective notion, and every individual decides whether to trust based on the evidence available for personal evaluation (although you might delegate this decision to a more authoritative source in certain circumstances). Also, trust is not symmetric two individuals do not need to have similar trust in each other. Even if two entities get the same evidence, they might not necessarily interpret this information in the same way. Trust is also situation-specific; trust in one environment does not directly transfer to another environment. So a notion of context is necessary. Despite this situational nature, there is some agreement on a dispositional aspect of trust as a measure of your propensity to believe in others' trustworthiness. Social scientists also highlight trust's dynamic properties: It is self-preserving and self-amplifying, it increases through periodic successful interactions, and it degrades through disuse or misuse. Trust is inherently linked to risk; there is no reason to trust if there is no risk involved. This relationship implies that cooperation is less likely with higher risk unless the benefits from cooperating are worth the risk. So reasoning about trust lets entities accept risk when interacting with others. [remove some of the above part and put it in the introduction]

Familiarity with trust models from the social sciences is a good starting point for our search of a foundational, comprehensive formal model of trust. We start with a quote taken from [77] about Deutch's definition of trust [42]:

- the individual is confronted with an ambiguous path, a path that can lead to an event perceived to be beneficial or to an event perceived to be harmful;
- he perceives that the occurrence of the two events above is contingent on the behaviour of another person;
- he perceives the strength of the harmful event to be greater than the strength of the positive one.

If he chooses to take an ambiguous path with such properties, I shall say he makes a trusting choice; if he chooses not to take the path, he makes a distrustful choice.

We all make trusting decisions, most of us every day of our lives, and many times per day [76]. According to Luhmann, social life is impossible without trust because "the fear about our existence will grow without confidential communication". Trust describes our attitude towards events produced by human actions and which are, therefore, at least potentially subject to our control, to the extent that we may monitor and influence the actions of others. The need of trust for human beings can be also thought of as an appropriate starting point for the derivation of rules for proper behaviour or how to act successfully in the world of complexity and uncertainty. As a matter of fact, it is this complexity and uncertainty that creates the need for trust. The basis of this need are two structural aspects of the modern world:

- the modern world is complex and produces uncontrolled complexity;

- dangers are replaced by risks.

We quote from [66]: "Where there is trust there are increased possibilities for experience and action, there is an increase in the complexity of the social system and also in the number of possibilities which can be reconciled with its structure, because trust constitutes a more effective form of complexity reduction".

We also mention McKnight and Chervany [78], who provide a typology of trust used to classify existing research on trust in domains like sociology, psychology, management, economics, and political sciences. Trust is thereby classified conceptually in six categories: *disposition*, when entity *a* is naturally inclined to trust; *situation*, when *a* trusts a particular scenario; *structure*, when *a* trusts impersonally the structure *b* is part of; *belief*, when *a* believes *b* is trustworthy; *intention*, when *a* is willing to depend on *b*; *behaviour*, when *a* voluntarily depends on *b*. Orthogonally, the notion of *trustee* is classified in categories, the most relevant of which decree that *b* is trusted because of its *competence*, *benevolence*, *integrity*, or *predictability*. *Other characteristics* (including openness and carefulness), *Other trustees* (including people or institutions).

Trust-based systems

Registered parties behind firewalls in strictly controlled environments carry out most substantial, accountable computation. However, pervasive computing foresees a massively networked infrastructure supporting a large population of diverse but cooperating entities. Entities will be both autonomous and mobile and will have to handle unforeseen circumstances, ranging from unexpected interactions with other entities to disconnected operation. As pointed out in the introduction, this infrastructure introduces new security challenges that existing security models and mechanisms do not adequately address. Because of the infrastructure's scale, the security policy must encompass billions of potential collaborators. Mobile entities will often become disconnected from their home networks and must be able to make fully autonomous security decisions; they cannot rely on specific security infrastructures such as certificate authorities and authorization servers. Although certificate authorities might help establishing other collaborators' identities, in the environment envisaged, identity conveys no a priori information about a principal's likely behavior.

If trust is a fundamental concept in human behaviour, and has enabled collaboration between humans and organisations for millennia, the ultimate aim of our research on trust-based systems is to transfer such forms of collaboration to modern computing scenarios. There will clearly be differences between the informal notion of trust explored in the social sciences and the kind of formality needed for computing. Mainly, our models need in the end to be operational, so as to be implementable as part of GC systems. Equally important is their role in providing a formal understanding of how trust is formed from complex interactions between individuals, so as to support reasoning about properties of trust-based systems.

We want to point out that in a trust-based setting, the system will no longer be in the same situation as traditional ones. In the classical world there is usually a certification about properties, which is most of the times impossible to prove. Whereas, trust-based implies that systems will expose theirself to some risks.

We believe that a good mathematical model of computational trust should be capable of expressing all such aspects, as well as further notions of primary relevance in computing, e.g. that trust information is time dependent and, in general, varies very rapidly. Also, it should be sufficiently general to allow complex structures representing combinations of different types of trust.

Previous work on trust

Trust is a pervasive notion, thoroughly studied in a variety of different fields, including social sciences, economics and philosophy. Here we only survey recent work on trust as a subject in computing; the reader is referred to [78] for a broader interpretation. A detailed survey can be found in Grandison and Sloman's [54].

Most of the existing relevant work concerns system building. In [100], Rivest *et al.* describe SDSI, a public key infrastructure featuring a decentralised name space which allows principals to create their own local names to refer to other principals' keys and in general, names. Ellison *et al.* [44] proposed a variation of the model which contributes flexible means to specify authorisation policies. The proposals are now merged in a single approach, dubbed SPKI/SDSI. Other systems of practical relevance include PGP [123], based on keys signed by trusted certificating authorities; KeyNote [17], which provides a single, unified language for both local policies and credential containing predicates to describe the trusted actions granted by (the holders of) specific public keys; and REFEREE [39], which uses a tri-valued logic which enriches the booleans with a value `unknown`. Trust in the framework of mobile agents is discussed e.g. in [117]. Delegation plays a relevant rôle in trust-based distributed systems. A classification of delegation schemes is proposed by Ding *et al.* [43], where they discuss implementation and analyse appropriate protocols. The ideas expressed in [43] lie at a level different from ours, as their focus is exclusively on access control.

The theoretical work can be broadly divided in two main streams: logics, where the trust engine is responsible for constructing [25, 24, 65, 69, 71] or checking [7] a proof that the desired request is valid; and computational models [116, 40], like our approach.

Burrows *et al.* propose the BAN logic [25], a language for expressing properties of and reasoning about the authentication process between two entities. The language is founded on cryptographic reasoning with logical operators dealing with notions of shared keys, public keys, encrypted statements, secrets, nonce freshness and statement jurisdiction. In [24], Abadi *et al.* enhance the language by introducing delegation and groups of principals: each principal can have a particular role in particular actions. The Authorisation Specification Language (ASL) by Jajodia *et al.* [65] separates explicitly policies and basic mechanisms, so as to allow a more flexible approach to the specification and implementation of trust systems. ASL supports also role-based access control.

Modal logics have a relevant place in specifying trust models, and have been used to express possibility, necessity, belief, knowledge, temporal progression, and more. Jones and Firozabadi [69] address the issue of reliability of agents' transmissions using a modal logic of actions [98] to model agents. Rangan [99] views a distributed system as a collection of communicating agents in which an agent's state is the history of its messages. Rangan's model builds on simple trust statements to define simple proper-

ties, which are then used to specify systems and analyse them with respect to properties of interest. Recently, Jøsang [71] proposed a logic of uncertain probabilities, a work which is related to our interval construction and can be recast as an instance of it in our framework. Specifically, Jøsang considers intervals of belief and disbelief over real numbers between 0 and 1.

Concerning computational models, Weeks [116] provides a model based on fixpoint computations which is of great relevance to our work. Winsborough and Li [118] study automated trust negotiation, an approach to regulate the exchange of sensitive credentials in untrusted environments. Clarke *et al.* [40] provide an algorithm for "certificate chain discovery" in SPKI/SDSI whereby principals build coherent chains of certificates to request and grant trust-based access to resources.

1.2 Structure of the dissertation

This dissertation is the result of the research done during the Phd-studies. Each chapter corresponds to one or more publication. Mainly, the document is divided into two parts: one part which concerns modeling of trust-based systems and another part that takes care of the expressive power of polyadic synchronisation in π-calculus. We have decided to include the latter in a new fresh part, as the work can be considered as independent from the rest of the dissertation. Nevertheless, polyadic synchronisation is also used in the first part, where a model concerning trust-based systems is introduced. Note that, studying the expressive power of polyadic synchronisation does not influence the results given before about trust.

In details, this is the list of chapters:

- *Front matter.*
  - *Chapter 1: Introduction.* This is the introduction of the dissertation and corresponds to this chapter which is about to end. In here we have widely introduced the area, always keeping out technical details.
  - *Chapter 2: Preliminaries.* In Chapter 2 we provide a quick overview of the preliminary notions we need for the rest of this document. Namely, the chapter is divided into two main parts: the first part is about theory of partial orders and the second is about process calculi. Concerning partial orders we report the most known results which are going to be used in Chapter 3. The second part concerns, in particular, the π-calculus, of which we report the main definitions and results. The chapter concludes with a brief introduction to encodings and a hierarchal classification of calculi.
- *Part I: Trust in Global Computing.*
  - *Chapter 3: The SECURE project.* This chapter is about the SECURE project and its main issues. The author of this dissertation has taken part to the project, collaborating to the development of the prefixed targets. In here, there is a full description of the project, with some explanations about the results achieved by the different collaborators. The chapter correspond

to the publication in [26], and details about trust life-cycle can also be found in [115].

- *Chapter 4: Trust Domains.* In here, we introduce a model for trust. Trust is related to a construction of intervals which resembles previous works done in semantics and computer arithmetics. Besides, we introduce a small language for describing trust-based policies. The chapter corresponds to [33].
- *Chapter 5: A Calculus for Trust Management.* This chapter is the core of this dissertation, and deeply studies `ctm`, a calculus for trust management. The calculus is a terse and powerful tool for modeling trust-based systems, and provides useful techniques for comparing systems. In this chapter, we extend what studied in [34], reporting proofs for the results claimed, and we also provide a proof method for dealing with terms equivalences.

- *Part II: On Polyadic Synchronisation.*

  - *Chapter 6: Polyadic Synchronisation and its expressive power.* In this chapter we study an interesting extension of channel communication for π-calculus-like calculi. This extension has been applied to `ctm` in the previous chapter, and now it is deeply studied in its details. This work corresponds to publications [31] and [30]. The chapter concludes with some ongoing work on types for polyadic synchronisation [32].

Chapter 2

Preliminary notions

The more the preliminaries, the better
— Alexandre Ballot, Amsterdam, 2000.

In this chapter we introduce some technical preliminaries that we will be using in the dissertation. First we will go through some theory of partial orders. We direct the reader for further details to [56, 119]. Then we will introduce the π-calculus [84], a formalism on which many of the results of this dissertation are based on.

2.1 Partial Orders

The theory of partial orders is widely used in computer science. Partial orders are particulary used in semantics of programming language as domains for denotational semantics [104, 97, 119]. In this thesis we will use some of this theory for modelling part of trust based systems. We report some basic definitions below.

A *preorder* $(X, \leq)$ is a reflexive and transitive relation $\leq$ on X. A preorder which respects the antisymmetric property is called a *partial order*.

An element $x \in X$ is said to be *maximal* if for all $y \in X$, $x \leq y$ implies $x = y$. An element $x \in X$ is *maximum* if for all $y \in X$ it holds that $y \leq x$. It follows that there can be at most one maximum element. Dually we can define *minimal* and *minimum* elements. A maximum element is also called *top* and is denoted by $\top$. The symbol $\bot$ denotes a minimum, also called *bottom*.

If $Y \subseteq X$ then an element $x \in X$ is an *upper bound* for Y whenever for all $y \in Y$ we have that $y \leq x$. Moreover if the set of all upper bounds for Y has a minimum, this is called the *least upper bound (lub)* and it is denoted by $\sqcup Y$. Dually we can define the *lower bound* and the *greatest lower bound (glb)* for Y. The greatest lower bound is denoted by $\sqcap Y$. When dealing with more then one ordering, we shall identify the different operators by indicating to which one they belong, e.g. $\sqcup_{\leq} X$ stands for the least upper bound of X wrt the ordering $\leq$, or we may also use $\vee$ for lub and $\wedge$ for glb.

Given two partial orders $(X, \leq)$ and $(X', \leq')$, a function f from X to X' is *monotonic* whenever $x \leq y$ implies $f(x) \leq f(y)$ for $x, y \in X$. The function f is called *anti-monotonic* whenever $x \leq y$ implies $f(y) \leq f(x)$ for $x, y \in X$.

A subset Y of X is a *chain* if for every y, y' in Y either $y \leq y'$ or $y' \leq y$. If X is a chain then $(X, \leq)$ is called a *total order*. An ω-*chain* c is a monotonic function from the set of ordered natural numbers ω to Y, i.e. $\mathsf{c} = (c_n)_{n \in \omega}$ such that $c_0 \leq c_1 \leq c_2 \leq \ldots$.

Functions from a set A to a partial order $(X, \leq)$ are *pointwise ordered* by $\leq_{A\to X}$ where $f \leq_{A\to X} g$ if and only if for all $a \in A$, $f(a) \leq g(a)$.

We now give the definition of a particular kind of partial order, called complete partial order.

Definition 2.1 (CPOs). A partial order $(D, \sqsubseteq)$ is a *complete partial order* (CPO) if it has a minimum element $\bot$ and each ω-chain c in D has a least upper bound $\bigsqcup \mathsf{c}$.

Note that if $(D, \sqsubseteq)$ is a CPO, then the pointwise ordered function space $([A \to D], \sqsubseteq_{A\to D})$, for some A, is also a CPO.

Given two CPOs $(D, \sqsubseteq)$ and $(D', \sqsubseteq')$ we say that a function $f : D \to D'$ is continuous if for each ω-chain c, it holds that $\bigsqcup f(\mathsf{c}) = f(\bigsqcup \mathsf{c})$. Note that continuity implies monotonicity. Moreover given two complete partial orders $\leq$ and $\preceq$ on a set X, we say that $\leq$ is continuous wrt $\preceq$ whenever for all ω-chain $\mathsf{c} = (c_n)_{n\in\omega}$ wrt the ordering $\leq$ and for all x, we have that for all indices i, $c_i \leq x$ implies $\bigsqcup_{\leq} c_i \preceq x$.

Given a CPO $(D, \sqsubseteq)$, a function $f : D \to D$ and $d \in D$, we say that d is a *prefixed point* whenever $f(d) \sqsubseteq d$. Moreover we say that d is a *fixed point* if $f(d) = d$. A fixed point d is the *least fixed point* if for all fixed point y it holds that $d \sqsubseteq y$.

The importance of CPOs here is that every continuous function $f : D \to D$ on a CPO has a least fixed point $\mathsf{fix}(f) \in D$ (cf. [119]). In the following the term $f^k(d)$ denotes k applications of the function f to d, i.e. $f^k(d) = \underbrace{f(f(...f}_{k \text{ times}}(d))$.

Theorem 2.1 (Fixed Point Theorem). *Let $(D, \sqsubseteq)$ be a CPO and $f : D \to D$ a continuous function. Then f has a least fixed point $d = \bigsqcup_{i\in\omega} f^i(\bot)$.*

Proof. We have to show that the least fixed point exists and is equal to $\bigsqcup_{i\in\omega} f^i(\bot)$.

- We have to show that $\bigsqcup_{i\in\omega} f^i(\bot)$ is a fixed point. Because of continuity of f we have that $f(\bigsqcup_{i\in\omega} f^i(\bot)) = \bigsqcup_{i\in\omega} f(f^i(\bot))$ which is equal to $\bigsqcup_{i\in\omega} f^{i+1}(\bot)$. By definitions of f^k and $\bot$ we get $\bigsqcup_{i\in\omega} f^{i+1}(\bot)) \sqcup \{\bot\} = \bigsqcup_{i\in\omega} f^i(\bot))$.

- Let y be a prefixed point, i.e. $f(y) \sqsubseteq y$. By definition of the bottom element we have that $\bot \sqsubseteq y$ and then by monotonicity $f(\bot) \sqsubseteq f(y)$. From this we can derive that $f(\bot) \sqsubseteq y$ and inductively $f^n(\bot) \sqsubseteq y$. Thus, $\bigsqcup_{i\in\omega} f^i(\bot) \sqsubseteq y$.

□

In the rest of this section, we shall introduce another type of partial order, a *complete lattice*.

Definition 2.2 (Complete lattice). A partial order $(D, \preceq)$ is a *complete lattice* if every $X \subseteq D$ has a least upper bound.

As for CPOs, if $(D, \preceq)$ is a lattice, then the pointwise ordered function space $([A \to D], \preceq_{A\to D})$, for some A, is also a lattice.

An ω-*cochain* in a lattice $(D, \preceq)$ is an antimonotonic function $\mathsf{c} : \omega \to D$, that is a function such that $i \leq j$ implies $c_j \leq c_i$. A function $f : D \longrightarrow D'$ is *co-continuous* iff for each ω-cochain c in D, it holds that $\bigsqcap f(\mathsf{c}) = f(\bigsqcap \mathsf{c})$; f is *bi-continuous* if it is continuous and co-continuous.

A similar result on least fixed points like the previous one on CPO's holds also for lattices (cf. [110]).

Theorem 2.2 (Knaster-Tarski Theorem). *Let $(X, \preceq)$ be a lattice and $f : X \to X$ a monotonic function. Then f has a least fixed point and it is the minimum of the prefixed points, i.e. $fix(f) = \bigsqcap\{x \in X \mid f(x) \preceq x\}$*

Proof. Let $Y = \{x \in X \mid f(x) \preceq x\}$. Certainly we have that $fix(f) \preceq x$ for any $x \in Y$. By monotonicity and definition of Y, it holds that $f(fix(f)) \preceq f(x) \preceq x$. But then also $f(fix(f)) \preceq \bigsqcap\{x \in X \mid f(x) \preceq x\}$ which shows that $fix(f)$ is a prefixed point and also the least one. Moreover as $f(fix(f)) \preceq fix(f)$ we have that $f(f(fix(f))) \preceq f(fix(f))$, which implies that $f(fix(f)) \in Y$ and so $fix(f) \preceq f(fix(f))$. In conclusion $f(fix(f)) = fix(f)$ which means that indeed it is a fixed point. □

2.2 The π-calculus

As mentioned in the introduction, the π-calculus [84] is a formalism for describing concurrent execution of communicating processes based on the idea, inherited from CCS [82], of synchronising over named channels. For over ten years research has been done on the π-calculus, and we consider it as an important starting point for introducing some of the results of this dissertation.

We give a brief introduction of syntax and semantics of mixed-choice π-calculus with matching (π for short). For a deeper treatment of the topic we recommend the reader to [83] and [102].

2.2.1 Syntax

Given a countable set of values Val ranged over by a, b, c, x, y, z, w, the syntax of π is defined as follows:

$$\begin{array}{lllll} P,Q ::= & 0 & (\text{NULL}^{\pi}) & \mid \Sigma_i\, \alpha_i.P_i & (\text{CHOICE}^{\pi}) \\ & \mid P \mid P & (\text{PAR}^{\pi}) & \mid (\nu x)P & (\text{RES}^{\pi}) \\ & \mid [x = y]P & (\text{MATCH}^{\pi}) & \mid\ !P & (\text{BANG}^{\pi}) \end{array}$$

where

$$\alpha ::= \ \tau \ \mid \ x(y) \ \mid \ \overline{x}\langle y \rangle$$

and represents the basic operations of the calculus: $x(y)$ is an input, $\overline{x}\langle y \rangle$ is an output and τ denotes an internal evolution step. The process $\Sigma_i\, \alpha_i.P_i$ represents mixed guarded choice, i.e. the possibity of executing non-deterministically one of the actions α_i and continuing as P_i. In the rest of this thesis we shall use the notation $\Pi_{1..n}P_i$ as a shorthand for multiple parallel composition $P_1|...|P_n$. Sums and products are usually finite in π-calculus. The process 0 stands for the inactive process, (νx) and ! are respectively restriction and replication: restriction makes the name x private to a process whereas replication $!P$ represents infinitely many copies of P. The matching operator $[x = y]P$ behaves like P if x is equal to y, and behaves like the inactive process 0 otherwise. Where necessary we will write prefixes of *pure* synchronisation in the style of CCS: $\overline{x}.P$ will be a shorthand for $\overline{x}\langle y \rangle.P$ for some y, whereas $x.P$ will stand for $x(y).P$ where

y does not appear in P. In this section we shall often use letters M and N to denote summations. We shall often omit trailing inactive processes.

The calculus above, compared with the original formulation of the calculus, drops the full-choice construct ($P + P$) in favour of the more well-behaved mixed choice, as found for example in [102].

2.2.2 Binding, Substitution, α-conversion and Structural Congruence

Binding of names is important and it must be taken care of very carefully. We now shortly introduce binding of names within processes. In the processes $z(x).P$ and $(\nu x)P$, the occurence of x in $z(x)$ and (νx) is said to be *binding* in P, where P is said to be the scope of x. In other words, we say that an occurence of x in P is *bound* if it is, or it lies within the scope of, a binding occurrence of that name. An occurence of a name is free whenever it is not bound. We can then define the functions $fn(P)$, $bn(P)$, and $n(P) = fn(P) \cup bn(P)$ respectively as *free names*, *bound names*, and *names* of P. For example, consider the process $(\nu x)\ (\overline{x}\langle b\rangle) \mid x(y).\overline{x}\langle y\rangle$. We can observe that the x is bound in $\overline{x}\langle b\rangle$ but it is not in $x(y).\overline{x}\langle y\rangle$ and that y is bound in $\overline{x}\langle y\rangle$.

A *substitution* is a function $\sigma : \mathsf{Val} \to \mathsf{Val}$ which is equal to the identity except on a finite subset of Val. Given a process P, we will write $P\sigma$ for the process resulting from the application of σ to the free names of P, i.e. for all $x \in fn(P)$, every occurence of x is syntactically replaced in P by $\sigma(x)$. Given names x and y, notation $P\{x/y\}$ is equivalent to $P\sigma$ for σ different from the identity only on the single name y and such that $\sigma(y) = x$. A *change of bound names* in a process Q is the replacement of a subterm $x(y).P$ or $(\nu x)P$ of Q respectively by $x(w).P\{w/y\}$ and $(\nu w)\ P\{w/x\}$ for some name w such that $w \notin fn(P)$. We say that two processes P and Q are α-*convertible* (written $P =_\alpha Q$) whenever Q can be obtained from P by a finite number of changes of bound names. The notion of α-conversion, together with the one of substitution, comes from the λ-calculus [10].

A context $\mathcal{C}[\cdot]$ is a process which contains a hole. Formally contexts for the π-calculus are given by the syntax

$$\mathcal{C}[\cdot] ::= \ \cdot \ \mid \ \alpha.\mathcal{C}[\cdot] + M \ \mid \ (\nu n)\ \mathcal{C}[\cdot] \ \mid \ \mathcal{C}[\cdot] \mid P \ \mid \ P \mid \mathcal{C}[\cdot]$$

The term $\mathcal{C}[P]$ denotes the result of replacing the hole in the context $\mathcal{C}$ by process P.

An equivalence relation $\mathcal{R}$ on the set of processes is called a *congruence* whenever $P\mathcal{R}Q$ implies $\mathcal{C}[P]\mathcal{R}\mathcal{C}[Q]$ for any context $C[\cdot]$. The *structural congruence* relation $\equiv_\pi$ states when two processes are to be considered syntactically equivalent, and is defined as the least congruence satisfying α-conversion, the commutative monoidal laws with respect to both (|,0) and (+,0) and the following axioms:

$$\begin{array}{ll}
(\textsc{Struct}_1^\pi) & [x = x]P \equiv_\pi P \\
(\textsc{Struct}_2^\pi) & (\nu x)P \mid Q \equiv_\pi (\nu x)(P \mid Q) \text{ if } x \notin fn(Q) \\
(\textsc{Struct}_3^\pi) & (\nu x)0 \equiv_\pi 0 \\
(\textsc{Struct}_4^\pi) & (\nu x)(\nu y)P \equiv_\pi (\nu y)(\nu x)P \\
(\textsc{Struct}_5^\pi) & !P \equiv_\pi !P \mid P
\end{array}$$

The first axiom states the meaning of the matching operator, i.e. when the two arguments of $=$ are equal we can then proceed as P. The axiom $(\textsc{Struct}_2^\pi)$ says that

whenever we want to extend the scope of a restriction (binding) to a new process in parallel, we have to take care of the fact that the restricted name (which must be private to P) may clash with another name in Q. This two axiom are going to be important in the definition of the reduction semantics in next section. We also add two extra axioms which may be useful for proving some extra laws and getting rid of unwanted terms. The axiom (STRUCT_3^π) says that restricting a name in the process 0 is equivalent to the process 0 itself. The last but one axiom allows to interchange adjacent restrictions. The axiom (STRUCT_5^π) expresses the semantics of replication, i.e. it is possible to get an unbounded number of copies of P.

2.2.3 Reduction Semantics and Barbed Congruence

We now show how the parallel components of a process can react one with another. By using the structural congruence defined in above, we define $\rightarrow^\pi$ as the least binary relation on processes satisfying the rules given in table 2.1.

$$(\text{COMM}^\pi)\ (x(y).P+M) \mid (\overline{x}\langle z\rangle.Q+N) \rightarrow_\pi P\{z/y\} \mid Q$$

$$(\text{TAU}^\pi)\ \tau.P+M \rightarrow_\pi P \qquad (\text{RES}^\pi)\ \frac{P \rightarrow_\pi P'}{(\nu x)\,P \rightarrow_\pi (\nu x)\,P'}$$

$$(\text{PAR}^\pi)\ \frac{P \rightarrow_\pi P'}{P \mid Q \rightarrow_\pi P' \mid Q} \qquad (\text{STRUCT}^\pi)\ \frac{P \equiv_\pi Q \quad Q \rightarrow_\pi Q' \quad P' \equiv_\pi Q'}{P \rightarrow_\pi P'}$$

Table 2.1: Reduction Rules for π-calculus.

The rule (COMM^π) defines communication, namely, if two processes willing to perform respectively an output and an input action, can be put next to each other by the parallel operator, then they can communicate. The result of the communication is that the process performing the input action gets the outputted name z, which is going to be substituted to the bound name y in P. The rule (PAR^π) is used to implement concurrency, i.e. the interleaving among processes. The structural congruence can be used through rule (STRUCT^π) where reductions can be proved from structural congruent processes. Rules (TAU^π) and (RES^π) are meant respectively for τ actions (silent) and processes operating under a restriction scope.

Example 2.1. We report in here an example of reduction taken from [102]. Let $P \triangleq (\nu x)\ (x(z).\overline{z}\langle y\rangle.0 \mid (\overline{x}\langle a\rangle.0 \mid \overline{x}\langle b\rangle.0))$ and $P_1 \triangleq (\nu x)\ (((\overline{x}\langle a\rangle.0+0) \mid x(z).\overline{z}\langle y\rangle.0+0) \mid \overline{x}\langle b\rangle.0)$. By structural congruence we have that $P \equiv_\pi P_1$. Applying the rule (COMM^π) we have that

$$(\overline{x}\langle a\rangle.0+0) \mid (x(z).\overline{z}\langle y\rangle.0+0) \rightarrow_\pi 0 \mid \overline{a}\langle y\rangle.0$$

Now by (PAR^π) and (RES^π),

$$P_1 \rightarrow_\pi P_3 \triangleq (\nu x)\ ((0 \mid \overline{a}\langle y\rangle.0) \mid \overline{x}\langle b\rangle.0)$$

and as $P_3 \equiv^{\pi} P_4 \triangleq (\nu x)\ (\overline{a}\langle y\rangle\,.\,0 \mid \overline{x}\langle b\rangle\,.\,0)$ we have, by rule (STRUCT$^{\pi}$), that also $P \rightarrow_{\pi} P_4$. Also,

$$P \rightarrow_{\pi} \overline{b}\langle y\rangle\,.\,0 \mid (\nu x)\,\overline{x}\langle a\rangle\,.\,0$$

It is natural to define the observables of a process P, i.e. what is *visible* through the parallel operator by a context Q, in order for interaction to take place. This justifies the following definition.

Definition 2.3 (Barbs). The main observability predicates of π-calculus are *barbs* defined as:

$$\begin{array}{llll} P \downarrow^{\pi}_{x} & \triangleq P \equiv_{\pi} (\nu x_1)...(\nu x_n)(x(y).Q + R \mid R') & P \Downarrow^{\pi}_{x} & \triangleq \exists P'.(P \rightarrow_{\pi} P' \wedge P' \downarrow^{\pi}_{x}) \\ P \downarrow^{\pi}_{\overline{x}} & \triangleq P \equiv_{\pi} (\nu x_1)...(\nu x_n)(\overline{x}\langle y\rangle.Q + R \mid R') & P \Downarrow^{\pi}_{\overline{x}} & \triangleq \exists P'.(P \rightarrow_{\pi} P' \wedge P' \downarrow^{\pi}_{\overline{x}}) \\ P \downarrow^{\pi} & \triangleq \exists x.(P \downarrow^{\pi}_{x} \vee P \downarrow^{\pi}_{\overline{x}}) & P \Downarrow^{\pi} & \triangleq \exists x.(P \Downarrow^{\pi}_{x} \vee P \Downarrow^{\pi}_{\overline{x}}) \end{array}$$

for some name $x \notin x_1, ..., x_n$, and some processes Q, R, and R'.

The previous definition could suggest a possible way of comparing processes. Basically, two processes are equivalent whenever they show the same barbs.

Definition 2.4 (Barbed Bisimilarity). A barbed bisimulation is a symmetric relation $\mathcal{R}$ on processes such that whenever $P\mathcal{R}Q$

- $P \downarrow^{\pi}$ implies $Q \downarrow^{\pi}$;
- $P \rightarrow_{\pi} P'$ implies $Q \rightarrow_{\pi} Q'$ and $P'\mathcal{R}Q'$.

Two processes are barbed bisimilar ($\dot{\simeq}_{\pi}$) if related by a barbed bisimulation.

Unfortunately barbed bisimilarity is not satisfactory as a process equivalence. Consider for example the equivalence $(\nu z)\ (\overline{z}\langle z\rangle\,.\,\overline{y}\langle a\rangle \mid z(w)) \dot{\simeq}_{\pi} \tau\,.\,\overline{y}\langle b\rangle$. The two processes should not be considered to have the same observable behaviour, becuase they send different names via y [102]. In fact we could write a context which is able to distiguish them, e.g. $- \mid y(x)\,.[x = a]\tau$. From this example, we can get the intuition that the equivalence above should be closed under all possible contexts.

Definition 2.5 (Barbed Congruence). Two processes P and Q are barbed congruent, written $P \simeq^{c}_{\pi} Q$, whenever $C[P] \dot{\simeq}_{\pi} C[Q]$ for all contexts $C[\,]$.

The relation $\simeq^{c}_{\pi}$ is just the restriction under all possible contexts of the barbed bisimilarity $\dot{\simeq}_{\pi}$. If we now consider the example above we have that $(\nu z)\ (\overline{z}\langle z\rangle\,.\,\overline{y}\langle a\rangle \mid z(w)) \not\simeq^{c}_{\pi} \tau\,.\,\overline{y}\langle b\rangle$. In fact, for $\mathcal{C}[\,] = [] \mid x(z)\,.\,z$, we have $\mathcal{C}[(\nu z)\ (\overline{z}\langle z\rangle\,.\,\overline{y}\langle a\rangle \mid z(w))] \not\dot{\simeq}_{\pi} \mathcal{C}[\tau\,.\,\overline{y}\langle b\rangle]$ as, after two reduction, the right component would show barb b, instead the left one would show a.

2.2.4 Labelled Transition System

Unfortunately proving that two processes are barbed congruent is very hard, as we have to quantify all over possible context. A standard way to overcome the problem is to define the semantics in a co-inductive way and then define a notion of bisimulation which is equivalent to the barbed one. We now define a labeled transition system (*lts* for short) for the π-calculus, i.e. a relation $\longrightarrow_{\pi} \subseteq \mathcal{P} \times \mu \times \mathcal{P}$ for $\mathcal{P}$ the set of processes.

Namely, given two processes and an action, $P \xrightarrow{\mu}_\pi P'$ means that process P will act as process P' after performing action μ. The actions of the lts are the prefixes plus the output of a restricted name $\overline{x}\langle \nu y\rangle$, as reported below.

$$(\text{ACTIONS}) \quad \mu ::= \alpha \mid \overline{x}\langle \nu y\rangle$$

The *subject* of an input or output action is the channel used for communication and the *object* is the parameter. Morever we extend functions fn and bn to actions in the following way:

μ	$fn(\mu)$	$bn(\mu)$	kind
$x(y)$	$\{x,y\}$	$\emptyset$	(INPUT)
$\overline{x}\langle y\rangle$	$\{x,y\}$	$\emptyset$	(FREE OUTPUT)
$\overline{x}\langle \nu y\rangle$	$\{x\}$	$\{y\}$	(BOUND OUTPUT)
τ	$\emptyset$	$\emptyset$	(TAU)

As for processes, the function n (names) on labels is defined as the union of fn and bn.

We define the *early* labeled transition system semantics $\cdot \xrightarrow{\cdot}_\pi \cdot$ as the least relation satisfying α-conversion and the rules in Table 2.2. We omit the symmetric rules for (COMM), (CLOSE) and (PAR). The semantic rules expressing interaction between processes are (COMM) and (CLOSE), and in both cases the premise of the rule requires that two parallel processes P and Q are able to perform two complementary actions with the same prefix.

Note that the semantics is called early as when performing an input action, the substitution of the inputted name is done in advance (see Table 2.2). This implies that the name is guessed by the process executing the input. On the contrary, in the late version of the lts semantics for the π-calculus, the substitution is done in the communication rule (see rule Com^π). There is not difference at a semantics level (it can be proved that they are equivalent [102]), but they imply two different bisimulation.

The following theorem states the correspondence between the reduction semantics and the early lts semantics.

Theorem 2.3. *Let P be a π-calculus process, then*

- *if* $P \rightarrow_\pi P'$ *then there exists* Q' *such that* $P \xrightarrow{\tau}_\pi Q'$ *and* $P' \equiv_\pi Q'$
- *if* $P \xrightarrow{\tau}_\pi P'$ *then there exists* Q' *such that* $P \rightarrow_\pi Q'$ *and* $Q' \equiv_\pi P'$.

Example 2.2. We now use the lts to show the example we gave for the reduction semantics. Applying the rules from table 2.2 we get

$$\dfrac{\dfrac{\dfrac{}{x(z).\overline{z}\langle y\rangle.0 \xrightarrow{x(a)}_\pi \overline{a}\langle y\rangle.0}\,(\text{INP}^\pi) \quad \dfrac{\dfrac{}{\overline{x}\langle a\rangle.0 \xrightarrow{\overline{x}\langle a\rangle}_\pi 0}\,(\text{OUT}^\pi)}{\overline{x}\langle a\rangle.0 \mid \overline{x}\langle b\rangle.0 \xrightarrow{\overline{x}\langle a\rangle}_\pi 0 \mid \overline{x}\langle b\rangle.0}\,(\text{PAR}^\pi)}{x(z).\overline{z}\langle y\rangle.0 \mid \overline{x}\langle a\rangle.0 \mid \overline{x}\langle b\rangle.0 \xrightarrow{\tau}_\pi \overline{a}\langle y\rangle.0 \mid 0 \mid \overline{x}\langle b\rangle.0}\,(\text{COM}^\pi)}{P \xrightarrow{\tau}_\pi P_4}\,(\text{RES}^\pi)$$

$$(\text{INP}^\pi)\ \frac{}{x(z).P \xrightarrow{x(y)}_\pi P\{y/z\}} \qquad (\text{OUT}^\pi)\ \frac{}{\overline{x}\langle y\rangle.P \xrightarrow{\overline{x}\langle y\rangle} P}$$

$$(\text{TAU}^\pi)\ \frac{}{\tau.P \xrightarrow{\tau}_\pi P} \qquad (\text{MAT}^\pi)\ \frac{P \xrightarrow{\mu}_\pi P'}{[x=x]P \xrightarrow{\mu}_\pi P'}$$

$$(\text{SUM}^\pi)\ \frac{P \xrightarrow{\mu}_\pi P'}{P+Q \xrightarrow{\mu}_\pi P'} \qquad (\text{COM}^\pi)\ \frac{P \xrightarrow{\overline{x}\langle y\rangle}_\pi P' \quad Q \xrightarrow{x(y)}_\pi Q'}{P \mid Q \xrightarrow{\tau}_\pi P' \mid Q'}$$

$$(\text{REP}^\pi)\ \frac{P \mid !P \xrightarrow{\mu}_\pi P'}{!P \xrightarrow{\mu}_\pi P'} \qquad (\text{CLO}^\pi)\ \frac{P \xrightarrow{\overline{x}\langle \nu z\rangle} P' \quad Q \xrightarrow{x(z)} Q'}{P \mid Q \xrightarrow{\tau} (\nu z)\, P' \mid Q'} \quad z \notin fn(Q)$$

$$(\text{RES}^\pi)\ \frac{P \xrightarrow{\mu}_\pi P'}{(\nu z)\, P \xrightarrow{\mu}_\pi (\nu z)\, P'} \quad z \notin n(\mu) \qquad (\text{OPE}^\pi)\ \frac{P \xrightarrow{\overline{x}\langle z\rangle}_\pi P'}{(\nu z)\, P \xrightarrow{\overline{x}\langle \nu z\rangle}_\pi P'} \quad z \neq x$$

$$(\text{PAR}^\pi)\ \frac{P \xrightarrow{\mu}_\pi P'}{P \mid Q \xrightarrow{\mu}_\pi P' \mid Q} \quad bn(\alpha) \cap fn(Q) = \emptyset$$

Table 2.2: Labeled Transition System for π

2.2.5 Bisimulations

Now that we have defined the lts, we can give a definition of bisimulation based on it. We report the definition of *strong early bisimilarity*, one of the basic behavioural equivalences defined on π processes. As hinted before, this is not the only existing bisimulation [102].

Definition 2.6 (Bisimilarity). A binary symmetric relation $\mathcal{S}$ on processes is an *early bisimulation* if and only if $P\,\mathcal{S}\,Q$ and $P \xrightarrow{\mu} P'$ implies $\exists Q' : Q \xrightarrow{\mu} Q' \wedge P'\,\mathcal{S}\,Q'$.
P is *early bisimilar* to Q ($P \dot{\sim}_\pi Q$) if and only if $P\,\mathcal{S}\,Q$ for some early bisimulation $\mathcal{S}$.

Note that this bisimulation is called early because of the type of semantics we have chosen. In the case we choose a late definition we would get the late bisimulation, resulting into a coarser equivalence.

The following expected result is well known (a proof can be found in [102])

Theorem 2.4 (Characterisation). *The two equivalences $\sim_\pi$ and $\simeq^c_\pi$ coincide.*

The previous relations are called *strong* because they distinguish also processes that differ only by internal actions. It is interesting in some cases to abstract over τ actions and consider bisimulation with respect to visible actions only. Let $\Longrightarrow$ be the

$$(\text{TV-NAME})\frac{}{\Gamma, x:T \vdash x:T}$$

$$(\text{T-PAR})\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \mid Q} \qquad (\text{T-NIL})\frac{}{\Gamma \vdash 0}$$

$$(\text{T-REP})\frac{\Gamma \vdash P}{\Gamma \vdash !P} \qquad (\text{T-RES})\frac{\Gamma, x:T \vdash P}{\Gamma \vdash (\nu x)\, P}$$

$$(\text{T-INP})\frac{\Gamma \vdash z:[\tilde{T}] \qquad \Gamma, \tilde{x}:\tilde{T} \vdash P}{\Gamma \vdash z(\tilde{x}).P} \quad (\text{T-OUT})\frac{\Gamma \vdash z:[\tilde{T}] \qquad \Gamma \vdash \tilde{x}:\tilde{T} \qquad \Gamma \vdash P}{\Gamma \vdash \overline{z}\langle\tilde{x}\rangle.P}$$

Table 2.3: Structural Typing for the π-calculus.

reflexive and transitive closure of $\xrightarrow{\tau}$ and $\stackrel{\mu}{\Longrightarrow}$ be $\Longrightarrow\xrightarrow{\mu}\Longrightarrow$. Moreover let $\xrightarrow{\hat{\mu}}$ be $\longrightarrow$ if $\mu = \tau$, $\xrightarrow{\mu}$ otherwise. The definition of *weak early bisimulation* ($\approx$) is obtained by replacing $Q \xrightarrow{\mu} Q'$ with $Q \stackrel{\hat{\mu}}{\Longrightarrow} Q'$ in Definition 2.6.

2.2.6 Types for π-calculus

Structural types for the π-calculus have been proposed first by Pierce and Sangiorgi [96] as a simplified version of Milner's sorts [81], which are very similar to the nominal types described below.

Structural Types

The basic structural type system for the π-calculus is based on an environment Γ, a partial function associating an exchange type, describing the objects of the communication, to each name (and therefore, *each channel*). The formal definition of types and environments is

$$S, T ::= [\tilde{T}] \qquad (\text{TYPES})$$

$$\Gamma ::= \Gamma, x:T \mid \emptyset \qquad (\text{TYPE ENV})$$

The typing rules are given in Table 2.3, where we use the shorthand notation $\tilde{x}:\tilde{T}$ for $x_1:T_1,\ldots,x_n:T_n$ where $\tilde{x} = x_1,\ldots,x_n$ and $\tilde{T} = T_1,\ldots T_n$.

The type system satisfies the standard property of preserving types under reduction, and guarantees that well-typed processes will not incur in communication errors.

$$\frac{}{\Delta, S : \tilde{T} \Vdash S : \tilde{T}}$$

$$(\text{T-INP})\frac{\Gamma \vdash z : S \qquad \Delta \Vdash S : \tilde{T} \qquad \Gamma, \tilde{x} : \tilde{T} \vdash_\Delta P}{\Gamma \vdash_\Delta z(\tilde{x}).P}$$

$$(\text{T-OUT})\frac{\Gamma \vdash z : S \qquad \Delta \Vdash S : \tilde{T} \qquad \Gamma \vdash \tilde{x} : \tilde{T} \qquad \Gamma \vdash_\Delta P}{\Gamma \vdash_\Delta \overline{z}\langle\tilde{x}\rangle.P}$$

Table 2.4: Structural Typing for the π-calculus.

Nominal Types

The basic nominal type system for the π-calculus is based on a set of *nominal types* $A, B \in \mathcal{N}_s$, an environment Γ associating a nominal type to each name, and a set of type definitions Δ, containing associations between nominal types and exchange types. Both Γ and Δ are partial functions. The formal definition of types and environments is

$$S, T ::= A \in \mathcal{N}_s \qquad (\text{TYPES})$$

$$\Gamma ::= \Gamma, x : T \mid \emptyset \qquad (\text{TYPE ENV})$$

$$\Delta ::= \Delta, S : \tilde{T} \mid \emptyset \qquad (\text{TYPE DEF})$$

We denote the empty tuple of types by $\diamond$. The typing rules are a straightforward adaptation of those for structural typing. In Table 2.4 we give the rules for definitions and input and output, the other ones are unchanged.

Nominal types are often used for real programming languages, and we will not discuss their advantages from a software engineering perspective here. For a discussion, see [109]. From the theoretical point of view, they have the advantage of typing terms which would require recursive type systems to type, without introducing any additional burden. On the other hand, typability depends on the definitions Δ, and is not only a property of a term.

The type system satisfies the standard property of preserving types under reduction, and guarantees that well-typed processes will not incur in communication errors.

2.3 Encodings

In the recent years, numerous calculi have appeared and concurrency theory has made important the issue of exploring their expressiveness. Most of these calculi are Turing-equivalent, i.e. they can be encoded by Turing machines. Nevertheless there are some reasonable critiria that an encoding may not respect, e.g. preserving parallel composition.

Formally, given two languages $\mathcal{L}$ and $\mathcal{L}'$, an encoding of $\mathcal{L}$ into $\mathcal{L}'$ is a function $[\![\cdot]\!] : \mathcal{L} \to \mathcal{L}'$ which given a program $l \in \mathcal{L}$ returns a corresponding program $l' \in \mathcal{L}'$. According to [92], an encoding for concurrent languages is *uniform*, if it translates the parallel operator homomorphically, and if it respects permutations on free names

$$[\![P|Q]\!] = [\![P]\!]|[\![Q]\!] \tag{2.1}$$

$$\forall\sigma\ \exists\theta\ [\![P\sigma]\!] = [\![P]\!]\theta \tag{2.2}$$

Condition (2.1) states that the degree of parallelism in the system must be preserved by the encoding, condition (2.2) states that the structure of the encoding respects permutations of free names. Since we are interested in the problem of encoding specific constructs that may occur in π-calculus-like terms, it is sensible to strengthen condition (2.2) to account for arbitrary substitutions. In fact, a process can be syntactically placed in the scope of an input, that in π-calculus behaves like a substitution. The following example illustrates our point.

Example 2.3 (Encoding Mismatching). Mismatch is an operator sometimes considered in the π-calculus: it consists of the production $P ::= [x \neq y].P$, and its semantics can be defined both by an lts rule or by a structural congruence rule:

$$\frac{P \xrightarrow{\mu} P'}{[x \neq y]P \xrightarrow{\mu} P'}\ x \neq y \qquad [x \neq x]P \equiv_\pi 0$$

Consider, for instance, the homomorphic encoding of π with mismatch ($\pi^{\neq}$ for short) in π with infinite products, where mismatching is translated as follows:

$$[\![[x \neq y]P]\!] \triangleq \begin{cases} \Pi_{w \in \mathcal{N}\setminus\{y\}}[x = w]P & \text{if } x \neq y \\ 0 & \text{otherwise} \end{cases}$$

This encoding is uniform and is correct in a very strong sense: $[\![[x \neq y]P]\!] \sim [x \neq y]P$ and $[\![[x \neq x]P]\!] \equiv_\pi [x \neq x]P$. Nonetheless it is not satisfactory because it does not respect arbitrary substitutions:

$$(\nu z)(\overline{z}\langle a, a\rangle | z(x,y).[x \neq y]P) \xrightarrow{\tau} 0 \tag{2.3}$$

($x \neq y$, for the input on z to be defined) and for the encoding to be meaningful, we would expect an equivalent behaviour by its translation, whereas

$$(\nu z)(\overline{z}\langle a, a\rangle | z(x,y).[\![[x \neq y]P]\!]) \xrightarrow{\tau} P\{a/x\}\{a/y\} \tag{2.4}$$

The problem is that the term $[\![[x \neq y]P]\!]$ contains syntactically the term $[x = x]P$ that is not affected by the changes made on y after the translation: it does not respect arbitrary substitutions.

□

Unfortunately, all the conditions we have seen above are not enough. According to Palamidessi [92], an encoding preserves a *reasonable* semantics if it

> "...distinguishes two processes P and Q whenever there exists a maximal (finite or infinite) computation of Q in which the intended observables (some visible actions) are different from the observables in any (maximal) computation of P."

In [95], Philips and Vigliotti give a formalisation of the statement above. In the following let a computation be a sequence of reductions $P \to \ldots \to P_i \to \ldots$ and let $Obs(\mathcal{C}) = \{x \mid \exists j.P_j \downarrow x\}$.

Definition 2.7 ([95]). Let $\mathcal{L}, \mathcal{L}'$ be process languages. An encoding $[\![\cdot]\!] : \mathcal{L} \to \mathcal{L}'$ is observation-respecting if for any $P \in \mathcal{L}$,

(a) for every computation $\mathcal{C}$ of P there exists a maximal computation $\mathcal{C}'$ of $[\![P]\!]$ such that $Obs(\mathcal{C} = Obs(\mathcal{C}')$

(b) for every computation $\mathcal{C}$ of $[\![P]\!]$ there exists a maximal computation $\mathcal{C}'$ of P such that $Obs(\mathcal{C} = Obs(\mathcal{C}')$

where a computation $\mathcal{C}$ is maximal if it cannot be extended, i.e. either $\mathcal{C}$ is infinite, or else it is of the form $P \to P_1 \ldots \to P_i$ where $P_i \not\to$.

Another appealing property that is not needed for the results in [92], but that is considered for example in [41], [88], and [87], is termination invariance. We consider a crucial property of a semantics to distinguish inactive processes (deadlocks) from processes involved in infinite internal computations (livelocks). We call *sensible* an encoding $[\![-]\!]$ which is (strongly) *uniform*, preserves a *reasonable* semantics, and distinguishes deadlocks from livelocks. We shall see some examples about encodings in the further chapters of this dissertation.

We conclude this section, by also mentioning a possible alternative to studying encodings of process calculi into others, which, somehow, is still something new which has not been really studied. The idea is to treat the problem with category theory. If it is possible to represent a process calculus with a category then we can see encodings like "functions" between them which satisfy certain requirements which should exactly be (some of) the conditions listed above.

2.4 The π_{Op} family

We now introduce some possible variants of the π-calculus, which will be needed in this dissertation.

The asynchronous π-calculus ($a\pi$) proposed independently by [62], and [20], is the sub-calculus without summations and matching, and where output can be prefixed only to the inactive process.

$$P ::= \ \overline{x}\langle y\rangle.0 \ \mid \ M \ \mid \ P|P \ \mid \ (\nu x)P \ \mid \ !P$$

$$M ::= \ 0 \ \mid \ x(y).P \ \mid \ \tau.P$$

Both the previous references show how to encode the synchronous output $\overline{x}\langle y\rangle.P$ in terms of the simpler asynchronous communication mechanism: the real difference with π consist in the absence of matching and, more remarkably, of choice.

The separate choice π-calculus (π^s) is the sub-calculus of π in which output and input prefixes cannot be present in the same summation. This restriction is captured by modifying the syntax of processes without affecting the semantic rules.

$$P ::= \ 0 \ \mid \ \Sigma_i \, \alpha_i^I.P_i \ \mid \ \Sigma_i \, \alpha_i^O.P_i \ \mid \ P|P \ \mid \ (\nu x)P \ \mid \ !P$$

$$\alpha^I ::= \tau \mid x(y)$$
$$\alpha^O ::= \tau \mid \overline{x}\langle y \rangle$$

In [88], Nestman and Pierce have shown how to encode input-guarded choice in $a\pi$.

We denote with π^m the calculus with mixed choice, but without matching. In [92], Palamidessi has proved that π^m is strictly more expressive than π^s. Nestmann [87] argues that "there are nevertheless good encodings between these calculi". Palamidessi's result is a quite important as it had been unknown for quite a few years. The gap between π^m and π^s is shown to exist by defininig a problem called *electoral system* where a group of processes have to elect a leader by sending a vote on a designed channel. It turns out that this leader election is not possible in π_s, i.e. when mixed choice is not available. The same result has been proved more recently in [95] for the Ambient Calculus.

All the sub-calculi that we have seen so far can be extended with matching or mismatching. We will denote the extended versions by appending apices to the calculus name. In the sequel, let π_{Op} be defined as $\{a\pi, \pi^s, \pi^m, a\pi^=, a\pi^{=,\neq}, \pi^{s,=}, \pi\}$. All the variants of the π-calculus we shall consider are summarised in Table 2.5.

	$\overline{a}.P$	$=$	$\neq$	$+_s$	$+_m$
$a\pi$	–	–	–	–	–
π^s	✓	–	–	✓	–
π^m	✓	–	–	–	✓
$a\pi^=$	–	✓	–	–	–
$a\pi^{=,\neq}$	–	✓	✓	–	–
$\pi^{s,=}$	✓	✓	–	✓	–
π	✓	✓	–	–	✓

Table 2.5: Some variants of the π-calculus.

Part I

Trust for Global Computing

Chapter 3

The SECURE project

As pointed out in the introduction, because of the infrastructure's dynamism, entities that offer services will be confronted with requests from unknown entities, and mobile entities will need to obtain services in unfamiliar, possibly hostile environments. A party facing such a complex world stands to benefit from interaction, but only if it can respond to new entities and assign meaningful privileges to them. The *Secure Environments for Collaboration among Ubiquitous Roaming Entities (SECURE)* project is designing a novel security approach that addresses these challenges. If successful, this approach will significantly benefit not only future systems but also various emerging mobile computing applications. It could also benefit collaborations over the Internet where correspondents' identities and intentions are difficult to establish with certainty.

This approach applies the human notion of trust. This naturally leads to a decentralized security management approach that can tolerate partial information, albeit one that has inherent risks for the trusting entity. Fundamentally, the ability to reason about trust and risk is what lets entities accept risk when interacting with other entities. The SECURE project seeks a formal basis for reasoning about trust and risk and for deploying verifiable security policies, embodied in a computational framework that is adaptable to various application scenarios. For example, consider the problem of routing messages in an ad hoc wireless network. An entity, or mobile node, with a message to send must rely on other nodes on the path to the intended destination to forward its message. Generally, the intermediate nodes might have no a priori relationship or agreement with the sender, which they might never have encountered before. Also, forwarding messages costs battery and processing power. Why should a sender rely on such nodes to help it? If multiple paths exist, which path should the sender put the most confidence in? These trusting decisions are informed by the degree to which the sender trusts the intermediate nodes to do the right thing based on observations of and experience with these nodes, their reputations, and possibly recommendations from third parties. These decisions are also mediated by the risk the sender takes. The sender probably requires less trust to send a low-importance message and more trust for a very important message that really needs to get through.

3.1 Understanding trust

We have pointed out in the introduction that humans use trust daily to promote interaction and accept risk in situations where they have only partial information. Trust lets one person assume that another will behave as expected. Despite the extensive study of trust in sociology, psychology, and philosophy, it remains an elusive concept that defies stringent definition. This is partly because trust is largely invisible and implicit in society. We saw that various definitions of trust have been offered, many of which depend on the author's viewpoint or the context in which he or she examines trust. Because of trust's multifaceted nature, it is difficult to form a unified definition. It is useful to examine dictionary definitions of trust to determine which are widely accepted. Common to these definitions are the notions of confidence, belief, faith, hope, expectation, dependence, and reliance on the integrity, ability, or character of a person or thing.

The SECURE project's approach is based on the premise that trust and risk are inexorably linked and must both be considered when making a decision about an ambiguous path whose outcome depends on another entity's actions.

3.2 Handling trusted interactions

The trust a principal needs for an interaction depends on the risk involved. This allows for appropriate security in pervasive environments without requiring excessive trust in straightforward cases.

When a system grants privileges to a principal, it expects the principal to use them in a particular way, for example, to update old address book entries with accurate information. However, the principal could deviate from this expected behavior, and the combined likelihood and severity of this is the risk of granting them a privilege.

3.2.1 Risk analysis

In SECURE, the risks of a trust-mediated action are decomposed by possible outcomes. Each outcome's risk depends on the other principal's trustworthiness (the likelihood) and the outcome's intrinsic cost. For example, an address update might itself be out-of-date or maliciously misleading. These two outcomes' costs would reflect the user's wasted time, and the likelihoods would depend on trust in the other party.

An outcome's costs could span a range of values. For example, a user might have received a correct phone book entry. This third outcome's cost could show a net benefit to the user, as the user might successfully use it later. However, if the number became out-ofdate by the time it was used, that would be a net loss. To reflect this uncertainty, you might represent the distribution of costs as a cost-PDF (probability density function).

Figure 3.1 illustrates a user contemplating a parameterized interaction with principal a. For each possible outcome, the user has a parameterized cost-PDF (a family of cost-PDFs) that represents the range of possible costs and benefits the user might incur should each outcome occur.

While the risk evaluator assesses the possible cost-PDFs, the trust calculator provides information t that determines the risk's likelihood based on the principal's iden-

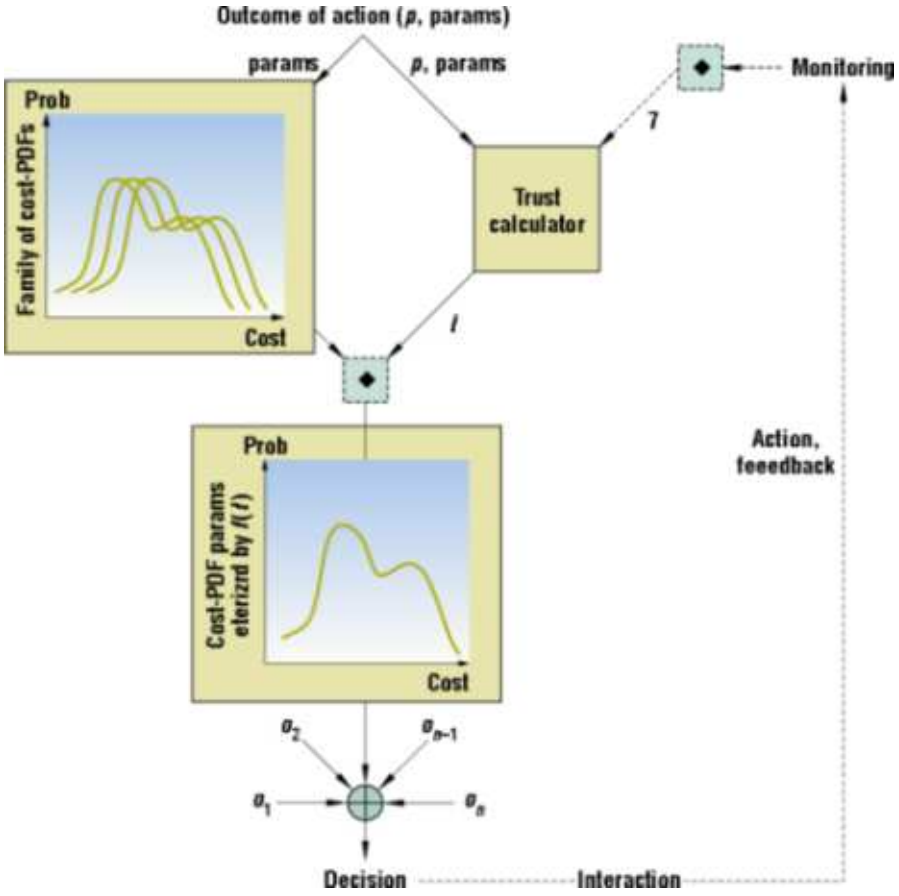


Figure 3.1: Decision making using trust and risk.

tity a and other parameters of the action. The risk evaluator then uses this trust information to select the appropriate cost-PDF.

Finally, the request analyzer combines all the outcomes' cost-PDFs to decide if the action should be taken or to arrange further interaction. Because any uncertainty is preserved right up to the decision point, this allows more complex decision making than simple thresholding, allowing responses such as `not sure` if there is not enough information.

For instance, if Liz's PDA received a phone number from Vinny's PDA, she might not think it was maliciously misleading based on her trust in Vinny's honesty. She might think it could be out-of-date, however, if Vinny had given her stale information before, attributing a higher risk to this outcome. Finally, she would consider the potential benefit of having a correct number, again moderated by Vinny's trustworthiness. Liz's PDA would do all these calculations on her behalf using its model of her trust beliefs, as Figure 3.2 illustrates. If the benefits outweighed the other outcomes' costs, the PDA would then accept the information. On the other hand, if John, a colleague from a competing research group, sent Liz an address book entry, her PDA might reject it after the same analysis because she did not know him. At this point, the request analyzer might seek out more information, maybe by discovering that John works with

Jean, who is trusted by Liz, or by interrupting Liz for confirmation.

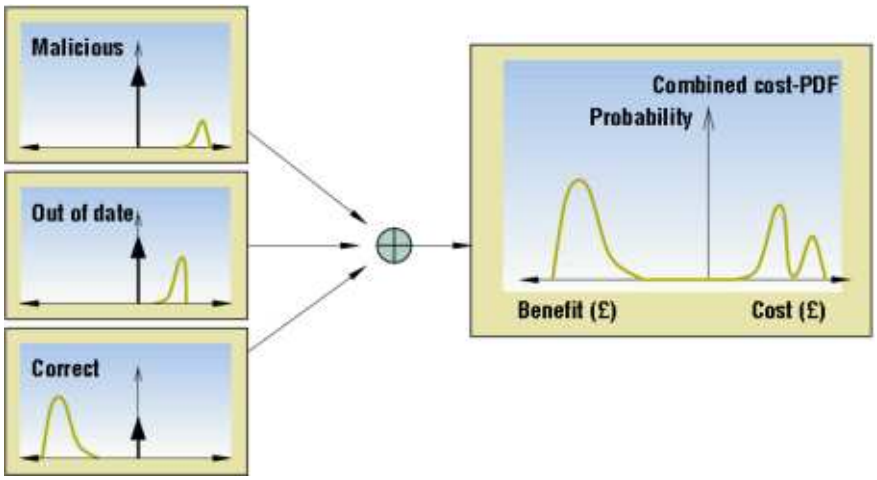


Figure 3.2: Illustration of risk analysis.

Interconnected address book categories can give structure to information [107]. By assigning each category a risk value and explicitly costing the user's time, sensitive entries (such as bank phone numbers) can naturally be protected with little user effort.

So this explicit risk analysis, which differentiates the SECURE approach from other trust-based approaches [3,120] balances the evidence that a principal is trustworthy against the risks if it is not. This allows sensible behavior in the face of uncertainty, but prevents abuse by incrementally updating trust assessments as more evidence becomes available.

3.2.2 Building trust

Recent work on trust management systems [16, 17, 44, 100] attempts to manage security in large-scale distributed networks by using credentials that delegate permissions. However, these systems focus on trust management's static element and neglect the dynamic component of trust formation. What does trust really consist of? Fundamentally, we base trusting decisions on trust information, encompassing evidence from personal observations of previous interactions and recommendations from partly trusted third parties. These two main sources of stored trust information let us dynamically form an opinion about another entity.

Personal observations of the entity's behavior, through recording outcomes of interactions, are essential in subjectively evaluating trustworthiness. These observations are evaluated against the principal's expected behavior to produce experiences [70]. The range of experience values reflects the effect of the observed outcome relative to the expected outcome, usually in terms of gain or loss. These values are ordered and classified into two sets, a trust-positive set and a trust-negative set. This evidence is aggregated with the evidence from previous interactions to give a comprehensive summary of the principal's interaction history.

Recommendations from trustworthy third parties can propagate trust in unknown entities by providing supporting evidence for decisions. The recommendation process

becomes more important when the trust evaluation based on observations is not precise enough. In such cases, an entity might need more information. Also, it could discard imprecise recommendations that provide little additional information. The decision, however, is left to the individual entity. Upon receiving a collaboration request, we can dynamically filter the available trust evidence to keep only what is relevant to the requested action. If no evidence from experience or recommendation is available for an entity, we must establish an initial trust value to encourage low-risk collaborations. We can determine this using many strategies [46]. This collaboration will provide further evidence on which we can base future trust formation. If enough evidence exists to reason about the entity's trustworthiness, then we will evaluate observations and recommendations to yield trust information. This information might be multidimensional, separate trust intervals might be formed for different aspects of trust in the interaction. We treat recommendation evidence separately from personal experience evidence, which has more influence on trust.

The trust model (see next chapter) operates using local trust policies. These local policies let the system use collected evidence and dictate the conditions in which the trust opinion, formed from the evidence, should be used. The policies also let us conditionally delegate trust evaluation to an outside entity, an important feature of the trust model. The difference between delegation and recommendation is that we delegate to entities similar to ourselves, which we might consider experts for the decision; in recommendation, however, we gather trust information from any principal in the environment and can seek more than one recommendation. We can also weigh recommendations according to our trust in the source as a recommender.

Our framework goes a step further than trust-based frameworks such as CONFIDANT [23] and similar approaches [4]. CONFIDANT, for instance, is designed for ad hoc network routing. No centralized and trusted network manager exists in such a network; each node must trust others to transmit its messages. Nodes can exchange reputation information to detect malicious nodes. A node might send reputation records that describe its first-hand experience with another node or trust records that encapsulate reputation information received from other nodes. Nodes depend on these records to make routing decisions. In our approach, a node's trust decision need not rely on exchanged reputation information but can also be delegated to another node. This leads to a more flexible range of trust policies and is more consistent with the human approach to trust formation.

3.3 Software framework

Even if we understand how to reason about trust formation and evolution and how to exploit trust in making access control decisions, we also need to ensure that we can feasibly implement the necessary algorithms for these processes in heterogeneous systems. So, the SECURE project has been developing a policy-neutral software architecture framework encompassing algorithms for trust management usable in various applications. Figure 3.3 illustrates the current version of our framework design.

When a principal a makes a request for interaction, the request passes through the application programming interface into the request analyzer. The request analyzer requests information about a from three sources: the entity recognition component,

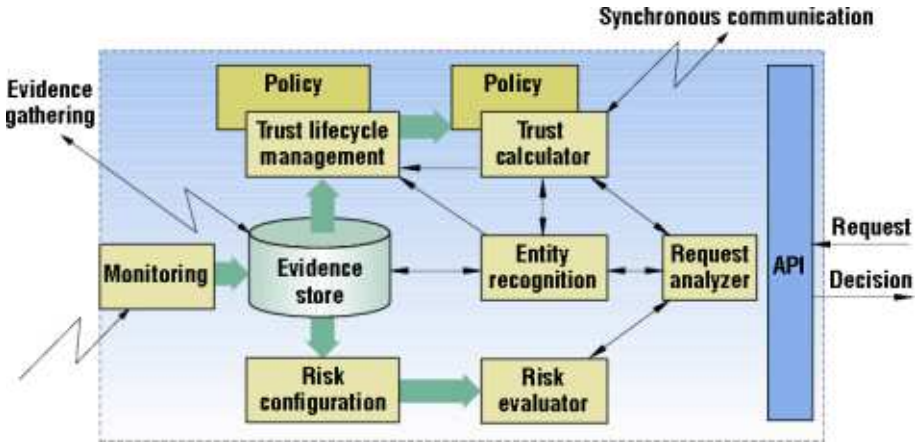


Figure 3.3: The framework for one party to an interaction. Large arrows signify data flows and thin arrows signify control flows.

the trust calculator, and the risk evaluator. The entity recognition component, which is responsible for recognizing new or previously encountered entities, requests verification that *a* is recognized (see [105]). Any other component can consult the entity recognition component to obtain recognition capabilities as necessary. Additionally, the request analyzer requests a trust calculation from the trust calculator. The trust calculator computes the least fixed point, as we discuss in Chapter4, using information gathered from the trust lifecycle management component and its local trust policy. As we mentioned earlier, the system can delegate the trust level calculation for *a* to another entity, thereby initiating synchronous communication with a remote entity. The trust calculator's local policies update on the basis of information fed from the trust lifecycle management component. This component handles trust formation, evolution, and exploitation on the basis of data drawn from the evidence store. The trust lifecycle management policy allows trust information to be weighted according to context specific criteria. The evidence store holds all trust- and risk-related data. It is updated with data from evidence gathering, such as recommendations and security updates collected in an asynchronous process, and from the monitoring component. The evidence store also responds to requests for recommendations from other entities. The monitoring component observes actual interaction with *a* and conveys the results of the interaction to the evidence store. The request analyzer also requests a risk assessment from the risk evaluator, which calculates the request's potential risk based on the local information stored in the risk configuration component, which is updated with information from the evidence store. The system assesses and aggregates all of the information it obtained about *a*. It returns the trust calculation and risk assessment to the request analyzer, which can then provide a decision to *a* regarding possible interaction.

3.4 Applications

Given the project's exploratory nature, it is important to evaluate the proposed mechanisms in the context of real applications. Here are two applications on which the

project has been working. Each application deals with numerous interacting entities. The entities might be strangers, and we cannot rely on the presence of a centralized service for security. Although the two applications are very different, the project has been developing instantiations of the framework presented earlier that will be used to support both of them.

3.4.1 Electronic purse

An electronic purse, or e-purse, is a device that stores money in electronic form and enables the transfer of money units to other e-purses. Mobile telephones are ideal candidates for implementing e-purses, they have smart cards for securely storing sums of money and are widespread enough for an epurse system to be deployed. In this scenario, you can transfer funds from the e-purse to the bank and vice versa via a GSM (Global System for Mobile Communications) or Internet connection; you make bank payments offline using SMS (Short Message Service) or Bluetooth.

An e-purse's main advantage over a traditional purse is that a person does not need to carry real money. When a real purse is lost or stolen, the money is also lost or stolen; with an e-purse, a thief must know the e-purse's PIN to access its contents. Also, an e-purse should maintain the anonymity property of payments, the buyer can pay the vendor without revealing his or her identity. A payment protocol that is particularly suited to the e-purse is the eCash protocol from DigiCash [37, 103]. In this protocol, each electronic coin has a unique serial number and the bank signs it using a key for that denomination. When a vendor who receives money cashes the e-purse contents with the bank, the bank detects cheating by comparing these coins and their serial numbers to those already cashed.

A principal in the e-payment system is a purse. While an e-purse typically is owned by a person, it might also be a payment device in a coffee or chocolate vendor. You recognize the principal through its public key, which is stored on its e-purse, `PubPurse`. The Entity Recognition subsystem ensures that the principal is what it claims to be. Both buyer and vendor principals receive a certificate containing their respective public key `PubPurse` signed by the bank: $\{\texttt{PubPurse}, n\}_{Bank}$. N denotes the epurse's serial number, which is used for principal identity information.

You can verify any request using the `PubPurse` of the initiator and its certificate. Because payment happens offline, the bank cannot be contacted to check the certificate's validity and the principal risks dealing with a principal that the bank recently blacklisted. To reduce the risk from replay attacks, where an attacker replays messages from other principals that it has overheard, `PubPurse` can be used in a challenge-response protocol relying on past interactions to increase confidence in the recognition level. For example, the ticket vending machine sends a request encrypted with the buyer's `PubPurse` asking how many tickets have been bought so far and the buyer should reply with the right answer encrypted with the vending machine's `PubPurse`.

In this system, the vendor who accepts e-cash for goods has the biggest risk. The vendor must wait until he or she goes online to deposit the cash to verify that it was not forged. There are thus two outcomes for a payment transaction: it succeeds if the bank accepts the money the vendor received, or it fails.

The first task of this application scenario's security administrator is to propose trust values and an ordering for them. Because you can link trust in a principal to

the number of positive experiences (the number of successful payment outcomes compared to the number of outcomes where the principal cheated) you can represent trust by a pair (m,n) of non-negative integers. Integer m represents the number of successful outcomes associated with the principal; n represents the number of unsuccessful outcomes.

The bottom, or unknown value is $(0,0)$. A trust value (m_1,n_1) represents more trust than (m_2,n_2) when the number of successful outcomes is superior and the number of unsuccessful outcomes is inferior ($m_1 \geq m_2$ and $n_1 \leq n_2$). The information ordering on the set of values is defined by considering a trust value (m_1,n_1) as conveying more information than a value (m_2,n_2) if it is possible to start from the value (m_1,n_1) and then perform some additional number of interactions, ending up with the value (m_2,n_2). This intuition leads us to define the information ordering by taking $(m_1,n_1) \sqsubset (m_2,n_2)$ if and only if $(m_1 \leq m_2)$ and $(n1 \leq n2)$. Figure 3.4 illustrates this partial ordering of trust values. In next chapter, we shall treat these things more formally.

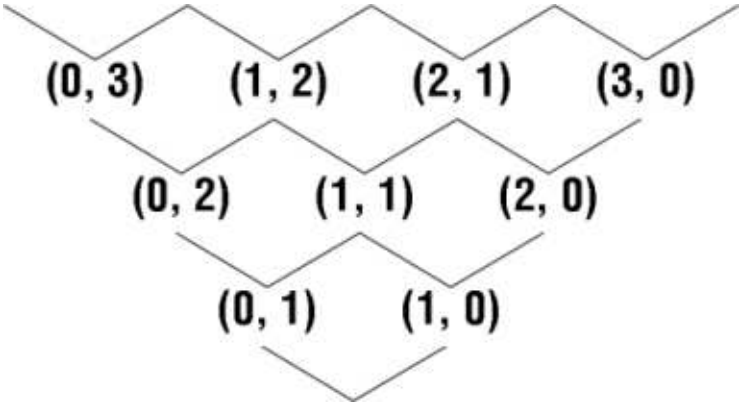


Figure 3.4: Trust ordering for an e-purse.

A principal's trust in another is mainly based on its experience with that principal. It can also be influenced by observations of that principal. In the e-purse system, observations take the form of messages (principal a cheated or principal a is honest) exchanged via principals, perhaps as an annex to payment messages. The security administrator can write the system's trust management policy to update the trust policy based on these messages. Obviously, such a message from the bank is more credible than one from another principal.

The second task for this system's security administrator is to design the risk policy. In this scenario, we deal with two mutually exclusive outcomes. The cost associated with the payment action is bounded by the amount of money being exchanged. One risk policy is to consider the probability of success to be independent of the cost (for example, an attacker is just as likely to cheat for 5 euro as he is for 50 euro). In this case, the cost-PDF is flat. Another possible risk policy is to consider the probability of cheating to be proportional to the amount being transferred. In this case, the probability of an unsuccessful outcome for an amount s of money could be defined as: $a \times c \times s^2$, where a is the trust parameter for the paying principal, calculated as $n/(n+m)$ for the trust values except for bottom (where $n+m=0$), and a is a constant defined to scale the

risk value calculated to the range $[0..1]$. This cost-PDF models a scenario in which the risk of cheating increases to the square of the sum of money involved in the transaction. A third PDF shown in Figure 3.5 is a linear curve defined as $\texttt{amount} \times T$, where T is based on the average degree of cheating that occurs in the e-payment system. The risk policy returns this function when the trust value calculated for a principal is bottom $(0,0)$.

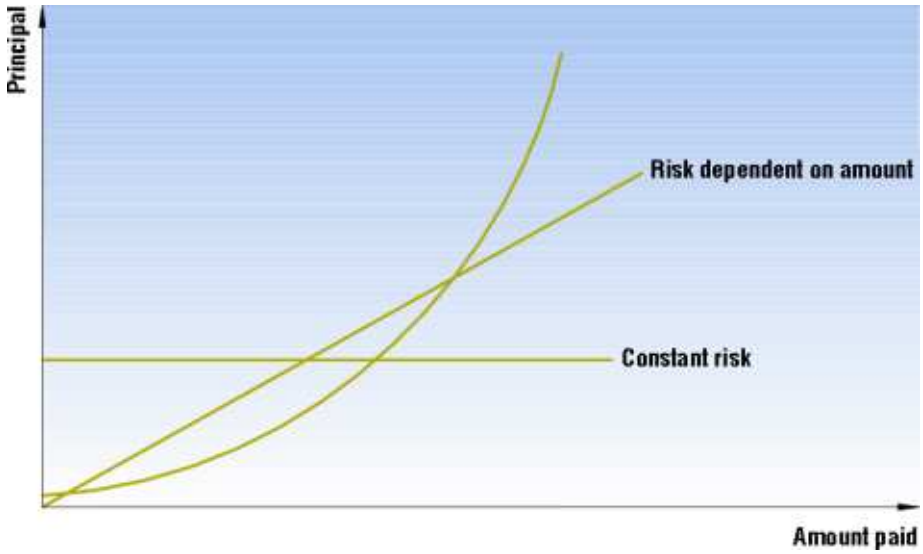


Figure 3.5: Risk PDFs for the e-purse application.

The security administrator's final task is to define the security policy. This policy considers the calculated risk and trust values and decides whether to permit the action. In this system, one possible policy is to print a warning message to the user's screen if the calculated risk value for the transaction exceeds a specified threshold. For example, using pseudocode,

```
if (riskOfFailurePDF(amount) ≤ ThresholdPay) return YES;
if (riskOfFailurePDF(amount) ≥ ThresholdDontPay) return NO;
else return DONT-KNOW.
```

The `riskOfFailurePDF` is the PDF returned by the risk policy and amount is the sum being paid in the transaction; `ThresholdPay` and `ThresholdDontPay` are the thresholds below and above that the e-purse pays and does not pay, respectively. Between these values, the user must intervene in the decision. For example, Carl might try to buy chocolate from a vending machine. The vending machine trusts Carl, whose trust value is $(13,7)$. Assuming proportional risk, with $c = 0.2$, $\texttt{ThresholdPay} = 0.1$, and $\texttt{ThresholdDontPay} = 0.2$, the answer would be `YES` for Carl to purchase 1 euro worth of chocolate but `N0` for 2 euro worth of chocolate.

3.4.2 Collaborative gaming

The demand is increasing for applications such as collaborative gaming, where players in different locations can participate in the same gaming session using portable devices. When money is at stake in these games, security measures are necessary because unknown and potentially untrustworthy players might enter gaming sessions.

Blackjack is a popular card game in which players gamble with a dealer over the value of a hand. In our prototype implementation, people play blackjack over a mobile ad hoc network using laptop computers or PDAs. For example, suppose Alice takes the 8 a.m. commuter train into the city for work every weekday. She wants to play an interactive game to pass the time, so she joins an ad hoc wireless network to see what collaborative gaming applications are available. She discovers an ongoing blackjack session in which Bob is the dealer and she requests admission to the game. Bob must decide whether or not to admit her; he must decide if he recognizes Alice, how much he trusts her as a gaming opponent, and how much risk is associated in playing blackjack with her.

The entity recognition process determines whether or not Bob has interacted with Alice before, as well as the level of confidence in recognizing her. If this is Bob's first time interacting with Alice, he might need to rely on trusted third-party recommendations about Alice. Recommendations can be exchanged verbally, by email, by distributed post-its, and more, and then recorded as evidence. On the basis of evidence from his own observations, from recommendations, or both, Bob can determine a trust level for Alice. Bob needs to trust that Alice will not cheat, spoof, or collude while he is gaming with her.

Note that this same trust is required in the casino version of blackjack. Also, because the dealer's odds of winning are more favorable than the players' odds, the players must consider the entity in the dealer role trustworthy. So the right to assume the advantageous dealer role can be considered a privilege earned through fair, trustworthy game playing. Alice must prove some level of trustworthiness if she wants to enter a game as a dealer.

As with e-purse, trust values and an ordering for them are necessary. In blackjack, trust is based on factors such as Alice's typical playing strategy and whether she pays her gambling debts. The security administrator generates an ordering similar to that in the e-purse application that represents both positive and negative experience results. This information could also help determine the risk of interacting with Alice. For example, information about what players are typically playing in the same games as Alice might enable Bob to reason about the probability that Alice will collude with other players. Bob can monitor this information throughout the course of the interaction and evaluate and store other players' results of playing blackjack with Alice. With his stored information, Bob might also need to respond to requests for recommendations from other players. Bob might want to delegate his trusting decision altogether. In this case, he must begin recognition of remote entities he could interact with for the purpose of delegation. Based on this information collection, Bob can assess whether he recognizes Alice, to what extent he is certain of recognition, how much he trusts Alice, and if that trust level is enough to interact with Alice given the overall risk inherent in the interaction.

Initial results of testing the prototype implementation [55] show that it reacts cor-

rectly to changes in an entity's interactive behavior, it adjusts trust levels and implements trust-based interaction accurately as trust rises and falls. The framework's design is continuously being refined, in particular, the approach to trust formation and evolution to address issues such as collusion and framing. We are also examining the application of trust to rolebased access control.

Chapter 4

Trust Domains

...trust is a social good to be protected just as much as the air we breath or the water we drink. When it is damaged, the community as a whole suffers; and when it is destroyed, societies falter or collapse!
— Bok, 1978, taken from [77]

As hinted in the previous chapters, we think of the standard deployment of a trust management system as consisting of a "*trust engine*" and a "*risk engine*" coupled together as part of a "*principal*." The trust engine is responsible for updating trust information based on direct and indirect observations or evidence, and to provide trust information to the risk engine as input to its procedures for handling requests. The risk engine will feed back information on principals' behaviours as updating input to the trust engine. Abstracting over this point of view, we single out as central issues for our trust model the aspects of trust *formation*, *evolution*, and *propagation*. The latter is particularly important in our intended application domain, where the set of active principals is large and open-ended, and centralised trust and ad-hoc methods of propagation of its variations make little sense. An important propagation mechanism is *referencing*, whereby principals cooperate to implement complex, intertwined "global" trusting schemes. Just to pin down the idea, bank b may be willing to trust client c to an overdraft limit x only if bank b' trusts it at least up to $2x/3$, and c itself does not trust d, a crook known to b. Note that, in literature, this term is often mixed up with *delegation*. In our understanding, delegation is transferring the possibility of making a decision to another principal. On the contrary referencing is just taking into account another principal's trust (opinion), which does not imply that it will be fully considered. Referencing has important consequences for trust representation, because it brings forward the idea of *trust policy*, i.e. algorithmic rules – such as bank b's above – to evaluate trust requests. In principle, trust among principals can be represented straightforwardly, as a function from pairs of principals to trust levels,

$$\texttt{GTrust} : \texttt{Principal} \longrightarrow \texttt{Principal} \longrightarrow \texttt{TrustDegree}$$

where $\texttt{GTrust}(a)$ is a function which associates to each principal b the value of a's trust in b. Delegation leads to model local policies, say b's, as functions

$$\texttt{TrustPolicy} : \texttt{GTrust} \longrightarrow \texttt{Principal} \longrightarrow \texttt{TrustDegree}$$

where the first argument is (a representation of) a universal trust function that b needs, to know b''s level of trust in c and whether or not c trusts d.

The domain of `TrustPolicy` makes the core of the issue clear: we are now entangled in a "*web of trust*," whereby each local policy makes reference to other principals' local policies. Technically, this means that policies are defined by mutual recursion, and global trust is the function determined collectively by the web of policies, the function that stitches them all together. This amounts to say that `GTrust` is the least *fixpoint* of the universal set of local policies, a fact first noticed in [116] which leads straight to *domain theory* [104].

Domains are kinds of partially ordered sets which underpin the semantic theory of programming languages and have therefore been studied extensively. Working with domains allows us to use a rich and well-established theory of fixpoints to develop a theory of security policies, as well as flexible constructions to build structured trust domains out of basic ones. This is precisely the context and the specific contribution of this chapter, which introduces a novel domain-like structure, the *trust structures*, to assign meaning and compute trust functions in a GC scenario. We anticipate that, in due time, techniques based on such theories will find their way as part of trust engines.

As domains are (complete) partial orders and trust degrees naturally come equipped with an ordering relation, a possible way forward is to apply the fixpoint theory to `TrustDegree` viewed as a domain. This is indeed the way of [116] and, as we motivate below, it is not a viable route for GC. There are very many reasons in a dynamic "web of trust" why a principal a trying to query b about c may not get the information it needs: b may be temporarily offline, or in the process of updating its policy, or experiencing a network delay, or perhaps unwilling to talk to a. Unfortunately, the fixpoint approach would in such cases evaluate the degree of trust of a in c to be the lowest trust level, and this decision would be wrong. It would yield the wrong semantics. Principal a should not distrust c, but accept that it has not yet had enough information to make a decision about c. What is worse with this confusion of "trust" with "knowledge," is that the information from b could then become available a few milliseconds after a's possibly wrong decision.

We counter this problem by maintaining two distinct order structures on trust values: a *trust ordering* and an *information ordering*. The former represents the degree of trustworthiness, with a least element representing, say, absolute distrust, and a greatest element representing absolute trust; the latter the degree of precision of trust information, with a least element representing no knowledge and a greatest element representing certainty. The domain-theoretic order used to compute the global trust function is the information order. Its key conceptual contribution is to introduce a notion of "*uncertainty*" in the trust value principals obtain by evaluating their policies. Its technical contribution is to provide for the "semantically right" fixpoint to be computed.

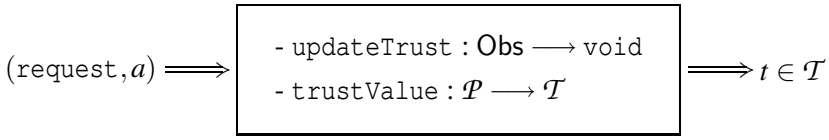
Following this lead, we introduce and study trust structures of the kind $(D, \preceq, \sqsubseteq)$, where the two order relations over the set D, carry the meaning illustrated above. We then provide constructions on trust structures – including an "interval" construction which endows complete lattices with a natural notion of uncertainty and lifts them to trust structures – and use the results to interpret a toy, yet significant policy language. We believe that introducing the information ordering alongside the trust ordering is a significant step towards a model of trust feasible in a GC scenario; it is a major point of departure from the work of Weeks [116], and the central contribution of this chapter.

4.1 A Model for Trust

The introduction has singled out the traits of trust most relevant to our computational scenario: trust involves *entities*, has a *degree*, is based on *observations* and ultimately determines the *interaction* among entities. Our model will target these aspects primarily.

Entities will be referred to as *principals*. They form a set $\mathcal{P}$ ranged over by $a, b, c, \ldots$ and p. We assume a set $\mathcal{T}$ of *trust values* whose elements represent degrees of trust. These can be simple values, such as $\{\texttt{trusted}, \texttt{distrusted}\}$, or also structured values, e.g. pairs where the first element represents an action, say access a file, and the second a trust level associated to that action; or perhaps vectors whose elements represent benevolence in different situations.

As trust varies with experience, a model should be capable of dealing with observations resulting from the principal's interaction with the environment. For clarity, let us isolate the principal's trust management from the rest of its behaviour, and think of each principal as having a "trust box," that is an "object" module containing all of its trust management operations and data. Thinking "object-oriented," we can envision this as an object used by the principal for processing trust related information. Assuming a set of observations Obs relevant to the concrete scenario of interest, the situation could be depicted as below.



The principal sends the message $\texttt{trustValue}(a)$ to ascertain its trust in another principal a, and the message $\texttt{updateTrust}(o)$ to add the observation o to its trust state.

In this chapter, we only focus on the trust box and assume, without loss of generality, that the remaining parts of the principal interact with it via appropriately exported methods.

4.1.1 Modelling the Trust Box

Principals' mutual trust can be modelled as a function which associates to each pair of principals a trust value t in $\mathcal{T}$:

$$m : \mathcal{P} \longrightarrow \mathcal{P} \longrightarrow \mathcal{T}$$

Function m applied to a and then to b returns the trust value $m(a)(b) \in \mathcal{T}$ expressing a's trust in b. This however does not mean that a single principal's trust can be modelled as a function from $\mathcal{P}$ to $\mathcal{T}$, since a's trust values may depend on other principals' values. For instance, a may wish to enforce that its trust in c is b's trust in c. Similarly, we may be willing to receive a message from unknown sources, provided somebody we know trusts the sender. This mechanism of relying on third-party assessments, known as *delegation*, is fundamental in all scenarios involving cooperation, including computational paradigms such as GC.

This leads us to a refined view of a principal's trust as being defined by a *policy*. According to such a view, each principal has a local policy π which contributes by way of delegation to form the global trust m. A policy expresses how the principal computes trust information given not just his own beliefs, but also other principals' beliefs. It follows that, as anticipated in the introduction, a's policy π_a has the type below, whose first argument represents the knowledge of third principals' policies that a needs to evaluate π_a.

$$\pi_a : (\mathcal{P} \longrightarrow \mathcal{P} \longrightarrow \mathcal{T}) \longrightarrow (\mathcal{P} \longrightarrow \mathcal{T})$$

We leave unspecified the way a policy is actually defined, as this definitely depends on the application. We study a relevant example of policy language in 4.3.

By collecting together the individual policies, we obtain a function $\Pi \triangleq \lambda p : \mathcal{P}.\pi_p$ whose type is (isomorphic to)

$$\Pi : (\mathcal{P} \longrightarrow \mathcal{P} \longrightarrow \mathcal{T}) \longrightarrow (\mathcal{P} \longrightarrow \mathcal{P} \longrightarrow \mathcal{T}).$$

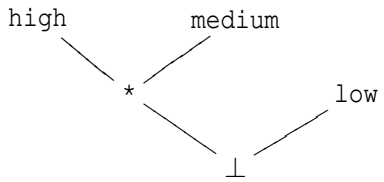
To interpret this collection of mutually recursive local policies as a global trust function m, we apply some basic domain theory, namely fixpoints and complete partial orders.

So, requiring $\mathcal{T}$ to be a CPO, which implies that the pointwise extension to $\mathcal{P} \to \mathcal{P} \to \mathcal{T}$ is a CPO too, and taking Π to be continuous, we can define the global trust as $m \triangleq \mathrm{fix}(\Pi)$, the *least fixpoint* of Π.

The question arises as to what order to take for $\sqsubseteq$. We maintain that it *cannot* be the order which measures the degree of trust. An example is worth many words. Let $\mathcal{T}$ be the CPO $\{\texttt{low} \leq \texttt{medium} \leq \texttt{high}\}$, where `low` stands for low trusted, `medium` for medium trusted and `high` for high trusted. Consider a policy π_a which delegates to b the degree of trust to assign to c. In this setup, a will assign `low` trust to c when it is not able to gather information about c from b. This however would be an erroneous conclusion when taking a decision, as the interruption in the flow of information does not bear any final meaning about trust, its most likely cause being a transient network delay that will soon be resolved. The right conclusion for a to draw is not to distrust c, but to acknowledge that it does not know (yet) whether or not to trust c. In other words, if we want to model dynamic networks, we cannot allow confusion between "`don't trust`" and "`I don't know`:" the latter only means lack of evidence for trust or distrust, the former implies a trust-based, possibly irreversible decision.

In order to make sense of our framework in a GC scenario, we need to acknowledge that principals only have a partial knowledge of their surroundings and, therefore, of *their own* trust values. We thus consider *approximate* trust values which embody a level of *uncertainty* as to which value we are actually presented with. Specifically, beside the usual *trust value ordering*, we equip trust values with a *trust information ordering*. While the former measures the degree of trustworthiness, the latter measures the degree of uncertainty present in our trust information, that is its information content. We will assume that the set $\mathcal{T}$ of (approximations of) trust values is a CPO with an ordering relation $\sqsubseteq$. Then $t \sqsubseteq t'$ means that t' "refines" t, by providing more information than t about what trust value is being approximated. With this understanding the continuity of Π is a very intuitive assumption: it asserts that the better determined the information from the other principals, the better determined is value returned by the policy. An example will help to fix these ideas.

Example 4.1. Let us refine the set of trust values $\mathcal{T}$ discussed previously by adding some new intermediate values $\{\bot, *, \texttt{low}, \texttt{medium}, \texttt{high}\}$, and consider the information ordering $\sqsubseteq$ specified by the following Hasse diagram.



Note that this ordering says nothing about what is more trust. It focus only on the quantity of information a principal has. The limit of any chain reflects the finest information. The element $*$ represents the uncertainty as to whether `high` or `medium` holds, while $\bot$ gives no hint at all about the actual trust value. Suppose we have a set of principals $\mathcal{P} = \{a, b, c\}$ with the following policies.

	a	b	c
a	`high`	$\bot$	`ask` b
b	$*$	`high`	`low`
c	`ask` b	`high`	`high`

where each row is a principal's policy. For instance the third row gives c's policy: c's trust in a is b's trust in a; c's trust in b is `high`. The computation of the least fixpoint happens by computing successive approximations following the standard theory, as illustrated below. We start from the least possible trust function:

	a	b	c
a	$\bot$	$\bot$	$\bot$
b	$\bot$	$\bot$	$\bot$
c	$\bot$	$\bot$	$\bot$

where there is absolute no knowledge and so any trust value could be anything, hence $\bot$. Each principal then inspects its own policy, and computes an approximated value using the other principals' current approximations of their values.

	a	b	c
a	`high`	$\bot$	$\bot$
b	$*$	`high`	`low`
c	$\bot$	`high`	`high`

Observe that here a is still totally undecided concerning the trust to assign to c, as it knows that the value $\bot$ received from b is itself uncertain. The successive iteration is however enough to solve all solvable uncertainties, and reach the global trust function (that is the last fixpoint) illustrated below.

	a	b	c
a	high	$\bot$	low
b	$*$	high	low
c	$*$	high	high

Note that the operators used for specifying policies (basically only delegation and constants) are continuous. This guarantees the existence of the least fixedpoint.

We reiterate that, importantly, the ordering $\sqsubseteq$ is not to be identified with the equally essential ordering "more trust."

Example 4.2 (Collecting Observations). In order to exemplify the idea of a trust structure with two distinct orders, assume that each observation can be classified either to be *positive* or *negative*. Then let $\mathcal{T}$ be the set of pairs $\{(p,n)|p,n \in \omega+\{\infty\},\ p \leq n\}$ where p stands for the number of positive experience and n for the total number of experiences. A suitable information ordering here would be

$$(p_1,n_1) \sqsubseteq (p_2,n_2) \text{ iff } n_1 \leq n_2,$$

with $(0,0)$ being the least element. This formalises the idea that the more experience we make, the more knowledge we have. That is, trust information becomes more and more precise with interactions. On the other hand, it is intuitively clear that negative experiences cannot lead to higher trust. Therefore, a suitable trust ordering could be

$$(p_1,n_1) \preceq (p_2,n_2) \text{ iff } p_1 \leq p_2.$$

4.2 Trust Structures

Having pointed out the need for order structures equipped at the same time with an information and a trust ordering, in this section we focus on the triples $(\mathcal{T},\preceq,\sqsubseteq)$, which we call *trust structures*, and study their basic properties.

When defining a trust management system, it is natural to start off with a set D of trust values, or degrees. On top of that, we are likely to need ways to compare and combine elements of D so as to form, say, a degree which comprehends a given set of trust values, or represents the trust level common to several principals. This amounts to start with a complete lattice $(D,\leq)$, where those combinators can be considered as taking lubs or glbs of sets of values. To account for uncertainty, we define an operator I to extend a lattice $(D,\leq)$ to a trust structure $(\mathcal{T},\preceq,\sqsubseteq)$. The set $\mathcal{T}$ consists of the set of intervals over D which, besides containing a precise image of D – viz. the singletons – represent naturally the notion of approximation, or uncertainty about elements of D.

4.2.1 Interval Construction

We define now the ordering $\preceq$ which has been already considered in [73].

Definition 4.1. Given a complete lattice $(D,\leq)$ and $X,Y \subseteq D$ nonempty subsets we say that $X \preceq Y$ if and only if

$$\sqcap X \leq \sqcap Y \qquad \text{and} \qquad \sqcup X \leq \sqcup Y$$

Clearly, $\preceq$ is not a partial order on the subsets of D, as the antisymmetry law fails. We get a partial order by considering as usual the equivalence classes of $\sim = \preceq \cap \succeq$. It turns out that the intervals over D are a set of representatives of such classes.

Definition 4.2. For $(D, \leq)$ a complete lattice, the set $I(D) = \{[d_0, d_1] \mid d_0, d_1 \in D,\ d_0 \leq d_1\}$, where $[d_0, d_1] = \{d \mid d_0 \leq d \leq d_1\}$ is the *interval* of D determined by d_0 and d_1.

Proposition 4.1. *Let $X = [d_0, d_1]$ be an interval in D. Then, $\sqcap X$ is d_0 and $\sqcup X$ is d_1.*

Proof. It comes clearly from the definition of interval as for all $d \in D$ we have that $d_0 \leq d \leq d_1$. □

As a consequence of the proposition above we have that $X \sim [\wedge X, \vee X]$, for all $X \subseteq D$. Furthermore, $[d_0, d_1] \sim [d_0', d_1']$ implies that $d_0 = d_0'$ and $d_1 = d_1'$. The following lemma characterises $\preceq$ in terms of $\leq$.

Lemma 4.1. *For $[d_0, d_1]$ and $[d_0', d_1']$ intervals of D, we have $[d_0, d_1] \preceq [d_0', d_1']$ if and only if $d_0 \leq d_0'$ and $d_1 \leq d_1'$.*

Proof. $\Rightarrow$) If $[d_0, d_1] \preceq [d_0', d_1']$ then by definition of $\preceq$ and proposition 4.1 we have that $d_0 \leq d_0'$ and $d_1 \leq d_1'$.
$\Leftarrow$) If $d_0 \leq d_0'$ and $d_1 \leq d_1'$ then, again by definition of $\preceq$ and proposition 4.1, it holds that $[d_0, d_1] \preceq [d_0', d_1']$ □

We can now show that the lattice structure on $(D, \leq)$ is lifted to a lattice structure $(I(D), \preceq)$ on intervals.

Theorem 4.1. *$(I(D), \preceq)$ is a complete lattice.*

Proof. Let S be a subset of $I(D)$. We prove that its least upper bound exists. Observe that this is enough to conclude, as $\bigwedge X$ is equal to $\bigvee \{y \mid y \preceq x \text{ for all } x \in X\ \}$. Let S be $\{[d_0^i, d_1^i] \mid i \in J\}$ for some set J. We claim that $\bigvee S = [\vee d_0^i, \vee d_1^i]$.

As d_0^i and d_1^i are elements of a complete lattice $\vee d_0^i$ and $\vee d_1^i$ exist. Moreover for each j we have $d_0^j \leq d_1^j \leq \vee d_1^i$, so $\vee d_0^i \leq \vee d_1^i$ which implies $[\vee d_0^i, \vee d_1^i] \in I(D)$. Also, $[\vee d_0^i, \vee d_1^i]$ is an upper bound as for each j we have $[d_0^j, d_1^j] \preceq [\vee d_0^i, \vee d_1^i]$. Finally, $[\vee d_0^i, \vee d_1^i]$ is the least upper bound: if for each j we have that $[d_0^j, d_1^j] \preceq [d_0, d_1]$ then $[\vee d_0^i, \vee d_1^i] \preceq [d_0, d_1]$. This holds since $d_0^j \leq d_0$ and $d_1^j \leq d_1$ which means that $\vee d_0^i \leq d_0$ and $\vee d_1^i \leq d_1$. □

We now define an ordering on intervals which reflects their information contents. Such an ordering will be a CPO on which we base fixpoint computations. The task is quite easy: as the interval $[d_0, d_1]$ expresses a value between d_0 and d_1, the narrower the interval, the lesser the uncertainty. This leads directly to the following definition.

Definition 4.3. For $(D, \leq)$ a complete lattice and $X, Y \in I(D)$, define $X \sqsubseteq Y$ if $Y \subseteq X$.

Analogously to $\preceq$, we can characterise $\sqsubseteq$ in terms of $\leq$.

Lemma 4.2. *For $[d_0, d_1]$ and $[d_0', d_1']$ intervals of D, we have that $[d_0, d_1] \sqsubseteq [d_0', d_1']$ if and only if $d_0 \leq d_0'$ and $d_1' \leq d_1$.*

Proof. By definition $[d_0', d_1'] \subseteq [d_0, d_1]$. As they are intervals d_0 and d_1 are elements of $[d_0', d_1']$ and so we have $d_0 \leq d_0'$ and $d_1 \leq d_1'$. Conversely, if $d_0 \leq d_0'$ and $d_1' \leq d_1$, then it is clear that $[d_0', d_1'] \subseteq [d_0, d_1]$, and by definition of $\sqsubseteq$ we have $[d_0', d_1'] \sqsubseteq [d_0, d_1]$. □

Finally, as for the previous ordering, we have the following result.

Theorem 4.2. *$(I(D), \sqsubseteq)$ is a CPO.*

Proof. The least element of $(I(D), \sqsubseteq)$ is $D = [\wedge D, \vee D]$. Let $[d_0^n, d_1^n]_n$ be an ω-chain in $(I(D), \sqsubseteq)$. Then we claim that $\bigsqcup [d_0^n, d_1^n]_n = [\vee d_0^n, \wedge d_1^n]$. We need to prove that this is well-defined and that it is the least upper bound.

As in the proof of Theorem 4.1, $\vee d_0^i$ and $\wedge d_1^i$ exist. Moreover, if for all i and j it holds that $d_0^i \leq d_1^j$, then for all j we have that $\vee d_0^i \leq d_1^j$, hence $\vee d_0^i \leq \wedge d_1^i$, which implies that $[\vee d_0^n, \wedge d_1^n]$ is well defined. Interval $[\vee d_0^n, \wedge d_1^n]$ is an upper bound as for all j it holds that $[d_0^j, d_1^j] \sqsubseteq [\vee d_0^i, \wedge d_1^i]$. In fact for all j we have $d_0^j \leq \vee d_0^i$ and $\wedge d_1^i \leq d_1^j$, which means $[\vee d_0^i, \wedge d_1^i] \sqsubseteq [d_0^j, d_1^j]$. Finally, $[\vee d_0^n, \wedge d_1^n]$ is the least upper bound as for any interval $[d_0, d_1]$ if $[d_0^j, d_1^j] \sqsubseteq [d_0, d_1]$ for all j then $[\vee d_0^i, \wedge d_1^i] \sqsubseteq [d_0, d_1]$. In fact $d_0^j \leq d_0$ and $d_1 \leq d_1^j$. By definition of lub and glb we have that $\vee d_0^i \leq d_0$ and $d_1 \leq \wedge d_1^i$. □

The trust structures above give a method to model trust based systems. We remark that intervals are a natural way to express partial information: trust in a principal is $[d_0, d_1]$ when it could be any value between d_0 and d_1.

Example 4.3 (Intervals in [0,1]). Let R stand for the set of reals between 0 and 1, which is a complete lattice with the usual ordering $\leq$, and let us consider the set $I(R)$ of intervals in R. It follows from the previous results that $(I(R), \preceq)$ is a complete lattice and $(I(R), \sqsubseteq)$ is a complete partial order. The trust domain so obtained is particularly interesting, as it allows us to express complex policies. In particular, it is related to the uncertainty logic [71], where an interval $[d_0, d_1]$ in $I(R)$ is seen as a pair of numbers where d_0 is called belief and $1 - d_1$ disbelief. Although a formal comparison with Jøsang's logic is beyond the scope of our presentation, in the following we shall rework a few simple examples from [71] in the present framework.

An important property of $(I(D), \preceq, \sqsubseteq)$ is stated below.

Theorem 4.3. *Relation $\preceq$ is continuous with respect to $\sqsubseteq$ and, conversely, relation $\sqsubseteq$ is continuous with respect to $\preceq$.*

Proof. According to definition of continuity for an order respect to another, we have to show, in the first case, that for all ω-chain $\mathsf{c} = (c_n)_{n \in \omega}$ wrt the ordering $\preceq$ and for all intervals $[d_0, d_1]$, we have that for all i, $c_i \sqsubseteq [d_0, d_1]$ implies $\bigsqcup_{\preceq} c_i \sqsubseteq [d_0, d_1]$. Let $c_i = [d_0^{c_i}, d_1^{c_i}]$ and $\bigsqcup_{\preceq} c_i = [d_0^c, d_1^c]$. For all i, $c_i \sqsubseteq [d_0, d_1]$ implies that $[d_0^{c_i}, d_1^{c_i}] \sqsubseteq [d_0, d_1]$, hence $d_0^{c_i} \leq d_0$ and $d_1 \leq d_1^{c_i}$. Moreover, by definition of lub, we have that $[d_0^{c_i}, d_1^{c_i}] \preceq [d_0^c, d_1^c]$, hence $d_0^{c_i} \leq d_0^c$ and $d_1^{c_i} \leq d_1^c$. Now, from Theorem 4.1, we have that $d_0^c = \sqcup d_0^{c_i}$ and then, by definition of lub, $d_0^c \leq d_0$ as for all i, $d_0^{c_i} \leq d_0$. We have proved that $d_0^c \leq d_0$ and $d_1 \leq d_1^c$, i.e. $\bigsqcup_{\preceq} c_i \sqsubseteq [d_0, d_1]$.

Proving that $\sqsubseteq$ is continuous with respect to $\preceq$ proceeds similarly to the proof above.

□

4.2.2 Lifting Operators

The continuity of the function Π is an important requirement. This property depends on the operators used with the policies. In the sequel we give a useful result, with respect to our interval construction, which allows us to lift continuous operators in the original lattice $(D, \leq)$ to continuous operators in $(I(D), \sqsubseteq)$ and $(I(D), \preceq)$.

Definition 4.4. For $(D, \leq)$ and $(D', \leq')$ complete lattices and $f : D \longrightarrow D'$ a continuous function, let $I(f) : I(D) \longrightarrow I(D')$ be the *pointwise extension* of f defined as

$$I(f)([d_0, d_1]) = [f(d_0), f(d_1)].$$

Note that in this definition the continuity of f ensures that $I(f)$ is well defined.
The following proposition states that all ω-cochains in $(I(D), \sqsubseteq)$ have glbs.

Proposition 4.2. *Let $[d_0^n, d_1^n]$ be an ω-cochain in $(I(D), \sqsubseteq)$. Then $\sqcap[d_0^n, d_1^n] = [\wedge d_0^n, \vee d_1^n]$.*

Proof. Symmetric to that of Theorem 4.2. □

We can now give the following result about lifted functions in trust structures.

Theorem 4.4. *For $(D, \leq)$ and $(D', \leq')$ complete lattices and $f : D \longrightarrow D'$ a bi-continuous function, the pointwise extension $I(f)$ is bi-continuous with respect to both the information and the trust orderings.*

Proof. We show that $I(f)$ is bi-continuous with respect to the information ordering. First we prove that $I(f)$ is continuous, i.e. $\bigsqcup I(f)([d_0^n, d_1^n]) = I(f)(\bigsqcup[d_0^n, d_1^n])$ for $[d_0^n, d_1^n]$ an ω-chain. By definition of $I(f)$ we have that $\bigsqcup I(f)([d_0^n, d_1^n]) = \bigsqcup[f(d_0^n), f(d_1^n)]$ and from the proof of Theorem 4.2 and bi-continuity of f it follows that

$$\bigsqcup[f(d_0^n), f(d_1^n)] = [\vee f(d_0^n), \wedge f(d_1^n)] = [f(\vee d_0^n), f(\wedge d_1^n)] = I(f)(\bigsqcup[d_0^n, d_1^n]).$$

The proof that $I(f)$ is co-continuous follows the same patterns. Let $[d_0^n, d_1^n]$ be an ω-cochain. Then, $\bigsqcap I(f)([d_0^n, d_1^n]) = \bigsqcap[f(d_0^n), f(d_1^n)]$ and from the proposition above and by the bi-continuity of f it follows that

$$\bigsqcap[f(d_0^n), f(d_1^n)] = [\wedge f(d_0^n), \vee f(d_1^n)] = [f(\wedge d_0^n), f(\vee d_1^n)] = I(f)(\bigsqcap[d_0^n, d_1^n]).$$

The proof for the trust orderings proceeds similarly. □

In the following examples we show how to apply the previous theorem to some interesting operators.

Example 4.4 (Lub and glb operators). The most natural operators, regarding lattices, are lub and glb. It is easy to see that they are bi-continuous in a complete lattice $(D, \leq)$. Exploiting Theorem 4.4 we can now state that lub and glb with respect to $\preceq$ are bi-continuous over $(I(D), \sqsubseteq)$.

Example 4.5 (Multiplication and Sum). When considering the interval construction over R, as in Example 4.3, we can extend the operators of sum (weighted) and multiplication over the intervals. In fact, given two intervals $[d_0, d_1]$ and $[d_0', d_1']$, the product is defined as

$$[d_0, d_1] \cdot [d_0', d_1'] = [d_0 \cdot d_0', d_1 \cdot d_1'],$$

which is exactly the extension of multiplication over reals. Similarly we can define sum as

$$[d_0,d_1]+[d'_0,d'_1]=[d_0+d'_0-d_0\cdot d'_0,d_1+d'_1-d_1\cdot d'_1].$$

These operations appear in [71] under the names of conjunction and disjunction.

Example 4.6 (A non-lifted operator: Discounting). Discounting, as defined in [71], is an operator which weighs the trust value received from a delegation according to the trust in the delegated principal.

$$[d_0,d_1]\rhd[d'_0,d'_1]=[d_0\cdot d'_0,1-d_0\cdot(1-d'_1)]$$

This operator could be useful when referencing to other principal's trust. The left component can be thought of as the weight, which says how much we should consider the value which appears on the right hand side of $\rhd$. This operator is, obviously not lifted from the starting lattice.

4.2.3 Product and Function Constructors

Our model should satisfy "context dependent" trust. By this we mean that trusting a principal a to obtain information about restaurants does not mean that we trust a about, say, sailing. We can accommodate this kind of situation using a simple property of lattices and CPO's. Namely, we can form products of trust structures where each component accounts for a particular context. For instance, using a domain of the form *Restaurants* $\times$ *Sailing* will allow us to distinguish about a's dependability on the two issues of our example. The next theorem shows that extending the orders pointwise to products and function spaces gives the result we need.

Theorem 4.5. *Given two complete lattices $(D,\leq)$, $(D',\leq')$ and a generic set X then*

1. *$I(D\times D')$ is isomorphic to $I(D)\times I(D')$;*
2. *$X\longrightarrow I(D)$ is isomorphic to $I(X\longrightarrow D)$.*

Proof. In both cases we have to show that there exists a bijective correspondence H which preserves and reflects the orderings. We use as usual (d,d) to denote pairs and $\lambda x.t(x)$ to express the function which takes a value d and returns $t(d)$.

1. Let $[(d_0,d'_0),(d_1,d'_1)]$ and $[(d_2,d'_2),(d_3,d'_3)]$ be in $I(D\times D')$. We define the function $H:I(D\times D')\longrightarrow I(D)\times I(D')$ by

   $$H([(d_0,d'_0),(d_1,d'_1)])=([d_0,d_1],[d'_0,d'_1])$$

   which is easily seen to be bijective. We first show that H is well defined, i.e. that $d_0\leq d_1$ and $d'_0\leq' d'_1$. This follows at once, since $(d_0,d'_0)\leq_{D\times D'}(d_1,d'_1)$ where $\leq_{D\times D'}$ is the pointwise extension of $\leq$ and $\leq'$. Next we prove that H preserves and reflects $\preceq$. This amounts to prove that

   $$[(d_0,d'_0),(d_1,d'_1)]\preceq_{I(D\times D')}[(d_2,d'_2),(d_3,d'_3)] \tag{4.1}$$

   if and only if

   $$([d_0,d_1],[d'_0,d'_1])\preceq_{I(D)\times I(D')}([d_2,d_3],[d'_2,d'_3]) \tag{4.2}$$

By Lemma 4.1, relation (4.1) above holds if and only if

$$(d_0, d_0') \leq_{D\times D'} (d_2, d_2') \quad \text{and} \quad (d_1, d_1') \leq_{D\times D'} (d_3, d_3'). \tag{4.3}$$

Then, by the same Lemma 4.1, property (4.3) holds if and only if

$$[d_0, d_1] \preceq_{I(D)} [d_2, d_3] \quad \text{and} \quad [d_0', d_1'] \preceq_{I(D')} [d_2', d_3'],$$

which is the same as saying that (4.2) holds.

The proof is similar for the information ordering.

2. Let $[f_0, f_1]$ and $[f_0', f_1']$ be in $I(X \longrightarrow D)$ and g in $X \longrightarrow I(D)$. We define the bijection $H : I(X \longrightarrow D) \cong (X \longrightarrow I(D))$ by $H([f_0, f_1]) = \lambda x.[f_0(x), f_1(x)]$. It is easy to see that $H([f_0, f_1])$ is well defined. The function $H^{-1}(g) = [\lambda x. \wedge g(x), \lambda x. \vee g(x)]$ is the inverse of H. In fact, $H^{-1}(H([f_0, f_1])) = H^{-1}(\lambda x.[f_0(x), f_1(x)])$ and by definition of H^{-1} we have that the latter coincides with

$$[\lambda y. \wedge (\lambda x.[f_0(x), f_1(x)])(y), \lambda y. \vee (\lambda x.[f_0(x), f_1(x)])(y)],$$

which is $[\lambda x. \wedge [f_0(x), f_1(x)], \lambda x. \vee [f_0(x), f_1(x)]]$, i.e. $[f_0, f_1]$. Conversely, we have $H(H^{-1}(g)) = H([\lambda x. \wedge g(x), \lambda x. \vee g(x)])$ and, by definition of H, this is the same as

$$\lambda y.[(\lambda x. \wedge g(x))(y), (\lambda x. \vee g(x))(y)] = \lambda x.[\wedge g(x), \vee g(x)] = g.$$

We now need to show that H and H^{-1} preserve $\sqsubseteq$. Regarding H we have to prove

$$[f_0, f_1] \sqsubseteq_{I(X\longrightarrow D)} [f_0', f_1'] \tag{4.4}$$

implies

$$\lambda x.[f_0(x), f_1(x)] \sqsubseteq_{X\longrightarrow I(D)} \lambda x.[f_0'(x), f_1'(x)] \tag{4.5}$$

From Lemma 4.2 and (4.4) it follows that $f_0 \sqsubseteq_{X\longrightarrow D} f_0'$ and $f_1' \sqsubseteq_{X\longrightarrow D} f_1$ and, as the ordering is pointwise, we have that for all x, $f_0(x) \leq f_0'(x)$ and $f_1'(x) \leq f_1(x)$. Again by Lemma 4.2, we obtain $[f_0(x), f_1(x)] \sqsubseteq [f_0'(x), f_1'(x)]$ which, by pointwise ordering, implies (4.5).

Regarding H^{-1}, we show that

$$g \sqsubseteq_{X\longrightarrow I(D)} g' \tag{4.6}$$

implies

$$[\lambda x. \wedge g(x), \lambda x. \vee g(x)] \sqsubseteq_{I(X\longrightarrow D)} [\lambda x. \wedge g'(x), \lambda x. \vee g'(x)] \tag{4.7}$$

From (4.6) we have that, for any x, $[\wedge g(x), \vee g(x)] \sqsubseteq [\wedge g'(x), \vee g'(x)]$. It then follows that $\wedge g(x) \leq \wedge g'(x)$ and $\vee g'(x) \leq \vee g(x)$ which implies $\lambda x. \wedge g(x) \leq \lambda x. \wedge g'(x)$ and $\lambda x. \vee g'(x) \leq \lambda x. \vee g(x)$. Finally, again by Lemma 4.2, we have that (4.7) holds.

The proof for the trust ordering is similar and, thus, omitted. □

Remark 4.1. Theorem 4.4 holds for any bi-continuous function $f : D_0 \times \ldots \times D_n \longrightarrow D$. The pointwise lifting of f gives a function $I(f) : I(D_0 \times \ldots \times D_n) \longrightarrow I(D)$ and from the result above we have that $I(f)$ is (isomorphic to) a function $F : I(D_0) \times \ldots \times I(D_n) \longrightarrow I(D)$.

4.3 A Language for Policies

Following our discussion we propose to operate with a language for trust policies capable of expressing intervals, delegation, and a set of function constructions. We exemplify the approach by studying the simple policy language below.

4.3.1 Syntax

The language consists of the following syntactic categories, parametric over a fixed trust lattice $(D, \leq)$.

$$
\begin{array}{rll}
\pi ::= & \ulcorner p \urcorner & \text{(delegation)} \\
| & \lambda x : \mathcal{P}.\ \tau & \text{(abstraction)} \\
\\
p ::= & a \in \mathcal{P} & \text{(principal)} \\
| & x : \mathcal{P} & \text{(vars)} \\
\\
\tau ::= & [d,d] \in I(D) & \text{(value/var)} \\
| & \pi(p) & \text{(policy value)} \\
| & e \mapsto \tau;\tau & \text{(choice)} \\
| & \mathtt{op}(\tau_1 \ldots \tau_n) & \text{(lattice op)} \\
\\
e ::= & p = p & \text{(equality)} \\
| & e\ \mathtt{bop}\ e & \text{(boolean op)}
\end{array}
$$

Here `op` is a continuous function over $(I(D), \sqsubseteq)$, and `bop` is a standard boolean operator. The elements of the category p are either principals or variables. The main syntactic category is π: it can be either delegation to another principal or a λ-abstraction. An element of τ can be an interval, the application of a policy, a conditional or the application of a continuous operator `op`. The elements of e are boolean functions applied to equalities between elements of $\mathcal{P}$.

It is worth noticing that such a simple language goes beyond delegation interpreted strictly. In fact, rather that allowing principals to merely delegate somebody to decide on their behalf, it allows them to consults with each other to form complex, informed trust judgements. The examples to follow will clarify this concept.

4.3.2 Denotational Semantics

We provide a formal semantics for the language described above. As pointed out before, π is a policy. Hence the semantic domain, as described in 4.1.1, will be the codomain of the function

$$[\![\pi]\!]_\sigma : (\mathcal{P} \longrightarrow \mathcal{P} \longrightarrow \mathcal{T}) \longrightarrow (\mathcal{P} \longrightarrow \mathcal{T}),$$

where σ is an assignment of values in $\mathcal{P}$ to variables. The semantic function $[\![\cdot]\!]_\sigma$ is defined by structural induction on the syntax of π as follows.

$$[\![\ulcorner p \urcorner]\!]_{\sigma m} = m(\llbracket p \rrbracket)_{\sigma m};$$
$$[\![\lambda x : P.\ \tau]\!]_{\sigma m} = \lambda p : \mathcal{P}.(\llbracket \tau \rrbracket)_{\sigma\{p/x\}m}.$$

Here $(\llbracket\cdot\rrbracket)_{\sigma m}$ is a(n overloaded) function which given an assignment σ and a global trust function $m : \mathcal{P} \longrightarrow \mathcal{P} \longrightarrow \mathcal{T}$ maps elements of p, τ, and e respectively to the semantic domains $\mathcal{P}$, $I(D)$, and Bool as follows.

$$(\!|\ [d_0, d_1]\ |\!)_{\sigma m} = [d_0, d_1]$$
$$\llbracket \pi(p) \rrbracket_{\sigma m} = [\![\pi]\!]_{\sigma m}(\llbracket p \rrbracket)_{\sigma m}$$
$$\llbracket e \mapsto \tau_1; \tau_2 \rrbracket_{\sigma m} = \mathsf{if}\ (\llbracket e \rrbracket)_{\sigma m}\ \mathsf{then}\ (\llbracket \tau_1 \rrbracket)_{\sigma m}\ \mathsf{else}\ (\llbracket \tau_2 \rrbracket)_{\sigma m}$$
$$\llbracket \mathtt{op}(\tau_1 \ldots \tau_n) \rrbracket_{\sigma m} = \mathsf{op}\,((\llbracket \tau_1 \rrbracket)_{\sigma m}, \ldots, (\llbracket \tau_n \rrbracket)_{\sigma m})$$
$$\llbracket a \rrbracket_{\sigma m} = a \qquad \llbracket x \rrbracket_{\sigma m} = \sigma(x)$$
$$\llbracket p_1 = p_2 \rrbracket_{\sigma m} = (\llbracket p_1 \rrbracket)_{\sigma m} = (\llbracket p_2 \rrbracket)_{\sigma m}$$
$$\llbracket e_1\ \mathtt{bop}\ e_2 \rrbracket_{\sigma m} = (\llbracket e_1 \rrbracket)_{\sigma m}\ \mathsf{bop}\ (\llbracket e_2 \rrbracket)_{\sigma m}$$

Let $\{\pi_p\}_{p\in\mathcal{P}}$ be a an arbitrary collection of all policies, where $\pi_p = \lambda x : \mathcal{P}.\ \bot$ for all but a finite number of principals. The fixpoint semantics of $\{\pi_p\}_{p\in\mathcal{P}}$ is the global trust function determined by the collection of individual policies, and it is readily expressed in terms of $[\![\cdot]\!]_\sigma$:

$$\{\!|\ \{\pi_p\}_{p\in\mathcal{P}}\ |\!\}_\sigma = \mathsf{fix}(\lambda m.\lambda p.[\![\pi_p]\!]_{\sigma m}).$$

We believe that this policy language is sufficiently expressive for most application scenarios in GC, as supported by the following examples. Note however that our approach generalises to any choice of underlying trust structure $(\mathcal{T}, \preceq, \sqsubseteq)$, provided the operators used in the policy language are continuous with respect to the information ordering.

Example 4.7 (Read and Write access). Let $D = \{\mathtt{N}, \mathtt{W}, \mathtt{R}, \mathtt{RW}\}$ represent the access rights to principal's CVs. The set D is ordered by the relation $\leq$

$$\forall d \in D.\mathtt{N} \leq d \qquad \text{and} \qquad \forall d \in D.d \leq \mathtt{RW}.$$

Let us consider how to express some simple policies in our language. The following policy says that `LIZ`'s trust in `BOB` is at least `[W,RW]` and depends on what she thinks of `CARL`. Instead, `LIZ`'s trust in `CARL` will depend on her trust in `BOB`: if it is above `[W,W]` then `[R,RW]` otherwise `[N,RW]`.

$$\begin{aligned}\pi_{\mathtt{LIZ}} = \lambda x : \mathcal{P}.\ & \\ & x = \mathtt{BOB} \mapsto \mathtt{[W,RW]} \vee \ulcorner \mathtt{LIZ} \urcorner(\mathtt{CARL});\\ & x = \mathtt{CARL} \mapsto \\ & \quad (\mathtt{[W,W]} \preceq \ulcorner \mathtt{LIZ} \urcorner(\mathtt{BOB}) \mapsto \mathtt{[R,RW]}; \mathtt{[N,RW]});\\ & \mathtt{[N,RW]}\end{aligned}$$

This policy can be made dependent on someone else's belief. For instance, the above judgement about BOB is merged below with PAUL's belief (weighed by discounting).

$$\begin{aligned}\pi_{\texttt{LIZ}} = \lambda x : \mathcal{P}.\ & \\ & x = \texttt{BOB} \mapsto \texttt{[N,W]} \vee \ulcorner \texttt{LIZ} \urcorner(\texttt{CARL}) \\ & \qquad \vee \ulcorner \texttt{LIZ} \urcorner(\texttt{PAUL}) \rhd \ulcorner \texttt{PAUL} \urcorner(x); \\ & x = \texttt{CARL} \mapsto \\ & \qquad (\texttt{[W,W]} \preceq \ulcorner \texttt{LIZ} \urcorner(\texttt{BOB}) \mapsto \texttt{[R,RW]}; \texttt{[N,RW]}); \\ & \texttt{[N,RW]}\end{aligned}$$

In this case LIZ's trust in PAUL is the bottom value [N,RW] which is going to be the left argument of the discounting operator $\rhd$.

Example 4.8 (Spam Filter). Let R be as in Example 4.3. We illustrate some policies modelling filters for blocking spam emails. The set of principals $\mathcal{P}$ is the set of Internet domains from which we could receive emails, e.g. daimi.au.dk. A starting policy, where we suppose that our server spam.filter.edu knows no one, could be

$$\pi_1 = \lambda x : \mathcal{P}.\ x = \texttt{spam.filter.edu} \mapsto [1,1]; [0,1],$$

meaning that only internal emails are trusted. It could happen that spam.filter.edu starts interacting with other principals. A likely event is that it receives a list of other universities' Internet domains, and decides to trust them to a large extent, and actually use their beliefs. We could have

$$\pi_2 = \lambda x : \mathcal{P}.\ x \in \mathit{UniList} \mapsto [.75,1];\ \bigvee_{y \in \mathit{UniList}} \ulcorner y \urcorner(x) \vee \pi_1(x),$$

where we suppose that "$\in$" stands for a chain of nested conditionals for all the elements of *UniList*. Let us suppose now that the filter receives emails from a certain number of suspicious addresses, and would like to single them out and enforce a special treatment for them. The policy could be updated as

$$\pi_3 = \lambda x : \mathcal{P}.\ x \in \mathit{BadList} \mapsto [0,.5];\ \pi_2(x).$$

The spam-filter could then decide to add a new level of badness and create the new list *VeryBadList*. At the same time, it would like to change the policy for *BadList* putting certain restrictions on the intervals returned as other universities' opinions.

$$\begin{aligned}\pi_4 = \lambda x : \mathcal{P}.\ & \\ & x \in \mathit{VeryBadList} \mapsto [0,.2]; \\ & x \in \mathit{BadList} \mapsto \pi_2(x) \wedge [0,.5]; \\ & \pi_2(x).\end{aligned}$$

As illustrated in the *Spam Filter* example, we see trust evolution as being modelled by suitable updates of policies, as response to, e.g., observations of principal behaviour. However, it is still not clear exactly what update primitives are required in practice. We are currently working on developing a calculus of of trust and principal behaviour, with features for trust policy updates We will return on this in the concluding section.

Example 4.9 (Reputation Based Systems). The work [108] presents a reputation-based model of trust, where each principal a has an associated history H_a of observations, or *events*. A history $(e_1,\ldots,e_n)$ indicates that event e_i has happened after events $e_1,\ldots,e_{i-1}$, for all i. A principal can provide information to the others (a.k.a. 'recommending') based on its past history. This means that it is not trust being propagated between principals, but observations. Reputation is then defined to be (as a formula satisfied) when a principal has never been observed to ignore certain conditions, i.e., if it never misused a resource.

Our approach is flexible enough to express some of this. (A full treatment requires the integration of policy updates in the policy language.) The idea is to make history part of a policy π, so that a principal's trust decision process can be defined in terms of its own and other principals' past observations. Let us consider the example of a peer-to-peer file distribution system discussed in [108]. In such scenario, users are allowed to download provided that they allows at least one upload every three downloads. Let $\mathbb{B}$ be the set of ordered boolean values, with $\mathsf{ff} \leq \mathtt{tt}$, and let $\mathbb{N}_\bullet$ be the set of natural numbers completed with a top element ∞. Histories are elements of $\mathbb{H} = \mathcal{P} \to \mathbb{N}_\bullet \times \mathbb{N}_\bullet$, i.e. functions which assign to principals the numbers of uploads and downloads they performed in the past. Then, a's trust function π_a is of the kind

$$\mathcal{P} \to \mathbb{H} \to \mathbb{B}$$

where we understand that a after a history h trusts x to download if $\pi_a(x)(h)$ yields $\mathtt{tt}$. The SERVer's policy can be written as follows in a suitable "sugared" version of our language:

$$\pi_{\mathtt{SERV}} = \lambda p : \mathcal{P}.\lambda h : \mathbb{H}.\ \mathsf{let}\ (u,d) = h(p)\ \mathsf{in}\ d \leq 3u.$$

If access is granted, h is updated in view of the next invocation by increasing p's count of downloads and, correspondingly, its peer's count of uploads.

Chapter 5

A Calculus for Trust Management

Following up the discussion of the previous chapter, we introduce a process calculus for modelling trust management systems (ctm). We saw that already many models for trust based system have appeared in the literature, and most of them feature some sort of logic to describe trust policies. However, lacking a notion of protocol, such approaches typically fall short to describe the exact behaviour of systems, which is a fundamental property when security concerns are present. Consider for instance a server whose policy is to grant access only to a certain class of users, but whose (flawed) protocol of communication always allows access to a particular resource. Even though the policy may be correct, the whole system is not. A second aspect of paramount importance here is to allow principals' interactions to feedback to the security mechanisms and influence future policies. For instance, access rights can change for a principal in response to its behaviour, and how precisely this behaviour should be observed and the variation should take place ought to be part of the model.

The aim of this chapter is to develop a coherent framework centred on these two aspects, and establish its basic theory. In ctm, a principal is specified by a pair, a policy α and a protocol P, which interact in precise ways, as described below. The policy α informs the protocol P as to what actions are allowed at any given time, and works on the basis of evidence and observations collected from past interactions. Dually, P interacts with a network of other principals, and in doing so it produces the observations gathered in α. The protocol P will consult α when making a decision, e.g. whether or not to grant a specific service to a specific principal. Schematically, we can represent the situation as in the informal picture below.

$$(\texttt{Policy} \iff \texttt{Protocol}) \parallel \texttt{Network}$$

We model the "policy" side of the drawing with a decidable logic. The choice is a Datalog-like logic: a principal's policy will be represented as a set of formulas depending on a set of past observations. On the "protocol" side, our model is based on a process calculus in the style of the π-calculus [84]. More precisely, ctm is a calculus with locations linked by channel names. Each location (uniquely) identifies a principal, and the diagram above would be represented as $a\{\, P \,\}_\alpha \mid N$, where a is a principal with protocol P and policy α, in parallel with the rest of the network N. In ctm we associate the action of sending a message to another principal as granting a particular resource (viz. the service represented by the channel name). Outputs will then be guarded by

a formula ϕ from the logic, as for instance $\phi :: b \cdot \ell\langle\tilde{m}\rangle$, which according to the truth value of ϕ allows the protocol to send $\tilde{m}$ to b on channel (or method) ℓ. As a concrete example, a protocol like $Access(b,R) :: b \cdot l\langle n\rangle$ would stand for "if my policy grants b '*Access*' to R, then send n along l to b." Symmetrically, inputs represent requests of services, and form the observable basis mentioned above. For instance, if executing an input action $b \cdot \texttt{print}(y) \,.\, P$, we receive a message '`junk`' for y from b, we observe and record that b has attempted to print a corrupted file. As mentioned above, multiple channels at a single location allows to distinguish among different services provided by a principal. We assume in `ctm` that the identity of communicating principals cannot be corrupted (i.e. we assume implicitly that authenticity etc. is guaranteed by lower level protocols).

In order to allow principals to offer services to generic (as opposed to named) clients, `ctm` features a new form of input capability, which allows to abstract from the communicating principal. For instance, in order to offer a printing service for all, we would write $x \cdot \texttt{print}(y) \,.\, P$, where x is a variable, which at the time of interaction will be bound to the name of the principal requesting the service. We call this operation *global input*.

The calculus `ctm` seems a powerful tool for expressing several examples of trust-based systems for GC. Casting these ingredients in the world of process algebras provides many interesting theoretical notions, and in particular behavioural equivalences. The natural separation of principals in pairs of components, viz. policies and protocols, induces new notions of equivalences. In particular, besides studying equivalences of principals and networks, one can focus on protocols as well as policies. Technically, the main contribution of this chapter is to introduce a theory of observational equivalence for trust-based systems, which captures in a single and homogeneous framework equivalences for protocols, policies, principals and networks.

5.1 The Calculus

Let Val be a denumerable set of values ranged over by l, m and partitioned into sets $\mathcal{P}$ and N, respectively the set of principals (ranged over by a, b, c) and the set of names (ranged over by n). Moreover Var (ranged over by x, y, z) is a set of variables such that $\mathsf{Var} \cap \mathsf{Val} = \emptyset$. In the sequel we assume u, v in $\mathsf{Var} \cup \mathsf{Val}$ and p in $\mathsf{Var} \cup \mathcal{P}$. The letter Φ denotes the set of predicats ranged over by letter ϕ. As usual a tilde over a letter indicates the extension to vectors.

5.1.1 Abstract Policies

As explained above, each principal acts on a body of knowledge built on the past interactions with other principals. In this subsection, we take care of how such a knoledge is kept, i.e. with the use of policies. In order to do so, we give a very general definition which is somehow capable to embed numerous models that can be found in literature. The following defintion introduces the notion of policy frame.

Definition 5.1 (Policy Frame). A policy frame over Val is a triple $(\Lambda, \mathtt{upd}, \mathtt{fn})$ where

- Λ is the set of abstract policies, (decidable) functions from Φ to $\{tt, ff\}$ denoted by letters α, β;
- upd is the update, a (decidable) function of type $\Lambda \times \mathsf{Msg} \to \Lambda$;
- $\mathtt{fn} : \Lambda \to \mathcal{N}$ is the free names function.

We write $\alpha \vdash \phi$ whenever $\alpha(\phi) = tt$.

Policies are functions which given an element of Φ (to be seen as a predicate) return a truth value. The function upd is meant for the dynamic feature of principals. Each principal interacts with others, and always acquires new information so that the policy must be updated. The function $\mathtt{fn}$ may be thought as a more *ad-hoc* construct which refers to the protocol part of a principal and is used to deal with free names. Nevertheless a notion of free names can be surely seen as part of the likelihood of the knowledge a policy has.

A policy frame provides a very general setting where to define policies. In a possible application this approach would be too abstract, and as consequence there is the need of a language for writing down policies, e.g. the policy language seen in the previous chapter. One possible alternative to that policy language can be the work done in [72] where they give a declarative language, based on a pure-past variant of linear temporal logic, for writing interaction policies. A typical problem for this kind of models is how to efficiently re-evaluate a policy when new information is available. To this purpose, they also give two efficient algorithms which provide an interesting time/space trade-off.

Interaction Datatypes

We now introduce our suggestion for instantiating policy frames, i.e. using interaction datatypes, that can also be found in [34].

Messages in our calculus have form $a \cdot \tilde{l} \rhd \tilde{m}$ representing a message $\tilde{m}$ from principal a on channel $\tilde{l}$.

Definition 5.2 (Interaction Datatype). An interaction datatype $\mathcal{M}$ over Val is a triple $(\mathcal{S}, \mathcal{R}, \mathsf{upd})$ where $\mathcal{S}$ is a generic set of so-called interaction values, $\mathcal{R}$ is a set of decidable subset of $\mathcal{S} \times \mathsf{Val}^{\mathsf{k}}$, and upd is a function which given $s \in \mathcal{S}$ and a message $a \cdot \tilde{l} \rhd \tilde{m}$ returns an element of $\mathcal{S}$.

According to the above definition, the set $\mathcal{S}$ is a generic set: the idea is to build elements of $\mathcal{S}$ as representation of abstract information about past interactions with other principals. The set $\mathcal{R}$ defines the basic predicates binding together interaction values and elements of Val, and upd defines the effect in $\mathcal{S}$ by receiving a message.

Example 5.1 (Lists and Multisets). Let $\mathcal{S}$ be the set of lists with elements $a \cdot \tilde{l} \rhd \tilde{m}$, i.e. $\mathcal{S} = \{[a_1 \cdot \tilde{l}_1 \rhd \tilde{m}_1, \ldots, a_k \cdot \tilde{l}_k \rhd \tilde{m}_k] \mid k \geq 0\}$ and upd the operation of list concatenation. The set $\mathcal{R}$ could contain the relation $\mathsf{last}_{\tilde{m}}$ which holds true of lists whose last element carries the message $\tilde{m}$, and the relation $\mathsf{from}{\geq}5(a)$, satisfied whenever the number of messages in the list from a is larger than 5. Another interesting example is when $\mathcal{S}$ is the set of multisets over elements $a \cdot \tilde{l} \rhd \tilde{m}$ with multiset union as upd. Predicates can express the number of message occurrences, e.g. predicate $x \cdot - \rhd y < k$ is satisfied by

all elements of $\mathcal{S}$ such that the number of occurrences of elements $x \cdot z \rhd y$ is less than k.

Principals use policies to make decisions based on the information contained in an element $s \in \mathcal{S}$ of a given interaction datatype $\mathcal{M}$.

Definition 5.3 (Policy). Let $\mathcal{M} = (\mathcal{S}, \mathcal{R}, \mathsf{upd})$ be an interaction datatype, let $\mathcal{P}$ and $\mathcal{P}_{\mathcal{M}}$ be disjoint signatures of predicates symbols, with $\mathcal{P}_{\mathcal{M}}$ in one-to-one correspondence with $\mathcal{R}$. A policy π is defined as a set of rules of type $L(\tilde{u}) \leftarrow L_1(\tilde{u}_1), \ldots L_k(\tilde{u}_k)$ such that $L \in \mathcal{P}$ and $L_i \in \mathcal{P} \cup \mathcal{P}_{\mathcal{M}}$.

π is interpreted as a Datalog program [5] relative to an element $s \in \mathcal{S}$. More precisely, each rule in π is interpreted as Datalog implication, where predicate symbols in $\mathcal{P}_{\mathcal{R}}$ take as an implicit first argument the interaction value s. Given a pair (π, s) and a predicate $A(\tilde{l})$ we write $(\pi, s) \vdash A(\tilde{l})$ meaning that $A(\tilde{l})$ is entailed by the Datalog program π relative to s.

Notice that what shown in this example is just an instance of a policy frame. The function upd is just the update function of a policy frame, whereas the set of abstract policies Λ needs slightly more consideration. Pairs (π, s) can be seen as abstract policies α, but, obviously the function $\vdash$ (which is really the application of α to an element of Φ) is slightly more restricted. Concerning the notion of free names, we just translate it to the normal logic free names notion, e.g. the one in FOL. In the rest of this chapter we will always use interaction datatypes for the examples, as they provide a simple and concise language for describing policies.

5.1.2 Syntax

Let $(\Lambda, \mathsf{upd}, \mathtt{fn})$ be a policy frame over Val. The syntax of `ctm` is then featured by two main syntactic categories: *networks* (N) and *protocols* (P, Q).

$$
\begin{array}{rlr}
N, M ::= & \varepsilon & (\textsc{empty}) \\
 & \mid N \mid N & (\textsc{net-par}) \\
 & \mid a\{\ P\ \}_{\alpha} & (\textsc{principal}) \\
 & \mid (\nu n)\ N & (\textsc{new-net}) \\
 & & \\
Z ::= & p \cdot \tilde{u}(\tilde{x})\,.\,P & (\textsc{input}) \\
 & \mid \phi :: p \cdot \tilde{u}\langle \tilde{v} \rangle\,.\,P & (\textsc{output}) \\
 & \mid Z + Z & (\textsc{sum})
\end{array}
\qquad
\begin{array}{rlr}
P, Q ::= & 0 & (\textsc{null}) \\
 & \mid Z & (\textsc{sub}) \\
 & \mid P \mid P & (\textsc{par}) \\
 & \mid (\nu n)\ P & (\textsc{new}) \\
 & \mid\ !P & (\textsc{bang})
\end{array}
$$

A network N is composed of principals running in parallel. Each principal is equipped with a protocol P and a policy α. A network N is said to be *consistent* whenever principals names are unique, i.e. for each $a \in \mathcal{P}$ there is at most one subterm of N of the kind $a\{\ P\ \}_{\alpha}$ for some P and α. From now on we assume to work only with consistent networks.

A protocol P is given in the style of π-calculus [84]. The protocol 0 represents the inactive process. Terms (input) and (output) represent the main actions, and both can

be part of the standard sum operator (guarded choice). As remarked in the introduction, the input capability can either refer to a specific principal, or be global. The output action sends a message on a channel and is guarded by a predicate ϕ in the signature $\mathcal{P}$. For generality, we allow composite channel names as in [31]. The (bang) and (par) operators are standard. As for the π-calculus, we shall omit trailing inactive processes.

The functions for free names `fn` (resp. bound names `bn`) and free variables *fv* (resp. bound variables *bv*) are defined as usual on networks and protocols. Closed and open terms are defined as usual (with respect to variables). We recall from Chapter 2 that the symbol σ denotes a substitution from variables to names. Applying a substitution σ to a network N (or a protocol P) will be denoted by $N\sigma$ ($P\sigma$). The global input variable is a strong binder, e.g. in $x \cdot l(y).x \cdot l(y)$ the first x binds the second, instead the first y does not bind the second y. We omit trivial guards from outputs, i.e. $\mathtt{tt} :: b \cdot \tilde{l}\langle\tilde{m}\rangle$ will be written as $b \cdot \tilde{l}\langle\tilde{m}\rangle$ where `tt` denotes the "always" true predicate.

5.1.3 Reduction Semantics

In this section we give the formal semantics of the calculus in terms of reduction semantics. The structural congruence relation $\equiv$ is the least congruence relation on N such that $|$ and $+$ are commutative monoids on protocols, $|$ is a commutative monoid on networks, and such that it satisfies alpha-conversion and the rules

$$
\begin{array}{ll}
(\mathit{Struct}_1) & a\{\ !P \mid Q\ \}_\alpha \equiv a\{\ P \mid !P \mid Q\ \}_\alpha \\
(\mathit{Struct}_2) & (\nu n)\ (\nu n')\ W \equiv (\nu n')\ (\nu n)\ W \text{ for } W \in \{P,N\} \\
(\mathit{Struct}_3) & a\{\ (\nu n)\ P \mid Q\ \}_\alpha \equiv a\{\ (\nu n)\ (P \mid Q)\ \}_\alpha \quad \text{if } n \notin fn(Q) \\
(\mathit{Struct}_4) & (\nu n)\ N \mid M \equiv (\nu n)\ (N \mid M) \quad \text{if } n \notin fn(M) \\
(\mathit{Struct}_5) & a\{\ (\nu n)\ P\ \}_\alpha \equiv (\nu n)\ a\{\ P\ \}_\alpha \quad \text{if } n \notin \mathtt{fn}(\alpha)
\end{array}
$$

We define $\rightarrow$ as the least binary relation on N satisfying the rules given in Table 5.1. Rule (RCOM) defines communication between two principals. The operator $\odot$ is defined on tuples, as the most general unifier returning a substitution σ, whose application in the semantics is conditioned by successful unification. The rule (RINT) describes internal communication and is similar to (RCOM). Rules (RSTRUCT) and (RPAR) are standard. As usual we define $\Rightarrow$ as the reflexive and transitive closure of $\rightarrow$.

We can conclude this subsection with the following lemma which describes the structure of processes involved in reductions.

Lemma 5.1. *If $N \rightarrow M$ then one of the following results holds*

- *there are $\tilde{n}$, a, ϕ, b, $\tilde{l}$, $\tilde{m}$, P, P', P'', Q, Q', Q'', α, β and N' such that*
  - $N \equiv (\nu\tilde{n})\ (a\{\ \phi :: b \cdot \tilde{l}\langle\tilde{m}\rangle.P + P''\mid P'\ \}_\alpha \mid b\{\ p \cdot \tilde{l}(\tilde{x}).Q + Q'' \mid Q'\ \}_\beta \mid N')$
  - $M \equiv (\nu\tilde{n})\ (a\{\ P \mid P'\ \}_\alpha \mid b\{\ Q\sigma \mid Q'\ \}_{\beta'} \mid N')$
  - $\alpha \vdash \phi$
- *there are $\tilde{n}$, a, ϕ, $\tilde{l}$, $\tilde{m}$, P, P'', Q, Q', Q'', α and N' such that*

$$(\text{RCOM})\ \frac{R \triangleq p\cdot\tilde{l}(\tilde{x}).P \qquad S \triangleq \phi :: a\cdot\tilde{l}\langle\tilde{m}\rangle.Q \qquad \beta \vdash \phi \qquad \alpha' = \mathsf{upd}(\alpha, b\cdot\tilde{l}\rhd\tilde{m}) \qquad b : \tilde{m}\odot p : \tilde{x} = \sigma}{a\{\, R+P' \mid P'' \,\}_\alpha \mid b\{\, S+Q' \mid Q'' \,\}_\beta \to a\{\, P\sigma \mid P'' \,\}_{\alpha'} \mid b\{\, Q \mid Q'' \,\}_\beta}$$

$$(\text{RINT})\ \frac{\alpha \vdash \phi \qquad \alpha' = \mathsf{upd}(\alpha, a\cdot\tilde{l}\rhd\tilde{m}) \qquad a : \tilde{m}\odot p : \tilde{x} = \sigma}{a\{\, p\cdot\tilde{l}(\tilde{x}).P+P' \mid \phi :: a\cdot\tilde{l}\langle\tilde{m}\rangle.Q+Q' \mid Q'' \,\}_\alpha \to a\{\, P\sigma \mid Q \mid Q'' \,\}_{\alpha'}}$$

$$(\text{RRES})\ \frac{N \to N'}{(\nu n)\, N \to (\nu n)\, N'}$$

$$(\text{RSTRUCT})\ \frac{N \equiv N' \qquad N' \to M' \qquad M' \equiv M}{N \to M} \qquad (\text{RPAR})\ \frac{N \to M}{N \mid N' \to M \mid N'}$$

Table 5.1: Reduction Rules.

- $N \equiv (\nu\tilde{n})\, a\{\, \phi :: a\cdot\tilde{l}\langle\tilde{m}\rangle.P + P'' \mid p\cdot\tilde{l}(\tilde{x}).Q + Q'' \mid Q' \,\}_\alpha \mid N'$
- $M \equiv (\nu\tilde{n})\, a\{\, P \mid Q\sigma \mid Q' \,\}_{\alpha'} \mid N'$
- $\alpha \vdash \phi$

Proof. It follows from the definition of the reduction relation $\to$, by induction on the rules in table 5.1. □

5.1.4 An example

Suppose a printer a has two functions: black-and-white and colour printing. The latter service is more expensive and therefore "harder" to get access to. The system is trust-based, meaning that according to its behaviour a principal may not be allowed to use a printer. In `ctm` this corresponds to writing principal $a\{\, P \,\}_\alpha$ where the policy and the protocol are defined as follows. Let message `j` represent the reception of a 'junk document' and $\mathcal{M}$ be the interaction datatype of lists, where the predicate $a\cdot - \rhd \mathtt{j} < k$ checks that messages in the list of type $a\cdot\tilde{l}\rhd\mathtt{j}$ occur less than k times. We then define the policy π as $\{\, Access(x, Colour) \leftarrow x\cdot - \rhd \mathtt{j} < 3;\ Access(x, BW) \leftarrow x\cdot - \rhd \mathtt{j} < 6\}$ where $Access(x,y)$ is a predicate meaning that x can access y. Moreover we assume that $\mathsf{upd}()$ keeps only lists of length at most n deleting the oldest messages and judging if a message is junk. Finally protocol P is defined as

$$P = !x\cdot\mathtt{printC}(y).Access(x, Colour) :: printer\cdot\mathtt{printC}\langle y\rangle \mid$$
$$!x\cdot\mathtt{printBW}(y).Access(x, BW) :: printer\cdot\mathtt{printBW}\langle y\rangle$$

In this example the action of granting access to the printer is modelled by sending a message to *printer*. A user could then be modelled as principal b running the protocol

$$Q = a\cdot\mathtt{printC}\langle\mathtt{spam}\rangle.a\cdot\mathtt{printBW}\langle\mathtt{spam}\rangle.a\cdot\mathtt{printC}\langle\mathtt{spam}\rangle \mid a\cdot\mathtt{printC}\langle\mathtt{doc}\rangle$$

Suppose that upd() will store spam as j and consider the network $N = a\{\ P\ \}_{(\pi,\emptyset)} \mid b\{\ Q\ \}_{\alpha}$ where $\emptyset$ is the empty list. If $a \cdot \mathtt{printC}\langle \mathtt{doc} \rangle$ is executed first, b will get the authorisation to use the printer. But if the left component is all executed then b will no longer be able to colour-print as he has printed too much junk. Note that as we chose the function upd() to keep lists of length at most n, any principal can behave well for n times and regain trust.

5.2 Barbed Equivalences

We now move to study the semantic theory of ctm. We start with the notion of context, which is going to be used for defining equivalences for ctm.

5.2.1 Contexts

In this subsection we define the notion of context for networks, principals, protocols and policies as done in the preliminaries for the π-calculus. We shall use them further on for defining equivalences on ctm terms.

Definition 5.4 (Contexts).

- A *network context* $C[\cdot]$ is defined as $C[\cdot] = \ \cdot\ \mid\ (\nu\tilde{n})\ C[\cdot]\ \mid\ C[\cdot] \mid N$
- An *a-principal context* $C_a[\cdot_1, \cdot_2]$ is defined as $C_a[\cdot_1, \cdot_2] = a\{\ \cdot_1\ \}_{\cdot_2} \mid N$
- An *a-protocol context* $C_a[\cdot]$ is defined as $C_a[\cdot] = a\{\ \cdot \mid R\ \}_{\alpha} \mid N$
- An a,P*-policy context* $C_a^P[\cdot]$ is defined as $C_a^P[\cdot] = a\{\ P\ \}_{\cdot} \mid N$

Given a network context $C[\cdot]$, we say that $C[\cdot]$ is *consistent* with respect to a network N, whenever $C[N]$ is consistent.

5.2.2 Barbs

We now discuss the notion of observation formalised in terms of the actions offered to the environment. Formally we write $N \downarrow a \cdot b$ whenever one of the following conditions is satisfied:

- $N \downarrow a \cdot b$ if $N \equiv (\nu\tilde{n})\ a\{\ \phi :: b \cdot \tilde{l}\langle \tilde{m} \rangle . P + P' \mid Q\ \}_{\alpha} \mid N'$ and $\alpha \vdash \phi, b \notin \mathcal{P}(N')$;
- $N \downarrow a \cdot b$ if $N \equiv (\nu\tilde{n})\ a\{\ p \cdot \tilde{l}(\tilde{x}) . P + P' \mid Q\ \}_{\alpha} \mid N'$ and $b \notin \mathcal{P}(N')$.

where $\mathcal{P}(N')$ is the set of principals contained in N', $\tilde{l} \cap \tilde{n} = \emptyset$ and if $p \notin \mathsf{Var}$ then $p = a$. This definition excludes observing internal and restricted actions. Moreover we write $N \Downarrow a \cdot b$ whenever there exists M such that $N \Rightarrow M$ and $M \downarrow a \cdot b$.

5.2.3 Network Reduction Congruence

The first equivalence we introduce is on networks. We want to define a relation which equates networks in a reasonable way. Above we have defined what is observable, namely part of the operations that a network is able to perform in one move. This suggests that if two networks are to be equivalent they must have, at least, the same barbs. But this is not enough as it could be that they behave the same now, but completely differently in the future moves. To this purpose we should also impose that this equivalence for barbs is preserved by any reduction and any context where the two networks could be placed. Formally, what said is expressed as follows

Definition 5.5 (Network Reduction Congruence). A network reduction congruence is a symmetric relation $\mathcal{R}$ on networks such that whenever $N\mathcal{R}M$

- $N \downarrow a \cdot b$ implies $M \downarrow a \cdot b$;
- $N \rightarrow N'$ implies $M \rightarrow M'$ and $N'\mathcal{R}M'$;
- $C[N]\mathcal{R}C[M]$ for all network contexts $C[\cdot]$ consistent with respect to N and M.

Two networks are reduction congruent ($\simeq$) if related by a network reduction congruence.

The definition given above differs from the one we give in [34]. Nowadays, defining barbed congruence in the style of [63] is strongly recommended as it implies many simplifications when proving completeness of bisimulation proof methods. The definition we give in [34] takes into account only the first two points of the definition above, and then $\simeq$ is defined by closure under every possible context. The definition above implies the one in [34] but the opposite is not always true. We believe that it would hold for `ctm` as it does for the asynchronous π-calculus [50] but we do not consider it in this dissertation. Note that, in Chapter 2, we gave the other definition for the π-calculus. This was done on purpose, so to report, in this dissertation, both methods, and make the reader aware of the two.

We now show an example of network reduction congruence.

Example 5.2. Consider the networks $N = a\{\ a \cdot l\langle\rangle \mid x \cdot l'()\ \}_\emptyset \mid b\{\ P\ \}_\emptyset$ and $M = a\{\ x \cdot l'()\ \}_\emptyset \mid b\{\ P\ \}_\emptyset$. It is easy to see that the two networks are network reduction congruent: there is no context able to distinguish the two networks, as it is only possible to communicate to principal a via an output matching $x \cdot l'()$. In next section, we shall provide a method for proving these equivalences.

Remark 5.1. Note that whenever two networks are reduction congruent, then they must have the same topology. In fact, if that was not the case, we would get a contraddiction by finding a context which distinguishes them. Consider, for example, the networks $N = a\{\ P\ \}_\alpha$ and $M = b\{\ Q\ \}_\beta$ for some P, Q, α and β. If we take the context $C[\cdot] = \cdot \mid c\{\ y \cdot l(x)\ \}_\emptyset$ we then have that $C[N] \not\downarrow b \cdot a$ whereas $C[M] \downarrow b \cdot a$.

5.2.4 Barbed Equivalences for Principals

We now define three different barbed equivalences for principals: one on protocols, one on policies and one on principals.

Protocol Reduction Equivalence

Protocol reduction equivalence compares only protocols. The definition exploits network reduction congruence, namely we impose that given a protocol context, we say that two protocol are equivalent whenever given any protocol context where to be plugged, the resulting networks will always be reduction congruent. Formally,

Definition 5.6 (Protocol Reduction Equivalence). Given a principal a, we say that P and Q are a-reduction congruent, written $P \simeq_a Q$, if $C_a[P] \simeq C_a[Q]$ for all protocol contexts $C_a[\cdot]$.

Intuitively two protocols are equivalent whenever they are able to observe the same events, input the same data and granting access in the same way, i.e. guards are such that there is no policy able to distinguish them. For instance, $\phi :: b \cdot l\langle m \rangle$ and $\phi' :: b \cdot l\langle m \rangle$ are equated only if ϕ and ϕ' hold true for exactly the same set of policies α.

We remark that for protocol reduction equivalence, we made a choice among many. There are many alternatives when choosing the context where the protocol must be plugged. The definition we gave makes protocol reduction equivalence a "non-input" congruence, i.e. the equivalence holds for all those contexts which do not have a hole after an input. The same matter can be discussed for the π-calculus where barbed equivalence does not consider input contexts. From our point of view, composition of contexts is only interesting when used with the parallel operator and not with prefixing. Prefixing looks quite unnatural, but nevertheless, studying a full congruence for protocols would not require deep treatment.

Policy Reduction Equivalence

Varying the kind of contexts we use, we can use reduction congruence to assess policies with respect to a fixed protocol P. The idea is that, given P, two policies are going to be equivalent whenever they "control" P's behaviour in the same way.

Definition 5.7 (Policy Reduction Equivalence). Given a principal a, we say that α and β are a-barbed equivalent with respect to P, written $\alpha \simeq_a^P \beta$, if for all policy contexts $C_a^P[\cdot]$, we have $C_a^P[\alpha] \simeq C_a^P[\beta]$.

This equivalence allows, e.g., to remove formulas which P would never use.

In the following definition we introduce a notion of observation for protocols so that we can see which guards are ever going to be used, and so interest a policy.

Definition 5.8. We write

- $P \downarrow \phi$ if $P \equiv (\nu \tilde{n})\ (\phi :: b \cdot \tilde{l}\langle \tilde{m} \rangle . P + P' \mid P'')$ for $\tilde{n} \cap \tilde{l} = \emptyset$;
- $P \Downarrow \phi$ if there exists N and α such that $a\{\ P\ \}_\alpha \mid N \rightarrow^* a\{\ P'\ \}_{\alpha'} \mid N'$ and $P' \downarrow \phi$;
- $P \Downarrow H$ if $H = \{\phi \mid P \Downarrow \phi\}$.

Note that if $P \Downarrow H$ and for all $\phi \in H$ we have that $(\alpha, s) \vdash \phi$ if and only if $(\beta, s) \vdash \phi$ implies $\alpha \simeq_a^P \beta$. The opposite is of course not true, just consider the protocol $P = \phi :: b \cdot l\langle m \rangle \mid \phi' :: b \cdot l\langle m \rangle$ and policies α and β such that $\alpha \vdash \phi$, $\alpha \nvdash \phi'$, $\beta \vdash \phi'$ and $\beta \nvdash \phi$. In this case we have that $\alpha \simeq_a^P \beta$ but the policies entail different formulas.

In the following we write $\alpha \vdash H$ whenever $H = \{\phi \mid \alpha \vdash \phi\}$.

Theorem 5.1. *Suppose that* $\alpha \vdash H$ *and* $\beta \vdash H'$. *If* $\alpha \simeq_a^P \beta$ *for all P, then H and* H' *are equivalent, i.e. equal up to logical equivalence of the formulas they contain.*

Proof. It follows from the definition. □

Principal Reduction Equivalence. We now introduce the last of our equivalences, principal reduction equivalence.

Definition 5.9 (Principal Reduction Equivalence). Given a principal a, we say that (α, P) and (β, Q) are a-barbed equivalent, written $(\alpha, P) \simeq_a^{\mathcal{P}} (\beta, Q)$, if $C_a[\alpha, P] \simeq C_a[\beta, Q]$ for any a-principal context $C_a[\cdot_1, \cdot_2]$.

It is possible to define the previous two equivalences in terms of barbed principal equivalence. We are now able to state the following

Proposition 5.1. $P \simeq_a Q$ *if and only if for all* α *and protocols R we have that* $(\alpha, P \mid R) \simeq_a^{\mathcal{P}} (\alpha, Q \mid R)$.

Proof. Trivial. □

Proposition 5.2. $\alpha \simeq_a^P \beta$ *if and only if we have that* $(\alpha, P) \simeq_a^{\mathcal{P}} (\beta, P)$.

Proof. Trivial. □

Example 5.3 (Implication). We consider a variation of the printer access control example and apply principal reduction equivalence. Suppose that a server a manages two printers both offering colour and b/w printing as before. The only difference between them is that Printer 1 does not distinguish colour from b/w printing, while Printer 2 does, e.g., by granting access only for b/w printing. For $z \in \{1, 2\}$ we define the following protocol for a print server.

$$\begin{aligned} P(z) = &(\nu n)\ (a \cdot n() \mid !a \cdot n\langle\rangle \,.\, x \cdot z(y) \,.\, Access(z,x) :: x \cdot z\langle OK\rangle \,. \\ &(x \cdot z \cdot col() \,.\, Col(z,x) :: x \cdot z \cdot col\langle OK\rangle \,.\, a \cdot n() + \\ &x \cdot z \cdot bw() \,.\, BW(z,x) :: x \cdot z \cdot bw\langle OK\rangle \,.\, a \cdot n())) \end{aligned}$$

Note that the protocol first checks if it can give access to any type of printing, then verifies which one. The bang is used for writing a recursive protocol: after finishing dealing with a principal, the protocol will be ready once again to provide the service for printer z. The final server protocol is $P(1) \mid P(2)$; its policy is as below, where $\mathtt{j}$ and *doc* represents respectively a junk and a proper document.

$$\begin{aligned} \alpha = \{&Access(1,x) \leftarrow x \cdot 1 \rhd \mathtt{j} \leq x \cdot 1 \rhd doc;\ Col(1,x) \leftarrow Access(1,x); \\ &Col(2,x) \leftarrow x \cdot 2 \rhd \mathtt{j} = 0;\ BW(1,x) \leftarrow Access(1,x);\ BW(2,x) \leftarrow Access(2,x); \\ &Access(2,x) \leftarrow x \cdot 2 \rhd \mathtt{j} \leq 5\} \end{aligned}$$

Then the principal would be represented by the pair $(\alpha, P(1) \mid P(2))$. Using the equivalence $\simeq_a^{\mathcal{P}}$ we can rewrite the principal as $(\alpha, Q \mid P(2))$ where

$$Q = (\nu n)\ (a\cdot n() \mid !a\cdot n\langle\rangle\,.\,x\cdot 1(y)\,.\,Access(1,x) :: x\cdot 1\langle OK\rangle\,.\\ (x\cdot 1\cdot col()\,.\,x\cdot 1\cdot col\langle OK\rangle\,.\,a\cdot n()\ +\ x\cdot 1\cdot bw()\,.\,x\cdot 1\cdot bw\langle OK\rangle\,.\,a\cdot n()))$$

In fact $(\alpha, P(1) \mid P(2)) \simeq_a^{\mathcal{P}} (\alpha, Q \mid P(2))$ and this can be explained by the following argument. In protocol $P(1) \mid P(2)$ the output $x\cdot 1\cdot Col\langle \mathtt{OK}\rangle$ is guarded by $Col(1,x)$ instead in $Q \mid P(2)$ it is unguarded. We need to show that $Col(1,x)$ is always true at that point of the computation. In fact in α the predicate $Access(1,x)$ implies $Col(1,x)$ and the action $x\cdot 1\cdot bw$ does not change the value of $Access(1,x)$ (Q and $P(z)$ are sequential) as well as the inputs executed by the branch $P(2)$. We shall give more formal proof in Section 5.3.7.

It is straightforward to prove that $\alpha \simeq_a^{Q|P(2)} \alpha'$ where

$$\alpha' = \{\, Access(1,x) \leftarrow x\cdot 1 \rhd \mathtt{j} \le x\cdot 1 \rhd doc;\ Col(2,x) \leftarrow x\cdot 2 \rhd \mathtt{j} = 0;\\ Access(2,x) \leftarrow x\cdot 2 \rhd \mathtt{j} \le 5\}.$$

Now by Proposition 5.2 we then have that $(\alpha, P(1) \mid P(2)) \simeq_a^{\mathcal{P}} (\alpha', Q \mid P(2))$.

5.2.5 A Weak Reduction Congruence for Networks

Above we have analysed equivalences for manipulating single principals and networks. We now introduce a weaker kind of equivalence for networks, called weak reduction congruence. As explained in the preliminaries in the case of π-calculus, weak equivalences usually ignore silent moves, i.e. a reduction in another part of the network is not considered. We now give the definition for `ctm` networks.

Definition 5.10 (Weak Network Reduction Congruence). A network reduction congruence is a symmetric relation $\mathcal{R}$ on networks such that whenever $N\mathcal{R}M$

- $N \downarrow a\cdot b$ implies $M \Downarrow a\cdot b$;
- $N \rightarrow N'$ implies $M \Rightarrow M'$ and $N'\mathcal{R}M'$;
- $C[N]\mathcal{R}C[M]$ for all network contexts $C[\cdot]$ consistent with respect to N and M.

Two networks are reduction congruent ($\cong$) if related by a network reduction congruence.

From the previous definition it follows that the weak case includes the strong one.

Proposition 5.3. *Weak network reduction congruence includes network reduction congruence.*

Proof. It is clear by the definition, that any network reduction congruence is also weak. □

We now show an interesting application of such a congruence in an example where recommendations are taken into account.

Example 5.4 (Recommendations and Trust Security). Trusting someone may be a consequence of observed good behaviour, but it may also be a consequence of good recommendations from a trusted third party. In this example we show how to describe in `ctm` a system where principals' trust is based both on observations and recommendations. We consider a European network of banks where each bank issues mortgages for customers according to a policy. The policy grants a mortgage whenever the customer has always paid back previous mortgages and, additionally, other banks' opinions about the customer are positive. In `ctm`, granting the mortgage is equivalent to proving a predicate $\mathtt{G}(x)$ from the following policy

$$\pi_1(Y) = \{\mathtt{G}(x) \leftarrow Y \cdot - \rhd (x,\mathtt{Bad}) = 0, \mathtt{M}(x);\ \mathtt{Good}(x) \leftarrow \mathtt{M}(x),\ \mathtt{Bad}(x) \leftarrow \mathtt{NoM}(x)\}.$$

The interaction datatype is multiset. The predicates `Good` and `Bad` (which will be used for recommendations) depend on the predicate `M` and `NoM` only (local observations), instead the predicate `G` depends also on the recommendations received from Y. The predicate $\mathtt{M}(x)$ is assumed to check whether every mortgage granted to x is matched by a full repayment: in order to get a mortgage, there must be no outstanding mortgages. This is expressed by identifying messages with a fresh name w. We can then define the following template for banks.

$$\begin{aligned} P_1(X,Y) = {} & !x \cdot \mathtt{mg}(w) \,.\, (\nu k)\ (Y \cdot \mathtt{rec}\langle k,x\rangle \,.\, Y \cdot k(x,z) \,.\, \mathtt{G}(x) :: X \cdot \mathtt{gr}\langle x,w\rangle) \\ & \mid !X \cdot \mathtt{gr}(x,w) \,.\, x \cdot w\langle\rangle \,.\, x \cdot w() \\ & \mid !Y \cdot \mathtt{rec}(k,x) \,.\, (\mathtt{Good}(x) :: Y \cdot k\langle x,\mathtt{Good}\rangle + \mathtt{Bad}(x) :: Y \cdot k\langle x,\mathtt{Bad}\rangle) \end{aligned}$$

Bank X has three components: one for granting mortgages, one for accounting, and one for giving recommendations to another bank Y. When receiving a request from a on channel `mg` a fresh name w reserved for the particular transaction is received. Then X will send a request of recommendation to bank Y. At this point, if the predicate `G` is provable from the policy, the protocol will transfer the request to the accounting component (on channel `gr`), which will send an authorisation message to a, and will finally be waiting for a repayment. In case `G` is not provable, the request will be pending. As pointed out before, the policy will take care of denying further authorisations until repayment. The third component, which gives recommendations to bank Y, just checks whether the predicate `Good` or `Bad` are provable, and sends a message accordingly. Suppose now that B_F and B_I are respectively French and Italian banks. We can define a network of banks as follows.

$$N_1 = B_F\{\ P_1(B_F,B_I)\ \}_{(\pi_1(B_I),\emptyset)} \mid B_I\{\ P_1(B_I,B_F)\ \}_{(\pi_1(B_F),\emptyset)}$$

Suppose now that a customer a has just moved from Italy to France, and she is asking the French bank B_F for a mortgage, e.g. with the protocol $(\nu w)\ B_F \cdot \mathtt{mg}\langle w\rangle \,.\, B_F \cdot w() \,.\, B_F \cdot w\langle\rangle$. Let us now define a different network of banks using a third party (O) which is going to deal with all the requests. The following policy is used by principal O.

$$\pi_2(X,Y) = \{\mathtt{G}(X,x) \leftarrow \mathtt{M}(Y,x), \mathtt{M}(X,x)\}$$

All the operations previously performed by the banks will now be performed by this policy, and we no longer need recommendations. The following is part of the protocol for principal O.

$$F(W) = W(x,w)\,.\,\mathtt{G}(W,x) :: O\cdot W\cdot \mathtt{gr}\langle x,w\rangle \mid !O\cdot W\cdot \mathtt{gr}(x,w)\,.\,W\cdot w\langle\rangle\,.\,W\cdot w()$$

The banks will only forward messages from the principals to O and vice-versa.

$$P_2 = !x\cdot \mathtt{mg}(w)\,.\,O\langle x,w\rangle\,.\,O\cdot w()\,.\,x\cdot w\langle\rangle\,.\,x\cdot w()\,.\,O\cdot w\langle\rangle$$

The entire new network will be

$$N_2 = O\{\ !(F(B_F)+F(B_I))\ \}_{(\pi_2(B_F,B_I)\cup\pi_2(B_I,B_F),\emptyset)} \mid B_F\{\ P_2\ \}_{(\emptyset,\emptyset)} \mid B_I\{\ P_2\ \}_{(\emptyset,\emptyset)}$$

We then have that $N_1 \sim N_2$ accordingly to the definition of $\sim$. We shall prove this in Section 5.3.7 where we provide a proof method for this equivalence.

Note that in this example, O works as a "headquarter" which collects information from the banks. This is a further step from the work in the previous chapter (and [33]), where we computed principal trust policies as the least fixed point of a global function. Such a thing, plainly unfeasible in a distributed scenario, can be implemented in `ctm`. For instance, the global trust function can be expressed as the policy of a principal, e.g. O, and the fixed point as a computation. Then, using the network equivalence, one may be able to simplify the network and avoid the use of "headquarters," like in this example. This is generic technique for stating (and proving) the correctness of a "web of trust", with a specification in the form of a centralised "headquarter".

5.3 A Characterisation of Barbed Equivalences

As mentioned above, the equivalences introduced lack a proof method for showing whether two processes or two networks are in the same equivalence class. E.g., given two networks, proving that they are network equivalent would require a quantification over all possible contexts, and this is not what we want. In concurrency theory, it is common to provide a proof method for this purpose. In the preliminaries we saw how this is done for the π-calculus, and how powerful the method is.

5.3.1 A labelled transition system

We now give an alternative definition for the semantics of `ctm` in terms of labeled transition system. The new semantics consists of two relations: one for protocols and one for networks. In Table 5.2 there are the rules for the relation $\xrightarrow{\mu}_a$. The semantics (relation) deals with pairs policy-protocol and has the additional feature of showing labels μ which are defined by the grammar:

$$\mu ::= a\cdot\tilde{l}?\tilde{m} \mid a\cdot\tilde{l}!\nu\tilde{n} : \tilde{m} \mid \tau$$

$$(\textsc{Out})\ \frac{\alpha \vdash \phi}{(\alpha, \phi :: b \cdot \tilde{l}\langle \tilde{m} \rangle . P) \xrightarrow{b \cdot \tilde{l}!\nu\emptyset:\tilde{m}}_a (\alpha, P)}$$

$$(\textsc{In})\ \frac{\alpha' = \mathsf{upd}(\alpha, b \cdot \tilde{l} \rhd \tilde{m}) \quad b : \tilde{m} \odot p : \tilde{x} = \sigma}{(\alpha, p \cdot \tilde{l}(\tilde{x}) . P) \xrightarrow{b \cdot \tilde{l}?\tilde{m}}_a (\alpha', P\sigma)}$$

$$(\textsc{Com})\ \frac{(\alpha, P) \xrightarrow{a \cdot \tilde{l}!\nu\tilde{n}:\tilde{m}}_a (\alpha, P') \qquad (\alpha, Q) \xrightarrow{a \cdot \tilde{l}?\tilde{m}}_a (\alpha', Q')}{(\alpha, P \mid Q) \xrightarrow{\tau}_a (\alpha', (\nu\tilde{n})\ (P' \mid Q'))}$$

$$(\textsc{Open})\ \frac{(\alpha, P) \xrightarrow{b \cdot \tilde{l}!\nu\tilde{n}:\tilde{m}}_a (\alpha', P') \quad n \in \tilde{m} \backslash \tilde{n}}{(\alpha, (\nu n)\ P) \xrightarrow{b \cdot \tilde{l}!\nu n\tilde{n}:\tilde{m}}_a (\alpha', P')} n \notin \tilde{l}$$

$$(\textsc{Par})\ \frac{(\alpha, P) \xrightarrow{\mu}_a (\alpha', P')}{(\alpha, P \mid Q) \xrightarrow{\mu}_a (\alpha', P' \mid Q)} bn(\mu) \cap fn(Q) = \emptyset$$

$$(\textsc{Sum})\ \frac{(\alpha, Z) \xrightarrow{\mu}_a (\alpha', Z')}{(\alpha, Z + Z") \xrightarrow{\mu}_a (\alpha', Z')}$$

$$(\textsc{Bang})\ \frac{(\alpha, P \mid !P) \xrightarrow{\mu}_a (\alpha, P')}{(\alpha, !P) \xrightarrow{\mu}_a (\alpha', P')}$$

$$(\textsc{Res})\ \frac{(\alpha, P) \xrightarrow{\mu}_a (\alpha', P')}{(\alpha, (\nu n)\ P) \xrightarrow{\mu}_a (\alpha', (\nu n)\ P)} n \notin \mathtt{nams}(\mu)$$

Table 5.2: lts for protocols

We give a short explanation of all the rules so to clarify their meaning. The rule (OUT) gives semantics to the output operation. Everytime that a principal is able to perform an output, this is shown by the label $b \cdot \tilde{l}!\nu\emptyset\tilde{m}$, meaning that the protocol wants to send a message $\tilde{m}$ to principal b on channel $\tilde{l}$. The rule (IN) is the corresponding one for the operation of input. In here the label $b \cdot \tilde{l}?\tilde{m}$ is shown, where, non deterministically, a substitution $\tilde{m}$ for the variables $\tilde{x}$ is chosen. As seen for the π-calculus in Chapter 2, this identifies an early semantics. The rule (COM) takes care of internal communication, and somehow corresponds to the rule (RINT) for the reduction semantics. By using labels, this is shown with the action τ. The rule (OPEN) is meant to deal with restrictions and is similar to the one for the π-calculus, where we have a polyadic communication. Namely, two kind of checks are made: one is that the

$$(\textsc{Prin})\ \frac{(\alpha,P)\ \xrightarrow{\mu}_a\ (\alpha',P')}{a\{\ P\ \}_\alpha\ \xrightarrow{a\star\mu}\ a\{\ P'\ \}_{\alpha'}}\texttt{dest}(\mu)\neq a,\texttt{bn}(\mu)\cap\texttt{fn}(\alpha)=\emptyset$$

$$(\textsc{Com}_N)\ \frac{N\ \xrightarrow{a\cdot b\cdot\tilde{l}?\tilde{m}}\ N'\quad M\ \xrightarrow{b\cdot a\cdot\tilde{l}!\nu\tilde{n}:\tilde{m}}\ M'}{N\mid M\ \xrightarrow{\tau}\ (\nu\tilde{n})\ (N'\mid M')}$$

$$(\textsc{Open})\ \frac{N\ \xrightarrow{a\cdot b\cdot\tilde{l}!\nu\tilde{n}:\tilde{m}}\ N'\quad n\in\tilde{m}\backslash\tilde{n}}{(\nu n)\ N\ \xrightarrow{a\cdot b\cdot\tilde{l}!\nu n\tilde{n}:\tilde{m}}\ N'}n\notin\tilde{l}$$

$$(\textsc{Res})\ \frac{N\ \xrightarrow{\xi}\ N'}{(\nu n)\ N\ \xrightarrow{\xi}\ (\nu n)\ N'}n\notin\texttt{nams}(\xi)$$

$$(\textsc{Par}_N)\ \frac{N\ \xrightarrow{\xi}\ N'}{N\mid M\ \xrightarrow{\xi}\ N'\mid M}\quad\texttt{dest}(\xi)\cap\mathcal{P}(M)=bn(\xi)\cap fn(N')=\emptyset$$

Table 5.3: lts for networks

restricted name is not in the synchronisation vector $\tilde{l}$ in order to keep its privacy and the second checks that the restricted name n is actually being communicated over the channel. Rule (PAR) is used to propagate actions with parallel operator. Rules (SUM), (BANG) and (RES) are standard and equal to the ones for the π-calculus.

We can now move to define the semantics of networks in terms of a labelled transition system, i.e. the relation $\cdot\ \dot{\longrightarrow}\ \cdot\subseteq N\times\xi\times N$. The rules can be found in table 5.3 where ξ are the labels and are defined as

$$\xi ::= a\cdot b\cdot\tilde{l}?\tilde{m}\mid a\cdot b\cdot\tilde{l}!\nu\tilde{n}:\tilde{m}\mid\tau$$

In the following table we report the definitions for the functions on labels which may be used in the labelled transition system.

	$a\cdot b\cdot\tilde{l}?\tilde{m}$	$a\cdot b\cdot\tilde{l}!\nu\tilde{n}:\tilde{m}$	τ
sndr	$\{a\}$	$\{a\}$	$\{\}$
dest	$\{b\}$	$\{b\}$	$\{\}$
fn	$\{a,b,l_1,\ldots,l_{k_1},m_1\ldots m_{k_2}\}$	$\{a,b,l_1,\ldots,l_{k_1},m_1\ldots m_{k_3}\}$	$\{\}$
bn	$\{\}$	$\{n_1\ldots n_k\}$	$\{\}$

The functions above they can also be applied to elements μ excluding the function sndr.

The function nams() on labels is defined as usual as $\texttt{nams}(\xi)=\texttt{fn}(\xi)\cup\texttt{bn}(\xi)$

As done for networks, we now give a short explanation for protocols rules. The rule (PRIN) deals with the semantics of protocols and it is the only rule where this is done. Basically, any action μ that can be done by a protocol P under a policy α is brought up at network level by adding to the label the principal where the action happened. This is done by using the operator $\star$, which given a principal name a and a protocol label μ returns a network label ξ. In case μ is a τ then $\star$ returns τ. The other rules are very similar to the ones at protocol level. Note that the Rule (COM_N) takes care of external communication and this reflects what the Rule (RCOM) does in the reduction semantics.

We can now state some facts about the labelled transition system we have presented above. The following give a characterisation of the syntax based on the labels networks and protocols expose to a possible environment.

Lemma 5.2. *Given α and P such that $(\alpha, P) \xrightarrow{\mu}_a (\alpha', P')$ we have that*

1. *if $\mu = b \cdot \tilde{l}?\tilde{m}$ then $P \equiv (\nu\tilde{n}')\ (p \cdot \tilde{l}(\tilde{x}) \,.\, P'' + P''' \mid Q)$ for some P'', P''' and Q such that $p \in \mathsf{Var} \cup \{\mathsf{b}\}$ and $\tilde{l} \cap \tilde{n}' = \tilde{n}' \cap \tilde{m} = \emptyset$.*
2. *if $\mu = b \cdot \tilde{l}!\nu\tilde{n} : \tilde{m}$ then $P \equiv (\nu\tilde{n}')\ (\phi :: b \cdot \tilde{l}\langle\tilde{m}\rangle \,.\, P'' + P''' \mid Q)$ for some ϕ, P'', P''' and Q such that $\alpha \vdash \phi$, $\tilde{n} \subseteq \tilde{n}'$ and $\tilde{n} \cap \mathtt{fn}(Q) = \tilde{l} \cap \tilde{n}' = \emptyset$.*
3. *If $\mu = \tau$ then $P \equiv (\nu\tilde{n})\ (\phi :: b \cdot \tilde{l}\langle\tilde{m}\rangle \,.\, P' + P'' \mid p \cdot \tilde{l}(\tilde{x}) \,.\, Q' + Q'' \mid R)$ for some ϕ, P', P'', Q' and Q'' such that $\alpha \vdash \phi$.*

Proof.

1. We prove it by induction on the rules for the protocol lts in Table 5.2.

   - Rule (OUT) is not applicable;
   - if we apply rule (IN) we then have that $P = p \cdot \tilde{l}(\tilde{x}) \,.\, P'$ with $p \in \mathsf{Var} \cup \{\mathsf{b}\}$;
   - Rule (COM) is not applicable;
   - Rule (OPEN) is not applicable;
   - Rule (PAR) implies that $P = P^* \mid Q^*$ with $\mathtt{bn}(\mu) \cap \mathtt{fn}(Q^*) = \emptyset$ where $\mathtt{bn}(\mu) = \emptyset$. Now, by induction hypothesis, we have that $(\alpha, P^*) \xrightarrow{\mu}_a (\alpha', P^{**})$ implies that $P^* \equiv (\nu\tilde{n}')\ (p \cdot \tilde{l}(\tilde{x}) \,.\, P'' + P''' \mid Q)$ for some P'', P''' and Q such that $p \in \mathsf{Var} \cup \{\mathsf{b}\}$ and $\tilde{l} \cap \tilde{n}' = \tilde{n}' \cap \tilde{m} = \emptyset$. This implies that $P \equiv (\nu\tilde{n}')\ (p \cdot \tilde{l}(\tilde{x}) \,.\, P'' + P''' \mid Q) \mid Q^*$ and we finally have $P \equiv (\nu\tilde{n}'')\ ((p \cdot \tilde{l}(\tilde{x}) \,.\, P'' + P''' \mid Q)\sigma \mid Q^*)$ for fresh names $\tilde{n}''$ such that $\tilde{n}'' \cap \mathtt{fn}(Q^*) = \emptyset$ and $\tilde{l} \cap \tilde{n}'' = \tilde{n}'' \cap \tilde{m} = \emptyset$ and a substitution σ for α-converting $\tilde{n}'$ into $\tilde{n}''$ in $p \cdot \tilde{l}(\tilde{x}) \,.\, P'' + P''' \mid Q$;
   - The Rule (SUM) needs little attention. In fact, we have to be careful when dealing with sum as we might get terms which are not allowed by the syntax. As $P = Z + Z'$, we have, by induction hypothesis, that $(\alpha, Z) \xrightarrow{\mu}_a (\alpha', P^*)$ implies that $Z \equiv (\nu\tilde{n}')\ (p \cdot \tilde{l}(\tilde{x}) \,.\, P'' + P''' \mid Q)$ for some P'', P''' and Q such that $p \in \mathsf{Var} \cup \{\mathsf{b}\}$ and $\tilde{l} \cap \tilde{n}' = \tilde{n}' \cap \tilde{m}\ ^* = \emptyset$. Now, as Z is a component of a sum it can only be that $Q \equiv 0$, i.e. $Z \equiv (\nu\tilde{n}')\ (p \cdot \tilde{l}(\tilde{x}) \,.\, P'' + P''')$. We then get that $P \equiv (\nu\tilde{n}')\ (p \cdot \tilde{l}(\tilde{x}) \,.\, P'' + P''') + Z'$

and with a renaming similar to the proof for Rule (PAR) we have $P \equiv (\nu\tilde{n}'')\ ((p \cdot \tilde{l}(\tilde{x}) \,.\, P'')\sigma + (P'''\sigma + Z'))$;

- Rule (BANG) implies that $P = !P^*$. Then, by induction hypothesis, we get that $(\alpha, P^* \mid !P^*) \xrightarrow{\mu}_a (\alpha', P^{**})$ implies that $P^* \mid !P^* \equiv (\nu\tilde{n}')\ (p \cdot \tilde{l}(\tilde{x}) \,.\, P'' + P''' \mid Q)$ for some P'', P''' and Q such that $p \in \mathsf{Var} \cup \{\mathsf{b}\}$ and $\tilde{l} \cap \tilde{n}' = \tilde{n}' \cap \tilde{m} = \emptyset$ and as $P^* \mid !P^* \equiv P^*$, this holds also for P.
- When applying Rule (RES) we have that $P = (\nu n)\ P^*$ and then, again by induction hypothesis, we have that $(\alpha, P^*) \xrightarrow{\mu}_a (\alpha', P^{**})$ implies that $P^* \equiv (\nu\tilde{n}')\ (p \cdot \tilde{l}(\tilde{x}) \,.\, P'' + P''' \mid Q)$ for some P'', P''' and Q such that $p \in \mathsf{Var} \cup \{\mathsf{b}\}$ and $\tilde{l} \cap \tilde{n}' = \tilde{n}' \cap \tilde{m} = \emptyset$. To the last conditions we can add the rule side-condition, i.e. $n \notin \mathtt{nams}(\mu)$.

2. As well as the previous point, we proceed by induction on the rules for the protocol lts in Table 5.2.

   - if we apply rule (OUT) we then have that $P = \phi :: b \cdot \tilde{l}\langle\tilde{m}\rangle \,.\, P''$ with $\alpha \vdash \phi$;
   - Rule (IN) is not applicable;
   - Rule (COM) is not applicable;
   - When applying Rule (OPEN) we then have that $P = (\nu n)\ P^*$. By induction hypothesis, we have that for $n \notin \tilde{l}$ and $n \in \tilde{m} \backslash \tilde{n}$, $(\alpha, P^*) \xrightarrow{b \cdot \tilde{l} ! \nu\tilde{n} : \tilde{m}}_a (\alpha', P^{**})$ implies that $P^* \equiv (\nu\tilde{n}')\ (\phi :: b \cdot \tilde{l}\langle\tilde{m}\rangle \,.\, P'' + P''' \mid Q)$ for some ϕ, P'', P''' and Q such that $\alpha \vdash \phi$, $\tilde{n} \subseteq \tilde{n}'$ and $\tilde{n} \cap \mathtt{fn}(Q) = \tilde{l} \cap \tilde{n}' = \emptyset$. It follows that $P \equiv (\nu n^*)\ (\nu\tilde{n}')\ ((\phi :: b \cdot \tilde{l}\langle\tilde{m}\rangle \,.\, P'' + P''')\sigma \mid Q)$ such that $\alpha \vdash \phi$, $\tilde{n} \cup \{n^*\} \subseteq \tilde{n}' \cup \{n^*\}$, $(\tilde{n} \cup \{n^*\}) \cap \mathtt{fn}(Q) = \emptyset$, $\tilde{l} \cap (\tilde{n}' \cup \{n\}) = \emptyset$ (as $n \notin \tilde{l}$) and σ a substitution for replacing n with n^* in order to avoid clashes.
   - Rule (PAR) implies that $P = P^* \mid Q^*$ with $\mathtt{bn}(\mu) \cap \mathtt{fn}(Q^*) = \emptyset$. Now, by induction hypothesis, we have that $(\alpha, P^*) \xrightarrow{\mu}_a (\alpha', P^{**})$ implies that $P^* \equiv (\nu\tilde{n}')\ (\phi :: b \cdot \tilde{l}\langle\tilde{m}\rangle \,.\, P'' + P''' \mid Q)$ for some ϕ, P'', P''' and Q such that $\alpha \vdash \phi$, $\tilde{n} \subseteq \tilde{n}'$ and $\tilde{n} \cap \mathtt{fn}(Q) = \tilde{l} \cap \tilde{n}' = \emptyset$. This implies that $P \equiv (\nu\tilde{n}')\ (p \cdot \tilde{l}(\tilde{x}) \,.\, P'' + P''' \mid Q) \mid Q^*$ and, as $\mathtt{bn}(\mu) = \tilde{n}$ and $\mathtt{bn}(\mu) \cap \mathtt{fn}(Q^*) = \emptyset$ we can do an α-conversion as before;
   - Rule (SUM) similar to the input case;
   - Rule (BANG) similar to the input case;
   - Rule (RES) similar to the input case.

3. It follows from the previous two points.

□

Lemma 5.3. *The following results hold*

1. *If* $N \xrightarrow{a \cdot b \cdot \tilde{l} ? \tilde{m}} N'$ *then* $N \equiv (\nu\tilde{n}')\ (a\{\ p \cdot \tilde{l}(\tilde{x}) \,.\, P + P' \mid Q\ \}_\alpha \mid M)$ *for some P, Q,* ϕ*, M and such that* $p \in \mathsf{Var} \cup \{\mathsf{b}\}$*,* $b \neq a$ *and* $\tilde{l} \cap \tilde{n}' = \tilde{n}' \cap \tilde{m} = \emptyset$*;*

2. *If* $N \xrightarrow{a\cdot b\cdot \tilde{l}!\nu\tilde{n}:\tilde{m}} N'$ *then* $N \equiv (\nu\tilde{n}')\ (a\{\ \phi : b\cdot\tilde{l}\langle\tilde{m}\rangle\,.P+P' \mid Q\ \}_\alpha \mid M)$ *for some* P, Q, ϕ *and* M *such that* $\alpha \vdash \phi$, $b \neq a$, $\tilde{n} \subseteq \tilde{n}'$ *and* $\tilde{n} \cap (\mathtt{fn}(M) \cup \mathtt{fn}(Q)) = \tilde{l} \cap \tilde{n}' = \emptyset$;

3. *If* $N \xrightarrow{\tau} N'$ *then either*

   - $N \equiv (\nu\tilde{n})\ (a\{\ \phi :: b\cdot\tilde{l}\langle\tilde{m}\rangle\,.P+P' \mid Q\ \}_\alpha \mid b\{\ p\cdot\tilde{l}(\tilde{x})\,.R+R' \mid Q\ \}_\beta \mid M)$ *such that* $p \in \mathsf{Var}\cup\{\mathsf{b}\}$, $\alpha \vdash \phi$ *and* $b \notin \mathcal{P}(M)$

   or

   - $N \equiv (\nu\tilde{n})\ (a\{\ \phi :: b\cdot\tilde{l}\langle\tilde{m}\rangle\,.P+P' \mid p\cdot\tilde{l}(\tilde{x})\,.R+R' \mid Q\ \}_\alpha \mid M)$ *such that* $p \in \mathsf{Var}\cup\{\mathsf{a}\}$, $\alpha \vdash \phi$ *and* $a \notin \mathcal{P}(M)$

Proof.

1. By induction on the lts rules for networks from table 5.3.

   - Rule (PRIN) is the base of the induction and, by using the previous lemma, it follows what we want to prove;
   - Rule (COM_N) is not applicable;
   - Rule (OPEN) is not applicable;
   - Rule (RES) implies that $N = (\nu n)\ N^*$ and then we have that $N^* \xrightarrow{\xi} N^{**}$ witn $n \notin \mathtt{nams}(\xi)$ implies that $N^* \equiv (\nu\tilde{n}')\ (a\{\ p\cdot\tilde{l}(\tilde{x})\,.P \mid Q\ \}_\alpha \mid M)$ for some P, Q, ϕ, M and such that $p \in \mathsf{Var}\cup\{\mathsf{b}\}$, $b \neq a$ and $\tilde{l} \cap \tilde{n}' = \tilde{n}' \cap \tilde{m} = \emptyset$. Now, with a substitution and a renaming of bound names we can get what we want, similarly to previous lemma;
   - When applying Rule (PAR_N) we than have something similar to previous lemma: $N \equiv N^* \mid M^*$ and similarly, by induction we can build the term we need.

2. As in the previous case, consider all rules from table 5.3 such that $N \xrightarrow{a\cdot b\cdot\tilde{l}?\tilde{m}} N'$.

   - As in the previous point Rule (PRIN) is a consequence of the previous lemma;
   - The Rule (COM_N) is not applicable;
   - Rule (OPEN) implies that $N \equiv (\nu n)\ N^*$ and by induction hypothesis we have that, for $n \notin \tilde{l}$ and $n \in \tilde{m}\backslash\tilde{n}$, $N^* \xrightarrow{a\cdot b\cdot\tilde{l}!\nu\tilde{n}:\tilde{m}} N^{**}$ implies that $N^* \equiv (\nu\tilde{n}')\ (a\{\ \phi : b\cdot\tilde{l}\langle\tilde{m}\rangle\,.P \mid Q\ \}_\alpha \mid M)$ for some P, Q, ϕ and M such that $\alpha \vdash \phi$, $b \neq a$, $\tilde{n} \subseteq \tilde{n}'$ and $\tilde{n} \cap (\mathtt{fn}(M) \cup \mathtt{fn}(Q)) = \tilde{l} \cap \tilde{n}' = \emptyset$. It follows that $N \equiv (\nu n^*)\ (\nu\tilde{n}')\ (a\{\ (\phi : b\cdot\tilde{l}\langle\tilde{m}\rangle\,.P)\sigma \mid Q\ \}_\alpha \mid M)$ with n^* and σ for dealing with renaming as before;
   - (RES) is similar to the input case;
   - (PAR_N) is similar to the input case.

3. It follows by the two previous points.

□

We are now able to show that the *lts* semantics is equivalent to the reduction semantics we gave previously. The following theorem shows the correspondence between reduction steps and τ actions.

Lemma 5.4 (Harmony Lemma). *Let N be a network. Then*

1. *If $N \xrightarrow{\tau} M$ then $N \to M$;*
2. *If $N \to M$ then $N \xrightarrow{\tau} \equiv M$.*

Proof. We prove the two points separately:

1. Suppose that $N \xrightarrow{\tau} M$. Then, by Lemma 5.3, we have that either

   - $N \equiv (\nu\tilde{n})\ (a\{\ \phi :: b \cdot \tilde{l}\langle\tilde{m}\rangle . P + P' \mid Q\ \}_\alpha \mid b\{\ p \cdot \tilde{l}(\tilde{x}) . R + R' \mid Q\ \}_\beta \mid M)$ such that $p \in \mathsf{Var} \cup \{\mathsf{b}\}$, $\alpha \vdash \phi$ and $b \notin \mathcal{P}(M)$

   or

   - $N \equiv (\nu\tilde{n})\ (a\{\ \phi :: b \cdot \tilde{l}\langle\tilde{m}\rangle . P + P' \mid p \cdot \tilde{l}(\tilde{x}) . R + R' \mid Q\ \}_\alpha \mid M)$ such that $p \in \mathsf{Var} \cup \{\mathsf{a}\}$, $\alpha \vdash \phi$ and $a \notin \mathcal{P}(M)$

   It follows that $N \to M$ by applying the reduction rules (COMM) and (RINT);

2. Suppose that $N \to M$, then by Lemma 5.1, everything follows by applying the lts for networks.

□

Now that we have given the lts for `ctm` we are able to give the definitions of the bisimulations which are going to characterise the barbed congruence which we introduced in the previous section.

5.3.2 Network bisimulation

Network bisimulation is the bisimulation related to networks. It follows its definition.

Definition 5.11 (Network Bimulation). A network bisimulation is a symmetric binary relation $\mathcal{S}$ such that $N\mathcal{S}M$ and $N \xrightarrow{\xi} N'$, for $\mathtt{dest}(\xi) \cap \mathcal{P}(N \mid M) = \emptyset$, implies that $M \xrightarrow{\xi} M'$ for some M' and $N'\mathcal{S}M'$.
N and M are network bisimilar, written $N \sim M$, if they are related by network bisimulation.

The following theorem states that the bisimulation we have defined above is a congruence, i.e. given two networks which are bisimalar we can always plug them in any context and they will still be related by $\sim$.

Theorem 5.2. *Strong network bisimulation is a congruence.*

Proof. We prove this theorem by showing that there exists an equivalence relation $\mathcal{R}$ which is a congruence, a bisimulation and contains $\sim$. In other words, let $\mathcal{R}$ be the least equivalence relation satisfying the following closure conditions:

- $\sim \subseteq \mathcal{R}$;
- $N \mathcal{R} M$ implies $(N \mid O)\ \mathcal{R}\ (M \mid O)$ for all networks O.
- $N \mathcal{R} M$ implies $(\nu n)\ N\ \mathcal{R}\ (\nu n)\ M$ for all names n.

The equivalence relation $\mathcal{R}$ is clearly a congruence by definition. If we then to prove that $\mathcal{R}$ is a bisimulation it will follow that $\mathcal{R} \subseteq \sim$ and as by definition $\sim \subseteq \mathcal{R}$ we have that $\mathcal{R} = \sim$. The proof proceeds by induction on the closure above (we implicitly consider the cases such that $\mathtt{dest}(\xi) \cap \mathcal{P}(N \mid M) = \emptyset$):

- ($N \mathcal{R} M$ because $N \sim M$).
- ($(N \mid O)\ \mathcal{R}\ (M \mid O)$ because $N \sim M$). Suppose that $N \mid O \xrightarrow{\xi} O'$. By cases on ξ and the lts rules we have 8 possibilities that we regroup in 4 cases:
  - If, for any ξ, we apply the rule ($\textsc{Par}_N$) for the left parallel component then, assuming that $\mathtt{sndr}(\xi) \cap \mathtt{prin}(O) = bn(\xi) \cap fn(N') = \emptyset$, we have

    $$\frac{N \xrightarrow{\xi} N'}{N \mid O \xrightarrow{\xi} N' \mid O}$$

    Now, as $N \mathcal{R} M$, we have (by induction hypothesis) that $M \xrightarrow{\xi} M'$ with $N' \mathcal{R} M'$. Applying ($\textsc{Par}_N$) for the left parallel component we get $M \mid O \xrightarrow{\xi} M' \mid O$ and as $\mathcal{R}$ is closed by definition under the contexts of type $[] \mid O$, we have $N' \mid O\ \mathcal{R}\ M' \mid O$.
  - Applying the rule ($\textsc{Par}_N$) for the right parallel component, for any ξ and $\mathtt{sndr}(\xi) \cap \mathtt{prin}(O) = bn(\xi) \cap fn(N') = \emptyset$, we have

    $$\frac{O \xrightarrow{\xi} O''}{N \mid O \xrightarrow{\xi} N \mid O''}$$

    Reasoning similarly to the previous case, we trivially derive that $M \mid O \xrightarrow{\xi} M \mid O''$ with $N \mid O''\ \mathcal{R}\ M \mid O''$.
  - If we apply the rule ($\textsc{Com}_N$), for $\xi = \tau$ and $N \xrightarrow{a \cdot b \cdot \tilde{l}?\tilde{m}} N'$ we get

    $$\frac{N \xrightarrow{a \cdot b \cdot \tilde{l}?\tilde{m}} N' \quad O \xrightarrow{b \cdot a \cdot \tilde{l}!\nu\tilde{n}:\tilde{m}} O''}{N \mid O \xrightarrow{\tau} (\nu\tilde{n})\ (N' \mid O'')}$$

    As in the previous cases, by induction hypothesis, we have that $M \xrightarrow{\xi} M'$ with $N' \mathcal{R} M'$. We can then build the same interaction for M and, as expected, get $M \mid O \xrightarrow{\tau} (\nu\tilde{n})\ (M' \mid O'')$. Again, as we defined $\mathcal{R}$ as a closure with respect to the contexts $([] \mid O)$ and $(\nu n)\ []$, also $(\nu\tilde{n})\ (N' \mid O'')\ \mathcal{R}\ (\nu\tilde{n})\ (M' \mid O'')$.

– The last case is when swapping the actions of N and O of the previous case, i.e. for $\xi = \tau$ and $N \xrightarrow{a\cdot b\cdot \tilde{l}?\tilde{m}} N'$

$$\frac{N \xrightarrow{b\cdot a\cdot \tilde{l}!\nu\tilde{n}:\tilde{m}} N' \quad O \xrightarrow{a\cdot b\cdot \tilde{l}?\tilde{m}} O''}{N \mid O \xrightarrow{\tau} (\nu n)\,(N' \mid O'')}$$

Now it all follows identically to the previous case.

- $((\nu n)\,N\ \mathcal{R}\ (\nu n)\,M$ because $N \sim M$). Suppose that $(\nu n)\,N \xrightarrow{\xi} N'$. By cases on ξ and the lts rules we have 2 possibilities

  – Applying the rule (RES) for any ξ, $N'' = (\nu n)\,N'$ and $n \notin \mathtt{nams}(\xi)$ we have

  $$\frac{N \xrightarrow{\xi} N'}{(\nu n)\,N \xrightarrow{\xi} N''}$$

  By induction hypothesis we have that $M \xrightarrow{\xi} M'$ where $N'\mathcal{R}M'$. Applying the same rule (RES) we prove that $(\nu n)\,M \xrightarrow{\xi} (\nu n)\,M'$, and by definition of $\mathcal{R}$ we get that $(\nu n)\,N'\ \mathcal{R}\ (\nu n)\,M'$.

  – The second (and last) case is for rule (OPEN) with $\xi = a\cdot b\cdot \tilde{l}!\nu\tilde{n}' : \tilde{m}$ where $\tilde{n}' = \tilde{n} \cup \{n\}$ and $n \in \tilde{m}\backslash\tilde{n}$

  $$\frac{N \xrightarrow{a\cdot b\cdot \tilde{l}!\nu\tilde{n}:\tilde{m}} N'}{(\nu n)\,N \xrightarrow{a\cdot b\cdot \tilde{l}!\nu\tilde{n}':\tilde{m}} N'}$$

  This case concludes as in the previous ones.

□

We now prove that network bisimulation is a sound proof method for network reduction congruence, i.e. $\sim \subseteq \simeq$.

Theorem 5.3 (Soundness of Network Bisimulation). *Network barbed congruence includes network bisimulation.*

Proof. Soundness follows easily from the previous theorem. We have to show that network bisimulation is a network barbed congruence, i.e. it is a congruence, pointwise symmetric and reduction closed (by definition) and barb preserving. It only remains to show the latter. Suppose that $N \sim M$ and that $N \downarrow a\cdot b$. By definition of $\downarrow$ we have that either

- $N \equiv (\nu\tilde{n})\ a\{\ \phi :: b\cdot \tilde{l}\langle\tilde{m}\rangle\,.P + P' \mid Q\ \}_\alpha \mid N'$ and $\alpha \vdash \phi, b \notin \mathcal{P}(N')$

or

- $N \equiv (\nu\tilde{n})\ a\{\ p\cdot \tilde{l}(\tilde{x})\,.P + P' \mid Q\ \}_\alpha \mid N'$ and $b \notin \mathcal{P}(N')$.

But then, it follows that either $N \xrightarrow{a\cdot b\cdot \tilde{l}!\nu\tilde{n}:\tilde{m}} N'$ or $N \xrightarrow{a\cdot b\cdot \tilde{l}?\tilde{m}} N'$ and, as $N \sim M$, we also have that either $M \xrightarrow{a\cdot b\cdot \tilde{l}!\nu\tilde{n}:\tilde{m}} M'$ or $M \xrightarrow{a\cdot b\cdot \tilde{l}?\tilde{m}} M'$ with $N' \sim M'$. Now, by Lemma 5.3, it must be that either

- If $N \xrightarrow{a\cdot b\cdot \tilde{l}?\tilde{m}} N'$ then $N \equiv (\nu\tilde{n}')\ (a\{\ p\cdot\tilde{l}(\tilde{x})\,.P \mid Q\ \}_\alpha \mid M)$ for some P, Q, ϕ, M and such that $p \in \mathsf{Var}\cup\{\mathsf{b}\}$, $b \neq a$ and $\tilde{l}\cap\tilde{n}' = \tilde{n}'\cap\tilde{m} = \emptyset$;

or

- If $N \xrightarrow{a\cdot b\cdot \tilde{l}!\nu\tilde{n}:\tilde{m}} N'$ then $N \equiv (\nu\tilde{n}')\ (a\{\ \phi : b\cdot\tilde{l}\langle\tilde{m}\rangle\,.P \mid Q\ \}_\alpha \mid M)$ for some P, Q, ϕ and M such that $\alpha \vdash \phi$, $b \neq a$, $\tilde{n}\subseteq\tilde{n}'$ and $\tilde{n}\cap(\mathtt{fn}(M)\cup\mathtt{fn}(Q)) = \tilde{l}\cap\tilde{n}' = \emptyset$;

and again, by definition of $\downarrow$, we have that $M \downarrow a\cdot b$. □

We now move to study the opposite inclusion, namely that bisimulation is complete. In order to do so, we define *special contexts*, a special kind of contexts which are designed for exploring a network behaviour. Namely, when putting a network in a special context, we are able to tell whether the network is capable of doing certain action or not, depending on the number of reductions and what it reduces to. In the following, the values $\delta_{\mathtt{fail}}$ and $\delta^i_{\mathtt{ok}} \in \mathcal{P}$ are considered as special principal names, i.e. they never occur in any other part of the network. Moreover, with an abuse of notation, we write $i \in \tilde{m}\cap\tilde{n}$ for all those indices such that m_i is in $\tilde{n}$ and $i \in \tilde{m}\backslash\tilde{n}$ for all those indices such that m_i is not in $\tilde{n}$.

Definition 5.12 (Special Contexts). Let a and b be two principals, $\tilde{l}$ and $\tilde{m}$ two value vectors and N, M two networks. We define $C^{N,M}_\xi[\cdot]$ as,

- if $\xi = a\cdot b\cdot\tilde{l}?\tilde{m}$ then $C^{N,M}_\xi[\cdot] = \cdot \mid b\{\ a\cdot\tilde{l}\langle\tilde{m}\rangle\ \}_\emptyset$;
- if $\xi = a\cdot b\cdot\tilde{l}!\nu\tilde{n} : \tilde{m}$ then $C^{N,M}_\xi[\cdot] =$

$$\begin{aligned}\cdot \mid (\nu k_1, k_2)\ b\{\ & a\cdot\tilde{l}(\tilde{x})\,.(\Pi_{i\in\tilde{m}\cap\tilde{n}}\overline{b\cdot x_i\cdot k_1} \mid \Pi_{i\in\tilde{m}\backslash\tilde{n}}\overline{b\cdot x_i\cdot k_2}) \\ & \mid \Pi_{m\in\mathtt{fn}(N)\cup\mathtt{fn}(M)} b\cdot m\cdot k_1\,.\,\delta_{\mathtt{fail}} \\ & \mid \Pi_{m_i\in\tilde{m}\backslash\tilde{n}} b\cdot m_i\cdot k_2\,.\,\delta^i_{\mathtt{ok}} \\ & \}_\emptyset\end{aligned}$$

  for $\tilde{x}$ and $\tilde{m}$ with the same length;

We briefly explain the previous definition. In the first case, i.e. when ξ is an input, it easy to check that the network placed in the hole is capable of performing an input action, i.e. we shall only have to check that b does not have any barb. The output case is slightly more complicated, as we have to deal with extruded names. In fact, when b performs the input from a, we have to make sure that the free names received in $\tilde{x}$ are exactly what we are expecting, but also that all the others names sent over $\tilde{l}$, are not free names in N (and, as we shall see, also M). We now prove two properties for the contexts above.

Lemma 5.5. *Let $\xi = a\cdot b\cdot\tilde{l}?\tilde{m}$, M a network and suppose that $C^{N,M}_\xi[M] \to M' \mid b\{\ 0\ \}_\emptyset$ such that $C^{N,M}_\xi[M]$ is consistent. Then $M \xrightarrow{\xi} M'$.*

Proof. According to the way the context is structured, we need one reduction to get principal b in a state structurally congruent to $b\{\ 0\ \}_{\emptyset}$, i.e. the reduction must be a communication between a term from M and principal b. Formally, from Lemma 5.1, we have that

$$M \equiv (\nu\tilde{n}')\ (a\{\ p \cdot \tilde{l}(\tilde{x})\,.\,P + P' \mid Q\ \}_{\alpha} \mid O)$$

for $p \in \mathsf{Var} \cup \{\mathsf{b}\}$. It follows that $M \xrightarrow{\xi} M'$. □

Lemma 5.6. *Let $\xi = a \cdot b \cdot \tilde{l}!\nu\tilde{n} : \tilde{m}$, M, N two networks and suppose that, for $C_{\xi}^{N,M}[M]$ consistent, $C_{\xi}^{N,M}[M] \to \ldots \to M' \mid b\{\ \Pi_{i\in\tilde{m}\cap\tilde{n}}\overline{b \cdot n_i \cdot k_1} \mid \Pi_{m\in fn(N)\cup fn(M)} b \cdot m \cdot k_1\,.\,\delta_{fail} \mid \Pi_{m_i\in\tilde{m}\setminus\tilde{n}}\delta^{i}_{ok}\ \}_{\emptyset}$ where n_i are not in $fn(N) \cup fn(M)$ and the number of reduction is $|\tilde{m}\setminus\tilde{n}| + 1$, i.e. the number of names in $\tilde{m}$ which are also in $\tilde{n}$. Then $M \xrightarrow{\xi} M'$.*

Proof. The proof proceeds similarly to the one of previous lemma, but it is slightly more complicated because of the number of reductions. Suppose that $C_{\xi}^{N,M}[M] \to \ldots \to M_0$ with $|\tilde{m}\setminus\tilde{n}| + 1$ reductions, where

$$\begin{aligned} M_0 \equiv M' \mid (\nu k_1, k_2)\ b\{\ & \Pi_{i\in\tilde{m}\cap\tilde{n}}\overline{b \cdot n_i \cdot k_1} \\ & \mid \Pi_{m\in \mathtt{fn}(N)\cup\mathtt{fn}(M)} b \cdot m \cdot k_1\,.\,\delta_{\mathtt{fail}} \\ & \mid \Pi_{m_i\in\tilde{m}\setminus\tilde{n}}\delta^{i}_{\mathsf{ok}} \\ & \}_{\emptyset} \end{aligned}$$

This implies that we need to go through all possible matching tests and, this is the only possible way to do that. In fact, this configuration is only reachable when the vector $\tilde{m}$ is received from a and its bound names are only the ones in $\tilde{n}$. It follows that

$$M \equiv (\nu\tilde{n}')\ (a\{\ b \cdot \tilde{l}\langle\tilde{m}\rangle\,.\,P + P' \mid Q\ \}_{\alpha} \mid O)$$

and then $M \xrightarrow{\xi} M'$. □

We are now able to state and prove the completeness theorem, i.e. that network barbed congruence is included in network bisimulation.

Theorem 5.4 (Completeness of Network Bisimulation). *Reduction barbed congruence is included in network bisimulation.*

Proof. We prove that the relation $\mathcal{R} = \{(N, M) \mid N \simeq^{c} M\}$ is a bisimulation. The result will then follow by co-induction. Suppose that $N \xrightarrow{\xi} N'$ for $\mathtt{dest}(\xi) \cap \mathcal{P}(N \mid M) = \emptyset$ and consider the following cases:

- $\xi = a \cdot b \cdot \tilde{l}?\tilde{m}$.
  By definition of context $C_{\xi}^{N,M}[\cdot]$, it follows that $C_{\xi}^{N,M}[N] \xrightarrow{\tau} N_0 \equiv N' \mid b\{\ 0\ \}$ and by Lemma 5.4 we have that $C_{\xi}^{N,M}[N] \to N_0$. Moreover, as we are assuming that $N \simeq^{c} M$, by definition of reduction barbed congruence, we have that $C_{\xi}^{N,M}[N] \simeq^{c} C_{\xi}^{N,M}[M]$. This implies that there exists M_0 such that $C_{\xi}^{N,M}[M] \to M_0$ and $N_0 \simeq^{c}$

M_0. As, for all $c \in \mathcal{P}$, it holds that $N_0 \not\downarrow b \cdot c$, it follows that, given the way we defined the context, M_0 must have the following syntactical structure

$$M_0 \equiv M' \mid b\{\ 0\ \}$$

Now, as $\simeq^c$ is contextual, we can easily deduce that $N' \simeq^c M'$. By lemma 5.5, we conclude that also $M \xrightarrow{\xi} M'$.

- $\xi = a \cdot b \cdot \tilde{l}!\nu\tilde{n} : \tilde{m}$.
By definition of context $C^{N,M}_{\xi}[\cdot]$, it follows that $C^{N,M}_{\xi}[N] \underbrace{\xrightarrow{\tau} \ldots \xrightarrow{\tau}}_{|\tilde{m}\setminus\tilde{n}|+1 \text{ times}} N_0$ where

$$\begin{aligned} N_0 \equiv N' \mid (\nu k_1, k_2)\ b\{\ & \Pi_{i\in\tilde{m}\cap\tilde{n}}\overline{b \cdot n_i \cdot k_1} \\ & \mid \Pi_{m\in \mathtt{fn}(N)\cup\mathtt{fn}(M)} b \cdot m \cdot k_1 \,.\, \delta_{\mathtt{fail}} \\ & \mid \Pi_{m_i\in\tilde{m}\setminus\tilde{n}} \delta^i_{\mathrm{ok}} \\ & \}_{\emptyset} \end{aligned}$$

$N_0 \equiv N' \mid (\nu k_1, k_2)\ b\{\ \Pi_{i\in\tilde{m}\cap\tilde{n}}\overline{b \cdot n_i \cdot k_1} \mid \Pi_{m\in \mathtt{fn}(N)\cup\mathtt{fn}(M)} b \cdot m \cdot k_1 \,.\, \delta_{\mathtt{fail}} \mid \Pi_{m_i\in\tilde{m}\setminus\tilde{n}} \delta^i_{\mathrm{ok}}\ \}_{\emptyset}$ and, by the Harmony Lemma 5.4, we have that $C^{N,M}_{\xi}[N] \underbrace{\rightarrow \ldots \rightarrow}_{|\tilde{m}\setminus\tilde{n}|+1 \text{ times}} N_0$. Furthermore, we have that $N_0 \not\rightarrow$ such that principal b can change, as names n_i are not in $\mathtt{fn}(N) \cup \mathtt{fn}(M)$. Now, as we are assuming that $N \simeq^c M$, by definition of reduction barbed congruence, we have that $C^{N,M}_{\xi}[N] \simeq^c C^{N,M}_{\xi}[M]$. This implies that there exists M_0 such that $C^{N,M}_{\xi}[M] \underbrace{\rightarrow \ldots \rightarrow}_{|\tilde{m}\setminus\tilde{n}|+1 \text{ times}} M_0$ and $N_0 \simeq^c M_0$. At this point, as $N_0 \downarrow b \cdot \delta^i_{\mathrm{ok}}$ for all i such that $m_i \in \tilde{m}\setminus\tilde{n}$ and $N_0 \not\downarrow b \cdot \delta_{\mathtt{fail}}$, it follows that M_0 must have the following syntactical structure

$$\begin{aligned} M_0 \equiv M' \mid (\nu k_1, k_2)\ b\{\ & \Pi_{i\in\tilde{m}\cap\tilde{n}}\overline{b \cdot n_i \cdot k_1} \\ & \mid \Pi_{m\in \mathtt{fn}(N)\cup\mathtt{fn}(M)} b \cdot m \cdot k_1 \,.\, \delta_{\mathtt{fail}} \\ & \mid \Pi_{m_i\in\tilde{m}\setminus\tilde{n}} \delta^i_{\mathrm{ok}} \\ & \}_{\emptyset} \end{aligned}$$

Now, as $\simeq^c$ is contextual, we can easily deduce that $N' \simeq^c M'$. By lemma 5.6, we conclude that also $M \xrightarrow{\xi} M'$.

- $\xi = \tau$.
This case is trivial as the result follows just by using the harmony lemma 5.4.

□

5.3.3 Principal Bisimulation

We now move to study a proof method for principal equivalence, that we call *principal bisimulation*. We start by giving the definition, based on the labelled transition system seen above.

Definition 5.13 (Principal Bimulation). A pricipal bisimulation is a symmetric binary relation $\mathcal{S}_a$ on $\Pi \times P$ such that, for $(\alpha,P)\mathcal{S}_a(\beta,Q)$ and $\mathtt{sndr}(\mu) \neq a$, $(\alpha,P) \xrightarrow{\mu}_a (\alpha',P')$ implies that $(\beta,Q) \xrightarrow{\mu}_a (\beta',Q')$ and $(\alpha',P')\mathcal{S}(\beta',Q')$.
P and Q are principal bisimilar, written $P \sim_a^{\mathcal{P}} Q$, if they are related by principal bisimulation.

We now give some results about principal bisimulation. The next one, states that principal bisimulation is a congruence with respect to a-principal contexts.

Theorem 5.5. *For all a-principal contexts $C_a[\cdot_1,\cdot_2]$ if $(\alpha,P) \sim_a^{\mathcal{P}} (\beta,Q)$ then $C_a[\alpha,P] \sim C_a[\beta,Q]$*

Proof. We prove that $\{(C_a[\alpha,P],C_a[\beta,Q]) \mid (\alpha,P) \sim_a^{\mathcal{P}} (\beta,Q)\}$ is a network bisimulation. Suppose that $C_a[\alpha,P] \xrightarrow{\xi} N$ for $\mathtt{dest}(\xi) \cap \mathcal{P}(C_a[\alpha,P] \mid C_a[\beta,Q]) = \emptyset$. The proof proceeds by cases on the label α. We only show the output case, as the others are similar.

- If $\alpha = a \cdot b \cdot \tilde{l}!\nu\tilde{n} : \tilde{m}$ then, by Lemma 5.3 and the definition of principal context we have that $C_a[\alpha,P] \equiv (\nu\tilde{n}')\ (a\{\ \phi : b \cdot \tilde{l}\langle\tilde{m}\rangle\ .\ P'\ \}_\alpha \mid M)$ for some P', ϕ and M such that $\alpha \vdash \phi$, $b \neq a$, $\tilde{n} \subseteq \tilde{n}'$ and $\tilde{l} \cap \tilde{n}' = \emptyset$; it follows that $(\alpha,P) \xrightarrow{b\cdot\tilde{l}!\nu\tilde{n}:\tilde{m}}_a (\alpha,P')$ and then $(\beta,Q) \xrightarrow{b\cdot\tilde{l}!\nu\tilde{n}:\tilde{m}}_a (\beta,Q')$. By Rule (PRIN) and putting Q and β in the context we get $C_a[\beta,Q] \xrightarrow{\alpha} M$. It remains to prove that $N \sim M$, i.e. we only have to show that $a\{\ P'\ \}_\alpha \sim a\{\ Q'\ \}_\beta$. But this follows directly from the definition of network bisimulation and by the assumption that the pairs policy-protocol are principal bisimilar.

□

Exploiting the previous result, we are now able to prove that principal bisimulation is sound with respect to principal reduction equivalence.

Theorem 5.6. *Principal bisimulation $\sim_a^{\mathcal{P}}$ is included in principal reduction equivalence $\simeq_a^{\mathcal{P}}$.*

Proof. As done for network bisimulation, we have to show that principal bisimulation is a barbed principal equivalence. In order to do so, we must show that for all possible contexts $C_a[\cdot]$, $C_a[\alpha,P] \simeq C_a[\beta,Q]$ whenever $(\alpha,P) \sim_a^{\mathcal{P}} (\beta,Q)$. This follows from the previous theorem, and then what we want to prove will directly follow from the definition of principal barbed equivalence and Theorem 5.3.

□

As done for network bisimulation, we now show the opposite result, i.e. principal bisimulation is complete with respect to principal barbed equivalence.

Theorem 5.7. *Principal bisimulation $\sim_a^{\mathcal{P}}$ includes principal reduction equivalence $\simeq_a^{\mathcal{P}}$.*

Proof. We want to prove that the relation $\{((\alpha,P),(\beta,Q)) \mid (\alpha,P) \simeq_a^{\mathcal{P}} (\beta,Q)\}$ is a principal bisimulation. In order to show this we can exploit the previous results on network barbed bisimulation. Suppose that $(\alpha,P) \xrightarrow{\mu}_a (\alpha',P')$ and $(\alpha,P) \simeq_a^{\mathcal{P}} (\beta,Q)$. By definition of $\simeq_a^{\mathcal{P}}$, for any context $C_a[\cdot]$, it holds that $C_a[\alpha,P] \simeq C_a[\beta,Q]$ and from Theorem 5.4 also that $C_a[\alpha,P] \sim C_a[\beta,Q]$ and, in particular, we have that $a\{\ P\ \}_\alpha \sim a\{\ Q\ \}_\beta$. Applying rule (PRIN) from the lts for protocols, we have that $a\{\ P\ \}_\alpha \xrightarrow{a\star\mu} a\{\ P'\ \}_{\alpha'}$ and then also $a\{\ Q\ \}_\beta \xrightarrow{a\star\mu} a\{\ Q'\ \}_{\beta'}$. From this we have that $(\beta,Q) \xrightarrow{\mu}_a (\beta',Q')$. Moreover by induction hypothesis we have that $(\alpha',P') \sim_a^{\mathcal{P}} (\beta',Q')$ and this conlcudes the proof.

□

5.3.4 Protocol Bisimulation

We need to define something similar to principal bisimulation in order to characterise protocol reduction equivalence. But this can be done for free by exploiting what we have shown above about principal bisimulation.

We first define a different version of principal bisimulation, that we call *accurate principal bisimulation*. We basically need to consider also the internal actions that are not considered in principal bisimulation.

Definition 5.14 (Accurate Principal Bimulation). An accurate pricipal bisimulation is a symmetric binary relation $\mathcal{S}_a$ on $\Pi \times P$ such that $(\alpha,P)\mathcal{S}_a(\beta,Q)$ and $(\alpha,P) \xrightarrow{\mu}_a (\alpha',P')$ implies that $(\beta,Q) \xrightarrow{\mu}_a (\beta',Q')$ and $(\alpha',P')\mathcal{S}(\beta',Q')$ for some β' and Q'.
P and Q are accurate principal bisimilar, written $P \sim_a^{\mathcal{P}} Q$, if they are related by an accurate principal bisimulation.

Definition 5.15 (Protocol Bimulation). Given a principal a, we say that P and Q are a-protocol bisimilar, written $P \sim_a Q$, if for any α, $(\alpha,P) \sim_a^{\mathcal{P}} (\alpha,Q)$.

We report the first result on protocol bisimulation, i.e. it is a non-input congruence. This brings up to mind some similarities with early bisimulation for the π-calculus. We shall also see later that protocol bisimulation coincides with protocol reduction equivalence.

Theorem 5.8. *Protocol bisimulation is a non-input congruence.*

Proof. We need to show that there exists a relation which includes protocol bisimulation, which is a non-input congruence and is a bisimulation. The proof is similar to the one of Theorem 5.2.

□

We can now give the proofs of soundness and completeness for protocol bisimulation with respect to protocol reduction equivalence. We do not report the proofs, as they are very similar to the ones seen before for network bisimulation.

Theorem 5.9. *Protocol bisimulation is included in protocol reduction equivalence.*

Proof. It follows from previous theorems and by the definitions. □

Theorem 5.10. *Protocol reduction equivalence is included in protocol bisimulation.*

Proof. It follows from previous theorems and by the definitions. □

5.3.5 Weak network bisimulation

We conclude this section of characterisation of reduction equivalences, with weak network bisimulation. As stated before, when introducing weak network reduction congruence, sometimes it is important to ignore τ moves, i.e. reductions. In other words, we are interested in studying the behaviour of systems, but only from the outside, where we are not interested in internal moves, as we cannot observe what they are all about. We now give a weak network bisimulation which is sound and complete with respect to the weak reduction congruence we introduced in the previous section. Proofs in this section are omitted, as they follow the line of the ones for the strong case.

In the following, we write $N \stackrel{\mu}{\Longrightarrow} M$ whenever $N \Rightarrow N' \stackrel{\mu}{\longrightarrow} N'' \Rightarrow M$

Definition 5.16 (Weak Network Bimulation). A *weak network bisimulation* is a symmetric binary relation $\mathcal{S}$ such that $N\mathcal{S}M$ and $N \stackrel{\xi}{\longrightarrow} N'$, for $\mathtt{dest}(\xi) \cap \mathcal{P}(N \mid M) = \emptyset$, implies that $M \stackrel{\xi}{\Longrightarrow} M'$ for some M' and $N'\mathcal{S}M'$.
N and M are weak network bisimilar, written $N \approx M$, if they are related by network bisimulation.

We then get the same results we got for the strong case.

Theorem 5.11. *Weak network bisimulation is a congruence.*

Proof. Similar to the strong case.

□

Theorem 5.12 (Soundness). *Weak network reduction congruence includes weak network bisimulation.*

Proof. Similar to the strong case. □

Theorem 5.13 (Completeness). *Weak network reduction congruence is included in weak network bisimulation.*

Proof. Similar to the strong case. □

5.3.6 Summary of studied relations

In table 5.4 we report all the equivalences we have studied in this chapter, also giving the various relationships (if any) that there are among them.

5.3.7 Proof Examples

Below we report some examples of applications of the proof method we have introduced. Namely, we prove some of the examples and theorems seen before.

		$\simeq$	$\sim$	$\simeq_a$	$\sim_a$	$\cong$	$\approx$	$\simeq_a^{\mathcal{P}}$	$\sim_a^{\mathcal{P}}$
Network Reduction Congr.	$\simeq$	=	=	x	x	$\subset$	$\subset$	x	x
Network Bisimulation	$\sim$	=	=	x	x	$\subset$	$\subset$	x	x
Protocol Reduction Equiv.	$\simeq_a$	x	x	=	=	x	x	x	x
Protocol Bisimulation	$\sim_a$	x	x	=	=	x	x	x	x
Weak Network Reduction Congr.	$\cong$	$\supset$	$\supset$	x	x	=	=	x	x
Weak Network Bisimulation	$\approx$	$\supset$	$\supset$	x	x	=	=	x	x
Principal Reduction Equiv.	$\simeq_a^{\mathcal{P}}$	x	x	x	x	x	x	=	=
Principal Bisimulation	$\sim_a^{\mathcal{P}}$	x	x	x	x	x	x	=	=

Table 5.4: Equivalences for `ctm`

Proof Sketch of Example 5.3

In here, we want to prove that $(\alpha, P(1) \mid P(2)) \simeq_a^{\mathcal{P}} (\alpha, Q \mid P(2))$. Exploiting the theory that we have developed in the previous section, we just need to prove that $(\alpha, P(1) \mid P(2)) \sim_a^{\mathcal{P}} (\alpha, Q \mid P(2))$. We remind the reader that

$$\begin{aligned} P(z) = &(\nu n)\ (a\cdot n() \mid\ !a\cdot n\langle\rangle\,.\,x\cdot z(y)\,.\,Access(z,x) :: x\cdot z\langle OK\rangle\,.\\ &(x\cdot z\cdot col()\,.\,Col(z,x) :: x\cdot z\cdot col\langle OK\rangle\,.\,a\cdot n() +\\ &x\cdot z\cdot bw()\,.\,BW(z,x) :: x\cdot z\cdot bw\langle OK\rangle\,.\,a\cdot n())) \end{aligned}$$

and

$$\begin{aligned} Q = &(\nu n)\ (a\cdot n() \mid\ !a\cdot n\langle\rangle\,.\,x\cdot 1(y)\,.\,Access(1,x) :: x\cdot 1\langle OK\rangle\,.\\ &(x\cdot 1\cdot col()\,.\,x\cdot 1\cdot col\langle OK\rangle\,.\,a\cdot n() \ +\ x\cdot 1\cdot bw()\,.\,x\cdot 1\cdot bw\langle OK\rangle\,.\,a\cdot n())) \end{aligned}$$

Looking at processes, we basically have to prove that the actions that $P(1)$ can perform match the ones performed by Q (and vice versa). It seems clear that, given the syntactical structure, the protocols are exactly the same until, for all principals x, they get to the following configurations (where the policy α will become β)

$$\begin{aligned} &(\nu n)\ (x\cdot 1\cdot col\,.\,Col(1,x) :: x\cdot 1\cdot col\langle OK\rangle\,.\,a\cdot n\\ &\qquad + x\cdot 1\cdot bw\,.\,BW(1,x) :: x\cdot 1\cdot bw\langle OK\rangle\,.\,a\cdot n) \end{aligned}$$

and

$$(\nu n)\ (x\cdot 1\cdot col()\,.\,x\cdot 1\cdot col\langle OK\rangle\,.\,a\cdot n() \ +\ x\cdot 1\cdot bw()\,.\,x\cdot 1\cdot bw\langle OK\rangle\,.\,a\cdot n)$$

This all amounts to prove that if $(\beta, (\nu n)\ BW(1,x) :: x\cdot 1\cdot bw\langle OK\rangle\,.\,a\cdot n) \xrightarrow{x\cdot 1\cdot col!\nu\emptyset:OK}_a (\alpha'', a\cdot n)$ then $(\beta, x\cdot 1\cdot bw\langle OK\rangle\,.\,a\cdot n) \xrightarrow{x\cdot 1\cdot col!\nu\emptyset:OK}_a (\alpha'', a\cdot n)$ and vice versa. But this follows easily from the way we defined the starting policy α.

Proof Sketch of Example 5.4

As in the previous example, we give a hint of the proof avoiding going into details. We want to prove that the two networks N_1 and N_2 are weak network reduction congruent. We show this by proving that $N_1 \approx N_2$. We recall the syntax of the two networks

$$N_1 = B_F\{\ P_1(B_F, B_I)\ \}_{(\pi_1(B_I),\emptyset)} \mid B_I\{\ P_1(B_I, B_F)\ \}_{(\pi_1(B_F),\emptyset)}$$

where

$$\begin{aligned} P_1(X,Y) = {} & !x\cdot \mathtt{mg}(w)\,.\,(\nu k)\ (Y\cdot \mathtt{rec}\langle k,x\rangle\,.\,Y\cdot k(x,z)\,.\,\mathtt{G}(x) :: X\cdot \mathtt{gr}\langle x,w\rangle) \\ & \mid !X\cdot \mathtt{gr}(x,w)\,.\,x\cdot w\langle\rangle\,.\,x\cdot w() \\ & \mid !Y\cdot \mathtt{rec}(k,x)\,.(\mathtt{Good}(x) :: Y\cdot k\langle x,\mathtt{Good}\rangle + \mathtt{Bad}(x) :: Y\cdot k\langle x,\mathtt{Bad}\rangle) \end{aligned}$$

and

$$N_2 = O\{\ !(F(B_F)+F(B_I))\ \}_{(\pi_2(B_F,B_I)\cup\pi_2(B_I,B_F),\emptyset)} \mid B_F\{\ P_2\ \}_{(\emptyset,\emptyset)} \mid B_I\{\ P_2\ \}_{(\emptyset,\emptyset)}$$

where

$$F(W) = W(x,w)\,.\,\mathtt{G}(W,x) :: O\cdot W\cdot \mathtt{gr}\langle x,w\rangle \mid !O\cdot W\cdot \mathtt{gr}(x,w)\,.\,W\cdot w\langle\rangle\,.\,W\cdot w()$$

Using the weak network bisimulation, it becomes easy to prove the equivalence, as we only need to show that the two networks "weakly" perform the same actions. In fact, principal O in network N_2 is completely hidden to any context as it never interacts with any principal but the two banks. In other words, the two networks are equal for any external context.

5.4 On the expressive power of ctm

In this section we shall go trough some considerations on the expressivity of ctm. Our calculus features new operations, some of them new to mobility. We want to show that such operations, according to the criteria proposed in chapter 2, add expressive power to the calculus.

On the "asynchronous input". The input operator in `ctm` has not the same behaviour as in π-calculus. This could also influence the whole semantics of a pricipal, i.e. its behaviour. Looking at the problem more in details, we observe that every time an input is performed, either by interacting with another principal or by doing an internal communication, the policy of the principal is updated. In π-calculus the input operator has a continuation where the value received might influence the future computations of the whole process. Suppose now that we fix the input operator not to have any continuation, i.e. we only instantiate the input as $p\cdot\tilde{l}(\tilde{x})\,.\,0$ (we shall call it asynchronous because of the analogy with the asynchronous output). What we really want to stress out is the fact that, just performing an input with another principal without passing the inputted vector to a continuation might influence the whole principal parallel components, instead in a π-calculus-like setting, that thing does not happen.

Guarded input `ctm`. A second point about the expressivity of `ctm` that we want to treat is about guards. So far we have not fully justified the fact that only inputs are guarded. Our approach is that we see performing an output as granting another principal with something which can be seen with the message. Another possible way of seeing things is by guarding inputs, i.e. having something like $\phi :: p \cdot \tilde{l}(\tilde{x}) \,.\, P$. In this subsection we show that this can indeed be done in `ctm` with a very simple encoding, just by paying off few internal moves (τ's).

We can now give semantics to the new input operation. In other words we have to replace the rule (IN) in the lts with the following:

$$(\textsc{Guarded-In})\ \frac{\alpha' = \mathsf{upd}(\alpha, b \cdot \tilde{l} \rhd \tilde{m}) \quad b : \tilde{m} \odot p : \tilde{x} = \sigma \quad \alpha \vdash \phi}{(\alpha, \phi :: p \cdot \tilde{l}(\tilde{x}) \,.\, P) \xrightarrow{b \cdot \tilde{l} ? \tilde{m}}_a (\alpha', P\sigma)}$$

We can then see that we can get a similar semantics (modulo internal τ moves) by just giving the following encoding

Definition 5.17 (Encoding of guarded input). The encoding is an isomorphism with the only exception for the production (INPUT)

$$[\![\phi :: p \cdot \tilde{l}(\tilde{x}) \,.\, P]\!] \triangleq (\nu n)\ (\phi :: a \cdot n\langle\rangle \mid a \cdot n() \,.\, p \cdot \tilde{l}(\tilde{x}) \,.\, P)$$

It could also be argued that in the moment that the input is really performed it could be that the ϕ is no longer satisfied. This is true indeed, but we suppose that once that someone is been granted with something, i.e. a decision has been taken, there is no point of going back and so every decision must be definitive.

Global input. Our calculus uses a new input construct: global input. In this section we prove that such a construct adds expressiveness to the language. Let $\mathtt{ctm}^{-\phi}$ be the fragment of `ctm` where all inputs are guarded by **tt** and let $\mathtt{ctm}^{-x;\phi}$ be the fragment of $\mathtt{ctm}^{-\phi}$ without global input. Moreover let S be a list of observations $a_1 \cdot b_1; \ldots a_k \cdot b_k; \ldots$ and $N \Downarrow S$ if and only if $N \rightarrow^* N_1 \rightarrow^* \ldots N_k \rightarrow^*$ and $N_1 \downarrow a_1 \cdot b_1, \ldots, N_k \downarrow a_k \cdot b_k, \ldots$ In the following, with abuse of notation, we will use $[\![-]\!]$ for both networks and protocols.

Definition 5.18. An encoding $[\![-]\!] : \mathtt{ctm}^{-\phi} \longrightarrow \mathtt{ctm}^{-x;\phi}$ is *correct* whenever for all protocols P, Q and networks N, M

- $[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!]$
- $[\![N \mid M]\!] = [\![N]\!] \mid [\![M]\!]$
- $[\![a\{\ P\ \}_\alpha]\!] = a\{\ [\![P]\!]\ \}_\alpha$
- for any N, $a\{\ P\ \}_\alpha \mid N \Downarrow S$ if and only if $[\![a\{\ P\ \}_\alpha \mid N]\!] \Downarrow S$

The first three rules represent the notion of uniform encoding, while the last one corresponds to the notion of reasonable encoding we saw in the preliminaries chapter.

Theorem 5.14. *There is no correct encoding* $[\![-]\!]$ *from* $\mathtt{ctm}_P^{-\phi}$ *into* $\mathtt{ctm}_P^{-x;\phi}$.

Proof. [Sketch] Suppose there exists such an encoding and consider $a\{\ x \cdot l(y)\ \}_\alpha$. Principal a is such that $a\{\ x \cdot l(y)\ \}_\alpha \downarrow a \cdot b$ for any b. Now we have that $a\{\ [\![x \cdot l(y)]\!]\ \}_\alpha \not\Downarrow a \cdot b$ for all b. In fact we can prove by induction on the protocols of $\mathtt{ctm}_P^{\neg x:\phi}$ that such a protocol does not exists. □

5.5 Related Work

To the best of our knowledge, the notion of trust has never been fully treated in process calculi. In Dπ [6, 60] policies are statically specified, not allowing dynamic updates; [57] considers a formalism for cryptographic protocols, similar to ours: communications are guarded by logical formulas meant for proving correctness, whereas protocols are expressed with strand-spaces. Concerning policies for access control, there are many works on logics, where a trust engine is responsible for constructing [24, 65, 69] or checking [7] a proof that a desired request is valid. In [75] and [11] authors provide a decidable logic for policies, proposing variants of Datalog. In particular, *Cassandra*, provides a formalism for expressing policies in a GC scenario, where, as in our case, each principal has its own policy. They also allow references to other principals' polices and delegation, using fixed-point computations as in [33].

Part II

On Polyadic Synchronisation

Chapter 6

Polyadic Synchronisation and its expressive power

6.1 Introduction

In the previous chapters, we stressed the fact that process calculi provide a useful framework in which to reason about the theory of concurrent and distributed systems. They are praised both for great simplicity and expressiveness. The π-calculus is a terse and powerful language which describes the behaviour of concurrent systems, and is endowed with a rich body of theoretical results. However, evidence has been accumulated which suggests that it is inadequate to express certain aspects of distributed systems. In fact, the literature is rich with extensions of the π-calculus explicitly designed for modeling such systems.

Following *Occam's razor* principle[1], we study a minimal extension of the π-calculus in which to express concepts relevant to distributed systems such as *localities* and *encryption*. We propose ${}^{e}\pi$: the π-calculus with *polyadic synchronisation*, a generalisation of the synchronisation mechanism which allows channel names to be composite. The idea is that the subject of an input or output action is no longer restricted to be a single name, but is now a vector of names. For example we allow prefixes such as $a \cdot b(x)$ and $\overline{a \cdot b}\langle v \rangle$, where the channel is identified by vector $a \cdot b$. It turns out that this construct cannot be encoded in π-calculus without introducing divergence.

6.1.1 Examples of polyadic synchronisation

To justify our proposal, we introduce concrete scenarios and theoretical issues where polyadic synchronisation turns out to be helpful.

[1] Occam's razor is a logical principle attributed to the medieval philosopher William of Occam (or Ockham). The principle states that one should not make more assumptions than the minimum needed. This principle is often called the principle of parsimony. It underlies all scientific modeling and theory building. It admonishes us to choose from a set of otherwise equivalent models of a given phenomenon the simplest one. In any given model, Occam's razor helps us to "shave off" those concepts, variables or constructs that are not really needed to explain the phenomenon. By doing that, developing the model will become much easier, and there is less chance of introducing inconsistencies, ambiguities and redundancies.(F. Heylighen).

Two practical problems

The first example is EDπ [29], where we use a construct which represents atomic transactions as a means to model e-services. Interaction between a client and a server takes place only if both parties agree on a set of service parameters. The second example is the calculus of objects (CO) of [113], upon which the TyCO programming language is based [112]. It models distributed object systems where messages are dispatched to a certain object if and only if it provides the method invoked in the request, and it is ready to execute it. Its semantics requires agreement on both object identity and method name, in order for a call to be dispatched.

Both these examples can be generalised as instances of the problem of matching *atomically* vectors of values among different processes (the *matching problem*). The solution consists in bringing the values to be matched directly in the *interface* of each process towards the system, and providing semantic rules that allow interaction if and only if those interfaces are compatible.

Modeling locations

The ability of synchronising on many names at the same time, allows for *localities* to be represented in ${}^{e}\pi$. Many distributed calculi refer to an explicit notion of location, intended as a unit of distribution where computation takes place. Some of them are presented as extensions of the π-calculus, and are based on the idea that processes running in parallel inside some location can independently migrate or communicate with other processes, locally or remotely. Examples of languages of such kind are to be found in [59], [6], [35], [106], and [48]. From a pragmatic point of view, it emerged clearly that these models were to some extent more appropriate than the π-calculus to describe physical distribution. From a theoretical point of view, the *necessity* of these variants has not been fully explored. Take the Distributed π-calculus of [59] (where π-like processes are explicitly enclosed in locations), as a paradigmatic example. Its main reduction rule states that communication among two processes can take place only when they are in the same location:

$$(\text{RCOMM}) \quad l[a\langle v\rangle.P] \mid l[a(x).Q] \;\mapsto\; l[P] \mid l[Q\{v/x\}]$$

Our point is that a location can be seen as a name characterising all the interactions in which a process participates: hence it can be modeled as an additional synchronisation parameter in all the communications of a located process. Migration is simply the dynamic (re)binding of the location component of each prefix. For example the result of encoding the Dπ network

$$l[\overline{a}\langle m\rangle.P \mid a(x).\mathtt{go}\ x.\overline{b}\langle v\rangle.Q] \mid m[b(y).R]$$

is a process in this extension of π-calculus (${}^{e}\pi$), where the migration construct $\mathtt{go}\ x$ disappears and the three threads of execution are run in parallel:

$$\overline{l\cdot a}\langle m\rangle.P \mid l\cdot a(x).\overline{x\cdot b}\langle v\rangle.Q \mid m\cdot b(y).R.$$

Note that rule (RCOMM) reported above, stating that two processes are allowed to react if and only if they share two values (location and channel) at the same time, is

another instance of the matching problem. Another example of how localities can be expressed in terms of polyadic synchronisation is in [30], where we give a divergence-free encoding of the Local Area π-calculus of [38] in ${}^{e}\pi$.

Partial restriction and matching

Polyadic synchronisation also enhances the π-calculus in that it allows for *partial restriction*; that is, it gives the ability to restrict only some of the names taking part in a communication. It turns out that thanks to this feature, matching can be expressed as a special form of communication, and therefore is not given as primitive in ${}^{e}\pi$. Partial restriction allows us also to model cryptographic protocols, as explained below.

Modeling cryptography

We claim that ${}^{e}\pi$ can express an interesting class of security protocols. In fact, it is possible to use polyadic synchronisation to encode *secure channels*: the sending along public channel a of datum m encrypted under key k is expressed as $\overline{a \cdot k}\langle m \rangle.P$, implying that m can be received only by an agent knowing the secret password k, beside the public name a. With respect to the Spi-calculus of [2], this first solution lacks for example the power of expressing keys obtained by hashing data. Consider now a different, more expressive way to represent encryption in ${}^{e}\pi$. We propose constructs for encrypting and decrypting data, such that encrypted messages are represented as *names* (therefore can still be encrypted, sent, or used as keys), and encryption is nondeterministic (encrypting the same message under the same key two times yields different results). These constructs are:

$$\begin{aligned} [\![\texttt{encrypt}\ m \overset{\varsigma}{\mapsto}_{k} x\ \texttt{in}\ P]\!] &= (\nu x)\ (!\overline{x \cdot k}\langle m \rangle \mid P) \\ [\![\texttt{decrypt}\ x \overset{\varsigma}{\mapsto}_{k} m\ \texttt{in}\ P]\!] &= x \cdot k(m).P \end{aligned}$$

The first construct encrypts data m under key k and returns the encrypted message as the fresh name x, to be used in all the scope embraced by P. Decryption of message x through key k binds name m in the continuation P to the original message provided that the key is the same as the one used to encrypt it. Note how x, the result of the encryption, is a restricted name whose scope is process P. This is not a limitation because the scope of x can be extended in the standard way using extrusion, allowing modularity in the definition of processes. For example, the proces *System* below evolves to a state where R and S share the classified data m, and that A cannot compromise the security of the protocol.

$$\begin{aligned} \textit{Receiver}: &\quad (\nu\ k)\ \overline{\textit{secure}}\langle k \rangle.\textit{public}(y).\texttt{decrypt}\ y \overset{\varsigma}{\mapsto}_{k} w\ \texttt{in}\ R \\ \textit{Sender}: &\quad (\nu\ m)\ \textit{secure}(z).\texttt{encrypt}\ m \overset{\varsigma}{\mapsto}_{z} x\ \texttt{in}\ \overline{\textit{public}}\langle x \rangle.S \\ \textit{System}: &\quad (\nu\ \textit{secure})\ (\textit{Sender} \mid \textit{Receiver}) \mid A \end{aligned}$$

In this chapter we stress the fact that the notion of *sensible encoding* as a subset of the requirements that the encoding of an operator in a language should satisfy in order to be considered meaningful.

We introduce ${}^e\pi$ in Section 6.2, and in Section 6.2.2 we show how a restricted form of polyadic synchronisation can be encoded in π-calculus, at the price of introducing divergence.

Section 6.3 starts with the answer to an open question: we show that matching enhances the expressive power of the π-calculus (Although its usefulness has been challenged in the literature, for example by [79].). In ${}^e\pi$, matching can be encoded up-to strong bisimulation congruence. Also mixed choice is known to increase the expressiveness of π-calculus: we extend the results of [92] to ${}^e\pi$, showing that polyadic synchronisation does not have the power to encode mixed choice. In Section 6.3.3 we show that polyadic synchronisation cannot be encoded in the full π-calculus (and therefore it is orthogonal to mixed choice) without introducing divergence, and we generalise the result to higher degrees of synchronisation.

In Section 6.4 we show how the π-calculi that we have considered are partially ordered by expressivity, and constitute a complete lattice. The chapter concludes with a suggestion for typing polyadic synchronisation.

6.1.2 Previous research related to polyadic synchronisation

In [47], they extend CCS with composite prefixes in order to model transactions, using a mixed form of polyadic synchronisation and synchronisation between multiple parties. Boudol and Castellani [21] have studied the impact of considering finite computations as atomic steps on concurrent languages semantics, yet it seems that polyadic synchronisation is not expressible in their framework. Nestmann [86] has studied the expressive power of the *joint input*, a liberalisation of the *join patterns* of [49], in the π-calculus framework: it can be seen as a form of *bi-adic* synchronisation for input processes only. The main reduction rule for joint input is

$$\overline{a}\langle c\rangle \mid \overline{b}\langle d\rangle \mid \{a(x)|b(y)\}.P \searrow P\{c/x\}\{d/y\}$$

where $\{a(x)|b(y)\}.P$ could be seen as similar to the ${}^e\pi$ process $a\cdot\ b(x,y).P$. As this example shows, there is a fundamental difference in the way of expressing outputs in the two languages: where the joint input requires multi-way synchronisation, ${}^e\pi$ mantains the interaction confined to two processes. For this reason, joint input does not provide a solution to the matching problem.

Appendix A of [2] mentions synchronisation on tuples to point out that π-calculus could be made more resistant to security attacks, but the subject is not developed any further. In [80], they considered a form of multi-way synchronisation, superseding both polyadic synchronisation and joint input, that raised many interesting but non trivial questions on the theory. Our independent development is strongly biased towards enhancing the expressive power of π-calculus without introducing significant changes in the underlying equational theory.

6.2 Polyadic Synchronisation in π-calculus

Our proposal is to extend the synchronisation mechanism of π-calculus to the case where channels are denoted by *vectors of names*, allowing interaction to happen only

when such vectors match element-wise. Synchronisation remains atomic: we enforce an *all-or-nothing* behaviour. A typical reduction might look like

$$x \cdot y(z).P | \overline{x \cdot y}\langle w \rangle.Q \xrightarrow{\tau} P\{w/z\} | Q$$

6.2.1 Syntax and semantics of ${}^e\pi$

To define ${}^e\pi$ we need to generalise the syntax for prefixes given in precedence to the one given below, where k and j are any two natural numbers.

$$(\text{PREFIXES})\ \alpha ::= \ \tau \mid \ x_1 \cdot \ldots \cdot x_k(y) \ \mid \ \overline{x_1 \cdot \ldots \cdot x_j}\langle y \rangle$$

A channel is now a vector of names, the *synchronisation vector*: π-calculus is the instance where only vectors of length one are allowed. Synchronisation vectors will be denoted by letters u and v. The syntax of processes is the same as the one for π^m, and all the definitions given in Chapter 2 are straightforwardly adapted to the case where a vector substitutes a single name in the subjects of actions. As an example, we report two specific reduction rules where vector u has replaced name x.

$$\frac{P \xrightarrow{\overline{u}\langle \nu y \rangle} P',\ Q \xrightarrow{u(y)} Q'}{P \mid Q \xrightarrow{\tau} (\nu y)\ (P' \mid Q')}\ (\text{CLOSE}) \qquad \frac{P \xrightarrow{\overline{u}\langle y \rangle} P'}{(\nu y)\ P \xrightarrow{\overline{u}\langle \nu y \rangle} P'} y \notin u\ (\text{OPEN})$$

The only exception regards matching: as we will show in Lemma 6.8, matching can be derived in ${}^e\pi$ and therefore we exclude both the syntactic production and the semantic rule (MATCH) from the definition of ${}^e\pi$. As a consequence, Observation 6.2 holds also for ${}^e\pi$.

It is worth noting that since restriction is defined on names rather than on channels as a whole, process $P \triangleq x \cdot y(z).Q$ is such that $P \downarrow_{x \cdot y}$, but $R \triangleq (\nu x)x \cdot\ y(z).Q$ is such that $R \not\downarrow$, even if $y \in fn(R)$.

The $\pi_{\mathbb{N}}$ family

We denote with π_k the sub-language where the length of synchronisation vectors is at most k. A degenerate case is π_0, where channels are nameless and processes interact through a global ether (It is essentially the local communication mechanism of the Ambient Calculus of [35].): $\langle y \rangle.P \mid (x).Q \longrightarrow P \mid Q\{y/x\}$. The sub-calculi defined in Chapter 2 can be analogously extended to polyadic synchronisation. In particular, π^m is π_1 where prefixes with vectors of length 0 are ruled out, and ${}^e\pi = \bigcup_n \pi_n$. The family of calculi $\{a\pi_0, \pi_0, a\pi_1, \pi_1, a\pi_2, \pi_2, \ldots\}$ will be denoted by $\pi_{\mathbb{N}}$ (see also Table 6.1).

6.2.2 Encoding polyadic synchronisation in π-calculus

It is possible to define a strongly uniform encoding of bi-adic synchronisation in polyadic $a\pi^{=,\neq}$, where we use the conditional construct $[a = b]P, Q$ as an abbreviation for $[a = b]P | [a \neq b]Q$. By transitivity, the same idea can be used to encode higher degrees of synchronisation. The encoding is a homomorphism, except for the following cases ($w, x, z \notin fn(P)$):

Variants of π-calculus:

	$\overline{a}.P$	$=$	$\neq$	$+_s$	$+_m$
$a\pi$	–	–	–	–	–
π^s	✓	–	–	✓	–
π^m	✓	–	–	–	✓
$a\pi^=$	–	✓	–	–	–
$a\pi^{=,\neq}$	–	✓	✓	–	–
$\pi^{s,=}$	✓	✓	–	✓	–
π	✓	✓	–	–	✓

The $\pi_{\mathbb{N}}$ family:

	$\overline{a}.P$	$+_m$	$a_1 \cdot \ldots \cdot a_n$
$a\pi_0$	–	–	0
$a\pi_n$	–	–	$\leq n$
π_0	✓	✓	0
π_n	✓	✓	$\leq n$
${}^e\pi$	✓	✓	$< \omega$

Table 6.1: Some variants of the π-calculus.

$$[\![\overline{b\cdot a}\langle c\rangle]\!] \triangleq (\nu z)(\overline{b}\langle a,c,z\rangle \,|\, \overline{z})$$
$$[\![b\cdot a(y).P]\!] \triangleq (\nu w)(\overline{w} \mid\, !w.b(x,y,z).[x=a](z\,|\,[\![P]\!]),(\overline{b}\langle x,y,z\rangle \,|\, \overline{w}))$$

In the encoding of the input we simulate the prefix $b\cdot a(y).P$ in two steps: an input on b and a matching on a, introducing the need to backtrack in case of failure. The parameter z added to the communication distinguishes the behaviour of the two branches of the conditional, and is needed to preserve the soundness of the translation. Note that if matching fails, the original state is restored, introducing the possibility of divergence.

$$[\![\overline{b\cdot a}\langle c\rangle]\!] \triangleq \overline{b}\langle a,c,\nu z\rangle \qquad [\![\overline{b\cdot a}\langle \nu c\rangle]\!] \triangleq \overline{b}\langle a,\nu c,\nu z\rangle$$
$$[\![b\cdot a(y)]\!] \triangleq b(x,y,z) \qquad [\![\tau]\!] \triangleq \tau$$

Table 6.2: Translation of actions

In Table 6.2 we define the correspondence among the actions of the source terms and those of the target terms. Note that the correspondence in the observables is not very strong, in particular the translation of an input action replaces a free name with a bound name (in $[\![b\cdot a(y)]\!] \triangleq b(x,y,z)$, x replaces a). We look now at the properties of the encoding.

Proposition 6.1. *The encoding of $a\pi_2$ in $a\pi^{=,\neq}$ is strongly uniform.*

Proof. The encoding is homomorphic with respect to parallel composition. By noting that the encoding preserves free names, and by a straightforward induction, follows that for all substitutions σ, $[\![P]\!]\sigma = [\![P\sigma]\!]$. □

The encoding is sound with respect to $\dot{\approx}$, whereas it is not complete, as it can be seen from the following example.

Example 6.1. Let $P \triangleq (\nu a)(\overline{b\cdot a})$ and $Q \triangleq (\nu a)(\overline{c\cdot a})$. We have that $P \dot{\approx} 0 \dot{\approx} Q$ whereas $[\![P]\!] \not\dot{\approx}_\pi [\![Q]\!]$ since $[\![P]\!] \downarrow_b$ and $[\![P]\!] \not\downarrow_c$, but $[\![Q]\!] \downarrow_c$ and $[\![Q]\!] \not\downarrow_b$.

Theorem 6.1. *For all processes P and Q in $a\pi_2$:*

1. *If $P \xrightarrow{\tau} Q$ then $[\![P]\!] \xrightarrow{\tau}\xrightarrow{\tau}\xrightarrow{\tau}\sim [\![Q]\!]$.*
2. *If $[\![P]\!] \longrightarrow Q'$ then there exists Q such that $P \longrightarrow Q$ and $Q' \approx [\![Q]\!]$.*
3. *If $[\![P]\!] \approx [\![Q]\!]$ then $P \approx Q$.*

The existence of the encoding shows that *it is* possible, to some extent, to achieve the effects of polyadic synchronisation in π, as long as one is not concerned with termination properties.

Proof.

In the following we will distinguish between a relation (e.g. structural congruence) in $a\pi^{=,\neq}$ and the corresponding notion in $a\pi_2$, by labeling the former with the symbol π (e.g. $\equiv_\pi$). We report below the lemmata used in the proof of Theorem 6.1.

We start with a simple property based on the definition of barbs.

Observation 6.1. For any process P in $a\pi_2 \backslash a\pi_1$:

1. if $P \xrightarrow{\overline{b\cdot a}\langle c\rangle} Q$ then $P \equiv \overline{b\cdot a}\langle c\rangle \,|\, Q$;
2. if $P \xrightarrow{\overline{b\cdot a}\langle \nu c\rangle} Q$ then $P \equiv (\nu c)(\overline{b\cdot a}\langle c\rangle \,|\, Q)$;
3. if $P \xrightarrow{b\cdot a(x)} Q$ then for some P_1, P_2, and some $\tilde{n}$ such that $a, b \notin \tilde{n}$, $P \equiv (\nu\tilde{n})(b\cdot a(x).P_1 \,|\, P_2)$ and $Q \equiv (\nu\tilde{n})(P_1 \,|\, P_2)$.

Proof. Follow directly adapting Definition 2.3 to ${}^e\pi$. □

We now establish a strong operational correspondence between the actions of a term and the actions of its encoding.

Lemma 6.1. *For any process P in $a\pi_2 \backslash a\pi_1$, if $P \xrightarrow{\mu} Q$ then:*

1. *if $\mu \in \{\overline{b\cdot a}\langle c\rangle, \overline{b\cdot a}\langle \nu c\rangle\}$ then, for any $z \notin fn(P)$, $[\![P]\!] \xrightarrow{[\![\mu]\!]}_\pi \overline{z}\,|\,[\![Q]\!]$;*
2. *if $\mu = b\cdot a(y)$ then $\exists Q'.[\![P]\!] \xrightarrow{\tau}_\pi \xrightarrow{b(x,y,z)}_\pi Q'$ and for any $\sigma : \{x,y,z\} \to \mathcal{N}$,*
   - *if $\sigma(x) = a$, $Q'\sigma \sim_\pi z\sigma | [\![Q]\!]\sigma$;*
   - *if $\sigma(x) \neq a$, $Q'\sigma \sim_\pi \overline{b}\langle x,y,z\rangle\sigma | [\![P]\!]\sigma$;*
3. *if $\mu = \tau$ then $[\![P]\!] \xrightarrow{\tau}_\pi \xrightarrow{\tau}_\pi \xrightarrow{\tau}_\pi \sim_\pi [\![Q]\!]$.*

Proof. 1. If $\mu = \overline{b\cdot a}\langle c\rangle$, by Observation 6.1 we have that $P \equiv \overline{b\cdot a}\langle c\rangle \,|\, Q \xrightarrow{\overline{b\cdot a}\langle c\rangle} Q$. By definition of the encoding and lts, follows

$$[\![P]\!] \equiv_\pi (\nu z)(\overline{b}\langle a,c,z\rangle \,|\, \overline{z}) \,|\, [\![Q]\!] \xrightarrow{\overline{b}\langle a,c,\nu z\rangle}_\pi \overline{z}\,|\,[\![Q]\!]$$

Note that using alpha conversion on z before applying the (OPEN) rule, it is possible to derive an analogous transition for any $z' \notin fn(P)$.

The case for $\mu = \overline{b\cdot a}\langle \nu y\rangle$ is analogous to the previous case.

2. If $P \xrightarrow{b\cdot a(y)} Q$ then by Observation 6.1 we have that $P \equiv (\nu\tilde{n})(b\cdot a(y).P_1 \mid P_2)$ and $Q \equiv (\nu\tilde{n})(P_1 \mid P_2)$. Consequently, by definition of encoding and lts, $[\![P]\!] \xrightarrow{\tau}_\pi \xrightarrow{b(x,y,z)}_\pi Q'$ where

$$\begin{aligned} Q' \equiv_\pi (\nu\tilde{n})(\ (\nu w)(\quad & [x=a](z|[\![P_1]\!]), (\overline{b}\langle x,y,z\rangle \mid \overline{w}) \mid \\ & !w.b(x,y,z).[x=a](z\,|\,[\![P_1]\!]), (\overline{b}\langle x,y,z\rangle \mid \overline{w})\ ) \mid \\ & [\![P_2]\!]\ ) \end{aligned}$$

Now for all σ, if $\sigma(x) = a$ then $Q'\sigma \equiv_\pi$
$(\nu\tilde{n})(z\sigma|[\![P_1]\!]\sigma|(\nu w)(!w.b(x,y,z).[x=a](z\,|\,[\![P_1]\!]), (\overline{b}\langle x,y,z\rangle \mid \overline{w}))\sigma \,|\, [\![P_2]\!]\sigma)$
$\equiv_\pi z\sigma|(\nu\tilde{n})([\![P_1]\!] \mid [\![P_2]\!]|(\nu w)(!w.b(x,y,z).[x=a](z\,|\,[\![P_1]\!]), (\overline{b}\langle x,y,z\rangle \mid \overline{w})))\sigma$
$\sim_\pi z\sigma|[\![Q]\!]\sigma;$

on the other hand, if $\sigma(x) \neq a$ then,
$Q'\sigma \equiv_\pi$
$(\nu\tilde{n})([\sigma(x)=a](z\sigma|[\![P_1]\!]\sigma) \,|\, (\nu w)([\sigma(x)\neq a](\overline{b}\langle x,y,z\rangle\sigma \mid \overline{w})$
$\mid !w.b(x,y,z).[x=a](z\,|\,[\![P_1]\!]), (\overline{b}\langle x,y,z\rangle \mid \overline{w}))\sigma \,|\, [\![P_2]\!]\sigma)$
$\sim_\pi \overline{b}\langle x,y,z\rangle\sigma|[\![P]\!]\sigma.$

Note that in both cases the restrictions on $\tilde{n}$ and w do not interfere with the names substituted by σ, because of the latter being capture-avoiding.

3. $\mu = \tau$. A τ action could be achieved by the rules (COM), (PAR), (CLOSE), (BANG) and (RES). We just show the (COM) case as the other ones are similar. Applying rule (COM) we have that $P = P_1 \mid P_2$ for some P_1, P_2, and

$$\frac{P_1 \xrightarrow{b\cdot a(y)} Q_1,\ P_2 \xrightarrow{\overline{b\cdot a}\langle c\rangle} Q_2}{P_1 \mid P_2 \xrightarrow{\tau} Q_1\{c/y\} \mid Q_2}$$

By point (2) there exists Q', Q'' and Q''' such that

- $[\![P_1]\!] \xrightarrow{\tau}_\pi Q''$,
- $Q'' \xrightarrow{b(x,y,z)}_\pi Q'$,
- and if $\sigma(x) = a$, $Q'\sigma \sim_\pi z\sigma|[\![Q_1]\!]\sigma$.

By point (1) and rule (CLOSE)

$$\frac{Q'' \xrightarrow{b(x,y,z)}_\pi Q',\ [\![P_2]\!] \xrightarrow{\overline{b}\langle a,c,\nu z\rangle}_\pi \overline{z}|\,[\![Q_2]\!],\ \sigma = \{a,c,z/x,y,z\}}{Q'' \mid [\![P_2]\!] \xrightarrow{\tau}_\pi (\nu z)(Q'\sigma \mid \overline{z}\,|\,[\![Q_2]\!])}$$

where $\sigma(a) = x$ and again by (COM) and (RES)

$$\frac{Q'\sigma \xrightarrow{z}_\pi Q''',\ \overline{z}\,|\,[\![Q_2]\!] \xrightarrow{\overline{z}}_\pi [\![Q_2]\!]}{(\nu z)(Q'\sigma \mid (\overline{z}\,|\,[\![P_2]\!])) \xrightarrow{\tau}_\pi (\nu z)(Q''' \mid [\![Q_2]\!])}$$

Note that z is not free in Q''' or $[\![Q_2]\!]$, giving us $(\nu z)(Q''' \mid [\![Q_2]\!]) \equiv_\pi Q''' \mid [\![Q_2]\!]$, which allows us to conclude, again by point (2), $Q''' \mid [\![Q_2]\!] \sim_\pi [\![Q_1]\!] \mid [\![Q_2]\!]$.

□

The following lemma explores the relation between weak actions, from a term to its encoding.

Lemma 6.2. *For any process P in $a\pi_2 \backslash a\pi_1$*

1. *if $P \xrightarrow{\tau} Q$ then $[\![P]\!] \Longrightarrow_\pi \xrightarrow{\tau}_\pi \xrightarrow{\tau}_\pi \xrightarrow{\tau}_\pi \Longrightarrow_\pi \sim_\pi [\![Q]\!]$;*
2. *if $P \xrightarrow{\mu} Q$, where $\mu \in \{\overline{b \cdot a}\langle c\rangle, \overline{b \cdot a}\langle \nu y\rangle\}$, then for any $z \notin fn(P)$, $[\![P]\!] \overset{[\![\mu]\!]}{\Longrightarrow}_\pi \sim_\pi \overline{z}\,|\,[\![Q]\!]$;*
3. *if $P \longrightarrow R \xrightarrow{b\cdot a(y)} \longrightarrow Q$ then $\exists R', Q'.[\![P]\!] \Longrightarrow_\pi R' \xrightarrow{\tau}_\pi \xrightarrow{b(x,y,z)}_\pi \Longrightarrow_\pi Q'$, and for any $\sigma : \{x,y,z\} \to \mathcal{N}$,*
   - *if $\sigma(x) = a$, $Q'\sigma \sim_\pi z\sigma | [\![Q]\!]\sigma$;*
   - *if $\sigma(x) \neq a$, $Q'\sigma \sim_\pi \overline{b}\langle x,y,z\rangle\sigma | R'\sigma$ and $R' \sim_\pi [\![R]\!]$.*

Proof. 1. By point (3) of Lemma 6.1 and by transitivity of $\sim_\pi$.

2. By points (1) above and (1) of Lemma 6.1.

3. By points (1) above and (2) of Lemma 6.1.

□

We now establish a limited form of correspondence between strong actions in encoded terms and actions in the original terms. This lemma will be useful to establish the weak correspondence needed to prove soundness.

Lemma 6.3. *For any process P in $a\pi_2 \backslash a\pi_1$:*

1. *If $[\![P]\!] \xrightarrow{\mu}_\pi Q'$, where $\mu \in \{\overline{b}\langle a,c,\nu z\rangle, \overline{b}\langle a,\nu c,\nu z\rangle\}$, then there exists Q such that $P \xrightarrow{\overline{b\cdot a}\langle c\rangle} Q$ (respectively $\overline{b \cdot a}\langle \nu c\rangle$), $Q' \equiv_\pi \overline{z}\,|\,[\![Q]\!]$ and $z \notin fn(Q)$.*
2. *If $[\![P]\!] \xrightarrow{\tau}_\pi \xrightarrow{b(x,y,z)}_\pi Q'$ then there exists a, Q such that for any $\sigma : \{x,y,z\} \to \mathcal{N}$,*
   - *if $\sigma(x) = a$, $Q'\sigma \sim_\pi z\sigma\,|\,[\![Q]\!]\sigma$;*
   - *if $\sigma(x) \neq a$, $Q'\sigma \sim_\pi \overline{b}\langle x,y,z\rangle\sigma\,|\,[\![P]\!]\sigma$;*

   and, if $a \in fn(P)$ then $P \xrightarrow{b\cdot a(y)} Q$;

Proof. 1. By Definition 2.3, if $[\![P]\!] \xrightarrow{\overline{b}\langle a,c,\nu z\rangle}_\pi Q'$ then by the presence of a bound name as third parameter in the output action and by definition of encoding, $P \equiv \overline{b \cdot a}\langle c\rangle | Q$ and $[\![P]\!] \equiv_\pi (\nu z)(\overline{b}\langle a,c,z\rangle\,|\,\overline{z}\rangle\,|\,[\![Q]\!]$. By rule (OPEN) $[\![P]\!] \xrightarrow{\overline{b}\langle a,c,\nu z\rangle} \overline{z}\,|\,[\![Q]\!]$ and by construction $z \notin fn(Q)$.

   The case with $\xrightarrow{\overline{b\cdot a}\langle \nu c\rangle}$ is analogous.

2. Analogous to point (1), reasoning similarly to point (2) of Lemma 6.1.

□

The next two lemmata are technical lemmata needed to prove the weak operational correspondence between tau actions.

Lemma 6.4. *If* $[\![P]\!] \dot{\approx}_\pi P'$ *and* $P' \xrightarrow{\tau}_\pi Q'$ *then* $\exists Q.[\![P]\!] \Longrightarrow_\pi [\![Q]\!]$ *and* $[\![Q]\!] \dot{\approx}_\pi Q'$.

Proof. If $[\![P]\!] \dot{\approx}_\pi Q'$ we are done taking $Q \equiv P$. If $[\![P]\!] \not\dot{\approx}_\pi Q'$, then by the hypothesis $[\![P]\!] \dot{\approx}_\pi P'$ we have that it must be the case that $\exists R.[\![P]\!] \Longrightarrow_\pi R$, and $R \dot{\approx}_\pi Q'$ but $R \not\dot{\approx}_\pi [\![P]\!]$. By definition of encoding, the only actions that $[\![P]\!]$ can perform without preserving weak bisimulation, come from the translation of a communication in ${}^e\pi$. We can chose a trace of actions, originating from $[\![P]\!]$ such that for each such reduction there is one and only one step of initialisation and termination of the protocol. Since all those additional steps preserve weak bisimilarity, we have that $[\![P]\!] \Longrightarrow_\pi R$ implies $\exists Q.[\![P]\!] \Longrightarrow_\pi [\![Q]\!]$ where $R \dot{\approx}_\pi [\![Q]\!]$. By transitivity of $\dot{\approx}_\pi$, we conclude. □

Lemma 6.5. *If* $[\![P]\!] \Longrightarrow_\pi [\![Q]\!]$ *then* $P \longrightarrow Q$.

Proof. Follows by definition of encoding and by induction on the number of reduction steps. □

We can now state the weak operational correspondence between actions in encoded terms and actions in source terms.

Lemma 6.6. *For any process* P *in* $a\pi_2 \backslash a\pi_1$*:*

1. *If* $[\![P]\!] \overset{\tau}{\Longrightarrow}_\pi Q'$ *then there exists* Q *such that* $P \longrightarrow Q$ *and* $[\![Q]\!] \approx_\pi Q'$.
2. *If* $[\![P]\!] \overset{\mu}{\Longrightarrow}_\pi Q'$ *where* $\mu \in \{\overline{b}\langle a, c, \nu z\rangle, \overline{b}\langle a, \nu c, \nu z\rangle\}$ *then there exists* Q *such that* $P \xrightarrow{\overline{b \cdot a}\langle c\rangle} Q$ *(respectively* $\overline{b \cdot a}\langle \nu c\rangle$*) and* $Q' \approx_\pi \overline{z}|[\![Q]\!]$ *and* $z \notin fn(Q)$.
3. *If* $[\![P]\!] \Longrightarrow_\pi R' \xrightarrow{\tau}_\pi \xrightarrow{b(x,y,z)}_\pi \Longrightarrow_\pi Q'$ *then there exists* a, Q, R *such that* $P \longrightarrow R$ *and, for any* $\sigma : \{x,y,z\} \to \mathcal{N}$,
   - *if* $\sigma(x) = a$, $Q'\sigma \approx_\pi z\sigma\,|\,[\![Q]\!]\sigma$;
   - *if* $\sigma(x) \neq a$, $Q'\sigma \approx_\pi \overline{b}\langle x,y,z\rangle\sigma\,|\,R'\sigma$ *and* $R' \approx_\pi [\![R]\!]$;

   and if $a \in fn(P)$ *then* $R \xrightarrow{b \cdot a(y)} \longrightarrow Q$.

Proof.
1. Follows from the preceding two lemmata.
2. Follows by point (1) above and by point (1) in Lemma 6.3.
3. Follows by point (1) above and by point (2) in Lemma 6.3.
4. Follows by point (1) above and by point (5) in Lemma 6.3.

□

The following lemma states the soundness of the encoding.

Lemma 6.7. *If* $[\![P]\!] \dot{\approx}_\pi [\![Q]\!]$ *then* $P \dot{\approx} Q$.

Proof. In the proof we reason up to weak bisimilarity, following a remark of [101]: the technique is sound because we are considering weak actions both for the player and for the adversary in the bisimulation game. We split the proof in four cases.

- $P \xrightarrow{\overline{b\cdot a}\langle y\rangle} P_1$: by point (2) of Lemma 6.2 $[\![P]\!] \overset{\overline{b}\langle a,y,\nu z\rangle}{\Longrightarrow}_\pi \sim \overline{z}\,|\,[\![P_1]\!]$, for any $z \notin fn(P)$; by the hypothesis $[\![P]\!] \dot{\approx}_\pi [\![Q]\!]$, $[\![Q]\!] \overset{\overline{b}\langle a,y,\nu z\rangle}{\Longrightarrow}_\pi Q' \dot{\approx}_\pi \overline{z}\,|\,[\![P_1]\!]$; by point (2) of Lemma 6.6 we have that $\exists Q_1.\ Q \xrightarrow{\overline{b\cdot a}\langle y\rangle} Q_1$ and $Q' \dot{\approx}_\pi \overline{z}|[\![Q_1]\!]$; by transitivity of $\dot{\approx}_\pi$ we have that $\overline{z}\,|\,[\![P_1]\!] \dot{\approx}_\pi \overline{z}|[\![Q_1]\!]$; by definition of $\dot{\approx}_\pi$, noting that z is fresh, we have $[\![P_1]\!] \dot{\approx}_\pi [\![Q_1]\!]$; inductively, $P_1 \dot{\approx} Q_1$.

- $P \xrightarrow{\overline{b\cdot a}\langle \nu y\rangle} P_1$: similar to the previous case.

- $P \xrightarrow{b\cdot a(y)} P_1$: by point (3) of Lemma 6.2, there exists P' such that $[\![P]\!] \overset{b(x,y,z)}{\Longrightarrow}_\pi P'$ and for any $\sigma : \{x,y,z\} \to \mathcal{N}$, if $\sigma(x) = a$, $P'\sigma \sim_\pi z\sigma|[\![P_1]\!]\sigma$, where $z \notin fn(P)$, by definition of encoding. By the hypothesis $[\![P]\!] \dot{\approx}_\pi [\![Q]\!]$ and by definition of $\dot{\approx}_\pi$, it must be the case that $[\![Q]\!] \overset{b(x,y,z)}{\Longrightarrow}_\pi Q'\sigma \dot{\approx}_\pi P'\sigma$. By point (4) of Lemma 6.6, we have that there exist a', Q_1 such that, for any $\sigma : \{x,y,z\} \to \mathcal{N}$, if $\sigma(x) = a'$, then $Q'\sigma \approx_\pi z\sigma\,|\,[\![Q_1]\!]\sigma$, and if $a' \in fn(Q)$ then $Q \xrightarrow{b\cdot a(y)} Q_1$. Considering all the substitutions σ such that $\sigma(x) = a$ and $\sigma(z) = w$ for some fresh name w, from the hypothesis $Q'\sigma \dot{\approx}_\pi P'\sigma$ we have that $a = a'$, obtaining $w|[\![P_1]\!]\sigma \dot{\approx}_\pi w\,|\,[\![Q_1]\!]\sigma$. From the freshness of w follows that $[\![P_1]\!]\sigma \dot{\approx}_\pi [\![Q_1]\!]\sigma$, whence by uniformity of the encoding, follows inductively $\forall\sigma.P_1\sigma \dot{\approx}_\pi Q_1\sigma$.

- $P \xrightarrow{\tau} P_1$: by point (1) of Lemma 6.2, there exists P' such that $[\![P]\!] \xrightarrow{\tau} P' \sim_\pi [\![P_1]\!]$; by the hypothesis $[\![P]\!] \dot{\approx}_\pi [\![Q]\!]$ and by definition of $\dot{\approx}_\pi$, $\exists Q'.\ [\![Q]\!] \overset{\tau}{\Longrightarrow}_\pi Q'$, and $P' \dot{\approx}_\pi Q'$; by transitivity of $\dot{\approx}_\pi$ and by point (1) of Lemma 6.6 we have that $\exists Q_1.Q \longrightarrow Q_1$ and $Q' \approx_\pi [\![Q_1]\!] \dot{\approx}_\pi [\![P_1]\!]$; inductively, $P_1 \dot{\approx} Q_1$.

□

Resuming, we report the statement of Theorem 6.1 below.

Theorem 6.1. *For all processes P and Q in aπ₂:*

1. *If $P \xrightarrow{\tau} Q$ then $[\![P]\!] \xrightarrow{\tau}\xrightarrow{\tau}\xrightarrow{\tau}\sim [\![Q]\!]$.*
2. *If $[\![P]\!] \Longrightarrow_\pi Q'$ then there exists Q such that $P \longrightarrow Q$ and $Q' \approx [\![Q]\!]$.*
3. *If $[\![P]\!] \dot{\approx} [\![Q]\!]$ then $P \dot{\approx} Q$.*

Proof. 1. Point (3) of Lemma 6.1.

2. Point (1) of Lemma 6.6.

3. Lemma 6.7.

□

6.3 Expressivity of Polyadic Synchronisation

6.3.1 Matching

The π-calculus Rule (MATCH) seems to contradict the intuition that barbs (i.e. observability) should depend on the *structure* of a process, regardless of the actual value of its state (the set of bindings between names and channels). In fact π^m, the sub-calculus without matching, enjoys the following property.

Observation 6.2. For any process P in π^m, for any name x, and for any substitution σ, $P \downarrow_x$ if and only if $P\sigma \downarrow_{\sigma(x)}$, and $P \downarrow_{\overline{x}}$ if and only if $P\sigma \downarrow_{\overline{\sigma(x)}}$.

Proof. By cases on the syntax and by definition of barbs. □

Conversely, this property does not hold in π: given $M \triangleq [x=y]x$, $\sigma_1 = \{z/x\}\{w/y\}$ (where $z \neq w$), and $\sigma_2 = \{z/x\}\{z/y\}$ we have that $M\sigma_1 \not\downarrow$ but $M\sigma_2 \downarrow_z$.

Remark 6.1. In the light of what we have just said, we advocate a different definition of matching in π:

$$\textit{Syntax}: \; P ::= \ldots \mid [x=y]\tau.P \quad \textit{Semantics}: \; \overline{[x=x]\tau.P \xrightarrow{\tau} P} \quad (\text{MATCH})$$

This rule would allow Observation 6.2 to be extended to the whole language, supporting the intuition that the difference introduced in the observability of the process is due to an internal reduction: matching becomes an *operation*, like in most other languages. We will see in Section 6.3.1 how this definition is supported by a correspondence with the derivation of matching in ${}^e\pi$.

We now show that the matching operator cannot be derived in π-calculus, and therefore needs to be taken as primitive. In ${}^e\pi$ instead it is possible to define matching as a derived operator. This fact constitutes a first separation result between the expressivity of the two languages. For example, in π^m it is not possible to write a tester process able to tell wether or not two arbitrary names denote the same channel, without disturbing the communications on the channel (or channels) denoted by those names. The peculiarity of matching is in fact to allow a process to evolve if and only if two names denote the same observable. Consequently the intended observables to be preserved by a reasonable semantics of matching are all the visible actions performed on the channel names that can be tested for equality, and therefore are all the barbs.

The negative result

We start noting some properties of processes in π and π^m.

Observation 6.3. For any processes P, Q in π or π^m, if $P \xrightarrow{\tau} Q$ then, for any substitution σ, $P\sigma \xrightarrow{\tau} Q\sigma$.

Proof. By cases on the syntax the reduction must be defined on two input and output prefixes with the same syntactical subject, and therefore remains executable under any substitution on P. □

The crucial property that characterises the absence of matching in π^m is that if a substitution does not affect the barbs of a process, then it does not increase its ability to reduce.

Proposition 6.2. *For any process in π^m, for any substitution σ with domain $\mathcal{D}_\sigma$, if $(\forall x \in \mathcal{D}_\sigma.\ P \not\downarrow_x \wedge P \not\downarrow_{\overline{x}})$ holds and $P\sigma \xrightarrow{\tau} Q\sigma$, then $P \xrightarrow{\tau} Q$.*

Proof. Similarly to the proof of Observation 6.3, if $P \not\downarrow_x$ and $P \not\downarrow_{\overline{x}}$ for every $x \in \mathcal{D}_\sigma$, then the reduction $P\sigma \xrightarrow{\tau} Q\sigma$ cannot be obtained by prefixes whose subjects are syntactically different, and therefore the reduction remains executable also without applying the substitution. □

This law does not hold both in π and in ${}^e\pi$. Consider the substitution $\sigma = \{x/y\}$, the π process $P_\pi \triangleq [x=y]\tau.Q$, and the ${}^e\pi$ process $P_{e_\pi} \triangleq (\nu z)(\overline{z \cdot x}|z \cdot y.Q)$: both $P_\pi \not\downarrow$ and $P_{e_\pi} \not\downarrow$, but both $P_\pi\sigma$ and $P_{e_\pi}\sigma$ can reduce whereas P_π and P_{e_π} cannot. From Proposition 6.2 and Observation 6.3 follows this useful corollary.

Corollary 6.1. *Let σ range over arbitrary substitutions. For any π^m process P, if $P \not\downarrow$ then $(\exists\sigma.P\sigma \xrightarrow{\tau} Q\sigma) \Rightarrow (\forall\sigma.P\sigma \xrightarrow{\tau} Q\sigma)$.*

Theorem 6.2 (Matching). *There exists no sensible encoding of $a\pi^=$ in π^m.*

Proof. Suppose that $[\![-]\!]$ is such an encoding: we have noted that a reasonable semantics of matching must preserve observations on barbs (The result holds also for a weaker notion of observation preservance: replacing (6.1) with $P \Downarrow \Leftrightarrow [\![P]\!] \Downarrow$.), and therefore

$$\forall x.\ P \Downarrow_x \Leftrightarrow [\![P]\!] \Downarrow_x; \quad \forall x.\ P \Downarrow_{\overline{x}} \Leftrightarrow [\![P]\!] \Downarrow_{\overline{x}} \tag{6.1}$$

which implies in particular that $P \not\Downarrow \Leftrightarrow [\![P]\!] \not\Downarrow$. Considering $M \triangleq [x=y]x$ we have that, by definition of matching and by (6.1),

$$\forall\sigma.\ M\sigma \Downarrow_{\sigma(x)} \Leftrightarrow [\![M\sigma]\!] \Downarrow_{\sigma(x)} \Leftrightarrow \sigma(x) = \sigma(y) \tag{6.2}$$

where σ is an arbitrary substitution. By strong uniformity the previous condition becomes

$$\forall\sigma\exists\theta.\ [\![M]\!]\theta \Downarrow_{\sigma(x)} \Leftrightarrow \sigma(x) = \sigma(y) \tag{6.3}$$

Consider now two substitutions σ_1 and σ_2 such that $\sigma_1(x) = \sigma_1(y)$ and $\sigma_2(x) \neq \sigma_2(y)$. By (6.2) we have that $M\sigma_1 \Downarrow_{\sigma_1(x)}$ and $M\sigma_2 \not\Downarrow$, and by (6.3) follows that

$$\exists\theta_1,\theta_2.\ [\![M]\!]\theta_1 \Downarrow_{\sigma_1(x)} \ \wedge\ [\![M]\!]\theta_2 \not\Downarrow \tag{6.4}$$

Expanding the definition of $\not\Downarrow$ we have that

$$[\![M]\!]\theta_2 \not\Downarrow \triangleq\ \forall M'.[\![M]\!]\theta_2 \longrightarrow M' \ \wedge\ M' \not\downarrow \tag{6.5}$$

By reflexivity and transitivity of $\longrightarrow$, applying at each step Observation 6.2 and Corollary 6.1, we can conclude that

$$\forall\theta.\ [\![M]\!]\theta \not\Downarrow \tag{6.6}$$

contradicts (6.4), therefore a sensible encoding of matching in π^m cannot exist. □

Remark 6.2. In the proof we have used process $[x=y]x$ as a counterexample for generality: it belongs both to the syntax of [84] and of [102]. This term doesn't enjoy Observation 6.2, but the proof holds also for process $[x=y]\tau.x$, which conforms to our definition of matching (Remark 6.1), clarifying that the key property characterising the non-encodability of matching is Proposition 6.2.

The positive result

We show that the encoding of matching in ${}^e\pi$ preserves strong full bisimilarity. Our encoding confirms the intuition that matching requires both input and output capabilities on a channel.

Lemma 6.8 (Encoding of Matching in ${}^e\pi$). *Let P be any process in ${}^e\pi^=$ such that $z \notin fn(P)$, and let $[\![[x=y]\tau.P]\!] \triangleq (\nu z)\,(\overline{z \cdot x} \mid z \cdot y.P)$. Then $[x=y]\tau.P$ and $[\![[x=y]\tau.P]\!]$ are strongly full bisimilar ($\sim$).*

Proof. By definition of $\sim$, given P and Q, $P \sim Q$ if and only if for all σ, $P\sigma \dot{\sim} Q\sigma$. We split the proof according to the behaviour of σ on x and y:

- $\sigma(x) \neq \sigma(y)$: both processes $[\sigma(x)=\sigma(y)]\tau.(P\sigma)$ and $(\nu z)(\overline{z \cdot \sigma(x)} \mid z \cdot \sigma(y).(P\sigma))$ cannot reduce and cannot perform any barb, therefore they are bisimilar;

- $\sigma(x) = \sigma(y)$: again both processes cannot perform any barb, and both reduce to the same process

$$[\sigma(x)=\sigma(y)]\tau.(P\sigma) \xrightarrow{\tau} P\sigma \tag{6.7}$$

$$(\nu z)(\overline{z \cdot \sigma(x)} \mid z \cdot \sigma(y).(P\sigma)) \xrightarrow{\tau} P\sigma \tag{6.8}$$

  By reflexivity of $\dot{\sim}$, $P\sigma \dot{\sim} P\sigma$.

□

6.3.2 Mixed Choice

We consider now the expressive power of the mixed choice construct. Palamidessi's result, separating π^m from π^s, holds analogously in our setting. The key point is that the impossibility to break the symmetry between identical processes that constitutes the core of the negative result, is not influenced by the ability to synchronise on more than one channel name. In fact, we can show that Lemma 4.1 of [92] generalises to our setting.

Lemma 6.9. *Let β be $\overline{u}\langle x\rangle$ or $\overline{u}\langle \nu x\rangle$, and let P be a process of π^s_n for some n. Assume that P can make two transitions $P \xrightarrow{\beta} Q$ and $P \xrightarrow{\nu(y)} R$. Then there exists S such that $Q \xrightarrow{\nu(y)} S$ and $R \xrightarrow{\beta} S$.*

Proof. Consider the syntactical definition of summations and prefixes in π^s_n, where $k, j \leq n$

$$P ::= \ \ldots \ \mid \ \Sigma_i \, \alpha^I_i.P_i \ \mid \ \Sigma_i \, \alpha^O_i.P_i \ \mid \ \ldots \tag{6.9}$$

$$\alpha^I ::= \ \ \tau \mid x_1 \cdot \ldots \cdot x_k(y); \quad \alpha^O ::= \ \ \tau \mid \overline{x_1 \cdot \ldots \cdot x_j}\langle y\rangle \tag{6.10}$$

and recall the syntactical characterisation of barbs given in Definition 2.3

$$\exists y, Q.P \xrightarrow{x(y)} Q \Leftrightarrow P \equiv (\nu x_1)\ldots(\nu x_n)(x(y).Q + M|N) \tag{6.11}$$

$$\exists y, Q.(P \xrightarrow{\overline{x}\langle y\rangle} Q \vee P \xrightarrow{\overline{x}\langle \nu y\rangle} Q) \Leftrightarrow P \equiv (\nu x_1)\ldots(\nu x_n)(\overline{x}\langle y\rangle.Q + M|N) \tag{6.12}$$

where x is different from each $x_1,...,x_n$. We have that

$$P \xrightarrow{v(y)} R \Leftrightarrow P \equiv (\nu x_1)...(\nu x_n)(v(y).Q_1 + M_1 | N_1) \tag{6.13}$$

$$P \xrightarrow{\beta} Q \Leftrightarrow P \equiv (\nu x_1)...(\nu x_n)(\overline{u}\langle x\rangle.Q_2 + M_2 | N_2) \tag{6.14}$$

where the names in u and v are all different from $x_1,...,x_n$. By (6.9) and (6.10) we have that M_1 cannot contain an output at the top level, and M_2 cannot contain an input. Therefore

$$N_1 \equiv N | \overline{u}\langle x\rangle.Q_2 + M_2 \tag{6.15}$$

$$N_2 \equiv N' | v(y).Q_1 + M_1 \tag{6.16}$$

that allows us to conclude that $N \equiv N'$ and

$$P \equiv (\nu x_1)...(\nu x_n)(\overline{u}\langle x\rangle.Q_1 + M_1 | N | v(y).Q_1 + M_2) \tag{6.17}$$

By applications of rules (RES), (OPEN), (PAR) and (PREFIX) follows the thesis. □

Remark 6.3. Technically speaking our Theorem 6.2, which shows that π^s is less expressive than $\pi^{s,=}$, restricts the generality of the separation result given by [92], because Lemma 4.1 shown in that reference is proved only for processes of π^s. Nonetheless it is an easy exercise to adapt the proof of the cited lemma to processes in $\pi^{s,=}$, restoring the full generality of Palamidessi's result.

Given the confluence property shown in Lemma 6.9, we can claim that an analogous of Palamidessi's separation result holds also in our setting.

Proposition 6.3 (Mixed Choice). *For any n,m, there exist no uniform encoding of π_n into $a\pi_m$ which preserves a reasonable semantics.*

Proof. By inspection of the cases in the proof of Theorem 4.2 of [92], replacing Lemma 4.1 of the reference with Lemma 6.9, we have that a symmetric electoral system cannot be encoded in $a\pi_m$, for any m. On the other hand, Claim 1 of the reference stating the existence of symmetric electoral systems in π^m, holds also for π_n with $n > 0$, noting that π^m is a sub-calculus of π_1. In the degenerate case of π_0 we have that for example

$$(\langle y\rangle + (x).\overline{o}\langle y\rangle) \mid (\langle z\rangle + (x).\overline{o}\langle z\rangle)$$

is a simple symmetric electoral system with two components. □

6.3.3 Polyadic Synchronisation

We show our main expressivity result: the expressive power of calculi in $\pi_{\mathbb{N}}$ depends on the degree of synchronisation. In particular, we show that there exists a separation problem: it is not possible to write two non-divergent processes in π_n to detect whether two vectors of $n+1$ identifiers are equal, whereas it is possible in π_{n+1}.

Matching Systems

We define a family of binary relations on processes called *Matching Systems*. In the following, let a *server template* of degree n be a process whose free names (the *process identifiers*) are $x_1,...,x_n$, and let a *client template* be defined as a server template with an additional free name y (the *process index*). In a matching system, copies of a template of each kind (say S and C) are instantiated in parallel as in $C_1\sigma_1|..|C_h\sigma_h|!S_1\theta_1|...|!S_k\theta_k$, where a substitution is applied to each process in order to "personalise" its identifiers. If the same substitution is applied to an instance of a client C_i and to one of a server S_j, the two instances are meant to recognise each other and perform some kind of meaningful behaviour. Therefore, a natural requirement is that the recognition process shall be finite. The process index constitutes the unique identity of a client. To represent the end of a (successful) recognition process between C_i and S_j, we require S_j to notify the index i of C_i on an additional global channel o that must be used only for this purpose (Without invalidating the result, we allow this special channel also in π_0, that normally cannot have channels.). We allow special observations on o, of the form $P\downarrow_{\overline{o}\langle i\rangle}$, in order to note the object of the communication as well as the subject. The Matching Problem MP_n consists in finding two processes that constitute a matching system of degree n, according to the following definition.

Definition 6.1. (Matching System) A client template C and a server template S of degree n constitute a Matching System $MS_n(C,S)$ of degree n if and only if

- for all finite set of server indexes J,
- for all finite sets of *fresh* client indexes $I \subset (\mathcal{N}\setminus\{x_1,...,x_n,o\})$,
- for any set of substitutions $\{\sigma_i\}_I$ and $\{\theta_j\}_J$ with domain $\{x_1,...,x_n\}$ (the process identifiers) and codomain $\mathcal{N}\setminus(I\cup\{o\})$,
- for all the processes P of the form $\Pi_{i\in I}\, C_i\{i/y\}\sigma_i \mid \Pi_{j\in J}\, !S_j\theta_j$

the following properties hold:

1. there is no infinite sequence of reductions starting from process P;
2. an output $\overline{o}\langle i\rangle$ is observable if and only if there are a client i and a server j with the same identifiers: $\forall i\in I.\ (P\Downarrow_{\overline{o}\langle i\rangle}\Leftrightarrow \exists j\in J.\ \sigma_i=\theta_j)$.

Note that the only condition on the server indexes is that J is finite. In fact the server indexes play a role only at the meta-level, and are not needed operationally.

Example 6.2. The π_0 processes $C'=\langle y\rangle$ and $S'=(w).\overline{o}\langle w\rangle$ constitute a matching system $MS_0(C',S')$: C' and S' have no identifiers, and therefore every exchange of indices is legal. For example:

$$\langle i^1\rangle|\langle i^2\rangle|\langle i^3\rangle|!(w).\overline{o}\langle w\rangle \xrightarrow{\tau}\xrightarrow{\tau}\xrightarrow{\tau} \overline{o}\langle i^1\rangle|\overline{o}\langle i^2\rangle|\overline{o}\langle i^3\rangle|!(w).\overline{o}\langle w\rangle \tag{6.18}$$

and the conditions of Definition 6.1 are trivially satisfied.

Example 6.3. The $a\pi$ processes $C = \overline{x}\langle y\rangle$ and $S = x(w).\overline{o}\langle w\rangle$ constitute a matching system $MS_1(C,S)$. It is easy to verify that for any possible substitution parallel instances of the processes interact if and only if the channels resulting from a substitution on x are equal, and in that case the identifier of the client is correctly forwarded on channel o by a server.

A crucial property of matching systems is to be *open*, in the sense that a process cannot make assumptions on the parallel context where it is executed. In fact, matching systems are closed under parallel instantiation: given $MS_m(C,S)$ and two instances

$$I,J,\ \Sigma \triangleq \{\sigma_i\}_I,\ \Theta \triangleq \{\theta_j\}_J,\ P \triangleq \Pi_{i\in I}\, C_i\{i/y\}\sigma_i \mid \Pi_{j\in J}\, !S_j\theta_j \tag{6.19}$$

$$I',J',\ \Gamma \triangleq \{\gamma_i\}_{I'},\ \Delta \triangleq \{\delta_j\}_{J'},\ P' \triangleq \Pi_{i\in I'}\, C_i\{i/y\}\gamma_i \mid \Pi_{j\in J'}\, !S_j\delta_j \tag{6.20}$$

also the parallel instance

$$I\cup I',\ J\cup J',\ \Sigma\cup\Gamma,\ \Theta\cup\Delta,\ P'' \triangleq P|P' \tag{6.21}$$

is a legal instance of the same matching system, provided that $I\cap I' = \emptyset$.

The $\pi_{\mathbb{N}}$ Hierarchy

Theorem 6.3 (Expressivity). *For all non-negative integer numbers n and m, the problem MP_m has a solution in π_n if and only if $n \geq m$.*

Proof. **(⇐)** We give a process in $a\pi_m$ (therefore also in π_n) providing a solution to MP_m:

$$C \triangleq \overline{x_1\cdot\ldots\cdot x_m}\langle y\rangle;\quad S \triangleq x_1\cdot\ldots\cdot x_m(w).\overline{o}\langle w\rangle \tag{6.22}$$

The degenerate case for $m = 0$ is reported in Example 6.2. As required by Definition 6.1, every instance of $MS_m(C,S)$ terminates: there are only a finite number of output prefixes at the top level, and the only outputs in the continuations of the servers are on channel o, but no input on o is allowed in the system. The only eventuality in which an output $\overline{o}\langle i\rangle$ may be observed is when communication happens between two instances of C and S subject to the same substitution.

(⇒) Consider the minimal case where $m = n+1$ and assume that MP_{n+1} has a solution in π_n. We show that this hypothesis leads to a contradiction. Let C,S be those two π_n process templates of degree $n+1$ such that $MS_{n+1}(C,S)$. They must necessarily satisfy conditions (1) and (2) of Definition 6.1 for all the well-formed instantiations of their parameters. In particular, we recall that matching systems are closed under parallel instantiations. With a slight abuse of notation, let C_i stand in the sequel for $C_i\{i/y\}$. Consider the instance

$$P \triangleq C_i\sigma_i \mid !S_j\theta_j \tag{6.23}$$

where $\sigma_i = \theta_j$: by condition (2) we have that $P \Downarrow \overline{o}\langle i\rangle$. It must be the case that neither $C_i\sigma_i \Downarrow_{\overline{o}\langle i\rangle}$ nor $S_j\theta_j \Downarrow_{\overline{o}\langle i\rangle}$, otherwise the well-formed instances $P_1 \triangleq C_i\sigma_i$ and $P_2 \triangleq !S_j\theta_j$ would not respect condition (2). Similarly, $\overline{o}\langle i\rangle$ cannot be made observable by an interaction of $C_i\sigma_i$ with a context not containing $!S_j\theta_j$. We conclude that at least a synchronisation must take place between a client and the corresponding server in order to verify the compatibility of the identifiers.

Since the identifiers to be tested are $n+1$ and both C and S by hypothesis are π_n processes, a barb presented by a client can contain at most n free names. Without loss of generality, suppose that $C_i\sigma_i \downarrow_u$, and $u = \sigma_i(x_1) \cdot \ldots \cdot \sigma_i(x_n)$. For interaction to take place, it must be the case that $S_j\theta_j \downarrow_{\overline{u}}$. Considering θ_k such that $\theta_k = \theta_j$ on the first n identifiers but $\theta_k(x_{n+1}) \neq \theta_i(x_{n+1})$, we have from Observation 6.2 that $S_k\theta_k \downarrow_{\overline{u}}$, and consequently process

$$P_3 \triangleq C_i\sigma_i \mid !S_k\theta_k \tag{6.24}$$

is such that both $P_3 \downarrow_u$ and $P_3 \downarrow_{\overline{u}}$. This shows that in principle C_i can communicate with a process that is not its right partner. Considering now

$$P_4 \triangleq C_i\sigma_i \mid !S_k\theta_k|!S_j\theta_j \tag{6.25}$$

it may be the case that $P_4 \xrightarrow{\tau} P_4'|!S_j\theta_j$ and consequently P_4' must eventually attempt a synchronisation with S_j in order to satisfy condition (2). We have shown that $C_i\sigma_i$ must continue to attempt synchronisation until eventually identifies S_j, since it cannot make assumptions on the parallel context.

Noting that the set $J_\downarrow = \{u|S_j\theta_j \downarrow_u\}$ is finite (the free names of $S_j\theta_j$ are its $n+1$ identifiers, and channel o) we can build a finite instance of $MS_n(C,S)$ that contains an infinite loop, contradicting condition (1)

$$P_\omega \triangleq C_i\sigma_i|\Pi_{h\in H}!S_h\theta_h \tag{6.26}$$

where $\forall u \in J_\downarrow.\exists h \in H.\theta_h(x_1) \cdot \ldots \cdot \theta_h(x_n) = u \wedge \theta_h(x_{n+1}) \neq \sigma_i(x_{n+1})$. □

Remark 6.4. Observation 6.2 holds also for π and ${}^e\pi^=$, provided that σ is injective. Therefore direction ($\Rightarrow$) of the proof holds also if matching is allowed as a primitive operator in C and S, taking care to chose in the counterexample an injective substitution θ_j.

We now look at the separation result from the perspective of encodings.

Proposition 6.4 (Polyadic Synchronisation). *There exists no* sensible *encoding of $a\pi_{n+1}$ in π_n, for any n.*

Proof. Supposing that $[\![-]\!]$ is such an encoding, we derive a contradiction. By part ($\Leftarrow$) of the proof of Theorem 6.3, there are two $a\pi_{n+1}$ processes C and S providing a solution for MP_{n+1}. A sensible encoding preserves the properties of a matching system and therefore we would have that the two π_n processes $[\![C]\!]$ and $[\![S]\!]$ provide a solution for MP_{n+1}, contradicting part ($\Rightarrow$) of Theorem 6.3. □

6.4 A hierarchy of Expressiveness

To conclude we compare the dialects of π-calculus that we have considered so far by means of a lattice of expressivity induced by the notion of sensible encoding.

Definition 6.2. Given two process calculi $\mathcal{P}$ and $\mathcal{Q}$ in a set of calculi $\mathcal{S}$, we write:

- $\mathcal{P} \preceq \mathcal{Q}$ if there exists a *sensible* encoding of $\mathcal{P}$ in $\mathcal{Q}$;
- $\mathcal{P} \simeq \mathcal{Q}$ if both $\mathcal{P} \preceq \mathcal{Q}$ and $\mathcal{Q} \preceq \mathcal{P}$;

- $\mathcal{P} \not\preceq \mathcal{Q}$ if it is not the case that $\mathcal{P} \preceq \mathcal{Q}$;
- $\mathcal{P} \prec \mathcal{Q}$ if $\mathcal{P} \preceq \mathcal{Q}$ and $\mathcal{Q} \not\preceq \mathcal{P}$;
- $\mathcal{P} \not\simeq \mathcal{Q}$ if both $\mathcal{P} \not\preceq \mathcal{Q}$ and $\mathcal{Q} \not\preceq \mathcal{P}$.

Lemma 6.10. *Let $\mathcal{S} = \pi_{\mathbb{N}} \cup \pi_{Op}$, let $\mathcal{S}_{/\simeq}$ be $\mathcal{S}$ quotiented by $\simeq$, and for each $[\mathcal{P}]_\simeq, [\mathcal{Q}]_\simeq \in \mathcal{S}_{/\simeq}$, let $[\mathcal{P}]_\simeq \preceq_{/\simeq} [\mathcal{Q}]_\simeq$ if and only if $\mathcal{P} \preceq \mathcal{Q}$. Then $(\mathcal{S}_{/\simeq}, \preceq_{/\simeq})$ is a complete lattice with bottom element $a\pi_0$ and top element ${}^e\pi$, corresponding to the diagram reported in Table 6.3 (where $\mathcal{P} \longrightarrow \mathcal{Q}$ means $\mathcal{P} \succ \mathcal{Q}$).*

Proof. Noticing that $\preceq$ is a preorder, and $\simeq$ is the equivalence relation induced on S by $\preceq$, follows immediately that $(\mathcal{S}_{/\simeq}, \preceq_{/\simeq})$ is a partial order. We now exhaustively check all the significative relations implied by the diagram.

- $a\pi \simeq \pi^s$, $a\pi^= \simeq \pi^{s,=}$: in both cases $\preceq$ is an embedding and $\succeq$ follows from [87], noticing in the formulation of [102] that the encoding behaves well with respect to arbitrary substitutions.
- $a\pi \prec a\pi^=$, $\pi^m \prec \pi$: in both cases $\preceq$ is an embedding and $\not\simeq$ follows from Theorem 6.2, in the second case noticing that $a\pi^= \preceq \pi$.
- $a\pi \prec \pi^m$, $a\pi^= \prec \pi$, $a\pi_n \prec \pi_n$: $\preceq$ is an embedding, $\not\simeq$ follows respectively from [92], Remark 6.3, and Proposition 6.3.
- (a) $a\pi_n \prec a\pi_{n+1}$, $\pi_n \prec \pi_{n+1}$: in both cases $\preceq$ is an embedding and $\not\simeq$ follows from Proposition 6.4;
  (b) $a\pi_0 \prec a\pi$, $\pi_0 \prec \pi$: in both cases $\preceq$ is a simple encoding where anonymous communication is translated by communicating on the same unrestricted channel, and $\not\preceq$ follows from Proposition 6.4 noticing that the process given in part ($\Leftarrow$) of the proof of Theorem 6.3 for $n = 1$ belongs to $a\pi$;
  (c) $a\pi^= \prec a\pi_2$, $\pi \prec \pi_2$: in both cases $\preceq$ is a simple encoding where matching is translated according to Lemma 6.8, and $\not\preceq$ follows from Proposition 6.4 and Remark 6.4.
- $\pi_n \not\simeq a\pi_{n+1}$: $\not\preceq$ from Proposition 6.3 and $\not\simeq$ from Proposition 6.4.
- $a\pi^= \not\simeq \pi_m$: $\not\preceq$ from Proposition 6.4 by noticing that in part ($\Leftarrow$) of the proof of Theorem 6.2, M belongs to $a\pi$, and $\not\simeq$ from Remark 6.3.

According to the ordering $\preceq$, the results above establish that $a\pi_0$ is the bottom element, ${}^e\pi$ is the top element and each subset X of $\mathcal{S}_{/\simeq}$ has limits in $\mathcal{S}_{/\simeq}$. □

From Table 6.3 emerges that the two constructs of polyadic synchronisation and mixed choice can be considered orthogonal. On the other hand, in the light of the results presented in Section 6.3.1, matching introduces a difference only when binary synchronisation is not available.

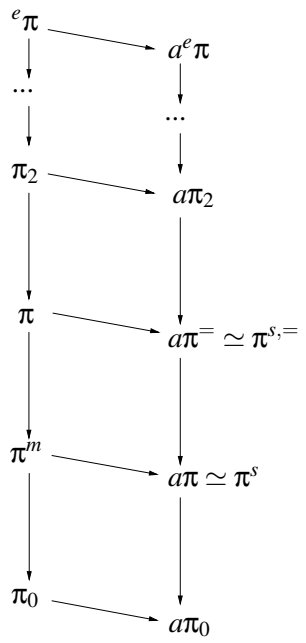


Table 6.3: Expressivity Lattice

Remark 6.5. If a comparison operator and a total order on names were provided, the leader election problem could be easily solved in the π-calculus without mixed choice, using for example the *LCR algorithm* of [36]. Analogously, if a composition operator on names was provided as primitive, also the Matching Problem would be solvable in π-calculus: the process identifiers could be composed together to constitute a single channel name.

6.5 Typing Polyadic Synchronisation

Below we extend the structural and nominal typing approach for the π-calculus to $^e\pi$, and in both cases we obtain proper extensions of the typing discipline, in the sense that the restriction of the system to $^e\pi_1$ gives exactly the corresponding type system for the π-calculus.

A naive extension of the π-calculus type systems would be unsound. For example the term

$$z(x,y).(\overline{x \cdot y}\langle 5\rangle \mid \overline{x \cdot y}\langle true\rangle)$$

leads to the derivation

$$\frac{z : [T,S], x : T, y : S \vdash \overline{x \cdot y}\langle 5\rangle \qquad z : [T,S], x : T, y : S \vdash \overline{x \cdot y}\langle true\rangle}{z : [T,S] \vdash z(x,y).(\overline{x \cdot y}\langle 5\rangle \mid \overline{x \cdot y}\langle true\rangle)}$$

where we fall short of a way to relate T and S to the type of channel $x \cdot y$. We will consider below various way to relate the type of each name constituting a channel with

the type of the channel itself.

Subtyping does not help in this case, because the problem is not one of containment of values, but of compatibility of the communication structure.

6.5.1 Structural Types for ${}^e\pi$

Uniform typing

The simplest extension of structural types for the π-calculus that we consider uses the same definitions of Section 2.2.6, and modifies the typing rules for input and output to require the same exchange type $[\tilde{T}]$ for each name z_i in a synchronisation vector:

$$(\text{T-INP})\frac{\Gamma \vdash z_i : [\tilde{T}] \qquad \Gamma, \tilde{x} : \tilde{T} \vdash P}{\Gamma \vdash z_1 \cdot \ldots \cdot z_n(\tilde{x}).P} \quad (\text{T-OUT})\frac{\Gamma \vdash z_i : [\tilde{T}] \qquad \Gamma \vdash \tilde{x} : \tilde{T} \qquad \Gamma \vdash P}{\Gamma \vdash \overline{z_1 \cdot \ldots \cdot z_n}\langle \tilde{x} \rangle.P}$$

This choice, forcing each name in a synchronisation vector to have the same type, limits drastically the set of well-typed ${}^e\pi$-terms, and coincides with the π-calculus structural typing in the case of ${}^e\pi_1$.

The type system satisfies the standard property of preserving types under reduction, and guarantees that well-typed processes will not incur in communication errors.

Example 6.4. As a positive point, the encoding of matching (of names of type $[\,]$)

$$[x = y].P \triangleq (\nu c)(\overline{c \cdot x}\langle\rangle \mid c \cdot y().P)$$

is typable following this approach:

$$\frac{\dfrac{\ldots \qquad \ldots}{\Gamma, x : [\,], y : [\,], c : [\,] \vdash \overline{c \cdot x}\langle\rangle \mid c \cdot y().P}}{\Gamma, x : [\,], y : [\,] \vdash [x = y].P}$$

Type concatenation

We now consider an alternative choice, where each name is treated as a monadic channel, and the type of a synchronisation vector is the tuple obtained by concatenation.

$$(\text{T-INP})\frac{\Gamma \vdash z_i : [T_i] \qquad \Gamma, \tilde{x} : T_1, \ldots, T_n \vdash P}{\Gamma \vdash z_1 \cdot \ldots \cdot z_n(\tilde{x}_n).P}$$

$$(\text{T-OUT})\frac{\Gamma \vdash z_i : [T_i] \qquad \Gamma \vdash \tilde{x} : T_1, \ldots, T_n \qquad \Gamma \vdash P}{\Gamma \vdash \overline{z_1 \cdot \ldots \cdot z_n}\langle \tilde{x}_n \rangle.P}$$

In absence of base types, it is possible to omit all types from the rules above, and a simple syntactic checking that synchronisation vectors are always the same length as the tuples they exchange, suffices to guarantee the absence of run time errors.

It follows that there exists no computable encoding of the polyadic π-calculus into this language, as communication errors are undecidable in that setting.

(TYPES)	$S,T ::= A \in \mathcal{N}_s$	A is a symbolic name type
(TYPE ENV)	$\Gamma ::= \Gamma, x : T \mid \emptyset$	x has symbolic name type T
(EXCHANGE POL)	$\Delta ::= \Delta, (\tilde{S}) : \tilde{T} \mid \emptyset$	channels of subject $\tilde{S}$ have object $\tilde{T}$

Table 6.4: Syntax for Types and Environments

Other approaches

We might consider the alternative choices given below

1. the type of a channel is the one of the first (or last) element of a synchronisation vector: this approach allows to type either different channels at the same location, or different locations having the same channel, e.g. the term below is typable if a vector takes the time of the last name

$$\overline{l \cdot a}\langle 5\rangle \mid \overline{l \cdot b}\langle \text{“}Hi!\text{”}\rangle;$$

2. similar to the type concatenation approach, but this time each name is treated as a polyadic channel: this system does not have principal types, because it is not possible to decide univocally where to split the object vector in corresponding of each name in a synchronisation vector;

3. the type of a name is a bidimensional infinite matrix where one coordinate represents the length of a vector, the other the position of the name in the assumed vector, and the value is again a structural exchange type (i.e. a tuple of matrixes).

6.5.2 Nominal Types for ${}^e\pi$

The nominal typing system for ${}^e\pi$ is a straightforward adaptation of the one for the π-calculus, where the only change is that the set of type definitions Δ, now called an *exchange policy*, contain associations between *vectors* of symbolic name types (corresponding to the concept of synchronisation vectors) and exchange types.

Let $\mathcal{N}_s$ be a denumerable set of *symbolic name types* disjoint from $\mathcal{N}$, ranged over by A,B,C. Our type system is based on two components: an environment Γ, associating each name to a symbolic name type, and an *exchange policy* Δ, assigning exchange types to vectors of symbolic name types. For example, given a term $x \cdot z(\tilde{y}).P$, Γ will record symbolic name types for x,z (say A,B), and Δ will prescribe for the vector $A \cdot B$ exchanges compatible with the type given by Γ to $\tilde{y}$. Both Γ and Δ are functions, no multiple definitions are allowed. The formal definition for types and environments is given in Table 6.4.

The rule for typing values, vectors of symbolic name types and processes are given in Table 6.5, where we use the shorthand notation $\tilde{x} : \tilde{T}$ for $x_1 : T_1, \ldots, x_n : T_n$ where $\tilde{x} = x_1, \ldots, x_n$ and $\tilde{T} = T_1, \ldots T_n$.

$$(\text{Tv-name})\frac{}{\Gamma, x : T \vdash x : T} \qquad (\text{T-Pol}) \qquad \frac{}{\Delta, (\tilde{N}) : \tilde{T} \Vdash (\tilde{N}) : \tilde{T}}$$

$$(\text{T-Par})\frac{\Gamma \vdash_\Delta P \qquad \Gamma \vdash_\Delta Q}{\Gamma \vdash_\Delta P \mid Q} \qquad (\text{T-Nil})\frac{}{\Gamma \vdash_\Delta 0}$$

$$(\text{T-Rep})\frac{\Gamma \vdash_\Delta P}{\Gamma \vdash_\Delta !P} \qquad (\text{T-Res})\frac{\Gamma, x : C \vdash_\Delta P}{\Gamma \vdash_\Delta (\nu x)\, P}$$

$$(\text{T-Inp})\frac{\Gamma \vdash z_i : S_i \qquad \Delta \Vdash (\tilde{S}) : \tilde{T} \qquad \Gamma, \tilde{x} : \tilde{T} \vdash_\Delta P}{\Gamma \vdash_\Delta z_1 \cdot \ldots \cdot z_n(\tilde{x}).P}$$

$$(\text{T-Out})\frac{\Gamma \vdash z_i : S_i \qquad \Delta \Vdash (\tilde{S}) : \tilde{T} \qquad \Gamma \vdash \tilde{y} : \tilde{T} \qquad \Gamma \vdash_\Delta P}{\Gamma \vdash_\Delta \overline{z_1 \cdot \ldots \cdot z_n}\langle \tilde{y} \rangle.P}$$

Table 6.5: Nominal types for ${}^e\pi$.

Results

Our type system satisfies the standard property of preserving types under reduction, and guarantees that well-typed processes will not incur in communication errors, as stated by the theorems below.

Lemma 6.11 (Strengthening). *If $\Gamma, x : C \vdash_\Delta P$ and x is not free in P then $\Gamma \vdash_\Delta P$.*

Proof. By induction on the typing rules.

- *Induction Base.*
  (T-Nil). Trivial.
- *Inductive Cases.*
  - (T-Par). For $P = P' \mid Q$, we have that $\Gamma, x : C \vdash_\Delta P' : \diamond$ and $\Gamma, x : C \vdash_\Delta Q : \diamond$ as $\Gamma, x : C \vdash_\Delta P' \mid Q : \diamond$. By induction hypothesis and rule (T-Par) again we have $\Gamma \vdash_\Delta P' \mid Q$.
  - (T-Rep). Trivial.
  - (T-Res). Trivial.
  - (T-Inp). By applying this rule, as $P = a_1 \cdot \ldots \cdot a_n(x_1, \ldots, x_m).P'$ we have that

$$\frac{\Gamma, x : C \vdash_\Delta a_i : N_i}{\Gamma, x : C, x_1 : V_1, \ldots, x_m : V_m \vdash_\Delta P' : \diamond}$$

As we know that x is not in the free names of P we then have that $\Gamma \vdash_\Delta a_i : N_i$, and by induction hypothesis we have that $\Gamma, x_1 : V_1, \ldots, x_m : V_m \vdash_\Delta P' : \diamond$ and so by rule (T-INP) again we get $\Gamma \vdash_\Delta P$.

– (T-OUT). Similar to previous case.

□

Lemma 6.12 (Weakening). *If $\Gamma \vdash_\Delta P$ then $\Gamma, x : C \vdash_\Delta P$ for any type C and any x not defined in Γ.*

Proof. Similar to proof of Lemma 6.11. □

Lemma 6.13 (Substitution Lemma). *If $\Gamma, x : C \vdash_\Delta P$ and $\Gamma \vdash_\Delta v : C$ then $\Gamma, x : C \vdash_\Delta P\{v/x\}$.*

Proof. By induction on the derivation of $\Gamma, x : C \vdash_\Delta P$.

- *Induction Base.*
  (T-NIL). Trivial.
- *Inductive Cases.*

  – (T-PAR). For $P = P' \mid Q$, we have that $\Gamma, x : C \vdash_\Delta P' : \diamond$ and $\Gamma, x : C \vdash_\Delta Q : \diamond$ as $\Gamma, x : C \vdash_\Delta P' \mid Q : \diamond$. By induction hypothesis and rule (T-PAR) again we have $\Gamma, x : C \vdash_\Delta (P' \mid Q)\{v/x\}$.

  – (T-REP). Trivial.

  – (T-RES). Trivial.

  – (T-INP). By applying this rule, as $P = a_1 \cdot \ldots \cdot a_n(x_1, \ldots, x_m).P'$ we have that

  $$\frac{\Gamma, x : C \vdash_\Delta a_i : N_i \qquad \Gamma, x : C, x_1 : V_1, \ldots, x_m : V_m \vdash_\Delta P' : \diamond}{}$$

  By induction hypothesis we get that $\Gamma, x : C, x_1 : V_1, \ldots, x_m : V_m \vdash_\Delta P'\{v/x\} : \diamond$. If x is not an element in $a_1 \cdot \ldots \cdot a_n$ then this case is proved. If there exists an i such that $x = a_i$ then we have by rule (TV-NAME) that $\Gamma, x : C \vdash x : C$.

  – (T-OUT). Similar to previous case.

□

Lemma 6.14 (Subject Congruence). *Assume that $\Gamma \vdash_\Delta P$ and $P \equiv Q$. Then we have that $\Gamma \vdash_\Delta Q$.*

Proof. We prove it by induction on the definition of $\equiv$.

- Monoidal conditions. Trivial.
- (STRUCT_1^π), (STRUCT_3^π) and (STRUCT_4^π). Trivial.
- (STRUCT_2^π). Under the assumption that $z \notin fn(Q)$ we have that:

  - if $\Gamma \vdash_\Delta (\nu z)\ (P \mid Q)$ then by rule (T-RES) we have that $\Gamma, z : C \vdash_\Delta P \mid Q : \diamond$ and by (T-PAR) we have tha $\Gamma, z : C \vdash_\Delta P : \diamond$ and $\Gamma, z : C \vdash_\Delta Q : \diamond$. Now, applying strengthening (Lemma 6.11) we have that $\Gamma \vdash_\Delta Q : \diamond$. It follows by (T-RES) and (T-PAR) again that $\Gamma \vdash_\Delta (\nu x)\ P \mid Q$.
  - if $\Gamma \vdash_\Delta (\nu x)\ P \mid Q$ then, similarly to the previous case we can prove $\Gamma \vdash_\Delta (\nu z)\ (P \mid Q)$ by using Lemma 6.12 (weakening).

- (STRUCT$^{\pi}_{5}$). Trivial.

Theorem 6.4 (Subject Reduction). *Assume that* $\Gamma \vdash_\Delta P$ *and* $P \longrightarrow Q$. *Then we have that* $\Gamma \vdash_\Delta Q$.

Proof. By induction on the depth of the derivation of $P \xrightarrow{\alpha} Q$. We consider only the most interesting cases.

- *Induction Base.*
  This is the case when we consider the reduction rule (COMM$^{\pi}$). By assumption we have

$$\Gamma \vdash_\Delta u(\tilde{x}).P' \mid \overline{u}\langle \tilde{y} \rangle.Q' \tag{6.27}$$
$$u(\tilde{x}).P' \mid \overline{u}\langle \tilde{y} \rangle.Q' \longrightarrow P'\{\tilde{y}/\tilde{x}\} \mid Q' \tag{6.28}$$

  If we now assume that $u = a_1 \cdot \ldots \cdot a_n$, by rule (T-PAR), and then by rules (T-INP) and (T-OUT) we have the following conditions. By applying the rule (T-INP) we have that

$$\Gamma, x_1 : V_1, \ldots, x_m : V_m \vdash_\Delta P' : \diamond \tag{6.29}$$

  By applying the rule (T-OUT) we have that

$$\Gamma \vdash_\Delta y_j : V_j \tag{6.30}$$
$$\Gamma \vdash_\Delta Q' : \diamond \tag{6.31}$$

  Now by (6.29), (6.30) and applying m times Lemma 6.13 we have hat

$$\Gamma \vdash_\Delta P' : \diamond \tag{6.32}$$

  Hence by (6.31), (6.32) and the rule (T-PAR) we have that $\Gamma \vdash_\Delta P$.

- *Inductive Cases.*
  We have to prove the inductive cases which include rules (RES$^{\pi}$), (PAR$^{\pi}$) and (STRUCT$^{\pi}$).

– Applying rule (RES$^\pi$) we have, by assumption, that

$$P \longrightarrow Q \tag{6.33}$$
$$\Gamma \vdash_\Delta (\nu x)\, P \tag{6.34}$$

Applying rule (T-RES) to (6.34) we have that

$$\Gamma, x : C \vdash_\Delta P : \diamond \tag{6.35}$$

Now by (6.33), (6.35) and induction hypothesis it follows $\Gamma, x : C \vdash_\Delta Q : \diamond$ and by rule (RES$^\pi$) we have that $\Gamma \vdash_\Delta (\nu x)\, Q$.

– The case of rule (PAR$^\pi$) is similar to the previous one.

– Concerning rule (STRUCT$^\pi$) it follows from Lemma 6.14, as in the previous cases.

Theorem 6.5 (Type Safety). *If $\Gamma \vdash_\Delta P$ then $P \not\to$.*

Proof. (Sketch) We show that for all Γ, Δ, P, if $P\dagger$ then $\Gamma \nvdash_\Delta P$. By definition of $-\dagger$ we have that $P \equiv (\nu \tilde{z})\ (P_1 \mid u(\tilde{x}_n).R \mid \overline{u}\langle \tilde{y}_m \rangle.Q)$ where $m \neq n$, and by Lemma 6.14 we need to show that $\Gamma \nvdash_\Delta (\nu \tilde{z})\ (P_1 \mid u(\tilde{x}_n).R \mid \overline{u}\langle \tilde{y}_m \rangle.Q)$. Repeatedly using (T-RES) and (T-PAR) we will reach the two sub-derivations for rules (T-INP) and (T-OUT) with conclusion, respectively $\Gamma \vdash_\Delta u(\tilde{x}_n).R$ and $\Gamma \vdash_\Delta \overline{u}\langle \tilde{y}_m \rangle.Q$, with premises necessarily incompatible. □

Examples

Despite its simplicity, combining named types together with polyadic synchronisation, yields great expressive power to our basic type system.

In the examples below, for the purpose of readability, we will use arbitrary identifiers to range over both names, synchronisation vectors, and symbolic name types.

Example 6.5. The generic encoding of matching as presented in above

$$[x = y].P \triangleq (\nu c)(\overline{c \cdot x}\langle\rangle \mid c \cdot y().P)$$

is typable with the nominal discipline:

$$\frac{\dfrac{\cdots \qquad \cdots}{\Gamma, x : A, y : A, c : B \vdash_\Delta \overline{c \cdot x}\langle\rangle \mid c \cdot y().P}}{\Gamma, x : A, y : A \vdash_\Delta [x = y].P}$$

where B can be any symbolic name type such that Δ contains $(B, A) : \diamond$.

Example 6.6. In the structural approach a channel cannot send its own name, unless including the apparatus of recursive types in the system. In the nominal approach this expressiveness comes for free: the example below is well-typed.

$$\Gamma = a : FW,\ \Delta = (FW) : FW, INT,\ \Gamma \vdash_\Delta !a(x,y).\overline{a}\langle x, y\rangle \mid \overline{a}\langle a, 5\rangle$$

Example 6.7. As another example we show how we can encode a limited form of polymorphism (generic operations) in our typed calculus. Let the identity function (with the corresponding invocation) as proposed for example in [53] be

$$!\mathtt{Id}(x,y).\overline{y}\langle x\rangle \qquad (\nu z)\ \overline{\mathtt{Id}}\langle v\rangle.z(w).P$$

Clearly, channel `Id` should be typable for all types T such that $x:T$ and $y:A$ with $A:T$ in Δ. Without explicitly introducing polymorphism, this generality is not achievable, but we can limit ourselves to a bounded quantification on T and encode polymorphism in our typed calculus. Suppose that T can be either `STRING` or `BOOL`. The new identity function becomes

$$!\mathtt{Id}\cdot s(x,y).\overline{y}\langle x\rangle \mid !\mathtt{Id}\cdot b(x,y).\overline{y}\langle x\rangle$$

with corresponding invocations

$$(\nu z)\ \overline{\mathtt{Id}\cdot s}\langle \text{“}Hello!\text{”}, z\rangle.z(w).P \qquad (\nu z)\ \overline{\mathtt{Id}\cdot b}\langle true, z\rangle.z(w).Q$$

and the exchange policy will contain

$$(\mathtt{ID},\mathtt{S}):\mathtt{STRING},\mathtt{STRINGchan} \qquad (\mathtt{ID},\mathtt{B}):\mathtt{BOOL},\mathtt{BOOLchan}$$

where the associations $b:\mathtt{B}, s:\mathtt{S}, \mathtt{Id}:\mathtt{ID},\ldots$ are in Γ. The point is that `Id` can still be sent along a channel as a polymorphic name, as for example in

$$\overline{\mathtt{op}}\langle\mathtt{Id}\rangle \mid \mathtt{op}(x).(bool(y).(\nu z)\overline{x\cdot b}\langle y\rangle.z(w).P \mid string(y).(\nu z)\overline{x\cdot s}\langle y\rangle.z(w).Q)$$

Example 6.8. In the setting of the previous example, we show that it is possible to define a properly polymorphic application operator in ${}^{e}\pi$ using our simple nominal typing discipline. Consider the term below:

$$\mathsf{apply}(t,f,v).(w)P = (\nu z)(\overline{f\cdot t}\langle z,v\rangle.z(w).P)$$

which invokes the version of f having type t with argument v. The term is well typed for all Γ,Δ,A,B,C,D,P such that

$$\Gamma\vdash t:A, f:B, v:C \qquad \Delta\Vdash D:[C], B\cdot A:[D,C] \qquad \Gamma, w:C\vdash_{\Delta} P$$

Chapter 7

Conclusions

This is the end, beautiful friend,
this is the end, my only friend, the end
— Jim Morrison, The Doors, 1967.

We shall conclude this dissertation with an overall summary of its contents which backs up related work and summaries found at the end of each previous chapter. In addition to this, we shall also discuss possible direction for future work.

This dissertation is about studying formal models for large scale systems based on the human notion of trust.

The SECURE project has deeply investigated many of the issues brought forward in this dissertation. But still, this is not the end of the story. There is still a lot to be done, and its successors will develop further systems based on new theories. This project has been one of the first tentative of formalising the human notion of trust and taking it into computer systems.

In Chapter 4, we presented a novel model for trust in distributed dynamic networks. The model builds on basic ideas from trust management systems and relies on domain theory to provide a semantic model for the interpretation of trust policies in trust-based security systems. Our technical contribution is based on bi-ordered structures $(\mathcal{T}, \preceq, \sqsubseteq)$, where the information ordering $\sqsubseteq$ measures the information contents of data, and is needed to compute the fix-point of mutually recursive policies, while the trust ordering $\preceq$ measures trust degrees and is used to make trust-informed decisions. Trust and information orderings, as relations, are continuous with respect to each other. Following this lead, we presented an interval construction as a canonical way to add uncertainty to trust lattices, and used the theory to guide the design and underpin the semantics of a simple, yet realistic policy language. We believe that the model can be used to explain existing trust-based systems, as well as help to design of new ones.

We based our investigation on the notion of (complete) lattice, since it is the standard in the literature. However, there are reasons to believe that *upper semi-lattices* – that is ordered structures in which only bounded sets have least upper bounds – provide a better starting model. From a modelling perspective, it is easy to think of situations in which it should not be possible to form the join of two trust level. For instance, in a starship's auto-destruction system, the capabilities "*possess key A*" and "*possess key B*" to ignite cannot be joined, as the capability of possessing *both* the keys is not contemplated in the system. From a theoretical point of view, the absence of a top

element simplifies the development of trust structures and enriches their theory.

We remark that the constructions illustrated here can be understood in abstract (categorical) terms. We have chosen to spell them out in set theoretical details to reach a wider audience. In particular, looking at the partial order $(D, \leq)$ as a category, our interval construction I can be seen as the free construction of a *double category* with all ω-filtered colimits. Specifically, $\preceq$ and $\sqsubseteq$ are respectively the horizontal and vertical arrows, while least upper bounds and their (mutual) commutation laws are expressed by as colimits. Furthermore, the $(I(D), \preceq)$ component of the interval construction is exactly the *functor category* $Arr \to D$, where $Arr = \bullet \to \bullet$ is the category with due objects and one non-identity arrow between them. More generally, the construction is related to the Yoneda embedding, as the image of the *hom-functor* $\mathsf{Hom}_D : D^{op} \times D \to$ *Set* is $(I(D), \sqsubseteq)$.

In Chapter 5, we have introduced `ctm`, a calculus for trust management. The calculus enjoys many new features which fit in global computing scenarios making use of the notion of trust.

Principals in `ctm` have two components: the policy and the protocol. The policy consists of an immutable part, α, and a variable s. The former expresses the logic of the policy, i.e. the rules following which decisions are taken, on the basis of past experiences. The latter records the observations which make up such experiences, as a function of the messages exchanged in interactions between principals.

It may be objected that this yields a generic concurrent calculus of stateful entities, and not a calculus specifically designed to represent trust-based systems. This is actually not the case. The key to the matter is that, while s is definitely a kind of store, principals have absolutely no control as to what it stores, or when it stores it: s is updated uniquely and exactly to reflect the outcome of interactions. These include feedback on untrusted clients and advice from trusted principals. In particular, a principal cannot store arbitrary values to s, or retrieve them from it. In other words, the calculus represents faithfully a distributed set of principals interacting with each other according to trust policies and risk assessment based on computational histories. Similarly it is not possible to compare `ctm` to an extension to locations of the applied π-calculus [1] as the latter does not model the notion of collecting observations even though function guards can represent policies.

We remark also that our use of guards works quite effectively with the choice of synchronous communications, to abstract the sequence of actions service request, risk assessment, response to client, and record observation, in a single, atomic step where trust-based decisions are localised.

We also have shown that `ctm` is a powerful calculus which, also, introduces new features so adding expressivity. We believe that in `ctm` we can encode many other existing tentatives of providing an operational model for trust-based systems, e.g. we are able to embed the systems provided in [11], [22] and [57].

In the last chapter, we have extended the synchronisation mechanism of π-calculus with polyadic synchronisation, where channels are vectors of names. It is a simple idea that has been adopted implicitly in many other calculi, but a formal treatment of its expressivity has not been given until now.

We have shown that matching cannot be encoded in π-calculus, whereas it is expressed naturally in terms of polyadic synchronisation. We have shown how a restricted form of polyadic synchronisation can be encoded *weakly* in π-calculus and

how, in the general case, the higher the degree of synchronisation of a calculus, the greater its expressive power. We have adapted the results on mixed choice to ${}^{e}\pi$, concluding that polyadic synchronisation and choice are independent from one another.

We have not delved into the question of the expressivity of mismatching, and we conjecture that it is not encodable in ${}^{e}\pi$ (and consequently in π). Mismatching is seldom considered in the literature, it does not seem to have many applications, and it complicates the equational theory of π-calculus. A remarkable exception is the work of [51], who proposes π_{B} (an extension of π-calculus with a *blocking* operator) to reason about the concept of dynamic binding in process calculi. The author shows that π_{B} and $\pi^{=,\neq}$ are mutually encodable. Polyadic synchronisation allows dynamic binding to be expressed in the π-calculus framework in a different way. The Dπ example reported in the beginning of the chapter, shows how a migrating process can gain access to the local names of a subsystem without requiring any explicit communication: the dynamic binding and re-binding of names is implicit in the semantics.

A possible interpretation of our main expressivity result is that locations are a *fundamental* concept in distributed calculi, since the attempt to encode them in models with simple synchronisation in general introduces divergence. We believe that ${}^{e}\pi$ has the expressive power to represent nested locations, but only in a static setting. The Ambient Calculus instead, seems to be beyond the reach of the expressive power of the model of synchronisation adopted by π-calculi.

The chapter about polyadic synchronisation concludes with a suggestion for a possible type system. This is just a first step, and could be seen as still ongoing work. The idea is to have further developments of these types, and then search for applicability to the `ctm` framework where they could turn to be useful in the static control of trust-based systems.

Also note that the ability to encode cryptography, distribution and concurrent objects in the ${}^{e}\pi$ setting can improve the understanding of these issues.

It would also be interesting to consider further extensions of the synchronisation mechanism, exploring the ideas of [80] in the light of the recent work of [86] concerning joint input, and of our work on polyadic synchronisation.

Bibliography

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. of the 28th symposium on Principles of Programming Languages (POPL'01)*, pages 104–115. ACM Press, 2001.

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

[3] A. Abdul-Rahman and S. Hailes. Supporting trust in virtual communities. In *Hawaii International Conference on System Sciences 33*, pages 1769–1777, 2000.

[4] K. Aberer and Z. Despotovic. Managing trust in a peer-2-peer information system. In *Proc. of 10th International Conference on Information and Knowledge Management*, 2001.

[5] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*. Addison-Wesley, 1995.

[6] R. Amadio, G. Boudol, and C. Lhoussaine. The receptive distributed pi-calculus. In *Proc. of the FST-TCS '99*, volume 1738 of *LNCS*. Springer-Verlag, 1999.

[7] A. W. Appel and E. W. Felten. Proof-carrying authentication. In *Proc. 6th ACM Conference on Computer and Communications Security*, 1999.

[8] J. Baeten. A brief history of process algebra. Technical Report CSR 04-02, Technische Universiteit Eindhoven, 2004.

[9] J. Baeten and W. Weijland. *Process Algebra*. Cambridge University Press, 1990.

[10] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North Holland, 2nd edition, 1984.

[11] M. Y. Becker and P. Sewell. Cassandra: Flexible trust management, applied to electronic health records. In *Proc. of CSFW '04*. IEEE Computer Society Press, 2004.

[12] H. Bekiĉ. Towards a mathematical theory of processes. Technical Report TR 25.125, IBM Laboratory, Vienna, 1971.

[13] M. Berger, K. Honda, and N. Yoshida. Sequentiality and the π-calculus. In *Proc. TLCA'01*, 2001.

[14] M. Berger, K. Honda, and N. Yoshida. Genericity and the pi-calculus. In *Proc. FOSSACS'03*, 2003.

[15] J. Bergstra and J. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985.

[16] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proc. IEEE Conference on Security and Privacy, Oakland*, 1996.

[17] M. Blaze, J. Feigenbaum, and J. Lacy. KeyNote: Trust management for public-key infrastructure. *LNCS*, 1550:59–63, 1999.

[18] C. Bodei, P. Degano, F. Nielson, and H. R. Nielson. Static analysis of processes for no read-up and no write-down. In *Proc. FOSSACS'99*, number 1578 in Lecture Notes in Computer Science, pages 120–134. Springer, 1999.

[19] C. Bodei, P. Degano, H. R. Nielson, and F. Nielson. Static analysis for secrecy and non-interference in networks of processes. In *Proc. PACT'01*, number 2127 in Lecture Notes in Computer Science, pages 27–41. Springer, 2001.

[20] G. Boudol. Asynchrony and the π-calculus. Rapporte de Recherche 1702, INRIA Sofia-Antipolis, 1992.

[21] G. Boudol and I. Castellani. Concurrency and atomicity. *Theoretical Computer Science*, 59(1-2):25–84, July 1988.

[22] C. Braghin, D. Gorla, and V. Sassone. A distributed calculus for role-based access control. In *Proc. of CSFW '04*. IEEE Computer Society Press, 2004.

[23] S. Buchegger and J. Y. Le Boudec. The effect of rumor spreading in reputation systems for mobile ad-hoc networks. In *Proceedings of WiOpt '03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, Sophia-Antipolis, France, March 2003.

[24] M. Burrows, M. Abadi, B. W. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *LNCS*, 576:1–23, 1991.

[25] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *In Proceedings of the Royal Society, Series A*, 426:18–36, 1991.

[26] V. Cahill, E. Gray, J.-M. Seigneur, C. Jensen, Y. Chen, B. Shand, N. Dimmock, A. Twigg, J. Bacon, C. English, W. Wagealla, S. Terzis, P. Nixon, G. Serugendo, C. Bryce, M. Carbone, K. Krukow, and M. Nielsen. Using Trust for Secure Collaboration in Uncertain Environments. *IEEE Pervasive Computing Magazine*, 2(3):52–61, 2003.

[27] L. Caires and L. Cardelli. A spatial logic for concurrency (Part I). *Information and Computation*, 186/2:194–235, 2003.

[28] L. Caires and L. Cardelli. A spatial logic for concurrency (Part II). *To appear in Information and Computation*, 2005.

[29] M. Carbone, M. Coccia, G. Ferrari, and S. Maffeis. Process algebra-guided design of java mobile network applications. Extended Abstract in Inf. Proc. of FMTJP'01, 2001.

[30] M. Carbone and S. Maffeis. On the expressive power of polyadic synchronisation in π-calculus. In *EXPRESS '02*, volume 68.2 of *ENTCS*. Elsevier Science Publishers, 2002.

[31] M. Carbone and S. Maffeis. On the expressive power of polyadic synchronisation in π-calculus. *Nordic Journal of Computing (NJC)*, 10(2), September 2003.

[32] M. Carbone, S. Maffeis, and A. Ravara. Typing polyadic synchronisation. Unpublished manuscript, Apr 2004.

[33] M. Carbone, M. Nielsen, and V. Sassone. A formal model for trust in dynamic networks. In *International Conference on Software Engineering and Formal Methods (SEFM'03)*. IEEE, 2003.

[34] M. Carbone, M. Nielsen, and V. Sassone. A calculus for trust management. In *Proc. of the FST-TCS '04*, volume 3328 of *LNCS*, pages 161–173. Springer-Verlag, 2004.

[35] L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000. An extended abstract appeared in *Proceedings of FoSSaCS '98*: 140–155.

[36] E. J. H. Chang and R. Roberts. An improved algorithm for decentralized extrema-finding in circular configurations of processes. *Communications of the ACM*, 22(5):281–283, May 1979.

[37] D. Chaum. Achieving electronic privacy. *Scientific American*, 267(2):96–101, 1992.

[38] T. Chothia and I. Stark. A distributed π-calculus with local areas of communication. In *Proceedings of HLCL '00*, number 41.2 in Electronic Notes in Theoretical Computer Science. Elsevier, 2001.

[39] Y.-H. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick, and M. Strauss. REFEREE: Trust management for web applications. *Computer Networks and ISDN Systems*, 29(8-13):953–964, 1997.

[40] D. Clarke, J.-E. Elien, C. Ellison, M. Fredette, A. Morcos, and R. L. Rivest. Certificate chain discovery in SPKI/SDSI. `http://theory.lcs.mit.edu/~rivest`, 1999.

[41] F. S. de Boer and C. Palamidessi. Embedding as a tool for language comparison. *Information and Computation*, 108(1):128–157, Jan. 1994.

[42] M. Deutsch. Cooperation and trust: Some theoretical notes. In *Jones M.R. (ed.), Nebraska Symposium on Motivations*, 1962.

[43] Y. Ding, P. Horster, and H. Petersen. A new approach for delegation using hierarchical delegation tokens. In *Communications and Multimedia Security*, pages 128–143, 1996.

[44] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. M. Thomas, and T. Ylonen. SPKI certificate theory. *Internet RFC 2693*, 1999.

[45] U. Engberg and M. Nielsen. A calculus of communicating systems with name-passing. Technical Report PB-208, DAIMI, University of Aarhus, 1986.

[46] C. English, W. Wagealla, P. Nixon, S. Terzis, H. Lowe, and A. McGettrick. Trusting collaboration in global computing systems. *LNCS*, 2692:136–149, 2003.

[47] G. Ferrari. Atomicity and concurrency control in process calculi. *FUNDINF: Fundamenta Informatica*, 29.4:341–368, 1997.

[48] C. Fournet. *The Join-Calculus: a Calculus for Distributed Mobile Programming*. Ph.D. thesis, Ecole Polytechnique, 1998.

[49] C. Fournet and G. Gonthier. The reflexive CHAM and the join-calculus. In *Proceedings of the 23rd ACM Symposium on Principles of Programming Languages*, pages 372–385, 1996.

[50] C. Fournet and G. Gonthier. A hierarchy of equivalences for asynchronous calculi. In *Proc. ICALP '98*, volume 1443 of *LNCS*, pages 844–855. Springer-Verlag, 1998.

[51] J. L. V. Frontana. *Dynamic Binding of Names in Calculi for Mobile Processes*. Ph.D. thesis, Department of Microelectronics and Information Technology, Mar. 2001.

[52] D. Gambetta. Trust: Making and breaking cooperative relations. http://www.sociology.ox.ac.uk/papers/trustbook.html, 2000.

[53] S. Gay. Some type systems for the pi-calculus. Unpublished draft, 1999.

[54] T. Grandison and M. Sloman. A survey of trust in internet application. *IEEE Communications Surveys, Fourth Quarter*, 2000.

[55] E. Gray, P. O'Connell, C. Jensen, S. Weber, J.-M. Seigneur, and C. Yong. Towards a Framework for Assessing Trust-Based Admission Control in Collaborative Ad Hoc Applications. Technical Report 66, Dept. of Computer Science, Trinity College Dublin, December 2002.

[56] G. Gräzer. *Lattice Theory: First Concepts and Distributive Lattices*. Freeman and Company, 1971.

[57] J. Guttman, J. Thayer, J. Carlson, J. Herzog, J. Ramsdell, and B. Sniffen. Trust management in strand spaces: A rely-guarantee method. In *Proc. of ESOP '04*, LNCS. Springer-Verlag, 2004.

[58] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of ACM*, 32(1):137–161, 1985.

[59] M. Hennessy and J. Riely. Resource access control in systems of mobile agents. In *Proceedings of HLCL '98*, volume 16.3 of *ENTCS*, pages 3–17. Elsevier Science Publishers, 1998.

[60] M. Hennessy and J. Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173(1):82–120, 2002.

[61] C. Hoare. *Communicating Sequential Processes*. Prentice Holl, New York, 1985.

[62] K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *Proceedings of ECOOP*, volume 512, pages 133–147, Berlin, Heidelberg, New York, Tokyo, 1991. Springer-Verlag.

[63] K. Honda and N. Yoshida. On reduction-based process semantics. In *Proc. of the FST-TCS '93*, volume 761 of *LNCS*, pages 373–387. Springer-Verlag, 1993.

[64] K. Honda, N. Yoshida, and M. Berger. Control in the π-calculus. In *Proc. Fourth ACM-SIGPLAN Continuation Workshop (CW'04)*, 2004.

[65] S. Jajodia, P. Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. In *Proc. of the 1997 IEEE Symposium on Security and Privacy, Oakland, CA*, 1997.

[66] J. Jalava. Trust or confidence: comparing Luhmann's and Giddens' views of trust. In *Proc. of 5th Conference of the European Sociological Association, 'Visions and Divisions'*, 2001.

[67] O. Jensen and R. Milner. Bigraphs and mobile processes. In *Conference Record of POPL'03: The 30th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Jan. 2003.

[68] O. Jensen and R. Milner. Bigraphs and mobile processes. Technical Report TR570-580, Cambridge Computer Laboratory, 2003-04.

[69] A. J. I. Jones and B. S. Firozabadi. On the characterisation of a trusting agent. In *Workshop on Deception, Trust and Fraud in Agent Societies*, 2000.

[70] C. Jonker and J. Treur. Formal analysis of models for the dynamics of trust based on experiences. *LNAI*, 1647:221–232, 1999.

[71] A. Jøsang. A logic for uncertain probabilities. *Fuzziness and Knowledge-Based Systems*, 9(3), 2001.

[72] K. Krukow, M. Nielsen, and V. Sassone. A formal framework for concrete reputation-systems with applications to history-based access control. Submitted. Available online: `http://www.brics.dk/~krukow`, 2005.

[73] U. W. Kulish and W. L. Miranker. *Computer Arithmetic in Theory and Practice*. Academic Press, 1981.

[74] F. Levi and S. Maffeis. On abstract interpretation of mobile ambients. *Information and Computation*, 188(2):179–240, 2004.

[75] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130. IEEE Computer Society Press, May 2002.

[76] N. Luhmann. *Trust and Power*. Chichester:Wiley, 1979.

[77] S. Marsh. *Formalising Trust as a Computational Concept*. Ph.D. thesis, Department of Computing Science and Mathematics, University of Stirling, 1994.

[78] D. H. McKnight and N. L. Chervany. The meanings of trust. *Trust in Cyber-Societies - LNAI*, 2246:27–54, 2001.

[79] M. Merro and D. Sangiorgi. On asynchrony in name-passing calculi. *LNCS*, 1443:856–867, 1998.

[80] R. Milner. Reduction and transition semantics for the π-calculus with multi-names. Unpublished manuscript RM18, mar 1991.

[81] R. Milner. The polyadic π-calculus: A tutorial. In *Logic and Algebra of Specification*. Springer-Verlag, Heidelberg, 1993.

[82] R. Milner. *Communication and concurrency*. Prentice Hall International (UK) Ltd., 1995.

[83] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, May 1999.

[84] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, I and II. *Information and Computation*, 100(1):1–40,41–77, Sept. 1992.

[85] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 114(1):149–171, 1993.

[86] U. Nestmann. On the expressive power of joint input. In *EXPRESS '98: Expressiveness in Concurrency (Nice, France, September 7, 1998)*, volume 16.2 of *ENTCS*. Elsevier Science Publishers, 1998.

[87] U. Nestmann. What is a good encoding of guarded choice? *Information and Computation*, 156:287–319, 2000.

[88] U. Nestmann and B. C. Pierce. Decoding choice encodings. *Journal of Information and Computation*, 163:1–59, 2000.

[89] R. D. Nicola, G. L. Ferrari, and R. Pugliese. KLAIM: A kernel language for agents interaction and mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, May 1998. Special Issue: Mobility and Network Aware Computing.

[90] R. D. Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[91] F. Nielson, R. R. Hansen, and H. R. Nielson. Abstract interpretation of Mobile Ambients. *Science of Computer Programming*, 47:145–175, 2003.

[92] C. Palamidessi. Comparing the expressive power of the synchronous and asynchronous π-calculi. *Mathematical Structures in Computer Science*, To appear, 2003.

[93] J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Thirteenth Annual Symposium on Logic in Computer Science (LICS) (Indiana)*, pages 176–185. IEEE, Computer Society Press, July 1998.

[94] C. Petri. *Kommunikation mit Automaten*. Ph.D. thesis, Institut fuer Instrumentelle Mathematik, Bonn, 1962.

[95] I. Phillips and M. Vigliotti. Electoral systems in ambient calculi. In *Proc. of 7th International Conference on Foundations of Software Science and Computation Structures*, volume 2987, pages 408–422. LNCS, 2004.

[96] B. C. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–453, Oct. 1996.

[97] G. D. Plotkin. Domains. Technical report, University of Edinburgh, 1983.

[98] I. Pörn. Some basic concepts of action. In *S. Stenlund (ed.), Logical Theory and Semantic Analysis. Reidel, Dordrecht*, 1974.

[99] P. V. Rangan. An axiomatic basis of trust in distributed systems. In *Symposium on Security and Privacy*, 1998.

[100] R. L. Rivest and B. Lampson. SDSI – A simple distributed security infrastructure. Presented at CRYPTO'96 Rumpsession, 1996.

[101] D. Sangiorgi and R. Milner. Techniques of weak bisimulation up to. Revised version of an article appeared in the Proc. CONCUR'92, LNCS 630, 1992.

[102] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[103] B. Schoenmakers. Security aspects of the Ecash^{TM} payment system. *State of the Art in Applied Cryptography: Course on Computer Security and Industrial Cryptography, LNCS (1)*, 528:338–353, 1998.

[104] D. S. Scott. Domains for denotational semantics. *ICALP '82 - LNCS*, 140, 1982.

[105] J.-M. Seigneur and C. D. Jensen. The claim tool kit for ad-hoc recognition of peer entities. *Journal of Science of Computer Programming*, 52(1), 2004.

[106] P. Sewell, P. Wojciechowski, and B. C. Pierce. Location independence for mobile agents. In *Proceedings of ICCL '98*, volume 1686 of *LNCS*. Springer-Verlag, Sept. 1999.

[107] B. Shand, N. Dimmock, and J. Bacon. Trust for transparent, ubiquitous collaboration. In *First IEEE Annual Conference on Pervasive Computing and Communications*. IEEE, 2003.

[108] V. Shmatikov and C. Talcott. Reputation-based trust management. In *Workshop on Issues in the Theory of Security (WITS)*, 2003.

[109] J. Siméon and P. Wadler. The essence of xml. In *Proc. of POPL '03*, January 2003.

[110] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[111] F. Valencia. *Temporal Concurrent Constraint Programming*. Ph.D. thesis, BRICS, Department of Computer Science, University of Aarhus, Denmark, 2002.

[112] V. T. Vasconcelos. *A process-calculus approach to typed concurrent objects*. PhD thesis, Keio University, 1994.

[113] V. T. Vasconcelos and M. Tokoro. A typing system for a calculus of objects. In *Object Technologies for Advanced Software*, volume 742 of *LNCS*, pages 460–474. Springer-Verlag, Nov. 1993.

[114] A. Venet. Abstract interpretation of the pi-calculus. In *Selected papers from the 5th LOMAPS Workshop on Analysis and Verification of Multiple-Agent Languages*, pages 51–75. Springer-Verlag, 1997.

[115] W. Wagealla, M. Carbone, C. English, S. Terzis, H. Lowe, and P. Nixon. A formal model for trust lifecycle management. In *the 1st International Workshop on Formal Aspect of Security and Trust (FAST)*, 2003.

[116] S. Weeks. Understanding trust management systems. In *Proc. IEEE Symposium on Security and Privacy, Oakland*, 2001.

[117] U. G. Wilhelm, L. Buttyàn, and S. Staamann. On the problem of trust in mobile agent systems. In *Symposium on Network and Distributed System Security*. Internet Society, 1998.

[118] W. H. Winsborough and N. Li. Towards practical automated trust negotiation. In *IEEE 3rd Intl. Workshop on Policies for Distributed Systems and Networks*, 2002.

[119] G. Winskel. *The Formal Semantics of Programming Languages: An Introduction*. The MIT Press, 1993.

[120] L. Xiong and L. Liu. Building trust in decentralized peer-to-peer electronic communities. In *The 5th International Conference on Electronic Commerce Research(ICECR-5)*, October 2002.

[121] N. Yoshida, M. Berger, and K. Honda. Strong Normalisation in the π-Calculus. In *Proc. LICS'01*, pages 311–322. IEEE, 2001. The full version to appear in *Journal of Inf. & Comp.*.

[122] P. Zimmer. On the expressiveness of pure safe ambients. *Mathematical Structures in Computer Science*, 13:721–770, 2003.

[123] P. Zimmermann. *PGP Source Code and Internals*. The MIT Press, 1995.