UPPSALA
UNIVERSITET

Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 1397

Culling Concurrency Theory

Reusable and trustworthy meta-theory, proof techniques and separation results

JOHANNES ÅMAN POHJOLA

ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2016

ISSN 1651-6214
ISBN 978-91-554-9639-5
urn:nbn:se:uu:diva-297488

Dissertation presented at Uppsala University to be publicly examined in ITC/2446, Lägerhyddsvägen 2, Uppsala, Thursday, 22 September 2016 at 13:15 for the degree of Doctor of Philosophy. The examination will be conducted in English. Faculty examiner: Professor Uwe Nestmann (Technische Universität Berlin).

Abstract
Åman Pohjola, J. 2016. Culling Concurrency Theory. Reusable and trustworthy meta-theory, proof techniques and separation results. *Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology* 1397. 113 pp. Uppsala: Acta Universitatis Upsaliensis. ISBN 978-91-554-9639-5.

As concurrent systems become ever more complex and ever more ubiquitous, the need to understand and verify them grows ever larger. For this we need formal modelling languages that are well understood, with rigorously verified foundations and proof techniques, applicable to a wide variety of concurrent systems.

Defining modelling languages is easy; there is a stupefying variety of them in the literature. Verifying their foundations and proof techniques, and developing an understanding of their interrelationship with other modelling languages, is difficult, tedious and error-prone. The contributions of this thesis support these tasks in reusable and trustworthy ways, by results that apply to a wide variety of modelling languages, verified to the highest standards of mathematical rigour in an interactive theorem prover.

To this end, we extend psi-calculi - a family of process calculi with reusable foundations for formal verification - with several new language features. We prove that the bisimulation meta-theory of psi-calculi carries over to these extended settings. This widens the scope of psi-calculi to important application areas, such as cryptography and wireless communication. We develop bisimulation up-to techniques - powerful proof techniques for showing that two processes exhibit the same observable behaviour - that apply to all psi-calculi. By showing how psi-calculi can encode dynamic priorities under very strong quality criteria, we demonstrate that the expressive power is greater than previously thought. Finally, we develop a simple and widely applicable technique for showing that a process calculus adds expressiveness over another, based on little more than whether parallel components may act independently or not. Many separation results, both novel ones and strengthenings of known results from the literature, emerge as special cases of this technique.

Johannes Åman Pohjola, Department of Information Technology, Computing Science, Box 337, Uppsala University, SE-75105 Uppsala, Sweden.

ISSN 1651-6214
ISBN 978-91-554-9639-5
urn:nbn:se:uu:diva-297488 (http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-297488)

List of papers

This thesis is based on the following papers, which are referred to in the text by their Roman numerals.

I Johannes Borgström, Shuqin Huang, Magnus Johansson, Palle Raabjerg, Björn Victor, Johannes Åman Pohjola, and Joachim Parrow. Broadcast Psi-calculi with an Application to Wireless Protocols. In *Software and Systems Modeling*, volume 14(1), pages 201–216. Springer-Verlag, 2015. Extended version of Paper VII.

II Joachim Parrow, Johannes Borgström, Palle Raabjerg, and Johannes Åman Pohjola. Higher-order psi-calculi. In *Mathematical Structures in Computer Science*, volume 24(2). Cambridge University Press, 2014.

III Johannes Borgström, Ramūnas Gutkovas, Joachim Parrow, Björn Victor, and Johannes Åman Pohjola. A Sorted Semantic Framework for Applied Process Calculi. In *Logical Methods in Computer Science*, volume 12(1). 2016. Extended version of Paper VIII.

IV Johannes Åman Pohjola and Joachim Parrow. Priorities Without Priorities: Representing Preemption in Psi-Calculi. In *Proceedings Combined 21st International Workshop on Expressiveness in Concurrency and 11th Workshop on Structural Operational Semantics, EXPRESS 2014, and 11th Workshop on Structural Operational Semantics, SOS 2014, Rome, Italy, 1st September 2014*, volume 160 of *EPTCS*, pages 2–15. 2014.

V Johannes Åman Pohjola and Joachim Parrow. Bisimulation Up-To Techniques for Psi-Calculi. In *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs, Saint Petersburg, FL, USA, January 20-22, 2016*, pages 142–153. ACM, 2016.

VI Johannes Åman Pohjola and Joachim Parrow. The Expressive Power of Monotonic Parallel Composition. In *Proceedings of the 25th European Symposium on Programming, ESOP 2016*, volume 9632 of *Lecture Notes in Computer Science*. Springer-Verlag, 2016.

Reprints were made with permission from the publishers. The following additional papers are subsumed by the papers above, and are therefore not included in the thesis.

VII Johannes Borgström, Shuqin Huang, Magnus Johansson, Palle Raabjerg, Björn Victor, Johannes Åman Pohjola, and Joachim Parrow. Broadcast Psi-calculi with an Application to Wireless Protocols. In Gilles Barthe, Alberto Pardo, and Gerardo Schneider, editors, *Software Engineering and Formal Methods: SEFM 2011*, volume 7041 of *Lecture Notes in Computer Science*, pages 74—89. Springer-Verlag, 2011.

VIII Johannes Borgström, Ramūnas Gutkovas, Joachim Parrow, Björn Victor, and Johannes Åman Pohjola. A Sorted Semantic Framework for Applied Process Calculi (Extended Abstract). In Martín Abadi and Alberto Lluch-Lafuente, editors, *Trustworthy Global Computing - 8th International Symposium, TGC 2013, Buenos Aires, Argentina, August 30-31, 2013, Revised Selected Papers*, volume 8358 of *Lecture Notes in Computer Science*, pages 103–118. Springer-Verlag, 2013.

An earlier version of Paper VIII, entitled *Sorted Psi-calculi with Generalised Pattern Matching*, was presented at the *5th Interaction and Concurrency Experience (ICE) 2012*, Stockholm, Sweden. An earlier version of Paper V was presented at the *1st International Workshop on Meta Models for Process Languages (MeMo) 2014*, Berlin, Germany.

My contributions to co-authored papers

Paper I

I contributed about a third of the mechanical proofs of correctness of standard structural properties of bisimilarity, including the laws of scope extension and commutativity of binders. I have also done all the mechanical proofs pertaining to the LUNAR model and higher-order broadcast psi-calculi.

Paper II

I contributed almost all of the mechanical proofs of correctness of standard congruence and structural properties of bisimilarity. Mechanical proofs of all theorems pertaining to higher-order bisimilarity, canonical instances and the encoding of operators are wholly my work. I also came up with the idea of canonical instances and worked out the precise conditions under which the operator encodings are valid.

Paper III

I contributed mechanical proofs of correctness of standard congruence and structural properties of bisimilarity, preservation of well-formedness and backwards compatibility of pattern matching. I have also contributed some pen-and-paper proofs. These are the proofs pertaining to lifting of results about bisimilarity from trivially sorted calculi to sorted calculi, as well as operational correspondences with the polyadic pi-calculus, value-passing CCS and the polyadic synchronisation pi-calculus.

Papers IV-VI

I am the principal author and investigator in all parts of these papers.

Relationship with previous thesis

This PhD thesis can be seen as a continuation of my licentiate thesis [ÅP13]. In the Swedish higher-education system the licentiate is a thesis similar in structure to a PhD thesis, that students are encouraged to write half-way through the PhD program. Some of the material in this PhD thesis was already published in my licentiate thesis:

- Papers I–II.
- An earlier version of Paper III, namely Paper VIII.
- A technical report [ÅPBP+13] outlining the extension of psi-calculi with priorities used in Paper IV (but not its encoding back into psi-calculi without priorities).
- An unpublished draft of the respectfulness-based approach to bisimulation up-to techniques for psi-calculi, that we outline in Section 5 of Paper V.

Additionally, the comprehensive summary is partially based on the comprehensive summary of the licentiate thesis.

Svensk sammanfattning

Moderna datorsystem är parallella system: de består av flera parallella delprocesser som kan pågå samtidigt, oberoende av varandra. Internet är ett svindlande stort parallellt system. Datorer som till antalet vida överstiger jordens mänskliga befolkning använder internet varje dag för att kommunicera med varandra; det blir allt svårare att tänka sig hur samhället skulle kunna fungera annars. Internettjänster som för användaren framstår som sammanhållna är i själva verket storskaliga distribuerade system. De kan bestå av enorma mängder datorer spridda över datacenter i hela världen; en enda Google-sökning kan hanteras av tusentals samspelande datorer. Du har sannolikt ett parallellt system din ficka — moderna telefoner tar emot sms, spelar musik och visar webbsidor samtidigt. Var och en av dessa arbetsuppgifter utförs vanligtvis av flera samarbetande delprogram som körs samtidigt.

Parallella system styr över ständigt fler och ständigt viktigare aspekter av våra liv. Därför blir det viktigt att verifiera systemen, dvs. förvissa sig om att de verkligen beter sig som avsett.

När vi verifierar system använder vi modeller, som förhåller sig till datorsystem som en ritning förhåller sig till en byggnad. Man kan analysera modellerna, för att göra förutsägelser om huruvida systemen kommer uppvisa något oönskat beteende; till exempel säkerhetsläckor eller att systemet kan låsa sig. Modellerna uttrycks med hjälp av särskilda modellspråk. Byggnadsritningar har sitt eget modellspråk, nämligen de konventioner som styr vilka symboler, linjer, måttenheter osv. som används, och hur de ska utläsas. Sådana konventioner gör det möjligt att beskriva ett datorsystem på ett sätt som är entydigt, men som ändå abstraherar bort från ovidkommande detaljer.

Ett grundläggande modellspråk för verifiering av parallella system är pi-kalkyl, som tack vare att det är enkelt och renodlat men ändå uttrycksfullt haft ett stort inflytande på forskningen. Men när forskare ska studera ett nytt fenomen eller verifiera ett nytt system använder vi sällan pi-kalkyl i sin ursprungliga form. Detta eftersom den inte har de språkfinesser som behövs för alla tänkbara tillämpningar. Istället skapar vi en mer eller mindre genomtänkt variant, skräddarsydd för tillämpningen ifråga. Det är nästan som om pi-kalkylen förökar sig. En utomstående betraktare uppfattar lätt vetenskapsområdet som ett spretigt virrvarr av snarlika men distinkta modellspråk. Till och med områdets anhängare liknar ofta situationen vid en djungel.

En viktig egenskap som en användbar processkalkyl bör ha är kompositionalitet, dvs. att när ett stort system ska verifieras, kan man dela upp systemet i mindre beståndsdelar som kan verifieras separat. Moderna datorsystem är så

ohemult stora att ingen människa själv kan bygga dem, eller ens förstå dem, i sin helhet. Därför är de uppbyggda av flera mindre delsystem, som kan byggas mer eller mindre separat, för att sedan sättas ihop till ett större system. Kompositionalitet är viktigt eftersom det möjliggör motsvarande uppdelning även på verifieringsnivå. Även små och till synes oskyldiga ändringar i definitionen av en kompositionell processkalkyl kan göra den icke-kompositionell.

Att definiera en ny processkalkyl är lätt, men en hel del arbete behöver utföras för att visa att kalkylen är användbar. Kalkylens kompositionalitet måste fastställas. Matematiska bevismetoder för att underlätta verifieringen behöver utvecklas. Kalkylens uttrycksfullhet bör studeras, så att vi vet vad den kan användas till, och huruvida den tillför något nytt. Var och en av dessa uppgifter kräver svåra och arbetsintensiva matematiska bevis.

I den här avhandlingen presenterar vi följande bidrag, som alla syftar till att underlätta arbetet med att visa att nya processkalkyler är användbara:

- Vi utökar psi-kalkylerna — en familj av kompositionella processkalkyler — med flera nya språkfinesser. Vi visar att dessa utökningar bevarar kompositionalitet. Detta medför att psi-kalkyler kan användas för viktiga tillämpningsområden såsom kryptografi och trådlösa nätverk.
- Vi utvecklar en kraftfull bevisteknik för att visa att två processer har samma beteende. Tekniken är tillämpbar på alla psi-kalkyler.
- Vi visar att psi-kalkyler är mer uttrycksfulla än vad man hittills trott, genom att visa hur de kan användas för att uttrycka modeller där vissa beteenden prioriteras över andra.
- Vi utvecklar en metod för att visa att en processkalkyl är mer uttrycksfull än en annan. Metoden är enkel och har många tillämpningar.

Vi måste kunna lita på verifiering. Därför måste vi också kunna lita på de grunder som verifieringen stödjer sig på, såsom kompositionalitet. Så det är viktigt att vi inte gör några misstag när vi utvecklar våra teorier. I görligaste mån maskinkollar vi därför alla våra resultat med en så kallad teorembevisare, ett datorprogram som kan bevisa matematiska påståenden, och som inte lämnar något som helst utrymme för att slarva med detaljer.

När man skriver ett matematiskt bevis för en mänsklig läsare brukar man utelämna många detaljer. Det är bra: de behövs inte för att förmedla en övergripande förståelse för varför något är sant, och en kompetent läsare bör själv kunna fylla i detaljerna i mån av intresse. Men om vi vill vara tvärsäkra på att beviset är korrekt, då finns det uppenbara risker. Utelämnade bevissteg kan te sig enkla och intuitiva vid en ytlig anblick, men svåra eller till och med felaktiga om man tittar närmare. Att skriva bevis som inte utelämnar några detaljer löser inte problemet — åtminstone inte så länge både upphovsman och läsare är människor, felbara som vi ju är. Lösningen är att göra bevisen maskinläsbara. Den datoriserade teorembevisaren godtar bara de härledningar som följer matematikens grundlagar; för den är inget enkelt eller intuitivt. Det kan vara en arbetsintensiv metod; resultaten i den här avhandlingen understöds av hundratusentals rader bevis. Men datorsystem hanterar idag arbetsuppgifter

där korrekt beteende är en fråga om liv eller död, så det är värt den extra mödan att vara extra säker.

Acknowledgements

Thanks first and foremost to my supervisor, Joachim Parrow, for all his patience, care and generosity. None of this would have been possible without him. He has set me free to meander about and learn from my own mistakes, yet has always been there to offer guidance, hear out my latest half-baked idea, or cover my writings in ink.

Björn Victor, my co-supervisor, has also been a consistent source of good advice. Thanks to him, and the other past and present members of the mobility group, for the many interesting discussions and fun times.

Thanks to the many anonymous reviewers who have rejected my papers over the years, for teaching me the difference between a paper I want to write and a paper you want to read.

Thanks to (in alphabetical order) Sofia Cassel, Camilla Malm, Joachim Parrow, Martin Rynoson, and Björn Victor, for reading and commenting on drafts of this thesis, or parts thereof.

Thanks again to Camilla for all the love, for helping me stay grounded, for making me Laphroaig-flavoured ice cream even though she hates whisky with a passion, and many such wifely contributions.

Thanks to to all my friends both at and outside work, and all my family, who have without exception been very supportive.

Finally, a special thanks to Linnea, who — under the impression that it is some kind of fairy tale anthology — has been by far the most enthusiastic reader of my licentiate thesis. If this PhD thesis has a readership half as enthusiastic or half as cute, I will consider myself very lucky.

Contents

1. Introduction

Culling is a way of controlling the population of wild animals. Elk, for example, have few natural predators; left to their own devices they would breed to an unsustainably large population. Potentially, this may result in damage to forests and crops, decreased biodiversity, and more frequent traffic accidents. An annual culling of the Swedish elk population, namely the hunting season, is motivated by precisely these concerns. But the reader need not worry; no animals were harmed during the writing of this thesis. Rather, we consider ways to achieve a more sustainable population of concurrency theories, or in other words, ways of culling concurrency theory.

Concurrency is the phenomenon of events happening at the same time. In computer science, a concurrent process is one that consists of several parts that may go about their computations independently of each other. These parts may also interact by e.g. passing messages to each other, and share resources. A sequential process, on the contrary, is characterised by events happening one at a time, in a pre-defined order. At the dawn of the computer age, most computing was sequential; today, sequential computing is an increasingly endangered species. The Internet is a concurrent process on a dizzying scale. Computing devices far outnumbering the human population on earth use it every day to communicate with each other, and it gets increasingly difficult to imagine how society could possibly function without it. Internet services that present themselves to the user as a single, monolithic service are in fact often large-scale distributed systems, comprising thousands of computing devices in multiple datacenters all over the world; just a single Google search may involve thousands of machines. There may well be a concurrent system in your pocket — modern phones receive text messages, play music and display web sites concurrently. Each of these tasks is usually accomplished by a multi-threaded program, i.e. a program composed of several sequential sub-programs running concurrently. In short, it is concurrent systems all the way down.

Concurrent processes are somewhat chaotic by nature, and hence significantly more difficult to reason about than sequential systems. Picture yourself blindfolded, seated by the dinner table, knife and fork in hand, with a bowl of pasta in front of you. In this scenario, the task of eating the pasta is a lot like a sequential process: first you take a bite, then you take another, and so on. Now, picture ten more people seated around the table — each with their own knife, fork and blindfold — concurrently eating pasta from the same bowl as you. The difficulties of eating pasta introduced in this scenario, compared to when you were seated alone, are largely the same as the added difficulties involved

with concurrent systems over sequential ones. The blindfold represents the fact that a process is unaware of what other processes are doing concurrently, unless it is explicitly communicated.

In the worst case, all your fellow eaters are selfish boors who abide by no etiquette; to even grab a bite becomes almost impossible if you must first fight your way through an unseen tangle of arms, hands and cutlery all trying to fend you off. Hence the first order of business is to impose some etiquette — or in computer science terms, a *protocol* — on the eaters. A simple protocol might be to enforce that only one eater is active at a time, by requiring (1) that when an eater begins to eat or finishes, she announces this to the other eaters; and (2) that when one has announced that she is eating, others do not attempt to eat. To guarantee that systems behave as intended, protocols must be carefully designed and analysed to address concurrency issues such as the following.

- In the above protocol, what happens when two or more eaters simultaneously announce that they begin eating?
- How can we be sure that everyone eventually gets to eat?
- How can we ensure that eaters do not get stuck waiting for one another in perpetuity? What if while eating, an eater falls into a carbohydrate coma before announcing she is finished?
- What happens if one or more eaters fail to hear an announcement?
- What happens if an eater fails to follow the protocol, whether intentionally or not? How can we prevent unauthorised third parties from accessing the pasta, or interfering with correct protocol execution?

Even for the relatively simple task of eating pasta, addressing such issues is subtle and fraught with difficulty; our proposed protocol above is hardly satisfactory. Real-world concurrent systems must handle tasks that are orders of magnitude more complex, and failure to address the above issues can have life-and-death consequences: one would rather not drive a car where these issues are not well addressed.

Gaining confidence that such systems are indeed correct is clearly an important problem. There are two main ways to go about this: testing and formal verification.

In testing, we run the system and see whether anything goes wrong. While testing is an indispensable tool, its limitations become more apparent in a concurrent setting. For a sequential system, we can reasonably expect the same test to always yield the same result. Because interactions between the parts of a concurrent system may unfold in many different ways, a system where testing found no errors today may still exhibit errors on the exact same test runs tomorrow.

In formal verification, we construct a mathematical model of the system, and show that the model exhibits no undesired behaviour through rigorous mathematical proof. Where testing can justify the claim “we found no errors”, formal verification can justify the claim “there are no errors”, at least up-to

the faithfulness of the system model. We distinguish between low-level verification and high-level verification. In low-level verification, we ask whether a particular system *implementation* is correct. In high-level verification, we ask whether a particular system *design* is correct, abstracting away from the details of the implementation.

To verify concurrent systems, we need mathematical and logical tools to help us reason formally about concurrency. Concurrency theory, then, is simply an umbrella term for such tools. In this thesis, the emphasis is on modelling languages geared towards high-level verification of concurrent systems, called *process calculi* or *process algebras*, and the reasoning tools associated with them. A fundamental model of concurrency in this tradition is the pi-calculus. It is simple and parsimonious, and its impact on the field has been tremendous.

Yet when process calculists endeavour to study a new phenomenon or verify a new system, we rarely use the pi-calculus. Instead, a more or less ad-hoc variant of the pi-calculus is constructed specifically for the application at hand. It is almost as if the pi-calculus breeds. Outside observers can be forgiven for coming away with the impression that the field is a sprawling mess of similar but distinct formalisms, whose interrelationship is poorly understood; the jungle metaphor is frequently used to describe the situation, even by the field's adherents.

Does this situation call for culling measures? Yes, but not in quite the same way or for quite the same reasons as with elk. The unsustainable population that we wish to cull is the theory about process calculi, rather than the process calculi themselves. Every time a new process calculus is defined, a certain amount of theoretical groundwork is needed to show that it is useful. Formal verification of system models written in the calculus must rest on firm foundations. Proof techniques to facilitate this formal verification need to be developed. The expressiveness of the calculus should be studied, to understand whether it adds something new over well-known languages such as the pi-calculus. Each of these tasks requires mathematical proofs that are difficult, tedious and error-prone; carrying them out for a new process calculus is the subject of many PhD theses.

Theories cannot be culled from a population in the same way that elk can: once a theory has been developed it cannot well be undeveloped. Instead, we cull by subsumption. The idea is to develop reusable theory. We want important results to be established once and for all to hold for many process calculi, rather than over and over again for particular process calculi.

In this thesis, we address the challenge of culling concurrency theory in the following main ways:

- We extend psi-calculi — a family of process calculi with reusable foundations for formal verification, that encompasses the pi-calculus and many of its extensions — with several new language features. This ex-

tends the scope of these foundations to encompass important application areas, such as cryptography and wireless communication.
- We develop a powerful proof technique for showing that two processes exhibit the same observable behaviour, that applies to all psi-calculi.
- We show that the expressive power of psi-calculi is greater than previously thought, by showing how it can encode prioritised behaviour.
- We develop a simple and widely applicable technique for showing that a process calculus adds expressiveness over another.

An abundance of concurrency theories is unlikely to result in damage to forests and crops, or decreased biodiversity. But it may well lead to more frequent traffic accidents, if theory that is not carefully developed is used for verification of automotive systems. Thus, it is important that there are no mistakes in our theories. For this reason, we go to great lengths to make sure that as many of our proofs as possible are machine-checked by an *interactive theorem prover*, a computer program that proves mathematical theorems and leaves no room whatsoever for skimping on details.

Mathematical proofs written for a human audience typically leave gaps in the argumentation: inferences are made with vague and underspecified justification, or none at all, under the tacit assumption that a competent reader could fill in the missing details if so inclined. This style of proof is useful if we want to convey an understanding of *why* something is true; when we want absolute certainty that our proof is correct, it is somewhat unreliable. It may be that the missing inference steps, while intuitively obvious at a glance, are in fact far from obvious or even faulty.

We could write full proofs from first principles, but as long as both author and audience are human, we have failed to eliminate the main source of error: human fallibility. The remedy, then, is to write our proofs for a machine audience. To a theorem prover, nothing is obvious, and a proof will be rejected unless every last detail can be justified from first principles.

Of course, theorem provers need to be significantly more trustworthy than the average computer program if we are to reap any benefit from them. In his famous Turing Award lecture, Tony Hoare remarked that

> [t]here are two ways to write code: write code so simple there are obviously no bugs in it, or write code so complex that there are no obvious bugs in it.

Theorem provers are carefully designed so that the part that needs to be trusted is in the former category; a luxury that concurrent systems rarely enjoy.

1.1 Outline

The remainder of this thesis is structured as follows.

Chapter 2: Background
This chapter recapitulates background material. It is intended for an audience with a general orientation in computer science, but without expert knowledge in process calculi or theorem proving.

Section 2.1 explains how we model concurrent processes and give them meaning, through the example of Milner's Calculus of Communicating Systems (CCS), an ancestor of the pi-calculus.

Section 2.2 introduces bisimilarity, the most studied behavioural equivalence in concurrency theory. We explain and exemplify the associated proof method for showing that processes are equivalent, give an alternate lattice-theoretic characterisation, and discuss its relation to other behavioural equivalences.

Section 2.3 reviews the literature on expressiveness, with emphasis on explaining and motivating the many different ways of answering the question: what constitutes a good encoding of one concurrency model into another?

Section 2.4 recapitulates nominal logic, a formal approach for reasoning about syntax with binders.

Section 2.5 introduces interactive theorem proving, an approach for developing highly trustworthy formal mathematical proofs with the aid of a computer. In particular we emphasise Isabelle, our theorem prover of choice; we explain the main principles behind it, and illustrate the proof development process with an extended example about the aforementioned lattice-theoretic characterisation of bisimilarity.

Section 2.6 recapitulates psi-calculi, a family of concurrency models in the tradition of CCS and the pi-calculus whose bisimulation meta-theory has been verified in Isabelle. We explain its syntax and semantics, and recapitulate the main definitions and results pertaining to the bisimulation meta-theory.

Chapter 3: Contributions
This chapter summarises the novel contributions of this thesis. The reader will need cursory knowledge of all areas discussed in Chapter 2.

Section 3.1 introduces four extensions of psi-calculi, in order to increase its expressiveness and modelling convenience: broadcast communication, higher-order data, generalised pattern matching, and a sort system.

Section 3.2 introduces a technique for simplifying bisimulation proofs in psi-calculi, with particular emphasis on its benefits for proof engineering (the practice of developing and maintaining large formal proofs).

Section 3.3 introduces priorities as another extension of psi-calculi, along with an encoding of priorities into the original psi-calculi.

Section 3.4 introduces a simple and general method for showing that one model of concurrency cannot encode another, based on little more than whether the latter allows parallel components to act independently or not. Several non-encodability results, both novel and from the literature, are shown to emerge as special cases.

Chapter 4: Related work
In this chapter we review the scientific literature on topics that are related to the contributions of this thesis.

Chapter 5: Conclusion
Here we conclude by discussing our contribution, future work and impact.

Back matter
Finally, Papers I–VI comprise the contributions that we summarised in Chapter 3. They are written with an expert audience in mind, though my hope is that non-experts should also find them accessible after reading the executive summary.

2. Background

In this chapter, we recapitulate the background material necessary for understanding and contextualising the contributions of this thesis.

2.1 Modelling concurrency

A *process* is a system in which *events* may happen. Depending on the application, these notions can be instantiated to be almost anything: molecules and chemical reactions, networks and packages, CPUs and machine-code instructions, and so on. In this thesis we will often reason about processes and events in the abstract, without being overly specific about what they might represent.

A simple formalism that encompasses both sequential and concurrent processes is *labelled transition systems* [Kel75], often abbreviated *LTS*, where transitions between the states of a system are labelled with events:

Definition 1 (Labelled transition systems)**.** *A* labelled transition system *is a triple* $(\Sigma, \rightarrow, A)$*, where* Σ *is a set of* states *(or* processes*), A is a set of* labels *(or* actions*), and* $\rightarrow \subseteq \Sigma \times \alpha \times \Sigma$ *is a set of* transitions*.*

We will write $P \xrightarrow{\alpha} Q$, meaning that from the state P we can do an action α leading to the state Q, for $(P, \alpha, Q) \in \rightarrow$. Note that we make no distinction between a process and the state of a process: updating the state of a process is the same as transitioning to a new process. When the successor state is unimportant we write $P \xrightarrow{\alpha}$ to mean $\exists P'.P \xrightarrow{\alpha} P'$. This notation extends to sequences of labels in the natural way, so that e.g. $P \xrightarrow{\alpha} \xrightarrow{\beta}$ means $\exists P'.P \xrightarrow{\alpha} P' \wedge P' \xrightarrow{\beta}$. For the absence of transitions we write $P \not\xrightarrow{\alpha}$ to mean $\neg\, P \xrightarrow{\alpha}$.

As an example of a labelled transition system for modelling concurrency, we will present a simplified version of Milner's *calculus of communicating systems* [Mil80], henceforth abbreviated *CCS*. In CCS, we model parallel processes that may engage in one-on-one synchronisation on abstract communication mediums called *ports*.

We assume a set of ports, ranged over by $a, b, c, \ldots, x, y, z$. Further, we assume a *complement* operation $\bar{\cdot}$, which is a function from ports to ports such that $\overline{\bar{a}} = a$ for all ports a. Synchronisations in CCS are between a port and its complement. The actions of CCS are the ports, plus the disjoint special symbol τ (called the *silent action*). We use α, β to range over actions. In order to

drive the intuition, we may think of $\overline{a}$ as emitting a signal, and a as receiving a signal. However, note that CCS makes no formal distinction between emission and receipt. We might just as well have used the dual intuition, with the roles of $\overline{a}$ and a interchanged; what matters is that they are two complementary ends of a synchronisation action.

We proceed by defining the processes of CCS.

Definition 2. *The processes of CCS are defined by the following grammar:*

$$\begin{array}{rcll} P,Q,R & := & \alpha.P & \textit{Prefix} \\ & & 0 & \textit{Nil} \\ & & P \mid Q & \textit{Parallel} \\ & & P+Q & \textit{Choice} \\ & & (\nu x)P & \textit{Restriction} \end{array}$$

0 is a process that does nothing. $\alpha.P$ is a process that does the action α, then continues as P. We will often abbreviate $\alpha.0$ as simply α. The parallel execution of P and Q is denoted by $P \mid Q$. The sum $P+Q$ may behave as either P or Q. $(\nu x)P$ means that x is restricted to P; hence, in $Q \mid (\nu x)P$ no interaction between Q and P on the port x is possible.

Example 3. *A man lives alone in a house with a broken thermometer. The thermometer (denoted T) will — independently of the actual weather — non-deterministically emit a temperature reading of either* **warm** *or* **cold***:*

$$T \triangleq \overline{\mathbf{warm}} + \overline{\mathbf{cold}}$$

Before going from his house in the morning, the man (denoted M) obtains a reading from his thermometer to determine whether to put on a jacket:

$$M \triangleq \mathbf{warm}.\overline{\mathbf{go}} + \mathbf{cold}.\overline{\mathbf{jacket}}.\overline{\mathbf{go}}$$

Since the thermometer is inside the house (denoted H), readings may not be obtained from outside:

$$H \triangleq (\nu\,\mathbf{warm})(\nu\,\mathbf{cold})(T \mid M)$$

The semantics (i.e. the transition relation) of CCS is given by means of a *structured operational semantics*, often abbreviated as *SOS*. The technique was first introduced by Plotkin [Plo81]. In an SOS, the transitions from a state are defined inductively in terms of the transitions of its components.

Definition 4. *The transitions of CCS are those that can be inferred by repeatedly applying the rules of Table 2.1.*

$$\textsc{Pre}\ \frac{}{a.P \xrightarrow{a} P} \qquad \textsc{Res}\ \frac{P \xrightarrow{\alpha} P' \qquad \alpha \neq b \qquad \overline{\alpha} \neq b}{(\nu b)P \xrightarrow{\alpha} (\nu b)P'}$$

$$\textsc{Sum}\ \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'} \qquad \textsc{Com}\ \frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{\overline{a}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\textsc{Par}\ \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$$

Table 2.1. *Structured operational semantics of CCS. Symmetric versions of the rules* PAR *and* SUM *are elided.*

The rules are written with premises on top, and conclusions on the bottom. The PRE rule means that the process $a.P$ may always synchronise with the environment on the port a. The COM rule says that two parallel processes, each offering synchronisation on the other's complement, may synchronise; and that this synchronisation is a silent action that third-party processes cannot observe or interact with. The PAR rule means that either one of two parallel processes may perform independent actions without synchronising with the other. Note that with the symmetric version PAR elided from Table 2.1, we may infer $P \mid Q \xrightarrow{\alpha} P' \mid Q'$ whenever we may infer $Q \mid P \xrightarrow{\alpha} Q' \mid P'$. The RES rule enforces that in the process $(\nu a)P$, any synchronisation on a is an internal synchronisation between two parallel components of P. Finally, the SUM rule and its symmetric counterpart allows the process $P+Q$ to do anything that either P or Q may do. Once an action originating from e.g. P has been performed, the option to act as Q is forfeit.

Example 5. *We recall the processes M and H representing the man and house of Example 3. If the man M obtains a* **warm** *reading from his thermometer T, we can use the rules of Table 2.1 to infer the following transition from $T \mid M$:*

$$\textsc{Com}\ \frac{\textsc{Sum}\ \dfrac{\textsc{Pre}\ \dfrac{}{\overline{\mathbf{warm}}.0 \xrightarrow{\overline{\mathbf{warm}}} 0}}{T \xrightarrow{\overline{\mathbf{warm}}} 0} \qquad \textsc{Sum}\ \dfrac{\textsc{Pre}\ \dfrac{}{\mathbf{warm}.\overline{\mathbf{go}} \xrightarrow{\mathbf{warm}} \overline{\mathbf{go}}}}{M \xrightarrow{\mathbf{warm}} \overline{\mathbf{go}}}}{T \mid M \xrightarrow{\tau} 0 \mid \overline{\mathbf{go}}}$$

By applying the RES *rule twice we can extend this derivation to H:*

$$H \xrightarrow{\tau} (\nu\,\mathbf{warm})(\nu\,\mathbf{cold})(0 \mid \overline{\mathbf{go}})$$

Through a similar derivation, we can infer that the man may then go:

$$(\nu\,\mathbf{warm})(\nu\,\mathbf{cold})(0 \mid \overline{\mathbf{go}}) \xrightarrow{\overline{\mathbf{go}}} (\nu\,\mathbf{warm})(\nu\,\mathbf{cold})(0 \mid 0)$$

We can also infer a transition sequence

$$H \xrightarrow{\tau} \xrightarrow{\overline{\mathbf{jacket}}} \xrightarrow{\overline{\mathbf{go}}}$$

corresponding to the case where the thermometer emits a **cold** *reading.*

We will sometimes be interested in unlabelled transition relations, called *reduction relations*. A reduction relation describes the independent behaviour of a process, i.e. the behaviour that a process can engage in without needing to synchronise with the outside world. For CCS the reductions are simply the τ-labelled transitions:

$$P \longrightarrow P' \quad \triangleq \quad P \xrightarrow{\tau} P'$$

We will often be more interested in whether a particular state is reachable, and less interested in how many transitions must be taken to do so. For this purpose we introduce *weak* reductions, written $P \Longrightarrow P'$, meaning that there is a way to start from P and end up in P' after some number of reductions. Formally, $\Longrightarrow$ is the reflexive and transitive closure of $\longrightarrow$.

The two main points so far are as follows. First, that CCS is a *compositional* model of concurrency: models of smaller systems can easily be composed to form larger systems by means of algebraic operators such as $+$ and $|$. Second, that CCS is a *formal* model of concurrency: the transition relation unambiguously defines exactly the set of all possible behaviours that a CCS process may exhibit. This flavour of concurrency model is called a *process calculus* or *process algebra*. While the term is rather broad, process calculi typically emphasise interaction, compositionality and algebraic reasoning techniques. The work in this thesis is in the tradition of process calculi that originates with CCS.

However, there are countless other models of concurrency, stressing different aspects of concurrency at different levels of abstraction and formality. To cite just a few examples, *Petri nets* [Pet66] de-emphasise compositionality while putting a greater emphasis on control flow, the *Actor model* [HBS73] emphasises message passing and how processes may respond upon receiving messages, and *temporal logics* such as LTL [Pnu77] and TLA [Lam94] strongly emphasise how events are structured in time. Even within process calculi there is a long and diverse history, with three main branches stemming from CCS, CSP [BHR84] and ACP [BK84]; for an historical overview we refer to Baeten [Bae05].

2.2 Bisimilarity

A natural question to ask is what it means for two processes to have the same behaviour, leading to the topic of *behavioural equivalences*. Intuitively, a behavioural equivalence equates two processes if and only if they have the same behaviour. The exact nature of the equivalence of course depends on what we mean by "behaviour". Here we take the view that the behaviour of a process is described by the labels that its transitions may exhibit. One of the canonical behavioural equivalences is then *bisimulation*, which was first considered by Park [Par81] as an equivalence relation between various kinds of automata. Adapted to arbitrary labelled transition systems, the definition is as follows:

Definition 6 (Bisimulation)**.** *A* bisimulation relation *for an LTS* $(\Sigma, \rightarrow, \alpha)$ *is a set* $\mathscr{R} \subseteq \Sigma \times \Sigma$ *such that for all* $(P,Q) \in \mathscr{R}$*:*

1. *For all* P'*,* α *such that* $P \xrightarrow{\alpha} P'$*, there exists* Q' *such that* $Q \xrightarrow{\alpha} Q'$ *and* $(P',Q') \in \mathscr{R}$*.*
2. *For all* Q'*,* α *such that* $Q \xrightarrow{\alpha} Q'$*, there exists* P' *such that* $P \xrightarrow{\alpha} P'$ *and* $(P',Q') \in \mathscr{R}$*.*

Intuitively, two processes are related by a bisimulation if for each transition that one of them can take, the other can take a transition with the same label, and the resulting processes are again related by the bisimulation.

We are interested in whether pairs of processes are related by *some* bisimulation relation, and somewhat less interested in which particular bisimulation relations that might be. The notion of *bisimilarity* is useful for reasoning on this level of abstraction.

Definition 7 (Bisimilarity)**.** Bisimilarity*, denoted* $\dot{\sim}$*, is a binary relation on processes. Two processes* P,Q *are* bisimilar*, written* $P \dot{\sim} Q$*, if there exists a bisimulation relation* $\mathscr{R}$ *such that* $(P,Q) \in \mathscr{R}$*.*

The importance of bisimulation relations, then, is that they offer a proof technique for bisimilarity- In order to establish $P \dot{\sim} Q$, we exhibit a bisimulation relation that includes the pair (P,Q). By this proof method, it is straightforward to establish that bisimilarity enjoys the following basic properties:

Theorem 1.

1. *Bisimilarity is a bisimulation relation.*
2. *Bisimilarity is reflexive, symmetric and transitive.*
3. *CCS parallel composition is commutative and associative:*

$$P \mid Q \dot{\sim} Q \mid P \qquad\qquad P \mid (Q \mid R) \dot{\sim} (P \mid Q) \mid R$$

Proof.

1. For any P, Q, α such that $P \dot{\sim} Q$ and $P \xrightarrow{\alpha} P'$, we must find Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \dot{\sim} Q'$. By definition of $\dot{\sim}$ there exists a bisimulation $\mathscr{R}$ such that $(P,Q) \in \mathscr{R}$. Since $P \xrightarrow{\alpha} P'$, by Definition 6 there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $(P',Q') \in \mathscr{R}$. Since $\mathscr{R}$ is a bisimulation relation, $P' \dot{\sim} Q'$.
2. By showing that the following are bisimulation relations:

   $$\{(P,P) : \mathbf{true}\} \qquad \{(P,Q) : Q \dot{\sim} P\} \qquad \{(P,Q) : \exists R.\, P \dot{\sim} R \wedge R \dot{\sim} Q\}$$

   To see that $\{(P,P) : \mathbf{true}\}$ is a bisimulation relation, note that proof obligation arising from Definition 6 is that every process can take the same transitions as itself.
   For symmetry, let $(R,S) \in \{(P,Q) : Q \dot{\sim} P\}$ with $R \xrightarrow{\alpha} R'$. We must find S' such that $S \xrightarrow{\alpha} S'$ and $(R',S') \in \{(P,Q) : Q \dot{\sim} P\}$. We have $S \dot{\sim} R$ and, since $\dot{\sim}$ is a bisimulation relation, by Definition 6 there exists S' such that $S \xrightarrow{\alpha} S'$ and $S' \dot{\sim} R'$. Hence $(R',S') \in \{(P,Q) : Q \dot{\sim} P\}$.
   A similar argument applies to the transitivity proof.
3. By showing that the following are bisimulation relations:

   $$\{(P \mid Q, Q \mid P) : \mathbf{true}\} \qquad \{(P \mid (Q \mid R), (P \mid Q) \mid R) : \mathbf{true}\}$$

   For any transition $P \mid Q \xrightarrow{\alpha} P'$, we may derive $Q \mid P \xrightarrow{\alpha} Q'$ where Q' is exactly P' with its two outermost parallel components swapped, by taking the derivation of $P \mid Q \xrightarrow{\alpha} P'$ and changing the last rule application as follows. If the last rule is PAR we apply instead its symmetric counterpart; if it is COM, we apply COM with the places of P and Q switched.
   A similar argument applies to the associativity proof. □

Bisimilarity often makes distinctions that are too fine for practical purposes. For example, consider again the broken thermometer scenario of Example 3. Since the temperature reading is unobservable from outside the house, and independent of the actual weather, we might argue that the observable behaviour of the system would be the same if the man, after some introspection, nondeterministically decides whether to put on a jacket or not without checking the thermometer. In this scenario, we model the man as:

$$M' \triangleq \tau.(\overline{\mathbf{jacket}}.\overline{\mathbf{go}} + \overline{\mathbf{go}})$$

and the house as

$$H' \triangleq (\nu\, \mathbf{warm})(\nu\, \mathbf{cold})(T \mid M')$$

With bisimilarity, we may prove that in this setting, the house and the man are bisimilar: $H' \dot{\sim} M'$. We might also expect both house models H and H' to

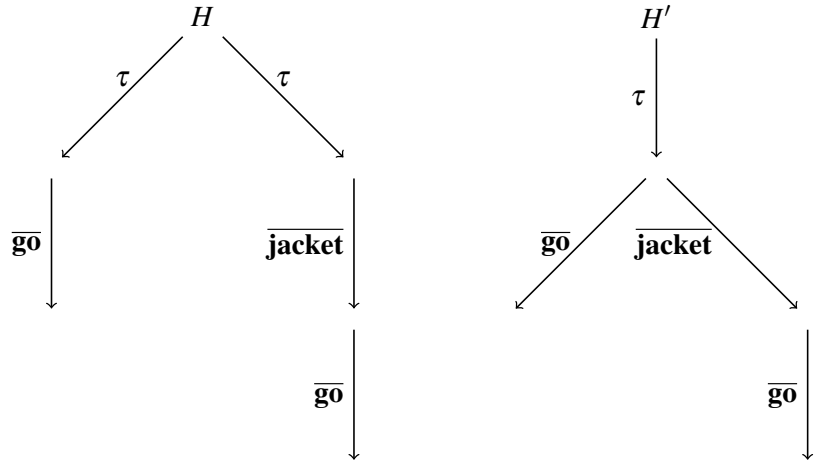

Figure 2.1. The transition behaviour of the processes H and H'.

be bisimilar, but this is not the case. The difference in transition behaviour is illustrated by Figure 2.1. After taking an initial τ transition, H will reach a state where the man is committed to either wearing a jacket or not; H' however remains uncommitted. This leads to an observable difference in behaviour since in an environment that declines to offer a **jacket**, the man might be trapped in H but not in H'. Other equivalences may relate H and H'. For an example, they are *trace equivalent*, which means that the sequences of labels that runs of H and H' may exhibit are the same. Formally, two processes P and Q are trace equivalent if for all $\alpha_0, \alpha_1, \ldots, \alpha_n$

$$P \xrightarrow{\alpha_0} \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} \quad \text{iff} \quad Q \xrightarrow{\alpha_0} \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n}$$

Even when we are primarily concerned with a coarser equivalence, bisimilarity may still be of interest. It turns out that bisimilarity is included in practically all other behavioural equivalences [vG90]. Hence, establishing that P and Q are bisimilar also establishes that they are e.g. trace equivalent, and we may use the bisimulation proof method as an (incomplete) technique for proving trace equivalence. For an example, since we showed that CCS parallel composition is associative and commutative, we obtain for free the fact that scores of coarser behavioural equivalences satisfy the same equations. The fact that we chose to work with bisimilarity saves us the trouble of having to re-prove the result in these settings.

While the above presentation of bisimilarity has the advantage of being standard and fairly straightforward, there are many others. The following lattice-theoretic definition appears to originate with Milner and Park [San09], and will prove useful when we consider improvements of the bisimulation proof method in Section 3.2. We first define the auxiliary function b, called the *bisim-*

ilarity functional. Intuitively, $b(\mathscr{R})$ is the set of process pairs from whence all outgoing transitions lead to pairs in $\mathscr{R}$.

Definition 8 (Bisimilarity functional)**.** *The* bisimilarity functional*, denoted b, is a function from relations on processes to relations on processes, and is defined as*

$$\begin{aligned} b(\mathscr{R}) \quad \triangleq \quad & \{(P,Q) : \\ & \forall P', \alpha.\, P \xrightarrow{\alpha} P' \quad \Rightarrow \quad \exists Q'.\, Q \xrightarrow{\alpha} Q' \wedge (P',Q') \in \mathscr{R} \\ & \quad \wedge \\ & \forall Q', \alpha.\, Q \xrightarrow{\alpha} Q' \quad \Rightarrow \quad \exists P'.\, P \xrightarrow{\alpha} P' \wedge (P',Q') \in \mathscr{R}\} \end{aligned}$$

For a CCS example, $b(\{(0,0)\})$ is the set of process pairs that can transition only to 0, in single steps with the same labels. These pairs are (a,a), (b,b), $(a+a,a)$, $(a,a+(\nu b)b)$, $(a+b,b+a)$, and so on.

The object of interest here is not the functional itself, but its *greatest fixed point*, denoted $\mathbf{gfp}(b)$. Without diving too deep into lattice theory, it suffices to know that a *fixed point* of b is any relation $\mathscr{R}$ such that $\mathscr{R} = b(\mathscr{R})$. The greatest fixed point of b is the union of all fixed points. We then have that:

Theorem 2.

1. *$\mathscr{R}$ is a bisimulation relation iff $\mathscr{R} \subseteq b(\mathscr{R})$.*
2. $\mathbf{gfp}(b) = \dot{\sim}$

Proof. This theorem will be proved in Section 2.5.1. □

In this thesis, we will use bisimulation for three main purposes.

First, to justify algebraic laws about processes, such as the associativity and commutativity of parallel composition in CCS. These can be regarded as fundamental, natural and obvious properties of parallelism. We might postulate them rather than deriving them, as is often done by building them into the semantics [BB92, Mil92]. A philosophical justification for not doing so is found in Bertrand Russell's view on postulates: that the advantages of postulating "are the same as the advantages of theft over honest toil" [Rus19, p. 72]. A more practical consideration is that postulating algebraic laws can complicate proofs by induction over the derivation of transitions: an inner induction over the definition of the postulated process equivalence is often needed.

Second, to justify compositional reasoning. It turns out that bisimilarity is a congruence wrt. all the operators of CCS, which means that we may swap bisimilar processes for bisimilar processes in any context while preserving bisimilarity. Hence CCS is compositional not only on the level of process descriptions, but on the level of proofs about process behaviour. Like many

process calculists we will take great pains to define languages where bisimilarity satisfies reasonable congruence properties.

Third, to reason about correctness of encodings. If we claim that a process P of language $\mathscr{L}$ can be encoded as a process P' of language $\mathscr{L}'$, it is reasonable to expect some notion of behavioural equivalence to hold between P and P'. Finer equivalences correspond to more behaviour being preserved by the encoding. A more thorough discussion of the correctness of encodings can be found in Section 2.3.

There exists a plethora of variants of bisimulation in the literature, accommodating different formalisms and different notions of which behaviour should be distinguished. To disambiguate it from the others, the variant presented above is called *strong labelled bisimulation*. Of the many existing variants we mention only *weak labelled bisimulation* [Mil89], which differs from the strong version in that silent actions (denoted τ) are considered internal and unobservable. Weak labelled bisimulation then distinguishes processes by externally observable behaviour only: instead of matching the transitions exactly, action for action, τ actions can be imitated by zero or more τ actions, and observable actions can be imitated by the same observable action plus any number of τ actions. For further reading on the topic of bisimulation, Sangiorgi's book [San12] is a good starting point. For other behavioural equivalences we refer to the excellent surveys by van Glabbeek [vG90, vG93].

2.3 Expressiveness

Given the massive proliferation of models of concurrency in general, and of process calculi in particular, it is natural to wonder how they are related. More precisely, are there system behaviours that can be expressed in one model but not another? If we find such behaviour, we say that it *separates* the models. To give the question a precise answer, we must first define what exactly we mean by "express". Unsurprisingly, concurrency theorists do not agree on a single definition. Measures of expressiveness are almost as proliferated as models of concurrency are. Hence, the original question of how two models relate can be given many different answers depending on which measures we apply.

This rather confusing state of the art has some unfortunate consequences; for an example, it is difficult to compare expressiveness results from the literature if they are obtained under different measures. Efforts to ameliorate the situation include surveys [Par08], standard criteria [Gor10b] and methods for comparing criteria [PvG15]. While such efforts are valuable, the current proliferation is not necessarily a cause for regret.

Milan Kundera [Kun80] famously wrote about the Czech word *litost* that

> as for the meaning of this word, I have looked in vain in other languages for an equivalent, though I find it difficult to imagine how anyone can understand the

> human soul without it. [...] Litost is a state of torment created by the sudden sight of one's own misery.

Assume for the sake of argument that there is indeed no equivalent word in other languages. Is *litost*, then, not expressible in English? The answer will depend on what we want to achieve. If we wish to translate a poem that uses the word, exactly preserving both meaning and meter, we will fall short. If we wish to write a dictionary entry on it, the number of syllables used by the translation becomes unimportant, and Kundera's definition above will do the job.

We should feel no *litost* over the proliferation of expressiveness measures, for it is with models of concurrency as with natural languages: different measures are appropriate for different purposes, and the purposes of concurrent systems are manifold. Hence, this section endeavours to offer a brief overview of how concurrency theorists measure expressiveness and why, without championing any one measure as superior to the others.

We distinguish between *absolute* and *relative* expressiveness. In absolute expressiveness, we pose a problem and ask whether a given language can solve it or not. In relative expressiveness, we ask whether one language can be translated into another. Since the latter area is where this thesis' contributions to expressiveness are, our survey follows suit by focusing exclusively on relative expressiveness.

2.3.1 Relative expressiveness

In *relative expressiveness* we ask whether, given two modelling languages $\mathscr{L}_1$ and $\mathscr{L}_2$, there exists an *encoding function* that translates processes of $\mathscr{L}_1$ into processes of $\mathscr{L}_2$. We will use P, S to range over processes of $\mathscr{L}_1$, and Q, R to range over processes of $\mathscr{L}_2$. Encoding functions will be written with semantic brackets, so $[\![P]\!]$ denotes the encoding of the process P. The main source of proliferation here is the many different criteria imposed on $[\![\cdot]\!]$ found in the literature. The rest of this section is a catalogue of such criteria, each measuring different notions of expressiveness in different ways.

Full abstraction

An encoding is *fully abstract* [Plo77] if it translates equivalent processes into equivalent processes, i.e. if for all P, S it holds that

$$P \asymp_1 S \quad \Longleftrightarrow \quad [\![P]\!] \asymp_2 [\![S]\!]$$

where $\asymp_1$ and $\asymp_2$ are some appropriate behavioural equivalences of $\mathscr{L}_1$ and $\mathscr{L}_2$. Fully abstract encodings are of practical interest in many settings. If $\asymp_2$ is endowed with a powerful proof technique, full abstraction lets us use the same technique for proofs about $\asymp_1$; this approach is taken by Madiot et

al. [MPS14]. If $\asymp_1$ and $\asymp_2$ respect some security properties of interest, fully abstract encodings can be used to achieve *secure compilation* [Aba99], where security features of the high-level language cannot be bypassed in the low-level language.

The use of full abstraction to measure of expressiveness, though widespread, is controversial [GN14, Par16]. The issue is that the preservation of equivalence does not imply the preservation of meaning. For an example, consider an encoding from the set $\{\textbf{apples}, \textbf{oranges}\}$ to itself that maps **apples** to **oranges** and vice versa. This encoding is fully abstract wrt. any reasonable equivalence on fruit, but does not preserve meaning. We conclude that full abstraction must be combined with other criteria to be a useful measure.

Operational correspondence

An encoding satisfies *operational correspondence* if it preserves and reflects transition behaviour. This can be formulated in different ways depending on how tight we want the correspondence to be. The tightest correspondence one could reasonably hope for is *strong* operational correspondence, when there is a one-to-one correspondence between transitions in the source and target language.

Definition 9 (Strong operational correspondence)**.** *An encoding* $[\![\cdot]\!]$ *satisfies* strong operational correspondence wrt. $\asymp_2$ *if*

1. *For all* P, P'*, if* $P \longrightarrow P'$ *then* $\exists Q'.\ [\![P]\!] \longrightarrow Q' \wedge Q' \asymp_2 [\![P']\!]$*; and*
2. *for all* P, Q'*, if* $[\![P]\!] \longrightarrow Q'$ *then* $\exists P'.\ P \longrightarrow P' \wedge [\![P']\!] \asymp_2 Q'$*.*

Intuitively, an encoding satisfies strong operational correspondence if for every reduction step from a source language process there is a matching reduction step from its encoding, and vice versa.

The purpose of the relation $\asymp_2$ here is to allow for garbage collection. In practice most encodings tend to leave behind junk: residuals of auxiliary mechanisms that facilitate the imitation of a transition, but are afterwards just inert clutter in the process description. It also moves the emphasis from syntax to semantics: we ask not whether the encoding can reach exactly $[\![P']\!]$, but whether it can reach some semantically equivalent process.

A practical benefit of strong operational correspondence is that it preserves effectiveness. For an example, suppose $\mathscr{L}_2$ is endowed with a powerful tool for analysing processes by following transitions. An encoding from $\mathscr{L}_1$ to $\mathscr{L}_2$ that satisfies operational correspondence allows us to reuse this tool for $\mathscr{L}_1$. If the operational correspondence is strong, we can be sure that the encoding itself will not introduce a state-space explosion that renders the analysis impractical.

If we ask whether we may imitate the transition behaviour at all, and not whether we may do so effectively, strong operational correspondence is none-

theless rather draconian. A more commonly used variation is *weak* operational correspondence, whose earliest use that I am aware of is by Walker [Wal95].

Definition 10 (Weak operational correspondence)**.** *An encoding* $[\![\cdot]\!]$ *satisfies* weak operational correspondence wrt. $\asymp_2$ *if*

1. *For all* P, P'*, if* $P \Longrightarrow P'$ *then* $\exists Q'.\ [\![P]\!] \Longrightarrow Q' \wedge Q' \asymp_2 [\![P']\!]$*; and*
2. *for all* P, Q'*, if* $[\![P]\!] \Longrightarrow Q'$ *then*

$$\exists Q''\, P'.\ Q' \Longrightarrow Q'' \wedge P \Longrightarrow P' \wedge [\![P']\!] \asymp_2 Q''$$

In weak operational correspondence, a reduction sequence and its imitation need not use the same number of reduction steps. This allows the encoding to use protocols whereby several transitions are used to imitate a single source language transition. Such protocols may result in an encoding having intermediate states that do not directly correspond to any source language states. Hence, we cannot expect that every target language transition sequence leads to a state that has a corresponding source language state. Whereas Clause 9.2 requires that Q' itself corresponds to a reachable source language state, Clause 10.2 requires only that Q' may eventually reach such a state.

Observable behaviour

The astute reader will have noticed that the discussion of operational correspondence above only considers the internal, unobservable behaviour of a process. It is reasonable to expect that encodings also respect observable behaviour to some degree. Without committing to any particular notion of observables, we can surely agree that observables are a property of processes. Hence we let an *observable*, ranged over by O, be a processes predicate $\mathscr{L}_1 \uplus \mathscr{L}_2 \to \mathbb{B}$. Let $\mathscr{O}$ range over sets of observables.

Definition 11 (Preserving and reflecting observations)**.** *An encoding* $[\![\cdot]\!]$ *is*

1. $\mathscr{O}$-preserving *if for all* $P \in \mathscr{L}_1$ *and all* $O \in \mathscr{O}$*,* $O(P)$ *implies* $O([\![P]\!])$*.*
2. $\mathscr{O}$-reflecting *if for all* $P \in \mathscr{L}_1$ *and all* $O \in \mathscr{O}$*,* $O([\![P]\!])$ *implies* $O(P)$*.*

An encoding is $\mathscr{O}$-sensitive *if it is* $\mathscr{O}$*-preserving and* $\mathscr{O}$*-reflecting.*

By instantiating $\mathscr{O}$ in various ways we obtain criteria for different kinds of observables. Below we briefly discuss two popular ways of instantiating $\mathscr{O}$.

Gorla [Gor10b] proposes analysing encodings in a setting where only two observables are considered: whether a process can *succeed*, and whether it is *divergent*. In order to succinctly define success, assume that both $\mathscr{L}_1$ and $\mathscr{L}_2$ have a parallel operator $|$ and a special *successful process* $\checkmark$. We may then define a predicate

$$O_{\checkmark}(P) \quad \triangleq \quad \exists P'.P \Longrightarrow \checkmark | P'$$

which intuitively means that P may reach a successful state, and require encodings to be $O_{\checkmark}$-sensitive.

A *divergent process* is one that has an infinite sequence of outgoing transitions

$$P \longrightarrow P' \longrightarrow P'' \longrightarrow \ldots$$

We write $O_\infty(P)$ to mean that P is divergent. Gorla proposes that encodings should be $O_\infty(P)$-reflecting, meaning that an encoding may not introduce infinite behaviour where none was present originally. Note that the ability to observe divergence implies stronger discriminatory power than what is granted by weak bisimulation: an inert process is weakly bisimilar to one that can do infinitely many internal actions. Sometimes, encodings are also required to be $O_\infty(P)$-preserving [Gor10a], meaning that if a process can diverge then so can its encoding.

The main advantage of Gorla's approach to observables is economy: since we make rather few assumptions about what must be present in the languages under consideration, it is applicable to languages whose transition labels are of very different nature. If on the other hand we are comparing two languages whose transition labels are similar or even identical, an approach is to compare those directly. This leads to the notion of *barbs* [MS92], which are abstract descriptions of the interface that a process exposes to its environment.

Definition 12 (Barbs)**.** *A process P* exposes the barb α*, denoted* $P \downarrow_\alpha$*, if* $\alpha \neq \tau$ *and there exists* P' *such that* $P \xrightarrow{\alpha} P'$.

Thus, a more fine-grained approach to observable behaviour than Gorla's is to require that the encoding is sensitive to barbs. In practice, this is usually weakened in several ways. First, we may restrict attention to only a subset of the barbs, in order to allow some labels to play a special role in the encoding. Second, we may restrict attention to some component of the labels, rather than whole labels. Third, we may require only that the label in question is exhibited eventually, as opposed to immediately; this is achieved by replacing $\longrightarrow$ with $\Longrightarrow$ in Definition 12.

Computational correspondence

If we require both operational correspondence and sensitivity to some observables, the result is a set of criteria that is sensitive to the causal dependency between internal and observable behaviour. *Computational correspondence* is an alternative that is less sensitive to the order in which observable events happen. Rather than compare observables pointwise at every process pair $(P, [\![P]\!])$, we take a global view and compare the set of observables seen through an entire run of the process. Intuitively, this can be thought of as allowing the encoding to introduce a greater degree of asynchrony. The earliest use of computational correspondence as a criterion is by Palamidessi [Pal97a] (where such

encodings are called "reasonable"); the definitions below follow Phillips and Vigliotti [PV04] (where such encodings are called "observation-respecting").

A *computation from P* is a finite or infinite sequence of processes

$$P_0, P_1, P_2, \ldots$$

such that $P_0 = P$ and $P_i \longrightarrow P_{i+1}$ for all i. Let $\mathscr{C}$ range over computations. A computation is *maximal* if it is infinite, or if the last element of the sequence cannot be further reduced. We extend observables to computations: $O(\mathscr{C})$ holds iff $O(P_i)$ holds for some $P_i \in \mathscr{C}$.

Definition 13 (Computational correspondence)**.** *An encoding* $[\![\cdot]\!]$ *satisfies* computational correspondence wrt. $\mathscr{O}$ *if for all* $P \in \mathscr{L}_1$ *it holds that*

1. *for all maximal computations* $\mathscr{C}$ *from P, there exists a maximal computation* $\mathscr{C}'$ *from* $[\![P]\!]$ *such that for all* $O \in \mathscr{O}$, $O(\mathscr{C}) = O(\mathscr{C}')$*; and*
2. *for all maximal computations* $\mathscr{C}'$ *from* $[\![P]\!]$*, there exists a maximal computation* $\mathscr{C}$ *from P such that for all* $O \in \mathscr{O}$, $O(\mathscr{C}) = O(\mathscr{C}')$.

Sometimes, Clause 13.1 is weakened by replacing $O(\mathscr{C}) = O(\mathscr{C}')$ with $O(\mathscr{C}) \supseteq O(\mathscr{C}')$ [VBG07]. We may think of this as allowing encodings that may deadlock or livelock before managing to mimic all the source language behaviour. Since this formulation does not guarantee that the encoding preserves any useful behaviour at all, it is mainly used for separation results.

Compositionality

So far we have only considered *semantic* criteria on encodings, that measure whether the *processes* of $\mathscr{L}_1$ are expressible in $\mathscr{L}_2$. In this section, we instead focus on *syntactic* criteria that measure whether the *operators* of $\mathscr{L}_1$ are expressible in $\mathscr{L}_2$. Broadly speaking, we say that an encoding of an operator is *compositional* if it is structurally defined in terms of the encodings of the operands. The first to consider the expressibility of operators in such terms was probably de Simone [dS85]; the definitions below follow Parrow's survey [Par08].

To formalise this idea, we need the notion of *process context*. A *context*, ranged over by C, is a process with a hole. $C[P]$ means the result of filling the hole in C with P. For an example, if $C = P \mid (Q + \cdot)$ (where $\cdot$ denotes a hole) then $C[R] = P \mid (Q + R)$. This definition can be generalised to contexts with arbitrarily many holes, where $C[\widetilde{P}]$ denotes the result of filling the i:th hole of C with P_i for all i.

Definition 14 (Compositionality)**.** *Let* **op** *be an n-ary operator of* $\mathscr{L}_1$*. An encoding* $[\![\cdot]\!]$ translates **op** compositionally *if there is a context C such that for all* $P_0, \ldots, P_n$

$$[\![\mathbf{op}(P_0, \ldots, P_n)]\!] = C[[\![P_0]\!], \ldots, [\![P_n]\!]]$$

We say that $[\![\cdot]\!]$ is compositional if it translates all operators of $\mathscr{L}_1$ compositionally.

As the reader has no doubt guessed, many variants of this definition are in use. For an example, the context C may be allowed to depend on some properties of the operands, and not just the operator. The most common approach is to let the context depend on the free names of the operands [Gor10b].

Compositionality allows us to introduce *local* coordinators at every operator for coordinating the operands, but it does not allow the introduction of a *global* coordinator for all processes. For an example, suppose we want to encode a conversation between n people sitting in a room as a Facebook conversation. The acts of speaking and listening can be encoded as sending and receiving messages to and from the Facebook server. A room $P_0 \mid \ldots \mid P_n$, where each P_i is a person, can then be encoded as $F \mid [\![P_0]\!] \mid \ldots \mid [\![P_n]\!]$, where F denotes the Facebook server. This encoding seems sensible and of clear practical value, but it is not compositional since F does not arise from the encoding of any one operator. To address this we may use *weakly compositional* encodings [Par08], that admit an otherwise compositional encoding to be wrapped in an outermost context.

Definition 15 (Weak compositionality)**.** *An encoding $[\![\cdot]\!]$ is* weakly compositional *if there is a context C and a compositional encoding $[\![\cdot]\!]_c$ such that for all P it holds that $[\![P]\!] = C[[\![P]\!]_c]$.*

Another common concern is whether encodings preserve the degree of distribution of the source language, i.e. the degree to which components of a system may be run at different locations. The most common criterion used for this purpose is to insist on homomorphic translation of the parallel operator [Pal97a]:

$$[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!]$$

2.3.2 Summary

We have reviewed a selection of correctness criteria on encodings that occur in the literature. The criteria can be broadly categorised as either *semantic*, i.e. concerned with the preservation of behaviour; or *syntactic*, i.e. concerned with the preservation of structure. So which criteria should we impose on our encodings?

The (perhaps disappointing) answer is that there are no hard and fast rules; it is a matter of taste, and more importantly of what properties are important for the intended application. However, there are some rules of thumb:

- Any one criterion taken on its own will typically fail to rule out useless encodings. For example, an encoding into CCS that maps every process to 0 will satisfy weak operational correspondence. Hence, a mix of several criteria should be considered.
- An encoding is more convincing if it satisfies stronger criteria. A separation result is more convincing if it is derived under weak criteria.

2.4 Nominal sets

Most process calculi have scoping mechanisms that allow processes to hide information from their environment. In Section 2.1 we saw the (νx) construct of CCS, used to restrict the scope of ports. More advanced process calculi may allow e.g. cryptographic keys to be restricted in the same way. If processes can receive input, this calls for locally scoped placeholders, analogous to function parameters in programming languages.

When we reason formally about syntax with scoping mechanisms, such as within a theorem prover, we must consider the problem of *alpha-equivalence*. For an illustrating example, consider the following elementary facts about the natural numbers:

$$\forall x \in \mathbb{N} : x \geq 0 \qquad\qquad \forall y \in \mathbb{N} : y \geq 0$$

Clearly the two expressions above state the same logical fact: that every natural number is greater than or equal to zero. But if we view them syntactically, they differ slightly. One uses x to quantify over $\mathbb{N}$ while the other uses y. Intuitively, the names x and y are local within the scope of their respective $\forall$ quantifiers; we say that they are *bound* by the quantifiers. Names that are not bound will be called *free*. Intuitively we view the expressions above as one and the same, and we can express this intuition by saying that they are *alpha-equivalent*. Somewhat informally, we say that two expressions are alpha-equivalent if they differ only in the choice of bound names. An alpha-conversion is the act of substituting one alpha-equivalent expression for another.

In informal proofs that reason about bound variables, a common practice is to use Barendregt's variable convention [Bar81], stating that "all bound variables are chosen to be different from the free variables". While useful since it allows informal proofs to avoid getting bogged down in tedious alpha-conversion arguments, Urban et al. demonstrate that it can result in faulty proofs when used carelessly in inductive arguments [UBN07]. Hence it will not do for our purposes: we will rely heavily on inductive proofs over syntax with bound variables, and need a formal treatment of alpha-conversion ironclad enough for a theorem prover formalisation. We turn instead to the *nominal sets* of Gabbay and Pitts [GP01, Pit03], an approach where a formal

treatment of bound names is built from a notion of what it means to exchange names.

We assume a countably infinite set of *names*[1] $\mathcal{N}$, ranged over by a, b, c, x, y, z. A *permutation*, ranged over by p, is a sequence of name pairs. A nominal set is a set equipped with a *permutation action* $\cdot$ which applies name permutations to elements of the set. Intuitively, applying a permutation $(a\,b)\cdot X$ exchanges all occurrences of a for b in X, and vice versa. Formally, a permutation action is any function $\cdot$ satisfying

$$p\cdot(p'\cdot X) = (p\circ p')\cdot X \qquad\qquad i\cdot X = X$$

where p, p' range over permutations of $\mathcal{N}$, and i is the identity permutation. This machinery allows us to define what it means for a name to occur in an element of a nominal set independently of the concrete structure of the element: a name occurs in an element if it can be affected by permutations. Formally, we say that a set S of names *supports* X iff

$$\forall a, b \notin S : (a\,b)\cdot X = X$$

Intuitively, S supports X if all names occurring in X are in S. If a finite set of names supports X, there exists a unique least S such that S supports X, which we call the *support* of X and denote $\mathsf{n}(X)$. Henceforth, we will only be concerned with nominal sets such that all their elements have finite support, unless otherwise noted. We also introduce the notation $a\#X$, pronounced "a is *fresh* in X", as shorthand for $a \notin \mathsf{n}(X)$.

Using these tools, we can construct a formal justification for alpha-equivalence. If X belongs to a nominal set, we can define a new nominal set $[a]X$ by taking (a, X) quotiented by alpha-conversions of a:

$$[a]X \triangleq \{(b, (b\,a)\cdot X) : b = a \vee b\#X\}$$

This construct corresponds to X with a bound. When we define syntax with binding constructs, such as in the example with the $\forall$ quantifier at the beginning of this section, we will implicitly be using this quotient construct unless otherwise noted. This technique allows a restricted form of Barendregt's variable convention to be recovered in inductive arguments, leading to formal proofs where many explicit alpha-conversions can be avoided [UBN07].

Another useful concept is that of *equivariance*. A function symbol f is equivariant if for all p it holds that $p\cdot f(X) = f(p\cdot X)$. A constant symbol X is equivariant if for all p it holds that $p\cdot X = X$. Intuitively, something is equivariant if it treats all names equally.

This somewhat terse treatment of nominal techniques only covers the small part thereof that we apply in this thesis. A good starting point for a reader

[1] Names are sometimes called *atoms* in the literature.

interested in more details is a survey by Gabbay [Gab11], which treats the more foundational work underlying the application to syntax with binders.

For the purposes of formal treatment of syntax with binders, alternatives to nominal sets include *de Bruijn indexes* [dB72], *higher-order abstract syntax* [PE88] and *locally nameless representation* [Cha12]. With de Bruijn indexes, names are represented by natural numbers indicating the distance between a bound name and its binder in the syntax tree, so e.g. $\forall x.\exists y.x > y$ becomes $\forall\exists 2 > 1$. The main advantage of this representation is that alpha-equivalence and syntactic equality coincides. Disadvantages include low readability, and that explicit arithmetic on indexes tends to creep into the statement of definitions and theorems [Hir97, Bri08]. The locally nameless approach uses de Bruijn indexes, but only for bound names: free names represent themselves. This change largely removes the need for arithmetic on indexes, at the cost of making it difficult to formulate statements where the same name occurs both free and bound, such as inductive definitions. In the higher-order abstract syntax approach, binders are represented by functions in a metalanguage. This yields alpha-conversion and substitution essentially for free, but does not admit explicit manipulation of bound names.

2.5 Theorem proving in Isabelle

Interactive theorem provers are computer programs that can formally prove mathematical theorems. They can check that proofs written by a user really are correct. Most provers also have proof procedures that can automate the more tedious parts of proofs.

Isabelle [NPW02] is an interactive theorem prover that was originally developed by Lawrence Paulson in the mid-80s [Pau86], and remains under active development and use to this day. Most of the main results presented in Papers I-VI have been formally proven using Isabelle.

Proofs in Isabelle are highly trustworthy, since Isabelle uses the approach introduced by Milner in the LCF theorem prover [Mil79], where theorems are represented by an abstract datatype `thm`. The constructors of the `thm` datatype are the inference rules of the logic. In a strongly typed programming language[2], this method enforces the correctness of all theorems proved in the system (up to correctness of the implementation of the interface for `thm`), since the only way to construct a theorem's representation as `thm` is to perform its proof. This approach also entails that Isabelle can be extended with new proof procedures, definitions, user interfaces etc. on top of this core without endangering soundness, even in the presence of programming errors in the extensions.

There are many other theorem provers besides Isabelle, such as Coq [BC04] and HOL [GM93]. The choice of Isabelle for this work is motivated primarily

[2]LCF and Isabelle are implemented in Standard ML [MTH90].

```
locale LTS =
  fixes trans :: "'state ⇒ 'label ⇒ 'state ⇒ bool"
begin
```

Figure 2.2. Locale preamble.

by the existence of the HOL/Nominal package (sometimes called Nominal Isabelle) by Urban et al. [Urb08, UB06, UT05, HU10], which implements techniques for reasoning about the nominal sets introduced in Section 2.4. Features include support for inductive datatype definitions modulo alpha-equivalence, and primitive recursive definitions over such datatypes. We also benefit from automated proof procedures for reasoning about freshness and equivariance.

Many other tools available in Isabelle are of great value to our work. We write our proofs in Wenzel's Isar language [Wen99] for human-readable proof scripts, in which proofs are structured with a syntax that mimics the way proofs are written in natural language, so as to be readable by someone who is not an Isabelle expert. Ballarin's *locale* construct [Bal03, Bal14] is a sectioning concept allowing theories to be parametrised on arbitrary but fixed types, terms and functions that satisfy certain assumptions. Concrete instantiations of a locale can be obtained by simply proving that the concrete types, terms and functions satisfy the assumptions.

2.5.1 Extended example

In this section we aim to give the uninitiated reader a front seat view of what an Isabelle development might look like. For our development we will revisit the discussion of bisimulation from Section 2.2. Building on top of the extensive library support for standard mathematical notions such as sets and lattices, we begin by defining labelled transition systems, bisimulation, bisimilarity and the bisimilarity functional. The development then culminates in a proof of Theorem 2, where we claimed a correspondence between the standard definition of bisimilarity and its lattice-theoretic definition via the bisimilarity functional. Figures 2.2–2.7 show the full Isabelle code corresponding to these definitions and theorems. At 58 lines of code, it is quite close to the exposition from Section 2.2 in size; to a reader who is comfortable with functional programming notation as well ordinary mathematical notation, it is quite close in readability also. The LaTeX code for typesetting the figures has been automatically generated from the Isabelle sources, using the built-in document preparation system. This safeguards against gaps between what has been proven and what is presented.

Figure 2.2 defines the locale `LTS` that we will be working in. We assume an arbitrary but fixed constant `trans` satisfying the given type signature, that models our transition relation. The idea is that the proposition `trans P α P'`

```
definition bisimulation :: "('state × 'state) set ⇒
bool"
where "bisimulation R ≡
  (∀ P Q α P'. ((P,Q) ∈ R ∧ trans P α P') ⟶ (∃ Q'.
trans Q α Q' ∧ (P',Q') ∈ R))
∧ (∀ P Q α Q'. ((P,Q) ∈ R ∧ trans Q α Q') ⟶ (∃ P'.
trans P α P' ∧ (P',Q') ∈ R))"

definition bisimilarity :: "('state × 'state) set"
where "bisimilarity ≡ {(P,Q) | P Q. ∃ R. (P,Q) ∈ R ∧
bisimulation R}"
```

Figure 2.3. Definitions of bisimulation and bisimilarity

```
lemma bisim_is_bisim:
 shows "bisimulation bisimilarity"
unfolding bisimilarity_def bisimulation_def
by blast
```

Figure 2.4. Proof that bisimilarity is a bisimulation.

corresponds to $P \xrightarrow{\alpha} P'$. Note that `'state` and `'label` are type variables, that can be instantiated to be any Isabelle type in a concrete setting.

In Figure 2.3 we define the predicate `bisimulation`, which returns `True` iff its argument is a bisimulation relation; and `bisimilarity`, the set of all process pairs that are related by some bisimulation. These correspond exactly to Definitions 6 and 7.

We prove our first lemma in Figure 2.4. The first line declares the name of our lemma to be `bisim_is_bisim`. The second line states the proposition we are proving, namely that bisimilarity is a bisimulation relation. The third and fourth lines constitute the proof, which in this case instructs Isabelle to unfold the definitions of bisimulation and bisimilarity, and then invoke the built-in automatic proof procedure `blast`. Isabelle comes with many such proof procedures — in this section we use `blast`, `auto` and `metis` — each with its own strengths and weaknesses. `blast` [Pau99] perform *classical reasoning*, and is geared towards proofs that mainly involve introduction and elimination of e.g. logical connectives, set constructors and quantifiers. `auto` relies on term rewriting interleaved with proof search; the Isabelle developers find that "it is impossible to describe succinctly what `auto` does due to its heuristic, ad hoc nature" [BBN11, p. 14]. It is suitable for proofs involving a mix of classical reasoning and equational reasoning. `metis`[3] [Hur03] is similar to `blast`; it is often faster and more reliable when the number of inference rules used in the proof is small. Knowing which to apply when, and with what parameters, is a matter of experience and guesswork.

[3] In ancient Greek religion, Metis is the mother of wisdom.

```
definition b :: "('state × 'state) set ⇒ ('state ×
'state) set"
where
"b R ≡
 {(P,Q) | P Q.
   (∀ α P'. trans P α P' ⟶ (∃ Q'. trans Q α Q' ∧
(P',Q') ∈ R))
∧ (∀ α Q'. trans Q α Q' ⟶ (∃ P'. trans P α P' ∧
(P',Q') ∈ R))}"
```

Figure 2.5. The bisimilarity functional.

```
lemma bisimulation_iff_in_fun:
 shows "bisimulation R ⟷ (R ⊆ b R)"
by(auto simp add: bisimulation_def b_def)

lemma mono_b:
  shows "mono b"
by(rule monoI) (force simp add: b_def)
```

Figure 2.6. Proofs of Theorem 1.1 and that `b` is monotonic.

The definition of the bisimilarity functional `b` in Figure 2.5 corresponds exactly to Definition 8. In Figure 2.6 we prove the first half of Theorem 1, and also that `b` is monotonic. Monotonicity is important here since it implies that `b` has a greatest fixed point; this detail was glossed over in Section 2.2. Note that since we have not defined a constant named `R`, the use of `R` in the statement of the first lemma is implicitly all-quantified. We may elide its type since Isabelle infers the types of variables automatically. The proofs are again by invoking automatic proof procedures, though with one explicit use of the introduction rule for monotonicity to begin the latter proof.

A proof with more interesting structure is found in Figure 2.7, where we show that bisimilarity is the greatest fixed point of `b`. The keyword **proof** signals the beginning of a structured proof. Since we are proving an equality between sets, we may use the strategy of showing that membership in one implies membership in the other, and vice versa. The arguments supplied to **proof** constitute a formal justification for this proof structure, by explicitly citing the introduction rules for set and predicate equality. In the left-to-right direction, we fix two states `P` and `Q`, and assume they are related by `gfp b`. By an elementary fact about fixed points we know that `gfp b ⊆ b(gfp b)`. By Theorem 2.1 this is equivalent to `gfp b` being a bisimulation relation. Hence `P` and `Q` are related by a bisimulation relation, which suffices for them to be bisimilar. By the keyword **next** we signal to Isabelle that we move on to proving the next goal, namely the right-to-left direction. By the principle of coinduction, we may prove `(P,Q) ∈ gfp b` by finding a relation `R` that includes `(P,Q)` and is

```
theorem "gfp b = bisimilarity"
proof(rule set_eqI, rule prod_cases[OF iffI])
  fix P Q
  assume "(P,Q) ∈ gfp b"
  have "gfp b ⊆ b(gfp b)"
    by(rule gfp_lemma2[OF mono_b])
  hence "bisimulation(gfp b)"
    by(subst bisimulation_iff_in_fun)
  thus "(P,Q) ∈ bisimilarity"
    using `(P,Q) ∈ gfp b`
    by(auto simp add: bisimilarity_def)
next
  fix P Q
  assume "(P,Q) ∈ bisimilarity"
  moreover have "bisimilarity ⊆ b bisimilarity"
    by(metis bisimulation_iff_in_fun bisim_is_bisim)
  ultimately show "(P,Q) ∈ gfp b"
    by(coinduct rule: weak_coinduct)
qed
```

Figure 2.7. Proof of Theorem 2.2.

a post-fixed point of `b` (meaning that `b R` can be no smaller than `R`). Bisimilarity is such a relation. **qed** signals the end of the proof.

Comparing the prose of the previous paragraph with the proof script of Figure 2.7 reveals that the structure of the formal proof closely mirrors the informal proof idea. The level of abstraction is also similar. The main difference is that in the Isabelle code, every intermediate step is justified.

2.6 Psi-calculi

Psi-calculi is a family of process calculi, in the tradition of CCS and the pi-calculus [MPW92]. It provides as an easy way to obtain a custom modelling language for concurrent systems, with precisely the features useful for the application in question.

We may also think of psi-calculi as a powerful proof technique for standard meta-theoretical results, along the following lines. Suppose that a calculus $\mathscr{P}$ is a psi-calculus. Then it follows that standard algebraic and congruence properties of strong and weak bisimulation hold in $\mathscr{P}$ [BJPV09, JBPV10], by proofs that are machine-checked in Nominal Isabelle. In this view, the relation between psi-calculi and a psi-calculus is roughly analogous to the relation between group theory and arithmetic on the natural numbers. For this reason, psi-calculi represents a major contribution to the endeavour of culling concurrency theory.

A psi-calculus is created by instantiating parameters corresponding to what notions of data terms and logical dependencies are needed. The parameters are taken as nominal sets, which lets us reason formally about what it means to bind into them even though we assume nothing about their concrete structure.

In this section, we will recapitulate definitions and results pertaining to psi-calculi that are relevant to the scope of this thesis. Since psi-calculi underpins all but one of the papers comprising the thesis, we will go into considerably more detail here than in previous sections. We also endeavour to give some intuition, though we recommend Johansson's PhD thesis [Joh10] for a treatment of psi-calculi with more motivation and examples. Other developments of and for psi-calculi that we do not use in this thesis include a symbolic semantics [JVP10], algorithms for computing strong and weak bisimulations [JVP12], a workbench implemented in PolyML based on them [Gut11, BGRV13], several type systems [Hüt11, Hüt13], and representations of event structures [NPH14] and actor networks [Pri14].

2.6.1 Parameters and requisites

A psi-calculus is obtained by supplying a notion of *terms*, corresponding to both communication channels and the messages that can be sent on them, as well as a logic with *assertions* and *conditions*. Assertions can be thought of as facts about the environment in which a process executes, and conditions as logical statements that may be true or false in any given assertion environment.

The terms $\mathbf{T}$ ranged over by M, N, L, T, the assertions $\mathbf{A}$ ranged over by Ψ, and the conditions $\mathbf{C}$ ranged over by φ can be chosen to be any finitely supported nominal sets. The precise relationship between assertions and conditions is given by an *entailment relation* $\vdash \subseteq \mathbf{A} \times \mathbf{C}$. Among the conditions there can be *channel equivalence* clauses $M \leftrightarrows N$, meaning that the terms M and N represent the same communication channel.

For each of $\mathbf{T}, \mathbf{A}, \mathbf{C}$, a corresponding notion of *substitution* must also be supplied as a parameter. Intuitively, the substitution $X[\widetilde{x} := \widetilde{T}]$ simultaneously replaces all occurrences of $\widetilde{x}$ in X with the corresponding element of $\widetilde{T}$. In practice, substitution can be chosen to be any function that satisfies certain requisites (see Table 2.2). Intuitively, the requisites are that the substitution function must treat all names equally; that for purposes of alpha-conversion, the names $\widetilde{x}$ may be treated as if they bind into X in $X[\widetilde{x} := \widetilde{T}]$; and that when X is a term containing $\widetilde{x}$, the names of $\widetilde{T}$ must occur in the result of the substitution. Hence substitution may have behaviour that runs counter to the intuition of a simultaneous substitution — in this way, "substitution" is perhaps a misnomer. A substitution $[\widetilde{x} := \widetilde{T}]$ is *well-formed* if $|\widetilde{x}| = |\widetilde{T}|$ and the names $\widetilde{x}$ are pairwise distinct. Henceforth we will only consider well-formed substitutions unless otherwise noted. We will use σ to range over sequences of substitutions, and use $X\sigma$ to mean their successive application to X.

Parameters	Requisites
$\mathbf{T}$	All $M \in \mathbf{T}$ finitely supported.
$\mathbf{A}$	All $\Psi \in \mathbf{A}$ finitely supported.
$\mathbf{C}$	All $\Phi \in \mathbf{C}$ finitely supported.
$\mathbf{1} \in \mathbf{A}$	$\mathsf{n}(\mathbf{1}) = \emptyset$
$\vdash \subseteq \mathbf{A} \times \mathbf{C}$	$\vdash$ equivariant.
$\leftrightarrow : \mathbf{T} \times \mathbf{T} \Rightarrow \mathbf{C}$	$\leftrightarrow$ equivariant.
	$\Psi \vdash M \leftrightarrow N \;\Rightarrow\; \Psi \vdash N \leftrightarrow M$
	$\Psi \vdash M \leftrightarrow N \wedge \Psi \vdash N \leftrightarrow L$
	$\quad \Rightarrow \Psi \vdash M \leftrightarrow L$
$\otimes : \mathbf{A} \times \mathbf{A} \Rightarrow \mathbf{A}$	$\otimes$ equivariant.
	$\Psi \otimes \mathbf{1} \simeq \Psi$
	$\Psi \otimes \Psi' \simeq \Psi' \otimes \Psi$
	$(\Psi \otimes \Psi') \otimes \Psi'' \simeq \Psi \otimes (\Psi' \otimes \Psi'')$
	$\Psi \simeq \Psi' \;\Rightarrow\; \Psi \otimes \Psi'' \simeq \Psi' \otimes \Psi''$
$\_[\_ := \_] : \mathbf{X} \times \mathscr{N}^\star \times \mathbf{T}^\star \Rightarrow \mathbf{X}$	$\_[\_ := \_]$ equivariant
	$\widetilde{x}, \widetilde{y}$ distinct $\wedge\ \widetilde{y} \# \widetilde{x}, X$
	$\quad \Rightarrow X[\widetilde{x} := \widetilde{T}] =$
	$\quad\quad ((\widetilde{y}\,\widetilde{x}) \cdot X)[(\widetilde{y}\,\widetilde{x}) \cdot \widetilde{x} := \widetilde{T}]$
	$X \in \mathbf{T} \wedge \widetilde{x}$ distinct $\wedge\ \widetilde{x} \subseteq \mathsf{n}(X)$
	$\quad \Rightarrow \mathsf{n}(\widetilde{T}) \subseteq \mathsf{n}(X[\widetilde{x} := \widetilde{T}])$

Table 2.2. *Parameters and requisites.* $\mathbf{X}$ *ranges over* $\mathbf{T}, \mathbf{A}, \mathbf{C}$. *The relation* $\simeq$ *equates two assertions iff they entail the same conditions.*

A notion of what it means to compose two assertions, denoted by $\otimes$, is also a parameter. It can be chosen to be any equivariant function that forms an abelian monoid with respect to $\mathbf{1}$, which is called the *unit assertion* and is another parameter. $\otimes$ can be thought of as logical conjunction of assertions and $\mathbf{1}$ as the least informative assertion, though again it is possible to choose them in a way that runs counter to this intuition, e.g. by using a non-monotonic logic.

One more requisite needs to be mentioned: we require the channel equivalence relation $\leftrightarrow$ to be symmetric and transitive. Note however that reflexivity is not required. Symmetry and transitivity entails that if a term is channel equivalent to something, it is channel equivalent to itself. Omitting reflexivity allows there to be terms that are not usable as communication channels.

2.6.2 Syntax and semantics

We will explain the syntax and semantics of psi-calculi through a running example where $\mathbf{A} \triangleq \mathscr{P}_{\text{fin}}(\mathbf{C})$ and $\vdash \;\triangleq\; \ni$ (i.e. where an assertion is the finite set of conditions that are currently true), where $\leftrightarrow$ is syntactic equality on terms,

and where $\otimes \triangleq \cup$ and $\mathbf{1} \triangleq \emptyset$. Substitution is the standard capture-avoiding syntactic replacement.

From these parameters, we construct a psi-calculus using the structured operational semantics approach that we used for CCS in Section 2.1. We will introduce the syntax and semantics on an operator by operator basis; some of the operators will be familiar from CCS. The agents $\mathbf{P}$ of a psi-calculus are ranged over by P, Q, R. Actions are ranged over by α. Transitions are of kind $\Psi \rhd P \xrightarrow{\alpha} P'$, meaning that in the assertion environment Ψ, P can do α and evolve to P'.

Note that the semantic rules as presented here apply to all psi-calculi, and do not rely on any particulars of the running example.

Nil

The simplest process construct is *nil*, denoted $\mathbf{0}$, which is a process that does nothing and can be thought of as denoting termination. No semantic rules describe its behaviour, since it has none.

Output

The *output* prefix $\overline{M}N.P$ sends the message N on the channel M, and then proceeds as P. Its behaviour is defined by the single inference rule

$$\textsc{Out}\ \frac{\Psi \vdash M \leftrightarrows K}{\Psi \rhd \overline{M}N.P \xrightarrow{\overline{K}N} P}$$

where the action $\overline{K}N$ signals readiness to send a message. Note that the subject on the label need not be the same as in the prefix, but by the precondition they must be equivalent in the current environment. In our running example $\leftrightarrows$ is syntactic equality and hence we will always have $M = K$ where the precondition holds by default.

Input

The *input* prefix $\underline{M}(\lambda\tilde{x})N.P$, where $\tilde{x}$ binds into N and P, receives a message on channel M that matches the pattern $(\lambda\tilde{x})N$. Pattern matching is defined in terms of substitution, as seen in the derivation rule

$$\textsc{In}\ \frac{\Psi \vdash M \leftrightarrows K}{\Psi \rhd \underline{M}(\lambda\tilde{x})N.P \xrightarrow{\underline{K}N[\tilde{x}:=\tilde{L}]} P[\tilde{x}:=\tilde{L}]}$$

where we require $\tilde{x} \subseteq \mathsf{n}(N)$ and for $\tilde{x}$ to be distinct.

For an example, suppose that among the terms there are pairs of terms, written (M, N). A process that receives a pair of names, and then sends the first along the second, can be written

$$\underline{M}(\lambda x,y)(x,y).\bar{x}y.\mathbf{0}$$

Since $(x,y)[x,y := 1,2] = (1,2)$, we can use the IN rule to derive the transition

$$\Psi \rhd \underline{M}(\lambda x,y)(x,y).\bar{x}y.\mathbf{0} \xrightarrow{\underline{M}(1,2)} \bar{1}2.\mathbf{0}$$

Moreover, note that this agent cannot receive a message that is not a tuple, since no substitution can produce such a message from the tuple pattern.

Case

Case statements denote condition-guarded, non-deterministic choice. For an example, the agent $\mathbf{case}\ \varphi : \overline{M}N.\mathbf{0} \,[]\, \varphi' : \underline{M}(\lambda\varepsilon)N.\mathbf{0}$ can send N on M in an environment where φ is true, can receive N on M in an environment where φ' is true, can do either of the above in an environment where both hold, and neither if neither hold. When either branch is chosen, the process commits to that branch and the option to execute the other branch is forfeit. The semantics is defined as follows:

$$\textsc{Case}\ \frac{\Psi \rhd P_i \xrightarrow{\alpha} P' \qquad \Psi \vdash \varphi_i}{\Psi \rhd \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \xrightarrow{\alpha} P'}$$

If the psi-calculus under consideration has a condition $\top$ that is true in every environment, we will use $P+Q$ as an abbreviation for $\mathbf{case}\ \top : P \,[]\, \top : Q$, allowing us to recover the familiar choice construct from CCS.

Restriction

In a *restriction* $(\nu a)P$, a binds into P. This means that the name a is local to P, and cannot be used outside the scope of the ν binder. An example application is cryptographic models, where bound names typically represent private keys or random nonces. The behaviour of restrictions is captured by two rules. The SCOPE rule intuitively states that scopes have no effect on actions that do not use the bound name:

$$\textsc{Scope}\ \frac{\Psi \rhd P \xrightarrow{\alpha} P'}{\Psi \rhd (\nu b)P \xrightarrow{\alpha} (\nu b)P'}\ b\#\alpha,\Psi$$

In particular, note that the freshness side conditions prevent bound names from being used outside their scope boundaries. Bound names may however be sent as part of messages across scope boundaries via the OPEN rule. This corresponds to sharing a secret, and causes the scope to be extended to cover also the receiver. This feature is traditionally called *scope extrusion* [MPW92].

$$\textsc{Open}\ \frac{\Psi \rhd P \xrightarrow{\overline{M}(\nu\widetilde{a})N} P'}{\Psi \rhd (\nu b)P \xrightarrow{\overline{M}(\nu\widetilde{a}\cup\{b\})N} P'}\ \begin{array}{l} b\#\widetilde{a},\Psi,M \\ b \in \mathsf{n}(N) \end{array}$$

The action $\overline{M}(\nu\widetilde{a})N$ is called a *bound output*, where the names $\widetilde{a}$ bind into both N and the residual process P'. We identify $\overline{M}(\nu\varepsilon)N$ and $\overline{M}N$. The expression $\tilde{a}\cup\{b\}$ means the sequence $\tilde{a}$ with b inserted anywhere.

Parallel composition and assertion

The *parallel composition* $P \mid Q$ is a process that executes both P and Q concurrently. Two processes composed in this manner can interact in two distinct ways. The first is by revealing assertions to each other, thus changing the current assertion environment at runtime. For this purpose there is a process construct $(\!|\Psi|\!)$ that asserts Ψ to its environment. This assertion may influence which channel equivalence clauses hold, and which branches of **case** statements may be executed. We achieve this by the parallel composition rule

$$\textsc{Par}\ \frac{\Psi_Q \otimes \Psi \rhd P \xrightarrow{\alpha} P'}{\Psi \rhd P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \mathrm{bn}(\alpha)\#Q$$

where Ψ_Q denotes the composition of the top-level assertions that Q asserts to its environment. For an example, if

$$Q = (\!|\Psi|\!) \mid (\!|\Psi'|\!) \mid \overline{M}N.(\!|\Psi''|\!)$$

then $\Psi_Q = \Psi \otimes \Psi'$. A symmetric version of the PAR rule is elided. $\mathrm{bn}(\alpha)$ denotes the binding names of α. As an example of the kind of interactions possible, consider the agents $P = \mathbf{case}\ \mathbf{flag} : \overline{M}N.\mathbf{0}$ and $Q = \overline{K}N.(\!|\{\mathbf{flag}\}|\!)$ and the execution of $P \mid Q$ in the environment $\emptyset$. The assertion $\{\mathbf{flag}\}$ in Q occurs under an input or output prefix; we call such assertions *guarded*. Only the unguarded assertions of Q are considered part of Ψ_Q, so $\Psi_Q = \emptyset$. Hence the transition from P is not enabled from the start, since **flag** is not set. However, we can derive the following:

$$\textsc{Par}\ \frac{\textsc{Out}\ \dfrac{}{\emptyset\cup\emptyset \rhd Q \xrightarrow{\overline{K}N} (\!|\{\mathbf{flag}\}|\!)}}{\emptyset \rhd P \mid Q \xrightarrow{\overline{K}N} P \mid (\!|\{\mathbf{flag}\}|\!)}$$

After the output on K, **flag** is no longer guarded and we can infer the transition from P by the following derivation:

$$\textsc{Par}\ \dfrac{\textsc{Case}\ \dfrac{\textsc{Out}\ \dfrac{}{\{\mathbf{flag}\}\cup\emptyset \rhd \overline{M}N.\mathbf{0} \xrightarrow{\overline{M}N} \mathbf{0}} \qquad \mathbf{flag} \in \{\mathbf{flag}\}\cup\emptyset}{\{\mathbf{flag}\}\cup\emptyset \rhd P \xrightarrow{\overline{M}N} \mathbf{0}}}{\emptyset \rhd P \mid (\!|\{\mathbf{flag}\}|\!) \xrightarrow{\overline{M}N} \mathbf{0} \mid (\!|\{\mathbf{flag}\}|\!)}$$

The interaction of restrictions and assertions in processes is captured formally by the notion of *frames*, ranged over by F, G. A frame $F = (\nu \widetilde{b}_F)\Psi_F$ is an assertion Ψ_F with a list of names $\widetilde{b}_F$ that bind into it. The frame of a process P, denoted $\mathscr{F}(P)$, is its unguarded binders and the capture-avoiding composition of all its unguarded assertions, or $\mathbf{1}$ if there are no unguarded assertions. For an example, $\mathscr{F}((\!|\Psi|\!) \mid (\nu a)(\!|\Psi'|\!)) = (\nu a)(\Psi \otimes \Psi')$ if $a \# \Psi$.

The parallel composition operator also enables processes to send messages to each other. As a simple example, suppose that names are terms, and consider the agent $P = \underline{a}(\lambda x).\overline{b}x.\mathbf{0}$ which receives a message on channel a and then passes it along on channel b, and $Q = \overline{a}y.\mathbf{0}$. Since P is ready to receive a message on a, and Q is ready to send one, they may communicate, resulting in a transition

$$\Psi \rhd P \mid Q \xrightarrow{\tau} \overline{b}y \mid \mathbf{0}$$

where τ is the internal action familiar from CCS, denoting that $P \mid Q$ engage in an internal communication which the environment cannot interact with. Note that the x in P has been instantiated to the y that was received from Q.

Formally, the semantic rule for communication is as follows:

$$\textsc{Com}\ \dfrac{\Psi_Q \otimes \Psi \rhd P \xrightarrow{\overline{M}(\nu \widetilde{a})N} P' \qquad \Psi_P \otimes \Psi \rhd Q \xrightarrow{\underline{K}N} Q' \qquad \Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K}{\Psi \rhd P \mid Q \xrightarrow{\tau} (\nu \widetilde{a})(P' \mid Q')}\ \widetilde{a} \# Q$$

A symmetric version is elided, and we assume that $\mathscr{F}(P) = (\nu \widetilde{b}_P)\Psi_P$ and $\mathscr{F}(Q) = (\nu \widetilde{b}_Q)\Psi_Q$ where $\widetilde{b}_P$ is fresh for all of $\Psi, \widetilde{b}_Q, Q, M$ and P, and that $\widetilde{b}_Q$ is correspondingly fresh. In the rule PAR introduced previously, we assume that $\mathscr{F}(Q) = (\nu \widetilde{b}_Q)\Psi_Q$ where $\widetilde{b}_Q$ is fresh for Ψ, P and α.

We show an example illustrating the interaction of the COM and OPEN rules. Let P and Q be as follows:

$$P = \underline{x}(\lambda z)z.\overline{z}a.\mathbf{0} \qquad\qquad Q = (\nu y)\overline{x}y.\underline{y}(\lambda z)z.\mathbf{0}$$

P receives a term on channel x and then sends a along the received channel. Q sends the private name y along x, and then receives a message along the

Process syntax	Well-formedness requisites
$\mathbf{0}$	
$(\!\|\Psi\|\!)$	
$\overline{M}N.P$	P well-formed.
$\underline{M}(\lambda\widetilde{x})N.P$	P well-formed, $\widetilde{x}$ distinct and $\widetilde{x} \subseteq N$.
$\mathbf{case}\ \widetilde{\varphi} : \widetilde{P}$	All $P \in \widetilde{P}$ well-formed and guarded.
$P \mid Q$	P and Q well-formed.
$(\nu x)P$	P well-formed.
$!P$	P well-formed and guarded.

Table 2.3. *Process syntax and well-formedness*

private name. Their parallel composition can act as follows, where we elide the frames since they will not impact the derivation:

$$\textsc{Com}\ \frac{\textsc{In}\ \dfrac{}{P \xrightarrow{\underline{x}y} \overline{y}a.\mathbf{0}} \qquad \textsc{Open}\ \dfrac{\textsc{Out}\ \dfrac{}{\overline{x}y.\underline{y}(\lambda z)z.\mathbf{0} \xrightarrow{\overline{x}y} \underline{y}(\lambda z)z.\mathbf{0}}}{Q \xrightarrow{\overline{x}(\nu y)y} \underline{y}(\lambda z)z.\mathbf{0}}}{P \mid Q \xrightarrow{\tau} (\nu y)(\overline{y}a.\mathbf{0} \mid \underline{y}(\lambda z)z.\mathbf{0})}$$

This enables a communication $(\nu y)(\overline{y}a.\mathbf{0} \mid \underline{y}(\lambda z)z.\mathbf{0}) \xrightarrow{\tau} (\nu y)(\mathbf{0} \mid \mathbf{0})$ along the local name y, whose derivation we do not show.

Replication

Finally, replication $!P$ is a means of expressing infinite behaviour, such as loops and recursion. Intuitively, it behaves as an infinite parallel composition $P \mid P \mid \ldots$ — this is captured by the rule

$$\textsc{Rep}\ \frac{\Psi \rhd P \mid !P \xrightarrow{\alpha} P'}{\Psi \rhd !P \xrightarrow{\alpha} P'}$$

$$\textsc{In}\ \frac{\Psi \vdash M \leftrightarrows K}{\Psi \rhd \underline{M}(\lambda\widetilde{y})N.P \xrightarrow{\underline{K}N[\widetilde{y}:=\widetilde{L}]} P[\widetilde{y}:=\widetilde{L}]} \qquad \textsc{Out}\ \frac{\Psi \vdash M \leftrightarrows K}{\Psi \rhd \overline{M}N.P \xrightarrow{\overline{K}N} P}$$

$$\textsc{Case}\ \frac{\Psi \rhd P_i \xrightarrow{\alpha} P' \qquad \Psi \vdash \varphi_i}{\Psi \rhd \mathbf{case}\widetilde{\varphi} : \widetilde{P} \xrightarrow{\alpha} P'}$$

$$\textsc{Com}\ \frac{\begin{array}{c}\Psi_Q \otimes \Psi \rhd P \xrightarrow{\overline{M}(\nu\widetilde{a})N} P' \\ \Psi_P \otimes \Psi \rhd Q \xrightarrow{\underline{K}N} Q' \qquad \Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrows K\end{array}}{\Psi \rhd P \mid Q \xrightarrow{\tau} (\nu\widetilde{a})(P' \mid Q')}\ \widetilde{a}\#Q$$

$$\textsc{Par}\ \frac{\Psi_Q \otimes \Psi \rhd P \xrightarrow{\alpha} P'}{\Psi \rhd P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \mathrm{bn}(\alpha)\#Q$$

$$\textsc{Scope}\ \frac{\Psi \rhd P \xrightarrow{\alpha} P'}{\Psi \rhd (\nu b)P \xrightarrow{\alpha} (\nu b)P'}\ b\#\alpha,\Psi$$

$$\textsc{Open}\ \frac{\Psi \rhd P \xrightarrow{\overline{M}(\nu\widetilde{a})N} P'}{\Psi \rhd (\nu b)P \xrightarrow{\overline{M}(\nu\widetilde{a}\cup\{b\})N} P'}\ \begin{array}{l} b\#\widetilde{a},\Psi,M \\ b \in \mathsf{n}(()N)\end{array} \qquad \textsc{Rep}\ \frac{\Psi \rhd P \mid !P \xrightarrow{\alpha} P'}{\Psi \rhd !P \xrightarrow{\alpha} P'}$$

Table 2.4. *Operational semantics. Symmetric versions of* COM *and* PAR *are elided. In the rule* COM *we assume that* $\mathscr{F}(P) = (\nu\widetilde{b_P})\Psi_P$ *and* $\mathscr{F}(Q) = (\nu\widetilde{b_Q})\Psi_Q$ *where* $\widetilde{b_P}$ *is fresh for all of* $\Psi, \widetilde{b_Q}, Q, M$ *and* P*, and that* $\widetilde{b_Q}$ *is correspondingly fresh. In the rule* PAR *we assume that* $\mathscr{F}(Q) = (\nu\widetilde{b_Q})\Psi_Q$ *where* $\widetilde{b_Q}$ *is fresh for* Ψ, P *and* α*. In* OPEN *the expression* $\tilde{a} \cup \{b\}$ *means the sequence* $\tilde{a}$ *with* b *inserted anywhere.*

2.6.3 Bisimulation

An attentive reader may have noticed that our way of defining the transition relation differs from the labelled transition systems introduced in Section 2.1 in two ways. First, it is parametrised on an assertion environment. This is in itself a rather innocent change: one could imagine an alternative presentation where the label is a cross-product of an assertion and an action, writing $P \xrightarrow{(\Psi,\alpha)} P'$ for $\Psi \rhd P \xrightarrow{\alpha} P'$. A more drastic change is that the label binds into the process after the arrow. This is common in process calculi with mobility, and means that our semantics is more closely related to the *Nominal SOS* approach of Cimini et al. [CMRG12] than Plotkin's original approach, though psi-calculi predate [CMRG12].

These differences from the LTS approach suggest a different definition of bisimulation than the one introduced in Section 2.2. The definition is as follows, where the *static equivalence* relation $\simeq$ relates frames if and only if they entail the same conditions.

Definition 16 (Strong bisimulation)**.** *A relation $\mathscr{R} \subseteq \mathbf{A} \times \mathbf{P} \times \mathbf{P}$ is a* strong bisimulation *iff for every $(\Psi, P, Q) \in \mathscr{R}$*

1. $\Psi \otimes \mathscr{F}(P) \simeq \Psi \otimes \mathscr{F}(Q)$ *(static equivalence)*
2. $(\Psi, Q, P) \in \mathscr{R}$ *(symmetry)*
3. $\forall \Psi'.(\Psi \otimes \Psi', P, Q) \in \mathscr{R}$ *(extension of arbitrary assertion)*
4. *If $\Psi \rhd P \xrightarrow{\alpha} P'$ and $\mathrm{bn}(\alpha)\#\Psi, Q$, then there exists Q' such that $\Psi \rhd Q \xrightarrow{\alpha} Q'$ and $(\Psi, P', Q') \in \mathscr{R}$ (simulation)*

We define bisimilarity $\dot{\sim}$ *to be the largest bisimulation, and write $\Psi \rhd P \dot{\sim} Q$, or $P \dot{\sim}_\Psi Q$, to mean $(\Psi, P, Q) \in \dot{\sim}$.*

We give the intuition behind the clauses in reverse order. Clause 16.4 is essentially the first clause of Definition 6, amended with freshness conditions. They ensure that the choice of concrete representatives for the bound names of α does not impact the derivation of the matching transition in Q. Clause 16.3 says that for two processes to be bisimilar in an environment Ψ, they must also be bisimilar in every extension of Ψ. Without this requisite, bisimilarity would not be preserved by the parallel operator. Clause 16.2 is simply a convenience that lets us avoid having two simulation clauses. Finally, Clause 16.1 states that two bisimilar processes must have equivalent frames. Hence we regard the frame of a process as part of its behaviour, in addition to its transition behaviour.

It is desirable for bisimilarity to be a *congruence*, i.e. closed under all operators of the language. Unfortunately, bisimilarity is not closed under the input construct in psi-calculi, for the same reason as in the pi-calculus. Suppose φ is a condition which is true in every environment. Then $P \dot{\sim}_\Psi \mathbf{case}\ \varphi : P$. However, suppose that an input prefix α may yield a substitution σ such that $\Psi \not\vdash \varphi\sigma$ when consumed. Then $\alpha.P \not\dot{\sim}_\Psi \alpha.\mathbf{case}\ \varphi : P$

The standard solution to such problems in name-passing calculi is to obtain a congruence by closing bisimilarity under all substitutions:

Definition 17 (Strong congruence)**.** Strong congruence $\sim$ *is defined as:*

$$P \sim_\Psi Q \triangleq \forall \sigma.\ P\sigma \dot{\sim}_\Psi Q\sigma$$

We write $P \sim Q$ to mean $P \sim_{\mathbf{1}} Q$.

Finally we mention *weak bisimilarity*, denoted $\dot{\approx}$, which is a notion of bisimilarity that abstracts away from the internal behaviour of processes. The intuition is that processes should only be considered equivalent if they can-

not be distinguished by another process observing them. This is achieved by refining the definition of bisimulation so that when a process imitates the behaviour of another process, it may perform any number of τ steps along with the matching behaviour. The precise definition of weak bisimilarity for psi-calculi is technically complicated, and will not be presented here. The interested reader will find it in Paper III; for a more thorough discussion we refer to [JBPV10, Joh10].

The main source of complications is that we allow non-monotonic assertion logics, where adding assertions to the environment may falsify conditions that held previously. Let τ be shorthand for a prefix whose only action is an internal action.[4] A reader might expect weak bisimulation to satisfy the following equation, as is the case in CCS:

$$\tau.P \mathrel{\dot{\approx}} P$$

After all, $\tau.P$ will unlock all the behaviour of P after just one internal transition. Alas, the equation is invalid. For an example, suppose the frame of P asserts $\neg\varphi$ for some condition φ that is initially true. Then consider

$$R \triangleq \mathbf{case}\ \varphi : Q \mid \tau.P \qquad\qquad S \triangleq \mathbf{case}\ \varphi : Q \mid P$$

The process R may behave as Q since φ holds, but since the frame of P falsifies the guard φ, S cannot. Hence the parallel context $\mathbf{case}\ \varphi : Q$ may distinguish $\tau.P$ from P, so they cannot be bisimilar. However, the law $\tau.P \mathrel{\dot{\approx}} P$ will still hold in interesting special cases, such as when the assertion logic is monotonic, or when P is assertion guarded. There are also other related laws that hold in the general case, such as

$$\tau.\tau.P \mathrel{\dot{\approx}} \tau.P$$

For the same reasons as in the pi-calculus, it turns out that weak bisimulation is not a congruence wrt. input and case. We obtain *weak congruence*, denoted $\approx$, in the standard manner: by closing weak bisimilarity under substitutions, and requiring that initial τ actions can be simulated by a non-empty sequence of τ actions.

2.6.4 Congruence and algebraic laws

The following results from [BJPV09, JBPV10] demonstrate that the notions of bisimulation discussed in Section 2.6.3 are useful, in the sense that they justify compositional and algebraic reasoning about processes. Establishing such

[4] For an example, this can be encoded as $[\![\tau.P]\!] = (\nu a)(\overline{a}a.[\![P]\!] \mid \underline{a}(\lambda\varepsilon)a.\mathbf{0})$ in a psi-calculus where channel equivalence is identity on names and names are terms.

results for a process calculus is often difficult, tedious and error-prone work; if the process calculus is a psi-calculus these results are already established.

First, strong bisimulation is a congruence with respect to all operators save for input (note the closure under substitutions in the premise of Clause 3.6):

Theorem 3 (Congruence properties of strong bisimulation)**.**

1. $\Psi \rhd P \dot{\sim} Q \quad \Rightarrow \quad \Psi \rhd P \mid R \dot{\sim} Q \mid R$
2. $\Psi \rhd P \dot{\sim} Q \quad \Rightarrow \quad \Psi \rhd (\nu x)P \dot{\sim} (\nu x)Q$ *if* $x\#\Psi$
3. $\Psi \rhd P \dot{\sim} Q \quad \Rightarrow \quad \Psi \rhd !P \dot{\sim} !Q$ *if* P *and* Q *are guarded*
4. $\forall i.\Psi \rhd P_i \dot{\sim} Q_i \quad \Rightarrow \quad \Psi \rhd \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \dot{\sim} \mathbf{case}\ \widetilde{\varphi} : \widetilde{Q}$ *if* $\widetilde{P}, \widetilde{Q}$ *are guarded*
5. $\Psi \rhd P \dot{\sim} Q \quad \Rightarrow \quad \Psi \rhd \overline{M}N.P \dot{\sim} \overline{M}N.Q$
6. $\forall \widetilde{T}.\Psi \rhd P[\widetilde{x} := \widetilde{T}] \dot{\sim} Q[\widetilde{x} := \widetilde{T}] \quad \Rightarrow \quad \Psi \rhd \underline{M}(\lambda \widetilde{x})N.P \dot{\sim} \underline{M}(\lambda \widetilde{x})N.Q$ *if* $\widetilde{x}\#\Psi$

Theorem 4. $\sim_\Psi$ *is a congruence for all* Ψ*.*

The following structural laws about strong bisimulation and congruence show that parallel composition forms an abelian monoid with $\mathbf{0}$ as unit, and that the scope of restrictions can be extended in a capture-avoiding manner.

Theorem 5 (Structural laws of strong congruence)**.**

1. $\Psi \rhd P \sim P \mid \mathbf{0}$
2. $\Psi \rhd P \mid (Q \mid R) \sim (P \mid Q) \mid R$
3. $\Psi \rhd P \mid Q \sim Q \mid P$
4. $\Psi \rhd (\nu a)\mathbf{0} \sim \mathbf{0}$
5. $\Psi \rhd P \mid (\nu a)Q \sim (\nu a)(P \mid Q)$ *if* $a\#P$
6. $\Psi \rhd \overline{M}N.(\nu a)P \sim (\nu a)\overline{M}N.P$ *if* $a\#M,N$
7. $\Psi \rhd \underline{M}(\lambda \widetilde{x})N.(\nu a)P \sim (\nu a)\underline{M}(\lambda \widetilde{x})N.P$ *if* $a\#\widetilde{x},M,N$
8. $\Psi \rhd \mathbf{case}\ \widetilde{\varphi} : \widetilde{(\nu a)P} \sim (\nu a)\mathbf{case}\ \widetilde{\varphi} : \widetilde{P}$ *if* $a\#\widetilde{\varphi}$ *and* $\widetilde{P}$ *are guarded*
9. $\Psi \rhd (\nu a)(\nu b)P \sim (\nu b)(\nu a)P$
10. $\Psi \rhd !P \sim P \mid !P$ *if* P *is guarded*

Finally, similar results apply also to weak bisimulation and congruence.

Theorem 6. *All congruence properties of* $\dot{\sim}$ *established in Theorem 3, except for Clause 3.4 pertaining to the case construct, also hold for* $\dot{\approx}$*.*

Theorem 7. $\approx$ *is a congruence.*

Theorem 8. *All structural laws of* $\sim$ *established in Theorem 5 also hold for* $\approx$*.*

A strong point of psi-calculi is that all theorems presented in this section have been formally proven using Nominal Isabelle. At roughly 35000 lines of Isabelle proof scripts [Ben12], this represents a cyclopean investment of labour by Bengtson that we adapt and re-use to formalise new results throughout this thesis.

3. Contributions

In this chapter, we summarise the novel contributions of this thesis. A major part of my contributions to joint papers are the Isabelle formalisations, but with the exception of Paper V they are rarely discussed explicitly in the papers themselves. For this reason, we will discuss the nature of the formalisations and the main challenges involved at a level of detail beyond that of the original papers.

3.1 Extensions of psi-calculi

This section summarises the content of Papers I–III. They share a common theme, which is to increase the expressiveness and modelling convenience of psi-calculi by extending its syntax and semantics.

3.1.1 Broadcast communication

In this section we summarise Paper I.

If we want to apply psi-calculi to reason about wireless communication, we immediately discover two rather awkward gaps between reality and modelling language. First, psi-calculi are based on point-to-point communication, where each message sent is received by exactly one process. Wireless communication is broadcast communication, where each transmission may be received by any number of processes, and possibly lost. Second, connectivity between nodes in wireless communication may be asymmetric, if some nodes have weaker transmitter than others; and intransitive, if the transmission range of my neighbours reaches beyond my own. Channel equivalence is ill-suited to model such connectivity, since we require it to be symmetric and transitive.

Broadcast psi-calculi extend psi-calculi to close the above gaps. It is a conservative extension since the existing mechanisms for point-to-point communication are kept, and play by the same rules as before. No change to the process syntax is needed; the existing input and output prefixes may now additionally denote broadcast input and broadcast output. Whether a particular prefix can be used for broadcasting is determined by the connectivity judgements $\Psi \rhd M \mathrel{\dot{\prec}} K$, meaning that in the environment Ψ, messages may be transmitted from M on the medium K. Conversely $\Psi \rhd K \mathrel{\dot{\prec}} N$ means that in Ψ, messages transmitted on the medium K may be received by N. By thus

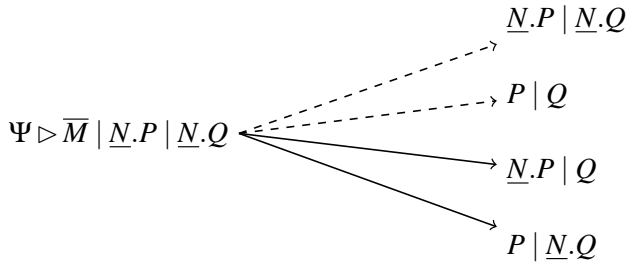

Figure 3.1. The possible behaviours of an unreliable broadcast action, assuming $\Psi \vdash M \dot{\prec} K \dot{\succ} N$. Dashed lines indicate transitions that are unavailable with point-to-point communication.

decoupling the ability to send from the ability to receive, we no longer need to impose symmetric and transitive connectivity. This comes at a price: for technical reasons related to scope extension, we require that a medium contains no more names than the transmitters and receivers that are connected to it.

The semantics of broadcast communication comprises five new rules added to the operational semantics; it is defined so that output is non-blocking, i.e. an output may fire at any time, regardless of whether anyone listens to the transmission. When an output fires, it may be received by unboundedly many listeners. The broadcast is *unreliable*, in the sense that a potential receiver may always fail to hear a transmission. Figure 3.1 shows the possible outcomes of broadcast communication in a simple system with one sender and two receivers.

The meta-theory pertaining to strong bisimulation from the original psi-calculi carry over to broadcast psi-calculi, and formal proofs in Isabelle have been carried out:

Theorem 9 (Theorems 8–10 of Paper I)**.** *All results about strong bisimulation and strong congruence described in Section 2.6 also apply to broadcast psi-calculi.*

We illustrate how broadcast psi-calculi can be used by applying it to analyse a routing protocol for mobile ad-hoc networks, namely LUNAR [TGRW04]. A tailored psi-calculus is defined, where the terms correspond to the type of messages under consideration in the LUNAR protocol, and the assertions describe the network topology as well as the routes that are currently established. It is shown that this modelling language is a psi-calculus, and we use it to prove a basic reachability property of the protocol.

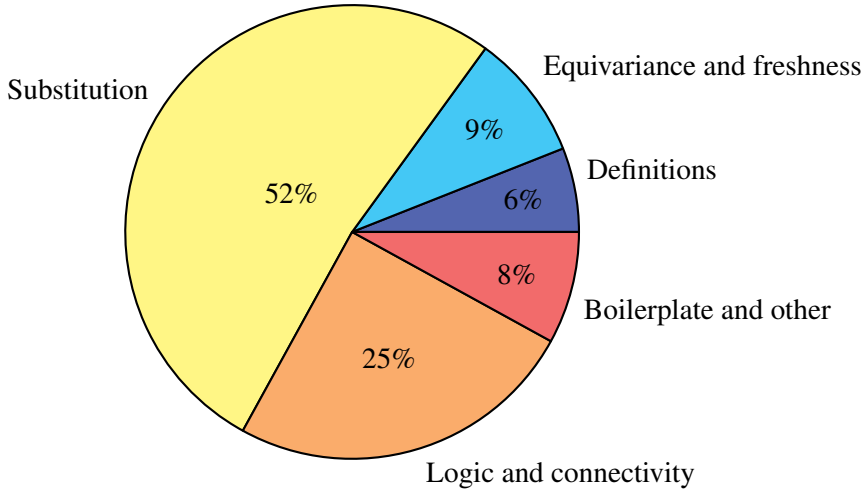

Figure 3.2. Anatomy of the LUNAR psi-calculus instantiation in Isabelle, (1297 LoC in total).

Formalising broadcast psi-calculi

Broadcast psi-calculi is, to the best of my knowledge, the first broadcast calculus with a machine-checked meta-theory. The treatment of syntax, semantics and meta-theory consists of approximately 33,000 lines of proof script, representing a substantial investment of labour. Fortunately, since it is a conservative extension of the original psi-calculi we did not have to start from scratch. The corresponding part of the original psi-calculi formalisation, which we reuse almost in entirety, comprises about 21,000 lines of code.

To a large extent, the task of extending the old proof scripts introduces difficulties that are more quantitative than qualitative in nature. Five new inference rules are added to the semantics, meaning five more cases to consider in every rule induction. This is largely grunt work, but some interesting challenges do present themselves in the bisimulation proofs. Among these, the foremost challenge is the proof that commuting binders preserves bisimilarity; for an account of the proof idea we refer to Paper I.

Beyond the meta-theoretical proofs, we also construct the most substantial instance of psi-calculi to date, namely the calculus in which we model the LUNAR protocol. We formally prove that this instantiation satisfies the requisites for being a psi-calculus. Through the locale interpretation mechanism, we then obtain mechanised syntax, semantics and bisimulation meta-theory for free. By studying the anatomy of this development, we may gain insight into what kind of effort is currently required to obtain a custom psi-calculus tailored for high-level protocol verification.

The total size of the development is 1297 lines of code. The contents of these lines are subdivided as in Figure 3.2. 6% of the lines comprise definitions of datatypes and functions, with a further 9% dedicated to establishing that these definitions behave well wrt. permutations and freshness. 52% of the code goes towards establishing that the substitution functions on the terms, assertions and conditions satisfies the three laws of substitution in Table 2.2. 26% go towards establishing the remaining requisites about the assertion logic and channel connectivity predicates: abelian monoid laws and compositionality of $\otimes$, symmetry and transitivity of $\leftrightarrows$, and support constraints on $\dot{\succ}$ and $\dot{\prec}$.

It is interesting to note that outside a theorem proving environment, only the 31% that comprises definitions, logic and connectivity are likely to receive any explicit treatment at all; technicalities about names and substitutions are tacitly treated according to the maxim "what could go wrong?" But occasionally, something does go wrong. For example, the polyadic pi-calculus instance as defined in the very first paper on psi-calculi [BJPV09] has no sensible definition of term substitution that satisfies the requisites. This suggests that substitution deserves to be treated carefully, and it is good that Isabelle forces us to do so. On the other hand, 52% seems excessive: the problem is not *that* deep. An interesting direction for future work would be proof automation to partially alleviate this burden. Initial efforts in this direction direction are reported in [ÅP10], though with the recent Eisbach tactics language for Isabelle [MWM14] the problem should now be amenable to a more lightweight approach.

3.1.2 Higher-order data

Here we summarise Paper II.

A process calculus is *higher-order* if processes are first-class citizens, i.e. if processes may be sent and received just like data terms. In this sense, psi-calculi are already higher-order: nothing prevents us from instantiating the parameters so that the processes are among the terms. Hence we can use processes as messages or communication channels, and do pattern matching on them, just like any other terms. What is missing is a mechanism for executing a process that has been received as a message. Higher-order psi-calculi fill this gap by making very small changes to the definition of psi-calculi; small enough so that we can give a complete account in just the following paragraph.

We introduce the **run** construct into the process syntax, where the process **run** M may act as any process that M is an alias for. What it means for M to be an alias for P is formalised by *clauses* of the form $M \Leftarrow P$, where we require that $\mathsf{n}(P) \subseteq \mathsf{n}(M)$ and that P is assertion guarded. The entailment relation is extended so that assertions may entail clauses in addition to conditions. A single new rule is introduced into the operational semantics:

$$\textsc{Invoke}\ \frac{\Psi \rhd P \xrightarrow{\alpha} P' \qquad \Psi \vdash M \Leftarrow P}{\Psi \rhd \mathbf{run}\, M \xrightarrow{\alpha} P'}$$

To obtain a process-passing calculus where received processes can be executed, it suffices to include processes among the terms, and let every process be an alias for itself: let $\Psi \vdash P \Leftarrow P$ hold for all Ψ, P. A process that receives another process on the channel N and then runs it can then be written

$$\underline{N}(\lambda x)x.\mathbf{run}\, x$$

and to run an arbitrary process P we may pair it with a sender $\overline{N}P$. But the main source of novelty and expressive power here is that we do not need to do this. A more flexible approach is to pass arbitrary terms as aliases, while asserting a clause for the corresponding process. The sender in the above example can then be written as:

$$\overline{N}M \mid (\!|M \Leftarrow P|\!)$$

This opens up several possibilities. We can describe non-determinism, since $N \Leftarrow P$ is not necessarily the only clause that pertains to the alias N. If $N \Leftarrow Q$ also holds, the process **run** N behaves exactly as $P + Q$. We can describe recursion, since the process P in $N \Leftarrow P$ may of course contain **run** N as a subprocess. Since the assertion environment may change, we can alter the behaviour of a particular alias by revealing new assertions pertaining to it. Hence, the binding between N and P in $N \Leftarrow P$ is a dynamic binding.

For every (first-order) psi-calculus we may define a *canonical higher-order instance*, which extends the original calculus so that parameterised clauses may be asserted. Hence we lift every psi-calculus to higher-order in one go.

Under certain natural requisites on the psi-calculus, we prove in Isabelle that the $+$ and ! operators can be encoded without using $+$ and ! in the same calculus. The encoding is compositional, and source and target are strongly bisimilar. Along similar lines, we also show that we can encode the n-ary **case** operator using only a unary **case** operator. In fact we may go further and drop **case** entirely, though this is not elaborated upon in Paper II: if the assertion logic under consideration features implication (denoted $\rightarrow$), unary **case** $\varphi : P$ may be represented as

$$(\nu a)(\mathbf{run}\, M_a \mid (\!|\varphi \rightarrow (M_a \Leftarrow P)|\!))$$

where a is a fresh name and M_a is a term that contains a and the names of P.

We show in Isabelle that all results from Section 2.6 about strong and weak bisimulation carry over to higher-order psi-calculi. However, this equivalence

is somewhat unsatisfactory in a higher-order language. From previous work on higher-order equivalences [Tho89, Tho93, San93] we have come to expect that if e.g. P and Q are bisimilar, then so is $\overline{N}P$ and $\overline{N}Q$. This fails to hold for our (first-order) equivalences since their output labels will be syntactically different. Intuitively, the reason such a law holds in many previous higher-order calculi is that the only thing one can do with processes is to send, receive and run them. Due to the generality of psi-calculi, processes can be used in many ways that allow bisimilar but distinct processes to be distinguished when passed as messages For an example, an input pattern may decompose a received process into its outermost operands, or a **case** statement may compare two processes for syntactic equality. Hence, if we insist that sending syntactically distinct but semantically equivalent processes as messages constitutes equivalent behaviour, there can be no sensible higher-order equivalence that applies to all psi-calculi.

In alias-passing calculi, this issue is somewhat besides the point; there is no need to equate the sending of equivalent process since we send aliases, not processes. Thus, we propose a notion of strong higher-order bisimilarity (henceforth *HO-bisimilarity*) that is more suited to the alias-passing paradigm. The idea is that if P and Q are HO-bisimilar, we also want $(\!|M \Leftarrow P|\!)$ and $(\!|M \Leftarrow Q|\!)$ to be HO-bisimilar. We obtain such an equivalence by slightly changing the static equivalence clause of Definition 16, and prove in Isabelle that the resulting equivalence satisfies all the algebraic and congruence laws familiar from Section 2.6.

Formalising higher-order psi-calculi

All theorems about higher-order psi-calculi reported on in Paper II are formally proven in Isabelle. In total, the development consists of 55,497 lines of code and represents many man-months of labour.[1] The contents of these lines is subdivided as shown in Figure 3.3.

55% of the development goes towards showing that the results about strong and weak bisimulation, and their induced congruences, are still valid in higher-order psi-calculi. This work represents nowhere near 55% of the development *effort*; it was completed in a matter of days. Similarly to broadcast psi-calculi, the work was done by starting from Bengtson's proofs and making changes as the need arises — usually all that is needed is to add an extra case for the INVOKE rule to inductive proofs. These new cases present few problems, for two reasons. First, the simplicity of the rule. Second, the rule is largely irrelevant to the most difficult proofs: those about transitions emanating from the parallel, restriction and replication operators. By comparison, the new rules for broadcast have complexity on par with the COM rule, and are relevant to all proofs about the aforementioned operators. We are also aided by our

[1] Paper II reports a figure of 63,334. This appears to be an error caused by including irrelevant files in the tally, that happened to reside in the same directory.

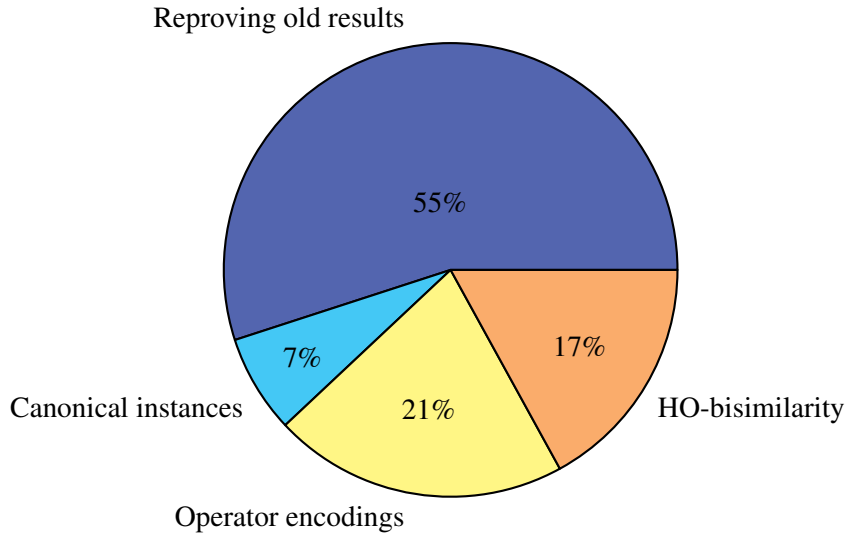

Figure 3.3. Anatomy of the Isabelle formalisation of higher-order psi-calculi (55,497 LoC total).

design decision to treat clauses as a subset of the conditions, similarly to how channel equivalence is treated, rather than as new syntactic constructs. This makes the old proof scripts more reusable by e.g. circumventing the need to redefine static equivalence.

17% of the proof scripts, but several man-months of labour, go towards defining HO-bisimilarity and showing that the algebraic and congruence properties of (first-order) bisimilarity carry over. The bulk of the effort here goes towards proving that HO-bisimilarity is preserved by parallel. On a high level, this proof turns out to require significant new ideas when compared to the first-order case, involving the use of sophisticated bisimulation up-to techniques. On a lower level, many new technical difficulties regarding the treatment of bound names present themselves. For an example, in several places we need to reason about the transitive closure of relations within inductive proofs where it is necessary to avoid clashes with certain bound names. Suppose that some names are appropriately fresh in P and Q. Suppose further that P and Q are related by the transitive closure of $\mathscr{R}$, so that for some $\tilde{R}$ we have

$$P \mathscr{R} R_0 \mathscr{R} \ldots \mathscr{R} R_n \mathscr{R} Q$$

The difficulty is that freshness in P and Q does not imply freshness in R_i. To overcome this difficulty we define a notion of *name-avoiding transitive closure*, which is a notion of transitive closure parameterised on a set of names that intermediate processes must avoid. This set of names must be carefully massaged to be large enough to guarantee freshness, yet small enough to guar-

antee inclusion in the relation under consideration. In performing this work, Isabelle has been an indispensable tool in discovering the precise formulation of technical lemmas and induction hypotheses. I surmise that without a proof assistant, finding e.g. which assumptions are needed for Lemma 4.29 from Paper II to hold, is beyond the reach of mere mortals.

21% of the proof scripts, and again several man-months of labour, go towards showing that the encodings of replication, choice and n-ary case are valid. While the encodings look natural and obvious to the human observer, it turns out that their correctness depends on many implicit assumptions about the assertion logic of the psi-calculus under consideration; for an example, that assertions in the outside environment cannot disable locally scoped clauses. Again, we believe that discovering precisely which assumptions are needed would have been very difficult without a proof assistant. In our bisimulation proofs, these assumptions manifest themselves as 10–15 additional requisites for membership in the candidate relation, making it very tedious to show membership in the candidate relation for derivative processes. The replication proof is further bogged down by a candidate relation that is closed under restriction and parallel composition; in Paper V we show how to significantly improve this proof by using bisimulation up-to context techniques.

The final 7% of the proof scripts concern canonical higher-order instances. We define them, prove that they satisfy the requisites for being a higher-order psi-calculus, and the requisites for the encodings discussed in the previous paragraph to be valid. The main complication here is, perhaps surprisingly, to define them. While canonical instances are rather simple to define on paper, we must resolve ambiguities in the paper definition, and circumvent several technical restrictions of the nominal logic package to define them in Isabelle. The way Definition 3.4 of Paper II is presented makes it unclear whether a canonical instance is second-order (where clauses may only contain first-order processes) or ω-order (where clauses may contain higher-order processes). The former would be easier to implement, but for increased generality we prefer instead the latter interpretation. As for the technical complications, we may not define nominal datatypes inside a locale. Even if we could, nominal datatypes may not have recursion beneath type constructors, so e.g. the following is not allowed:

```
nominal_datatype name_btree = Leaf name | Node "name_btree list"
```

To circumvent these limitations we define, before the locale, seven polymorphic mutually inductive datatypes that respectively encode: higher-order assertions, higher-order conditions, clauses, binding sequences in parameterised clauses, higher-order processes, input binding sequences in higher-order processes, and case expressions of higher-order processes. Since we could not reuse the existing type constructors for lists and processes, bijections between these pre-existing datatypes and our local copies must be manually defined and managed in proofs. Further, the support for defining functions on nominal

datatypes is quite limited: only structural recursion on the first argument, with the other arguments unchanged in recursive calls, is supported. The result is a formalisation that is difficult to read and clumsy to reason about. We believe this tedium could be partially alleviated if recent advances in datatype representation [BHL+14] and local type definitions [KP] for Isabelle could be integrated with the nominal package.

3.1.3 Generalised pattern matching

This section is based on Paper III.

As described in Section 2.6, pattern matching in the original psi-calculi is defined in terms of substitution, an approach that is common in the literature [Gel85, HJ06] but inconvenient in e.g. cryptographic applications.

In calculi for cryptography, encryption and decryption are often modelled using binary function symbols **enc** and **dec**, respectively. Here $\mathbf{enc}(m,k)$ represents the encryption of message m with key k, and $\mathbf{dec}(c,k)$ represents the decryption of ciphertext c with key k. This seems easy to accommodate in psi-calculi by simply adding these constructs into the term language, but a closer look reveals that psi-calculi has some undesirable properties for this scenario. First, it would be nice to incorporate a rewrite rule $\mathbf{dec}(\mathbf{enc}(m,k),k) \to m$ for decrypting encrypted messages into the term language. Such rewrites could be performed as a subroutine of the substitution function. But if the key carries names, this would violate the name preservation law of term substitution which states that all names in $\widetilde{T}$ must occur in $X[\widetilde{x} := \widetilde{T}]$ if $\widetilde{x} \subseteq \mathsf{n}(X)$. As an example, consider the substitution $\mathbf{dec}(x,k)[x := \mathbf{enc}(m,k)] = m$ where k does not occur in the result.

Moreover, the pattern matching facility of psi-calculi is such that every term is a pattern, and every name in a pattern can be considered a pattern variable. For an example, the input prefix $\underline{M}(\lambda x)\mathbf{enc}(m,x).P$ receives an encrypted message and uses pattern matching to extract the key from it, which is clearly not the intention of a cryptographic model.

We address these shortcomings by decoupling terms from patterns, and decoupling pattern matching from substitution. Hence we introduce the *patterns* $\mathbf{X}$, ranged over by X, as a new parameter of psi-calculi. Input prefixes are now of kind $\underline{M}(\lambda\widetilde{x})X$. When defining a psi-calculus, we may impose custom restrictions on which names in a pattern may be λ-bound. Through this mechanism we may repair the degenerate cryptographic example above, by disallowing the λ-binding of k in $\mathbf{enc}(m,x)$.

The indirect definition of pattern matching via substitution is supplanted by a direct definition of precisely which substitutions may arise when an input pattern interacts with a received message. Formally this is captured by a predicate MATCH, which given a pattern X, input binders $\widetilde{x}$, a message M, and a term sequences $\widetilde{T}$, is true iff the substitution $[\widetilde{x} := \widetilde{T}]$ may arise when

M is received by $(\lambda\widetilde{x})X$. In particular, if there are no substitutions that may arise through interaction of a particular message and pattern, the message may not be received. For an example, suppose messages are tuples (d,c) of datums d and MD5 checksums c. A calculus where an input prefix $\underline{M}(\lambda x)x$ will receive the datum iff the checksum is correct can be obtained by letting $\text{MATCH}(x,x,(d,c),e)$ hold iff $d = e$ and $d = \mathbf{MD5}(c)$.

This allows for very fine-grained control of pattern matching when defining a psi-calculus. Moreover, we no longer need the name preservation law of substitution, since its only use was to ensure that pattern matching satisfies certain name preservation properties. With pattern matching and substitution decoupled, we are no longer barred from incorporating the above example of term rewriting into the substitution function.

The usual meta-theory of bisimulation carries over:

Theorem 10 (Theorem 3.13 of Paper III)**.** *All the results in Section 2.6 also apply to psi-calculi with generalised pattern matching.*

In Paper III, several examples demonstrate how these features can be instantiated to capture notions of both deterministic and non-deterministic computation such as Peano arithmetic, Dolev-Yao message algebras [DY83], and the lambda calculus with an ambiguous choice operator [McC63]. We also show a kind of subject reduction property: the set of well-formed processes is preserved by the transition relation, which is not necessarily the case in the original psi-calculi.

Formalising generalised pattern matching

The proof of Theorem 10 is mechanised in Isabelle. This effort presented little difficulty: the changes to the language are mostly local to the rule for input, and the majority of the proofs require no change at all. The bulk of the effort was to prove formally that the new notion of pattern matching is well-behaved wrt. the treatment of bound names. Scenarios like these, where existing proofs must be re-checked under small changes to the definitions, are among those where interactive theorem provers truly shine: through a very small labour investment we obtain complete certainty that under the new definitions, nothing broke.

3.1.4 Sorts

This section is based on Paper III.

In the polyadic pi-calculus [Mil91], names are channels on which sequences of names can be transmitted. At first glance, this seems easy to represent as a psi-calculus by letting $\mathbf{T} = \mathcal{N}^\star$ and setting channel equivalence to be syntactic equality on names. However, note that substitutions in psi-calculi are total

functions that replaces names with arbitrary terms — hence substitutions such as

$$\langle x_1, \ldots, x_n \rangle [x_i := \langle y_1, \ldots, y_m \rangle]$$

must be accounted for, even though no input prefix in the polyadic pi-calculus could give rise to it. A solution is to extend the set of terms from sequences of names to n-ary trees with names as leaves. But if our intention is to exactly capture the polyadic pi-calculus, this solution is unattractive; we have introduced "junk" into the term language that bears no relation to anything in the polyadic pi-calculus.

In sorted psi-calculi, we refine the notion of names so that there is a countable set of *name sorts* $\mathscr{S}_{\mathscr{N}}$. For each sort $s \in \mathscr{S}_{\mathscr{N}}$ there is a countably infinite set of names, disjoint from the names of all other sorts. Terms and patterns are also assigned sorts (not necessarily disjoint from the name sorts). Let $\text{SORT}(M)$ denote the sort of M. When defining a psi-calculus, we may restrict the set of well-formed processes by instantiating the following *compatibility predicates*:

$\underline{\propto}$	$\subseteq$	$\mathscr{S} \times \mathscr{S}$	Can be used to receive
$\overline{\propto}$	$\subseteq$	$\mathscr{S} \times \mathscr{S}$	Can be used to send
$\prec$	$\subseteq$	$\mathscr{S} \times \mathscr{S}$	Can be substituted by
$\mathscr{S}_\nu$	$\subseteq$	$\mathscr{S}$	Can be bound by name restriction

For an example, the process $\overline{M}N.\mathbf{0}$ is well-formed iff $\text{SORT}(M) \mathrel{\overline{\propto}} \text{SORT}(N)$. The substitutability relation $\prec$ does not directly impact the process syntax, but is used to restrict the set of admissible substitutions. Returning to the polyadic pi-calculus, we can now remedy the problem of junk terms. Let $\text{SORT}(x) = \mathbf{name}$ and $\text{SORT}(\langle x_1, \ldots, x_n \rangle) = \mathbf{seq}$. By defining $\prec = \{(\mathbf{name}, \mathbf{name})\}$ we make sure that names cannot be substituted by sequences of names, so we need not account for substitutions that generate junk. If we also let $\underline{\propto} = \overline{\propto} = \{(\mathbf{name}, \mathbf{seq})\}$ the well-formed input and output prefixes will be precisely those of the polyadic pi-calculus, namely those where channels are names and messages are sequences of names.

With such fine-grained control over which processes are well-formed and which are not, we are able to encode existing process calculi with very strong quality criteria; so strong that we prefer to call them *representations* rather than encodings. A representation is an encoding that is compositional and satisfies strong operational correspondence of labelled transitions. A *complete representation* is one where for every process Q of the target language, $[\![P]\!] \sim Q$ for some P. Intuitively, this requisite captures junk freedom, in the sense that the target language contains no behaviour that is absent in the source language. We obtain a complete representations of both sorted and unsorted polyadic pi-calculus, and of value-passing CCS [Mil89]. We also obtain a representation of the polyadic synchronisation pi-calculus [CM03].

Formalising sorts
Unfortunately, formalising this sort system in its entirety is currently beyond the reach of Nominal Isabelle. While there is support for having multiple name sorts, each name sort must be individually declared before use. For an example, if we want to distinguish between names and variables we may begin our development as follows:

atom_decl *name*
atom_decl *variable*

In our case this is of little use, since the set of sorts under consideration is a (locale) parameter rather than a priori knowledge. Even if we had a mechanism for taking a set of sort names, and issuing **atom_decl** commands for each of them, further problems arise. When defining the inductive datatype representing process syntax, the disjointness of the name sorts means that rather than a single restriction operator, we would need one distinct restriction operator per name sort under consideration. Even worse, the set of sorts may be infinite, meaning that we would need to issue infinitely many **atom_decl** commands, and define inductive datatypes with infinitely many constructors. Clearly this is not the way forward, and we should seriously consider whether Nominal Isabelle needs further development, or whether it is the right tool for the job in the first place. We could make some progress if we migrate to version 2 of Nominal Isabelle [HU10], where there is experimental support for multi-sorted atoms. Unfortunately, the set of sorts must then be countable and fixed in advance, so we would not have as much parametricity as we would like.

In the meantime, we formalise a manageable subset of the sort system. This subset admits arbitrarily many term sorts, but only a single name sort. Of the previously discussed calculi, this subset suffices to represent the unsorted polyadic pi-calculus and the polyadic synchronisation pi-calculus, but not value-passing CCS or the sorted polyadic pi-calculus. The usual results about bisimulation require no new effort since the semantics is unchanged. However, the proofs about bisimulation congruence change slightly since we only consider well-sorted substitutions. The main property of interest we prove is subject reduction: that derivatives of well-formed processes are well-formed; at 170 lines, this is a relatively straightforward development.

3.2 Bisimulation up-to techniques

In this section we summarise Paper V.

Bisimulation up-to techniques are methods for reducing the size of relations needed in bisimulation proofs. Without up-to techniques, showing that a relation $\mathscr{R}$ is a bisimulation essentially boils down to showing that the leftmost diagram in Figure 3.4 commutes for all pairs $(P,Q) \in \mathscr{R}$.

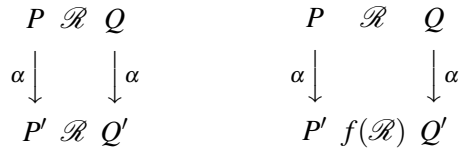

Figure 3.4. Illustrations of the bisimulation proof method (left), and the bisimulation up-to f proof method (right).

Note that the same relation is used as both the source and target of the transition, corresponding to the upper and lower parts the diagram. Small relations are desirable since it means fewer transitions to follow in the upper part. Large relations are desirable since it makes it easier to close the diagram in the lower part. So should we choose $\mathscr{R}$ to be small or large? With up-to techniques we can have it both ways, by using functions to enlarge $\mathscr{R}$. A *sound* up-to technique is a function f on relations such that the rightmost diagram in Figure 3.4 suffices to prove $\mathscr{R} \subseteq \dot{\sim}$.

As extensions of psi-calculi grow in complexity, so does the bisimulation up-to techniques used to prove their meta-theory, and of course each new such proof technique must be proven to be sound. Since the up-to techniques used often share many elements, some proof re-use would be desirable, but it is not clear how to combine old soundness results to derive new ones in general. This leads to duplicated effort in both the soundness proofs themselves, and in the bisimulation proofs in lieu of more advanced proof techniques.

Sangiorgi [San98] and Pous [Pou07a] have singled out subsets of the sound up-to techniques that enjoy nice compositionality properties; these are respectively called the *respectful* and the *compatible* functions. They are both closed under useful constructors such as union, composition and chaining. Hence, the soundness of elaborate up-to techniques follows immediately from the soundness of smaller building blocks. The compatible functions allow us to consider arbitrary coinductive objects and not just bisimulation. This turns out to be helpful in the case of psi-calculi since our definition of bisimulation is different from standard LTS bisimulation. Below we only present the special case where the objects under consideration are relations.

Let f, g, h range over functions from relations to relations. We define an ordering on functions pointwise, so that $f \leq g$ holds if for every relation $\mathscr{R}$ we have that $f(\mathscr{R}) \subseteq g(\mathscr{R})$. Recall from Sections 2.2 and 2.5.1 that $\mathbf{gfp}(g)$ denotes the greatest fixed-point of g, and that a post-fixed point of g is any relation $\mathscr{R}$ satisfying $\mathscr{R} \subseteq g(\mathscr{R})$. A function f is *g-compatible* wrt. the monotone function g iff f semi-commutes with g, i.e. if

$$f \circ g \leq g \circ f$$

Every g-compatible function f enjoys the property that whenever a relation satisfies $\mathscr{R} \subseteq g(f(\mathscr{R}))$, then $\mathscr{R} \subseteq \mathbf{gfp}(g)$. In the special case of b-compatible functions, where b is the bisimilarity functional from Section 2.2, this is precisely soundness of the rightmost diagram in Figure 3.4.

Without specifying g, one can prove that the identity function, and the constant functions that always return post-fixed points of g, are g-compatible; and that function composition and union preserve g-compatibility. In the special case of b, we further have that chaining and transitive closure preserve b-compatibility. In CCS, the context closure

$$\{(C[P], C[Q]) : (P, Q) \in \mathscr{R}\}$$

also preserves b-compatibility.

Our contribution is to apply this theory of compatible functions to psi-calculi, where the main difference from previous work on up-to techniques involves the treatment of assertions. Getting started is easy: all we need is a functional b whose greatest fixed point is precisely bisimilarity on psi-calculi, as seen in Definition 16, and we get the general results above for free. However, this b is not unique. Different choices of b will lead to different notions of sound functions, compatible functions, compositionality properties, and different proof obligations in bisimulation proofs. Choosing a b that strikes a good balance between these aims is the main design decision involved. A natural first candidate to consider is the one implicitly used in Definition 16, whose post-fixed points are the bisimulation relations. While a workable choice, it is somewhat inflexible: the symmetry clause makes it difficult to reason about asymmetric candidate relations in proofs, and it is unclear whether closure under parallel contexts is compatible. The functional we prefer makes trade-offs that result in pros and cons as follows:

+ Admits candidate relations that are asymmetric, non-equivariant and not closed under extension of the assertion environment.
+ Chaining, transitive closure, and most context closures are compatible, including the practically important cases of restriction and parallel contexts.
+ Name permutations, and rewriting of frames modulo static equivalence, are compatible.
− The proof obligation for the simulation case of bisimulation proofs is strengthened, so that instead of the derivative processes being related by the current environment, they must be related in every extension of the current environment.
− The closure of a relation under extension is not sound. Hence singleton relations of the form $\{(\Psi, P, Q)\}$ are rarely admissible in practice; it is usually necessary to consider instead relations with arbitrary environments, i.e.

$$\bigcup_{\Psi}\{(\Psi, P, Q)\}$$

The two negative points seem more like annoyances than serious problems. For the first negative point, in our case studies we find that the cost of discharging the extra proof obligation is a handful of lines at worst. For the second negative point, though one relation is singleton and the other infinite, the difference in practice between their associated bisimulation proofs is just one extra universal generalisation.

We formalise all our definitions and theorems in Nominal Isabelle, including compatibility of the important bisimulation up-to context techniques. Compatibility proofs involve a simulation case which is similar to that of bisimulation proofs, but uses different relations in the top and bottom parts of the diagram. Here we benefit greatly from Bengtson's foresight when developing the original psi-calculi formalisation: simulation proofs are factored out of the congruence proofs for bisimulation, so as to be reusable in other contexts. For example, Bengtson proves a lemma[2] that given a triple of the form $(\Psi, P \mid R, Q \mid R)$ and a relation $\mathscr{R}$ satisfying side-conditions that are too many to mention here, all transitions from $(\Psi, P \mid R, Q \mid R)$ lead to $\mathscr{R}$. Often, these lemmas are directly applicable to our setting. Sometimes the side-conditions turn out to be overly tuned to bisimulation proofs, and we must find other side-conditions that hold in our setting; the most technically challenging proof is to derive such a simulation lemma for triples as above with an outermost parallel operator.

As case studies, we show two examples where the use of up-to techniques has simplified our proofs of known results. The most drastic simplification is obtained in the correctness proof for the encoding of replication discussed in Section 3.1.2, where bisimulation up-to context techniques allows us to shorten the proof from 8788 lines to 3263 lines. We also prove a few structural laws about the replication operator.

From a proof engineering perspective, a measure of our success is that adding up-to techniques to the formalisation actually decreases its overall size, when we account for the simplifications to existing bisimulation proofs they enable. Deriving the new proof techniques adds about 3750 lines of code to the psi-calculi formalisation. This is smaller than the decrease in size of just the correctness proof for the replication encoding.

3.3 Encoding priorities in psi-calculi

In this section we summarise Paper IV.

Priorities allow certain actions to take precedence over others. This is useful when modelling systems because it admits more fine-grained control over

[2]Lemma 27.38 of [Ben10].

the model's behaviour. Phenomena that exhibit prioritised behaviour include e.g. interrupts in operating systems, and exception handling in programming languages.

A common approach to implementing priorities in the literature is to explicitly annotate prefixes with a priority level [CH88, BGLG93, Pra94]; we call this *action priorities*. Low priority actions are only available if no synchronisation between high priority actions is possible. This is a *static* approach to priority, in that the priority level of a particular action is not subject to change. *Dynamic* priorities, by contrast, allow the priority level of particular actions to change as the system evolves.

We show that augmenting psi-calculi with dynamic action priorities adds no expressive power.

First, we define an extension of psi-calculi with priorities, where the entailment relation is extended to judgements $\Psi \vdash \mathtt{prio}(M) = p$, meaning that communication on the channel M has priority level $p \in \mathbb{N}$ in the environment Ψ. We use the convention that 0 denotes the *highest* priority. We make appropriate changes to the semantics to make sure the priorities are enforced. We show how the extension with priorities can be instantiated to capture the $\pi@$ calculus [Ver07], and prove in Isabelle that the structural and algebraic laws of strong bisimulation from Section 2.6 remain valid under the new semantics.

Second, we show that the expressive power of this extension is already available without it. For each psi-calculus with priorities we construct a psi-calculus without priorities, and an associated encoding function from the former to the latter. The key insight behind the translation is that since channel equivalence depends on the assertion environment, all transitions involving any particular channel M can be disabled by making sure the environment entails no channel equivalences $M \leftrightarrows N$ involving it. Priorities disable certain transitions based on what prefixes are on offer at the top level. Hence, if we let the assertion environment include information about what the top level prefixes are, then the channel equivalence judgement can defined so that it allows actions only on the highest-priority channels currently available; thus, we can have prioritised behaviour without endowing psi-calculi with an explicit notion of priorities.

A main technical difficulty with the encoding is that after a transition has been taken, the assertion environment must be updated to reflect the absence of the thus consumed prefixes. Otherwise, high-priority prefixes long since consumed will block low-priority prefixes forever. This necessitates a non-monotonic assertion logic, which is difficult to define in a compositional manner. We address this by an approach based on multisets of prefixes with negative occurrence. While this leads to a rather involved definition of entailment, the translation function is very simple and satisfies strong correspondence properties. The encoding is homomorphic on all operators except prefixes, which are encoded as follows

$$[\![\alpha.P]\!] = (\!|\alpha|\!) \mid \alpha.((\!|-\alpha|\!) \mid [\![P]\!])$$

In other words, in parallel to every prefix we add a positive occurrence of it to the assertion environment. In parallel with its continuation, we add a negative occurrence to cancel the previous positive occurrence once the transition has been taken.

For an example of how the encoding operates, consider a prioritised psi-calculus where the channels are names annotated with a priority level a_p such that $\Psi \vdash \mathtt{prio}(a_p) = p$, and where channel equivalence is syntactic equality. Consider the process

$$P \triangleq \overline{a_0} \mid a_0 \mid b_1$$

Initially we have that $P \xrightarrow{\overline{b_1}}\!\!\!\!\!/$, since the high-priority synchronisation on a takes precedence. The encoding of P is as follows, modulo some garbage collection:

$$[\![P]\!] = (\!|\overline{a_0}|\!) \mid \overline{a_0}.(\!|-\overline{a_0}|\!) \mid (\!|a_0|\!) \mid a_0.(\!|-a_0|\!) \mid (\!|b_1|\!) \mid b_1.(\!|-b_1|\!)$$

We have that $\mathscr{F}([\![P]\!]) = \{\overline{a_0}, a_0, b_1\}$, i.e. precisely the multiset of top-level prefixes in P. There is no initial $\overline{b_1}$-transition from $[\![P]\!]$ because in order to derive one, $\overline{b_1}$ must be channel equivalent to something in the environment $\{\overline{a_0}, a_0, b_1\}$. Since $\mathscr{F}([\![P]\!])$ contains two matching prefixes that can synchronise at the highest priority level 0, channels with lower priority are not channel equivalent to anything. However, there is a transition $[\![P]\!] \xrightarrow{\tau} Q'$ for

$$Q' \triangleq (\!|\overline{a_0}|\!) \mid (\!|-\overline{a_0}|\!) \mid (\!|a_0|\!) \mid (\!|-a_0|\!) \mid (\!|b_1|\!) \mid b_1.(\!|-b_1|\!)$$

By unguarding the negative occurrences of the consumed prefixes, the frame evolves to

$$\mathscr{F}(Q') = \{\overline{a_0}, -\overline{a_0}, a_0, -a_0, b_1\} \simeq \{b_1\}$$

Hence $\mathscr{F}(Q') \vdash b_1 \leftrightarroweq b_1$ holds, and we may derive a transition $Q' \xrightarrow{b_1} Q''$ for some Q''.

This encoding enjoys very strong quality criteria; for an example, it satisfies every criterion discussed in Section 2.3 save for full abstraction. In particular we have strong operational correspondence of labelled transitions wrt. structural congruence. By contrast, most encodings studied in the literature tend to introduce a protocol that uses many target language transitions to encode one source language transition.

Formalising priorities

When implementing the semantics of psi-calculi with priorities in Isabelle, the main technical challenge is how deal with *negative premises* [Gro93] in

the operational semantics, i.e. rules where the absence of some transitions are premises for others. Such premises arise naturally in a setting with priorities. For an example, consider a CCS-like language where $\underline{\tau}$ is a high priority internal action, and τ is low priority. A communication rule for low-priority actions can be written

$$\frac{P \xrightarrow{\bar{a}} P' \qquad Q \xrightarrow{a} Q' \qquad P \mid Q \xrightarrow{\underline{\tau}}\!\!\!\!\!\!/\;\;}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

The third premise makes sure we cannot derive a low priority action if there is a high priority action available. While this is arguably the most natural way to define prioritised semantics, it is difficult to reason about. Because of the negative premise the rule is not monotonic, and the semantics cannot be read as an inductive definition in the usual sense. The problem of giving meaning to operational semantics with negative premises has been treated by several approaches in the literature [Gro93, BG96, vG04], and we could base our Isabelle implementation on them. But doing so would create a significant gap between our formalisation and the pre-existing psi-calculi formalisation by Bengtson; a gap that we would like to keep as small as possible in order to facilitate proof re-use. For an example, stratifying our semantics as in [Gro93] would imply that wherever Bengtson may proceed by induction on the depth of inference, we need to use (possibly transfinite) induction on strata.

Our solution is that whenever a negative premise $P \xrightarrow{\alpha}\!\!\!\!\!\!/\;\;$ would occur in the semantics, we use instead a premise $P \xrightarrow{\alpha}\!\!\!\!\!\!/\;\;_U$ on the auxiliary transition relation $\longrightarrow_U$. This relation is defined by precisely the rules of $\longrightarrow$, but without the negative premises. Thus all premises are positive, and both $\longrightarrow_U$ and $\longrightarrow$ are proper inductive definitions. The disadvantage of this approach is, of course, that we need to implement two transition relations instead of one. The cost of doing so, however, is quite low because of their similarity to each other. In particular, re-use is facilitated by the fact that $\longrightarrow_U$ is almost exactly the original psi-calculi semantics, and that $\longrightarrow \subseteq \longrightarrow_U$. That this approach defines the same semantics as the approach with explicit negative premises is shown in [ÅPBP+13].

3.4 Expressiveness of monotonic parallel composition

In this section, we take a step back from psi-calculi and consider instead arbitrary transition systems endowed with a notion of parallel composition. It is based on Paper VI.

We say that a parallel operator $\mid$ is *monotonic* if one operand cannot constrain the behaviour of the other; or in other words, if the following rule holds:

$$\textsc{Par}\ \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q}$$

We offer a simple and general method for obtaining separation results between models of concurrency, based on little more than whether the parallel operator is monotonic or not. The separation is obtained by considering solvability of a consensus problem, that can be intuitively stated as follows:

n observationally equivalent greasers walk into a bar. The greasers may, possibly after conferring with each other, non-deterministically declare that either Elvis Presley or Jerry Lee Lewis is the greatest singer of all time. Once either Elvis or Jerry has been declared, the other may not be declared; otherwise a bar fight will break out. A solution to the consensus problem is to find a process P, representing a greaser, such that no matter what n is, both Elvis and Jerry are possible outcomes and it can be guaranteed that a fight will never break out. Such a solution we call a *consensus process*.

The main insight is that with monotonic parallel composition, there cannot exist a consensus process. Because of the PAR rule the greasers may always split into two cliques that independently reach different outcomes. It follows immediately that a model of concurrency that contains a consensus process cannot be encoded into one that has monotonic parallel composition, if we require that the encoding translates consensus processes into consensus processes.

There are similar separation results to be found in the literature [EM99, Phi01, VBG07]; these are formulated for particular process calculi, and are derived from the impossibility of solving a certain kind of the leader election problem with the monotonic parallel operators of CCS and the pi-calculus. Our work improves on this state of the art in two main ways.

- Leveraging consensus instead of leader election admits a simpler problem statement. This allows us to strengthen the results of [EM99, Phi01, VBG07], since the criteria on encodings needed to preserve consensus processes are weaker than those needed to preserve electoral systems.
- We abstract away from the particulars of any one process calculus, giving a general separation result. From the general result, many particular separation results follow immediately as special cases. We illustrate this point by showing applications to calculi across a wide spectrum of application domains: broadcast communication, priorities, concurrent constraints, time, name fusion, and psi-calculi.

The abstract result goes as follows. A *composition* $\otimes$ is an associative and communicative binary function on the processes of a transition system. A composition is *monotonic* if $P \Longrightarrow P'$ implies $P \otimes Q \Longrightarrow P' \otimes Q$ for all P, P', Q. Let $P^n_\otimes$ denote n copies of P composed using $\otimes$, so e.g. $P^2_\otimes = P \otimes P$.

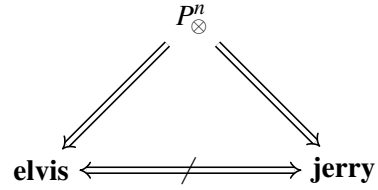

Figure 3.5. Illustration of a consensus process.

Let **elvis**, **jerry** be two process predicates. We say that P is a *consensus process over* $\otimes$ if for all $n > 1$, $P^n_\otimes$ behaves as illustrated in Figure 3.5: from P^n, states where **elvis** holds and states where **jerry** holds are both reachable; but **elvis** is not reachable from **jerry**, and vice versa. We also assume that the set of states satisfying **elvis**, **jerry** is closed under composition with derivatives of P^n.

The above two paragraphs contain all the background we need to state our main result:

Theorem 11. *There are no consensus processes over monotonic composition operators.*

In order to derive relative expressiveness results from this theorem, we define two sets of criteria on encodings, and prove that they both map consensus processes to consensus processes. The first kind of encoding satisfies weak operational correspondence and is **elvis**, **jerry**-sensitive. The second kind satisfies computational correspondence. Both sets of criteria use the novel compositionality-style requisite

$$[\![P \otimes_S Q]\!] = [\![P]\!] \otimes_T [\![Q]\!]$$

Where $\otimes_S, \otimes_T$ are the compositions used to compose greasers in the source and target language, respectively. This is weaker than homomorphism since the compositions need not be operators in the language, and orthogonal to compositionality since compositions need not be contexts and contexts need not be associative, commutative or monotonic.

Formalising monotonic parallel composition

We formalise all results from Paper VI that are independent of particular process calculi. Our mechanisation consists of 1000 lines of code and took less than a week in total to develop, meaning that the marginal cost of mechanising these results has been rather small. Since binders are irrelevant to these proofs, we may work in Isabelle/HOL rather than Nominal Isabelle. In fact, the setting of our proofs is just the LTS locale of Section 2.5.1, augmented so

that it also assumes the existence and algebraic properties of the composition function $\otimes$.

Perhaps surprisingly, the most challenging work was the proof of a technical lemma concerning computations: namely, that for every process P there is a maximal computation starting from P. We have chosen to mechanise computations using Lochbihler's formalisation of coinductive lists [Loc10]. This allows us to handle finite and infinite computations in a uniform way. In particular we can do coinductive proofs about list predicates, similarly to the coinductive proofs about bisimulation relations in Sections 2.2 and 2.5.1. Since we do not have a concrete transition system to work with, obtaining a witness to the existence of a maximal computation is problematic. We achieve this using iteration over the Hilbert choice function ε, which given a predicate A returns an arbitrary witness to the predicate if one exists (using the axiom of choice). We then prove by coinduction that the sequence $P_0, P_1, \ldots$ is a maximal computation from P, where $P_0 = P$, and $P_{n+1} = \varepsilon(\lambda P'.\ P_n \longrightarrow P')$, and the candidate predicate is the set of all such iterations from P. It is unclear to us if a proof exists that avoids using the axiom of choice, at least if we insist on carrying out the proof in an abstract setting.

3.5 Summary

In summary, the main contributions of this thesis are the following.

- Extensions of psi-calculi with several new language features: broadcast communication, higher-order data, priorities, a sort system, and better pattern matching.
- Formal proofs in Isabelle that these extensions preserve the bisimulation meta-theory of psi-calculi, and formal proofs of some new results concerning meta-theory, expressiveness and applications.
- A high quality encoding from the extension of psi-calculi with priorities to the original psi-calculi.
- Compositional techniques for simplifying bisimulation proofs that apply to all psi-calculi, with applications to making the Isabelle proof archive more maintainable.
- A simple and widely applicable method for establishing separation results between languages, based on whether parallel components may behave independently or not.
- Applying this method to strengthen four separation results from the literature, and derive four novel separation results.

4. Related work

In this chapter we discuss related work. The reader may also consult Papers I–VI, where more detailed comments on some of these works, and others not mentioned here, can be found.

4.1 Process calculi

The field of process calculi has a long history, and the earliest calculi go back to the 1970's and 1980's, with three main branches originating from CCS [Mil80], CSP [BHR84] and ACP [BK84]; ours is the CCS branch. For a historical overview we refer to Baeten [Bae05].

Psi-calculi has its roots in the pi-calculus [MPW92] by Milner, Parrow and Walker. It is a fundamental calculus for concurrent processes in much the same way as the λ-calculus is a fundamental calculus for sequential processes. The communication model is one-to-one synchronous communication, and the only messages that can be exchanged are channel names. The number of variants and extensions of the pi-calculus that have been proposed are far too many to mention here. We focus only on two that are particularly relevant, namely the spi calculus and the applied pi-calculus.

The spi calculus [AG97] by Abadi and Gordon is an early example of a pi-calculus extension, intended to model cryptographic protocols at a more concrete level than pure pi-calculus. To this end, term constructs such as pairs and integers are added along with primitives for cryptographic operations — the exact formulation varies depending on the application under consideration. The focus is on using testing equivalences between processes to verify authenticity and secrecy properties of protocols.

A successor to the spi calculus is the applied pi-calculus [AF01] by Abadi and Fournet. It has many features that are similar to, and indeed has inspired, features in psi-calculi. There is a notion of *active substitutions*, substitutions waiting to be applied, that are part of the process environment much like assertions in psi-calculi. Unlike assertions, active substitutions can only occur at the top level. Message terms in the applied pi-calculus are generated by names, variables and function symbols taken from an arbitrary finite signature. This signature, as well as an equational theory over it, can be seen as parameters of the applied pi-calculus. In psi-calculi, the terms can be taken from any finitely supported nominal set regardless of the concrete structure of terms, and any equational theory over them can be expressed through the entailment relation. Overall, we find that psi-calculi capture both of these phenomena using mechanisms that are simpler yet more generally applicable [BJPV11].

4.2 Process calculi and theorem proving

The first formalisation of a process calculus in a theorem proving environment appears to be due to Cleaveland and Panangaden [CP88], who describe a formalisation of CCS in the Nuprl system, though the authors focus on an earlier step, namely developing a denotational semantic model of concurrency that they then implement CCS on top of as a case study.

Another early formalisation of CCS is due to Nesi [Nes92], who formalises its syntax and axiomatisation in HOL. Operational semantics and bisimulation are not considered (and hence the axiomatisation is postulated rather than derived), as the emphasis is on using induction over the process syntax to reason about system descriptions.

The first theorem prover formalisation of the pi-calculus is due to Melham [Mel94], who uses HOL. The treatment of bound names is not dependent on a particular representation: the definitions are parametrised on a free type variable representing the type of names, and a choice function that given a set of such names always returns a fresh name is assumed. The process syntax is raw, in the sense that it does not identify alpha-equivalent terms. Instead, proving that alpha-equivalence is a bisimulation appears to be the idea, although the author does not mention actually doing so. In fact, the only bisimulation proofs that appear to have been completed at the time of publication are the abelian monoid laws of summation. These are not proofs where the treatment of bound names play any significant role, making it difficult to assess the practicality of Melham's approach.

Hirschkoff [Hir97] has formalised the pi-calculus using Coq. This is the first formalisation of the polyadic pi-calculus, and utilises de Bruijn-indices to represent names. In this representation, alpha-equivalent terms have the same syntactic representation, in contrast with nominal data where syntax is quotiented by alpha-equivalence. This results in a simple treatment of bound names, at the cost of often having to do arithmetic on the indices representing free names, both in technical lemmas and in the statements of the main definitions and theorems. Bisimulation up-to techniques are used to prove the standard congruence and structural equivalence results, as well as the so-called replication theorems [Mil91].

The Ph.D. thesis of Briais [Bri08] contains a formalisation of the spi calculus in Coq. Since the thesis considers several spi calculus dialects and Briais strives to give them a uniform treatment, the formalisation is parametrised on the type and behaviour of guards and actions; the dialects under consideration emerge through different ways of instantiating these parameters. Unlike psi-calculi, there are no general results about bisimulation gained through such instantiation. Aside from technical lemmas and definitions, the only general result appears to be that structural congruence is preserved by reduction for instantiations that "satisfy some conditions" (it is not made explicit which conditions). The other main result is derived separately for every instantiation,

and shows a correspondence between the symbolic and labelled transition relations. Names are represented by de Bruijn-indices similarly to Hirschkoff's approach, with the novelty of a general notion of so-called "de Bruijn-types" that characterises what it means for a type to use a de Bruijn representation — this is roughly analogous to the notion of nominal datatypes that we employ, and is used to alleviate the burden of having to prove the many technical lemmas related to arithmetic on de Bruijn-indices separately for every type under consideration.

Boulier and Schmitt have developed a Coq formalisation of HOCore [BS12]. Predating the first formal publication of higher-order psi-calculi in Raabjerg's licentiate thesis [Raa12] by nine months, it earns the distinction of being the first published theorem prover formalisation of a higher-order process calculus. The main results are soundness of IO-bisimilarity with regards to barbed congruence, as well as decidability of IO-bisimilarity. Bound names are represented in the locally nameless style, meaning that explicit reasoning about a well-formedness predicate is required. A follow-up paper by Maksimović and Schmitt [MS15] expands this work by considering many variants of strong bisimilarity that are all shown to coincide; an axiomatisation is also shown to be sound and complete.

Finally, the most closely related work is by Bengtson and Parrow on formalising the pi-calculus [BP07a, BP07b] and psi-calculi [BP09, BPW16] using Nominal Isabelle. The work on psi-calculi, and the foundations it is built upon, has been amply discussed in Sections 2 and 3. It largely subsumes the pi-calculus formalisation, though the latter contains some results not available in the more general setting of psi-calculi. Notably these include late equivalences and a sound and complete axiomatisation of strong late bisimilarity. Though Johansson defines a late semantics [Joh10], there is as yet no work on axiomatisations or late equivalences for psi-calculi.

4.3 Broadcast in process calculi

Calculi with global synchronisation mechanisms go back to the 1980's. CSP, due to Brookes, Hoare and Roscoe [BHR84], features an *interface parallel* operator $\parallel$ such that $P \parallel Q$ can engage in an action only if both P and Q can do so. Milner's SCCS [Mil83] is a generalisation of CCS where the set of actions is taken to be an arbitrary abelian group. The parallel composition (here called "product" and denoted $\times$) is such that if $P \xrightarrow{\alpha} P'$ and $Q \xrightarrow{\beta} Q'$, then $P \times Q \xrightarrow{\alpha\beta} P' \times Q'$, where $\alpha\beta$ is the result of applying the group operator to the actions; Holmer shows that broadcast communication emerges as a special case of this synchronisation mechanism [Hol93].

The first proposed calculus to feature broadcasts in the sense of one-to-many communication is Prasad's CBS [Pra91, Pra93, Pra95]. Though Prasad

employs process algebraic methods, the main focus is on using broadcasts as a programming paradigm for settings such as local area networks.

Ene and Muntean introduce the $b\pi$ calculus in [EM99], which is an adaptation of the pi-calculus to use broadcast communication instead of point-to-point communication. The main focus is on establishing a separation result, namely that the pi-calculus cannot uniformly encode $b\pi$ up to any "reasonable" equivalence. The intention is to capture reliable broadcasts (the authors state that "a process which [listens] on a channel a, cannot ignore any value [sent] on this channel"). The purported formalisation of this is to use an auxiliary "discard" relation that characterises when a process is allowed to discard a message, but since the discard relation is written so that every process can discard every input action, the semantics fails to capture the intended behaviour. Consider the process $\overline{a}|(a|a)$ (where we elide input and output objects). Since a may discard the action a, the parallel rule may be used to infer $a|a \xrightarrow{a} \mathbf{0}|a$, and the communication rule to infer $\overline{a}|(a|a) \xrightarrow{\overline{a}} \mathbf{0}|(\mathbf{0}|a)$, which is an example of unreliable broadcast behaviour. $\overline{a}|a$, however, exhibits reliable behaviour. In [EM01], an alternative formulation of the discard relation is presented which does not suffer from this issue. In [Ene01], a version with polyadic communication and an LTS semantics without structural congruence is presented.

In the last decade, the widespread adoption of wireless networks has resulted in a renewed interest in calculi for broadcast communication [NH06, MS06, God07, God10, LS10, SRS10, GFM08]. A thorough comparison of broadcast psi-calculi and all of these calculi can be found in Paper I; we shall briefly discuss a few calculi that are absent in that comparison.

A main feature of Merro and Sibilio's aTCWS [MS10] is the inclusion of time in the model. The time model is such that when no nodes wish to broadcast a message, a timeout event is propagated through the network and all pending message receptions time out. In psi-calculi, the same behaviour could be achieved by modelling the timeout event as a low-priority reliable broadcast [ÅPBP$^+$13].

The Applied Quality Calculus by Vigo, Nielson and Nielson [VNN13] focuses on reasoning about unsolicited messages, with explicit notions of failed and unwanted communication. A feature of this calculus that is reminiscent of psi-calculi is restrictions of the form $(\nu\widetilde{a};W)P$, where W is a *world* containing facts about the names in $\widetilde{a}$ and the relation between them. The derivation of transitions from P is parametrised on this world, in a manner that is similar to the way assertions influence execution in psi-calculi. It would be interesting to further study the relationship between worlds and assertions.

The Secure Broadcast Ambients of Gunter and Yasmeen [GY09] feature ambients [CG98] combined with broadcast communication. The aim is to achieve a calculus where broadcasts occur within dynamically reconfigurable domains. Like broadcast psi-calculi, Secure Broadcast Ambients have been

implemented using Nominal Isabelle [YG08], though the work appears to be limited to formalising the definitions rather than proving properties about them.

Finally we mention AWN [FvGH+12], the Algebra of Wireless Networks. The aim is to define a process algebra with the necessary features to model "real life" wireless mesh networks. The resulting language can hardly be described as parsimonious: it has syntax and semantics in three layers, respectively corresponding to sequential processes, parallel processes and networks; an explicit treatment of locally stored data structures within processes; and distinct primitives for broadcast, unicast, groupcast and node-internal communication. Despite the richness of the language, it is carefully designed so that it generates the same transition system — modulo strong bisimilarity — as a variant that is in de Simone format [dS85]. That strong bisimilarity is a congruence follows immediately from the fact that all operators in de Simone format preserve bisimilarity. As a major case study, AWN is used to verify the AODV routing protocol [PBRD03]. This extensive verification effort, including an Isabelle implementation and an extension with explicitly modelled time, is detailed over several follow-up papers [HvGT+12, BvGH14, BvGH16a, BvGH16b].

4.4 Higher-order process calculi

Higher-order process calculi probably originate with Thomsen's Calculus of Higher Order Communicating Systems (CHOCS for short) [Tho89]. CHOCS extends CCS by treating processes as first class objects that may be passed as values. It is demonstrated that CHOCS can simulate the untyped λ-calculus, and that recursion can be encoded using process passing. A more recent formulation called Plain CHOCS [Tho93] refines the way bound names are treated. The resulting calculus is shown to be able to simulate the pi-calculus, and vice versa. Sangiorgi obtains a fully abstract encoding of pi-calculus in his higher-order pi-calculus [San93], which unlike CHOCS is ω-order rather than second-order. A similar result is obtained for asynchronous variants of the calculi in [San01].

The idea of defining a notion of higher-order bisimulation by allowing processes occurring on labels to be mimicked by bisimilar labels appears to have been independently discovered by Astesiano, Giovini and Reggio [AGR88], and by Boudol [Bou89]. Sangiorgi obtains a less discriminatory equivalence called *context bisimulation* [San94] by introducing a universal quantification over contexts in which the processes on the label may be put. He further studies a more efficient characterisation of context bisimulation, a topic which is investigated further by Jeffrey and Rathke [JR05].

Schmitt and Stefani's Kell calculus [SS04] introduce a feature called *passivation*, which allow running processes to be consumed as messages by other

processes that occurs above it in a locality hierarchy. The intended application area is wide-area distributed systems, and the main results are that strong context bisimulation congruence is a congruence, and that it coincides with barbed congruence.

Lanese et al. study a minimal higher-order calculus called HOCore as a vehicle for exploring the expressiveness of higher-order calculi. An interesting result is that the calculus is Turing complete (and hence termination is undecidable), yet bisimulation is decidable [LPSS11]. Termination in higher-order process calculi has since been explored: Demangeon, Hirschkoff and Sangiorgi study type systems for termination [DHS10]; and Lago, Martini and Sangiorgi study a particular class of processes that are guaranteed to terminate in polynomial time [LMS10].

Sato and Sumii introduce a higher-order version of the applied pi-calculus, which like the first-order applied pi-calculus is parametrised on a term algebra and a corresponding term rewriting system for messages [SS09]. Unlike the original formulation of the applied pi-calculus, active substitutions are absent; instead, encrypted messages are sent as-is on labels. The focus is on the interplay between process-passing and cryptographic operations expressed with such terms, and the main result is the soundness of an up-to context technique that makes the authors' notion of bisimulation more tractable. A `let`-like process construct for decomposing terms is present, and since processes may occur among terms, this mechanism can be used to decompose the syntax of received processes much like the pattern matching facilities in higher-order psi-calculi. Decomposition of encrypted messages is prevented by explicitly differentiating unapplied function symbols (that may be decomposed) from applied ones (that may not). A process construct $run(M)$ for executing received processes is also present, and can proceed as any process that the term M evaluates to according to the term rewriting system. Unlike our **run** construct, the environment has no influence on this evaluation. The main purpose of the *run* construct appears to be that it forces invocation of a received process to take a τ step.

4.5 Sorts in process calculi

A sort system for the pi-calculus was first introduced by Milner [Mil91], as a means of ensuring that the arities of polyadic inputs and outputs would match. This initial effort has inspired a plethora of work on type systems for process calculi, of which we mention only a few that are geared towards a general account of type systems.

Honda [Hon96] gives such an account for so-called rooted process structures, where the focus is on using types to control how processes may be composed.

Igarashi and Kobayashi [IK04] propose an abstract notion of type systems for the pi-calculus, where types are taken to be slightly more abstract descriptions of processes. It is parametrised on a subtyping relation and a consistency condition that can be instantiated differently depending on what kind of properties (e.g. arity matching, deadlock freedom) the type system is intended to check.

Hüttel introduces a general method of equipping psi-calculi with type systems [Hüt11]. It is parametrised on a nominal datatype representing types, and a set of rules for making type judgements for terms, assertions and conditions. From certain requisites on these parameters, general subject reduction and channel safety results are derived. Hüttel makes stronger assumptions on the psi-calculi parameters than what we do in this thesis. In particular, Hüttel's terms, assertions and conditions must be algebraic datatypes such that substitution distributes over the constructors, whereas we admit arbitrary nominal sets where substitution may include computation on data.

A commonality between all of the above is that they are intended for controlling the behaviour of processes. Sorted psi-calculi is primarily intended for an earlier step, namely controlling the construction of calculi so that they more accurately reflect the intention of the calculus designer.

4.6 Pattern matching in process calculi

Haack and Jeffrey introduce the pattern-matching spi calculus [HJ06], which has a pattern matching input construct that is similar to the original psi-calculi. Specifically, input patterns have the form $\exists\widetilde{x}.M[\overline{A}];P$, where the pattern variables $\widetilde{x}$ bind into the term M, which is taken from a fixed algebraic datatype of messages. $\overline{A}$ relates to the type system and is ignored by the operational semantics, and P is the continuation. A message N matches the pattern M if $N = M[\widetilde{x} := \widetilde{L}]$ for some $\widetilde{L}$, exactly as in the original psi-calculi. The problem where pattern matching can be used to bypass encryption discussed in Section 3.1.3 is avoided by restricting attention to "implementable" patterns, where secrets may be bound only if they can be synthesised from the non-secret, non-bound components of the pattern. Our VARS function is a more general solution to the same problem.

The calculus LYSA^{NS} by Buchholtz, Nielson and Nielson [BNN04] features patterns over a spi calculus-like term language where variables occurring in a pattern bind later occurrences of the same variable. For an example, consider a message $\mathtt{T}(\mathtt{a},\mathtt{b})$ that is simply a tuple of names, where $\mathtt{T}$ is the datatype constructor. $\mathtt{T}(\mathtt{a},\mathtt{b})$ matches the pattern $\mathtt{T}(\_,\_)$, where the wildcard pattern $\_$ matches anything. The construct $p\%x$ binds x to the value matched by pattern p in the rest of the pattern. Hence $\mathtt{T}(\mathtt{a},\mathtt{b})$ does not match the pattern $\mathtt{T}(\_\%\mathtt{x},\mathtt{x})$, but $\mathtt{T}(\mathtt{a},\mathtt{a})$ does. Similarly to the pattern-matching spi calculus and motivated by the same concerns, a syntactic restriction on how patterns are constructed

is imposed: namely, no wildcards may occur in encryption key position. The semantics of pattern matching is then defined so that the cleartext within an encrypted message cannot be pattern matched against unless the key is known.

Schmitt and Stefani's Kell calculus [SS04] is parametric on a language of input patterns taken from a grammar. A relation `match` is defined, similarly to ours, where the pattern ξ matches the message M yielding the substitution Θ iff $\Theta \in \texttt{match}(\xi, M)$. Unlike our work, the substitutions thus generated are the usual syntactic replacement and cannot encode computation on data. While the construction of patterns is rather constrained when compared to psi-calculi, the only requisite on the `match` relation appears to be decidability. In particular, there does not appear to be any requisites on the support of the resulting substitution or anything akin to equivariance, so nothing prevents a substitution from inventing fresh names on the right-hand side of transition arrows. It would be interesting to give the Kell calculus a thorough nominal reading in order to fully examine the ramifications of this.

4.7 Priorities in process calculi

Priorities for process calculi were first considered in an ACP setting by Baeten, Bergstra and Klop [BBK86], where a unary priority operator Θ over processes is defined that selects which of its subprocesses may act according to an arbitrary partial order over processes. It is given meaning by formulating axioms describing its algebraic properties.

The first process calculus to give an operational semantics with prioritised actions is due to Cleaveland and Hennessy [CH88], where a set of prioritised actions is added to the usual CCS actions, the idea being that prioritised τ actions pre-emt unprioritised actions. This is captured by a two-layered semantics: the "a priori semantics" defines what transitions would be possible ignoring priorities, and the second layer encodes negative premises in terms of the a priori semantics. Deprioritisation and prioritisation operators, whose semantics are reminiscent of CCS relabelling, are also considered.

A detailed survey of the subject of priorities in process calculi has been conducted by Cleaveland, Lüttgen and Natarajan [CLN01]. Approaches are classified according to two criteria: static vs. dynamic priorities, and global vs. local pre-emption. With static priorities, the priority level of labels are fixed as the system evolves, whereas in a dynamic approach they may change. In global pre-emption, a prioritised action has pre-emptive power throughout the whole system, whereas with local pre-emption priorities only apply locally as dictated by some scoping mechanism. In this taxonomy, our priority system for psi-calculi can be classified as dynamic and global. It would be interesting to develop a system of local priorities for psi-calculi, where priorities can be bound by name restrictions.

The first mobile calculus with priorities appears to have been introduced in the field of systems biology, in the form of Versari's $\pi@$ calculus [Ver07]. The main use of priority in this context is to use high priority levels to enable sequences of actions to be completed atomically, with no interleaving of other actions from the environment; for these purposes, the global and static priority system employed suffices. This atomicity is then exploited, along with structured channels, to obtain encodings of several biologically inspired pi-calculus variants.

More recently, John et al. introduce the attributed pi-calculus with priorities [JLNU10], a pi-calculus extension where prefixes are decorated with a general notion of interaction constraints called "attributes", that are shown to subsume priorities via an encoding of $\pi@$.

The expressiveness of priorities is not as widely studied as the expressiveness of other language features, such as mobility and choice. Still, a few scattered results can be found in the literature. Most of these are negative results, to the effect that priorities cannot be encoded in some formalism without priorities.

The most comprehensive study on the relative expressiveness of priorities in process calculi is due to Versari et al. [VBG07]. The authors consider two extensions of CCS with global and local priority, respectively. First, CCS with local priorities cannot be encoded into CCS with global priorities or into the pi-calculus. These results are obtained through possibility/impossibility of solving leader election; in Paper VI we strengthen the latter result by our approach based on solvability of consensus. Second, neither flavour of prioritised CCS can be encoded into the pi-calculus or in $b\pi$. The proof strategy for these results are based on possibility/impossibility of solving the *last man standing* problem, where n processes must determine whether $n = 1$ or $n > 1$.

Bliudze and Sifakis [BS08] study the expressive power of *glue operators* in component-based systems. An n-ary glue operator **op** takes n components with behaviour, and returns a composite component with new behaviour, obtained by performing memoryless synchronisation between the behaviours of its components. In this terminology, CCS parallel composition is a binary glue operator and restriction is a unary glue operator. Choice is not a glue operator since it is not memoryless; after a transition has been taken it remembers which branch was taken. The authors show that the glue operators of the *BIP* model for behaviour composition [BBS06] — where BIP stands for *Behaviour, Interaction and Priority*) — are strictly more expressive than the glue operators in the subset of BIP without priority operators. This is not in contradiction with our results on encoding priorities in psi-calculi, since our encoding function is not a glue operator. For encodings between process calculi, the requisite that the encoding context must be a glue operator implies at least that

$$[\![\mathbf{op}(P_1,\ldots,P_n)]\!] = C[P_1,\ldots,P_n]$$

for some memoryless context C; intuitively, the glue must treat each component as a black box, and may not modify the components themselves. This is a significant strengthening of the standard compositionality requisite in process calculi [Par08, Gor10b] where the encoding distributes onto sub-processes:

$$[\![\mathbf{op}(P_1,\ldots,P_n)]\!] = C[[\![P_1]\!],\ldots,[\![P_n]\!]]$$

A similar result is due to Aceto and Ingólfsdóttir [AI08], who show that the priority operator Θ cannot be expressed by operators in the tyft/tyxt [GV89] or tree rules [FvG96] formats. In contrast to our work on encoding priorities in psi-calculi, Aceto and Ingólfsdóttir are concerned with definability of operators, rather than with the existence of translation functions.

An interesting commonality between the latter works is that the same counterexample is used for both the BIP and tyft/tyxt results, namely the inability of any positive glue operator or tyft/tyxt operator, respectively, to distinguish between the following processes:

$$a.(a+b) \qquad\qquad a.(a+b)+a.a$$

where b takes priority over a.

4.8 Bisimulation up-to techniques

The first bisimulation up-to technique is Milner's bisimulation up-to bisimilarity [Mil89], which allows $\sim\mathscr{R}\sim$ to be used in place of $\mathscr{R}$ in the lower part of the bisimulation diagram. Sangiorgi generalised this to the notion of *respectful functions* [San98], which is a family of functions that enjoy nice compositionality properties, and soundness: if f is respectful then $f(\mathscr{R})$ may be used to close lower part of the bisimulation diagram in place of $\mathscr{R}$. These developments apply to strong bisimulation on labelled transition systems, and to the pi-calculus. Hirschkoff implemented respectful functions for the pi-calculus as part of the pi-calculus formalisation [Hir97] discussed in Section 4.2.

As discussed in Section 3.2, Pous was the first to abstract away from bisimulation and derive, instead, compositional up-to techniques for arbitrary coinductive objects [Pou07b], where bisimulation emerges as a special case. Since, up-to techniques have been given much attention, with many recent papers about refinements of the general method [HNDV13, Pou16, PW16] and its application to new settings [RBR13, BPPR14, BPPR15, RBR16, BH16, SV16]. A highlight is an algorithm for checking NFA equivalence [BP13] by Bonchi and Pous, where leveraging bisimulation up-to techniques leads to drastic speedups. In the following discussion we will focus our attention on recent works about up-to techniques and name-passing calculi.

Madiot et al. [MPS14] suggest to recover up-to techniques for higher-order calculi by translating them to first-order transition systems (i.e. where the transition labels do not carry binders). This is a promising idea, but for our purposes it would only be beneficial if the difficulty of developing up-to techniques is less than the difficulty of (a) defining a first-order transition system, (b) proving full abstraction and (c) deriving bisimulation up-to context in the first-order transition system. It seems unlikely that this would hold in the general case of psi-calculi, so we prefer our more direct approach. Staying in a setting similar to the existing psi-calculi formalisation also has the advantage that we may leverage the tremendous effort invested by Bengtson in developing infrastructure for reasoning about it in Isabelle.

Chaudhuri et al. [CCM15] formalise bisimulation up-to techniques for CCS and the pi-calculus with replication in the Abella theorem prover. The pi-calculus formalisation treats bound names with Abella's built-in ∇ quantifier for *generic judgements*, i.e. $\nabla x.P$ means that P holds for all x when nothing is assumed about x. A comparison between this specification style and Nominal Isabelle can be found in [GMN09]; a comparative disadvantage of Nominal Isabelle is that a notion of substitution must be defined by the user, while it comes for free in Abella. Their main results are: the soundness of bisimulation up-to union, bisimilarity, and context for CCS, and the soundness of bisimulation up-to bisimilarity for the pi-calculus. Bisimulation up-to context for the pi-calculus is deferred to future work, with a main hurdle being that "defining the notion of a process context in the π-calculus is tricky, because contexts are allowed to capture free names" [CCM15, p. 164]. We do not know enough about Abella to understand the precise technical difficulties involved in that setting, but in Nominal Isabelle defining contexts is straightforward. The key insight is that names must be treated as non-binding when occurring in a context, but as binding once the hole has been filled; for more details on how to set this up in Nominal Isabelle see Paper V.

Differences between settings aside, our work goes beyond that of Chaudhuri et al. in several ways: first, we consider compatibility whereas they only consider soundness; hence they lack a method for combining up-to techniques. Second, we go beyond the pi-calculus and derive, once and for all, up-to techniques for all pi-calculus extensions that fall within psi-calculi.

4.9 Monotonic parallel composition

There is a long-standing tradition in process calculi, originating with the work of Bougé [Bou88] and Palamidessi [Pal97b], of deriving separation results through possibility/impossibility of solving leader election. The idea that abandoning leader election can lead to stronger separation results has been pursued by Peters and Nestmann [PN10]. They revisit Palamidessi's result that mixed choice π-calculus is more expressive than separate choice π-calculus [Pal03],

derived using leader election-based techniques. Peters and Nestmann achieve a stronger version of the same result by abstracting away from leader election and focusing instead on the problem of breaking symmetries in general [PN10]. Similarly to our work, Peters and Nestmann can then drop Palamidessi's requirement that the translation respects substitutions, i.e. that for all P, σ it holds that $[\![P]\!]\sigma = [\![P\theta]\!]$ for some substitution θ. Another similarity is that the ability to break symmetries and the ability to solve consensus can both be seen as non-confluence properties.

Another paper that is concerned with general conditions for confluence is due to Keller [Kel75]; Plotkin gives this paper credit for introducing the notion of labelled transition systems [Plo81]. Three simple criteria on the transition system are identified as sufficient to guarantee global confluence, in the sense that any two states reachable from some initial state have a common successor. The criteria are determinism (transitions with the same source and label have the same successor), commutativity (if the process can take an α transition followed by β, it can also do β then α), and persistence (if a process can take an α-labelled transition, it can still do so after taking a differently labelled transition). While these criteria rule out any applications to concurrency, several results in the field of parallel computing are shown to emerge as special cases of this general result.

Banti et al. [BPT10] study an abstract criterion for separating process calculi called *replacement freeness*, meaning that processes with no observable behaviour can be plugged into any context without impacting the observable behaviour of the context. Formally, a process calculus is strongly replacement free if for every P with no observable behaviour and every C, Q, we have that

$$\exists \alpha.\, C[P] \downarrow_\alpha \quad \Rightarrow \quad \exists \beta.\, C[Q] \downarrow_\beta$$

Weak replacement freeness is as above, but where P must additionally have empty support. This gives rise to a three-tiered expressiveness hierarchy. Languages that are not replacement free, such as CCS and pi-calculus without matching, cannot be encoded into weakly replacement free languages, such as pi-calculus with name matching, polyadic synchronisation or pattern matching. The latter languages, in turn, cannot be encoded into the various languages with priorities that exemplify replacement free languages. The criteria used are quite weak: compositionality, preservation and reflection of the existence of observables, and (for some results) that the encoding of processes with non-overlapping support have non-overlapping support. The first two are orthogonal to our requisites. As for the latter, we impose no constraints at all on how the encoding treats names. Showing that a calculus is not replacement free is about as easy as exhibiting a consensus process in our setting, since we only need to exhibit C, P, Q violating the above definition. Establishing replacement freeness, however, can be quite involved because of the universal quantification over contexts. The preferred proof technique of the authors is

through proving that ω-*simulation*, i.e. the intersection of all finite approximations of the standard simulation preorder, satisfies congruence properties wrt. all operators of the language. By contrast, establishing that parallel composition is monotonic requires just a quick glance at the operational semantics.

Traditionally, homomorphism of the parallel operator has often been used as a criterion to state that encodings should preserve the degree of distribution. Our criterion that $[\![P \mid Q]\!] = [\![P]\!] \otimes [\![Q]\!]$ for some monotonic composition $\otimes$ is weaker, and can be seen as a less syntactic way to state that the degree of distribution should be preserved. Another way to obtain a weaker criterion that still guarantees preservation of the degree of distribution has been proposed by Peters et. al [PNG13]. Their approach is more focused on syntactic distributability, and more specifically tailored towards process calculi. It assumes that the process languages have a syntax where subprocesses can be composed with operators in an algebraic manner, and have *capabilities*, i.e. operators that are consumed when a transition is taken (such as prefixes in process calculi). Distributability is then a syntactic property of processes, where (roughly stated) a process is distributable into its set of top-level capabilities. An encoding preserves distributability if whenever a source process is distributable into some components, the target process is distributable into the same number of components, where each component of the target process is behaviourally equivalent to a corresponding component of the source process.

Hence our approach offers more flexibility in the kind of languages it can be applied to, and in designing encodings that are semantically but not syntactically distributable. For example, a *normal form* of a π-calculus process P is an equivalent process on the form $\Sigma_i\ \alpha_i.P_i$, where each P_i is also on normal form. An encoding that translates every agent of the finite fragment of the π-calculus to its normal form would satisfy our criterion that parallel must be translated by monotonic composition, but does not preserve the degree of distribution in the sense of Peters et. al. On the other hand, they do not insist on associativity and commutativity of the contexts that parallel translates into, which offers some flexibility in designing encodings that is not available with our criteria.

5. Conclusion

In this chapter, we conclude by discussing our contribution, related work, future work and impact.

5.1 Discussion

We have seen how psi-calculi can be extended with several new language features in order to broaden its applicability, without sacrificing generality or the associated machine-checked meta-theory. We have then studied the expressiveness of psi-calculi, and addressed the proof engineering challenge of keeping the formal proof archive maintainable. These endeavours are closely connected, in a way that may not be immediately apparent; we shall elaborate this point in the following extended discussion on the state of the proof archive and its future.

The original psi-calculi formalisation by Bengtson consists of about 30,000 lines of code. Add to that figure the recent developments described in this thesis and elsewhere [ÅPBP+13], and the result is about ten times Bengtson's figure. The main reason for this size explosion is excessive forking: each extension described here constitutes a fork of Bengtson's proof scripts.

One fork per extension is not enough, however: we sometimes need to integrate the extensions. For an example, the LUNAR model of Paper I uses both broadcast communication and process definitions. Hence we implement it in higher-order broadcast psi-calculi — and higher-order broadcast psi-calculi we implement by forking our proofs yet again. In the short term, the cost of thus combining two pre-existing extensions is low. In this case it took no more than an afternoon of work; a really nice perk of having invested in formal proofs is that it is cheap to obtain near-absolute certainty that small changes cause no unforeseen problems.

In the long term, this is hardly a sustainable practice; implementing every possible combination of the extensions in this manner would result in a combinatorial explosion. The set of possible extensions of psi-calculi that can be obtained by combination of the extensions described in this thesis can be depicted as a lattice (Figure 5.1). The bottom element is the original psi-calculi. Traversing the lattice upwards corresponds to piling on more extensions, until we reach the top element: sorted higher-order broadcast prioritised psi-calculi, abbreviated by the tongue twister SHoBPΨ. Unbroken lines denote extensions that we have implemented, including the verified meta-theory. Dotted lines are

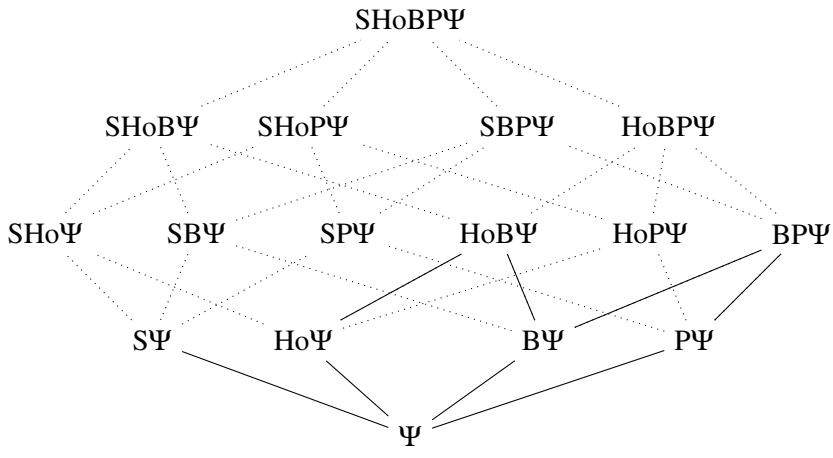

Figure 5.1. The lattice of possible combinations of our extensions to psi-calculi.

paths untravelled: extensions that should be straightforward albeit tedious to implement, should the need arise.

Ideally, we want current and future developments of psi-calculi in a form that is well integrated and easy to maintain. The encoding that we present in Paper IV, from psi-calculi with priorities into the original psi-calculi, suggests a way forward: rather than extending the semantics with new features, we may encode them using the existing features. Because the encoding satisfies such strong quality criteria, there are no strong arguments for maintaining the extension with priorities. Any psi-calculus with priorities could instead be obtained as an original psi-calculus through the encoding. Hence, the impact of the encoding on maintainability is to render half the lattice of Figure 5.1 superfluous.

One question remains open: how do we handle extensions where no good encoding is known or possible? Clearly the path forward is not to to populate every element of the lattice with an Isabelle implementation. A more attractive strategy would be to develop and maintain only the (current) top element. While this idea seems worth exploring in more detail, a number of arguments against it are immediately apparent. First, extensions (and combinations thereof) are not necessarily conservative, so we may lose expressiveness in doing so. Second, it may be difficult to do so in a way that maintains backwards-compatibility with previous developments. Third, the top element would be cumbersome to reason about and instantiate, with its myriad parameters, derivation rules, and multi-layered semantics. The resulting tedium may seriously bog down further development.

The separation between monotonic and non-monotonic parallel composition is of independent interest outside the context of psi-calculi, as we demonstrate by the many applications in Paper VI. In the context of psi-calculi it

helps shed light on *why* psi-calculi can encode priorities. It turns out that non-monotonic assertion logics are crucial, and that the restriction of psi-calculi to monotonic assertion logics considered in e.g. [JVP10] is significantly less expressive.

5.2 Future work

In this section we discuss some avenues for future work.

5.2.1 Weak equivalences

Weak bisimulation is important for applications. Since it abstracts from internal behaviour, it allows descriptions of a system at different levels of abstraction to be formally related. For an example, a way to show that a system correctly implements a specification is to show that the two are weakly bisimilar.

Currently, we have shown that the meta-theory pertaining to weak bisimulation for the original psi-calculi carries over to the higher-order, pattern matching and sorted extensions. We would of course like to do the same for the extensions with broadcast and priorities. Bisimulation up-to techniques for our weak equivalences would also be an interesting direction to explore.

5.2.2 Full abstraction of the priority encoding

As mentioned in Section 3.3, our encoding of priorities in psi-calculi satisfies virtually any reasonable quality criterion, save for full abstraction. Intuitively, the main reason for this is that with action priorities, a prefix is a single component that serves two purposes: to enable interaction with dual prefixes, and to block interaction between other, lower-priority prefixes. In our target language, enabling and blocking are decoupled into two components, so that prefixes enable interaction, and assertions block lower-priority interactions. This leads to the target language having more discriminatory observers that can interact with only the blocking component in unexpected ways. For an example, suppose $\overline{\alpha}$ is an output prefix, α is an input prefix that can communicate with $\overline{\alpha}$, and β is a lower priority prefix. We have that $\overline{\alpha}.\overline{\alpha} \sim \overline{\alpha} \mid \overline{\alpha}$ yet $[\![\overline{\alpha}.\overline{\alpha}]\!] \not\sim [\![\overline{\alpha} \mid \overline{\alpha}]\!]$. The reason is that the observer

$$P \triangleq (\!|\alpha|\!) \mid (\!|-\overline{\alpha}|\!) \mid \beta$$

that disables the blocking component of one $\overline{\alpha}$-prefix can tell the difference. From $P \mid [\![\overline{\alpha}.\overline{\alpha}]\!]$, there is an initial β-transition because the blocking power of the one top-level $\overline{\alpha}$ is cancelled. From $P \mid [\![\overline{\alpha} \mid \overline{\alpha}]\!]$ there is no initial β-transition because the blocking assertion of one of the two top-level $\overline{\alpha}$ prefixes will remain in force.

A simple idea that may well suffice to make the encoding fully abstract is to use restrictions to localise the power to disable. The idea is to tag blocking assertions with a fresh restricted name. Then, assertion composition can be defined so that blocking assertions may only be disabled by assertions carrying the same tag. This would prevent a malicious environment from interfering with the intended behaviour of the encoding in the above manner, without otherwise introducing significant complications.

5.2.3 Protocol verification with psi-calculi and Isabelle

In Paper I, we apply broadcast psi-calculi to verify a basic reachability property of the LUNAR protocol [TGRW04] for ad-hoc routing in wireless networks. Our proof is currently a pen-and-paper proof which involves tedious and error-prone manual following of transitions.

We do, however, have an Isabelle proof showing that the psi-calculus which we use to model LUNAR is indeed a psi-calculus (i.e. that the parameters satisfy the requisites). Thanks to the locale mechanism of Isabelle [Bal03], this means we also have a fully developed infrastructure for reasoning about the semantics of this calculus. We would like to use this infrastructure to formalise our reachability proof, for two reasons. First, it is a way to obtain added confidence in our results. Second, it is a way to evaluate how useful psi-calculi are for protocol verification in Isabelle, and seeing how it compares to e.g. Paulson's inductive approach [Pau98]. Our proposed approach would be complementary to protocol verification in the psi-calculi workbench [Gut11, BGRV13], offering less automation but greater trustworthiness, though they could also be combined by e.g. using the workbench as an oracle for inferring transitions in Isabelle.

5.2.4 Monotonicity and distributability

In Section 4.9 we discuss the differences between the distributability-preservation criterion of Peters et al. [PNG13], and our criterion that parallel is translated by a monotonic composition. An interesting direction for future work would be to investigate if the separation results derived under the latter criterion in Paper VI can also be derived under the former. This would strengthen the separation results, and offer more insight into the relationship between the criteria.

As a starting point for this investigation, we offer a simple proof that at least one of our separation results carries over. First we recapitulate the relevant definitions from [PNG13].

Definition 18 (Distributability)**.** *P is* distributable into $P_1, \ldots, P_n$ *if there exists* $P' \equiv P$ *such that:*

1. *For all* $1 \leq i \leq n$, P_i *is an unguarded subterm of* P *that contains at least one capability or constant other than* 0.
2. *In* $P_1, \ldots, P_n$ *there are no two occurrences of the same capability.*
3. *Each guarded subterm and each constant (different from* 0*) of* P' *is a subterm of at least one of the terms* $P_1, \ldots, P_n$.

Definition 19 (Preservation of distributability)**.** *An encoding* $[\![\cdot]\!]$ preserves distributability *if whenever* S *distributes into* $S_1, \ldots, S_n$*, there exists* $T_1, \ldots, T_n$ *that* $[\![S]\!]$ *distribute into, such that for all* $1 \leq i \leq n$, $[\![S_i]\!] \approx T_i$.

Theorem 12. *There exists no success-respecting, operationally corresponding and distributability-preserving encoding of CCS with priorities [CH88] into the* π*-calculus modulo success-respecting weak bisimulation.*

Proof. (Sketch) By contradiction. Assume $[\![\cdot]\!]$ is such an encoding. Let $P = \mathbf{fix}\, X.(\underline{\tau}.X)$ and $Q = \tau.\checkmark$. The process $P \mid Q$ in CCS with priorities cannot reach success since $\underline{\tau}$ always takes priority over τ. Since $P \mid Q$ is distributable into P and Q, $[\![P \mid Q]\!]$ must be distributable into R, S such that $R \approx [\![P]\!]$ and $S \approx [\![Q]\!]$; since the only non-guards in the π-calculus are parallel and restriction it must be the case that $[\![P \mid Q]\!] \equiv (\nu \widetilde{x})(R \mid S \mid C)$ for some $\widetilde{x}, C$. Hence $[\![P \mid Q]\!] \approx (\nu \widetilde{x})(R \mid [\![Q]\!] \mid C)$. By operational correspondence and success sensitivity there must be Q' such that $[\![Q]\!] \Longrightarrow \checkmark \mid Q'$. Hence $(\nu \widetilde{x})(R \mid [\![Q]\!] \mid C) \Longrightarrow (\nu \widetilde{x})(R \mid \checkmark \mid Q' \mid C)$; since $\approx$ is success-respecting $[\![P \mid Q]\!]$ can reach success, contradicting success sensitivity of $[\![\cdot]\!]$. □

It is unclear whether the setting of Peters et al. also admits a widely applicable separation result for monotonic parallel composition to be formulated in the abstract. The proof above relies on many details that are specific to the particular languages under consideration, and a smooth generalisation to other languages is not immediately apparent. A stronger proof that relies only on simple and abstract properties of languages, such as monotonicity of parallel composition, would be desirable.

5.2.5 Culling psi-calculi

Does psi-calculi have any superfluous operators, that we can remove without sacrificing expressiveness? More precisely, we ask whether there is an operator **op** which is redundant in the following sense: for every psi-calculus $\mathscr{P}$, there is a uniform way to construct a psi-calculus $\mathscr{P}_{\mathbf{op}}$ and a high-quality encoding $[\![\cdot]\!]$ from $\mathscr{P}$ to $\mathscr{P}_{\mathbf{op}}$, such that for all $P \in \mathscr{P}$, $[\![P]\!]$ contains no occurrence of **op**.

Any answer to this question would advance our understanding of the expressive power of psi-calculi. A negative answer would imbue psi-calculi as

they have so far been presented with a certain air of canonicity. A positive answer would have practical benefits: fewer operators would make psi-calculi less cumbersome to reason about.

Which operators are likely candidates for culling? It cannot possibly be replication, since without it we cannot express unbounded behaviour. Nor can it be the assertions, since without them parallel composition becomes monotonic.

The likeliest candidate seems to be **case**, which functions both as choice operator and as conditional operator. Conditionals could be emulated as preconditions for channel equivalence statements to hold. As for choice, there is plenty of previous work on choice encodings for the pi-calculus [NP00, Nes00, HP02, Pal03]. The encodings considered typically introduce elaborate protocols, and satisfy quality criteria that are too weak for our purposes. For example, all choice encodings for the pi-calculus satisfy only weak operational correspondence. It seems reasonable to expect that with assertions at our disposal we could achieve strong operational correspondence. An idea that seems promising is to mimic the conflict operator of Busi and Gorrieri [BG94] using assertions.

Culling **case** would render conditions a superfluous parameter, and enable some further simplifications to definitions and proofs.

5.2.6 An algebra of psi-calculi

Suppose we have two psi-calculi. A natural question to ask is then: can we combine them in a useful way to obtain a third, more expressive psi-calculus?

One way to approach this question is to study functions that operate on psi-calculi (henceforth called functors). In fact, this thesis already contains three examples of unary functors: the functor U used in the meta-theoretical proofs in Paper III, the functor $\mathscr{H}$ that maps a psi-calculus to its canonical higher-order extension from Paper II, and the (anonymous) functor used in Paper IV to encode psi-calculi with priorities. Binary functors could also be used to compose psi-calculi in different ways. Developing a library of such functors would be a worthwhile endeavour — it would essentially yield a toolbox of off-the-shelf components that can be used to construct complex psi-calculi from simpler building blocks, without having to redo the proofs that the parameters satisfy the requisites every time. It would also be interesting to study algebraic properties of such functors under some reasonable notion of equivalence between psi-calculi.

5.3 Impact

The original psi-calculi makes it easy to obtain a custom process calculus with a machine-checked meta-theory, at least for applications where the pi-calculus

extended with structured terms and arbitrary logical tests suffices. This thesis brings the gospel of reusable machine-checked meta-theory to new application areas where the language features we introduce are important, such as wireless networks, systems biology, cryptography and computer architecture. If a formal methods practitioner wishes to conduct a process algebraic study in one of these application areas, supporting the analysis with the rigour of an interactive theorem prover is now easier than ever before: man-months of arduous labour to obtain standard results can be saved by interpreting our locales instead of building a formalisation from scratch.

Existing process calculi that we show to be psi-calculi, such as $b\pi$, $\pi@$ and the pi-calculus with polyadic synchronisation, now enjoy significantly higher confidence in the correctness of their meta-theoretical results thanks to our machine-checked proofs. This translates into added confidence that formal verification of models written in these languages is based on sound principles.

Bisimulation-based verification in psi-calculi can be made significantly easier and more trustworthy by using our bisimulation up-to techniques. As our case studies in Paper V demonstrate, they yield drastically shorter proofs; and shorter proofs, in turn, means less time must be spent on their development and maintenance.

The expressiveness of disparate language features such as priority, time, broadcast communication, fusion, assertions and concurrent constraints is better and more uniformly understood. Deriving formal separation results in process calculi is typically arduous work; using our method, the labour involved is almost nil. Such results are helpful for language designers, who gain insight into what the impact of including or excluding features is. They can also save language users the futile effort of trying to implement something unimplementable.

References

[Aba99] Martín Abadi. Protection in programming-language translations. In Jan Vitek and Christian Damsgaard Jensen, editors, *Secure Internet Programming, Security Issues for Mobile and Distributed Objects*, volume 1603 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 1999.

[AdF15] Luca Aceto and David de Frutos-Escrig, editors. *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015*, volume 42 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[AF01] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proceedings of POPL '01*, pages 104–115. ACM, 2001.

[AG97] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.

[AGR88] Egidio Astesiano, Alessandro Giovini, and Gianna Reggio. Generalized bisimulation in relational specifications. In Robert Cori and Martin Wirsing, editors, *STACS*, volume 294 of *Lecture Notes in Computer Science*, pages 207–226. Springer, 1988.

[AI08] Luca Aceto and Anna Ingólfsdóttir. On the expressibility of priority. *Inf. Process. Lett.*, 109(1):83–85, 2008.

[ÅP10] Johannes Åman Pohjola. *Verifying Psi-calculi*. M. Sc. thesis, Department of Information Technology, Uppsala University, 2010.

[ÅP13] Johannes Åman Pohjola. *Bells and Whistles: Advanced Language Features in Psi-Calculi*. Licentiate thesis, Department of Information Technology, Uppsala University, October 2013.

[ÅPBP+13] Johannes Åman Pohjola, Johannes Borgström, Joachim Parrow, Palle Raabjerg, and Ioana Rodhe. Negative premises in applied process calculi. Technical Report 2013-014, Department of Information Technology, Uppsala University, June 2013.

[Bae05] Jos C. M. Baeten. A brief history of process algebra. *Theor. Comput. Sci.*, 335(2-3):131–146, May 2005.

[Bal03] Clemens Ballarin. Locales and locale expressions in Isabelle/Isar. In Stefano Berardi, Mario Coppo, and Ferruccio Damiani, editors, *Types for Proofs and Programs, International Workshop, TYPES*

2003, Torino, Italy, April 30 – May 4, 2003, Revised Selected Papers, volume 3085 of *Lecture Notes in Computer Science*, pages 34–50. Springer-Verlag, 2003.

[Bal14] Clemens Ballarin. Locales: A module system for mathematical theories. *J. Autom. Reasoning*, 52(2):123–153, 2014.

[Bar81] Hendrik P. Barendregt. *The lambda calculus : its syntax and semantics*. North-Holland Pub. Co, 1981.

[BB92] Gérard Berry and Gérard Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.

[BBK86] Jos C. M. Baeten, Jan A. Bergstra, and Jan Willem Klop. Syntax and defining equations for an interrupt mechanism in process algebra. *Fundamenta Informaticae*, IX(2):127–168, 1986.

[BBN11] Jasmin Christian Blanchette, Lukas Bulwahn, and Tobias Nipkow. Automatic proof and disproof in Isabelle/HOL. In *Frontiers of Combining Systems*, pages 12–27. Springer, 2011.

[BBS06] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling heterogeneous real-time components in BIP. In *Fourth IEEE International Conference on Software Engineering and Formal Methods (SEFM 2006), 11-15 September 2006, Pune, India*, pages 3–12. IEEE Computer Society, 2006.

[BC04] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer-Verlag, 2004.

[Ben10] Jesper Bengtson. *Formalising process calculi*. PhD thesis, Uppsala University, June 2010.

[Ben12] Jesper Bengtson. Psi-calculi in Isabelle. *Archive of Formal Proofs*, 2012, 2012.

[Bes93] Eike Best, editor. *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*. Springer, 1993.

[BG94] Nadia Busi and Roberto Gorrieri. Distributed conflicts in communicating systems. In Paolo Ciancarini, Oscar Nierstrasz, and Akinori Yonezawa, editors, *Object-Based Models and Languages for Concurrent Systems, ECOOP'94 Workshop on Models and Languages for Coordination of Parallelism and Distribution, Bologna, Italy, July 5, 1994, Selected Papers*, volume 924 of *Lecture Notes in Computer Science*, pages 49–65. Springer, 1994.

[BG96] Roland N. Bol and Jan Friso Groote. The meaning of negative premises in transition system specifications. *J. ACM*, 43(5):863–914, 1996.

[BGLG93] Patrice Brémond-Grégoire, Insup Lee, and Richard Gerber. ACSR: An algebra of communicating shared resources with

dense time and priorities. In Eike Best, editor, *CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, pages 417–431. Springer Berlin Heidelberg, 1993.

[BGRV13] Johannes Borgström, Ramunas Gutkovas, Ioana Rodhe, and Björn Victor. A parametric tool for applied process calculi. In Josep Carmona, Mihai T. Lazarescu, and Marta Pietkiewicz-Koutny, editors, *13th International Conference on Application of Concurrency to System Design, ACSD 2013, Barcelona, Spain, 8-10 July, 2013*, pages 180–185. IEEE Computer Society, 2013.

[BH16] Henning Basold and Helle Hvid Hansen. Well-definedness and observational equivalence for inductive–coinductive programs. *Journal of Logic and Computation*, abs/1605.04136, 2016. Advance Access published April 12, 2016.

[BHL+14] Jasmin Christian Blanchette, Johannes Hölzl, Andreas Lochbihler, Lorenz Panny, Andrei Popescu, and Dmitriy Traytel. Truly modular (co)datatypes for Isabelle/HOL. In Gerwin Klein and Ruben Gamboa, editors, *Interactive Theorem Proving - 5th International Conference, ITP 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8558 of *Lecture Notes in Computer Science*, pages 93–110. Springer, 2014.

[BHR84] Stephen D. Brookes, Charles Antony Richard Hoare, and A. William Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984.

[BJPV09] Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Psi-calculi: Mobile processes, nominal data, and logic. In *Proceedings of LICS 2009*, pages 39–48. IEEE, 2009.

[BJPV11] Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Psi-calculi: A framework for mobile processes with nominal data and logic. *Logical Methods in Computer Science*, 7(1), 2011.

[BK84] Jan A. Bergstra and Jan Willem Klop. The algebra of recursively defined processes and the algebra of regular processes. In Jan Paredaens, editor, *Automata, Languages and Programming*, volume 172 of *Lecture Notes in Computer Science*, pages 82–94. Springer Berlin Heidelberg, 1984.

[BNN04] Mikael Buchholtz, Hanne Riis Nielson, and Flemming Nielson. A calculus for control flow analysis of security protocols. *Int. J. Inf. Sec.*, 2(3-4):145–167, 2004.

[Bou88] Luc Bougé. On the existence of symmetric algorithms to find leaders in networks of communicating sequential processes. *Acta Inf.*, 25(2):179–201, 1988.

[Bou89] Gérard Boudol. Towards a lambda-calculus for concurrent and communicating systems. In Josep Díaz and Fernando Orejas, ed-

itors, *TAPSOFT, Vol.1*, volume 351 of *Lecture Notes in Computer Science*, pages 149–161. Springer, 1989.

[BP07a] Jesper Bengtson and Joachim Parrow. A completeness proof for bisimulation in the pi-calculus using Isabelle. *Electr. Notes Theor. Comput. Sci.*, 192(1):61–75, 2007.

[BP07b] Jesper Bengtson and Joachim Parrow. Formalising the pi-calculus using nominal logic. In *Proceedings of FoSSaCS 2007*, volume 4423 of *Lecture Notes in Computer Science*, pages 63–77. Springer-Verlag, 2007.

[BP09] Jesper Bengtson and Joachim Parrow. Psi-calculi in Isabelle. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Proc. of TPHOLs 2009*, volume 5674 of *Lecture Notes in Computer Science*, pages 99–114. Springer-Verlag, August 2009.

[BP13] Filippo Bonchi and Damien Pous. Checking NFA equivalence with bisimulations up to congruence. In Giacobazzi and Cousot [GC13], pages 457–468.

[BPPR14] Filippo Bonchi, Daniela Petrisan, Damien Pous, and Jurriaan Rot. Coinduction up-to in a fibrational setting. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 20:1–20:9. ACM, 2014.

[BPPR15] Filippo Bonchi, Daniela Petrisan, Damien Pous, and Jurriaan Rot. Lax bialgebras and up-to techniques for weak bisimulations. In Aceto and de Frutos-Escrig [AdF15], pages 240–253.

[BPT10] Federico Banti, Rosario Pugliese, and Francesco Tiezzi. A criterion for separating process calculi. In Fröschle and Valencia [FV10], pages 16–30.

[BPW16] Jesper Bengtson, Joachim Parrow, and Tjark Weber. Psi-calculi in isabelle. *J. Autom. Reasoning*, 56(1):1–47, 2016.

[Bri08] Sébastien Briais. *Theory and tool support for the formal verification of cryptographic protocols*. PhD thesis, EPFL, Lausanne, 2008.

[BS08] Simon Bliudze and Joseph Sifakis. A notion of glue expressiveness for component-based systems. In Franck van Breugel and Marsha Chechik, editors, *CONCUR 2008 - Concurrency Theory, 19th International Conference, CONCUR 2008, Toronto, Canada, August 19-22, 2008. Proceedings*, volume 5201 of *Lecture Notes in Computer Science*, pages 508–522. Springer, 2008.

[BS12] Simon Boulier and Alan Schmitt. Formalisation de HOCore en Coq. In *JFLA - Journées Francophones des Langages Applicatifs - 2012*, Carnac, France, February 2012.

[BvGH14] Timothy Bourke, Rob J. van Glabbeek, and Peter Höfner. A mechanized proof of loop freedom of the (untimed) AODV routing protocol. In Franck Cassez and Jean-François Raskin, editors, *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings*, volume 8837 of *Lecture Notes in Computer Science*, pages 47–63. Springer, 2014.

[BvGH16a] Timothy Bourke, Rob J. van Glabbeek, and Peter Höfner. Mechanizing a process algebra for network protocols. *J. Autom. Reasoning*, 56(3):309–341, 2016.

[BvGH16b] Emile Bres, Rob J. van Glabbeek, and Peter Höfner. A timed process algebra for wireless networks with an application in routing - (extended abstract). In Peter Thiemann, editor, *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9632 of *Lecture Notes in Computer Science*, pages 95–122. Springer, 2016.

[CCM15] Kaustuv Chaudhuri, Matteo Cimini, and Dale Miller. A lightweight formalization of the metatheory of bisimulation-up-to. In Xavier Leroy and Alwen Tiu, editors, *Proceedings of the 2015 Conference on Certified Programs and Proofs, CPP 2015, Mumbai, India, January 15-17, 2015*, pages 157–166. ACM, 2015.

[CG98] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In Maurice Nivat, editor, *FoSSaCS*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer, 1998.

[CH88] Rance Cleaveland and Matthew Hennessy. Priorities in process algebras. In *LICS*, pages 193–202. IEEE Computer Society, 1988.

[Cha12] Arthur Charguéraud. The locally nameless representation. *Journal of Automated Reasoning*, 49(3):363–408, 2012.

[CLN01] Rance Cleaveland, Gerald Lüttgen, and V. Natarajan. Priority in process algebra. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra*, pages 711–765. Elsevier Science Publishers, February 2001.

[CM03] Marco Carbone and Sergio Maffeis. On the expressive power of polyadic synchronisation in π-calculus. *Nordic Journal of Computing*, 10(2):70–98, 2003.

[CMRG12] Matteo Cimini, Mohammad Reza Mousavi, Michel A. Reniers, and Murdoch J. Gabbay. Nominal SOS. *Electr. Notes Theor. Comput. Sci.*, 286:103–116, 2012.

[CP88] Rance Cleaveland and Prakash Panangaden. Type theory and

concurrency. *International Journal of Parallel Programming*, 17(2):153–206, 1988.

[dB72] Nicolaas G. de Bruijn. Lambda calculus notation with nameless dummies. A tool for automatic formula manipulation with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 34:381–392, 1972.

[DHS10] Romain Demangeon, Daniel Hirschkoff, and Davide Sangiorgi. Termination in higher-order concurrent calculi. *J. Log. Algebr. Program.*, 79(7):550–577, 2010.

[dS85] Robert de Simone. Higher-level synchronising devices in Meije-SCCS. *Theoretical Computer Science*, 37:245 – 267, 1985.

[DY83] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[EM99] Cristian Ene and Traian Muntean. Expressiveness of point-to-point versus broadcast communications. In Gabriel Ciobanu and Gheorghe Paun, editors, *Proceedings of FCT'99*, volume 1684 of *Lecture Notes in Computer Science*, pages 258–268. Springer-Verlag, 1999.

[EM01] Cristian Ene and Traian Muntean. A broadcast-based calculus for communicating systems. In *Proceedings of the 15th International Parallel & Distributed Processing Symposium*, IPDPS '01, pages 149–, Washington, DC, USA, 2001. IEEE Computer Society.

[Ene01] Cristian Ene. *Un Modèle formel pour les systèmes mobiles a diffusion*. Phd thesis, Université de la Méditerranée – Marseille, 2001.

[FV10] Sibylle B. Fröschle and Frank D. Valencia, editors. *Proceedings 17th International Workshop on Expressiveness in Concurrency*, volume 41 of *EPTCS*, 2010.

[FvG96] Wan Fokkink and Rob J. van Glabbeek. Ntyft/Ntyxt rules reduce to ntree rules. *Inf. Comput.*, 126(1):1–10, 1996.

[FvGH+12] Ansgar Fehnker, Rob J. van Glabbeek, Peter Höfner, Annabelle McIver, Marius Portmann, and Wee Lum Tan. A process algebra for wireless mesh networks. In Helmut Seidl, editor, *Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7211 of *Lecture Notes in Computer Science*, pages 295–315. Springer, 2012.

[Gab11] Murdoch J. Gabbay. Foundations of nominal techniques: logic and semantics of variables in abstract syntax. *Bulletin of Symbolic Logic*, 17(2):161–229, 2011.

[GC13] Roberto Giacobazzi and Radhia Cousot, editors. *The 40th Annual*

ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013. ACM, 2013.

[Gel85] David Gelernter. Generative communication in Linda. *ACM TOPLAS*, 7(1):80–112, January 1985.

[GFM08] Fatemeh Ghassemi, Wan Fokkink, and Ali Movaghar. Restricted broadcast process theory. In Antonio Cerone and Stefan Gruner, editors, *Proceedings of SEFM 2008*, pages 345–354. IEEE Computer Society, 2008.

[GM93] Michael J. C. Gordon and Tom F. Melham, editors. *Introduction to HOL: a theorem proving environment for higher order logic*. Cambridge University Press, New York, NY, USA, 1993.

[GMN09] Andrew Gacek, Dale Miller, and Gopalan Nadathur. Reasoning in Abella about structural operational semantics specifications. *Electr. Notes Theor. Comput. Sci.*, 228:85–100, 2009.

[GN14] Daniele Gorla and Uwe Nestmann. Full abstraction for expressiveness: history, myths and facts. *Mathematical Structures in Computer Science*, FirstView:1–16, 11 2014.

[God07] Jens Christian Godskesen. A calculus for mobile ad hoc networks. In *Proceedings of COORDINATION 2007*, volume 4467 of *Lecture Notes in Computer Science*, pages 132–150. Springer-Verlag, 2007.

[God10] Jens Christian Godskesen. Observables for mobile and wireless broadcasting systems. In *Proceedings of COORDINATION 2010*, volume 6116 of *Lecture Notes in Computer Science*, pages 1–15. Springer-Verlag, 2010.

[Gor10a] Daniele Gorla. A taxonomy of process calculi for distribution and mobility. *Distributed Computing*, 23(4):273–299, 2010.

[Gor10b] Daniele Gorla. Towards a unified approach to encodability and separation results for process calculi. *Inf. Comput.*, 208(9):1031–1053, 2010.

[GP01] Murdoch J. Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2001.

[Gro93] Jan Friso Groote. Transition system specifications with negative premises. *Theor. Comput. Sci.*, 118(2):263–299, September 1993.

[Gut11] Ramūnas Gutkovas. *Exercising Psi-calculi: A Psi-calculi workbench*. M.Sc. thesis, Department of Information Technology, Uppsala University, June 2011.

[GV89] Jan Friso Groote and Frits Vaandrager. Structured operational semantics and bisimulation as a congruence. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simonetta Ronchi Rocca, editors, *Automata, Languages and Programming*, volume 372

of *Lecture Notes in Computer Science*, pages 423–438. Springer Berlin Heidelberg, 1989.

[GY09] Elsa Gunter and Ayesha Yasmeen. Secure broadcast ambients. In Pierpaolo Degano, Joshua Guttman, and Fabio Martinelli, editors, *Formal Aspects in Security and Trust*, volume 5491 of *Lecture Notes in Computer Science*, pages 257–271. Springer Berlin / Heidelberg, 2009.

[HBS73] Carl Hewitt, Peter Bishop, and Richard Steiger. A universal modular actor formalism for artificial intelligence. In *Proceedings of the 3rd international joint conference on Artificial intelligence*, pages 235–245. Morgan Kaufmann Publishers Inc., 1973.

[Hir97] Daniel Hirschkoff. A full formalisation of pi-calculus theory in the calculus of constructions. In *TPHOLs '97: Proceedings of the 10th International Conference on Theorem Proving in Higher Order Logics*, pages 153–169, London, UK, 1997. Springer-Verlag.

[HJ06] Christian Haack and Alan Jeffrey. Pattern-matching spi-calculus. *Information and Computation*, 204:1195–1263, 2006.

[HNDV13] Chung-Kil Hur, Georg Neis, Derek Dreyer, and Viktor Vafeiadis. The power of parameterization in coinductive proof. In Giacobazzi and Cousot [GC13], pages 193–206.

[Hol93] Uno Holmer. Interpreting broadcast communication in SCCS. In Best [Bes93], pages 188–201.

[Hon96] Kohei Honda. Composing processes. In Hans-Juergen Boehm and Guy L. Steele Jr., editors, *POPL*, pages 344–357. ACM Press, 1996.

[HP02] Oltea Mihaela Herescu and Catuscia Palamidessi. A randomized distributed encoding of the pi-calculus with mixed choice. In Ricardo A. Baeza-Yates, Ugo Montanari, and Nicola Santoro, editors, *IFIP TCS*, volume 223 of *IFIP Conference Proceedings*, pages 537–549. Kluwer, 2002.

[HU10] Brian Huffman and Christian Urban. A new foundation for Nominal Isabelle. In Matt Kaufmann and Lawrence C. Paulson, editors, *ITP*, volume 6172 of *Lecture Notes in Computer Science*, pages 35–50. Springer, 2010.

[Hur03] Joe Hurd. First-order proof tactics in higher-order logic theorem provers. In Myla Archer, Ben Di Vito, and César Muñoz, editors, *Design and Application of Strategies/Tactics in Higher Order Logics (STRATA 2003)*, number NASA/CP-2003-212448 in NASA Technical Reports, pages 56–68, September 2003.

[Hüt11] Hans Hüttel. Typed ψ-calculi. In Joost-Pieter Katoen and Barbara König, editors, *CONCUR*, volume 6901 of *Lecture Notes in Computer Science*, pages 265–279. Springer, 2011.

[Hüt13] Hans Hüttel. Types for resources in ψ-calculi. In Martín Abadi and Alberto Lluch-Lafuente, editors, *Trustworthy Global Com-*

puting - 8th International Symposium, TGC 2013, Buenos Aires, Argentina, August 30-31, 2013, Revised Selected Papers, volume 8358 of *Lecture Notes in Computer Science*, pages 83–102. Springer, 2013.

[HvGT+12] Peter Höfner, Rob J. van Glabbeek, Wee Lum Tan, Marius Portmann, Annabelle McIver, and Ansgar Fehnker. A rigorous analysis of AODV and its variants. In Albert Y. Zomaya, Björn Landfeldt, and Ravi Prakash, editors, *The 15th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, MSWiM '12, Paphos, Cyprus, October 21-25, 2012*, pages 203–212. ACM, 2012.

[IK04] Atsushi Igarashi and Naoki Kobayashi. A generic type system for the pi-calculus. *Theor. Comput. Sci.*, 311(1-3):121–163, 2004.

[JBPV10] Magnus Johansson, Jesper Bengtson, Joachim Parrow, and Björn Victor. Weak equivalences in psi-calculi. In *LICS*, pages 322–331. IEEE Computer Society, 2010.

[JLNU10] Mathias John, Cédric Lhoussaine, Joachim Niehren, and Adelinde Uhrmacher. The attributed pi-calculus with priorities. *Transactions on Computational Systems Biology XII*, 5945/2010:13–76, 2010.

[Joh10] Magnus Johansson. *Psi-calculi: a framework for mobile process calculi: Cook your own correct process calculus - just add data and logic*. PhD thesis, Uppsala University, Division of Computer Systems, 2010.

[JR05] Alan Jeffrey and Julian Rathke. Contextual equivalence for higher-order pi-calculus revisited. *Logical Methods in Computer Science*, 1(1), 2005.

[JVP10] Magnus Johansson, Björn Victor, and Joachim Parrow. A fully abstract symbolic semantics for psi-calculi. In *Proceedings of SOS 2009*, volume 18 of *EPTCS*, pages 17–31, 2010.

[JVP12] Magnus Johansson, Björn Victor, and Joachim Parrow. Computing strong and weak bisimulations for psi-calculi. *Journal of Logic and Algebraic Programming*, 81(3):162–180, 2012.

[Kel75] Robert M Keller. A fundamental theorem of asynchronous parallel computation. In *Parallel processing*, pages 102–112. Springer-Verlag, 1975.

[KP] Ondřej Kunčar and Andrei Popescu. From types to sets in Isabelle/HOL. Isabelle Workshop 2014.

[Kun80] Milan Kundera. *The Book of Laughter and Forgetting*. Alfred A. Knopf, New York, first edition, 1980.

[Lam94] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(3):872–923, 1994.

[LMS10] Ugo Dal Lago, Simone Martini, and Davide Sangiorgi. Light log-

ics and higher-order processes. In Fröschle and Valencia [FV10], pages 46–60.

[Loc10] Andreas Lochbihler. Coinductive. *Archive of Formal Proofs*, 2010, 2010.

[LPSS11] Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi, and Alan Schmitt. On the expressiveness and decidability of higher-order process calculi. *Inf. Comput.*, 209(2):198–226, 2011.

[LS10] Ivan Lanese and Davide Sangiorgi. An operational semantics for a calculus for wireless systems. *Theoretical Computer Science*, 411(19):1928–1948, 2010.

[McC63] John McCarthy. A basis for a mathematical theory of computation. *Computer Programming and Formal Systems*, pages 33–70, 1963.

[Mel94] Thomas F. Melham. A mechanized theory of the pi-calculus in HOL. *Nordic Journal of Computing*, 1(1):50–76, 1994.

[Mil79] Robin Milner. LCF: A way of doing proofs with a machine. In Jirí Becvár, editor, *MFCS*, volume 74 of *Lecture Notes in Computer Science*, pages 146–159. Springer, 1979.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*. Number 92 in Lecture Notes in Computer Science. Springer-Verlag, 1980.

[Mil83] Robin Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25(3):267 – 310, 1983.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice-Hall, Inc., 1989.

[Mil91] Robin Milner. The polyadic pi-calculus: a tutorial. Technical report, Logic and Algebra of Specification, 1991.

[Mil92] Robin Milner. Functions as processes. *Journal of Mathematical Structures in Computer Science*, 2(2):119–141, 1992. Previous version as Rapport de Recherche 1154, INRIA Sophia-Antipolis, 1990, and in *Proceedings of ICALP '91*, LNCS 443.

[MPS14] Jean-Marie Madiot, Damien Pous, and Davide Sangiorgi. Bisimulations up-to: Beyond first-order transition systems. In Paolo Baldan and Daniele Gorla, editors, *CONCUR 2014*, volume 8704 of *Lecture Notes in Computer Science*, pages 93–108. Springer Berlin Heidelberg, 2014.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I/II. *Inf. Comput.*, 100(1):1–77, 1992.

[MS92] Robin Milner and Davide Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Proceedings of ICALP '92*, volume 623 of *LNCS*, pages 685–695. Springer, 1992.

[MS06] Nicola Mezzetti and Davide Sangiorgi. Towards a calculus for wireless systems. *Electronic Notes in Theoretical Computer Science*, 158:331–353, 2006.

[MS10] Massimo Merro and Eleonora Sibilio. A timed calculus for wireless systems. In Farhad Arbab and Marjan Sirjani, editors, *Fundamentals of Software Engineering*, volume 5961 of *Lecture Notes in Computer Science*, pages 228–243. Springer Berlin Heidelberg, 2010.

[MS15] Petar Maksimovìc and Alan Schmitt. HOCore in Coq. In Christian Urban and Xingyuan Zhang, editors, *Interactive Theorem Proving - 6th International Conference, ITP 2015, Nanjing, China, August 24-27, 2015, Proceedings*, volume 9236 of *Lecture Notes in Computer Science*, pages 278–293. Springer, 2015.

[MTH90] Robin Milner, Mads Tofte, and Robert Harper. *Definition of Standard ML*. MIT Press, 1990.

[MWM14] Daniel Matichuk, Makarius Wenzel, and Toby Murray. An Isabelle proof method language. In *Interactive Theorem Proving*, pages 390–405. Springer, 2014.

[Nes92] Monica Nesi. Mechanizing a proof by induction of process algebra specifications in higher order logic. In Kim G. Larsen and Arne Skou, editors, *Computer Aided Verification*, volume 575 of *Lecture Notes in Computer Science*, pages 288–298. Springer Berlin Heidelberg, 1992.

[Nes00] Uwe Nestmann. What is a 'good' encoding of guarded choice? *Journal of Information and Computation*, 156:287–319, 2000. An extended abstract appeared in the *Proceedings of EXPRESS '97*, volume 7 of *ENTCS*.

[NH06] Sebastian Nanz and Chris Hankin. A framework for security analysis of mobile wireless networks. *Theoretical Computer Science*, 367(1-2):203–227, 2006.

[NP00] Uwe Nestmann and Benjamin C. Pierce. Decoding choice encodings. *Journal of Information and Computation*, 163:1–59, 2000. Also available as report BRICS-RS-99-42, Universities of Aalborg and Århus, Denmark, 1999. An extended abstract appeared in the Proceedings of *CONCGUR '96*, LNCS 1119, pages 179–194.

[NPH14] Håkon Normann, Cristian Prisacariu, and Thomas T. Hildebrandt. Concurrency models with causality and events as psi-calculi. In Ivan Lanese, Alberto Lluch-Lafuente, Ana Sokolova, and Hugo Torres Vieira, editors, *Proceedings 7th Interaction and Concurrency Experience, ICE 2014, Berlin, Germany, 6th June 2014.*, volume 166 of *EPTCS*, pages 4–20, 2014.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: a Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.

[Pal97a] Catuscia Palamidessi. Comparing the expressive power of the

synchronous and the asynchronous π-calculus. In *Proceedings of POPL '97*, pages 256–265. ACM, January 1997.

[Pal97b] Catuscia Palamidessi. Comparing the expressive power of the synchronous and the asynchronous π-calculus. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '97, pages 256–265, New York, NY, USA, 1997. ACM.

[Pal03] Catuscia Palamidessi. Comparing the expressive power of the synchronous and asynchronous pi-calculi. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003.

[Par81] David Park. Concurrency and automata on infinite sequences. In *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, pages 167–183, London, UK, UK, 1981. Springer-Verlag.

[Par08] Joachim Parrow. Expressiveness of process algebras. *Electr. Notes Theor. Comput. Sci.*, 209:173–186, 2008.

[Par16] Joachim Parrow. General conditions for full abstraction. *Mathematical Structures in Computer Science*, 26(4):655–657, 2016.

[Pau86] Lawrence C. Paulson. Natural deduction as higher-order resolution. *The Journal of Logic Programming*, 3(3):237–258, 1986.

[Pau98] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998.

[Pau99] Lawrence C. Paulson. A generic tableau prover and its integration with Isabelle. *Journal of Universal Computer Science*, 5(3):73–87, 1999.

[PBRD03] C. Perkins, E. Belding-Royer, and S. Das. Ad Hoc On-Demand Distance Vector (AODV) Routing. RFC 3561, RFC Editor, 2003.

[PE88] Frank Pfenning and Conal Elliott. Higher-order abstract syntax. In Richard L. Wexelblat, editor, *PLDI*, pages 199–208. ACM, 1988.

[Pet66] Carl Adam Petri. Communication with automata, 1966. DTIC Research Report AD0630125.

[Phi01] Iain Phillips. CCS with priority guards. In Kim G. Larsen and Mogens Nielsen, editors, *CONCUR 2001 — Concurrency Theory*, volume 2154 of *Lecture Notes in Computer Science*, pages 305–320. Springer Berlin Heidelberg, 2001.

[Pit03] Andrew M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186:165–193, 2003.

[Plo77] Gordon D. Plotkin. LCF considered as a programming language. *Theor. Comput. Sci.*, 5(3):223–255, 1977.

[Plo81] Gordon D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.

[PN10] Kirstin Peters and Uwe Nestmann. Breaking symmetries. In Fröschle and Valencia [FV10], pages 136–150.

[PNG13] Kirstin Peters, Uwe Nestmann, and Ursula Goltz. On distributability in process calculi. In Matthias Felleisen and Philippa Gardner, editors, *Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7792 of *Lecture Notes in Computer Science*, pages 310–329. Springer, 2013.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977.

[Pou07a] Damien Pous. Complete lattices and up-to techniques. In Zhong Shao, editor, *APLAS*, volume 4807 of *Lecture Notes in Computer Science*, pages 351–366. Springer, 2007.

[Pou07b] Damien Pous. New up-to techniques for weak bisimulation. *Theor. Comput. Sci.*, 380(1-2):164–180, 2007.

[Pou16] Damien Pous. Coinduction all the way up. In *Thirty-First Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), New York City, USA, July 5-8, 2016, Proceedings*, 2016. To appear, preliminary version at `https://hal.archives-ouvertes.fr/hal-01259622`.

[Pra91] K. V. S. Prasad. A calculus of broadcasting systems. In Samson Abramsky and T. S. E. Maibaum, editors, *TAPSOFT, Vol.1*, volume 493 of *Lecture Notes in Computer Science*, pages 338–358. Springer, 1991.

[Pra93] K. V. S. Prasad. A calculus of value broadcasts. In *IN PARLE'93*, pages 69–4. Springer Verlag LNCS, 1993.

[Pra94] K. V. S. Prasad. Broadcasting with priority. In Donald Sannella, editor, *ESOP*, volume 788 of *Lecture Notes in Computer Science*, pages 469–484. Springer, 1994.

[Pra95] K. V. S. Prasad. A calculus of broadcasting systems. *Science of Computer Programming*, 25(2-3):285–327, 1995.

[Pri14] Cristian Prisacariu. Actor network procedures as psi-calculi for security ceremonies. In Barbara Kordy, Sjouke Mauw, and Wolter Pieters, editors, *Proceedings First International Workshop on Graphical Models for Security, GraMSec 2014, Grenoble, France, April 12, 2014.*, volume 148 of *EPTCS*, pages 63–77, 2014.

[PV04] Iain Phillips and Maria Grazia Vigliotti. Electoral systems in ambient calculi. In Igor Walukiewicz, editor, *Foundations of Software Science and Computation Structures, 7th International*

Conference, FOSSACS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings, volume 2987 of *Lecture Notes in Computer Science*, pages 408–422. Springer, 2004.

[PvG15] Kirstin Peters and Rob J. van Glabbeek. Analysing and comparing encodability criteria. In Silvia Crafa and Daniel Gebler, editors, *Proceedings of the Combined 22th International Workshop on Expressiveness in Concurrency and 12th Workshop on Structural Operational Semantics, and 12th Workshop on Structural Operational Semantics, EXPRESS/SOS 2015, Madrid, Spain, 31st August 2015.*, volume 190 of *EPTCS*, pages 46–60, 2015.

[PW16] Joachim Parrow and Tjark Weber. The largest respectful function. *Logical Methods in Computer Science*, 12(2), 2016.

[Raa12] Palle Raabjerg. *Extending Psi-calculi and their Formal Proofs*. Licentiate thesis, Department of Information Technology, Uppsala University, November 2012.

[RBR13] Jurriaan Rot, Marcello M. Bonsangue, and Jan J. M. M. Rutten. Coalgebraic bisimulation-up-to. In Peter van Emde Boas, Frans C. A. Groen, Giuseppe F. Italiano, Jerzy R. Nawrocki, and Harald Sack, editors, *SOFSEM 2013: Theory and Practice of Computer Science, 39th International Conference on Current Trends in Theory and Practice of Computer Science, Špindlerův Mlýn, Czech Republic, January 26-31, 2013. Proceedings*, volume 7741 of *Lecture Notes in Computer Science*, pages 369–381. Springer, 2013.

[RBR16] Jurriaan Rot, Marcello M. Bonsangue, and Jan Rutten. Proving language inclusion and equivalence by coinduction. *Inf. Comput.*, 246:62–76, 2016.

[Rus19] Bertrand Russell. *Introduction to mathematical philosophy*. George Allen and Unwin, 1919.

[San93] Davide Sangiorgi. From pi-calculus to higher-order pi-calculus - and back. In Marie-Claude Gaudel and Jean-Pierre Jouannaud, editors, *TAPSOFT*, volume 668 of *Lecture Notes in Computer Science*, pages 151–166. Springer, 1993.

[San94] Davide Sangiorgi. Bisimulation in higher-order process calculi. In Ernst-Rüdiger Olderog, editor, *PROCOMET*, volume A-56 of *IFIP Transactions*, pages 207–224. North-Holland, 1994.

[San98] Davide Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8(5):447–479, October 1998.

[San01] Davide Sangiorgi. Asynchronous process calculi: the first- and higher-order paradigms. *Theor. Comput. Sci.*, 253(2):311–350, 2001.

[San09] Davide Sangiorgi. On the origins of bisimulation and coinduction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 31(4):15, 2009.

[San12] Davide Sangiorgi. *An introduction to bisimulation and coinduction*. Cambridge University Press, Cambridge, New York, 2012.

[SRS10] Anu Singh, C. R. Ramakrishnan, and Scott A. Smolka. A process calculus for mobile ad hoc networks. *Science of Computer Programming*, 75(6):440–469, 2010.

[SS04] Alan Schmitt and Jean-Bernard Stefani. The Kell Calculus: A family of higher-order distributed process calculi. In Corrado Priami and Paola Quaglia, editors, *Global Computing*, volume 3267 of *Lecture Notes in Computer Science*, pages 146–178. Springer, 2004.

[SS09] Nobuyuki Sato and Eijiro Sumii. The higher-order, call-by-value applied pi-calculus. In Zhenjiang Hu, editor, *APLAS*, volume 5904 of *Lecture Notes in Computer Science*, pages 311–326. Springer, 2009.

[SV16] Davide Sangiorgi and Valeria Vignudelli. Environmental bisimulations for probabilistic higher-order languages. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 595–607. ACM, 2016.

[TGRW04] Christian Tschudin, Richard Gold, Olof Rensfelt, and Oskar Wibling. LUNAR: a lightweight underlay network ad-hoc routing protocol and implementation. In *Proceedings of NEW2AN'04*, St. Petersburg, February 2004.

[Tho89] Bent Thomsen. A calculus of higher order communicating systems. In *POPL*, pages 143–154, 1989.

[Tho93] Bent Thomsen. Plain CHOCS: A second generation calculus for higher order processes. *Acta Inf.*, 30(1):1–59, 1993.

[UB06] Christian Urban and Stefan Berghofer. A recursion combinator for nominal datatypes implemented in Isabelle/HOL. In Ulrich Furbach and Natarajan Shankar, editors, *IJCAR*, volume 4130 of *Lecture Notes in Computer Science*, pages 498–512. Springer, 2006.

[UBN07] Christian Urban, Stefan Berghofer, and Michael Norrish. Barendregt's variable convention in rule inductions. In *Proc. of the 21th International Conference on Automated Deduction (CADE), volume 4603 of LNAI*, pages 35–50. Springer, 2007.

[Urb08] Christian Urban. Nominal techniques in Isabelle/HOL. *Journal of Automated Reasoning*, 40(4):327–356, May 2008.

[UT05] Christian Urban and Christine Tasson. Nominal techniques in Isabelle/HOL. In Robert Nieuwenhuis, editor, *Proceedings of*

CADE 2005, volume 3632 of *Lecture Notes in Computer Science*, pages 38–53. Springer-Verlag, 2005.

[VBG07] Cristian Versari, Nadia Busi, and Roberto Gorrieri. On the expressive power of global and local priority in process calculi. In *CONCUR*, pages 241–255, 2007.

[Ver07] Cristian Versari. A core calculus for a comparative analysis of bio-inspired calculi. In Rocco De Nicola, editor, *ESOP*, volume 4421 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2007.

[vG90] Rob J. van Glabbeek. The linear time-branching time spectrum (extended abstract). In Jos C. M. Baeten and Jan Willem Klop, editors, *CONCUR '90, Theories of Concurrency: Unification and Extension, Amsterdam, The Netherlands, August 27-30, 1990, Proceedings*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer, 1990.

[vG93] Rob J. van Glabbeek. The linear time - branching time spectrum II. In Best [Bes93], pages 66–81.

[vG04] Rob J. van Glabbeek. The meaning of negative premises in transition system specifications II. *J. Log. Algebr. Program.*, 60-61:229–258, 2004.

[VNN13] Roberto Vigo, Flemming Nielson, and Hanne Riis Nielson. Broadcast, denial-of-service, and secure communication. In Einar Broch Johnsen and Luigia Petre, editors, *IFM*, volume 7940 of *Lecture Notes in Computer Science*, pages 412–427. Springer, 2013.

[Wal95] David Walker. Objects in the π-calculus. *Journal of Information and Computation*, 116(2):253–271, 1995.

[Wen99] Markus Wenzel. Isar - a generic interpretative approach to readable formal proof documents. In *Theorem Proving in Higher Order Logics*, pages 167–184, 1999.

[YG08] Ayesha Yasmeen and Elsa L. Gunter. Implementing Secure Broadcast Ambients in Isabelle using Nominal Logic. In *Emerging Trends Report of the 21st International Conference on Theorem Proving in Higher Order Logics (TPHOLs 2008)*, pages 123–134, 2008.

Acta Universitatis Upsaliensis

Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 1397

Editor: The Dean of the Faculty of Science and Technology

A doctoral dissertation from the Faculty of Science and Technology, Uppsala University, is usually a summary of a number of papers. A few copies of the complete dissertation are kept at major Swedish research libraries, while the summary alone is distributed internationally through the series Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology. (Prior to January, 2005, the series was published under the title "Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology".)

Distribution: publications.uu.se
urn:nbn:se:uu:diva-297488

ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2016