UPPSALA
UNIVERSITET

IT Licentiate theses
2013-004

Bells and Whistles:
Advanced Language Features in Psi-Calculi

JOHANNES ÅMAN POHJOLA

UPPSALA UNIVERSITY
Department of Information Technology

Bells and Whistles:
Advanced Language Features in Psi-Calculi

Johannes Åman Pohjola
johannes.aman-pohjola@it.uu.se

October 2013

Division of Computing Science
Department of Information Technology
Uppsala University
Box 337
SE-751 05 Uppsala
Sweden

http://www.it.uu.se/

Dissertation for the degree of Licentiate of Philosophy in Computer Science

ISSN 1404-5117
Printed by the Department of Information Technology, Uppsala University, Sweden

Abstract

Psi-calculi is a parametric framework for process calculi similar to popular pi-calculus extensions such as the explicit fusion calculus, the applied pi-calculus and the spi calculus. Remarkably, machine-checked proofs of standard algebraic and congruence properties of bisimilarity apply to every instance of the framework.

The contribution of this licentiate thesis is to significantly extend the applicability and expressiveness of psi-calculi by incorporating several advanced language features into the framework: broadcasts, higher-order communication, generalised pattern matching, sorts and priorities. The extensions present several interesting technical challenges, such as negative premises. The machine-checked proofs for standard results about bisimilarity are generalised to each of these new settings, and the proof scripts are freely available online.

Acknowledgements

I am grateful to my supervisors, Joachim Parrow and Björn Victor, for providing support and guidance. I am also grateful to the co-authors of the papers that constitute this thesis, without whom the material herein would have been unimaginably less interesting. I am grateful to the rest of the Mobility group for providing a stimulating work environment.

I am also grateful to my friends and family — none mentioned, none forgotten; though my newborn daughter Linnea deserves a special thanks for having the courtesy to stay very calm throughout the writing process, allowing her father to focus more easily on writing this thesis.

List of papers

This thesis includes the following papers.

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. Accepted for publication in *Software and Systems Modeling*. An earlier conference version was published in *Proceedings of Software Engineering and Formal Methods*, volume 7041 of *Lecture Notes in Computer Science*, pages 74-89, Springer-Verlag, 2011.

Contributions: About a third of the mechanical proofs of correctness of standard structural properties of bisimilarity are due to me, including the laws of scope extensions and commutativity of binders. I have also done all the mechanical proofs pertaining to the LUNAR model and higher-order broadcast psi-calculi.

II Joachim Parrow, Johannes Borgström, Palle Raabjerg, and Johannes Åman Pohjola. Higher-order psi-calculi. In *Mathematical Structures in Computer Science*, Volume 23. Cambridge University Press, 2013.

Contributions: 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.

III Johannes Borgström, Ramūnas Gutkovas, Joachim Parrow, Björn Victor and Johannes Åman Pohjola. A Sorted Semantic Framework for Applied Process Calculi (extended abstract). Accepted to *TGC 2013*. An earlier version, entitled "Sorted psi-calculi with generalised pattern matching", was presented at *ICE 2012*.

Contributions: Mechanical proofs of correctness of standard congruence and structural properties of bisimilarity, preservation of well-formedness and backwards compatibility of the pattern matching are due to me. 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 and the polyadic synchronisation pi-calculus.

IV Johannes Åman Pohjola, Johannes Borgström, Joachim Parrow, Palle Raabjerg, Ioana Rodhe. Negative premises in applied process calculi. Technical Report 2013-014, Department of Information Technology, Uppsala University, 2013.

Contributions: I am the principal author and investigator, and took on a leadership role for this project. All the machine-checked proofs are mine; in particular, this includes all proofs pertaining to bisimilarity. Large parts of the text are authored by me, though writing has been a largely collaborative project, making it difficult to draw a clear line between different authors' contributions in this regard.

V Johannes Åman Pohjola. Bisimulation up-to techniques for psi-calculi. Unpublished draft, 2013.

Contributions: All aspects of this draft are wholly my work, though helpful comments by Joachim Parrow have helped improve the presentation.

Contents

Chapter 1

Introduction

An engineer designs solutions to problems, and the practice of modelling a solution before deploying it is central to the discipline of engineering. A civil engineer would not simply start building a bridge, a nuclear engineer would not simply start building a power plant — not before making extensive studies of prototypes and mathematical models. This initial step allows the engineer to predict with some confidence whether a design will work prior to its implementation.

Would a software engineer, then, simply start building a software system without first making extensive studies of prototypes and mathematical models? The answer to this question is often "yes". Software engineering has other quality assurance routines, but they usually enter the process at a later stage. The most common method of validating the quality of a software system is through testing, where the results of trial runs of the software are compared to the desired behaviour. However, note that testing yields no predictive power at the design stage — it is only applicable if there is a candidate implementation available to test.

As theoretical computer scientists, it is not our place to tell engineers how to do their job. However, should software engineers decide that they want to subject their designs to a rigorous scientific analysis that can make meaningful predictions about whether the design will work, we should be there to supply the tools to support that process.

The contribution of this thesis concerns the development of formal modelling languages, in which systems can be expressed in a rigorous manner and their correctness can be shown by formal reasoning, analogously to how a civil engineer can use the language of Newtonian physics to reduce the problem of the strength of materials of a bridge to a system of mathematical equations. We will focus our attention on languages for concurrent systems that synchronise via message passing, i.e. where several independent processes are running at the same time and can communicate with each other by sending and receiving messages. Such languages are called *process calculi* or *process algebras*.

More specifically, we work in the *psi-calculi* framework of Bengtson, Johansson, Parrow and Victor [BJPV09], which is a family of process calculi united by a common theory. In the field of process calculi, there is no one calculus to rule them all. Different language features and levels of abstraction are appropriate for different applications, so as new application domains arise, process calculists often find themselves introducing new calculi. Doing so however comes at the price of a certain amount of theoretical groundwork necessary to prove that the calculus is useful, which can constitute a significant labour investment. The contribution of psi-calculi is to define a family of process calculi where such results can be proved once and for all, instead of once for every new calculus. While the full expressive power of psi-calculi is yet to be explored, they encompass anything from minimalistic calculi for exploring fundamental theoretical questions in the field such as the pi-calculus [MPW92], to calculi with advanced features like the cryptographic primitives of the applied pi-calculus [AF01], or even highly specialised calculi tailored for expressing specific communication protocols [BHJ+11].

The work presented in this thesis advances the psi-calculi program primarily by introducing new language features, thus furthering their practical applicability while proving that the theoretical groundwork remains valid under the extensions. These features are:

- Two forms of broadcast communication (lossy and non-lossy), where messages have one sender and many receivers — the original communication model of psi-calculi is unicast communication, where there is one sender and exactly one receiver for many messages.

- Higher-order features, where messages can themselves be seen as processes and executed as such. This can be used to model for an example the dynamic deployment of subsystems within a larger system.

- A notion of data sorts for more precisely capturing the modelling language designer's intention. We illustrate this point by showing how to exactly capture several process calculi in the literature within our framework.

- A pattern matching mechanism that can capture computations on message terms, such as decryption of coded messages.

- A system of priorities between communication channels, which can also be used to express time. This allows more fine-grained control over system behaviour, and is often practically important: in a speaker announcement system, the fire alarm should take priority over today's lunch specials.

We also develop proof techniques for proving equivalence between processes within the framework.

Developing formalisms is not something to be taken lightly. We firmly believe that if we as theorists want engineers to eventually adopt the formal techniques we design, we ought to hold ourselves to at least the standard of

formality that our methods demand of engineers. It is somewhat embarrassing that in our field, informal and hand-written proofs of important results are the norm, where arguments by analogy, "as the reader may care to check" and similar rhetorical devices are frequently employed. Since the objects studied by process calculists can grow quite complex, it becomes difficult for us mere primates to keep track of all the details. Hence mistakes creep in easily, leading to erroneous results being published — Bengtson discusses a few examples in [Ben10, p. 463-464]. To remedy this, we go to great lengths to make sure that as many of our proofs as possible are machine-checked by a *theorem prover*, a computer program that proves mathematical theorems and leaves no room whatsoever for skimping on details. The resulting proof scripts are freely available online [ÅP13].

1.1 Outline

The structure of this thesis is as follows. The remainder of Chapter 1 will present background material, summarise the main contributions of this thesis, discuss related work and conclude. Sections 1.2.1-1.2.5 introduce some of the foundations that psi-calculi are built on: structured operational semantics, bisimulation, nominal sets and the theorem prover Isabelle. Section 1.3 presents previous work on psi-calculi. Section 1.4 summarises the new contributions to psi-calculi; related work and conclusions are discussed in Section 1.5. These sections are intended to be readable by an audience with a general orientation in computer science, who do not necessarily have a background in process calculi (although a reader familiar with the pi-calculus and related topics is certainly in for an easier ride).

Chapters 2-5 are the papers that constitute this thesis. Chapter 2 introduces broadcast psi-calculi, where lossy broadcast communication is introduced into the framework. Chapter 3 introduces higher-order psi-calculi, where messages can themselves be seen as processes and executed as such. Chapter 4 introduces sorted psi-calculi, where data sorts and an improved pattern matching mechanism are introduced. Chapter 5 extends the broadcasts of Chapter 2 to also capture non-lossy broadcast communication, and also introduces a priority system.

Finally, Chapter 6 discusses bisimulation up-to techniques for psi-calculi, a powerful proof technique for proving equivalences between processes.

Chapters 2-6 are intended to be readable primarily by fellow researchers in the field, though it is my hope that a more general audience will find them accessible after reading the introduction.

1.2 Background

1.2.1 Structured operational semantics

We will use the technique of *structured operational semantics*, often abbreviated as *SOS*, to give the semantics of psi-calculi. It was first introduced by Plotkin [Plo81], and has been applied in the field of concurrency theory to define the semantics of several different process calculi [Mil80, MPW92, Plo83].

Plotkin's approach to defining the semantics of languages is based on transition systems, which define a relation $\rightarrow$ between states of the language. In particular, we shall be concerned with *labelled transition systems*, often abbreviated as *LTS*, where transitions between states are annotated 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 $(P, \alpha, Q) \in \rightarrow$.

In an SOS, the transitions from a state is defined from the transitions of its components, by means of an inductive definition. We illustrate this with a simple example. Assume a set of atomic symbols ranged over by a, b. Consider a language whose states are 0 (denoting termination), a prefix operator $a.P$ (signifying output of a followed by the state P), and a *choice* operator $P + Q$ where P and Q are states (which may act as either P or Q). The actions of our language will be the aforementioned atomic symbols, and the act of outputting a symbol will be signified by it occurring as the label of a transition. The transition relation of this language, which we will refer to as CHOICE, is then defined inductively by the following rules:

$$\textsc{Out}\ \frac{}{a.P \xrightarrow{a} P} \qquad \textsc{Choice-L}\ \frac{P \xrightarrow{a} P'}{P + Q \xrightarrow{a} P'} \qquad \textsc{Choice-R}\ \frac{Q \xrightarrow{a} Q'}{P + Q \xrightarrow{a} Q'}$$

The rules are written with premises on top, and conclusions on the bottom. The purpose of CHOICE is purely illustrative — the only action available from a state is to output one of the atomic symbols on offer, rendering the language not very expressive. We will investigate CHOICE more formally in Section 1.2.2.

An alternative to the SOS approach which is sometimes used is *reduction semantics*, introduced by Milner [Mil90], inspired by Berry and Boudol [BB90]. Typically, they feature an explicit *structural congruence* rule, whereby certain natural algebraic laws for rewriting processes are postulated in the semantics, rather than derived from it. Such semantics also use an unlabelled transition relation that describes only how a process may evolve (or reduce), not what other processes may observe about it. Another alternative to the SOS paradigm

is *reduction contexts*, due to Felleisen and Hieb [FH92]. Rather than the (typically) syntax-directed rules which consume one operator at a time used in SOS or reduction semantics, reduction context semantics achieve a simpler formulation by an explicit closure under *contexts* in the rules. A context C is a process with a hole in it; $C[P]$ is the process C where the hole has been filled with P. We will not use such techniques in this thesis, preferring instead the SOS approach.

1.2.2 Bisimulation

In the previous section, we introduced Plotkin's approach to defining the behaviour of processes by means of a structured operational semantics. A natural question to ask is then 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 precisely one considers to be the behaviour of a process — for the purposes of this chapter, we will consider the labels associated to its transitions as such. The canonical choice of behavioural equivalence 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 2 (Bisimulation)**.** *A* bisimulation relation *for an LTS* $(\Sigma, \rightarrow, \alpha)$ *is a set* $\mathcal{R} \subseteq \Sigma \times \Sigma$ *such that for all* $(P, Q) \in \mathcal{R}$*:*

1. *If there are* P'*,* α *such that* $P \xrightarrow{\alpha} P'$*, there exists* Q' *such that* $Q \xrightarrow{\alpha} Q'$ *and* $(P', Q') \in \mathcal{R}$*.*

2. *If there are* Q'*,* α *such that* $Q \xrightarrow{\alpha} Q'$*, there exists* P' *such that* $P \xrightarrow{\alpha} P'$ *and* $(P', Q') \in \mathcal{R}$*.*

We will call two processes P and Q *bisimilar* if they are related by a bisimulation, and sometimes write $P \mathcal{R} Q$ to mean $(P, Q) \in \mathcal{R}$. Since bisimulation relations are closed under union, there is a largest bisimulation relation which we give the symbol $\sim$ and the name *bisimilarity*. We write $P \sim Q$ as shorthand for $(P, Q) \in \sim$.

Intuitively, two processes are bisimilar if for each transition that one of them can take, the other can take a transition such that the transitions lead into states where the processes can again imitate each other, and so on.

In the language CHOICE introduced in Section 1.2.1, we find that bisimulation lets us abstract away from structural differences between states that offer equivalent choices, as the following theorem illustrates.

Theorem 1. *In CHOICE, for all* P, Q, R, a*:*

1. $P + Q \sim Q + P$

2. $P + (Q + R) \sim (P + Q) + R$

3. $P + P \sim P$

4. $P + 0 \sim P$

5. *If* $P \sim Q$ *then* $P + R \sim Q + R$

6. *If* $P \sim Q$ *then* $a.P \sim a.Q$

We show proofs for a few of the cases in order to illustrate the bisimulation proof method in action, and leave the rest as exercises. A reader familiar with bisimulation can safely skip ahead.

Proof. We show only the proofs of clauses 1.1 and 1.6.

1. By showing that $\mathcal{R} \triangleq \{(P+Q, Q+P)\} \cup \{(P, P) : \mathbf{true}\}$ is a bisimulation relation. For $(R, R) \in \{(P, P) : \mathbf{true}\}$, it is trivial to see that any transition $R \xrightarrow{\alpha} R'$ can be matched by itself, and that $(R', R') \in \{(P, P) : \mathbf{true}\}$.

   Following transitions from $P + Q$, two cases must be considered:

   CHOICE-L: $P \xrightarrow{\alpha} P'$ holds, and the transition $P + Q \xrightarrow{\alpha} P'$ was derived using the CHOICE-L rule. Then $Q + P \xrightarrow{\alpha} P'$ can be derived with the CHOICE-R rule, which suffices since $(P', P') \in \{(P, P) : \mathbf{true}\}$.

   CHOICE-R: Similar.

   Finally, we must also consider transitions from $Q + P$ in a completely analogous manner.

6. By showing that $\mathcal{R} \triangleq \{(a.P, a.Q) : P \sim Q\} \cup \sim$ is a bisimulation relation. Transitions from $(P, Q) \in \sim$ lead back to $\sim$ since bisimilarity is a bisimulation relation. We then consider transitions from $a.P$, where $P \sim Q$. The only possible transition is $a.P \xrightarrow{a} P$. With the OUT rule we derive $a.Q \xrightarrow{a} Q$, which suffices since $P \sim Q$. When considering transitions from $a.Q$, the proof is similar, save for one detail: the fact that bisimilarity is symmetric, whose simple proof we omit, must be used.

□

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 certain labels (usually denoted by τ) are considered to be internal, unobservable actions. 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 recent book [San12] is a good starting point.

1.2.3 Rule formats and negative premises

The project of developing a general theory about structured operational semantics has been taken up by several authors, leading to the topic of *rule formats* [dS85, BIM88, GV89]. A rule format defines a class of SOS by constraining the formats that its inference rules may have, and offer a collection of results that hold for every SOS that adheres to the format, such as preservation properties of bisimulation.

One rule format that will be of particular interest to us is Groote's *ntyft/ntyxt* format [Gro93], due to its pioneering work on the semantics of *negative premises*. A negative premise is when the absence of a transition occurs as a premise in the derivation rule of another. Groote writes:

> Two problems arise when rules have negative premises. Often these problems seem to be ignored.
>
> 1. It is possible to give an inconsistent set of rules. This means that one can derive with the rules that a process can perform an action if and only if it cannot do so. In this case the rules do not define an operational semantics.
> 2. Even if the rules are consistent, it is not immediately obvious how these rules determine an operational semantics. The normal notion of provability of transitions where the rules are used as inference rules is not satisfactory.

Here, we convey the main ideas used by Groote to address these problems. For a more formal treatment we refer to [Gro93]. We will use the notation $P \stackrel{\alpha}{\not\longrightarrow}$ to mean that there are no α-transitions from P. Formally, this is defined as

$$P \stackrel{\alpha}{\not\longrightarrow} \triangleq \neg\exists P'.P \stackrel{\alpha}{\longrightarrow} P'$$

The first problem is addressed by introducing a notion of *stratification*. A stratification imposes a well-founded order on processes such that as one traverses a sequence of rule applications further towards the base case, the stratification is monotonically decreasing on the negative premises and monotonically non-increasing on the positive premises. The existence of a stratification hence

Definition 3 (Stratification)**.** *Suppose $P = (\Sigma, \rightarrow, A)$ is a labelled transition system defined by an SOS with negative premises. A function $S : (\Sigma \times A \times \Sigma) \Rightarrow \delta$, where δ is an ordinal, is called a* stratification *of P if for every rule*

$$\frac{\{t_k \xrightarrow{a_k} t'_k : k \in K\} \cup \{t_l \not\xrightarrow{a_l} : l \in L\}}{t \xrightarrow{a} t'}$$

of its operational semantics, where the contexts ranged over by t are processes that may contain variable names, and for every function σ which instantiates these variable names to concrete processes, it holds that:

1. *For all $k \in K$, $S(\sigma(\{t_k \xrightarrow{a_k} t'_k)) \leq S(\sigma(\{t \xrightarrow{a} t))$.*
2. *For all $l \in L$ and all contexts t'_l*

$$S(\sigma(\{t_l \xrightarrow{a_l} t'_l)) < S(\sigma(\{t \xrightarrow{a} t))$$

Table 1.1: Formal definition of stratification.

guarantees that the absence of a transition can not be a precondition for its derivation. Table 1.1 contains a formal definition of stratification.

The second problem stems from the fact that when the rules contains negative premises, the naive approach of viewing the transition system specification as an inductive definition would not work since the rules are then not monotonic. Given a stratification S, we can however view it as an inductive definition over strata, rather than over the rules directly, in the following sense.

Let $P = (\Sigma, \rightarrow, A)$ be a labelled transition system defined by an SOS with negative premises that has a stratification $S : (\Sigma \times A \times \Sigma) \Rightarrow \delta$. We let $\rightarrow_0^S$ denote those transitions that can be derived using the rules of P that do not have negative premises, where all transitions χ in the conclusion or premises are on stratum $S(\chi) = 0$. Then $\rightarrow_1^S$ denotes those transitions whose positive premises can be derived using the rules of P with conclusions at stratum 1, positive premises at stratum 1 or 0 and negative premises such that no matching positive premise is derivable in $\rightarrow_0^S$. We proceed thus for all strata, and define

$$\rightarrow_S \triangleq \bigcup_{0 \leq i < \delta} \rightarrow_i^S$$

to be the transition relation induced by P. Groote then goes on to show that if both S and S' are stratifications of P, then $\rightarrow^S = \rightarrow^{S'}$.

Other approaches to defining the meaning of transition systems with nega-

tive premises can be found in the literature. Bol and Groote introduce so-called *reductions* [BG96], a technique more generally applicable than stratification but which we will not need to make use of in this thesis. In it, the transitions are partitioned into sets of transitions that are certainly true, whose truth is unknown, and that are certainly false. With this information, P is reduced to another equivalent transition system where the truth and falsity of more transitions becomes certain. A good overview of the field is provided in a survey by van Glabbeek [vG04], which reviews the above and several other methods.

1.2.4 Nominal sets

When reasoning formally about syntax, such as within a theorem prover, the problem of *alpha-conversion* must be considered. For an illustratory 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, but if we view them syntactically they differ slightly: one quantifies over all x while the other quantifies over all 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 names is built into the underlying set theory.

We assume a countably infinite set of *names*[1] $\mathcal{N}$, ranged over by a, b, c, x, y, z. A nominal set is a set equipped with a *permutation action* $\cdot$. Intuitively, 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

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

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

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

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

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 permutations p, $p \cdot f(X) = f(p \cdot X)$. A constant symbol X is equivariant if for all permutations p, $p \cdot X = X$. 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 interested in more details is a recent 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 substituted for natural numbers indicating the binding depth of the name occurrence, an approach which in practice tends to cause arithmetic on indexes 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.

1.2.5 Isabelle

Isabelle [NPW02] is an *interactive theorem prover*, a computer program that can be used to formally prove mathematical theorems. Most of the main results presented in Chapters 2-6 have been formally proven using Isabelle, though we will generally not discuss the technical aspects of the formalisation in any detail. This section is intended to provide enough of an overview so that when we remark that a result has been "formalised in Isabelle" in later parts of this thesis, it is clear what this statement entails.

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 framework (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 by the existence of the HOL/Nominal package (sometimes called Nominal Isabelle) by Urban et al. [Urb08, UB06, UT05, HU10], which implements a framework for reasoning about the nominal sets introduced in Section 1.2.4. Features include support for inductive datatype definitions modulo alpha-equivalence which we use to define the syntax of psi-calculi, primitive recursive definitions over such datatypes, and some automated proof procedures to support 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 for human-readable proof scripts [Wen99], 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] 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. This mechanism is used to capture the parametric nature of psi-calculi. Finally, we would be amiss not to mention that the original

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

formalisation of psi-calculi in Isabelle is due to Bengtson [Ben10, BP09], and all Isabelle-related contributions of this thesis build on his work.

1.3 Psi-calculi

Psi-calculi can be seen as a powerful proof technique for standard meta-theoretical results, along the following lines. Suppose that a calculus $\mathcal{P}$ is a psi-calculus. Then it follows that standard algebraic properties of strong and weak bisimulation hold in $\mathcal{P}$ [BJPV09, JBPV10]. 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.

Another way to view psi-calculi is as an easy way to obtain a custom modelling language for concurrent systems, with precisely the features useful for the application in question.

A psi-calculus is created by instantiating parameters corresponding to what notions of data terms and logical dependencies are needed.

In this section, we will recapitulate definitions and results pertaining to psi-calculi that are relevant to the scope of this thesis. We also endeavour to give some intuition, though we recommend Johansson's PhD thesis [Joh10] for a treatment of psi-calculi that is more rich in terms of motivations and examples. Other developments of 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], and a type system [Hüt11].

1.3.1 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 $X[\tilde{x} := \tilde{T}]$ simultaneously replaces all occurrences of $\tilde{x}$ in X with the corresponding element of $\tilde{T}$. In practice, substitution can be chosen to be any function that satisfies certain requisites (see Table 1.2). Hence substitution may have behaviour that runs counter to the intuition of a simultaneous substitution — in this way,

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.
	$\Psi \otimes \mathbf{1} \simeq \Psi$
	$\Psi \otimes \Psi' \simeq \Psi' \otimes \Psi$
	$(\Psi \otimes \Psi') \otimes \Psi'' \simeq \Psi \otimes (\Psi' \otimes \Psi'')$
$\_[\_ := \_] : \mathbf{X} \times \mathcal{N}^\star \times \mathbf{T}^\star \Rightarrow \mathbf{X}$	$\_[\_ := \_]$ equivariant
	$\widetilde{x}, \widetilde{y}$ distinct $\wedge\, \widetilde{y} \# \widetilde{x}, X$
	$\Rightarrow X[\widetilde{x} := \widetilde{T}] = ((\widetilde{y}\ \widetilde{x}) \cdot X)[(\widetilde{y}\ \widetilde{x}) \cdot \widetilde{x} := \widetilde{T}]$
	$X \in \mathbf{T} \;\wedge\; \widetilde{x}$ distinct $\wedge\, \widetilde{x} \subseteq \mathsf{n}(X)$
	$\quad\Rightarrow \mathsf{n}(\widetilde{T}) \subseteq \mathsf{n}(X[\widetilde{x} := \widetilde{T}])$

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

Table 1.2: Parameters and requisites.

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.

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 regards to $\mathbf{1}$, a *unit assertion* which 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, eg. using a non-monotonic logic.

One more requisite needs to be mentioned: we require the channel equivalence relation $\dot{\leftrightarrow}$ 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 entails that there can be terms that are not usable as communication channels.

1.3.2 Syntax and semantics

For simplicity, we explain the syntax of psi-calculi through a running example where $\mathbf{A} \triangleq \mathcal{P}_{\text{fin}}(\mathbf{C})$ and $\vdash \triangleq \ni$ (ie. where an assertion is simply the set of conditions that are 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 from Section 1.2.1. We will introduce the syntax and semantics on an operator by operator basis. The agents $\mathbf{P}$ of a psi-calculus are ranged over by P, Q, R, actions are ranged over by α, and 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'.

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

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

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

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 $\leftrightarrow$ is syntactic equality and hence we will always have $M = K$ where the precondition holds by default.

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

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

Here $\widetilde{x}$ binds into N and P, and we require $\widetilde{x} \subseteq \mathsf{n}(N)$ and for $\widetilde{x}$ 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

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

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

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

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 statements denote condition-guarded, non-deterministic choice. For an example, the agent $\mathbf{case}\ \varphi : \overline{M}\, N.\mathbf{0} \,[]\, \varphi' : \underline{M}(\lambda\epsilon)N.\mathbf{0}$ 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:

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

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 $\overline{M}\,(\nu\epsilon)N$ and $\overline{M}\,N$. The expression $\nu\tilde{a} \cup \{b\}$ means the sequence $\tilde{a}$ with b inserted anywhere.

The *parallel composition* $P \mid Q$ is a process that executes both P and Q concurrently. Two processes composed in this manner may 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. This is achieved by the parallel composition rule

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

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 bound 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, hence $\Psi_Q = \emptyset$. 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:

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

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 $\mathcal{F}(P)$, 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, $\mathcal{F}((\!|\Psi|\!) \mid (\nu a)(\!|\Psi'|\!)) = (\nu a)(\Psi \mid \Psi')$ 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 $P = \underline{a}(\lambda x)x.\overline{b}\,x.\mathbf{0}$ 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 a special internal action, 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:

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

A symmetric version is elided, and we assume that $\mathcal{F}(P) = (\nu\widetilde{b}_P)\Psi_P$ and $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$ 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 $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$ 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 $P = \underline{x}(\lambda z)z.\overline{z}\,a.\mathbf{0}$ and $Q = (\nu y)\overline{x}\,y.\underline{y}(\lambda z)z.\mathbf{0}$. 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 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.

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

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

Process syntax	Well-formedness requisites
$\mathbf{0}$	
$(\!\lvert \Psi \rvert\!)$	
$\overline{M}\,N\,.\,P$	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 1.3: Process syntax and well-formedness

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

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

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

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

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

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

Table 1.4: Operational semantics. Symmetric versions of Com and Par are elided. In the rule Com we assume that $\mathcal{F}(P) = (\nu\widetilde{b}_P)\Psi_P$ and $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$ 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 $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$ 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.

1.3.3 Bisimulation

An attentive reader may have noticed that our way of defining the transition relation differs from the labelled transition systems introduced in Section 1.2.1 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 environment is considered part of the label instead. 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 this work.

These differences from the LTS approach suggest a different definition of bisimulation than the one introduced in Section 1.2.2. The definition is as follows, where the *static equivalence* relation $\simeq$ relates frames if and only if they entail the same conditions

Definition 4 (Strong bisimulation)**.** *A relation* $\mathcal{R} \subseteq \mathbf{A} \times \mathbf{P} \times \mathbf{P}$ *is a* strong bisimulation *iff for every* $(\Psi, P, Q) \in \mathcal{R}$

1. $\Psi \otimes \mathcal{F}(P) \simeq \Psi \otimes \mathcal{F}(Q)$ *(static equivalence)*
2. $(\Psi, Q, P) \in \mathcal{R}$ *(symmetry)*
3. $\forall \Psi'.(\Psi \otimes \Psi', P, Q) \in \mathcal{R}$ *(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* $(\Psi, P', Q') \in \mathcal{R}$ *(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 4.4 is essentially the first clause of Definition 2, 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 4.3 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 is not preserved by the parallel operator. Clause 4.2 is simply a convenience that lets us avoid having two simulation clauses. Finally, Clause 4.1 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*, ie. 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 5 (Strong congruence)**.** Strong congruence $\sim$ *is defined as:*

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

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

Finally we define *weak bisimulation*, where we abstract away from the internal behaviour of processes. The intuition is that processes should only be considered equivalent if they cannot 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.

We first introduce a few auxiliary notions. The *weak reduction* relation $\Longrightarrow$ is the reflexive and transitive closure of $\xrightarrow{\tau}$, so $\Psi \rhd P \Longrightarrow P'$ means that P evolves to P' in Ψ through zero or more τ steps. A *weak transition* $\Psi \rhd P \stackrel{\alpha}{\Longrightarrow} P'$ means that there exists P'', P''' such that $\Psi \rhd P \Longrightarrow P'' \xrightarrow{\alpha} P''' \Longrightarrow P'$, ie. that P evolves to P' in Ψ through one α step and any number of τ steps.

We say that P *statically implies* Q in the environment Ψ, written $P \leq_\Psi Q$, if $\Psi \otimes \mathcal{F}(Q)$ entails every condition that $\Psi \otimes \mathcal{F}(P)$ entails.

The definition of weak bisimulation is technically complicated because we allow for psi-calculi where the logic is non-monotonic. This means that a τ step in psi-calculi may reveal assertions that disable transitions in other parts of the environment. Indeed, the main source of complications in the following definition is the impact of such retracts.

Definition 6 (Weak bisimulation)**.** *A* weak bisimulation $\mathcal{R}$ *is a ternary relation between assertions and pairs of agents such that* $\mathcal{R}(\Psi, P, Q)$ *implies all of*

1. *Weak static implication:*

$$\begin{aligned}&\forall\Psi'\exists Q'', Q'.\\ &\quad \Psi \rhd Q \Longrightarrow Q'' \quad\wedge\quad P \leq_\Psi Q'' \quad\wedge\\ &\quad \Psi \otimes \Psi' \rhd Q'' \Longrightarrow Q' \quad\wedge\quad \mathcal{R}(\Psi \otimes \Psi', P, Q')\end{aligned}$$

2. *Symmetry:* $\mathcal{R}(\Psi, Q, P)$

3. *Extension of arbitrary assertion:*
   $\forall\Psi'.\ \mathcal{R}(\Psi \otimes \Psi', P, Q)$

4. *Weak simulation: for all* α, P' *such that* $\mathrm{bn}(\alpha)\#\Psi, Q$ *and* $\Psi \rhd P \xrightarrow{\alpha} P'$

it holds that

$$
\begin{array}{l}
\textit{if } \alpha = \tau : \exists Q'.\, \Psi \rhd Q \implies Q' \quad \wedge \quad \mathcal{R}(\Psi, P', Q') \\
\textit{if } \alpha \neq \tau : \forall \Psi' \exists Q'', Q'''. \\
\quad \Psi \rhd Q \implies Q''' \quad \wedge \quad P \leq_\Psi Q''' \quad \wedge \\
\quad \Psi \rhd Q''' \xrightarrow{\alpha} Q'' \quad \wedge \\
\quad \exists Q'.\, \Psi \otimes \Psi' \rhd Q'' \implies Q' \quad \wedge \quad \mathcal{R}(\Psi \otimes \Psi', P', Q')
\end{array}
$$

As compared to strong bisimulation, Clause 6.1 has been changed so that Q may perform a weak reduction before imitating the frame of P. A reader concerned about the change from static equivalence to static implication should note the symmetry clause present in both definitions; indeed, strong bisimulation can be given an equivalent formulation in terms of static implication. Clause 6.4 allows Q to imitate the transition from P with a weak transition. The technicalities such as the ordering of quantifiers and closures under arbitrary assertion environment extensions are quite delicate, and thoroughly motivated in [JBPV10].

In [JBPV10] it is shown that an equivalent but simpler formulation can be obtained for psi-calculi with monotonic logics, and that Definition 6 indeed captures the intuition that observers cannot tell weakly bisimilar processes apart. The latter is achieved by formalising observations as so-called *barbs*, which correspond to the output actions that a process may emit, and showing that a notion of barbed equivalence coincides with weak bisimilarity.

Weak bisimilarity is not preserved by input for the same reasons as strong bisimilarity. It is also not closed under case. To see this, let τ be shorthand for a prefix whose only action is an internal action.[3] Then $\tau.\mathbf{0} \mathrel{\dot{\approx}} \mathbf{0}$ holds, but

$$\mathbf{case}\ \varphi : \tau.\mathbf{0} \,[\!]\, \varphi : P \not\mathrel{\dot{\approx}} \mathbf{case}\ \varphi : \mathbf{0} \,[\!]\, \varphi : P$$

The former can do a τ-transition and discard the option to continue as P, but the latter cannot. We address this by requiring that τ actions be simulated by at least one τ action in the weak congruence relation:

Definition 7 (Weak congruence)**.** *P and Q are weakly τ-bisimilar, written $\Psi \rhd P \underset{\text{tau}}{\dot{\approx}} Q$, if $P \dot{\approx}_\Psi Q$ and they also satisfy* weak congruence simulation*:*

for all P' such that $\Psi \rhd P \xrightarrow{\tau} P'$ it holds:

$$\exists Q'.\, \Psi \rhd Q \stackrel{\tau}{\Longrightarrow} Q' \wedge P' \dot{\approx}_\Psi Q'$$

and similarly with the roles of P and Q exchanged. We define $P \approx Q$ to mean that for all Ψ, σ it holds that $\Psi \rhd P\sigma \underset{\text{tau}}{\dot{\approx}} Q\sigma$.

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

1.3.4 Main results

The following results demonstrate that the notions of bisimulation so far presented for psi-calculi are useful in the sense that they are closed under operators of the calculus, and satisfy natural algebraic laws. Establishing such results for a process calculus is often difficult and tedious work; if the process calculus is a psi-calculus these results are already established.

First, strong bisimulation is a congruence with regards to all operators save for input (note the closure under substitutions in the premise of Clause 2.6):

Theorem 2 (Congruence properties of strong bisimulation)**.**

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

Theorem 3. $\sim_\Psi$ *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 4 (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 5. *All congruence properties of* $\dot{\sim}$ *established in Theorem 2, except for Clause 2.4 pertaining to the case construct, also hold for* $\dot{\approx}$*.*

Theorem 6. $\approx$ *is a congruence.*

Theorem 7. *All structural laws of* $\sim$ *established in Theorem 4 also hold for* $\approx$*.*

A strong selling 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, this represents a cyclopean investment of labour by Bengtson that we adapt and re-use to formalise new results throughout this thesis.

1.4 Contributions

In this section, we summarise the novel contributions of this thesis.

1.4.1 Broadcasts

The original psi-calculi framework is based on point-to-point communication: a single sender and a single receiver can synchronise, yielding an internal action. In broadcast communication, there may be several receivers (but only one sender) for each transmission. We accommodate this by making a synchronisation between sender and receiver yield not an internal action, but an output action that may then be coupled with other input actions.

Two forms of broadcast communication are supported: *reliable* broadcasts, where messages are guaranteed to be received by everyone listening, and *unreliable* broadcasts, where there is no such guarantee. In broadcast psi-calculi, they can formally co-exist both with each other and with point-to-point communication, and it's even possible to dynamically change the reliability of a channel during execution of a process.

This is achieved by adding three new equivariant functions as parameters to the framework:

1. *Output connectivity* $\dot{\prec} : \mathbf{T} \times \mathbf{T} \Rightarrow \mathbf{C}$
2. *Input connectivity* $\dot{\succ} : \mathbf{T} \times \mathbf{T} \Rightarrow \mathbf{C}$
3. *Reliability* $\texttt{reliable} : \mathbf{T} \Rightarrow \mathbf{C}$

$M \mathrel{\dot{\prec}} K$ means that the prefix M can send messages on the channel K, and conversely $K \mathrel{\dot{\succ}} M$ means that M can receive messages on the channel K. For

an example, suppose that a radio receiver s is able to receive signals from the transmitter t at 100 mHz. This can be expressed as $t \dot{\prec} 100\text{ mHz} \dot{\succ} s$, and entails that a communication between s and t on the frequency in question is possible. Note that unlike the channel equivalence relation for point-to-point communication, we do not require symmetry or transitivity. This is appropriate for wireless communication — in the radio example s might have a weaker transmitter than t, so clearly the possibility of a symmetric communication should not be imposed by the framework. However, for technical reasons related to scope extension, we require that a channel contains no more names than the prefixes connected to it.

`reliable` M means that transmissions on the communication channel M cannot be lost. This is appropriate when modelling eg. communication on a hardware bus, and perhaps less appropriate when modelling wireless communication.

We introduce two new types of actions, in order to distinguish broadcast and point-to-point communication: *broadcast output* $\overline{!M}\,(\nu\widetilde{x})N$ and *broadcast input* $\underline{?M}\,N$. The same prefixes are used for both broadcast and point-to-point communication, so we do not need to introduce any new process syntax.

In order to account for broadcast communication, we need to introduce a few new rules into the semantics. First, the rules BrOut, BrIn and BrOpen are analogous to their point-to-point counterparts Out, In and Open.

$$\textsc{BrOut}\ \frac{\Psi \vdash M \dot{\prec} K}{\Psi \rhd \overline{M}\,N\,.\,P \xrightarrow{\overline{!K}\,N} P}$$

$$\textsc{BrIn}\ \frac{\Psi \vdash K \dot{\succ} M}{\Psi \rhd \underline{M}(\lambda\widetilde{y})N\,.\,P \xrightarrow{\underline{?K}\,N[\widetilde{y}:=\widetilde{L}]} P[\widetilde{y}:=\widetilde{L}]}$$

$$\textsc{BrOpen}\ \frac{\Psi \rhd P \xrightarrow{\overline{!K}\,(\nu\widetilde{a})N} P' \quad b\#\widetilde{a}, \Psi, K \quad b \in \mathrm{n}(N)}{\Psi \rhd (\nu b)P \xrightarrow{\overline{!K}\,(\nu\widetilde{a}\cup\{b\})N} P'}$$

The rule BrCom handles interaction between broadcast input and broadcast output actions.

$$\textsc{BrCom}\ \frac{\Psi_Q \otimes \Psi \rhd P \xrightarrow{\overline{!K}\,(\nu\widetilde{a})N} P' \qquad \Psi_P \otimes \Psi \rhd Q \xrightarrow{\underline{?K}\,N} Q'}{\Psi \rhd P \mid Q \xrightarrow{\overline{!K}\,(\nu\widetilde{a})N} P' \mid Q'}\ \widetilde{a}\#Q$$

The main change from Com is that the interaction does not yield a τ action, but a broadcast output action. This ensures that a broadcast can have multiple receivers, by successive applications of the BrCom rule.

There is also a similar rule for fusing two broadcast input actions into a single action:

$$\textsc{BrMerge}\ \frac{\Psi_Q \otimes \Psi \rhd P \xrightarrow{?K\ N} P' \qquad \Psi_P \otimes \Psi \rhd Q \xrightarrow{?K\ N} Q'}{\Psi \rhd P \mid Q \xrightarrow{?K\ N} P' \mid Q'}$$

This should be thought of as the broadcast distributing over parallel composition, rather than as an interaction between P and Q. The need for the BrMerge rule is perhaps less intuitively obvious than the BrCom rule. Without such a rule, it turns out that parallel composition is not associative. To see why, let $P = \underline{M}(\lambda\epsilon)N$ and $Q = \overline{M}\,N$ where $M \dot{\prec} M \dot{\succ} M$, and consider the agents $(Q \mid P) \mid P$ and $Q \mid (P \mid P)$. A transition where the message from Q reaches both copies of P cannot be derived without the BrMerge rule in the latter agent; in the former, it can be derived using only BrCom.

As for the Com rule, a symmetric version of BrCom is elided. In the rules BrCom and BrMerge we assume that $\mathcal{F}(P) = (\nu\widetilde{b}_P)\Psi_P$ and $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$ where $\widetilde{b}_P$ is fresh for $P, \widetilde{b}_Q, Q, K$ and Ψ, and that $\widetilde{b}_Q$ is fresh for $Q, \widetilde{b}_P, P, K$ and Ψ.

The BrClose rule states that broadcast outputs are unobservable outside the scope of the broadcast channel:

$$\textsc{BrClose}\ \frac{\Psi \rhd P \xrightarrow{!\overline{K}\ (\nu\widetilde{a})N} P'}{\Psi \rhd (\nu b)P \xrightarrow{\tau} (\nu b)(\nu\widetilde{a})P'}\ \begin{array}{l} b \in \mathsf{n}(K) \\ b\#\Psi \end{array}$$

These rules fully account for the semantics of broadcast save for one detail, namely reliable broadcasts. Reliable broadcasts, where a message must reach every able recipient, is incompatible with the Par rule, which allows messages sent by parallel components to be disregarded. We solve this by introducing a negative premise in the Par rule, which guarantees that it can only be applied to reliable broadcasts if both BrCom and BrMerge are inapplicable. The new rule looks like this:

$$\textsc{Par}\ \frac{\Psi_Q \otimes \Psi \rhd P \xrightarrow{\alpha} P'}{\Psi \rhd P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \begin{array}{l} \mathrm{ADMIT}(\alpha, \Psi \otimes \mathcal{F}(P), Q) \\ \mathrm{bn}(\alpha)\#Q \end{array}$$

Where the side-condition ADMIT is defined as:

$$\mathrm{ADMIT}(\alpha, F, Q) \triangleq \begin{cases} \textit{True} \text{ if } \alpha \text{ is not a broadcast action} \\ \neg(F \otimes \mathcal{F}(Q) \vdash \mathtt{reliable}(M) \wedge F \rhd Q \xrightarrow{?M\ N}\!\!\!\!\!\!/\ \ ) \\ \qquad \text{if } \alpha = \underline{?M}\ N \text{ or } \alpha = \overline{!M}\ (\nu\widetilde{a})N. \end{cases}$$

Recall from Section 1.2.3 that naive use of negative premises can create scenarios where a transition can be derived if and only if it cannot. Replication

together with reliable broadcast input can create such a scenario: consider the agent $P = !\underline{M}(x).\mathbf{0}$, where M is a reliable broadcast input channel. No transition can be derived from P — any derivation of such a transition has no base case due to the negative premise in the PAR rule. However, since P does not have a transition, it has a transition by the following "derivation":

$$\textsc{Rep}\ \frac{\textsc{Par}\ \dfrac{\underline{M}(x).\mathbf{0} \xrightarrow{\underline{?M}\,N} \mathbf{0} \qquad P \xrightarrow{\underline{?M}\,N}\!\!\!\!\!\!\rightarrow}{\underline{M}(x).\mathbf{0} \mid P \xrightarrow{\underline{?M}\,N} \mathbf{0} \mid P}}{P \xrightarrow{\underline{?M}\,N} \mathbf{0} \mid P}$$

Naturally, this will not do. Our solution is to forbid reliable broadcast inputs under replication. Hence we require that in the agent $!P$, P is *reliable reception guarded*. An agent is reliable reception guarded if none of its top-level input prefixes can be used for reliable broadcasts, regardless of which context it occurs in.

With this restriction in place, our semantics is indeed well-defined:

Theorem 8. *The transition relation $\longrightarrow$ of broadcast psi-calculi is stratifiable.*

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. *All results about strong bisimulation and strong congruence described in Section 1.3 also apply to broadcast psi-calculi.*

We illustrate how the framework 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 it is used to prove a basic reachability property of the protocol.

1.4.2 Higher-order data

In a higher-order process calculus, processes are first-class objects that may themselves be sent and received as messages. In that sense, psi-calculi are already higher-order: nothing prevents us from instantiating the framework so that the set of terms includes the set of processes. Hence we can use processes as messages or communication channels, and do pattern matching on them, just like any other terms. What is missing, however, 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 a handle for. What it means for M to be a handle 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:

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

We can immediately see that this is a conservative extension of psi-calculi: the original psi-calculi can be recovered by simply choosing the entailment relation so that $\Psi \nvdash M \Leftarrow P$ for all Ψ, M, P. The motivating example above, a mechanism for executing received processes, can be recovered by letting $\Psi \vdash P \Leftarrow P$ for all Ψ, P. A process that receives another process on channel M and then executes it can then be written as $\underline{M}(\lambda x)x.\mathbf{run}\ x$.

While sending processes in this manner is a standard technique in the literature [Tho89, Tho93, San93b], it is in fact unnecessary in psi-calculi; it suffices to send a handle that can be used to invoke the process. For an example, suppose that clauses may occur in assertions. The process

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

asserts that N is a handle for P and sends the handle along M. If the process $\underline{M}(\lambda x)x.\mathbf{run}\ x$ occurs in parallel, it can receive the handle and then invoke it as P, just as if the process itself had been sent. In psi-calculi, we will prefer sending handles to sending processes.

More advanced behaviour such as recursion and non-determinism can be easily obtained. Recursion since in $M \Leftarrow P$, the process P may of course contain **run** M. Non-determinism since $\Psi \vdash M \Leftarrow P$ and $\Psi \vdash M \Leftarrow P'$ may hold for distinct processes P, P'.

We show that the rich spectrum of behaviour that can be expressed with the **run** construct renders some operators of psi-calculi superfluous. For an example, replication $!P$ can be encoded as

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

where M_a is a handle that contains the name a, which is fresh in P. Through repeatedly applying the Invoke rule, M_a may spawn an unbounded number of copies of P just as $!P$ does. There are of course psi-calculi where handles and single-clause assertions that can be scoped in this manner are absent, hence this encoding does not apply to all psi-calculi. We use Isabelle to establish precise requisites on a psi-calculus such that this encoding is bisimilar to a replication. Similar encodings are achieved for the choice operator $+$, and the n-ary case construct can be encoded using only a unary case construct.

A notion of *canonical* higher-order psi calculi is provided, which is a means of constructing a higher-order psi calculus from any first-order psi calculus by enriching it with a notion of parametrised clauses on the form $M(\lambda\widetilde{x})N \Leftarrow P$, such that

$$\{M(\lambda\widetilde{x})N \Leftarrow P\} \vdash M\langle N[\widetilde{x} := \widetilde{T}]\rangle \Leftarrow P[\widetilde{x} := \widetilde{T}]$$

We show that a canonical higher-order psi-calculus satisfies all the requisites of higher-order psi-calculi, and that it is among the calculi for whom the above-mentioned encodings of replication, choice and n-ary case are valid.

The meta-theory of both strong and weak bisimulation carry over to higher-order psi-calculi:

Theorem 10. *All the results in Section 1.3 also apply to higher-order psi-calculi.*

The standard notion of bisimulation for first-order process calculi is often considered unsatisfactory in a higher-order setting. For an example, consider the processes $\overline{M}\,P.0$ and $\overline{M}\,P'.0$, where $P \mathrel{\dot{\sim}} P'$. It is reasonable to argue that they should be behaviourally equivalent even though they give rise to syntactically different output actions, since they will yield the same behaviour anyway once the process has been received and executed. However, in psi-calculi it is possible to do more with a received process than merely executing it. For an example, pattern matching can be used on a received process to find its outermost operator, or a case statement can be used to test whether a process is syntactically equal to another; neither of these would treat bisimilar processes-as-messages equally. It is even possible to use processes as communication channels, and $\Psi \vdash P \leftrightarrows P'$ need not entail $P \mathrel{\dot{\sim}_\Psi} P'$. This is in contrast to traditional higher-order process calculi, where processes can only be sent, received and executed; see eg. [Tho89, Tho93, San93b].

Hence, we take a different approach to higher-order equivalences, and strive to equate processes entailing bisimilar clauses rather than actions sending bisimilar processes.

Definition 8 (HO-Bisimulation)**.** *A* strong HO-bisimulation $\mathcal{R}$ *is a ternary relation between assertions and pairs of agents such that* $(\Psi, P, Q) \in \mathcal{R}$ *implies all of*

1. *Static equivalence:*

   (a) $\forall \varphi \in \mathbf{C}.\quad \Psi \otimes \mathcal{F}(P) \vdash \varphi \;\Rightarrow\; \Psi \otimes \mathcal{F}(Q) \vdash \varphi$

   (b) $\forall (M \Leftarrow P') \in \mathbf{Cl}.\quad \Psi \otimes \mathcal{F}(P) \vdash M \Leftarrow P' \;\Rightarrow$
   $\exists Q'.\, \Psi \otimes \mathcal{F}(Q) \vdash M \Leftarrow Q' \wedge (\mathbf{1}, P', Q') \in \mathcal{R}$

2. *Symmetry:* $(\Psi, Q, P) \in \mathcal{R}$

3. *Extension of arbitrary assertion:* $\forall \Psi'.\ (\Psi \otimes \Psi', P, Q) \in \mathcal{R}$

4. *Simulation: for all* α, P' *such that* $\mathrm{bn}(\alpha)\#\Psi, Q$ *there exists a* Q' *such that*

$$\textit{if } \Psi \rhd P \xrightarrow{\alpha} P' \textit{ then } \Psi \rhd Q \xrightarrow{\alpha} Q' \wedge (\Psi, P', Q') \in \mathcal{R}$$

We define $\Psi \rhd P \mathrel{\dot{\sim}^{\text{HO}}} Q$ *to mean that there exists a strong HO-bisimulation* $\mathcal{R}$ *such that* $\Psi \rhd P \,\mathcal{R}\, Q$*, and write* $P \mathrel{\dot{\sim}^{\text{HO}}} Q$ *for* $\mathbf{1} \rhd P \mathrel{\dot{\sim}^{\text{HO}}} Q$*.*

The only change from strong bisimulation is in the static equivalence clause, where (b) states that HO-bisimilar clauses rather than merely syntactically equivalent clauses are admissible. Strong higher-order congruence is defined by closing higher-order bisimilarity under substitutions, just as in the first-order case. This yields equivalence relations that satisfy the expected algebraic properties, and achieve our aim of identifying HO-bisimilar clauses:

Theorem 11. *All the results in Section 1.3 pertaining to strong bisimulation and strong congruence also apply to strong HO-bisimulation and HO-congruence.*

Theorem 12. *In a canonical higher-order psi-calculus, if* $P \mathrel{\dot{\sim}^{\text{HO}}} Q$ *then*

$$(\!|\{M \Leftarrow P\}|\!) \mathrel{\dot{\sim}^{\text{HO}}} (\!|\{M \Leftarrow Q\}|\!)$$

Note that when the approach of sending handles for processes rather than processes themselves is used, Theorem 12 is the counterpart to equating bisimilar messages. If $P \mathrel{\dot{\sim}^{\text{HO}}} P'$, while $\overline{M}\,P.0 \mathrel{\dot{\sim}^{\text{HO}}} \overline{M}\,P'.0$ does not hold in general, $\overline{M}\,N.0 \mid (\!|N \Leftarrow P|\!) \mathrel{\dot{\sim}^{\text{HO}}} \overline{M}\,N.0 \mid (\!|N \Leftarrow P'|\!)$ does.

1.4.3 Pattern matching

As described in Section 1.3, pattern matching in the original psi-calculi is defined in terms of substitution, an approach that is common in the literature [Gel85, HJ06].

In calculi for cryptographic applications, 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 type of 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 $\underline{M}(\lambda\widetilde{x})X$. The parameter VARS governs which parts of patterns may be λ-bound: formally, $\text{VARS}(X) \subseteq \mathcal{P}(\mathsf{n}(X))$ and in a bound pattern $(\lambda\widetilde{x})X$ we require that $\widetilde{x} \in \text{VARS}(X)$. This mechanism can be used to remedy the degenerate cryptographic example $\underline{M}(\lambda x)\mathbf{enc}(m,x).P$, where **enc** is used as a pattern to extract the encryption key, by excluding x from $\text{VARS}(\mathbf{enc}(m,x))$.

The MATCH function is another parameter which determines what it means for a term to match a pattern. The intuition is that if $\widetilde{T} \in \text{MATCH}(N,\widetilde{x},X)$, then the prefix $\underline{M}(\lambda\widetilde{x})X$ can be used to receive the message N, yielding a substitution $[\widetilde{x} := \widetilde{T}]$. This is captured by the new input rule:

$$\text{In}\ \frac{\Psi \vdash M \leftrightarrows K \qquad \widetilde{L} \in \text{MATCH}(N,\widetilde{x},X)}{\Psi \rhd \underline{M}(\lambda\widetilde{x})X.P \xrightarrow{\underline{K}\,N} P[\widetilde{x} := \widetilde{L}]}$$

The name preservation law of psi-calculi is only used to guarantee certain properties about substitutions occurring in the label. Here, no substitution happens in the label, rendering the name-preservation law obsolete. This allows us to safely use rewrite rules such as the decryption rule above.

Two requisites must be imposed on MATCH, beyond the usual equivariance. The first states that if $\widetilde{x} \in \text{VARS}(X)$ and $\widetilde{T} \in \text{MATCH}(N,\widetilde{x},X)$ then $|\widetilde{x}| = |\widetilde{T}|$ and $\mathsf{n}(\widetilde{T}) \subseteq \mathsf{n}(N) \cup (\mathsf{n}(X) - \widetilde{x})$. This guarantees that transitions cannot invent fresh names, an important technical lemma. The second is an alpha-conversion law, which intuitively guarantees that we can treat the names $\widetilde{x}$ in $\text{MATCH}(N,\widetilde{x},X)$ as binding into X.

For VARS, we require that if $\widetilde{x} \in \text{VARS}(X)$ and $\widetilde{x}\#\sigma$ then $\widetilde{x} \in \text{VARS}(X\sigma)$. This prevents substitutions on patterns from erasing pattern variables, a behaviour which is admissible in the original psi-calculi but leads to calculi where the set of well-formed processes (see Table 1.3) is not closed under the transition relation. With the new pattern matching mechanism, we are happy that this oversight has been corrected:

Theorem 13 (Subject reduction)**.** *If P is well-formed and $\Psi \rhd P \xrightarrow{\alpha} P'$, then P' is well-formed.*

Beyond that, we show that the usual meta-theoretical results carry over:

Theorem 14. *All the results in Section 1.3 also apply to psi-calculi with generalised pattern matching.*

In Chapter 4, several examples demonstrate how this framework 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].

1.4.4 Sorts

In the polyadic pi-calculus [Mil91], names are channels on which sequences of names can be transmitted. On 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* $\mathcal{S}_\mathcal{N}$. For each sort $s \in \mathcal{S}_\mathcal{N}$ there is a countably infinite set of names, disjoint from the names of all other sorts. We also assign sorts to terms and patterns from the set of sorts $\mathcal{S} \supseteq \mathcal{S}_\mathcal{N}$ with a function SORT, and overload it so that for all names a of sort $s \in \mathcal{S}_\mathcal{N}$, $\text{SORT}(a) = s$. The idea is to restrict the process syntax so that the use of terms and patterns in processes must respect the following compatibility predicates:

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

For an example, the process $\overline{M}\, N.\mathbf{0}$ is well-formed iff $\text{SORT}(M)\ \overline{\propto}\ \text{SORT}(N)$. The substitutability relation $\prec$ does not directly impact the process syntax, but is used to restrict the set of admissible substitutions. The compatibility predicates are parameters that may be chosen freely.

Returning to the example of the polyadic pi-calculus, we can remedy the problem of junk terms by using sorts. Let $\text{SORT}(n) = \mathbf{chan}$ for all $n \in \mathcal{N}$, and $\text{SORT}(\langle \widetilde{x} \rangle) = \mathbf{tup}$. If we set $\overline{\propto} = \underline{\propto} = \{\mathbf{chan}, \mathbf{tup}\}$, $\prec = \{(\mathbf{chan}, \mathbf{chan})\}$ and $\text{VARS}(\langle \widetilde{x} \rangle)\{\widetilde{x}\}$, we obtain a psi-calculus with a strong operational correspondence to the polyadic pi-calculus, in the sense that the syntax is isomorphic and the operational semantics agree modulo strong bisimulation.

However, a clarification is in order. In Chapter 4 we claim such an operational correspondence with the structured operational semantics for the polyadic pi-calculus presented in Sangiorgi's Ph.D. thesis [San93a]. Unfortunately, there is a difference between psi-calculi and Sangiorgi's treatment of the polyadic pi-calculus that we fail to account for. It concerns the treatment of bound output actions: in psi-calculi the binders $\widetilde{x}$ in $\overline{M}\,(\nu\widetilde{x})N$ are treated as a sequence, whereas Sangiorgi's thesis implicitly uses the convention that they are sets, and hence the order of the binders is immaterial. Thus there is not a one-to-one correspondence between labels in the calculi as we claim, but rather a single label in the polyadic pi-calculus corresponds to a set of labels in its psi-calculus representation; our proofs should be adjusted accordingly, which we are currently working towards.

Meta-theoretical results include subject reduction, plus the standard results concerning strong and weak bisimulation.

Theorem 15 (Subject reduction)**.** *If P is well-formed and $\Psi \rhd P \xrightarrow{\alpha} P'$, then P' is well-formed.*

Note that this subject reduction result is stronger than the corresponding result from Section 1.4.3, since well-formedness here includes well-sortedness.

Theorem 16. *All the results in Section 1.3, except those pertaining to weak congruence, also apply to sorted psi-calculi.*

While we strongly believe that the standard results for weak congruence also hold in this setting, limitations in Nominal Isabelle prevent us from carrying them out formally. Specifically, Nominal Isabelle is currently not compatible with this kind of parametric sort system; while it is possible to use several name sorts, they must be fixed in advance.

For Theorem 16, the corresponding results for the trivially sorted case are verified in Nominal Isabelle. Manual proofs lift these result to the general case, thus relying on hand-written proofs at a minimum. Note also that the lifting technique we use applies only to bisimilarity, and not its induced congruence; for these, hand-written proofs must be carried out from scratch, a process we believe is too error-prone to be worthwhile in the case of weak congruence.

Further results include proofs of strong operational correspondence with a handful of calculi beyond the polyadic pi-calculus, namely value-passing CCS [Mil89], the polyadic synchronisation pi-calculus [CM03], and the sorted polyadic pi-calculus [Mil91].

1.4.5 Priorities

Priorities allow certain actions to take precedence over others. This is useful when modelling systems because it admits more fine-grained control over the model's behaviour. Phenomena that exhibit prioritised behaviour include eg. 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]. In psi-calculi, we prefer a more dynamic approach, where the priority level of an action depends on the assertion environment. Specifically, a new equivariant operator

$$\texttt{has\_prio} : \mathbf{T} \times \mathbb{N} \Rightarrow \mathbf{C}$$

is added as a psi-calculus parameter. We write $\texttt{prio}(M) = p$ for $\texttt{has\_prio}(M, p)$. The intuition is that if $\Psi \vdash \texttt{prio}(M) = p$, then the priority level of communication on the channel M in the environment Ψ is p, where lower values of p indicate *higher* priority. An action with subject M may only occur if no higher priority internal action is available. Input and output actions do not in and of themselves block lower priority actions — if they did, bisimilarity would not preserved by restriction [CH88].

The semantics of psi-calculi with priorities is as the semantics of psi-calculi, but with two changes. The first is that τ actions are replaced with $\tau : p$ actions, where p is the priority level of the transition. The second is that the rules are augmented with side conditions that prevent a process from taking low priority actions. These side conditions constitute negative premises. Technically, we define the predicate

$$\text{HIGHEST}(\alpha, \Psi, P) := \neg\exists n.(\Psi \rhd P \xrightarrow{\tau:n} \wedge\, n < \texttt{PRIO}\,(\Psi \otimes \mathcal{F}(P), \alpha))$$

where $\texttt{PRIO}\,(F, \alpha)$ is defined to be p if $\alpha = \tau : p$ or if M is the subject of α and $F \vdash \texttt{prio}(M) = p$. Intuitively $\text{HIGHEST}(\alpha, \Psi, P)$ means that no τ transition whose priority is higher than that of α can be derived from P in Ψ.

We show the semantic rules that differ with the addition of priorities below. They are familiar from Section 1.3, except for two changes, the first being the addition of the HIGHEST side conditions. The second is a condition on the Com rule to ensure that the τ action is given the same priority as the input and output actions it is derived from.

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

$$\textsc{Par}\ \frac{\Psi_Q \otimes \Psi \rhd P \xrightarrow{\alpha} P'}{\Psi \rhd P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \begin{array}{l}\text{HIGHEST}(\alpha, \Psi, P \mid Q)\\ \text{bn}(\alpha)\#Q\end{array}$$

$$\textsc{Com}\ \frac{\begin{array}{c}\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K \qquad \Psi \vdash \texttt{prio}(M) = p\\ \Psi_Q \otimes \Psi \rhd P \xrightarrow{\overline{M}(\nu\widetilde{a})N} P'\\ \Psi_P \otimes \Psi \rhd Q \xrightarrow{K\,N} Q'\end{array}}{\Psi \rhd P \mid Q \xrightarrow{\tau:p} (\nu\widetilde{a})(P' \mid Q')}\ \begin{array}{l}\text{HIGHEST}(\tau : p, \Psi, P \mid Q)\\ \widetilde{a}\#Q\end{array}$$

Finally, we should mention that the choice of `has_prio` is subject to some natural constraints: channel equivalent terms must have the same unique priority level.

Since the semantics has been augmented with negative premises, we must show the existence of a stratification in order to guarantee that the transition relation is well-defined. A natural choice is to use the priority level of the action, which suffices.

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

Theorem 17. *All results about strong bisimulation and strong congruence described in Section 1.3 also apply to psi-calculi with priorities.*

We have also augmented broadcast psi-calculi with the same notion of priorities[4], and verified that Theorem 17 remains valid in such a setting. Timeouts then emerges as an interesting special case of reliable broadcasts with low priority. To see how this works, assume that there is a special reliable broadcast channel σ whose priority is lower than all other channels under consideration. A process which does the action α and proceeds as Q, unless a timeout occurs in which case it proceeds as Q, can be modelled as

$$\mathbf{case}\ \top : \alpha.P\ [] \top : \underline{\sigma}(\lambda\epsilon)\sigma.Q$$

It is assumed that this process occurs in a context where a timeout factory $!\overline{\sigma}\,\sigma$ occurs as a parallel component, and that $\top$ is a condition that is true in every assertion environment. As soon as no other synchronisation options are available, a timeout event is produced by the timeout factory, and propagated throughout the system via the reliable broadcast mechanism. As a case study, we have modelled the CAN bus arbitration protocol [ISO03] in psi-calculi using this timeout mechanism.

1.4.6 Bisimulation up-to techniques

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

$$\begin{array}{ccc} P & \mathcal{R} & Q \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle\alpha} \\ P' & \mathcal{R} & Q' \end{array}$$

[4]For details on how broadcast psi-calculi with priorities are defined, we refer to Chapter 5

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). Sangiorgi's bisimulation proof method [San98] is a systematic method for finding functions $\mathcal{F}$ such that the following diagram suffices:

$$\begin{array}{ccc} P & \mathcal{R} & Q \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle\alpha} \\ P' & \mathcal{F}(\mathcal{R}) & Q' \end{array}$$

This proof technique can significantly reduce the amount of work necessary in bisimulation proofs, since it allows known results about bisimilarity to be re-used when establishing the lower part, without including them in $\mathcal{R}$.

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.

We improve on this state of affairs by adopting the method of Sangiorgi, which uses a notion of *respectfulness* which guarantees soundness and is 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.

We show how Sangiorgi's method can be adapted to psi-calculi, where the main difference from previous work involves the treatment of assertions. We formalise all our definitions and theorems in Nominal Isabelle, and show examples where the use of up to-techniques has simplified our proofs of known results. We also prove a few new structural laws about the replication operator.

1.4.7 Summary of extensions

In Sections 1.4.1-1.4.5 above, we give the formal semantic rules covering the extensions of psi-calculi. The changes to the parameters, requisites and syntax, are often stated in an informal style, or in a formal style but integrated into the main body of text along with intuition and examples. In this section, we give the reader a summary of the formal definitions (not including semantics) of each extension presented in tabular form, and briefly discuss how the extensions can be combined.

Each of Tables 1.5-1.8 should be read as addendums to Tables 1.2 and 1.3. When the tables in this section define a symbol or process construct that already exists in the original psi-calculi, it should be read as a replacement rather than an addendum.

A natural question to ask when seeing the extensions side-by-side in this manner is: can we combine them? At a glance, it seems a simple matter to

Broadcast psi-calculi	
Parameters	Requisites
$\dot{\prec} : \mathbf{T} \times \mathbf{T} \Rightarrow \mathbf{C}$	$\dot{\prec}$ is equivariant.
	$M \dot{\prec} N \Rightarrow \mathsf{n}(M) \supseteq \mathsf{n}(N)$
$\dot{\succ} : \mathbf{T} \times \mathbf{T} \Rightarrow \mathbf{C}$	$\dot{\succ}$ is equivariant.
	$M \dot{\succ} N \Rightarrow \mathsf{n}(M) \subseteq \mathsf{n}(N)$
$\mathtt{reliable} : \mathbf{T} \Rightarrow \mathbf{C}$	`reliable` is equivariant.
Actions	Requisites
$!\overline{M}\,\widetilde{x}N$	$\widetilde{x}$ pair-wise distinct.
	$\widetilde{x} \subseteq \mathsf{n}(N)$ pair-wise distinct.
$?\underline{M}\,N$	
Process syntax	Well-formedness requisites
$!P$	P well-formed, P assertion guarded, P reliable reception guarded.

Table 1.5: Broadcast psi-calculi. A process is *reliable reception guarded* if each input is either guarded or its subject M satisfies $\forall \Psi, N, \sigma. \neg(\Psi \vdash N \dot{\succ} M\sigma \wedge \Psi \vdash \mathtt{reliable}(N))$.

amend psi-calculi with the contents of two or more tables at once, but formally this constitutes a change at the top of the proof tree. Hence, we must redo all proofs for every new combination of the extensions we consider.

Currently, the extensions that we have thus combined are higher-order psi-calculi with (unreliable) broadcast psi-calculi, and broadcast psi-calculi with priorities. In the first case, the effort involved was minimal thanks to our Isabelle proof repository; the main task involved is to textually combine the proof scripts, fill in a few small gaps manually and check that Isabelle can run the scripts to completion. I was able to completed this ordeal in a matter of hours. In the case of combining broadcast and priorities, simply textually combining the proof scripts would not suffice: the definition of HIGHEST must change to reflect the fact that broadcast output is non-blocking. Formally working out the implications of this change made this task a matter of a few weeks rather than a few hours.

Higher-order psi-calculi	
Parameters	Requisites
$\vdash \subseteq \mathbf{A} \times (\mathbf{C} \cup (\mathbf{T} \times \mathbf{P}))$	If $\Psi \vdash M \Leftarrow P$ then $\mathsf{n}(P) \subseteq \mathsf{n}(M)$. Otherwise as the original $\vdash$ operator.
Process syntax	
run M	

Table 1.6: Higher-order psi-calculi.

Psi-calculi with priorities	
Parameters	Requisites
$\mathtt{has\_prio} : \mathbf{T} \times \mathbb{N} \Rightarrow \mathbf{C}$	$\mathtt{has\_prio}$ equivariant. If $\Psi \vdash M \leftrightarrows N$ then there exists a unique p such that $\Psi \vdash \mathtt{has\_prio}(M, p)$ and $\Psi \vdash \mathtt{has\_prio}(N, p)$
Actions	
$\tau : n$	

Table 1.7: Psi-calculi with priorities.

Sorted psi-calculi with generalised pattern matching.	
Parameters	Requisites
$\mathcal{S}$	
$\mathcal{S}_\mathcal{N} \subseteq \mathcal{S}$	$\mathcal{S}_\mathcal{N}$ countable.
$\mathcal{S}_\nu \subseteq \mathcal{S}$	
$\underline{\propto} \subseteq \mathcal{S} \times \mathcal{S}$	
$\overline{\propto} \subseteq \mathcal{S} \times \mathcal{S}$	
$\prec \subseteq \mathcal{S} \times \mathcal{S}$	
$\mathbf{X}$	$\mathbf{X}$ finitely supported. $\mathbf{X}$ equipped with a substitution function as in Table 1.2.
$\text{SORT} : \mathcal{N} \uplus \mathbf{T} \uplus \mathbf{X} \Rightarrow \mathcal{S}$	SORT equivariant. $\text{SORT}(a) = s$ iff $a \in \mathcal{N}_s$. $\text{SORT}(M\sigma) \leq \text{SORT}(M)$.
$\text{VARS} : \mathbf{X} \Rightarrow \mathcal{P}(\mathcal{P}(\mathcal{N}))$	VARS equivariant. If $\widetilde{x} \in \text{VARS}(X)$ and $\widetilde{x}\#\sigma$ then $X\sigma \in \mathbf{X}$ and $\text{SORT}(X\sigma) \leq \text{SORT}(X)$ and $\widetilde{x} \in \text{VARS}(X\sigma)$.
$\text{MATCH} : \mathbf{T} \times \mathcal{N}^* \times \mathbf{X} \Rightarrow \mathcal{P}(\mathbf{T}^*)$	MATCH equivariant. If $\widetilde{x} \in v(X)$ are distinct and $\widetilde{N} \in \text{MATCH}(M, \widetilde{x}, X)$ then it must hold that $\text{SORT}(\widetilde{x}) \prec \text{SORT}(\widetilde{N})$, that $\lvert\widetilde{x}\rvert = \lvert\widetilde{N}\rvert$ that $\mathsf{n}(()\widetilde{N}) \subseteq \mathsf{n}(()M) \cup (\mathsf{n}(()X) \setminus \widetilde{x})$, and that for all name swappings $(\widetilde{x}\ \widetilde{y})$ we have $\widetilde{N} \in \text{MATCH}(M, \widetilde{y}, (\widetilde{x}\ \widetilde{y})X)$.
Process syntax	Well-formedness requisites
$\overline{M}\,N\,.\,P$	P well-formed, $\text{SORT}(M)\ \overline{\propto}\ \text{SORT}(N)$.
$\underline{M}(\lambda\widetilde{x})X\,.\,P$	P well-formed, $\text{SORT}(M)\ \underline{\propto}\ \text{SORT}(X)$, $\widetilde{x}$ distinct and $\widetilde{x} \in \text{VARS}(N)$.
$(\nu x)P$	P well-formed, $\text{SORT}(x) \in \mathcal{S}_\nu$

Table 1.8: Sorted psi-calculi with generalised pattern matching. $s_1 \leq s_2$ holds iff $\forall t \in \mathcal{S}.(s_2 \underline{\propto} t \Rightarrow s_1 \underline{\propto} t) \wedge (s_2 \overline{\propto} t \Rightarrow s_1 \overline{\propto} t) \wedge (t \underline{\propto} s_2 \Rightarrow t \underline{\propto} s_1) \wedge (t \overline{\propto} s_2 \Rightarrow t \overline{\propto} s_1)$.

1.5 Conclusion

We have seen how the psi-calculi framework can be extended with several advanced language features in order to broaden its scope of applicability, without sacrificing the framework's generality or its associated machine-checked metatheory. In this section, we conclude by evaluating the work, and discussing related work, impact and future work.

1.5.1 Evaluation

There are three main aims of the psi-calculi program:

1. Create a versatile framework for applied process calculi that can express a wide range of phenomena.

2. While doing so, maintain a "pure" theory with simple and clean definitions.

3. Support this by machine-checked proofs, for high confidence in important results.

On the first count, the work in this thesis significantly extends the range of phenomena captured by psi-calculi through the incorporation of several advanced language features.

On the second count, we preserve the purity of psi-calculi with the higher-order, (unreliable) broadcast, sorted and pattern matching extensions; while we introduce additional rules, parameters and syntax to accommodate the extensions, at its core the framework remains defined by a single inductive definition each for the syntax and semantics. We do however sacrifice some simplicity with the introduction of negative premises for reliable broadcasts and priorities, which entails that our transition relation cannot be read as an inductive definition in the usual sense. The alternatives to negative premises for these purposes that can be found in the literature, namely auxiliary relations such as initial action sets [CLN07], involve another sacrifice in that they introduce another layer into the operational semantics. We find the formulation in terms of negative premises to be more natural and clean.

On the third count, we provide machine-checked proofs for the results presented in this thesis, with two main exceptions: when there are technical limitations in the tools we use, and when treating correspondence results with other calculi, where we would also need to mechanise those — a significant additional effort that hardly seems motivated. An obvious benefit of this is near absolute confidence in the correctness of our results. The fact that we are able to reuse the work of Bengtson [Ben10] on the original psi-calculi formalisation throughout this thesis demonstrate an advantage of investing in a formal proof repository: it is easy to assess the impact of changes to the definitions without resorting to hand-waving.

Process calculi is a somewhat balkanised field of research, with countless competing formalisms on the market. Psi-calculi has the potential to act as a unifying framework for many of these formalisms, and we strengthen this potential not only by incorporating new language features, but also by providing several examples of existing calculi that now fall under the psi umbrella.

However, it is ironic that in order to achieve these unifying contributions, this thesis has arguably introduced a manner of balkanisation on the level of frameworks: there is now not just one psi-calculi framework, but a family of them. This presents interesting challenges when maintaining the formal proof repository, which has now been forked into many different branches and grown somewhat unwieldy. For comparison, the psi-calculi entry into the Archive of Formal Proofs currently constitutes 32294 lines of Isabelle code [Ben12], while the proof scripts for this thesis constitute 319522 lines of code [ÅP13]. While we incorporate much of his code and acknowledge that we stand on his shoulders, we believe we are justified in laying claim to Bengtson's assertion that

> this thesis contains the most extensive formalisations of process calculi ever done in a theorem prover.

1.5.2 Related work

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 stemming from CCS [Mil80], CSP [BHR84] and ACP [BK84]; ours is the CCS branch. For a historical overview we refer to Baeten [Bae05].

The psi-calculi framework has its roots in Milner, Parrow and Walker's pi-calculus [MPW92], 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 type of message term that can be exchanged is 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*, a substitution waiting to be applied that exist as part of the process environment much like

assertions do in psi-calculi. Message terms in the applied pi-calculus can be composed with 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].

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. Semantics and bisimulation is 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 are identified, and are in fact syntactically equal in their "raw" form without the need for any quotient constructs. 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. Sangiorgi's bisimulation proof method is 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 calcu-

lus in Coq. Since the thesis considers several spi calculus dialects and Briais strives to give them a uniform treatment, the formalisation is in fact a kind of parametric framework for spi calculi, parametrised on the type and behaviour of guards and actions. Unlike psi-calculi, instantiating the framework does not yield any results about bisimulation. Aside from technical lemmas and definitions, the only generic result appears to be that structural congruence is preserved by reduction for instantiations of the framework that "satisfy some conditions" (it is not made explicit which conditions). The other main result is derived separately for every instantiation of the framework, and show 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.

Finally, 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.

Broadcasts

Calculi with global synchronisation mechanisms go back to the 1980's. CSP [BHR84], due to Brookes, Hoare and Roscoe, features an *interface parallel* operator $\|$ such that $P \| Q$ 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. Broadcasts are reliable, and Prasad expresses this by introducing explicit "lose" actions to denote non-reception of a message, as a means of encoding negative premises.

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, can not 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. Unfortunately, commuting binders is then unsound with regards to labelled bisimilarity, in contradiction to [Ene01][Lemma 34]: consider the processes $P = \nu x \nu y \overline{a}\,\langle x, y\rangle$ and $Q = \nu y \nu x \overline{a}\,\langle x, y\rangle$. The only available transitions (up to alpha-equivalence) are $P \xrightarrow{\nu x \nu y \overline{a}\langle x,y\rangle} \mathbf{0}$ and $Q \xrightarrow{\nu y \nu x \overline{a}\langle x,y\rangle} \mathbf{0}$, respectively; hence they are not bisimilar since the labels are different. Two possible fixes are to either modify [Ene01][Rule (4), Table 3.5] so that the binder on the label in the conclusion can be inserted anywhere in the sequence, or to treat the binding sequence in bound output labels as a set rather than a sequence.

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, MM12, CHM12, GY09, VNN13]. A thorough comparison of broadcast psi-calculi and many of these calculi can be found in Chapter 2; 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.

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.

Finally, we mention the Secure Broadcast Ambients of Gunter and Yasmeen [GY09], where ambients [CG98] are combined with broadcast communication in order 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.

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 [San93b], 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 introduce a feature called *passivation* [SS04], 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]; 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 [SS09], which like the first-order applied pi-calculus is parametrised on a term algebra and a corresponding term rewriting system for messages. 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 in-

terplay 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.

Sorts

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 a general framework 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.

Pattern matching

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 1.4.3 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 occurrences of the same variable to the right. 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. However, the construct $p\%x$ binds x 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 $\Theta \in \mathtt{match}(\xi, M)$. 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.

Priorities

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 transitions 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, the 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 overlapping 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 $\pi@$.

1.5.3 Impact

The original psi-calculi framework made 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 $b\pi$ and the pi-calculus with polyadic synchronisation, now enjoy significantly higher confidence in the correctness of their meta-theoretical results thanks to our formal proofs.

1.5.4 Future Work

Parametric sorts in Nominal Isabelle

As mentioned in Section 1.4.4, the scheme from sorted psi-calculi where we treat the sets of sorts as a parameter is currently not amenable to formalisation in Nominal Isabelle. While it's possible to have more than one name sort, they must be individually declared in advance using the `atom_decl` command. Doing so is clearly not feasible when the set of name sorts under consideration is not known, and in fact not even necessarily finite.

Our current approach is to implement only the trivially sorted case in Nominal Isabelle, and use hand-written proofs to lift the results to many-sorted psi-calculi. This is unfortunate in two ways: trust in the results is reduced, and a gap between our formalisation and our work as presented on paper is introduced.

In order to remedy this, we are currently working on a modified version of Nominal Isabelle where our parametric sorting scheme can be expressed. The idea is simply to tag every name with a natural number, representing its sort. Our hope is to then recover Urban and Kaliszyk's infrastructure for reasoning about syntax with binders [UK12].

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.

Weak bisimulation has been successfully applied to calculi with priorities, though some care must be taken in order to obtain a relation which is closed under parallel composition (see eg. [CLN07]).

In a calculus with broadcasts, weak bisimulation (at least as it is usually defined) is not the correct choice if an observational equivalence is desired. Ene and Muntean argue this point in [EM02], championing instead the use of testing preorders. The motivating example they give (translated to our syntax) is that bisimulation will distinguish between

$$P = \overline{a}\,a.(\mathbf{case}\ \top : \overline{b}\,b.\mathbf{0}\ [\!]\ \top : \overline{c}\,c.\mathbf{0})$$

and

$$Q = \mathbf{case}\ \top : \overline{a}\,a.\overline{b}\,b.\mathbf{0}\ [\!]\ \top : \overline{a}\,a.\overline{c}\,c.\mathbf{0}$$

where the condition $\top$ is entailed by all assertions. If the outputs are point-to-point outputs, an observer can indeed distinguish P from Q; if the outputs are broadcasts, no observer can do so. The reason is that broadcast outputs are non-blocking, and hence the evolution of P and Q do not depend on the behaviour of an outside observer. With this example in mind, it might be more worthwhile to focus on developing theory of testing rather than a theory of weak bisimulation for broadcast psi-calculi.

Psi-calculi as a framework for protocol verification in Isabelle

In Chapter 2, we apply broadcast psi-calculi to verify a basic reachability property of the LUNAR protocol for ad-hoc routing in wireless networks [TGRW04]. 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 (ie. 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 psi-calculi as a framework for protocol verification in Isabelle, and seeing how it compares to eg. Paulson's inductive approach [Pau98].

On a related note, ongoing work by Raabjerg applies broadcast psi-calculi with priorities to the field of cache coherence protocols for multicore CPU architectures, with similar aims.

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 and 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 two examples of unary functors: the functor U used in the meta-theoretical proofs in Chapter 4, and the functor $\mathcal{H}$ that maps a psi-calculus to its canonical higher-order extension from Chapter 3. 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.

Integrating extensions

More work needs to be done to integrate the extensions of psi-calculi presented in this thesis, so that they can be used in the same calculus. So far we have successfully integrated (unreliable) broadcast with higher-order psi-calculi, sorted with pattern matching psi-calculi, and broadcast with priorities. By integrated, we mean that we have combined the features that constitute the extensions into one extension, and verified that the meta-theory carries over. Raabjerg is currently working on higher-order broadcast psi-calculi with priorities.

A beautiful way to generalise most of the contents of this licentiate thesis in one fell swoop would be to develop a general theory of extensions for psi-calculi, that could answer the following questions: what properties must an extension of psi-calculi satisfy in order for it to preserve the standard meta-theoretical results? What properties must it satisfy in order to smoothly combine with other extensions?

Similar questions have been studied in the field of rule formats. For an example, Groote and Vaandrager study a composition operator $\oplus$ over transition system specifications, and study what requisites on its arguments must be imposed in order for $S_0 \oplus S_1$ to be a conservative extension of S_0 [GV89]. We believe that if the problem of a general theory of extensions is approachable at all, ideas from the field of rule formats will be crucial.

Chapter 2

Broadcast psi-calculi

Paper I

Broadcast Psi-calculi

with an Application to Wireless Protocols

Johannes Borgström[1], Shuqin Huang[2], Magnus Johansson[1], Palle Raabjerg[1], Björn Victor[1], Johannes Åman Pohjola[1], Joachim Parrow[1]

[1] Department of Information Technology, Uppsala University, Sweden
[2] Peking University, China

September 2, 2013

Abstract Psi-calculi is a parametric framework for extensions of the pi-calculus, with arbitrary data structures and logical assertions for facts about data. In this paper we add primitives for broadcast communication in order to model wireless protocols. The additions preserve the purity of the psi-calculi semantics, and we formally prove the standard congruence and structural properties of bisimilarity. We demonstrate the expressive power of broadcast psi-calculi by modelling the wireless ad-hoc routing protocol LUNAR and verifying a basic reachability property.

1 Introduction

Psi-calculi is a parametric framework for extensions of the pi-calculus, with arbitrary data structures and logical assertions for facts about data. In earlier papers we have shown how psi-calculi can capture the same phenomena as other proposed extensions of the pi-calculus such as the applied pi-calculus, the spi-calculus, the fusion calculus, the concurrent constraint pi-calculus, and calculi with polyadic communication channels or pattern matching. Psi-calculi can be even more general, for example by allowing structured channels, higher-order formalisms such as the lambda calculus for data structures, and predicate logic for assertions [6].

In psi-calculi (described in Section 2) the purity of the semantics is on par with the original pi-calculus, the generality and expressiveness exceeds many earlier extensions of the pi-calculus, and the meta-theory is proved correct once and for all using the interactive theorem prover Isabelle/Nominal [34]. The communication paradigm in psi-calculi is binary: for each event there is one sender and one receiver, just as in the pi-calculus.

In several areas, e.g. wireless communications and hardware data buses, a natural paradigm is broadcast, where one transmission can be received by several processes. Broadcast communication cannot be uniformly encoded in the pi-calculus [8].

In this paper we extend the psi-calculi framework with primitives for synchronous unreliable broadcast. These require new operational actions and rules, and new connectivity predicates. In Section 3.1, we formally prove the congruence properties of bisimilarity and the soundness of structural equivalence laws using the Isabelle/Nominal theorem prover.

The connectivity predicates allow us to model systems with limited reachability, for instance where a transmitter only reaches nodes within a certain range, and systems with changing reachability, for instance due to physical mobility of nodes. In Section 4, we present a technique for treating different generations of connectivity information. Broadcast channels can be globally visible or have limited scope. Scoped channels can be protected from externally imposed connectivity changes, while permitting connectivity changes by processes within the scope of the channel. One of our main contributions is precise requirements that the connectivity predicates must satisfy, in order to model scoped broadcasts with dynamic connectivity, while still satisfying the meta-theoretical results of Section 3.1.

We demonstrate the expressive power of the resulting framework in Section 5, where we provide a model of the LUNAR protocol for routing in ad-hoc wireless networks [32]. The model follows the specification closely, and demonstrates several features of the psi-calculi framework: both unicast and broadcast communication, application-specific data structures and logics, classic unstructured channels as well as pairs corresponding to MAC address and port selector. Our model is significantly more succinct than earlier work [36,35] (ca 30 vs 250 lines). We show an expected basic reachability property of the model: if two network nodes, a sender and a receiver, are both in range of a third node, but not within range of each other, the LUNAR protocol can find a route and transparently handle the delivery of a packet from the sender to the receiver.

We discuss related work on process calculi for wireless broadcast in Section 6, and conclude and present ideas for future work in Section 7.

This paper is an extended version of [7] that adds clarifications, proofs, and elaborated examples of dynamic topology management.

2 Psi-calculi

This section is a brief recapitulation of psi-calculi; for an extensive treatment including more motivations and examples see [5,6], from which some examples and explanations below are taken.

We assume a countably infinite set of atomic *names* $\mathcal{N}$ ranged over by $a, b, \ldots, z$. Intuitively, names will represent the symbols that can be scoped, and also represent symbols acting as variables in the sense that they can

be subject to substitution. As a general framework for terms and other data containing names, we work in the formalism of *nominal sets* [25,9]. A nominal set is an ordinary set equipped with a formal notion of what it means for a name a to occur in an element X of the set, written $a \in \mathrm{n}(X)$ (often pronounced as "a is in the support of X"). We write $a\#X$, pronounced "a is fresh for X", for $a \notin \mathrm{n}(X)$, and if A is a finite set of names we write $A\#X$ to mean $\forall a \in A \,.\, a\#X$. We require all elements to have finite support, i.e., $\mathrm{n}(X)$ is finite for all X. In the following $\tilde{a}$ means a finite sequence of names, $a_1, \ldots, a_n$. The empty sequence is written ϵ and the concatenation of $\tilde{a}$ and $\tilde{b}$ is written $\tilde{a}\tilde{b}$. When occurring as an operand of a set operator, $\tilde{a}$ means the corresponding set of names $\{a_1, \ldots, a_n\}$. We also use sequences of other nominal sets in the same way. For names, we write $(\widetilde{a}\ \widetilde{b})$ for the *name swapping* that swaps each element of $\widetilde{a}$ with the corresponding element of $\widetilde{b}$; here it is implicit that $\widetilde{a}$ and $\widetilde{b}$ have the same length, and that the names in $\widetilde{a}$ (resp. $\widetilde{b}$) are pair-wise distinct. A function f is *equivariant* if $(a\ b) \cdot f(X) = f((a\ b) \cdot X)$ holds for all X, and similarly for functions and relations of any arity. Intuitively, equivariance means that all names are treated equally.

A *nominal datatype* is a nominal set together with a set of functions on it. In particular we shall consider substitution functions that substitute elements for names. If X is an element of a datatype, $\tilde{a}$ is a sequence of names without duplicates and $\tilde{Y}$ is an equally long sequence of elements of possibly another datatype, the *substitution* $X[\tilde{a} := \tilde{Y}]$ is an element of the same datatype as X. Substitution is required to satisfy a law akin to alpha-conversion: if $\widetilde{b}\#X, \widetilde{a}$ then $X[\widetilde{a} := \widetilde{T}] = ((\widetilde{b}\ \widetilde{a}) \cdot X)[\widetilde{b} := \widetilde{T}]$. Intuitively, this ensures that substitutions for bound names yield the same result no matter which alpha-equivalent version is used.

We use nominal datatypes in order to obtain a general framework, allowing many different instantiations. Our only requirements are on the notions of support, name swapping, and substitution. Thus we can handle datatypes that are not inductively defined, such as equivalence classes or sets defined by comprehension or co-induction. Examples include higher-order datatypes such as the lambda calculus. As long as the term language satisfies the axioms of a nominal datatype, it can be used in our framework. Similarly, the notions of conditions, i.e., the tests on data that agents can perform during their execution, and assertions, i.e., the facts that can be used to resolve conditions, are formulated as nominal datatypes. This means that logics with binders and even higher-order logics can be used. Moreover, alpha-variants of terms can be formally equated by taking the quotient of terms under alpha equality, thereby facilitating the formalism and proofs.

A psi-calculus is defined by instantiating three nominal data types and four operators:

Definition 1 (Psi-calculus parameters) *A psi-calculus requires the three (not necessarily disjoint) nominal data types: the (data) terms* $\mathbf{T}$*, ranged over by* M, N*, the conditions* $\mathbf{C}$*, ranged over by* φ*, the assertions* $\mathbf{A}$*, ranged*

over by Ψ, and the four equivariant operators:

$$\begin{array}{ll} \leftrightarrow : \mathbf{T} \times \mathbf{T} \to \mathbf{C} & \textit{Channel Equivalence} \\ \otimes : \mathbf{A} \times \mathbf{A} \to \mathbf{A} & \textit{Composition} \\ \mathbf{1} : \mathbf{A} & \textit{Unit} \\ \vdash \subseteq \mathbf{A} \times \mathbf{C} & \textit{Entailment} \end{array}$$

and substitution functions $[\widetilde{a} := \widetilde{M}]$, substituting terms for names, on each of $\mathbf{T}$, $\mathbf{C}$ and $\mathbf{A}$, where the substitution function on $\mathbf{T}$, in addition to the alpha-conversion-like law above, satisfies the following name preservation law: if $\widetilde{a} \subseteq \mathsf{n}(M)$ and $b \in \mathsf{n}(\widetilde{N})$ then $b \in \mathsf{n}(M[\widetilde{a} := \widetilde{N}])$.

The binary functions above will be written in infix. Thus, if M and N are terms then $M \leftrightarrow N$ is a condition, pronounced "M and N are channel equivalent" and if Ψ and Ψ' are assertions then so is $\Psi \otimes \Psi'$. Also we write $\Psi \vdash \varphi$, "Ψ entails φ", for $(\Psi, \varphi) \in \vdash$.

As an example, we can choose data terms inductively generated by some signature, assertions and conditions to be elements of a first-order logic with equality over these terms, entailment to be logical implication, $\otimes$ to be conjunction and $\mathbf{1}$ to be TRUE. We call this example instance **euf**.

We say that two assertions are equivalent, written $\Psi \simeq \Psi'$, if they entail the same conditions, i.e. for all φ we have that $\Psi \vdash \varphi \Leftrightarrow \Psi' \vdash \varphi$. We impose certain straightforward requisites on the sets and operators. In brief, channel equivalence must be symmetric and transitive (but not necessarily reflexive), $\otimes$ must be compositional with regard to $\simeq$, and the assertions with $(\otimes, \mathbf{1})$ form an abelian monoid modulo $\simeq$. In the **euf** instance we can let channel equivalence be term equality: symmetry and reflexivity clearly hold, logical conjunction does form an abelian monoid with TRUE as unit, and compositionality of assertion composition follows from the tautology $(\Psi \Rightarrow (\Psi_1 \Leftrightarrow \Psi_2)) \Rightarrow (\Psi \Rightarrow (\Psi' \wedge \Psi_1 \Leftrightarrow \Psi' \wedge \Psi_2))$. For details see [6].

A *frame* F can intuitively be thought of as an assertion with local names: it is of the form $(\nu\widetilde{b})\Psi$ where $\widetilde{b}$ is a sequence of names that bind into the assertion Ψ. We use F, G to range over frames. We overload Ψ to also mean the frame $(\nu\epsilon)\Psi$ and $\otimes$ to composition on frames defined by $(\nu\widetilde{b}_1)\Psi_1 \otimes (\nu\widetilde{b}_2)\Psi_2 = (\nu\widetilde{b}_1\widetilde{b}_2)(\Psi_1 \otimes \Psi_2)$ where $\widetilde{b}_1 \# \widetilde{b}_2, \Psi_2$ and vice versa. We write $\Psi \otimes F$ to mean $(\nu\epsilon)\Psi \otimes F$, and $(\nu c)((\nu\widetilde{b})\Psi)$ for $(\nu c\widetilde{b})\Psi$.

Alpha equivalent frames are identified. We define $F \vdash \varphi$ to mean that there exists an alpha variant $(\nu\widetilde{b})\Psi$ of F such that $\widetilde{b} \# \varphi$ and $\Psi \vdash \varphi$. We also define $F \simeq G$ to mean that for all φ it holds that $F \vdash \varphi$ iff $G \vdash \varphi$. Intuitively a condition is entailed by a frame if it is entailed by the assertion and does not contain any names bound by the frame, and two frames are equivalent if they entail the same conditions.

In the **euf** example, assume that the term $\mathsf{enc}(M, k)$ represents the encoding of message M with key k, and let Ψ be the assertion $C = \mathsf{enc}(M, k)$, stating that the ciphertext C is the result of encoding M by k. If an agent contains this assertion, the environment of the agent will be able to use it to resolve tests on the data. In particular it may infer that $C = \mathsf{enc}(M, k)$,

i.e., it can test if this C is the encryption of M. Access to the key k can be restricted by enclosing it in a scope: if the environment instead has access to the assertion $(\nu k)\Psi$, it can *not* infer that C is the encoding of M (assuming conditions only contain equality tests on terms, and no quantifiers). For more discussion see [6].

Definition 2 (Psi-calculus agents) *Given valid psi-calculus parameters as in Definition 1, the psi-calculus* agents*, ranged over by $P, Q, \ldots$, are of the following forms.*

$\mathbf{0}$	Nil
$\overline{M}N\,.\,P$	Output
$\underline{M}(\lambda\widetilde{x})N\,.\,P$	Input
$\mathbf{case}\ \varphi_1 : P_1 \,[]\, \cdots \,[]\, \varphi_n : P_n$	Case
$(\nu a)P$	Restriction
$P \mid Q$	Parallel
$!P$	Replication
$(\!\|\Psi\|\!)$	Assertion

Restriction binds a in P and Input binds $\widetilde{x}$ in both N and P. We identify alpha equivalent agents. An assertion is guarded *if it is a subterm of an Input or Output. An agent is* assertion guarded *if it contains no unguarded assertions. An agent is* well-formed *if in $\underline{M}(\lambda\widetilde{x})N.P$ it holds that $\widetilde{x} \subseteq \mathrm{n}(N)$ is a sequence without duplicates, that in a replication $!P$ the agent P is assertion guarded, and that in $\mathbf{case}\ \varphi_1 : P_1 \,[]\, \cdots \,[]\, \varphi_n : P_n$ the agents P_i are assertion guarded.*

In the Output and Input forms M is called the subject and N the object. Output and Input are similar to those in the pi-calculus, but arbitrary terms can function as both subjects and objects. In the input $\underline{M}(\lambda\widetilde{x})N.P$ the intuition is that the pattern $(\lambda\widetilde{x})N$ can match any term obtained by instantiating $\widetilde{x}$, e.g., $\underline{M}(\lambda x, y)f(x, y).P$ can only communicate with an output $\overline{M}f(N_1, N_2)$ for some data terms N_1, N_2. This can be thought of as a generalisation of the polyadic pi-calculus where the patterns are just tuples of names. Another significant extension is that we allow arbitrary data terms also as communication channels. Thus it is possible to include functions that create channels.

The **case** construct behaves as one of the P_i for which the corresponding φ_i is true. The agent $\mathbf{case}\ \varphi_1 : P_1 \,[]\, \cdots \,[]\, \varphi_n : P_n$ is sometimes abbreviated as $\mathbf{case}\ \widetilde{\varphi} : \widetilde{P}$, or if $n = 1$ as $\mathbf{if}\ \varphi_1\ \mathbf{then}\ P_1$. Input subjects are underlined to facilitate parsing of complicated expressions; in simple cases we often omit the underline. We sometimes write $\underline{M}(x).P$ for $\underline{M}(\lambda x)x.P$.

One of the simplest examples of a psi-calculus is the pi-calculus [22], which can be represented using names as the only data terms, $\mathbf{1}$ as the only assertion, and equality tests on names as conditions. Channel equivalence $\leftrightarrows$ is also equality on names. Substitution is the standard syntactic replacement

of names for names. Choice in the pi-calculus can be represented using the **case** statement: $P + Q$ corresponds to $(\nu a)(\mathbf{case}\ a = a\ :\ P\ []\ a = a\ :\ Q)$, where $a\#P, Q$, and the pi-calculus match construct $[a = b]P$ corresponds to **if** $a = b$ **then** P. The formal correspondence between this psi-calculus instance and the original pi-calculus is proved in [6].

As indicated in the encryption example above, the conditions tested in a process are affected by the assertions of parallel processes. For example in $P \mid Q$, the assertions of P can affect the conditions tested in Q, and thereby its transitions. We introduce the *frame of an agent* as the combination of its top level assertions, retaining all the binders: this is precisely what can affect a parallel agent. The *frame* $\mathcal{F}(P)$ *of an agent* P is defined inductively as follows:

$$\mathcal{F}(\underline{M}(\lambda\widetilde{x})N\,.\,P) = \mathcal{F}(\overline{M}\,N\,.\,P) = \mathcal{F}(\mathbf{0}) = \mathcal{F}(\mathbf{case}\ \widetilde{\varphi} : \widetilde{P}) = \mathcal{F}(!P) = \mathbf{1}$$
$$\mathcal{F}((\!|\Psi|\!)) = (\nu\epsilon)\Psi$$
$$\mathcal{F}(P \mid Q) = \mathcal{F}(P) \otimes \mathcal{F}(Q)$$
$$\mathcal{F}((\nu b)P) = (\nu b)\mathcal{F}(P)$$

For a simple example, if $a\#\Psi_1$:

$$\mathcal{F}((\!|\Psi_1|\!) \mid (\nu a)((\!|\Psi_2|\!) \mid \overline{M}\,N.(\!|\Psi_3|\!)) \quad = \quad (\nu a)(\Psi_1 \otimes \Psi_2)$$

Here Ψ_3 occurs under a prefix and is therefore not included in the frame.

The *actions* ranged over by α, β are of the following three kinds: Output $\overline{M}(\nu\tilde{a})N$ where $\alpha \subseteq \mathrm{n}(N)$, Input $\underline{M}\,N$, and Silent τ. Here we refer to M as the *subject* and N as the *object*. We define $\mathrm{bn}(\overline{M}(\nu\tilde{a})N) = \tilde{a}$, and $\mathrm{bn}(\alpha) = \emptyset$ if α is an input or τ. We also define $\mathrm{n}(\tau) = \emptyset$ and $\mathrm{n}(\alpha) = \mathrm{n}(M) \cup \mathrm{n}(N)$ for the input and output actions. As in the pi-calculus, the output $\overline{M}(\nu\tilde{a})N$ represents an action sending N along M and opening the scopes of the names $\tilde{a}$. Note in particular that the support of this action includes $\tilde{a}$. Thus $\overline{M}(\nu a)a$ and $\overline{M}(\nu b)b$ are different actions.

Definition 3 (Transitions)

A transition *is written* $\Psi \rhd P \xrightarrow{\alpha} P'$*, meaning that in the environment* Ψ *the well-formed agent* P *can do an* α *to become* P'*. The transitions are defined inductively in Table 1. We write* $P \xrightarrow{\alpha} P'$ *without an assertion to mean* $\mathbf{1} \rhd P \xrightarrow{\alpha} P'$*.*

Agents, frames and transitions are identified by alpha equivalence. In a transition the names in $\mathrm{bn}(\alpha)$ bind into both the action object and the derivative, therefore $\mathrm{bn}(\alpha)$ is in the support of α but not in the support of the transition. This means that the bound names can be chosen fresh, substituting each occurrence in both the object and the derivative.

The environmental assertions $\Psi \rhd \cdots$ in Table 1 express the effect that the environment has on the agent: enabling conditions in Case, giving rise to action subjects in In and Out and enabling interactions in Com. The environment Ψ increases towards the leaves of the derivation only in the

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

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

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

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

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

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

Table 1 Structured operational semantics. Symmetric versions of Com and Par are elided. In the rule Com we assume that $\mathcal{F}(P) = (\nu\widetilde{b}_P)\Psi_P$ and $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$ where $\widetilde{b}_P$ is fresh for all of $\Psi, \widetilde{b}_Q, Q, M$ and P, and that $\widetilde{b}_Q$ is similarly fresh. In the rule Par we assume that $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$ 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.

rules for the parallel operator, where an agent is part of the environment for another agent. If all environmental assertions are erased and channel equivalence replaced by identity we get the standard laws of the pi-calculus enriched with data structures.

For a simple example of a transition, suppose for an assertion Ψ and condition φ that $\Psi \vdash \varphi$. Assume that

$$\forall\Psi'.\Psi' \rhd Q \xrightarrow{\alpha} Q'$$

i.e., Q has an action α regardless of the environment. Then by the Case rule we get

$$\Psi \rhd \mathbf{if}\ \varphi\ \mathbf{then}\ Q \xrightarrow{\alpha} Q'$$

i.e., **if** φ **then** Q has the same transition if the environment is Ψ. Since $\mathcal{F}(\lVert\Psi\rVert) = \Psi$ and $\Psi \otimes \mathbf{1} = \Psi$, if $\mathrm{bn}(\alpha)\#\Psi$ we get by Par that

$$\mathbf{1} \rhd \lVert\Psi\rVert \mid \mathbf{if}\ \varphi\ \mathbf{then}\ Q \xrightarrow{\alpha} \lVert\Psi\rVert \mid Q'$$

The notion of strong bisimulation is used to formalise the intuition that two agents "behave in the same way".

Definition 4 (Strong bisimulation) *A* strong bisimulation $\mathcal{R}$ *is a ternary relation on assertions and pairs of agents such that* $\mathcal{R}(\Psi, P, Q)$ *implies*

1. *Static equivalence:* $\Psi \otimes \mathcal{F}(P) \simeq \Psi \otimes \mathcal{F}(Q)$*; and*
2. *Symmetry:* $\mathcal{R}(\Psi, Q, P)$*; and*
3. *Extension of arbitrary assertion:* $\forall \Psi'.\ \mathcal{R}(\Psi \otimes \Psi', P, Q)$*; and*
4. *Simulation: for all* α, P' *such that* $\Psi \rhd P \xrightarrow{\alpha} P'$ *and* $\mathrm{bn}(\alpha)\#\Psi, Q$*, there exists* Q' *such that* $\Psi \rhd Q \xrightarrow{\alpha} Q'$ *and* $\mathcal{R}(\Psi, P', Q')$*.*

We define $P \mathrel{\dot{\sim}_\Psi} Q$ *to mean that there exists a bisimulation* $\mathcal{R}$ *such that* $\mathcal{R}(\Psi, P, Q)$*, and write* $\dot{\sim}$ *for* $\dot{\sim}_{\mathbf{1}}$*.*

Strong bisimulation is a congruence in the usual sense: it is preserved by all operators except input prefix, and satisfies the expected algebraic laws such as scope extension $P \mid (\nu a)Q \mathrel{\dot{\sim}} (\nu a)(P \mid Q)$ if $a\#P$. For details see [5,6]. Note that these meta-theoretic results have been proven to hold for all psi-calculus instances using the interactive theorem prover Isabelle/Nominal [34].

Psi-calculi can capture the same phenomena as a wide range of previously proposed individual extensions of the pi-calculus. Examples in [5,6] range from foundational calculi such as polyadic pi-calculus, polyadic synchronisation pi-calculus, fusion calculus, and concurrent constraint calculi, to applied calculi for cryptography and systems with frequency hopping communication protocols. Each previous pi-calculus extension in the literature has needed new proofs of basic results such as scope extension and bisimulation congruence. Instead, formulated as psi-calculus instances, all the meta-theory of psi-calculi is automatically inherited.

3 Broadcast psi-calculi

In this section we extend the unicast psi-calculi of the previous section with a communication paradigm for synchronous unreliable non-blocking broadcast (suitable for modelling wireless communication). We introduce the notion of a *broadcast channel* as an abstraction of relevant properties of the transmission, such as frequency, sender location and signal strength. Formally a broadcast channel is just a term. We assume so called *connectivity predicates* that regulate which prefix subjects can send on or receive from which broadcast channels. These predicates may depend on assertions and therefore change as an agent evolves.

As an example, assume that the connectivity information Ψ allows the sender M_0 to send on the broadcast channel K, and receivers M_1 and M_2 to listen on K. We would then have the following transition:

$$\Psi \rhd \overline{M_0}\, N.P \mid \underline{M_1}(x).Q \mid \underline{M_2}(y).R \xrightarrow{\overline{!K}\, N} P \mid Q[x := N] \mid R[y := N]$$

Here, in one action two processes both receive the N sent along K, and moreover the action label retains the broadcast output action $!\overline{K}N$, meaning that in a larger context even more processes could receive N.

Formally, we assume a psi-calculus with the following extra predicates:

Definition 5 (Extra predicates for broadcast)

$$\dot{\prec} : \mathbf{T} \times \mathbf{T} \to \mathbf{C} \qquad \textit{Output Connectivity}$$
$$\dot{\succ} : \mathbf{T} \times \mathbf{T} \to \mathbf{C} \qquad \textit{Input Connectivity}$$

The first predicate, $M \mathrel{\dot{\prec}} K$, is pronounced "$M$ is out-connected to K" and means that an output prefix $\overline{M}\,N$ can result in a broadcast on channel K. The second, $K \mathrel{\dot{\succ}} M$, is pronounced "$M$ is in-connected to K" and means that an input prefix $\underline{M}(\lambda\tilde{x})N$ can receive broadcast messages from channel K. As usual in broadcast calculi, the receivers need to be using the same broadcast channel as the sender in order to receive a message.

As an example, we can model lookup in a routing table: if the term tab is a list of pairs of identifiers and channels we can let $\Psi \vdash \mathtt{lookup}(tab, id) \mathrel{\dot{\prec}} ch$ be true iff (id, ch) appears in the routing table tab. We can also model connectivity: if Ψ contains connectivity information between channels ch and receivers n we may let $\Psi \vdash ch \mathrel{\dot{\succ}} \mathtt{rcv}(n, ch)$ be true if n is connected to ch according to Ψ.

In contrast to unicast connectivity, we do not require broadcast connectedness to be symmetric or transitive, so in particular $M \mathrel{\dot{\prec}} K$ might not be equivalent to $K \mathrel{\dot{\succ}} M$. Instead, for technical reasons related to scope extension (cf. Example 13), broadcast channels must have no greater support than the input and output prefixes that send and receive on them.

Definition 6 (Requirements for broadcast)

1. $\Psi \vdash M \mathrel{\dot{\prec}} K \implies \mathsf{n}(M) \supseteq \mathsf{n}(K)$
2. $\Psi \vdash K \mathrel{\dot{\succ}} M \implies \mathsf{n}(K) \subseteq \mathsf{n}(M)$

Definition 7 (Transitions of Broadcast Psi) *To the actions of psi-calculi we add broadcast input, written $\underline{?K}\ N$ for a reception of N on K, and broadcast output, written $\overline{!K}\ (\nu\tilde{a})N$ for a broadcast of N on K, with names $\tilde{a}$ fresh in K. As before, we omit $(\nu\tilde{a})$ when $\tilde{a}$ is empty, and in examples we omit N when it is not relevant. The transitions of well-formed agents are defined inductively in Tables 2 and 1, where we let α range over both unicast and broadcast actions.*

The rule BrOut allows transmission on a broadcast channel K that the subject M of an output prefix is out-connected to. Similarly, the rule BrIn allows input from a broadcast channel K that the subject M of an input prefix is in-connected to. The environmental assertion Ψ determines if a prefix is connected to a broadcast channel and thus gives rise to a broadcast in BrIn and BrOut. In the same way it determines if a prefix is channel equivalent to something else and thus gives rise to a unicast in

$$\textsc{BrOut}\ \frac{\Psi \vdash M \mathrel{\dot{\prec}} K}{\Psi \rhd \overline{M}\,N\,.\,P \xrightarrow{!\overline{K}\,N} P} \qquad \textsc{BrIn}\ \frac{\Psi \vdash K \mathrel{\dot{\succ}} M}{\Psi \rhd \underline{M}(\lambda\widetilde{y})N\,.\,P \xrightarrow{?\underline{K}\,N[\widetilde{y}:=\widetilde{L}]} P[\widetilde{y}:=\widetilde{L}]}$$

$$\textsc{BrMerge}\ \frac{\Psi_Q \otimes \Psi \rhd P \xrightarrow{?\underline{K}\,N} P' \qquad \Psi_P \otimes \Psi \rhd Q \xrightarrow{?\underline{K}\,N} Q'}{\Psi \rhd P \mid Q \xrightarrow{?\underline{K}\,N} P' \mid Q'}$$

$$\textsc{BrCom}\ \frac{\Psi_Q \otimes \Psi \rhd P \xrightarrow{!\overline{K}\,(\nu\widetilde{a})N} P' \qquad \Psi_P \otimes \Psi \rhd Q \xrightarrow{?\underline{K}\,N} Q'}{\Psi \rhd P \mid Q \xrightarrow{!\overline{K}\,(\nu\widetilde{a})N} P' \mid Q'}\ \widetilde{a}\#Q$$

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

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

Table 2 Operational broadcast semantics. A symmetric version of BrCom is elided. In rules BrCom and BrMerge we assume that $\mathcal{F}(P) = (\nu\widetilde{b}_P)\Psi_P$ and $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$ where $\widetilde{b}_P$ is fresh for $P, \widetilde{b}_Q, Q, K$ and Ψ, and that $\widetilde{b}_Q$ is fresh for $Q, \widetilde{b}_P, P, K$ and Ψ.

In and Out. The same prefix could theoretically be used for both kinds of communication, although it may be unusual to find situations where that would be useful.

When two parallel processes both receive a broadcast on the same channel, the rule BrMerge combines the two actions. This rule is necessary to ensure the associativity of parallel composition. After a broadcast communication using BrCom, the resulting action is the original transmission. This is different from the unicast Com rule, where a communication yields an internal action τ. The BrOpen rule allows broadcast communication of data containing scoped names. Rule BrClose states that a broadcast transmission does not reach beyond its scope. This allows for broadcasting on restricted channels. Dually, the Scope rule (of Table 1) ensures that broadcast receivers on restricted channels cannot proceed unless a message is sent. The Par rule allows for broadcasts to bypass a process, as in most other broadcast calculi for wireless systems.

3.1 Meta-theory

We have developed a meta-theory for broadcast psi-calculi. Theorems 8, 10 and 11 give us assurance that any broadcast psi-calculus has a compositional labelled bisimilarity that respects important structural laws. The proofs of

these results are mostly straightforward extensions of the corresponding proofs for standard (unicast) psi-calculi [15,4], where some technical lemmas can be simplified because of the requirement of syntactic equality of channels in rules BrCom and BrMerge. Most of the added complications are caused by the fact that the BrCom rule defers the closing of the communication to BrClose; cf. Lemma 12. The proofs [28] are formally verified in the interactive theorem prover Isabelle/Nominal. The full formalisation of broadcast psi-calculi amounts to ca 33 000 lines of Isabelle code, of which about 21 000 lines are re-used from our earlier work [6].

In the following we restrict attention to well-formed agents.

Theorem 8 (Congruence properties of strong bisimulation) *For all Ψ:*

$$\begin{aligned}
P \mathrel{\dot\sim_\Psi} Q &\implies P \mid R \mathrel{\dot\sim_\Psi} Q \mid R \\
P \mathrel{\dot\sim_\Psi} Q &\implies (\nu a)P \mathrel{\dot\sim_\Psi} (\nu a)Q \qquad \textit{if } a\#\Psi \\
P \mathrel{\dot\sim_\Psi} Q &\implies !P \mathrel{\dot\sim_\Psi} !Q \qquad \textit{if } P, Q \textit{ assertion guarded} \\
\forall i. P_i \mathrel{\dot\sim_\Psi} Q_i &\implies \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \mathrel{\dot\sim_\Psi} \mathbf{case}\ \widetilde{\varphi} : \widetilde{Q} \\
P \mathrel{\dot\sim_\Psi} Q &\implies \overline{M}\,N\,.\,P \mathrel{\dot\sim_\Psi} \overline{M}\,N\,.\,Q \\
(\forall \widetilde{L}.\ P[\widetilde{x} := \widetilde{L}] \mathrel{\dot\sim_\Psi} Q[\widetilde{x} := \widetilde{L}]) &\implies \underline{M}(\lambda\widetilde{x})N\,.\,P \mathrel{\dot\sim_\Psi} \underline{M}(\lambda\widetilde{x})N\,.\,Q
\end{aligned}$$

As usual in channel-passing calculi, bisimulation is not a congruence for input prefix. We can characterise strong bisimulation congruence in the usual way.

Definition 9 (Strong Congruence) *$P \sim_\Psi Q$ iff for all sequences σ of substitutions it holds that $P\sigma \mathrel{\dot\sim_\Psi} Q\sigma$. We write $P \sim Q$ for $P \sim_{\mathbf{1}} Q$.*

Theorem 10 *Strong congruence $\sim_\Psi$ is a congruence for all Ψ.*

The standard rules of structural equivalence are sound for bisimilarity congruence.

Theorem 11 (Structural equivalence) *Assume that $a\#Q, \widetilde{x}, M, N, \widetilde{\varphi}$. Then*

$$\begin{array}{cc}
\mathbf{case}\ \widetilde{\varphi} : \widetilde{(\nu a)P} \sim (\nu a)\mathbf{case}\ \widetilde{\varphi} : \widetilde{P} & (\nu a)\mathbf{0} \sim \mathbf{0} \\
\underline{M}(\lambda\widetilde{x})N\,.\,(\nu a)P \sim (\nu a)\underline{M}(\lambda\widetilde{x})(N)\,.\,P & Q \mid (\nu a)P \sim (\nu a)(Q \mid P) \\
\overline{M}\,N\,.\,(\nu a)P \sim (\nu a)\overline{M}\,N\,.\,P & (\nu b)(\nu a)P \sim (\nu a)(\nu b)P \\
P \mid (Q \mid R) \sim (P \mid Q) \mid R & !P \sim P \mid !P \\
P \mid Q \sim Q \mid P & P \sim P \mid \mathbf{0}
\end{array}$$

When proving Theorem 11 we encountered an unusual complication in the proof of the commutativity of restriction, due to the BrClose rule. Since this rule can insert binder sequences under name restrictions, the simulation proof needs to allow for permutations of sequences of top-level binders. This is the main difference in our meta-theoretical proofs as compared to the original psi-calculi. We write $\widetilde{a} \equiv \widetilde{b}$ to denote that the sequence $\widetilde{a}$ is a rearrangement of $\widetilde{b}$, preserving the number of occurrences of each name.

Lemma 12 *For all Ψ, P, x, y, we have $(\nu y)(\nu x)P \dot{\sim}_\Psi (\nu x)(\nu y)P$.*

Proof In standard psi-calculi, the proof of this result uses the candidate relation $\mathcal{S}_0 \stackrel{def}{=} \{(\Psi, (\nu y)(\nu x)P, (\nu x)(\nu y)P) : x, y \# \Psi\}$. Here we inductively close this relation under restriction, yielding $\mathcal{S}$:

$$\mathcal{S} \stackrel{def}{=} \mathcal{S}_0 \cup \{(\Psi, (\nu a)P, (\nu a)Q) : (\Psi, P, Q) \in \mathcal{S} \wedge a \# \Psi\}$$

We show that $\mathcal{S}$ is a bisimulation up to transitivity [29] (at every Ψ). That is, we only require the derivatives after a simulation step to be related by $\mathcal{S}^*$, inductively defined as

$$\mathcal{S}^* \stackrel{def}{=} \{(\Psi, P, P)\} \cup \{(\Psi, P, R) \ : \ \exists Q.\ (\Psi, P, Q) \in \mathcal{S}^* \wedge (\Psi, Q, R) \in \mathcal{S}\}.$$

We have proven "up to transitivity" to be sound, i.e., every bisimulation up to transitivity is a subset of some ordinary bisimulation.

The interesting part of the proof is in the simulation clause. We here consider only the base case of the definition of $\mathcal{S}$ (i.e. $\mathcal{S}_0$), where we need to prove that for all α, P' such that $\mathrm{bn}(\alpha) \# \Psi, Q$ and $\Psi \rhd (\nu y)(\nu x)P \xrightarrow{\alpha} P'$ there exists a Q' such that $\Psi \rhd (\nu x)(\nu y)P \xrightarrow{\alpha} Q'$ and $(\Psi, P', Q') \in \mathcal{S}^*$.

We first define a relation $\mathcal{R}$ that safely approximates $\mathcal{S}^*$ (i.e. $\mathcal{R} \subseteq \mathcal{S}^*$) and is easier to work with.

$$\mathcal{R} \stackrel{def}{=} \{(\Psi, (\nu \widetilde{a})P, (\nu \widetilde{b})P) : \widetilde{a} \# \Psi \wedge \widetilde{a} \equiv \widetilde{b}\}.$$

By induction on the length of $\widetilde{a}$, we get that for all $\widetilde{a}, \widetilde{b}, \Psi, P$ such that $\widetilde{a} \# \Psi$ and $\widetilde{a} \equiv \widetilde{b}$ we have $(\Psi, (\nu \widetilde{a})P, (\nu \widetilde{b})P) \in \mathcal{S}^*$. From this follows that the relation $\mathcal{R} \subseteq \mathcal{S}^*$; in order to show that the derivatives $(\Psi, P', Q') \in \mathcal{S}$ after a simulation step, we instead prove $(\Psi, P', Q') \in \mathcal{R}$.
The simulation proof is by case analysis on the derivations of transitions of $(\nu y)(\nu x)P$. We here focus on on the following derivation.

$$\textsc{Scope}\ \frac{\textsc{BrClose}\ \dfrac{\Psi \rhd P \xrightarrow{\overline{!M}\ (\nu \widetilde{a})N} P'}{\Psi \rhd (\nu x)P \xrightarrow{\tau} (\nu x)(\nu \widetilde{a})P'}\ x \in \mathsf{n}(M), x \# \Psi}{\Psi \rhd (\nu y)(\nu x)P \xrightarrow{\tau} (\nu y)(\nu x)(\nu \widetilde{a})P'}\ y \# \tau, \Psi$$

We assume that $\widetilde{a} \# (\Psi, P, M, x, y)$. There are three cases to consider.

1. $y \# (\overline{!M}\ (\nu \widetilde{a})N)$: We have the following transition.

$$\textsc{BrClose}\ \frac{\textsc{Scope}\ \dfrac{\Psi \rhd P \xrightarrow{\overline{!M}\ (\nu \widetilde{a})N} P'}{\Psi \rhd (\nu y)P \xrightarrow{\overline{!M}\ (\nu \widetilde{a})N} (\nu y)P'}\ y \# \overline{!M}\ (\nu \widetilde{a})N, \Psi}{\Psi \rhd (\nu x)(\nu y)P \xrightarrow{\tau} (\nu x)(\nu \widetilde{a})(\nu y)P'}\ x \in \mathsf{n}(M), x \# \Psi$$

Since $x, y, \widetilde{a} \# \Psi$ and $(x, \widetilde{a}, y) \equiv (y, x, \widetilde{a})$ we have
$(\Psi, (\nu y)(\nu x)(\nu \widetilde{a})P', (\nu x)(\nu \widetilde{a})(\nu y)P') \in \mathcal{R} \subseteq \mathcal{S}^*$.

2. $y \in \mathsf{n}(\overline{!M}\ (\nu\widetilde{a})N)$ and $y \in \mathsf{n}(M)$: We have the following transition.

$$\textsc{Scope}\ \frac{\textsc{BrClose}\ \dfrac{\Psi \rhd P \xrightarrow{\overline{!M}\ (\nu\widetilde{a})N} P'}{\Psi \rhd (\nu y)P \xrightarrow{\tau} (\nu y)(\nu\widetilde{a})P'}\ y \in \mathsf{n}(M), y\#\Psi}{\Psi \rhd (\nu x)(\nu y)P \xrightarrow{\tau} (\nu x)(\nu y)(\nu\widetilde{a})P'}\ x\#\tau,\Psi$$

Since $x, y\#\Psi$ and $y, x \equiv x, y$ we have $(\Psi, (\nu y)(\nu x)(\nu\widetilde{a})P', (\nu x)(\nu y)(\nu\widetilde{a})P') \in \mathcal{R} \subseteq \mathcal{S}^*$

3. $y \in \mathsf{n}(\overline{!M}\ (\nu\widetilde{a})N)$ and $y\#M$: We then have $y \in \mathsf{n}(N)$, and derive

$$\textsc{BrClose}\ \frac{\textsc{BrOpen}\ \dfrac{\Psi \rhd P \xrightarrow{\overline{!M}\ (\nu\widetilde{a})N} P'}{\Psi \rhd (\nu y)P \xrightarrow{\overline{!M}\ (\nu y)(\nu\widetilde{a})N} P'}\ y\#\widetilde{a},\Psi,M,y \in \mathsf{n}(N)}{\Psi \rhd (\nu x)(\nu y)P \xrightarrow{\tau} (\nu x)(\nu y)(\nu\widetilde{a})P'}\ x \in \mathsf{n}(M), x\#\Psi$$

Since $x, y\#\Psi$ and $y, x \equiv x, y$ we have $(\Psi, (\nu y)(\nu x)(\nu\widetilde{a})P', (\nu x)(\nu y)(\nu\widetilde{a})P') \in \mathcal{R} \subseteq \mathcal{S}^*$. $\square$

The soundness proof for scope extension uses the same ideas as the proof of Lemma 12.

3.2 Motivating the Requisites

An apparently simpler way to define broadcast connectivity is to have just one binary connectivity predicate relating input and output prefixes, as $\dot{\leftrightarrow}$ does for unicast communication. However, such a predicate would need to be transitive and symmetric for Theorem 11 to hold, for the same reasons as in the original psi calculus (detailed in [6]). In wireless broadcast communication systems, symmetry and transitivity do not necessarily hold and the requirements would not be reasonable.

A weaker version of condition 2 (resp. 1) of Definition 6 would be to require $\mathsf{n}(K) \subseteq \mathsf{n}(M, \Psi)$ whenever $\Psi \vdash K \mathrel{\dot{\succ}} M$ (resp. $\Psi \vdash M \mathrel{\dot{\prec}} K$). However, this leads to structural equivalence not being sound for bisimulation: the scope extension case of Theorem 11 fails, as we see in the following example.

Example 13 We let $\mathbf{A} = \mathcal{P}_{\text{fin}}(\mathcal{N})$ with $\mathbf{1} = \emptyset$ and $\otimes = \cup$. We let $\mathbf{T} = \mathcal{N}$ and $\mathbf{C} = \{a \mathrel{\dot{\leftrightarrow}} b, a \mathrel{\dot{\prec}} b, a \mathrel{\dot{\succ}} b : a, b \in \mathcal{N}\}$. We define $\vdash$ by $\forall\Psi, a, b,\ \Psi \vdash b \mathrel{\dot{\prec}} b$ iff $b \in \Psi$ and $\Psi \vdash b \mathrel{\dot{\succ}} a$ iff $b \in \Psi$. Note that this defintion of entailment does not satisfy Definition 6, since we may have $\Psi \vdash b \mathrel{\dot{\succ}} a$ for some $b \neq a$.

We let $P := (\nu a)(\langle\!|\{a\}|\!\rangle \mid \overline{a}.\mathbf{0} \mid \underline{c}.\mathbf{0})$. Here $\mathbf{1} \rhd P \xrightarrow{\tau} (\nu a)(\langle\!|\{a\}|\!\rangle \mid \mathbf{0} \mid \mathbf{0})$. However, P results from scope extension from $Q := (\nu a)(\langle\!|\{a\}|\!\rangle \mid \overline{a}.\mathbf{0}) \mid \underline{c}.\mathbf{0}$, but Q does not have a corresponding transition under frame $\mathbf{1}$.

In contrast to unicast actions, the support of the subjects of broadcast actions is always included in the support of the process generating the action. This result is used in the proof of the scope extension case of Theorem 11, to show that a scope extension does not enable any additional broadcast communication.

Lemma 14 *If* $\Psi \rhd P \xrightarrow{\overline{!K}\,(\nu\tilde{a})N} P'$ *or* $\Psi \rhd P \xrightarrow{\underline{?K}\,N} P'$ *then* $\mathsf{n}(K) \subseteq \mathsf{n}(P)$.

Proof By induction on the derivation, using Definition 6 at the base cases.

4 Modelling network topology changes

When modelling wireless protocols, one important concern is dealing with connectivity changes. We here give general descriptions of methods of modelling different connectivity configurations using assertions.

The main idea is to allow for different generations of assertions by tagging assertions with a time. Only the most recent generation is used; a generation is made obsolete by composition with an assertion from a later generation. We here consider broadcast connectivity, but this technique can also be used in other scenarios where there is a need to retract assertions. In the following we assume a set of terms $\mathbf{B} \subseteq \mathbf{T}$ used as broadcast channels and in prefixes; we let B, B' range over elements of $\mathbf{B}$.

4.1 Simple topology

Here assertions are finite sets of connectivity information ($M \dot{\prec} K$ resp. $K \dot{\succ} M$), labelled with a time, with the empty set at time 0 as the unit assertion. Assertion composition intuitively computes the union of all connectivity information labelled with the most recent generation. The sets $\mathbf{C}$ and $\mathbf{A}$ are defined using constructors operating on terms. We define substitution on $\mathbf{C}$ and $\mathbf{A}$ homomorphically on their structure. For simplicity, we assume that no rewriting happens in broadcast output, i.e., that $\dot{\prec}$ is the equality relation of $\mathbf{B}$.

Formally,

$$
\begin{aligned}
\mathbf{C} &\triangleq \{\bot\} \cup \{\mathsf{currentGeneration}(g) : g \in \mathbb{N}\} \cup \\
&\quad \{K \dot{\succ} M : K, M \in \mathbf{T}\} \cup \{M \dot{\prec} K : K, M \in \mathbf{T}\} \\
\mathbf{A} &\triangleq \mathbb{N} \times \mathcal{P}_{\text{fin}}(\{\langle K \dot{\succ} M\rangle : K, M \in \mathbf{T}\}) \\
\mathbf{1} &\triangleq \langle 0, \emptyset\rangle
\end{aligned}
$$

$$
\langle g, S\rangle \otimes \langle g', T\rangle \triangleq \begin{cases} \langle g, S\rangle & \text{if } g > g' \\ \langle g', T\rangle & \text{if } g < g' \\ \langle g, S \cup T\rangle & \text{if } g = g' \end{cases}
$$

$\langle g, S\rangle \vdash \mathsf{currentGeneration}(g')$ iff $g = g'$
$\langle g, S\rangle \vdash B \dot{\prec} B'$ if $B = B'$
$\langle g, S\rangle \vdash B \dot{\succ} B'$ if $B \dot{\succ} B' \in S$ and $\mathsf{n}(B) \subseteq \mathsf{n}(B')$

Proposition 15 *Given* $\mathbf{T}$ *with a substitution function satisfying the requirements of Section 2, the definitions of* $\mathbf{C}$, $\mathbf{A}$, $\otimes$, $\mathbf{1}$ *and* $\vdash$ *as above and* $(M \leftrightarrow N) \triangleq \bot$ *satisfy the requirements of a broadcast psi-calculus.*

The assertion $\langle g, \{B \mathrel{\dot{\succ}} B'\}\rangle$ states that B' is in-connected to B in generation g if $\mathsf{n}(B) \subseteq \mathsf{n}(B')$. The condition $\mathsf{currentGeneration}(g)$ is used to test if g is the most recent generation. It is needed for assertion equivalence to be compositional: without this condition we would have $\langle 0, \{M \mathrel{\dot{\succ}} K\}\rangle \simeq \langle 1, \{M \mathrel{\dot{\succ}} K\rangle\}$ and $\langle 0, \{M \mathrel{\dot{\succ}} K\}\rangle \otimes \langle 1, \{K \mathrel{\dot{\succ}} M\}\rangle \not\simeq \langle 1, \{M \mathrel{\dot{\succ}} K\}\rangle \otimes \langle 1, \{K \mathrel{\dot{\succ}} M\}\rangle$, contradicting compositionality.

As an example, we can define a topology controller (assuming a suitable encoding of the τ prefix):

$$T = (\!|\langle 1, \emptyset\rangle|\!) \mid \tau\,.\,\big((\!|\langle 2, \{K \mathrel{\dot{\succ}} M, K \mathrel{\dot{\succ}} N\}\rangle|\!) \mid \tau\,.\,((\!|\langle 3, \{K \mathrel{\dot{\succ}} M\}\rangle|\!))\big)$$

In $P \mid T$, the process P broadcasts on K while T manages the topology. Initially $\mathcal{F}(T) = \langle 1, \emptyset\rangle$ and the broadcast is disconnected; after $T \xrightarrow{\tau} T'$ then $\mathcal{F}(T') = \langle 2, \{K \mathrel{\dot{\succ}} M, K \mathrel{\dot{\succ}} N\}\rangle$ and a broadcast on K can be received on both M and N, and after $T' \xrightarrow{\tau} T''$ then a broadcast can be received only on M, since $\mathcal{F}(T'') = \langle 3, \{K \mathrel{\dot{\succ}} M\}\rangle$.

Such a connectivity controller can also implement standard mobility models [11] over a discretized finite space. More fine-grained mobility models can be implemented by associating a generation with each possible connection, together with a flag for whether the connection is possible or not. In such a model, assertion $\{\langle 0, M \mathrel{\dot{\succ}} K, \mathrm{true}\rangle\}$ states that the link $M \mathrel{\dot{\succ}} K$ is enabled in its generation 0.

4.2 Scoped topology

As a variation of the example above we define a model where every name d corresponds to a broadcast channel with dynamic topology. The use of a name in the broadcast channel allows to restrict its scope.

$$\begin{array}{rcl}
\mathbf{B} & \triangleq & \{\mathsf{Bs}(d) : d \in \mathcal{N}\} \cup \{\mathsf{Br}(M, d) : M \in \mathbf{T},\ d \in \mathcal{N}\} \cup \mathcal{N} \\
\mathbf{C} & \triangleq & \{\bot\} \cup \{\mathsf{currentGeneration}(g, K) : g \in \mathbb{N},\ K \in \mathbf{T}\} \cup \\
 & & \{\mathsf{Bs}(M) \mathrel{\dot{\prec}} K) : M, K \in \mathbf{T}\} \cup \{K \mathrel{\dot{\succ}} \mathsf{Br}(M, N) : M, N, K \in \mathbf{T}\} \\
\mathbf{A} & \triangleq & \mathbf{T} \rightarrow_{\mathrm{fin}} \mathbb{N} \times \mathcal{P}_{\mathrm{fin}}(\{\mathsf{Conn}(M, N) : M, N \in \mathbf{T}\}) \\
\mathbf{1} & \triangleq & \emptyset
\end{array}$$

$$(\Psi \otimes \Psi')(M) \triangleq \begin{cases}
\langle g, S\rangle & \text{if } \Psi(M) = \langle g, S\rangle \wedge \ (M \notin \mathrm{dom}(\Psi') \vee \\
 & \qquad (\Psi'(M) = \langle j, T\rangle \wedge g > g')) \\
\langle g', T\rangle & \text{if } \Psi'(M) = \langle g', T\rangle \wedge (M \notin \mathrm{dom}(\Psi) \vee \\
 & \qquad (\Psi(M) = \langle g, S\rangle \wedge g < g')) \\
\langle g, S \cup T\rangle & \text{if } \Psi(M) = \langle g, S\rangle \wedge \ \Psi'(M) = \langle g, T\rangle
\end{cases}$$

$\Psi \vdash \mathsf{currentGeneration}(g, d)$ if $\Psi(d) = \langle g, S\rangle$
$\Psi \vdash \mathsf{Bs}(c) \dot{\prec} d$ if $c = d$
$\Psi \vdash c \dot{\succ} \mathsf{Br}(N, d)$ if $c = d$ and $\Psi(c) = \langle g, S\rangle$ with $\mathsf{Conn}(N, d) \in S$

Proposition 16 *Given* $\mathbf{T}$ *with a substitution function satisfying the requirements of Section 2, the definitions of* $\mathbf{C}$, $\mathbf{A}$, $\otimes$, $\mathbf{1}$ *and* $\vdash$ *as above and* $(M \nleftrightarrow N) \triangleq \bot$ *satisfy the requirements of a broadcast psi-calculus.*

We can then define a topology controller which gradually changes the topology from fully disconnected to "a listens on d and b listens on d":

$$T = (\!|d \mapsto \langle 1, \emptyset\rangle|\!) \mid \tau\,.\,((\!|d \mapsto \langle 2, \{\mathsf{Conn}(a, d)\}\rangle|\!) \\ \mid \tau\,.\,((\!|d \mapsto \langle 3, \{\mathsf{Conn}(a, d), \mathsf{Conn}(b, d)\}\rangle|\!)))$$

We now put a process P inside the scope of d in parallel with the topology controller as $(\nu d)(P \mid T)$. This ensures that P can communicate using broadcast on channel d while letting T, but not the environment, influence the topology. Moreover, no process in the environment can receive broadcasts from P (unless having previously received the bound name d. In this way, scoped topology enables hierarchical modelling of sub-systems using wireless broadcast.

5 The LUNAR protocol in Psi

In this section we present a model of the LUNAR routing protocol for mobile ad-hoc networks [32,33]. LUNAR is intended for small wireless networks, ca 15 nodes, with a network diameter of 3 hops. It does not handle route reparation, caching etc, and routes must be re-established every few seconds. It is reasonably simple in comparison to many other ad-hoc routing protocols, and allows us to focus on properties such as dynamic connectivity and broadcasting. It has previously been verified in [36,35] using SPIN and UPPAAL; our model is significantly more succinct and at an abstraction level closer to the specification.

The LUNAR protocol is at "layer 2.5", between the link and network layers in the Internet protocol stack. Addressing is by pairs of MAC/Ethernet addresses and 64-bit selectors, similarly to the IP address and port number used in UDP/TCP. The selectors are used to find the appropriate packet handler through the FIB (Forwarding Information Base) table.

Below, we define a psi-calculus for modelling the LUNAR protocol. In an effort to keep our model simple we abstract from details such as time-to-live (TTL) fields in messages, optional protocol fields, globally unique host identifiers, etc. These abstractions are similar to those made in [36,35]. We do not deal with time explicitly. In the SPIN verification, time is handled at an abstract level by using the Promela `timeout` predicate which is true when no other statement is executable, and checking that in this case, the protocol has succeeded in delivering a message (cf. Theorem 18).

5.1 The LUNAR broadcast psi-calculus

Channels are of two kinds: broadcast channels are terms node_i with (for simplicity) empty support, whose connectivity is given by the $\dot{\succ}$ and $\dot{\prec}$ predicates as defined in Section 4.1, and unicast channels which are pairs $\langle sel, mac\rangle$ where *sel* is a selector name and *mac* is a MAC address name. The *sel* part can also be a $\mathsf{RouteOf}(node, ip)$ construction, which looks up the route of an IP address *ip* in the routing table of the node *node*. Special channels $\langle \mathsf{delivered}, \mathsf{node}_i\rangle$ are used to signal delivery of a packet to the IP layer. Assertions are used to record requests originated at the local node with $\mathsf{Redirected}(node, sel)$, and with $\mathsf{HaveRoute}(node, destip, hops, sel)$ to specify found routes. The conditions contain predicates for testing if a route has been found ($\mathsf{HaveRoute}(node, ip)$), if a selector has been used for a request originating at the local node ($\mathsf{Redirected}(node, sel)$), and to extract the forwarder of a route ($\langle \mathsf{RouteOf}(node, ip), x\rangle \leftrightarrows \langle sel, x\rangle$).

LUNAR protocol messages are of two types. The first is a route request message $\mathsf{RREQ}(selector, targetIP, replyTo)$, where the *selector* identifies the request, *targetIP* is the IP address the route should reach, and *replyTo* is the $\langle sel, mac\rangle$ channel the response should be sent to. The second is a route reply message, $\mathsf{RREP}(hops, fwdptr)$), where *hops* is the number of hops to the destination, and *fwdptr* is a forwarding pointer, i.e. a $\langle sel, mac\rangle$ channel where packets can be sent.

The parameters of the LUNAR broadcast psi-calculus extend the simple topology calculus in Section 4.1. We define substitution in the standard way, as the syntactic replacement of names by terms. The sets $\mathbf{T}, \mathbf{C}$ and $\mathbf{A}$ are defined recursively using constructors operating on terms in order to be closed under substitution.

$$
\begin{aligned}
\mathbf{T} \triangleq\ & \mathcal{N} \cup \{\mathsf{node}_i : i \in \mathbb{N}\} \cup \{\mathsf{delivered}\} \cup \\
& \{\mathsf{RREQ}(Ser,\ TargIp, Rep) : Ser,\ TargIp,\ Rep \in \mathbf{T}\} \cup \\
& \{\mathsf{RREP}(i, Fwd) : i,\ Fwd \in \mathbf{T}\} \cup \\
& \{\mathsf{RouteOf}(Node, Ip) : Node,\ Ip \in \mathbf{T}\} \cup \\
& \{\langle Sel, N\rangle : Sel, N \in \mathbf{T}\} \cup \{N + 1 : N \in \mathbf{T}\} \cup \{0\} \\
\mathbf{C} \triangleq\ & \{M = N, M \leftrightarrows N, \mathsf{HaveRoute}(M, N), \mathsf{Redirected}(M, N) : M, N \in \mathbf{T}\} \cup \\
& \{K \dot{\succ} M : K, M \in \mathbf{T}\} \cup \{M \dot{\prec} K : K, M \in \mathbf{T}\} \cup \\
& \{\mathsf{currentGeneration}(g) : g \in \mathbb{N}\} \cup \{\neg\phi : \phi \in \mathbf{C}\} \\
\mathbf{A} \triangleq\ & \mathbb{N} \times \mathcal{P}_{\text{fin}}(\{\langle K \dot{\succ} M\rangle : K, M \in \mathbf{T}\}) \times \\
& \mathcal{P}_{\text{fin}}(\{\mathsf{HaveRoute}(M, N_1, i, N_2) : i,\ M, N_1, N_2 \in \mathbf{T}\} \cup \\
& \quad \{\mathsf{Redirected}(M, N) : M, N \in \mathbf{T}\}) \\
\mathbf{1} \triangleq\ & \langle 0, \emptyset, \emptyset\rangle
\end{aligned}
$$

$$
\langle g, S, A\rangle \otimes \langle g', T, B\rangle \triangleq \begin{cases} \langle g, S, A \cup B\rangle & \text{if } g > g' \\ \langle g', T, A \cup B\rangle & \text{if } g < g' \\ \langle g, S \cup T, A \cup B\rangle & \text{if } g = g' \end{cases}
$$

Given $\Psi = \langle g, S, A\rangle$, we let $\mathcal{R}_\Psi$ be the symmetric and transitive closure of the relation

$$\{(\langle a, b\rangle, \langle a, b\rangle) : a, b \in \mathcal{N}\} \cup \{(\langle \mathsf{delivered}, \mathsf{node}_i\rangle, \langle \mathsf{delivered}, \mathsf{node}_i\rangle) : i \in \mathbb{N}\} \cup$$
$$\{(\langle \mathsf{RouteOf}(\mathsf{node}_i, a), x\rangle, \langle b, x\rangle) : i \in \mathbb{N}, j \in \mathbf{T}, \mathsf{HaveRoute}(\mathsf{node}_i, a, j, b) \in A\}$$

Entailment is then defined as follows.

$$\begin{aligned}
\Psi &\vdash a = a,\ a \in \mathcal{N} \\
\Psi &\vdash M \dot{\leftrightarrow} N \text{ iff } (M, N) \in \mathcal{R}_\Psi \\
\langle g, S, A\rangle &\vdash \mathsf{currentGeneration}(g) \\
\Psi &\vdash M \dot{\prec} N \text{ iff } M = N \\
\langle g, S, A\rangle &\vdash M \dot{\succ} N \text{ iff } M \dot{\succ} N \in S \\
&\qquad\qquad \text{and } \mathsf{n}(M) \subseteq \mathsf{n}(N) \\
\langle g, S, A \cup \{\mathsf{HaveRoute}(\mathsf{node}_i, a, j, b)\}\rangle &\vdash \mathsf{HaveRoute}(\mathsf{node}_i, a) \\
\langle g, S, A \cup \{\mathsf{Redirected}(\mathsf{node}_i, s)\}\rangle &\vdash \mathsf{Redirected}(\mathsf{node}_i, s) \\
\Psi &\vdash \neg\varphi \text{ if not } \Psi \vdash \varphi
\end{aligned}$$

Theorem 17 *The LUNAR psi-calculus defined above satisfies all the requisites of a broadcast psi-calculus.*

This theorem has been formally proved in Isabelle/Nominal [2]. A sketch outlining the main ideas of the proof follows:

Proof (sketch) The requisites on the support of the broadcast channels are immediate from the definition. It is straight-forward to show the Abelian monoid laws for $\otimes, \mathbf{1}$. Transitivity and symmetry of channel equivalence holds by definition. The only nontrivial property is compositionality: We establish that $\Psi \otimes \Psi_1 \vdash \varphi$ and $\Psi_1 \simeq \Psi_2$ implies $\Psi \otimes \Psi_2 \vdash \varphi$ by induction on the structure of the condition φ. The only inductive step is for negation and this follows by symmetry of $\simeq$. If φ is a broadcast connectivity condition or $\mathsf{currentGeneration}(\mathsf{g})$, the proof is by case distinction on the relative generations of Ψ_1, Ψ_2 and Ψ. If φ is a channel equivalence an inner induction on the length of the chain of the involved $\mathsf{HaveRoute}$ elements in $\Psi \otimes \Psi_1$ is necessary. Each such element is either in Ψ and therefore also in $\Psi \otimes \Psi_2$, or in Ψ_1. In the latter case Ψ_1 entails a channel equivalence from this element alone and therefore Ψ_2 entails the same. Thus Ψ_2 must contain a suitable sequence of $\mathsf{HaveRoute}$ elements to derive this channel equivalence; this sequence is then in $\Psi \otimes \Psi_2$.

5.2 Representing process identifiers

We use process identifiers to improve the readability of the LUNAR protocol model. However, an astute reader will note that broadcast psi-calculi do not feature process identifiers - rather, replication is used as the mechanism for expressing infinite behaviour. In many other process calculi, process

identifiers and recursion can be encoded in a standard fashion using replication, see e.g. [30]. Unfortunately, there is currently no proof that the same encodability results apply to broadcast psi-calculi.

To introduce process identifiers on a more sound theoretical foundation, we combine broadcast psi-calculi with higher-order psi-calculi [24], an orthogonal extension of psi-calculi which allows terms to act as handles to invoke the behaviour of processes. In this setting, process identifiers are simply terms.

Briefly, higher-order psi-calculi introduce the notion of a *clause* $M \Leftarrow P$, meaning that the term M is a handle for invoking P. We extend the entailment relation $\vdash$ so that assertions can entail clauses in addition to conditions. Agents are extended with *invocations* $\mathbf{run}\ M$, and a single new rule is added to the semantics:

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

The calculi that result from adding the above-mentioned extensions to broadcast psi-calculi will be referred to as *higher-order broadcast psi-calculi*. We use Isabelle/Nominal to formally prove that all the meta-theoretic results presented in Section 3.1 apply not only to broadcast psi-calculi, but also to higher-order broadcast psi-calculi - hence we feel justified in claiming that broadcast and higher-order are orthogonal extensions. The proof scripts are available online [2].

Further, higher-order psi-calculi feature a lifting technique whereby an arbitrary first-order psi-calculus can be lifted to a corresponding canonical higher-order psi-calculus, extending it with parametrised clauses. In a canonical higher-order psi-calculus, sets of parametrised clauses on the form $M(N) \Leftarrow P$ are added to the assertions, such that $\{M(N) \Leftarrow P\} \vdash M(N[\widetilde{x} := \widetilde{T}]) \Leftarrow P[\widetilde{x} := \widetilde{T}]$.

In the following, we will implicitly be representing clauses using this feature of the canonical higher-order calculus corresponding to the LUNAR broadcast psi-calculus of Section 5.1.

5.3 The psi-calculus model of the LUNAR protocol

Figures 1-7 describe our psi-calculus model of the LUNAR protocol. Process declarations are of the form $M(\widetilde{N}) \Leftarrow P$, where M is a process identifier (and also a term, implicitly included in $\mathbf{T}$), $\widetilde{N}$ a list of terms where occurrences of names are binding, and P is a process s.t. $\mathsf{n}(P) \subseteq \mathsf{n}(\widetilde{N})$. In a process, we write $M(\widetilde{N})$ for invoking a process declaration $M(\widetilde{K}) \Leftarrow P$ such that $\widetilde{N} = \widetilde{K}[\widetilde{x} := \widetilde{L}]$ with $\widetilde{x} = \mathsf{n}(\widetilde{K})$, resulting in the process $P[\widetilde{x} := \widetilde{L}]$. For our purposes, lists can be adequately represented using the pairing construct included in the term language. We write $\mathbf{if}\ \varphi\ \mathbf{then}\ P\ \mathbf{else}\ Q$ for $\mathbf{case}\ \varphi : P\ [] \neg\varphi : Q$, and assume a suitable encoding of the τ prefix.

Our model of the protocol closely follows the informal protocol description in [33, Section 4]. Each figure in our model corresponds to one or more of part 0-5 of the protocol description. To allocate a selector, we simply bind a name; to associate (or bind) a selector to a packet handler we use a replicated process which receives on the unicast channel described by the pair of the selector and our MAC address. An example of this can be seen in the LunARP process declaration in Fig. 1. The description in [33, Section 4, step 0.a] says "`Allocate an unused "receiver chosen" selector S and bind it to a transient "source RREP packet handler"`", which in our process declaration corresponds to the binding of *rchosen* and the subprocess $!\,\underline{\langle rchosen, mymac\rangle}(x)\,.\,\mathsf{SRrepHandler}(mynode, mymac, destip, x)$.

In the informal protocol description [33], the FIB is "abused" (in steps 0.b and 1.b) by installing a null packet handler for the selector created when sending a route request. This FIB entry is only used to detect and avoid circular forwarding of route requests. We model this by an explicit assertion and a matching condition. An example can be seen is the subprocess $(\!|\mathsf{Redirected}(mynode, schosen)|\!)$ of the LunARP process declaration, and the test on the first line of the RreqHandler process declaration (Fig. 2) using the $\mathsf{Redirected}(mynode, schosen)$ condition.

The routing table is modelled using assertions, which illustrates how these can be used as a global data structure. Additions to the routing table are done in the SRrepHandler process definition (Fig. 4), which adds $(\!|\mathsf{HaveRoute}(mynode, destip, hops, rchosen)|\!)$ to the environment. Such assertions together form the routing table, which is tested in the IPtransmit process definition (Fig. 7) using the $\mathsf{HaveRoute}(mynode, destip)$ condition.

For simplicity we do not model route timeouts and the deletion of routes, but this could be done using the mechanism in Section 4.

The LUNAR procedure for route discovery starts when a node wants to send a message to a node it does not already have a route to (Fig. 7, **else** branch). It then (Fig. 1) associates a fresh selector with a response packet handler, and broadcasts a Route Request (RREQ) message to its neighbours. A node which receives a RREQ message (Fig. 2) for its own IP address sets up a packet handler to deliver IP packets, and includes the corresponding selector in a response Route Reply (RREP) message to the reply channel found in the RREQ message. If the RREQ message was not for its own IP address, the message is re-broadcast after replacing the reply channel with a freshly allocated reply selector and its own MAC address. When such an intermediary node receives a RREP message (Fig. 3), it increments the hop counter and forwards the RREP message to the source of the original RREQ message. When the originator of a RREQ message eventually receives the matching RREP (Fig. 4), it installs a route and informs the IP layer about it. The message can then be resent (Fig. 7, **then** branch) and delivered (Fig. 5) by unicast messages through the chain of intermediary forwarding nodes.

We show the basic correctness of the model by the following theorem, which in essence corresponds to the correct operation of an ad-hoc routing

$$\mathsf{LunARP}(mynode, mymac, destip) \Leftarrow (\nu rchosen, schosen)\left(\begin{array}{l} !\,\underline{\langle rchosen, mymac\rangle}(x)\,.\,\mathsf{SRrepHandler}(mynode, mymac, destip, x) \\ \mid (\!|\overline{\mathsf{Redirected}(mynode, schosen)}|\!) \\ \mid \overline{mynode}\langle \mathsf{RREQ}(schosen, destip, \langle rchosen, mymac\rangle)\rangle\,.\,\mathbf{0} \end{array}\right)$$

Fig. 1 Part 0: the initialisation step at the node that wishes to discover a route

$$\begin{array}{l} \mathsf{RreqHandler}(mynode, mymac, myip, \mathsf{RREQ}(schosen, destip, repchn)) \Leftarrow \\ \quad \textbf{if } \mathsf{Redirected}(mynode, schosen) \textbf{ then } \mathbf{0} \\ \quad \textbf{else } \tau\,.\,\Big((\!|\overline{\mathsf{Redirected}(mynode, schosen)}|\!) \mid \\ \qquad \textbf{if } destip = myip \textbf{ then} \qquad\qquad /^*\text{ Part 2: Target found }^*/ \\ \qquad\quad (\nu rchosen) \\ \qquad\qquad \left(\begin{array}{l} !\,\underline{\langle rchosen, mymac\rangle}(x)\,.\,\mathsf{IPdeliver}(x, mynode) \\ \mid \overline{repchn}\langle \mathsf{RREP}(0, \langle rchosen, mymac\rangle)\rangle\,.\,\mathbf{0} \end{array}\right) \\ \qquad \textbf{else} \\ \qquad\quad (\nu rchosen) \\ \qquad\qquad \left(\begin{array}{l} !\,\underline{\langle rchosen, mymac\rangle}(x)\,.\,\mathsf{IRrepHandler}(mymac, repchn, x) \\ \mid \overline{mynode}\langle \mathsf{RREQ}(schosen, destip, \langle rchosen, mymac\rangle)\rangle\,.\,\mathbf{0} \end{array}\right)\Big) \end{array}$$

Fig. 2 Part 1: RREQ packet handler, and Part 2: Target found branch

$$\mathsf{IRrepHandler}(mymac, repchn, \mathsf{RREP}(hops, fwdptr)) \Leftarrow (\nu rchosen)\left(\begin{array}{l} !\,\underline{\langle rchosen, mymac\rangle}(x)\,.\,\overline{fwdptr}\,x\,.\,\mathbf{0} \\ \mid \overline{repchn}\langle \mathsf{RREP}(hops+1, \langle rchosen, mymac\rangle)\rangle\,.\,\mathbf{0} \end{array}\right)$$

Fig. 3 Part 3: Intermediate RREP packet handler

$$\mathsf{SRrepHandler}(mynode, mymac, destip, \mathsf{RREP}(hops, fwdptr)) \Leftarrow (\nu rchosen)\left(\begin{array}{l} !\,\underline{\langle rchosen, mymac\rangle}(x)\,.\,\overline{fwdptr}\,x\,.\,\mathbf{0} \\ \mid (\!|\overline{\mathsf{HaveRoute}(mynode, destip, hops, rchosen)}|\!) \end{array}\right)$$

Fig. 4 Part 4: Source RREP packet handler

$$\mathsf{IPdeliver}(x, node) \Leftarrow \overline{\langle \mathsf{delivered}, node\rangle}\,x\,.\,\mathbf{0}$$

Fig. 5 Part 5: IP delivery

$$\mathsf{BrdHandler}(mynode, mac, ip) \Leftarrow \underline{mynode}(\lambda s, t, r)\mathsf{RREQ}(s, t, r)\,.\,\left(\begin{array}{l} \mathsf{RreqHandler}(mynode, mac, ip, \mathsf{RREQ}(s, t, r)) \\ \mid \mathsf{BrdHandler}(mynode, mac, ip) \end{array}\right)$$

Fig. 6 Broadcast handler

$$\begin{array}{l} \mathsf{IPtransmit}(mynode, mymac, destip, pkt) \Leftarrow \\ \quad \textbf{if } \mathsf{HaveRoute}(mynode, destip) \textbf{ then } \overline{\langle \mathsf{RouteOf}(mynode, destip), mymac\rangle}\,pkt\,.\,\mathbf{0} \\ \quad \textbf{else } \mathsf{LunARP}(mynode, mymac, destip) \end{array}$$

Fig. 7 IP transmission: if have route, send it to local forwarder, else ask for route

protocol [36, Definition 1]: if there is a path between two nodes, the protocol finds it, and it is possible to send packets along the path to the destination node.

The system to analyse consists of n nodes with their respective broadcast handler; node 0 attempts to transmit a packet to the IP address of node n.

$$\mathsf{Spec}_n(pkt, ip_0, \ldots, ip_n) \Leftarrow (\nu mac_0, \ldots, mac_n) \begin{pmatrix} \prod_{0 \le i \le n} \mathsf{BrdHandler}(\mathsf{node}_i, mac_i, ip_i) \\ \mid !\,\mathsf{IPtransmit}(\mathsf{node}_0, mac_0, ip_n, pkt) \end{pmatrix}$$

Theorem 18 *If Ψ connects node_0 and node_n via a node node_i (i.e. $\Psi \vdash \mathsf{node}_0 \dot{\succ} \mathsf{node}_i$ and $\Psi \vdash \mathsf{node}_i \dot{\succ} \mathsf{node}_n$), then*

$$\Psi \mid (\nu ip_0, \ldots, ip_n)\mathsf{Spec}_n(pkt, ip_0, \ldots, ip_n) \Longrightarrow \xrightarrow{\overline{\langle \mathsf{delivered}, \mathsf{node}_n \rangle} pkt} \Psi \mid (\nu ip_0, \ldots, ip_n)S$$

and $\mathcal{F}(S) \vdash \mathsf{HaveRoute}(\mathsf{node}_0, ip_n)$, where $\Longrightarrow$ stands for an interleaving of τ and broadcast output transitions.

Proof By following transitions.

The SPIN verification performed in [36] checks the same reachability property, for up to five nodes. Our analysis is valid for any n, but is limited to a configuration where the sender (node 0) and the receiver (node n) are only separated by a single node. This limitation is due to the labour of manually following transitions in a non-trivial specification. We are currently working on remedies for this: firstly by extending our symbolic semantics for psi-calculi [16], secondly by implementing the symbolic semantics in our tool for automatic verification [14], and thirdly and orthogonally, by implementing the LUNAR model in Isabelle/Nominal. These remedies are still work in progress. In the Isabelle approach, we hope to prove the following conjecture.

Conjecture 19 If Ψ connects node_0 and node_n via k proxy nodes $\mathsf{pn}_1, \ldots, \mathsf{pn}_k$, where $\{\mathsf{pn}_1, \ldots, \mathsf{pn}_k\} \subseteq \{\mathsf{node}_1, \ldots, \mathsf{node}_{n-1}\}$
(i.e. $\Psi \vdash \mathsf{node}_0 \dot{\succ} \mathsf{pn}_1, \mathsf{pn}_1 \dot{\succ} \mathsf{pn}_2, \ldots, \mathsf{pn}_{k-1} \dot{\succ} \mathsf{pn}_k, \mathsf{pn}_k \dot{\succ} \mathsf{node}_n$), then

$$\Psi \mid (\nu ip_0, \ldots, ip_n)\mathsf{Spec}_n(pkt, ip_0, \ldots, ip_n) \Longrightarrow \xrightarrow{\overline{\langle \mathsf{delivered}, \mathsf{node}_n \rangle} pkt} \Psi \mid (\nu ip_0, \ldots, ip_n)S$$

and $\mathcal{F}(S) \vdash \mathsf{HaveRoute}(\mathsf{node}_0, ip_n)$, where $\Longrightarrow$ stands for an interleaving of τ and broadcast output transitions.

The definition of BrdHandler illustrates a peculiarity of broadcast semantics: a reader well-versed in pi-calculus specifications with replication and recursion may consider a more concise variant of the definition using replication instead of recursion, e.g.

$$\mathsf{BrdHandler}'(mynode, mac, ip) \Leftarrow \\ !\,\underline{mynode}(\lambda s, t, r)\mathsf{RREQ}(s, t, r)\,.\,\mathsf{RreqHandler}(mynode, mac, ip, \mathsf{RREQ}(s, t, r))$$

However, when the input prefix is over a broadcast channel, as is the case here, the two are not equivalent since a single communication with $\mathsf{BrdHandler}'$ may result in arbitrarily many $\mathsf{RreqHandler}$ processes, while $\mathsf{BrdHandler}$ only results in one.

6 Related work

Process calculi with broadcast communication go back to the early 1980's. Milner developed SCCS [21] as a generalisation of CCS [20] to include multiway communication, of which broadcast can be seen as a special case. At the same time Austry and Boudol presented MEIJE [3] as a semantic basis for high-level hardware definition languages.

The first process calculus to seriously consider broadcast with an asynchronous parallel composition was CBS [26,27]. Its development is recorded in a series of papers, examining it from many perspectives. The main focus is on employing broadcast as a high level programming paradigm. CBS was later extended to the pi-calculus in the bπ formalism [8]. Here the broadcast communication channels are names that can be scoped and transmitted between agents. The main point of this work is to establish a separation result in expressiveness: in the pi-calculus, broadcast cannot be uniformly encoded by unicast.

Recent advances in wireless networks have created a renewed interest in the broadcast paradigm. The first process calculus with this in mind was probably CBS$^\sharp$ [23]. This is a development of CBS to include varying interconnection topologies. Input and output is performed on a universal ether and transitions are indexed with topologies which are sets of connectivity graphs; the connectivity graph matters for the input rule (reception is possible from any connected location). Main applications are on cryptography and routing protocols in mobile ad hoc wireless networks. CBS$^\sharp$ has been followed by several similar calculi. In CWS [19,17] the focus is on modelling low level interference. Communication actions have distinct beginnings and endings, and two actions may interfere if one begins before another has ended. The main result is an operational correspondence between a labelled semantics and a reduction semantics. CMAN [12] is a high level formalism extended with data types, just as the applied pi-calculus extends the original pi-calculus. Data can contain constructors and destructors. There are results on properties of weak bisimulation and an analysis of a cryptographic routing protocol. In the ω-calculus [31] emphasis is on expressing connectivity using sets of group names. An extension also includes separate unicast channels, making this formalism the first to accommodate both multicast and unicast in wireless networks. There are results about strong bisimulation and a verification of a mobile ad hoc network leader election protocol through weak bisimulation. RBPT [10] is similar and uses an alternative technique to represent topology changes, leading to smaller state spaces, and is also different in that it can accommodate an asymmetric neighbour relation (to model the fact that A can send to B but not the other way).

$bA\pi$ [13] is an extension of the applied pi-calculus [1] with broadcast, where connectivity information appears explicitly in the process terms and can change non-deterministically during execution. The claimed result of the paper is proving that a weak labelled bisimulation, for which connectivity is irrelevant, coincides with barbed equivalence. However, for the same reasons as in the applied pi-calculus (cf. [5]), labelled bisimilarity is not compositional in $bA\pi$, so the correspondence does not hold. A suggested fix is to remove communication of unicast channels from the calculus. We would finally mention CMN [18]. The claimed result is to compare two different kinds of semantics for a broadcast operation, but it is in error. The labelled transition semantics contains no rule for merging two inputs as in our BrMerge. As a consequence parallel composition fails to be associative. Consider the situation where P does an output and Q and R both do inputs. A broadcast communication involving all three agents can be derived from $(P|Q) \mid R$ but not from $P \mid (Q|R)$, since in the latter agent the component $Q|R$ cannot make an input involving both Q and R.

It is interesting to compare these formalisms and our broadcast psi from a few important perspectives. Firstly, the broadcast channels are explicitly represented in ω, bπ, CWS and CMN; they are mobile (in the sense that they can be transmitted) only in bπ. In ω, only unicast channels can be communicated. In broadcast psi, channels are represented as arbitrary mobile data terms which may contain any number of names. Secondly, the data transmitted in CMAN and $bA\pi$ is akin to the applied pi-calculus where data are drawn from an inductively defined set and contain names which may be scoped. In ω and bπ data are single names which may be scoped; in the other calculi data cannot contain scoped names. In broadcast psi data are arbitrary terms, drawn from a nominal set, and may include higher order objects as well as bound names. Finally, node mobility is represented explicitly as particular semantic rules in CMAN, CMN, $bA\pi$ and ω, and implicitly in the requirements of bisimulation in CBS$^\sharp$ and RBPT. In this respect broadcast psi calculi are similar to the latter: connectivity is determined by the assertions in the environment, and in a bisimulation these may change after each transition.

All calculi presented here use a kind of labelled transition semantics (LTS). bπ, $bA\pi$, CBS$^\sharp$, CWS and ω use it in conjunction with a structural congruence (SC), the rest (including broadcast psi) do not use a SC. In our experience SC is efficient in that the definitions become more compact and easy to understand, but introduces severe difficulties in making fully rigorous proofs. $bA\pi$, CWS, CMAN and CMN additionally use a reduction semantics using structural congruence (RS) and prove its agreement with the labelled semantics. Table 3 summarises some of the distinguishing features of calculi for wireless networks.

Finally, broadcast psi is different from the other calculi for wireless broadcast in that there is no stratification of the syntax into processes and networks. There is just the one kind of agent, suitable for expressing both processes operating in nodes and behaviours of entire networks. In contrast,

Calculus	Broadcast Channels	Scoped Data	Mobility	Semantics
$bA\pi$	-	term	in semantics	LTS+SC and RS
CBS$^\sharp$	-	-	in bisimulation	LTS+SC
CWS	constant	-	-	LTS+SC and RS
CMAN	-	term	in semantics	LTS and RS
CMN	name	-	in semantics	LTS and RS
ω	groups	name	in semantics	LTS+SC
RBPT	-	-	in bisimulation	LTS
Broadcast psi	term	term	in bisimulation	LTS

Table 3 Comparison of some process algebras for wireless broadcast.

the other calculi has one set of constructs to express processes and another to express networks, sometimes leading to duplication of effort (for example, there can be a parallel composition operator both at the process and network level). Our conclusion is that broadcast psi is conceptually simpler and more efficient for rigorous proofs, and yet more expressive.

7 Conclusion

We have extended the psi-calculi framework with broadcast communication, and formally proved using Isabelle/Nominal that the standard congruence and structural properties of bisimilarity hold also after the addition. We have shown how node mobility and network topology changes can be modelled using assertions. Since bisimilarity is closed under all assertions, two bisimilar processes are equivalent in all initial topologies and for all node mobility patterns. We demonstrated expressive power by modelling the LUNAR protocol for route discovery in wireless ad-hoc networks, and verified a basic correctness property of the protocol.

The proofs of the meta-theoretic results in Section 3.1 [28] are formally verified in the interactive theorem prover Isabelle/Nominal. The full formalisation of broadcast psi-calculi amounts to ca 33 000 lines of Isabelle code, of which about 21 000 lines are re-used from our earlier work [6].

The model of LUNAR is simplified for clarity and to make manual analysis more manageable. The simplifications are similar to those in the SPIN model by Wibling et al. [36], although we do not model timeouts. Their model [35] is ca 250 lines of SPIN code (excluding comments) while ours is approximately 30 lines. Our model could be improved at the cost of added complexity. For example, allowing broadcast channels to have non-empty support would let us hide broadcast actions, routing tables could be made local by including a scoped name per node, and route deletions could be modelled using generational mechanisms similar to Section 4.

We are currently working on extending the symbolic semantics for psi-calculi [16] with broadcast, and implementing the semantics in our tool

for automatic verification, the Psi-calculi Workbench [14]. We also plan to study weak bisimulation for the broadcast semantics. In order to model more aspects of wireless protocols, we would like to add general resource awareness (e.g. energy or time) to psi-calculi.

References

1. Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proceedings of POPL '01*, pages 104–115. ACM, 2001.
2. Johannes Åman Pohjola. Higher-order broadcast psi-calculus formalisation. `http://www.it.uu.se/research/group/mobility/theorem/higherorderbroadcast.tar.gz`, July 2013. Isabelle/HOL-Nominal formalisation of higher-order broadcast psi-calculi.
3. Didier Austry and Gérard Boudol. Algèbre de processus et synchronisation. *Theoretical Computer Science*, 30:91–131, 1984.
4. Jesper Bengtson. *Formalising process calculi*. PhD thesis, Uppsala University, June 2010.
5. 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.
6. 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. This is an extended version of [5].
7. 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, November 2011.
8. Cristian Ene and Traian Muntean. Expressiveness of point-to-point versus broadcast communications. In Gabriel Ciobanu and Gheorghe Paun, editors, *Proceedings of FCT*, volume 1684 of *Lecture Notes in Computer Science*, pages 258–268. Springer, 1999.
9. Murdoch Gabbay and Andrew Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2001.
10. 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.
11. Jens Chr. Godskesen and Sebastian Nanz. Mobility models and behavioural equivalence for wireless networks. In J. Field and V.T. Vasconcelos, editors, *Proc. Coordination Models and Languages*, number 5521 in Lecture Notes in Computer Science, pages 106–122. Springer-Verlag, 2009.
12. Jens Christian Godskesen. A calculus for mobile ad hoc networks. In Amy L. Murphy and Jan Vitek, editors, *Proceedings of COORDINATION 2007*, volume 4467 of *Lecture Notes in Computer Science*, pages 132–150. Springer-Verlag, 2007.
13. 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.

14. Ramūnas Gutkovas. *Exercising Psi-calculi: A Psi-calculi workbench.* M.Sc. thesis, Department of Information Technology, Uppsala University, June 2011.
15. Magnus Johansson. *Psi-calculi: a framework for mobile process calculi.* PhD thesis, Uppsala University, May 2010.
16. 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.
17. Ivan Lanese and Davide Sangiorgi. An operational semantics for a calculus for wireless systems. *Theoretical Computer Science*, 411(19):1928–1948, 2010.
18. Massimo Merro. An observational theory for mobile ad hoc networks (full version). *Journal of Information and Computation*, 207(2):194–208, 2009.
19. Nicola Mezzetti and Davide Sangiorgi. Towards a calculus for wireless systems. *Electronic Notes in Theoretical Computer Science*, 158:331–353, 2006.
20. Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.
21. Robin Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.
22. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part I/II. *Journal of Information and Computation*, 100:1–77, September 1992.
23. Sebastian Nanz and Chris Hankin. A framework for security analysis of mobile wireless networks. *Theoretical Computer Science*, 367(1-2):203–227, 2006.
24. Joachim Parrow, Johannes Borgström, Palle Raabjerg, and Johannes Åman Pohjola. Higher-order psi-calculi. *Mathematical Structures in Computer Science*, FirstView, June 2013.
25. A. M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186:165–193, 2003.
26. 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.
27. K. V. S. Prasad. A calculus of broadcasting systems. *Science of Computer Programming*, 25(2-3):285–327, 1995.
28. Palle Raabjerg and Johannes Åman Pohjola. Broadcast psi-calculus formalisation. `http://www.it.uu.se/research/group/mobility/theorem/broadcastpsi`, July 2011. Isabelle/HOL-Nominal formalisation of the definitions, theorems and proofs.
29. Davide Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8(5):447–479, 1998. An extended abstract appeared in the *Proceedings of MFCS '95*, LNCS 969: 479–488.
30. Davide Sangiorgi and David Walker. *The π-calculus: a Theory of Mobile Processes.* Cambridge University Press, 2001.
31. 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.
32. 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.
33. Christian F. Tschudin. Lightweight underlay network ad hoc routing (LUNAR) protocol. Internet Draft, Mobile Ad Hoc Networking Working Group, March 2004.

34. 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.
35. Oskar Wibling. SPIN and UPPAAL ad hoc routing protocol models. `http://www.it.uu.se/research/group/mobility/adhoc/gbt/other_examples`, 2004. Models of LUNAR scenarios used in [36].
36. Oskar Wibling, Joachim Parrow, and Arnold Pears. Automatized verification of ad hoc routing protocols. In David de Frutos-Escrig and Manuel Núñez, editors, *Formal Techniques for Networked and Distributed Systems (FORTE 2004)*, volume 3235 of *Lecture Notes in Computer Science*, pages 343–358. Springer-Verlag, 2004.

Chapter 3

Higher-order psi-calculi

Paper II

Higher-order psi-calculi

JOACHIM PARROW

JOHANNES BORGSTRÖM

PALLE RAABJERG

JOHANNES ÅMAN POHJOLA

Department of Information Technology, Uppsala University
Uppsala, Sweden[†]

Received 26 January 2011; Revised 21 January 2013

Psi-calculi is a parametric framework for extensions of the pi-calculus; in earlier work we have explored their expressiveness and algebraic theory. In this paper we consider higher-order psi-calculi through a technically surprisingly simple extension of the framework, and show how an arbitrary psi-calculus can be lifted to its higher-order counterpart in a canonical way. We illustrate this with examples and establish an algebraic theory of higher-order psi-calculi. The formal results are obtained by extending our proof repositories in Isabelle/Nominal.

Robin Milner in memoriam

Robin Milner pioneered developments in process algebras, higher-order formalisms, and interactive theorem provers. We hope he would have been pleased to see the different strands of his work combined in this way.

1. Introduction

Psi-calculi is a parametric framework for extensions of the pi-calculus to accommodate applications with complex data structures and high-level logics in a single general and parametric framework with machine-checked proofs. In earlier papers (BJPV09; BP09; JVP10; BJPV10) we have shown how psi-calculi can capture a range of phenomena such

[†] Email {joachim.parrow, johannes.borgstrom, palle.raabjerg, johannes.aman-pohjola}@it.uu.se.
Handled and communicated by Matthew Hennessy and Davide Sangiorgi. Partly supported by the Swedish Research Council grant UPMARC.

as cryptography and concurrent constraints, investigated strong and weak bisimulation, and provided a symbolic semantics. We claim that the theoretical development is more robust than in other calculi of comparable complexity, since we use a single inductive definition in the semantics and since we have checked most results in the theorem prover Isabelle/Nominal (Urb08).

In this paper we extend the framework to include higher-order agents, i.e., agents that can send agents as objects in communication. As an example of a traditional higher-order communication, the process $\overline{a}P\,.\,Q$ sends the process P along a and then continues as Q. A recipient looks like $a(X)\,.\,(R \mid X)$, receiving a process P and continuing as $R \mid P$, thus $\overline{a}P\,.\,Q \mid a(X)\,.\,(R \mid X)$ has a transition leading to $Q \mid (R \mid P)$. Higher-order computational paradigms date back to the lambda-calculus and many different formalisms are based on it. The first to study higher-order communication within a process calculus was probably Thomsen (Tho89; Tho93), and the area has been thoroughly investigated by Sangiorgi and others (San93; San96; San01; JR05; LPSS08; DHS09; LPSS10). There are several important problems related to type systems, to encoding higher-order behaviour using an ordinary calculus, and to the precise definition of bisimulation $\sim$. To appreciate the latter, consider an agent bisimulating $\overline{a}P\,.\,Q$. The normal definition would require the same action $\overline{a}P$ leading to an agent that bisimulates Q. In some circumstances this is too strong a requirement. Assume $P \sim P'$, then it is reasonable to let $\overline{a}P\,.\,Q \sim \overline{a}P'\,.\,Q$ even though they have different actions, since the only thing a recipient can do with the received object is to execute it, and here bisimilar agents are indistinguishable.

1.1. *Psi-calculi*

In the following we assume the reader to be acquainted with the basic ideas of process algebras based on the pi-calculus, and explain psi-calculi by a few simple examples. In a psi-calculus there are data terms $M, N, \ldots$ and we write $\overline{M}N\,.\,P$ to represent an agent sending the term N along the channel M (which is also a data term), continuing as the agent P. We write $\underline{K}(\lambda\widetilde{x})L\,.\,Q$ to represent an agent that can input along the channel K, receiving some object matching the pattern $\lambda\widetilde{x}L$. These two agents can interact under two conditions: first, the two channels must be *channel equivalent*, as defined by the channel equivalence predicate $M \leftrightarrows K$, and second, N must match the pattern, i.e., $N = L[\widetilde{x} := \widetilde{T}]$ for some sequence of terms $\widetilde{T}$. The receiving agent then continues as $Q[\widetilde{x} := \widetilde{T}]$.

Formally, a *transition* is of kind $\Psi \rhd P \xrightarrow{\alpha} P'$, meaning that when the environment contains the *assertion* Ψ the agent P can do an action α to become P'. An assertion embodies a collection of facts, to resolve among other things the channel equivalence predicate $\leftrightarrows$. To continue the example, we will have

$$\Psi \rhd \overline{M}N\,.\,P \mid \underline{K}(\lambda\widetilde{x})L\,.\,Q \xrightarrow{\tau} P \mid Q[\widetilde{x} := \widetilde{T}]$$

exactly when $N = L[\widetilde{x} := \widetilde{T}]$ and $\Psi \vdash M \leftrightarrows K$. The latter says that the assertion Ψ entails that M and K represent the same channel. In this way we can introduce an equational theory over a data structure for channels. Assertions are also used to resolve

the *conditions* φ in the **if** construct: we have that

$$\Psi \rhd \textbf{if } \varphi \textbf{ then } P \xrightarrow{\alpha} P'$$

if $\Psi \vdash \varphi$ and $\Psi \rhd P \xrightarrow{\alpha} P'$. In order to represent concurrent constraints and local knowledge, assertions can be used as agents: the agent $(\!|\Psi|\!)$ stands for an agent that asserts Ψ to its environment. For example, in

$$P \mid (\nu a)((\!|\Psi|\!) \mid Q)$$

the agent Q uses all entailments provided by Ψ, while P only uses those that do not contain the name a.

Assertions and conditions can, in general, form any logical theory. Also the data terms can be drawn from an arbitrary set. One of our major contributions has been to pinpoint the precise requirements on the data terms and logic for a calculus to be useful in the sense that the natural formulation of bisimulation satisfies the expected algebraic laws. It turns out that it is necessary to view the terms and logics as *nominal*. This means that there is a distinguished set of names, and for each term a well defined notion of *support*, intuitively corresponding to the names occurring in the term. Functions and relations must be equivariant, meaning that they treat all names equally. The logic must have a binary operator to combine assertions, corresponding to the parallel composition of processes, which must satisfy the axioms of an abelian monoid. Channel equivalence must be symmetric and transitive. In order to define the semantics of an input construct there must be a function to substitute terms for names, but it does not matter exactly what a substitution actually does to a term. These are all quite general requirements, and therefore psi-calculi accommodate a wide variety of extensions of the pi-calculus.

1.2. *Higher-order psi-calculi*

In one sense it is possible to have a naive higher-order psi-calculus without amending any definitions. Data can be *any* set satisfying the requirements mentioned above, in particular we may include the agents among the data terms. Thus the higher-order output and input exemplified above are already present. What is lacking is a construct to execute a received agent. A higher-order calculus usually includes the agent variables like X among the process constructors, making it possible to write e.g. $a(X).(X \mid R)$, which can receive any agent P and continue as $P \mid R$.

The route we shall take in this paper is more general and admits definitions of behaviours as recursive expressions, without a need to include a new syntactic category of process variables and higher-order substitution. Instead we introduce the notion of a *clause* $M \Leftarrow P$, meaning that the data term M can be used as a handle to invoke the behaviour of P in the agent **run** M. A sender can transmit the handle M in an ordinary output $\overline{a}M$ and a recipient can receive and run it as in $a(x).(\textbf{run } x \mid R)$.

Just like conditions, clauses are entailed by assertions. In that way we can use scoping to get local definitions of behaviour. For example, let $\{M_b \Leftarrow R\}$ be an assertion entailing

$M_b \Leftarrow R$ where b is in the support of M_b. Then, in

$$P \mid (\nu b)(Q \mid (\!|\{M_b \Leftarrow R\}|\!))$$

the agent Q can use the clause but P cannot, since it is out of the scope of b.

Formally, the clauses do not represent an extension of the psi framework since they can be included among the conditions. The only formal extension is the new agent form *invocation* $\mathbf{run}\ M$, to invoke an agent represented by M, with the corresponding rule of action

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

In this way we can perform higher order communication. In place of $\overline{a}P \,.\, Q \mid a(X) \,.\, (X \mid R)$ we write

$$(\nu b)(\overline{a}M_b \,.\, Q \mid (\!|\{M_b \Leftarrow P\}|\!)) \mid a(x) \,.\, (\mathbf{run}\ x \mid R)$$

Until the left hand component interacts along a, the scope of b prohibits the environment to use the clause. After the interaction this scope is extruded and the recipient can use M_b to invoke the received process. For example, let $P = \alpha \,.\, P'$. The communication results in a τ-transition, which can be followed by an invocation:

$$\begin{array}{lll} (\nu b)(\overline{a}M_b \,.\, Q \mid (\!|\{M_b \Leftarrow \alpha \,.\, P'\}|\!)) & \mid & a(x) \,.\, (\mathbf{run}\ x \mid R) \xrightarrow{\tau} \\ (\nu b)(Q \mid (\!|\{M_b \Leftarrow \alpha \,.\, P'\}|\!)) & \mid & \mathbf{run}\ M_b \mid R) \xrightarrow{\alpha} \\ (\nu b)(Q \mid (\!|\{M_b \Leftarrow \alpha \,.\, P'\}|\!)) & \mid & P' \mid R) \end{array}$$

In this way we send not the agent itself but rather a way to make it appear. This is reminiscent of the encoding of higher-order calculi into their first order counterparts:

$$(\nu b)\overline{a}b \,.\, (Q \mid !\, b \,.\, P) \mid a(x) \,.\, R$$

Here the trigger b is used in a normal communication to activate P. A purely syntactic difference is that in this encoding, the invocation will trigger an execution of P in the place from which it was sent, whereas in higher-order psi-calculi, the invocation rule means that P will execute in the place where it is invoked. Therefore, when M_b is a handle for P its support must include that of P; this makes sure that scope extrusions are enforced when a name in the support of P is restricted and M_b sent out of its scope.

In one important respect our work differs from previous work on higher-order calculi. Existing work (that we know of) explores fundamental constructions in extremely parsimonious calculi, to determine exactly what can be encoded with the higher-order paradigm or exactly how that can be encoded. Our aim, on the other side, is to extend a very rich framework, already containing arbitrarily advanced data types, with a higher-order construct that facilitates the natural representation of applications.

1.3. *Exposition*

In the next section we recapitulate the definitions of psi-calculi from (BJPV09; BJPV11). We give all definitions to make the paper formally self contained, referring to our earlier work for motivation and intuition. In Section 3 we present the smooth extensions to

higher-order psi-calculi, namely the clauses and the invocation rule. This provides a general framework and admits many different languages for expressing the clauses. As an example we show how to express process abstractions, and how we can construct a canonical higher-order calculus from a first-order one, by just adding a higher-order component to the assertions. In Section 4 we explore the algebraic theory of bisimulation. We inherit the definitions verbatim from first-order psi-calculi and all properties will still hold; moreover we show that Sum and Replication can be directly represented through higher-order constructs. We explore a slightly amended bisimulation definition which is more natural in a higher-order context. All proofs of all theorems presented in this paper have been formalised in the interactive theorem prover Isabelle, and we comment briefly on our experiences. We end with a comparison of alternative bisimulations and a conclusion with ideas for further work.

2. Psi-calculi

This section recapitulates the relevant parts of (BJPV09; BJPV11).

We assume a countably infinite set of atomic *names* $\mathcal{N}$ ranged over by $a, \ldots, z$. A *nominal set* (Pit03; GP01) is a set equipped with *name swapping* functions written $(a\ b)$, for any names a, b. An intuition is that for any member X it holds that $(a\ b) \cdot X$ is X with a replaced by b and b replaced by a. Formally, a name swapping is any function satisfying certain natural axioms such as $(a\ b) \cdot ((a\ b) \cdot X) = X$. One main point of this is that even though we have not defined any particular syntax we can define what it means for a name to "occur" in an element: it is simply that it can be affected by swappings. The names occurring in this way in an element X constitute the *support* of X, written $\mathrm{n}(X)$. We write $a\#X$, pronounced "a is fresh for X", for $a \notin \mathrm{n}(X)$. If A is a set or a sequence of names we write $A\#X$ to mean $\forall a \in A\ .\ a\#X$. We require all elements to have finite support, i.e., $\mathrm{n}(X)$ is finite for all X. A function f is *equivariant* if $(a\ b) \cdot f(X) = f((a\ b) \cdot X)$ holds for all X, and similarly for functions and relations of any arity. Intuitively, this means that all names are treated equally.

In the following $\tilde{a}$ means a finite sequence of distinct names, $a_1, \ldots, a_n$. The empty sequence is written ϵ and the concatenation of $\tilde{a}$ and $\tilde{b}$ is written $\tilde{a}\tilde{b}$. When occurring as an operand of a set operator, $\tilde{a}$ means the corresponding set of names $\{a_1, \ldots, a_n\}$. We also use sequences of other nominal sets in the same way, except that we then do not require that all elements in the sequence are pairwise different. We use A, B to range over finite sets of names.

A *nominal datatype* is a nominal set together with a set of equivariant functions on it. In particular we shall consider substitution functions that substitute elements for names. If X is an element of a datatype, the *substitution* $X[\tilde{a} := \tilde{Y}]$ is an element of the same datatype as X. Substitution is required to satisfy a kind of alpha-conversion law: if $\tilde{b}\#X, \tilde{a}$ then $X[\tilde{a} := \tilde{T}] = ((\tilde{b}\ \tilde{a}) \cdot X)[\tilde{b} := \tilde{T}]$; here it is implicit that $\tilde{a}$ and $\tilde{b}$ have the same length, and $(\tilde{a}\ \tilde{b})$ swaps each element of $\tilde{a}$ with the corresponding element of $\tilde{b}$. The name preservation law $\tilde{a} \subseteq \mathrm{n}(N) \wedge b \in \mathrm{n}(\tilde{M}) \implies b \in \mathrm{n}(N[\tilde{a} := \tilde{M}])$ will be important for some substitutions. Apart from these laws we do not require any particular behaviour of substitution.

Formally, a psi-calculus is defined by instantiating three nominal datatypes and four operators:

Definition 1 (Psi-calculus parameters)**.** *A psi-calculus requires the three (not necessarily disjoint) nominal datatypes:*

$\mathbf{T}$	*the (data) terms, ranged over by* M, N
$\mathbf{C}$	*the conditions, ranged over by* φ
$\mathbf{A}$	*the assertions, ranged over by* Ψ

and the four equivariant operators:

$\leftrightarrow : \mathbf{T} \times \mathbf{T} \to \mathbf{C}$	*Channel Equivalence*
$\otimes : \mathbf{A} \times \mathbf{A} \to \mathbf{A}$	*Composition*
$\mathbf{1} : \mathbf{A}$	*Unit*
$\vdash \subseteq \mathbf{A} \times \mathbf{C}$	*Entailment*

and substitution functions $[\widetilde{a} := \widetilde{M}]$*, substituting terms for names, on each of* $\mathbf{T}$*,* $\mathbf{C}$ *and* $\mathbf{A}$*, where the substitution function on* $\mathbf{T}$ *satisfies name preservation.*

The binary functions above will be written in infix. Thus, if M and N are terms then $M \leftrightarrow N$ is a condition, pronounced "M and N are channel equivalent" and if Ψ and Ψ' are assertions then so is $\Psi \otimes \Psi'$. Also we write $\Psi \vdash \varphi$, pronounced "Ψ entails φ", for $(\Psi, \varphi) \in \vdash$.

Definition 2 (assertion equivalence)**.** *Two assertions are* equivalent*, written* $\Psi \simeq \Psi'$*, if for all* φ *we have that* $\Psi \vdash \varphi \Leftrightarrow \Psi' \vdash \varphi$*.*

The requisites on valid psi-calculus parameters are:

Definition 3 (Requisites on valid psi-calculus parameters)**.**

Channel Symmetry:	$\Psi \vdash M \leftrightarrow N \implies \Psi \vdash N \leftrightarrow M$
Channel Transitivity:	$\Psi \vdash M \leftrightarrow N \wedge \Psi \vdash N \leftrightarrow L \implies \Psi \vdash M \leftrightarrow L$
Compositionality:	$\Psi \simeq \Psi' \implies \Psi \otimes \Psi'' \simeq \Psi' \otimes \Psi''$
Identity:	$\Psi \otimes \mathbf{1} \simeq \Psi$
Associativity:	$(\Psi \otimes \Psi') \otimes \Psi'' \simeq \Psi \otimes (\Psi' \otimes \Psi'')$
Commutativity:	$\Psi \otimes \Psi' \simeq \Psi' \otimes \Psi$

Our requisites on a psi-calculus are that the channel equivalence is a partial equivalence relation, that $\otimes$ is compositional, and that the equivalence classes of assertions form an abelian monoid.

Definition 4 (Frame)**.** *A* frame *is of the form* $(\nu\widetilde{b})\Psi$ *where* $\widetilde{b}$ *is a sequence of names that bind into the assertion* Ψ*. We identify alpha variants of frames.*

We use F, G to range over frames. Since we identify alpha variants we can choose the bound names arbitrarily. We write just Ψ for $(\nu\epsilon)\Psi$ when there is no risk of confusing a frame with an assertion, and $\otimes$ to mean composition on frames defined by $(\nu\widetilde{b}_1)\Psi_1 \otimes (\nu\widetilde{b}_2)\Psi_2 = (\nu\widetilde{b}_1\widetilde{b}_2)\Psi_1 \otimes \Psi_2$ where $\widetilde{b}_1 \# \widetilde{b}_2, \Psi_2$ and vice versa. We write $(\nu c)((\nu\widetilde{b})\Psi)$ to mean $(\nu c\widetilde{b})\Psi$.

Definition 5. *We define* $F \vdash \varphi$ *to mean that there exists an alpha variant* $(\nu\widetilde{b})\Psi$ *of* F

such that $\widetilde{b}\#\varphi$ *and* $\Psi \vdash \varphi$*. We also define* $F \simeq G$ *to mean that for all* φ *it holds that* $F \vdash \varphi$ *iff* $G \vdash \varphi$*.*

Definition 6 (psi-calculus agents)**.** *Given valid psi-calculus parameters as in Definitions 1 and 3, the psi-calculus* agents $\mathbf{P}$*, ranged over by* $P, Q, \ldots$*, are of the following forms.*

$\mathbf{0}$	Nil
$\overline{M}N\,.\,P$	Output
$\underline{M}(\lambda\widetilde{x})N\,.\,P$	Input
$\mathbf{case}\ \varphi_1 : P_1\ [\!]\ \cdots\ [\!]\ \varphi_n : P_n$	Case
$(\nu a)P$	Restriction
$P \mid Q$	Parallel
$!P$	Replication
$(\!\vert\Psi\vert\!)$	Assertion

Restriction binds a *in* P *and Input binds* $\tilde{x}$ *in both* N *and* P*. We identify alpha equivalent agents. An assertion is* guarded *if it is a subterm of an Input or Output. An agent is* assertion guarded *if it contains no unguarded assertions. An agent is* well formed *if in* $\underline{M}(\lambda\widetilde{x})N.P$ *it holds that* $\widetilde{x} \subseteq \mathrm{n}(N)$ *is a sequence without duplicates, that in a replication* $!P$ *the agent* P *is assertion guarded, and that in* $\mathbf{case}\ \varphi_1 : P_1\ [\!]\ \cdots\ [\!]\ \varphi_n : P_n$ *the agents* P_i *are assertion guarded.*

In the Output and Input forms M is called the subject and N the object. Output and Input are similar to those in the pi-calculus, but arbitrary terms can function as both subjects and objects. In the input $\underline{M}(\lambda\widetilde{x})N.P$ the intuition is that the pattern $(\lambda\widetilde{x})N$ can match any term obtained by instantiating $\widetilde{x}$, e.g., $\underline{M}(\lambda x, y)f(x, y).P$ can only communicate with an output $\overline{M}f(N_1, N_2)$ for some data terms N_1, N_2. This can be thought of as a generalization of the polyadic pi-calculus where the patterns are just tuples of (distinct, bound) names. Another significant extension is that we allow arbitrary data terms also as communication channels. Thus it is possible to include functions that create channels.

The **case** construct as expected works by behaving as one of the P_i for which the corresponding φ_i is true. $\mathbf{case}\ \varphi_1 : P_1\ [\!]\ \cdots\ [\!]\ \varphi_n : P_n$ is sometimes abbreviated as $\mathbf{case}\ \widetilde{\varphi} : \widetilde{P}$, or if $n = 1$ as $\mathbf{if}\ \varphi_1\ \mathbf{then}\ P_1$. In psi-calculi where a condition $\top$ exists such that $\Psi \vdash \top$ for all Ψ we write $P + Q$ to mean $\mathbf{case}\ \top : P\ [\!]\ \top : Q$.

Input subjects are underlined to facilitate parsing of complicated expressions; in simple cases we often omit the underline. In the traditional pi-calculus terms are just names and its input construct $a(x)\,.\,P$ can be represented as $\underline{a}(\lambda x)x.P$. In some of the examples to follow we shall use the simpler notation $a(x)\,.\,P$ for this input form, and sometimes we omit a trailing $\mathbf{0}$, writing just $\overline{M}N$ for $\overline{M}N\,.\,\mathbf{0}$.

In the standard pi-calculus the transitions from a parallel composition $P \mid Q$ can be uniquely determined by the transitions from its components, but in psi-calculi the situation is more complex. Here the assertions contained in P can affect the conditions tested in Q and vice versa. For this reason we introduce the notion of the *frame of an agent* as the combination of its top level assertions, retaining all the binders. It is precisely this that can affect a parallel agent.

Definition 7 (Frame of an agent)**.** *The* frame $\mathcal{F}(P)$ of an agent P *is defined inductively as follows:*

$$\mathcal{F}(\mathbf{0}) = \mathcal{F}(\underline{M}(\lambda\tilde{x})N.P) = \mathcal{F}(\overline{M}N.P) = \mathcal{F}(\mathbf{case}\ \widetilde{\varphi} : \widetilde{P}) = \mathcal{F}(!P) = \mathbf{1}$$
$$\mathcal{F}((\!|\Psi|\!)) = \Psi$$
$$\mathcal{F}(P \mid Q) = \mathcal{F}(P) \otimes \mathcal{F}(Q)$$
$$\mathcal{F}((\nu b)P) = (\nu b)\mathcal{F}(P)$$

An agent where all assertions are guarded thus has a frame equivalent to $\mathbf{1}$. In the following we often write $(\nu\widetilde{b_P})\Psi_P$ for $\mathcal{F}(P)$, but note that this is not a unique representation since frames are identified up to alpha equivalence.

The actions α that agents can perform are of three kinds: output, input, and the silent action τ. The input actions are of the early kind, meaning that they contain the object received. The operational semantics consists of transitions of the form $\Psi \rhd P \xrightarrow{\alpha} P'$. This transition intuitively means that P can perform an action α leading to P', in an environment that asserts Ψ.

Definition 8 (Actions)**.** *The* actions *ranged over by* α, β *are of the following three kinds:*

$\overline{M}(\nu\tilde{a})N$	*Output, where* $\tilde{a} \subseteq \mathrm{n}(N)$
$\underline{M}\,N$	*Input*
τ	*Silent*

For actions we refer to M as the *subject* and N as the *object*. We define $\mathrm{bn}(\overline{M}(\nu\tilde{a})N) = \tilde{a}$, and $\mathrm{bn}(\alpha) = \emptyset$ if α is an input or τ. We also define $\mathrm{n}(\tau) = \emptyset$ and $\mathrm{n}(\alpha) = \mathrm{n}(N) \cup \mathrm{n}(M)$ if α is an output or input. As in the pi-calculus, the output $\overline{M}(\nu\tilde{a})N$ represents an action sending N along M and opening the scopes of the names $\tilde{a}$. Note in particular that the support of this action includes $\tilde{a}$. Thus $\overline{M}(\nu a)a$ and $\overline{M}(\nu b)b$ are different actions.

Definition 9 (Transitions)**.** *The transitions are defined inductively in Table 1. We write* $P \xrightarrow{\alpha} P'$ *to mean* $\mathbf{1} \rhd P \xrightarrow{\alpha} P'$*. In* In *the substitution is defined by induction on agents, using substitution on terms, assertions and conditions for the base cases and avoiding captures through alpha-conversion in the standard way.*

Both agents and frames are identified by alpha equivalence. This means that we can choose the bound names fresh in the premise of a rule. In a transition the names in $\mathrm{bn}(\alpha)$ count as binding into both the action object and the derivative, and transitions are identified up to alpha equivalence. This means that the bound names can be chosen fresh, substituting each occurrence in both the object and the derivative. This is the reason why $\mathrm{bn}(\alpha)$ is in the support of the output action: otherwise it could be alpha-converted in the action alone. Also, for the side conditions in Scope and Open it is important that $\mathrm{bn}(\alpha) \subseteq \mathrm{n}(\alpha)$. In rules Par and Com, the freshness conditions on the involved frames will ensure that if a name is bound in one agent its representative in a frame is distinct from names in parallel agents, and also (in Par) that it does not occur on the transition label.

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

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

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

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

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

Table 1. Operational semantics. Symmetric versions of Com and Par are elided. In the rule Com we assume that $\mathcal{F}(P) = (\nu\widetilde{b}_P)\Psi_P$ and $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$ 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 $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$ 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.

3. Higher-order Psi-calculi

We now proceed to formalise the extension to higher-order psi-calculi described in the introduction. Technically this means adopting a specific form of assertion and condition and extending the framework with a construct $\mathbf{run}\ M$.

3.1. *Basic definitions*

In a *higher-order* psi-calculus we use one particular nominal datatype of *clauses*:

$$\mathbf{Cl} = \{M \Leftarrow P :\ M \in \mathbf{T} \wedge P \in \mathbf{P} \wedge \mathrm{n}(M) \supseteq \mathrm{n}(P) \wedge P \text{ assertion guarded}\}$$

and the entailment relation is extended to $\vdash \subseteq \mathbf{A} \times (\mathbf{C} \uplus \mathbf{Cl})$, where we write $\Psi \vdash \varphi$ for $\Psi \vdash (0, \varphi)$ and $\Psi \vdash M \Leftarrow P$ for $\Psi \vdash (1, M \Leftarrow P)$.

We amend the definition of assertion equivalence to mean that the assertions entail the same conditions *and clauses*. This extension is not formally necessary since we could instead adjoin $\mathbf{Cl}$ to the conditions, but calling $M \Leftarrow P$ a "condition" is a misnomer we want to avoid.

Definition 10 (Higher-order agents)**.** *The higher-order agents in a psi-calculus extend those of an ordinary calculus with one new kind of agent:*

$$\mathbf{run}\ M \quad \textit{Invoke an agent for which } M \textit{ is a handle}$$

We define $\mathcal{F}(\mathbf{run}\ M)$ *to be* $\mathbf{1}$*.*

Finally there is the new transition rule:

Definition 11 (Higher-order transitions)**.** *The transitions in a higher-order psi-calculus are those that can be derived from the rules in Table 1 plus the one additional rule*

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

We are free to choose any language we want for the assertions as long as the requisites in Definition 3 hold. Let us in a few simple examples consider a language where assertions are finite sets of clauses and composition $\otimes$ corresponds to union.

A higher-order communication is simply an instance of ordinary communication inferred with the Com rule. As an example, if $P \Leftarrow P$ is entailed by all assertions, i.e., an agent is always a handle for itself,

$$\overline{a}P \,.\, Q \mid a(x) \,.\, (\mathbf{run}\ x \mid R) \xrightarrow{\tau} Q \mid \mathbf{run}\ P \mid R[x := P]$$

This corresponds to sending the program code. A recipient can both execute it and use it as data. For example R can be $\mathbf{if}\ x = P'\ \mathbf{then}\ \ldots$, checking if the received P is syntactically the same as some other agent P'. To prevent the latter, instead send a handle M to represent P:

$$\begin{array}{l} \overline{a}M \,.\, Q \mid (\!|\{M \Leftarrow P\}|\!) \mid a(x) \,.\, (\mathbf{run}\ x \mid R) \quad \xrightarrow{\tau} \\ \qquad Q \mid (\!|\{M \Leftarrow P\}|\!) \mid \qquad (\mathbf{run}\ M \mid R[x := M]) \end{array}$$

In Section 3.3 we shall define canonical higher-order calculi; in these receiving a handle M means that the code of P cannot be directly inspected: all that can be done with the process P is to execute it. Thus our semantics gives a uniform way to capture both a direct higher-order communication, where the recipient gets access to the code, and an indirect where the recipient only obtains a possibility to execute it. This is different from all existing higher-order semantics known to us, and reminiscent of the way encryption is represented in psi-calculi in (BJPV11).

For another example, consider that there are shared private names between a process P being sent and its original environment Q:

$$(\nu b)\overline{a}M \,.\, (Q \mid (\!|\{M \Leftarrow P\}|\!)) \xrightarrow{\alpha} Q \mid (\!|\{M \Leftarrow P\}|\!)$$

If $b \in \mathrm{n}(P)$ then also $b \in \mathrm{n}(M)$, and hence b is extruded whenever M is sent, i.e. $\alpha = \overline{a}(\nu b)M$. This means that wherever M is received the shared link b to Q will still work.

As an example of an invocation, consider the following transition:

$$\begin{array}{lll} \mathbf{1} \rhd & (\nu b)(Q \mid (\!|\{M_b \Leftarrow \alpha \,.\, P\}|\!)) & \mid \quad (\nu c)(\mathbf{run}\ M_b \mid R)) \quad \xrightarrow{\alpha} \\ & (\nu b)(Q \mid (\!|\{M_b \Leftarrow \alpha \,.\, P\}|\!)) & \mid \quad (\nu c)(P \mid R)) \end{array}$$

A derivation of this transition uses the Invocation rule

$$\frac{\{M_b \Leftarrow \alpha \,.\, P\} \vdash M_b \Leftarrow \alpha \,.\, P \qquad \{M_b \Leftarrow \alpha \,.\, P\} \rhd \alpha \,.\, P \xrightarrow{\alpha} P}{\{M_b \Leftarrow \alpha \,.\, P\} \rhd \mathbf{run}\, M_b \xrightarrow{\alpha} P}$$

Through PAR and SCOPE we get

$$\{M_b \Leftarrow \alpha \,.\, P\} \rhd (\nu c)(\mathbf{run}\ M_b \mid R) \xrightarrow{\alpha} (\nu c)(P \mid R)$$

The conditions on SCOPE require $c\#\alpha$ and also $c\#\{M_b \Leftarrow \alpha \,.\, P\}$; the latter implies $c\#P$. Through PAR:

$$\mathbf{1} \rhd (\!|\{M_b \Leftarrow \alpha \,.\, P\}|\!) \mid (\nu c)(\mathbf{run}\ M_b \mid R) \xrightarrow{\alpha} (\!|\{M_b \Leftarrow \alpha \,.\, P\}|\!) \mid (\nu c)(P \mid R)$$

and finally through PAR and SCOPE again we get the desired transition.

Example: Representing non-determinism Since the same handle can be used to invoke different agents, we can represent nondeterminism. Instead of $P + Q$ we can choose $a\#P, Q$ and write

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

We can represent the **case** construct by a unary **if then** as follows: In place of $\mathbf{case}\ \varphi_1 : P_1 \,[\!]\, \cdots \,[\!]\, \varphi_n : P_n$ we write (choosing $a\#P_i, \varphi_i$)

$$(\nu a)(\mathbf{run}\ M_a \mid (\!|\{M_a \Leftarrow \mathbf{if}\ \varphi_1\ \mathbf{then}\ P_1, \cdots, M_a \Leftarrow \mathbf{if}\ \varphi_n\ \mathbf{then}\ P_n\}|\!))$$

One intuitive reason this works is that an invocation only occurs when a transition happens.

Representing fixpoints and replication Some versions of CCS and similar calculi use a special fixpoint operator $\mathbf{fix}\, X \,.\, P$, where X is an agent variable, with the rule of action

$$\textsc{Fix}\ \frac{P[X := \mathbf{fix}\, X \,.\, P] \xrightarrow{\alpha} P'}{\mathbf{fix}\, X \,.\, P \xrightarrow{\alpha} P'}$$

The substitution in the premise is of a higher-order kind, replacing an agent variable by an agent. We can represent this as follows. Let the agent variable X be represented by a term M_a with support $\mathrm{n}(P) \cup \{a\}$ where $a\#P$. Then $\mathbf{fix}\, X \,.\, P$ behaves exactly as

$$(\nu a)(\mathbf{run}\ M_a \mid (\!|\{M_a \Leftarrow P[X := \mathbf{run}\ M_a]\}|\!))$$

In this way, replication $!P$ can be seen as the fixpoint $\mathbf{fix}\, X \,.\, P|X$, and replication can be represented as

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

which is reminiscent of the encoding of replication in the higher-order pi-calculus. In Section 4.2 we shall formulate the precise conditions on higher-order psi-calculi where these encodings are possible.

3.2. *Process abstractions and parameters*

For a higher-order psi-calculus to be useful there should be a high level language for expressing clauses. This can be achieved by choosing the psi-calculus parameters in a suitable way, without any further extension of our framework.

Here is one example of such a language which accommodates process abstractions and application in the standard way. It assumes a binary operator on terms $\bullet\langle\bullet\rangle$; in other words, if M and N are terms then so is $M\langle N\rangle$.

Definition 12. *A* parametrised clause *is of the form* $M(\lambda\tilde{x})N \Leftarrow P$, *with* $\tilde{x}$ *binding in* N *and* P. *The corresponding definition of entailment is*

$$M(\lambda\tilde{x})N \Leftarrow P \in \Psi \quad \Longrightarrow \quad \Psi \vdash M\langle N[\tilde{x} := \tilde{L}]\rangle \Leftarrow P[\tilde{x} := \tilde{L}]$$

for all $\tilde{L}$ *of the same length as* $\tilde{x}$ *such that* $\mathrm{n}(M\langle N[\tilde{x} := \tilde{L}]\rangle) \supseteq \mathrm{n}(P[\tilde{x} := \tilde{L}])$.

With parametrised clauses we can formulate recursive behaviour in a convenient way, since an invocation of M can be present in P. Consider for example the definitions for an agent enacting a stack. The parameter of the stack is its current content, represented as a list, and its behaviour is given by the two parametrised clauses

$$\textsc{Stack}(\lambda x)x \Leftarrow \underline{\mathit{Push}}(\lambda y)y \,.\, \mathbf{run}\ \textsc{Stack}\langle \mathrm{cons}(y, x)\rangle$$
$$\textsc{Stack}(\lambda x, y)\mathrm{cons}(x, y) \Leftarrow \overline{\mathit{Pop}}\, x \,.\, \mathbf{run}\ \textsc{Stack}\langle y\rangle$$

We use different fonts to distinguish different kinds of terms; formally this has no consequence but it makes the agents easier to read. Stack, *Push* and *Pop* are just terms, the first representing a handle and the other communication channels. The support of *Push* and *Pop* must either be added to the formal parameter in the clauses of Stack or to the support of the term Stack itself, to satisfy the criterion on the names in clauses. Finally, $\mathrm{cons}(M, N)$ is a term representing the usual list constructor.

Note that a non-empty stack matches both clauses. As an example, let Ψ contain these two parametrised clauses and let nil be a term representing the empty list. For $x = \mathrm{nil}$ we get

$$\Psi \vdash \textsc{Stack}\langle \mathrm{nil}\rangle \Leftarrow \underline{\mathit{Push}}(\lambda y)y \,.\, \mathbf{run}\ \textsc{Stack}\langle \mathrm{cons}(y, \mathrm{nil})\rangle$$

and thus

$$\Psi \rhd \mathbf{run}\ \textsc{Stack}\langle \mathrm{nil}\rangle \xrightarrow{\underline{\mathit{Push}}\, M} \mathbf{run}\ \textsc{Stack}\langle \mathrm{cons}(M, \mathrm{nil})\rangle$$

and this agent can continue in two different ways: one is

$$\Psi \rhd \mathbf{run}\ \textsc{Stack}\langle \mathrm{cons}(M, \mathrm{nil})\rangle \xrightarrow{\underline{\mathit{Push}}\, M'} \mathbf{run}\ \textsc{Stack}\langle \mathrm{cons}(M', \mathrm{cons}(M, \mathrm{nil}))\rangle$$

and the other is, using the second clause with $x = M$ and $y = \mathrm{nil}$:

$$\Psi \rhd \mathbf{run}\ \textsc{Stack}\langle \mathrm{cons}(M, \mathrm{nil})\rangle \xrightarrow{\overline{\mathit{Pop}}\, M} \mathbf{run}\ \textsc{Stack}\langle \mathrm{nil}\rangle$$

This kind of recursion is often a very convenient way to model iterative behaviour. The earliest process algebras such as CCS use it extensively in applications. We say that a clause $M \Leftarrow P$ is *universal* if $\Psi \vdash M \Leftarrow P$ for all Ψ. In order to represent recursion in the CCS way it is enough to consider universal clauses. In higher-order psi-calculi we can additionally use local definitions, since they reside in assertions where their names can be given local scope, and gain the possibility to transmit the agents by sending the handles like Stack. We can represent a "stack factory" which repeatedly sends out the handle to recipients as $!\overline{a}\,\textsc{Stack} \,.\, \mathbf{0}$. Each recipient will get its own stack, which will develop independently of other copies. As formulated here all stacks will use the same channels

Push and *Pop*; private channels can be achieved by including their names in the formal parameters of the clauses:

$$\textsc{Stack}(\lambda i, o, x)i, o, x \Leftarrow \underline{i}(\lambda y)y \,.\, \mathbf{run}\ \textsc{Stack}\langle i, o, \mathrm{cons}(y, x)\rangle$$
$$\textsc{Stack}(\lambda i, o, x, y)i, o, \mathrm{cons}(x, y) \Leftarrow \overline{o}x \,.\, \mathbf{run}\ \textsc{Stack}\langle i, o, y\rangle$$

Here each recipient must supply the terms to use for input and output channels as formal parameters when invoking Stack. An alternative is to let each Stack carry those terms and in an initial interaction reveal them to the recipient.

$$\textsc{StackStart} \Leftarrow \overline{c}\langle \mathit{Push}, \mathit{Pop}\rangle \,.\, \mathbf{run}\ \textsc{Stack}\langle(\mathit{Push}, \mathit{Pop}, \mathrm{nil})\rangle$$

Here the support of *Push* and *Pop*, call it $\tilde{b}$, must be included in the support of StackStart. A recipient of StackStart must begin by receiving, along c, the terms for interacting with the stack. In the stack factory, there is then a choice of where to bind $\tilde{b}$.

$$(\nu\tilde{b})!\overline{a}\,\textsc{StackStart} \,.\, \mathbf{0}$$

represents a stack factory that produces stacks all working on the *same* private channels, whereas

$$!(\nu\tilde{b})\overline{a}\,\textsc{StackStart} \,.\, \mathbf{0}$$

represents a factory producing stacks all working on *different* private channels.

3.3. *Canonical higher-order instances*

Given an arbitrary first-order psi-calculus $\mathcal{C}$, we here show how to lift it to a higher-order psi-calculus $\mathcal{H}(\mathcal{C})$ in a systematic way. In our earlier work (BJPV09) we have demonstrated psi-calculi corresponding to the pi-calculus, the polyadic pi-calculus, and explicit fusions; we have also given calculi that capture the same phenomena as the applied pi-calculus and concurrent constraints. Out of these, only the pi-calculus has until now been given in a higher-order variant. Our result here is to lift all of them in one go.

The main idea is to build $\mathcal{H}(\mathcal{C})$ by starting from $\mathcal{C}$ and adding the parametrised clauses described above. An assertion of $\mathcal{H}(\mathcal{C})$ thus is a pair where the first component is an assertion in $\mathcal{C}$ and the second component is a finite set of parametrised clauses. Composition of assertions is defined component-wise, with identity element $(\mathbf{1}, \emptyset)$. We finally define a notion of substitution on sets of process abstractions, which we do point-wise and capture-avoiding, using the substitution functions of $\mathcal{C}$.

Parametrised clauses use a binary function on terms $\bullet\langle\bullet\rangle : \mathbf{T} \times \mathbf{T} \to \mathbf{T}$. We could choose this function to be standard pairing, if present in the term language, but our result holds for any such equivariant function.

Definition 13 (Canonical higher-order psi-calculi)**.** *Let a psi-calculus $\mathcal{C}$ be defined by the parameters $\mathbf{T}, \mathbf{C}, \mathbf{A}, \leftrightarrows, \otimes, \mathbf{1}, \vdash$. Let $\mathbf{S}$ be the set of finite sets of parametrised clauses as defined above. The canonical higher-order psi-calculus $\mathcal{H}(\mathcal{C})$ extends $\mathcal{C}$ by adding the* **run** *M agent and its semantic rule, and is defined by the parameters $\mathbf{T}_{\mathcal{H}}, \mathbf{C}_{\mathcal{H}}, \mathbf{A}_{\mathcal{H}}, \leftrightarrows_{\mathcal{H}}$*

$, \otimes_{\mathcal{H}}, \mathbf{1}_{\mathcal{H}}, \vdash_{\mathcal{H}}$ *where*

$$\begin{array}{rcl}
\mathbf{T}_{\mathcal{H}} & = & \mathbf{T} \\
\mathbf{C}_{\mathcal{H}} & = & \mathbf{C} \\
\mathbf{A}_{\mathcal{H}} & = & \mathbf{A} \times \mathbf{S} \\
\dot{\leftrightarrow}_{\mathcal{H}} & = & \dot{\leftrightarrow} \\
(\Psi_1, S_1) \otimes_{\mathcal{H}} (\Psi_2, S_2) & = & (\Psi_1 \otimes \Psi_2,\ S_1 \cup S_2) \\
\mathbf{1}_{\mathcal{H}} & = & (\mathbf{1}, \emptyset) \\
(\Psi, S) \vdash_{\mathcal{H}} \varphi & \textit{if} & \Psi \vdash \varphi \quad \textit{for } \varphi \in \mathbf{C} \\
(\Psi, S) \vdash_{\mathcal{H}} M \Leftarrow P & \textit{if} & \exists \widetilde{L}, K, \widetilde{x}, N, Q.\ \mathrm{n}(M) \supseteq \mathrm{n}(P) \wedge (K(\lambda \widetilde{x})N \Leftarrow Q) \in S \\
 & & \qquad \wedge\ M = K\langle N[\tilde{x} := \tilde{L}]\rangle \wedge P = Q[\tilde{x} := \tilde{L}]
\end{array}$$

For substitution, assuming $\tilde{x}\#\tilde{y}, \tilde{L}$ *we define*
$(M(\lambda\tilde{x})N \Leftarrow P)[\tilde{y} := \tilde{L}]$ *to be* $M[\tilde{y} := \tilde{L}](\lambda\tilde{x})N[\tilde{y} := \tilde{L}] \Leftarrow P[\tilde{y} := \tilde{L}]$
and $(\Psi, S)[\tilde{x} := \tilde{L}]$ *to be* $(\Psi[\tilde{x} := \tilde{L}], \{X[\tilde{x} := \tilde{L}] \mid X \in S\})$.

For a simple example, let us construct a canonical higher-order psi-calculus corresponding to the higher-order pi-calculus. A psi-calculus corresponding to the pi-calculus has been presented in (BJPV09). Here the terms are just names, so lifting would yield a calculus of limited use: in any clause $a \Leftarrow P$ we require $\mathrm{n}(a) \supseteq \mathrm{n}(P)$, and therefore only agents with singleton sorts can be invoked. An extension to admit invocation of arbitrary agents is to let the terms include tuples of names. Because of the requirement of closure under substitution of terms for names these tuples must then be nested. This yields the psi-calculus **Tup**:

Definition 14 (The psi-calculus **Tup**)**.**

$$\begin{array}{rcl}
\mathbf{T} & \stackrel{\text{def}}{=} & \mathcal{N} \cup \{\tilde{M}\ :\ \forall i. M_i \in \mathbf{T}\} \\
\mathbf{C} & \stackrel{\text{def}}{=} & \{M = N\ :\ M, N \in \mathbf{T}\} \\
\mathbf{A} & \stackrel{\text{def}}{=} & \{\mathbf{1}\} \\
M \leftrightarrow N & \stackrel{\text{def}}{=} & M = N \\
\vdash & \stackrel{\text{def}}{=} & \{(\mathbf{1}, M, M)\ :\ M \in \mathbf{T}\}
\end{array}$$

We define $M\langle N\rangle$ as the pair M, N, and gain a canonical higher-order pi-calculus as $\mathcal{H}(\mathbf{Tup})$. As a simple example, let $S = \{M(\lambda\tilde{x})\tilde{x} \Leftarrow P\}$ with $(\mathbf{1}, S) \rhd P[\tilde{x} := \tilde{L}] \xrightarrow{\alpha} P'$. We can then use M to invoke P with parameters $\tilde{L}$ as follows:

$$(\mathbf{1}, \emptyset) \rhd \mathbf{run}\ M\langle\tilde{L}\rangle \mid (\!|\mathbf{1}, S|\!) \xrightarrow{\alpha} P' \mid (\!|\mathbf{1}, S|\!)$$

Theorem 15. *For all* $\mathcal{C}$ *and* $\bullet\langle\bullet\rangle$*,* $\mathcal{H}(\mathcal{C})$ *is a higher-order psi-calculus.*

The theorem amounts to showing that $\mathcal{H}(\mathcal{C})$ satisfies the requirement on the substitution function informally explained in Section 2 and formally set out in (BJPV11), and the requisites on the entailment relation in Definition 3. The proof has been verified in Isabelle, where the challenge was more related to getting the nominal data type constructions correct than expressing the proof strategy.

4. Algebraic theory

We here establish the expected algebraic properties of bisimilarity and proceed to investigate the representations of Sum and Replication. We then define an alternative definition of bisimulation for higher-order communication and establish that it enjoys the same properties. The informal proof ideas for the most challenging part, that higher-order bisimilarity is preserved by parallel composition, are explained in some detail. All proofs have been formally checked in the interactive theorem prover Isabelle and we briefly comment on our experiences.

4.1. *Bisimulation*

We begin by recollecting the definition from (BJPV09), to which we refer for examples and intuitions.

Definition 16 (Bisimulation)**.** *A* strong bisimulation $\mathcal{R}$ *is a ternary relation between assertions and pairs of agents such that* $(\Psi, P, Q) \in \mathcal{R}$ *implies all of*

1 Static equivalence: $\Psi \otimes \mathcal{F}(P) \simeq \Psi \otimes \mathcal{F}(Q)$
2 Symmetry: $(\Psi, Q, P) \in \mathcal{R}$
3 Extension of arbitrary assertion: $\forall \Psi'.\ (\Psi \otimes \Psi', P, Q) \in \mathcal{R}$
4 Simulation: for all α, P' *such that* $\mathrm{bn}(\alpha) \# \Psi, Q$ *there exists a* Q' *such that*

$$\textit{if } \Psi \rhd P \xrightarrow{\alpha} P' \textit{ then } \Psi \rhd Q \xrightarrow{\alpha} Q' \wedge (\Psi, P', Q') \in \mathcal{R}$$

If $\mathcal{R}$ *is a ternary relation between assertions and pairs of agents then we sometimes write* $\Psi \rhd P\ \mathcal{R}\ Q$ *for* $(\Psi, P, Q) \in \mathcal{R}$*. We define* $\Psi \rhd P \mathrel{\dot{\sim}} Q$ *to mean that there exists a strong bisimulation* $\mathcal{R}$ *such that* $\Psi \rhd P\ \mathcal{R}\ Q$*, and write* $P \mathrel{\dot{\sim}} Q$ *for* $\mathbf{1} \rhd P \mathrel{\dot{\sim}} Q$*.*

For higher-order psi-calculi exactly the same definition applies, where frame equivalence means that two frames entail the same conditions *and clauses*.

In the following we restrict attention to well formed agents. The compositionality properties of strong bisimilarity for a higher-order calculus are the same as has previously been established for psi-calculi:

Theorem 17. *For all* Ψ*:*

1 $\Psi \rhd P \mathrel{\dot{\sim}} Q \Longrightarrow \Psi \rhd P \mid R \mathrel{\dot{\sim}} Q \mid R.$
2 $\Psi \rhd P \mathrel{\dot{\sim}} Q \Longrightarrow \Psi \rhd (\nu a)P \mathrel{\dot{\sim}} (\nu a)Q \qquad \textit{if } a \# \Psi.$
3 $\Psi \rhd P \mathrel{\dot{\sim}} Q \Longrightarrow \Psi \rhd\ !P \mathrel{\dot{\sim}} !Q$
4 $\forall i.\Psi \rhd P_i \mathrel{\dot{\sim}} Q_i \Longrightarrow \Psi \rhd \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \mathrel{\dot{\sim}} \mathbf{case}\ \widetilde{\varphi} : \widetilde{Q}$
5 $\Psi \rhd P \mathrel{\dot{\sim}} Q \Longrightarrow \Psi \rhd \overline{M}N.P \mathrel{\dot{\sim}} \overline{M}N.Q.$
6 $(\forall \widetilde{L}.\ \Psi \rhd P[\widetilde{a} := \widetilde{L}] \mathrel{\dot{\sim}} Q[\widetilde{a} := \widetilde{L}]) \implies$
$\Psi \rhd \underline{M}(\lambda \widetilde{a})N.P \mathrel{\dot{\sim}} \underline{M}(\lambda \widetilde{a})N.Q \qquad \textit{if } \widetilde{a} \# \Psi.$

We say that a relation on agents is a *congruence* if it is preserved by all operators, i.e., as in Theorem 17 and additionally by input. Strong bisimilarity is not a congruence since it is not preserved by input. As in similar situations, we get a congruence by closing under all possible substitutions.

Definition 18. $\Psi \rhd P \sim Q$ *means that for all sequences* σ *of substitutions it holds that* $\Psi \rhd P\sigma \mathrel{\dot\sim} Q\sigma$*, and we write* $P \sim Q$ *for* $\mathbf{1} \rhd P \sim Q$*.*

Theorem 19. *For every* Ψ*, the binary relation* $\{(P,Q) : \Psi \rhd P \sim Q\}$ *is a congruence.*

The usual structural laws hold for strong congruence:

Theorem 20. $\sim$ *satisfies the following structural laws:*

$$
\begin{array}{rcll}
P & \sim & P \mid \mathbf{0} & \\
P \mid (Q \mid R) & \sim & (P \mid Q) \mid R & \\
P \mid Q & \sim & Q \mid P & \\
(\nu a)\mathbf{0} & \sim & \mathbf{0} & \\
P \mid (\nu a)Q & \sim & (\nu a)(P \mid Q) & \textit{if } a\#P \\
\overline{M}N.(\nu a)P & \sim & (\nu a)\overline{M}N.P & \textit{if } a\#M,N \\
\underline{M}(\lambda\widetilde{x})N.(\nu a)P & \sim & (\nu a)\underline{M}(\lambda\widetilde{x})(N).P & \textit{if } a\#\widetilde{x},M,N \\
\mathbf{case}\ \widetilde{\varphi} : \widetilde{(\nu a)P} & \sim & (\nu a)\mathbf{case}\ \widetilde{\varphi} : \widetilde{P} & \textit{if } a\#\widetilde{\varphi} \\
(\nu a)(\nu b)P & \sim & (\nu b)(\nu a)P & \\
!P & \sim & P \mid !P &
\end{array}
$$

These results all concern strong bisimulation. The corresponding results for weak bisimulation also hold; we shall not recapitulate them here:

Theorem 21. *All results on the algebraic properties of weak bisimulation as defined and presented in (BJPV10) also hold in higher-order psi-calculi.*

The proof ideas for all results in this subsection are similar to the our previously published results for (non-higher-order) psi-calculi, and the formal proofs in Isabelle required very little modification.

4.2. *Encoding operators*

We here formalise the ideas from Section 3.1 and establish when the operators Replication, Sum and n-ary **case** can be encoded. Recapitulating the idea of the encoding of replication $!P$ as

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

we immediately see that it needs an assertion $\{M_a \Leftarrow P \mid \mathbf{run}\ M_a\}$ which, intuitively, entails the clause $M_a \Leftarrow P \mid \mathbf{run}\ M_a$ and nothing else. We call such an assertion a *characteristic assertion* for M_a and $P \mid \mathbf{run}\ M_a$, and in the corresponding encoding of **case** we need characteristic assertions for sequences of agents $\widetilde{P}$ with a common handle. The full definition is:

Definition 22. *In a higher-order calculus, for a finite sequence of agents* $\widetilde{P} = P_1,\ldots,P_n$ *and term* M *with* $\mathrm{n}(\widetilde{P}) \subseteq \mathrm{n}(M)$ *the assertion* $\Psi^{M\Leftarrow\widetilde{P}}$ *is* characteristic *for* M *and* $\widetilde{P}$ *if the following holds for all agents* Q*, assertions* Ψ *and clauses and conditions* ξ*:*

1 $\Psi \vdash M \Leftarrow Q$ *implies* $\mathrm{n}(M) \subseteq \mathrm{n}(\Psi)$
2 $\Psi \otimes \Psi^{M\Leftarrow\widetilde{P}} \vdash \xi$ *iff* $(\xi = M \Leftarrow P_i \ \vee\ \Psi \vdash \xi)$
3 $\mathrm{n}(\Psi^{M\Leftarrow P}) = \mathrm{n}(M)$

The first is a general requirement on the calculus and makes sure that an environment cannot bestow additional invocation possibilities to the handles used in the encodings. For example, suppose that $\mathbf{1} \vdash M_a \Leftarrow Q$, violating the requirement, then clearly $(\nu a)\mathbf{run}\ M_a \cdots$ can enact Q, in other words our encoding of $!P$ could also enact Q. Requirement 1 excludes this possibility since $a \in \mathrm{n}(M_a)$ and $a \notin \mathrm{n}(\mathbf{1}) = \emptyset$. The second requirement means that the characteristic assertion only has the effect to entail its clauses, no matter how it is combined with other assertions. The third requirement ensures that the characteristic assertion does not invent names that do not occur in its handle.

Characteristic assertions fortunately exist in most canonical higher-order calculi. We need to restrict attention to calculi with a *unit term* $() \in \mathbf{T}$ such that $\mathrm{n}(()) = \emptyset$, and the pairing function satisfies $M\langle N\rangle = M'\langle N'\rangle \implies M = M'$, and for all $T \in \mathbf{T} \cup \mathbf{A} \cup \mathbf{C}$, $T[\varepsilon := \varepsilon] = T$. The reason is technical: in a canonical calculus we use parametrised clauses, where the handles must be treated as distinct, and in situations where no parameter is actually needed we use $()$ as a dummy and communications give rise to empty substitutions. In assertions we then write $M \Leftarrow P$ for the parametrised clause $M(\lambda\varepsilon)() \Leftarrow P$, and in processes $\mathbf{run}\ M$ for the invocation $\mathbf{run}\ M()$.

Theorem 23. *In a canonical higher order-calculus with unit term, pairing and empty substitution as above, if* $\mathrm{n}(\widetilde{P}) \subseteq \mathrm{n}(M)$*, and* $\widetilde{P} \neq \varepsilon$ *then the assertion*

$$(\mathbf{1}, \{M \Leftarrow P_i \ :\ P_i \in \widetilde{P}\})$$

is characteristic for M *and* $\widetilde{P}$*.*

The following formal theorems of the encodings hold for arbitrary higher-order calculi, and are particularly relevant for canonical calculi where characteristic assertions can be expressed easily.

Theorem 24. *In a higher-order calculus with the* $+$ *operator (i.e., there exists a condition* $\top$*, cf. the discussion following Definition 6), for all assertion guarded* P, Q *and names* $a\#P, Q$ *and terms* M *with* $\mathrm{n}(P, Q, a) \subseteq \mathrm{n}(M)$ *and assertions* $\Psi^{M \Leftarrow P,Q}$ *characteristic for* M *and* P, Q *it holds that*

$$P + Q \mathrel{\dot{\sim}} (\nu a)(\mathbf{run}\ M \mid (\!|\Psi^{M \Leftarrow P,Q}|\!))$$

Theorem 25. *In a higher-order calculus, for all assertion guarded* $\tilde{P} = P_1, \ldots, P_n$*, conditions* $\tilde{\varphi} = \varphi_1, \ldots, \varphi_n$*, names* $a\#\widetilde{P}, \widetilde{\varphi}$ *and terms* M *with* $\mathrm{n}(\widetilde{P}, \widetilde{\varphi}, a) \subseteq \mathrm{n}(M)$ *and assertions* $\Psi^{M \Leftarrow \mathbf{if}\ \varphi_1\ \mathbf{then}\ P_1, \ldots, \mathbf{if}\ \varphi_n\ \mathbf{then}\ P_n}$ *characteristic for* M *and the agent*
$\mathbf{if}\ \varphi_1\ \mathbf{then}\ P_1, \ldots, \mathbf{if}\ \varphi_n\ \mathbf{then}\ P_n$ *it holds that*

$$\begin{aligned}&\mathbf{case}\ \varphi_1 : P_1 \mathbin{[]} \cdots \mathbin{[]} \varphi_n : P_n \\ &\quad \mathrel{\dot{\sim}} (\nu a)(\mathbf{run}\ M \mid (\!|\Psi^{M \Leftarrow (\mathbf{if}\ \varphi_1\ \mathbf{then}\ P_1), \ldots, (\mathbf{if}\ \varphi_n\ \mathbf{then}\ P_n)}|\!))\end{aligned}$$

Theorem 26. *In a higher-order calculus, for all assertion guarded* P*, names* $a\#P$ *and terms* M *with* $\mathrm{n}(P, a) \subseteq \mathrm{n}(M)$ *and assertions* $\Psi^{M \Leftarrow P \mid \mathbf{run}\ M}$ *characteristic for* M *and* $P \mid \mathbf{run}\ M$ *it holds that*

$$!P \mathrel{\dot{\sim}} (\nu a)(\mathbf{run}\ M \mid (\!|\Psi^{M \Leftarrow P \mid \mathbf{run}\ M}|\!))$$

As an example of the encoding of Replication, consider a transition from

$$(\nu a)(\mathbf{run}\ M \mid (\!|\Psi^{M \Leftarrow P \mid \mathbf{run}\ M}|\!))$$

It can only be by invocation where $P \mid \mathbf{run}\ M$ has a transition leading to $P' \mid \mathbf{run}\ M$ and results in

$$(\nu a)(P' \mid \mathbf{run}\ M \mid (\!|\Psi^{M \Leftarrow P \mid \mathbf{run}\ M}|\!))$$

Using Theorem 20 and $a\#P'$ we rewrite this as

$$P' \mid (\nu a)(\mathbf{run}\ M \mid (\!|\Psi^{M \Leftarrow P \mid \mathbf{run}\ M}|\!))$$

In other words, the transition precisely corresponds to the transition of $!P$ derived from $P \mid !P$.

Clearly, for these theorems to be applicable there must exist terms M with large enough support to represent handles. This is the case for e.g. $\mathcal{H}(\mathbf{Tup})$ from Section 3.3, which has terms with arbitrarily large finite support.

4.3. *Higher-order bisimulation*

In higher-order process calculi the standard notion of bisimilarity is often found unsatisfactory since it requires actions to match exactly: an action $\overline{a}P$ must be simulated by an identical action. Therefore, if $P \neq P'$ we will have $\overline{a}P \,.\, \mathbf{0} \not\sim \overline{a}P' \,.\, \mathbf{0}$, even if $P \sim P'$, which spoils the claim for $\sim$ to be a congruence in the ordinary sense of the word.

In psi-calculi the data terms can be anything, even processes, but here the distinction between $\overline{a}P \,.\, \mathbf{0}$ and $\overline{a}P' \,.\, \mathbf{0}$ is necessary since the semantics allows a recipient to use the received process in a variety of ways. For example, there are psi-calculi where it is possible to receive a process and test whether it is syntactically equal to another process, as in $a(x) \,.\, \mathbf{if}\ x = Q\ \mathbf{then}\ \ldots$, or to subject it to pattern matching in order to find its outermost operator; this corresponds to inspecting the process code.

In a higher-order process calculus we can instead transmit the possibility to invoke a process, as in $(\nu b)\overline{a}M_b \,.\, (\!|\{M_b \Leftarrow P\}|\!)$. A recipient of M_b has no other use for this handle than to invoke P. Therefore, if $P \,\dot{\sim}\, P'$ it is reasonable to expect the two processes

$$\begin{array}{lcl} Q & = & (\nu b)\overline{a}M_b \,.\, (\!|\{M_b \Leftarrow P\}|\!) \\ Q' & = & (\nu b)\overline{a}M_b \,.\, (\!|\{M_b \Leftarrow P'\}|\!) \end{array}$$

to be bisimilar, since it should not matter which of P or P' is invoked. But with the current definition of bisimilarity, $Q \not\dot{\sim} Q'$. Consider a transition from Q which opens the scope of b. The resulting agent is simply $(\!|\{M_b \Leftarrow P\}|\!)$. The corresponding transition from Q' leads to $(\!|\{M_b \Leftarrow P'\}|\!)$. These are not bisimilar since they are not statically equivalent: $\{M_b \Leftarrow P\} \not\simeq \{M_b \Leftarrow P'\}$, since they do not entail exactly the same clauses.

This suggests that a slightly relaxed version of bisimilarity is more appropriate, where we weaken static equivalence to require bisimilar (rather than identical) entailed clauses.

Definition 27 (HO-Bisimulation)**.** *A* strong HO-bisimulation $\mathcal{R}$ *is a ternary relation between assertions and pairs of agents such that* $(\Psi, P, Q) \in \mathcal{R}$ *implies all of*

1 Static equivalence:

(a) $\forall \varphi \in \mathbf{C}.\quad \Psi \otimes \mathcal{F}(P) \vdash \varphi \;\Rightarrow\; \Psi \otimes \mathcal{F}(Q) \vdash \varphi$
(b) $\forall (M \Leftarrow P') \in \mathbf{Cl}.\quad \Psi \otimes \mathcal{F}(P) \vdash M \Leftarrow P' \;\Rightarrow$
$\exists Q'.\, \Psi \otimes \mathcal{F}(Q) \vdash M \Leftarrow Q' \wedge (\mathbf{1}, P', Q') \in \mathcal{R}$
where $\mathcal{F}(P) = (\nu \widetilde{b}_P)\Psi_P$ *and* $\mathcal{F}(Q) = (\nu \widetilde{b}_Q)\Psi_Q$ *and* $\widetilde{b}_P \widetilde{b}_Q \# \Psi, M$.

2 Symmetry: $(\Psi, Q, P) \in \mathcal{R}$

3 Extension of arbitrary assertion: $\forall \Psi'.\ (\Psi \otimes \Psi', P, Q) \in \mathcal{R}$

4 Simulation: for all α, P' *such that* $\mathrm{bn}(\alpha) \# \Psi, Q$ *there exists a* Q' *such that*

$$\textit{if } \Psi \rhd P \xrightarrow{\alpha} P' \textit{ then } \Psi \rhd Q \xrightarrow{\alpha} Q' \wedge (\Psi, P', Q') \in \mathcal{R}$$

We define $\Psi \rhd P \mathrel{\dot{\sim}}^{\text{HO}} Q$ *to mean that there exists a strong HO-bisimulation* $\mathcal{R}$ *such that* $\Psi \rhd P\, \mathcal{R}\, Q$*, and write* $P \mathrel{\dot{\sim}}^{\text{HO}} Q$ *for* $\mathbf{1} \rhd P \mathrel{\dot{\sim}}^{\text{HO}} Q$.

The only difference between bisimulation and HO-bisimulation is in Clause 1, which here is split into different requirements for conditions and clauses.

Theorem 28. *In a higher-order psi-calculus, for all assertion guarded* P, Q *and terms* M *with* $\mathrm{n}(P, Q) \subseteq \mathrm{n}(M)$ *with characteristic assertions* $\Psi^{M \Leftarrow P}$ *and* $\Psi^{M \Leftarrow Q}$*, it holds that*

$$P \mathrel{\dot{\sim}}^{\text{HO}} Q \Rightarrow (\!|\Psi^{M \Leftarrow P}|\!) \mathrel{\dot{\sim}}^{\text{HO}} (\!|\Psi^{M \Leftarrow Q}|\!)$$

The proof boils down to showing that

$$\{(\Psi, (\!|\Psi^{M \Leftarrow P}|\!), (\!|\Psi^{M \Leftarrow Q}|\!)) : \Psi \in \mathbf{A}, P \mathrel{\dot{\sim}}^{\text{HO}}_{\Psi} Q\} \cup \mathrel{\dot{\sim}}^{\text{HO}}$$

is a HO-bisimulation. The only nontrivial part is static equivalence. In order to prove this we use Definition 22(2), that $\Psi \otimes \Psi^{M \Leftarrow P} \vdash \xi$ iff $(\xi = M \Leftarrow P \;\vee\; \Psi \vdash \xi)$. The proof holds for all calculi with characteristic assertions, and in particular it holds for canonical calculi by Theorem 23.

In the rest of this section we study the algebraic properties of HO-bisimulation in arbitrary calculi (not only canonical ones). The original bisimulation is still a valid proof technique:

Theorem 29. $\Psi \rhd P \mathrel{\dot{\sim}} Q \implies \Psi \rhd P \mathrel{\dot{\sim}}^{\text{HO}} Q$

The proof is that $\mathrel{\dot{\sim}}$ is a HO-bisimulation: take $Q' = P'$ in Clause 1(b). Thus we immediately get a set of useful algebraic laws:

Corollary 30. $\mathrel{\dot{\sim}}^{\text{HO}}$ *satisfies all structural laws of Theorem 20.*

HO-bisimulation is compositional in the same way as ordinary bisimulation:

Theorem 31. *For all* Ψ*:*

1 $\Psi \rhd P \mathrel{\dot{\sim}}^{\text{HO}} Q \Longrightarrow \Psi \rhd P \mid R \mathrel{\dot{\sim}}^{\text{HO}} Q \mid R.$
2 $\Psi \rhd P \mathrel{\dot{\sim}}^{\text{HO}} Q \Longrightarrow \Psi \rhd (\nu a)P \mathrel{\dot{\sim}}^{\text{HO}} (\nu a)Q \qquad \textit{if } a \# \Psi.$
3 $\Psi \rhd P \mathrel{\dot{\sim}}^{\text{HO}} Q \Longrightarrow \Psi \rhd\, !P \mathrel{\dot{\sim}}^{\text{HO}} !Q \qquad \textit{if } \mathrm{guarded}(P, Q).$
4 $\forall i. \Psi \rhd P_i \mathrel{\dot{\sim}}^{\text{HO}} Q_i \Longrightarrow \Psi \rhd \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \mathrel{\dot{\sim}}^{\text{HO}} \mathbf{case}\ \widetilde{\varphi} : \widetilde{Q} \qquad \textit{if } \mathrm{guarded}(\widetilde{P}, \widetilde{Q}).$
5 $\Psi \rhd P \mathrel{\dot{\sim}}^{\text{HO}} Q \Longrightarrow \Psi \rhd \overline{M}N.P \mathrel{\dot{\sim}}^{\text{HO}} \overline{M}N.Q.$
6 $(\forall \widetilde{L}.\ \Psi \rhd P[\widetilde{a} := \widetilde{L}] \mathrel{\dot{\sim}}^{\text{HO}} Q[\widetilde{a} := \widetilde{L}]) \implies$
$\Psi \rhd \underline{M}(\lambda \widetilde{a})N.P \mathrel{\dot{\sim}}^{\text{HO}} \underline{M}(\lambda \widetilde{a})N.Q \qquad \textit{if } \widetilde{a} \# \Psi.$

Combining Theorem 31 and Theorem 28 we get the desired result for our motivating example: in a canonical higher-order psi calculus it holds that

$$P \mathrel{\dot{\sim}^{\text{HO}}} P' \;\Rightarrow\; (\nu b)\overline{a}M_b \,.\, (\!|\Psi^{M_b \Leftarrow P}|\!) \mathrel{\dot{\sim}^{\text{HO}}} (\nu b)\overline{a}M_b \,.\, (\!|\Psi^{M_b \Leftarrow P'}|\!)$$

We can characterise higher-order bisimulation congruence in the usual way:

Definition 32. *$\Psi \rhd P \sim^{\text{HO}} Q$ iff for all sequences σ of substitutions it holds that $\Psi \rhd P\sigma \mathrel{\dot{\sim}^{\text{HO}}} Q\sigma$. We write $P \sim^{\text{HO}} Q$ for $\mathbf{1} \rhd P \sim^{\text{HO}} Q$.*

Theorem 33. *For every Ψ, the binary relation $\{(P, Q) \;:\; \Psi \rhd P \sim^{\text{HO}} Q\}$ is a congruence.*

Theorem 34. *$\sim^{\text{HO}}$ satisfies the following structural laws:*

$$\begin{array}{rcll}
P & \sim^{\text{HO}} & P \mid \mathbf{0} & \\
P \mid (Q \mid R) & \sim^{\text{HO}} & (P \mid Q) \mid R & \\
P \mid Q & \sim^{\text{HO}} & Q \mid P & \\
(\nu a)\mathbf{0} & \sim^{\text{HO}} & \mathbf{0} & \\
P \mid (\nu a)Q & \sim^{\text{HO}} & (\nu a)(P \mid Q) & \text{if } a\#P \\
\overline{M}N.(\nu a)P & \sim^{\text{HO}} & (\nu a)\overline{M}N.P & \text{if } a\#M, N \\
\underline{M}(\lambda\widetilde{x})N.(\nu a)P & \sim^{\text{HO}} & (\nu a)\underline{M}(\lambda\widetilde{x})(N).P & \text{if } a\#\widetilde{x}, M, N \\
\mathbf{case}\ \widetilde{\varphi} : \widetilde{(\nu a)P} & \sim^{\text{HO}} & (\nu a)\mathbf{case}\ \widetilde{\varphi} : \widetilde{P} & \text{if } a\#\widetilde{\varphi} \\
(\nu a)(\nu b)P & \sim^{\text{HO}} & (\nu b)(\nu a)P & \\
!P & \sim^{\text{HO}} & P \mid !P &
\end{array}$$

4.4. *Informal proofs*

Most of the proofs follow the corresponding results in the original psi-calculi closely. We here present the most challenging part where new proof ideas are needed for Theorem 31.1, that higher-order bisimilarity is preserved by parallel. One main complication is that the INVOCATION rule can be used multiple times during the derivation of a transition. Another complication is that the relation $\{(P \mid R, Q \mid R) : P \mathrel{\dot{\sim}^{\text{HO}}} Q\}$ is no longer a bisimulation: If P and Q are different their assertions can enable different invocations in R, so a transition from R leads to agents outside the relation. In the proof, we therefore work with bisimulation up to transitivity (San98). For technical reasons, in the proofs we additionally parametrise the transitive closure on a set of names that must not appear in processes.

The proof of compositionality for ordinary bisimulation is described in some detail in (BJPV11; Joh10), to which we refer for motivating examples and a discussion of the proof structure. We here focus on the main differences in the higher-order case, including the use of up-to techniques.

Definition 35 (Up-to techniques)**.** *We inductively define the following up-to techniques: up to union with HO-bisimilarity (U), up to restriction (R) and up to transitivity (T).*

$$\begin{aligned}
U(\mathcal{R}) &:= \mathcal{R} \cup \dot{\sim}^{\text{HO}} \\
R(\mathcal{R}) &:= \{(\Psi, (\nu\widetilde{a})P, (\nu\widetilde{a})Q) : \widetilde{a}\#\Psi \wedge (\Psi, P, Q) \in \mathcal{R}\} \\
T(\mathcal{R}) &:= \mathcal{R} \cup \{(\Psi, P, R) : (\Psi, P, Q) \in T(\mathcal{R}) \wedge (\Psi, Q, R) \in T(\mathcal{R})\}
\end{aligned}$$

A HO-bisimulation up to S is defined as a HO-bisimulation, except that the derivatives after a simulation step or an invocation should be related by $S(\mathcal{R})$ instead of $\mathcal{R}$.

Definition 36 (HO-bisimulation up-to)**.** *If S is a function from ternary relations to ternary relations, then $\mathcal{R}$ is a bisimulation up to S if $\mathcal{R}$ satisfies Definition 27 with $S(\mathcal{R})$ substituted for $\mathcal{R}$ in clauses 1(b) and 4.*

The up-to techniques of Definition 35 are sound.

Theorem 37. *If $\mathcal{R}$ is a HO-bisimulation up to $T \circ U \circ R$, then $\mathcal{R} \subseteq \dot{\sim}^{\text{HO}}$.*

Proof. The proof is standard: If $\mathcal{R}$ is a HO-bisimulation up to $T \circ U \circ R$, then $T(U(R(\mathcal{R})))$ is a HO-bisimulation and $\mathcal{R} \subseteq T(U(R(\mathcal{R})))$. □

In the inductive proofs of technical lemmas below, we often need to strengthen the notion of transitivity by parametrising on a finite set of names that are fresh for the processes under consideration and therefore must be avoided.

Definition 38 (Name-avoiding transitivity)**.** *If $\mathcal{R}$ is a ternary relation, then $T_{\text{a}}(\mathcal{R})$ is inductively defined as follows.*

$$T_{\text{a}}(\mathcal{R}) := \{(B, \Psi, P, Q) : B\#P, Q \wedge \Psi \rhd P\ \mathcal{R}\ Q\} \cup \{(B, \Psi, P, R) : (B, \Psi, P, Q) \in T_{\text{a}}(\mathcal{R}) \wedge (B, \Psi, Q, R) \in T_{\text{a}}(\mathcal{R}))\}$$

If $\mathcal{R}' = T_{\text{a}}(\mathcal{R})$, we write $\Psi \rhd_B P\ \mathcal{R}'\ Q$ for $(B, \Psi, P, Q) \in \mathcal{R}'$.

We write $F \rhd P\ \mathcal{R}\ Q$ if $F = (\nu\widetilde{x})\Psi$ such that $\Psi \rhd P\ \mathcal{R}\ Q$ and $\widetilde{x}\#P, Q$.

Note that $\Psi \rhd P\ (T(\mathcal{R}))\ Q$ iff $\Psi \rhd_\emptyset P\ (T_{\text{a}}(\mathcal{R}))\ Q$.

In the remainder of the proof, we will work with the candidate relation $\mathcal{S}$ defined below.

$$\mathcal{S} := \{(\Psi, P|R, Q|R) \mid \Psi \otimes \mathcal{F}(R) \rhd P \dot{\sim}^{\text{HO}} Q\}$$

We first seek to show that $\mathcal{S}$ is a HO-bisimulation up to $T \circ U \circ R$; compositionality of higher-order bisimulation then follows using the soundness of the up-to techniques. We write $\overline{\mathcal{S}}$ for $T_{\text{a}}(U(R(\mathcal{S})))$. The proof begins by showing some closure properties of $\overline{\mathcal{S}}$ that are used in the induction cases of the main lemmas. We then recall some technical lemmas from (BJPV11) about the choice of subjects in transitions. The main lemmas (Lemma 43 and 44) concern the simulation case of the definition of HO-bisimilarity, in particular transitions of R and communications between P and R, respectively.

The following closure properties of $\overline{\mathcal{S}}$ hold. Intuitively, $\overline{\mathcal{S}}$ is a congruence with respect to parallel composition and restriction, is preserved by bisimilarity, and is monotonic in B and Ψ modulo $\simeq$.

Lemma 39. *If $\Psi \rhd_B P\ \overline{\mathcal{S}}\ Q$ then*

1 if $\Psi \simeq \Psi' \otimes \Psi_R$, $\mathcal{F}(R) = (\nu\widetilde{b}_R)\Psi_R$, $\widetilde{b}_R \subseteq B$, $B\#R$ and $\widetilde{b}_R\#\Psi', R$ then $\Psi' \rhd_{B\setminus\widetilde{b}_R} (P \mid R)\ \overline{\mathcal{S}}\ (Q \mid R)$; and

2 if $a\#\Psi$ then $\Psi \rhd_B (\nu a)P\ \overline{\mathcal{S}}\ (\nu a)Q$; and

3 if $\Psi \rhd P' \dot{\sim} P$ and $\Psi \rhd Q \dot{\sim} Q'$ and $B\#P', Q'$ then $\Psi \rhd_B P'\ \overline{\mathcal{S}}\ Q'$; and

4 $\Psi \otimes \Psi' \rhd_B P\ \overline{\mathcal{S}}\ Q$; and

5 $\Psi \rhd_{B\setminus B'} P\ \overline{\mathcal{S}}\ Q$; and

6 if $\Psi \simeq \Psi_2$ then $\Psi_2 \rhd_B P\ \overline{\mathcal{S}}\ Q$.

Proof. By induction on the definition of T_{a}. □

We recall three lemmas used in the compositionality proof for first-order bisimilarity (BJPV11). In their statements we assume that $\mathcal{F}(P) = (\nu\widetilde{b}_P)\Psi_P$. These lemmas have been proven to hold also for higher-order psi-calculi. The first lemma states that when performing a non-tau transition, the frame of the process grows such that the bound names in the frame and the action can be chosen fresh for an arbitrary finite set of names B.

Lemma 40 (Frame grows when doing transitions)**.**

1 If $\Psi \rhd P \xrightarrow{\underline{M}\,N} P'$ *and* $\widetilde{b}_P\#P, N, B$, *then* $\exists\Psi', \widetilde{b}_{P'}, \Psi_{P'}$ *s.t.* $\mathcal{F}(P') = (\nu\widetilde{b}_{P'})\Psi_{P'}$ *and* $\Psi_P\otimes\Psi' \simeq \Psi_{P'}$ *and* $\widetilde{b}_{P'}\#B, P'$.

2 If $\Psi \rhd P \xrightarrow{\overline{M}\,(\nu\widetilde{a})N} P'$, $\widetilde{b}_P\#P, \widetilde{a}, B$, *and* $\widetilde{a}\#P, B$, *then* $\exists p, \Psi', \widetilde{b}_{P'}, \Psi_{P'}$ *s.t.* $\mathcal{F}(P') = (\nu\widetilde{b}_{P'})\Psi_{P'}$ *and* $\Psi_{P'} \simeq (p\cdot\Psi_P)\otimes\Psi'$ *and* $\widetilde{b}_{P'}\#B, P', N, \widetilde{a}$ *and* $(p\cdot\widetilde{a})\#B, P', N, \widetilde{b}_{P'}$ *and* $p \subseteq \widetilde{a}\times(p\cdot\widetilde{a})$.

The second lemma states that given a non-tau transition of P and a finite set of names B that are fresh for P, we can find a term K that is channel-equivalent to the subject of the transition such that B is fresh for K.

Lemma 41 (Find fresh subject)**.**

$$\begin{array}{rl} & B\#P \\ \wedge & \Psi \rhd P \xrightarrow{\alpha} P' \text{ where } \alpha \neq \tau \\ \wedge & \widetilde{b}_P\#\Psi, P, \mathrm{subj}(\alpha), B \\ \Longrightarrow & \exists K.\, B\#K \wedge \Psi\otimes\Psi_P \vdash K \leftrightarrows \mathrm{subj}(\alpha) \end{array}$$

The third lemma states that if a process P performs a non-tau transition, and K is channel-equivalent to the subject of the transition, then P can perform the same transition with K as subject.

Lemma 42 (Subject rewriting)**.**

$$\begin{array}{rl} & \Psi \rhd P \xrightarrow{\alpha} P' \\ \wedge & \Psi_P\otimes\Psi \vdash K \leftrightarrows M \\ \wedge & \widetilde{b}_P\#\Psi, P, K, M \\ \Longrightarrow & \Psi \rhd P \xrightarrow{\alpha'} P' \end{array}$$

when $\alpha = \overline{M}\,(\nu\widetilde{a})N$ *and* $\alpha' = \overline{K}\,(\nu\widetilde{a})N$, *or* $\alpha = \underline{M}\,N$ *and* $\alpha' = \underline{K}\,N$.

We can now show our main technical lemma, which intuitively states that if P and Q are bisimilar in the environment of R, and R makes a transition in the environment of P, then R can make the same transition in the environment of Q, leading to $\overline{\mathcal{S}}$-related derivatives. The proof makes use of a set B of names that are required to be fresh, which grows in the induction case. A similar lemma applies in first-order psi-calculi (BJPV11), where the derivatives are always syntactically equal (not just related by $\overline{\mathcal{S}}$).

Lemma 43 (frame switching lemma)**.**

$$\begin{array}{rl} & \Psi \otimes \Psi_R \rhd P \mathrel{\dot{\sim}^{\text{HO}}} Q \\ \wedge & \Psi \otimes \Psi_P \rhd R \xrightarrow{\alpha} R_P \\ \wedge & \mathcal{F}(P) = (\nu \widetilde{b}_P)\Psi_P \\ \wedge & \mathcal{F}(Q) = (\nu \widetilde{b}_Q)\Psi_Q \\ \wedge & \mathcal{F}(R) = (\nu \widetilde{b}_R)\Psi_R \\ \wedge & \widetilde{b}_P \# \alpha, \Psi, R \\ \wedge & \widetilde{b}_Q \# \alpha, \Psi, R \\ \wedge & \widetilde{b}_R \# \alpha, \widetilde{b}_P, \widetilde{b}_Q, \Psi, P, Q, R \\ \wedge & \text{bn}(\alpha) \# \Psi, P, Q, R \\ \wedge & B \# \Psi, P, Q, R, \text{obj}(\alpha), R_P, \widetilde{b}_R \\ \Longrightarrow & \exists R_Q. \\ & \Psi \otimes \Psi_Q \rhd R \xrightarrow{\alpha} R_Q \\ \wedge & \Psi \rhd_B (P|R_P) \mathrel{\overline{\mathcal{S}}} (Q|R_Q) \end{array}$$

Proof. By induction on the derivation of the transition of R. The base cases are as in (BJPV11). We here show some interesting induction cases.

Inv Here $R = \mathbf{run}\ M$ and $\mathcal{F}(R) = \mathbf{1}$ and the transition is derived like

$$\textsc{Invocation}\ \frac{\Psi \otimes \Psi_P \vdash M \Leftarrow R_1 \qquad \Psi \otimes \Psi_P \rhd R_1 \xrightarrow{\alpha} R_P}{\Psi \otimes \Psi_P \rhd \mathbf{run}\ M \xrightarrow{\alpha} R_P}$$

By induction, there is R' such that $\Psi \otimes \Psi_Q \rhd R_1 \xrightarrow{\alpha} R'$ and $\Psi \rhd_B (P|R_P) \mathrel{\overline{\mathcal{S}}} (Q|R')$.
Since $\Psi \otimes \mathbf{1} \rhd P \mathrel{\dot{\sim}^{\text{HO}}} Q$ there is R_2 such that $\Psi \otimes \Psi_Q \vdash M \Leftarrow R_2$ and $\mathbf{1} \rhd R_1 \mathrel{\dot{\sim}^{\text{HO}}} R_2$. Then $\Psi \otimes \Psi_Q \rhd R_1 \mathrel{\dot{\sim}^{\text{HO}}} R_2$, so there is R_Q such that $\Psi \otimes \Psi_Q \rhd R_2 \xrightarrow{\alpha} R_Q$ and $\Psi \otimes \Psi_Q \rhd R' \mathrel{\dot{\sim}^{\text{HO}}} R_Q$. By the definition of $\mathcal{S}$, we then get $\Psi \rhd (Q|R') \mathrel{\mathcal{S}} (Q|R_Q)$. Since $B \# R, \text{obj}(\alpha)$ we get $B \# R_Q$, so $\Psi \rhd_B (Q|R') \mathrel{\overline{\mathcal{S}}} (Q|R_Q)$. By transitivity $\Psi \rhd_B (P|R_P) \mathrel{\overline{\mathcal{S}}} (Q|R_Q)$.

Scope In this case we have that $\mathcal{F}((\nu b)R) = (\nu b)\mathcal{F}(R)$ where $\mathcal{F}(R) = (\nu \widetilde{b}_R)\Psi_R$, so $\mathcal{F}((\nu b)R) = (\nu b \widetilde{b}_R)\Psi_R$. We assume that $b \# \widetilde{b}_R$; since $b\widetilde{b}_R \# (\nu b)R$ we then have $\widetilde{b}_R \# R$. The transition is derived like

$$\textsc{Scope}\ \frac{\Psi \otimes \Psi_P \rhd R \xrightarrow{\alpha} R_P}{\Psi \otimes \Psi_P \rhd (\nu b)R \xrightarrow{\alpha} (\nu b)R_P}\ b \# \alpha, \Psi \otimes \Psi_P$$

By induction we get that there exists R_Q such that $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\alpha} R_Q$ and $\Psi \rhd_{B \cup \{b\}} (P|R_P) \mathrel{\overline{\mathcal{S}}} (Q|R_Q)$. Using Scope, $\Psi \otimes \Psi_Q \rhd (\nu b)R \xrightarrow{\alpha} (\nu b)R_Q$. Using Lemma 39 we get $\Psi \rhd_B (P|(\nu b)R_P) \mathrel{\overline{\mathcal{S}}} (Q|(\nu b)R_Q)$.

Par Here $\mathcal{F}(R_1 \mid R_2) = (\nu \widetilde{b}_{R_1} \widetilde{b}_{R_2})\Psi_{R_1} \otimes \Psi_{R_2}$ with $\widetilde{b}_{R_1} \# \widetilde{b}_{R_2}, \Psi_{R_2}$ and $\widetilde{b}_{R_2} \# \widetilde{b}_{R_1}, \Psi_{R_1}$. The

transition is derived like

$$\textsc{Par}\ \frac{\Psi_{R_2}\otimes\Psi\otimes\Psi_P \rhd R_1 \xrightarrow{\alpha} R_P}{\Psi\otimes\Psi_P \rhd R_1 \mid R_2 \xrightarrow{\alpha} R_P \mid R_2}\ \mathrm{bn}(\alpha)\#R_2$$

We know that $\widetilde{b}_P\#\Psi, R_1 \mid R_2$ and that $\widetilde{b}_{R_1}\widetilde{b}_{R_2}\#R_1 \mid R_2, \widetilde{b}_P$. This gives us that also $\widetilde{b}_P\#\Psi_{R_2}\otimes\Psi, R_1$ and that $\widetilde{b}_{R_1}\#\Psi_{R_2}\otimes\Psi, R_1$.

By induction we get R_Q such that $\Psi_{R_2}\otimes\Psi\otimes\Psi_Q \rhd R_1 \xrightarrow{\alpha} R_Q$ and $\Psi_{R_2}\otimes\Psi \rhd_{B\cup\widetilde{b}_{R_2}} (P|R_P)\ \overline{\mathcal{S}}\ (Q|R_Q)$. We then derive

$$\textsc{Par}\ \frac{\Psi_{R_2}\otimes\Psi\otimes\Psi_Q \rhd R_1 \xrightarrow{\alpha} R_Q}{\Psi\otimes\Psi_Q \rhd R_1 \mid R_2 \xrightarrow{\alpha} R_Q \mid R_2}\ \mathrm{bn}(\alpha)\#R_2$$

Using Lemma 39 we get that $\Psi \rhd_B (P|R_P|R_2)\ \overline{\mathcal{S}}\ (Q|R_Q|R_2)$.

Com Here $\mathcal{F}(R_1 \mid R_2) = (\nu\widetilde{b}_{R_1}\widetilde{b}_{R_2})\Psi_{R_1}\otimes\Psi_{R_2}$ with $\widetilde{b}_{R_1}\#\widetilde{b}_{R_2}, \Psi_{R_2}$ and vice versa. The transition is derived like

$$\textsc{Com}\ \frac{\begin{array}{c}\Psi_{R_2}\otimes\Psi\otimes\Psi_P \rhd R_1 \xrightarrow{\overline{M}\,(\nu\widetilde{a})N} R_{P1} \\ \Psi_{R_1}\otimes\Psi\otimes\Psi_P \rhd R_2 \xrightarrow{K\,N} R_{P2} \qquad \Psi\otimes\Psi_P\otimes\Psi_{R_1}\otimes\Psi_{R_2} \vdash M \leftrightarrows K\end{array}}{\Psi\otimes\Psi_P \rhd R_1 \mid R_2 \xrightarrow{\tau} (\nu\widetilde{a})(R_{P1} \mid R_{P2})}$$

We assume that $\widetilde{b}_P\#\widetilde{a}$ (otherwise α-convert $\widetilde{a}$ as necessary). Since $\widetilde{b}_P\#R_1 \mid R_2$ we get $\widetilde{b}_P\#N$. However, we cannot use the induction hypothesis directly since we do not know that $\widetilde{b}_P\#M$ and $\widetilde{b}_P\#K$, respectively.

Let $B_1 = \widetilde{b}_P \cup \widetilde{b}_{R_2}$. We have that $\widetilde{b}_{R_1}\#\Psi_{R_2}, \Psi, \Psi_P, R_1, M, B'$. By Lemma 41 we get that there exists M' such that $B_1\#M'$ and $\Psi\otimes\Psi_P\otimes\Psi_{R_1}\otimes\Psi_{R_2} \vdash M \leftrightarrows M'$. Similarly, by applying Lemma 41 to the transition of R_2 we get K' such that $\widetilde{b}_P\widetilde{b}_{R_1}\#K'$ and $\Psi\otimes\Psi_P\otimes\Psi_{R_1}\otimes\Psi_{R_2} \vdash K \leftrightarrows K'$.

By symmetry and transitivity of $\leftrightarrows$ we then get that $\Psi\otimes\Psi_P\otimes\Psi_{R_1}\otimes\Psi_{R_2} \vdash M' \leftrightarrows K'$.

By Lemma 42 we get that $\Psi_{R_2}\otimes\Psi\otimes\Psi_P \rhd R_1 \xrightarrow{\overline{K'}\,(\nu\widetilde{a})N} R_{P1}$ and that $\Psi_{R_1}\otimes\Psi\otimes\Psi_P \rhd R_2 \xrightarrow{M'\,N} R_{P2}$. By induction we learn that $\Psi_{R_2}\otimes\Psi\otimes\Psi_Q \rhd R_1 \xrightarrow{\overline{K'}\,(\nu\widetilde{a})N} R_{Q1}$ and $\Psi_{R_1}\otimes\Psi\otimes\Psi_Q \rhd R_2 \xrightarrow{M'\,N} R_{Q2}$ such that $\Psi_{R_2}\otimes\Psi \rhd_{B\cup\widetilde{b}_{R_2}} (P|R_{P1})\ \overline{\mathcal{S}}\ (Q|R_{Q1})$ and $\Psi_{R_1}\otimes\Psi \rhd_{B\cup\widetilde{b}_{R_1}} (P|R_{P2})\ \overline{\mathcal{S}}\ (Q|R_{Q2})$.

Since $\widetilde{b}_P\#K', M'$ we get that $\Psi\otimes\mathcal{F}(P)\otimes\Psi_{R_1}\otimes\Psi_{R_2} \vdash M' \leftrightarrows K'$. From $\Psi\otimes\Psi_{R_1}\otimes\Psi_{R_2} \rhd P \dot{\sim}^{\mathrm{HO}} Q$ we then get that $\Psi\otimes\mathcal{F}(Q)\otimes\Psi_{R_1}\otimes\Psi_{R_2} \vdash M' \leftrightarrows K'$. We finally get that $\Psi\otimes\Psi_Q\otimes\Psi_{R_1}\otimes\Psi_{R_2} \vdash M' \leftrightarrows K'$, permitting the following derivation:

$$\textsc{Com}\ \frac{\begin{array}{c}\Psi_{R_2}\otimes\Psi\otimes\Psi_Q \rhd R_1 \xrightarrow{\overline{K'}\,(\nu\widetilde{a})N} R_{Q1} \\ \Psi_{R_1}\otimes\Psi\otimes\Psi_Q \rhd R_2 \xrightarrow{M'\,N} R_{Q2} \qquad \Psi\otimes\Psi_Q\otimes\Psi_{R_1}\otimes\Psi_{R_2} \vdash M' \leftrightarrows K'\end{array}}{\Psi\otimes\Psi_Q \rhd R_1 \mid R_2 \xrightarrow{\tau} (\nu\widetilde{a})(R_{Q1} \mid R_{Q2})}$$

Assume that $\mathcal{F}(R_{P2}) = (\nu\widetilde{b}_{R_{P2}})\Psi_{R_{P2}}$ and $\mathcal{F}(R_{Q1}) = (\nu\widetilde{b}_{R_{Q1}})\Psi_{R_{Q1}}$ with $\widetilde{b}_{R_{Q1}}\#\widetilde{b}_{R_{P2}}$ and $\widetilde{b}_{R_{Q1}}\widetilde{b}_{R_{P2}}\#\Psi, P, Q, R, N$. Since $\Psi_{R_{P2}} \simeq \Psi_{R_2}\otimes\Psi_2$ for some Ψ_2 we have $\Psi_{R_{P2}}\otimes$

$\Psi \rhd_{B \cup \widetilde{b}_{R_2}} (P|R_{P1})\ \overline{\mathcal{S}}\ (Q|R_{Q1})$ by Lemma 39. Then $\Psi \rhd_B (P|R_{P1} \mid R_{P2})\ \overline{\mathcal{S}}\ (Q|R_{Q1} \mid R_{P2})$ by Lemma 39.1. Similarly, $\Psi \rhd_B (P|R_{Q1} \mid R_{P2})\ \overline{\mathcal{S}}\ (Q|R_{Q1} \mid R_{Q2})$.
By symmetry of $\dot{\sim}^{\text{HO}}$ we get that $\Psi \otimes \Psi_R \rhd Q \dot{\sim}^{\text{HO}} P$, and by extension of arbitrary assertion (Definition 27.3) we get $\Psi \otimes \Psi_{R_{Q1}} \otimes \Psi_{R_{P2}} \rhd Q \dot{\sim}^{\text{HO}} P$.
By the definition of $\mathcal{S}$ we get $\Psi \rhd (Q \mid R_{Q1} \mid R_{P2})\ \mathcal{S}\ (P \mid R_{Q1} \mid R_{P2})$. Since $B \# R_1, N$ we then have $\Psi \rhd_B (Q \mid R_{Q1} \mid R_{P2})\ \overline{\mathcal{S}}\ (P \mid R_{Q1} \mid R_{P2})$. By transitivity of $\overline{\mathcal{S}}$ we then get $\Psi \rhd_B (P \mid R_{P1} \mid R_{P2})\ \overline{\mathcal{S}}\ (Q \mid R_{Q1} \mid R_{Q2})$. Using Lemma 39, $\Psi \rhd_B P \mid (\nu \widetilde{a})(R_{P1} \mid R_{P2})\ \overline{\mathcal{S}}\ Q \mid (\nu \widetilde{a})(R_{Q1} \mid R_{Q2})$. □

A variant of Lemma 43 treats the case where R makes a transition in the environment of P that can communicate with a transition of P in environment of R. The processes R and Q can then perform matching transitions, leading to $\overline{\mathcal{S}}$-related derivatives.

Lemma 44 (subject switching lemma)**.**

$$
\begin{array}{rll}
 & \Psi \otimes \Psi_R \rhd P \dot{\sim}^{\text{HO}} Q & \\
\wedge & \mathcal{F}(P) = (\nu \widetilde{b}_P)\Psi_P & \\
\wedge & \mathcal{F}(Q) = (\nu \widetilde{b}_Q)\Psi_Q & \\
\wedge & \mathcal{F}(R) = (\nu \widetilde{b}_R)\Psi_R & \\
\wedge & \Psi \otimes \Psi_R \rhd P \xrightarrow{\overline{M}\,(\nu \widetilde{a})N} P' & \\
\wedge & \Psi \otimes \Psi_P \rhd R \xrightarrow{M\,N} R_P & \\
\wedge & \Psi \otimes \Psi_P \otimes \Psi_R \vdash K \leftrightarrow M & \\
\wedge & \widetilde{b}_P \# R, M, N, \Psi, P, Q & \\
\wedge & \widetilde{b}_Q \# R, M, N, \Psi, P, Q & \\
\wedge & \widetilde{b}_R \# K, N, \Psi, P, \widetilde{b}_P, \Psi_P, Q, \widetilde{b}_Q, \Psi_Q, R & \\
\wedge & \widetilde{a} \# M, \Psi, P, Q, R, \widetilde{b}_P, \widetilde{b}_Q & \\
\wedge & B \# P, Q, R, N, \widetilde{a}, \widetilde{b}_P, \widetilde{b}_Q, P' & \\
\Longrightarrow & \exists M', R_Q, Q'. & \\
 & \widetilde{b}_R, B \# M' & \\
\wedge & \Psi \otimes \Psi_Q \otimes \Psi_R \vdash K \leftrightarrow M' & \\
\wedge & \Psi \otimes \Psi_Q \rhd R \xrightarrow{M'\,N} R_Q & \\
\wedge & \Psi \otimes \Psi_R \rhd Q \xrightarrow{\overline{M}\,(\nu \widetilde{a})N} Q' & \\
\wedge & \Psi \otimes \Psi_P \rhd_B (P' \mid R_P)\ \overline{\mathcal{S}}\ (Q' \mid R_Q) &
\end{array}
$$

Proof. By induction on the transition of R, similar to Lemma 43. □

The statement of Lemma 44 also needs to hold for output transitions of R, mutatis mutandis. We can then show the desired result.

Theorem 45. *$\mathcal{S}$ is a HO-bisimulation up to $T \circ U \circ R$.*

Proof sketch.. Assume that $\Psi \rhd P|R\ \mathcal{S}\ Q|R$, i.e. that $\Psi \otimes \mathcal{F}(R) \rhd P \dot{\sim}^{\text{HO}} Q$. Symmetry, extension with arbitrary assertion, and static equivalence of conditions follow from the same properties of $\dot{\sim}^{\text{HO}}$. Static equivalence of clauses up to $T \circ U \circ R$ follows from the static equivalence of clauses of $\Psi \otimes \mathcal{F}(R) \rhd P \dot{\sim}^{\text{HO}} Q$.

We prove simulation up to $T \circ U \circ R$ by case analysis on the derivation of the transition of $P \mid R$. Recall that $\Psi \rhd P'\ (T(\mathcal{R}))\ Q'$ iff $\Psi \rhd_\emptyset P'\ (T_{\mathrm{a}}(\mathcal{R}))\ Q'$.

Par-L By bisimilarity of P and Q.

Par-R Here $\Psi \otimes \Psi_P \rhd R \xrightarrow{\alpha} R_P$. By Lemma 43 $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\alpha} R_Q$ with $\Psi \rhd_\emptyset P \mid R_P\ \overline{\mathcal{S}}\ Q \mid R_Q$.

Com-L Here $\Psi \otimes \Psi_R \rhd P \xrightarrow{\overline{K}\,(\nu\widetilde{a})N} P'$, $\Psi \otimes \Psi_P \rhd R \xrightarrow{M\,N} R_P$, $\Psi \otimes \Psi_P \otimes \Psi_R \vdash K \leftrightarrows M$ and $\widetilde{b}_P \# K$ and $\widetilde{b}_R \# M$. We may assume that $\widetilde{a} \# \Psi_R$.

By bisimilarity $\Psi \otimes \Psi_R \rhd Q \xrightarrow{\overline{K}\,(\nu\widetilde{a})N} Q'$ with $\Psi \otimes \Psi_R \rhd P' \mathrel{\dot{\sim}}^{\text{HO}} Q'$. By Lemma 44 there are M', R_Q with $\widetilde{b}_R \# M'$ such that $\Psi \otimes \Psi_Q \otimes \Psi_R \vdash K \leftrightarrows M'$ and $\Psi \otimes \Psi_Q \rhd R \xrightarrow{M'\,N} R_Q$ and $\Psi \rhd_\emptyset (P' \mid R_P)\ \overline{\mathcal{S}}\ (Q' \mid R_Q)$.

Finally, by Lemma 39 we get $\Psi \rhd_\emptyset (\nu\widetilde{a})(P' \mid R_P)\ \overline{\mathcal{S}}\ (\nu\widetilde{a})(Q' \mid R_Q)$.

Com-R As Com-L. □

It now follows that Theorem 31(1) holds.

Corollary 46. $P \mathrel{\dot{\sim}}^{\text{HO}}_{\Psi} Q \Longrightarrow P \mid R \mathrel{\dot{\sim}}^{\text{HO}}_{\Psi} Q \mid R$.

Proof. Assume that $\mathcal{F}(R) = (\nu\widetilde{b}_R)\Psi_R$ with $\widetilde{b}_R \# \Psi, P, Q$. By extension of arbitrary assertion we get $P \mathrel{\dot{\sim}}^{\text{HO}}_{\Psi \otimes \Psi_P} Q$, so $\Psi \rhd (P|R)\ \mathcal{S}\ (Q|R)$ by the definition of $\mathcal{S}$. By Theorem 45 and Theorem 37 we get $\mathcal{S} \subseteq \mathrel{\dot{\sim}}^{\text{HO}}$, so $P \mid R \mathrel{\dot{\sim}}^{\text{HO}}_{\Psi} Q \mid R$. □

4.5. *Formal proofs*

All theorems in this paper have been machine-checked with the interactive theorem prover Isabelle. The proof scripts (ÅPR) are adapted and extended from Bengtson's formalisation of psi-calculi (Ben10). They constitute 63334 lines of Isabelle code; Bengtson's code is 37417 lines. The bulk of the new code pertains to Theorems 26 and 31, which have quite involved proofs that depart significantly from Bengtson's. It is interesting to observe how wildly the effort involved in conducting the proofs varies. We briefly recount our experiences here.

With only minor modifications to Bengtson's proofs, we were able to re-prove all of the meta-theoretical results for psi-calculi (Theorems 17, 19, 20, and 21) in a matter of days. We believe that situations like these, where results need to be reestablished under slightly different definitions, are among those where theorem provers truly shine.

By contrast, HO-bisimulation (Theorems 29, 31, and 28) is an example of a small change to the definitions which gave rise to man-months of work rather than days. This is because certain technical lemmas on which the old proofs depend are no longer valid in the context of HO-bisimulation. Hence, completely new proofs and proof ideas had to be developed. However, with HO-bisimulation in place, HO-congruence (Theorem 32) was mechanised in a matter of minutes.

The proofs pertaining to canonical instances and the encoding of operators (Theorems 15, 24, 25, and 26) also represent man-months of work, but for different reasons. Here simple and intuitive proof ideas turned out to be cumbersome to mechanise. In the case of Theorem 15, the encoding of canonical instances is complicated and unintuitive, because of the necessity to sidestep certain technical restrictions in the framework; for an

example, nominal datatype definitions cannot depend on locale parameters. Theorem 26 gives rise to almost 9000 lines of proof script, even though the proof is conceptually simple. The main problem is the unwieldy candidate relation used for the proof, which includes many assumptions about the underlying psi-calculus. Moreover, it is closed under parallel composition and restriction, which significantly increases the size of the transition derivation trees we must follow and the amount of manual alpha-conversion we must perform, respectively. We believe that a much shorter proof can be obtained if a bisimulation up-to context technique is used instead, but we do not currently have a proof that such a technique is sound.

4.6. *Comparing higher-order equivalences*

Our definition of HO-bisimilarity is technically nontrivial and we here motivate it. Our primary concern is to not depart too much from the original bisimilarity since we have invested a substantial effort in an Isabelle proof repository and strive to re-use as much as possible. Therefore our approach is to amend the original definition as little as possible in order to validate Theorem 28. Even so, there are a number of alternatives in the precise formulation of Clause 1(b). The current definition requires in the conclusion that $\mathcal{R}(\mathbf{1}, P', Q')$, i.e., that P' and Q' are again bisimilar in the assertion $\mathbf{1}$, which by Clause 3 is the same as requiring $\forall\Psi.\mathcal{R}(\Psi, P', Q')$. As a consequence, the following strengthening of Theorem 28 (note the assertions Ψ) is not true in general:

$$\Psi \rhd P \mathrel{\dot{\sim}}^{\text{HO}} Q \Rightarrow \Psi \rhd (\!|\Psi^{M \Leftarrow P}|\!) \mathrel{\dot{\sim}}^{\text{HO}} (\!|\Psi^{M \Leftarrow Q}|\!)$$

We have failed to define a version of higher-order bisimilarity where this holds. An obvious attempt is to adjust Clause 1(b) to use $(\Psi, P', Q') \in \mathcal{R}$, i.e., with Ψ in place of $\mathbf{1}$, but with this we fail to prove Theorem 31.1, i.e., that bisimilarity is preserved by parallel composition. The reason is that our proof strategy using the relation $\mathcal{S}$ in Section 4.4 relies on the fact that

$$\Psi \otimes \Psi' \rhd P \mathrel{\dot{\sim}} Q \quad \Rightarrow \quad \Psi \rhd (\!|\Psi'|\!) \mid P \mathrel{\dot{\sim}} (\!|\Psi'|\!) \mid Q$$

This holds for ordinary bisimulation and for higher-order bisimulation, but fails if Clause 1(b) uses $(\Psi, P', Q') \in \mathcal{R}$. The counterexample is somewhat artificial and it remains to be seen if we can formulate a subset of higher-order calculi where this property holds, or if there is a different proof strategy for Theorem 31.1 that does not require the property, involving another candidate bisimulation relation.

Another possibility would be to include even more information in the assertion, as in $(\Psi \otimes \mathcal{F}(P), P', Q') \in \mathcal{R}$. In this case we instead fail to establish that HO-bisimilarity is transitive; again we do not know if there is a counterexample. The problems are highly technical and mainly involves how freshness conditions are propagated in the proofs.

The definition does not aspire to full abstraction with respect to observational criteria, and in this way it is very different from most existing work on higher-order calculi. It can immediately be seen that it is not complete in any sensible respect: the agents $\mathbf{0}$ and $(\!|\{M \Leftarrow \mathbf{0}\}|\!)$ should be indistinguishable from an observation viewpoint since neither has a transition and $M \Leftarrow \mathbf{0}$ does not give M any invocation possibilities, yet they fail

bisimilarity on Clause 1(b). On the other hand it is straightforward to establish soundness for reasonable criteria. For example, say that a process P has the barb M if P has a transition with subject M, and that a congruence relation is barbed if related agents have the same barbs. Clause 4 in Definition 27 directly gives that HO-bisimilarity is barbed.

5. Conclusion

We have defined higher-order psi-calculi in a smooth extension from ordinary psi-calculi, meaning that we can re-use much of the mechanised proofs. Ordinary psi-calculi can be lifted in a systematic way to higher-order counterparts, yielding higher-order versions of the applied pi-calculus and the concurrent constraint pi-calculus.

We have integrated the proofs with our existing proof repositories based on Isabelle/Nominal. In some cases this process is surprisingly easy. In other places there are roadblocks related to the exact working of nominal datatypes with complicated constructors and locales. Yet we regard this effort as worthwhile. For the main results like Theorem 31 it is not efficient to embark on manual proofs; in psi-calculi these are notoriously error-prone because of the length, the number of cases to check, and the numerous side conditions related to freshness of names.

There are several interesting avenues to explore. One obvious is higher-order weak bisimulation and congruence. Here an immediate problem is that we can encode Sum, and therefore the usual example that weak bisimulation is not preserved by Sum may imply that it is not preserved by Parallel. For example, let us define weak higher order bisimulation by adapting the weak bisimulation from (BJPV10) in the same way as we here do for strong bisimulation. In other words, we require that a clause needs a weakly bisimilar clause. Then consider the agents

$$\begin{array}{lcl} P & = & (\!|\{M \Leftarrow \tau\,.\,a\,.\,\mathbf{0}\}|\!) \\ Q & = & (\!|\{M \Leftarrow a\,.\,\mathbf{0}\}|\!) \\ R & = & \mathbf{run}\ M \mid (\!|\{M \Leftarrow b\,.\,\mathbf{0}\}|\!) \end{array}$$

Here we have that P and Q are weakly higher-order bisimilar but $P|R$ and $Q|R$ are not. This indicates that a less straightforward definition will be necessary. An obvious attempt is to require of clauses that they are weakly congruent (rather than weakly bisimilar), and this requires that both weak congruence and weak bisimilarity are defined in one simultaneous co-inductive definition, since each depends on the other.

The relationship between a calculus and its canonical higher-order counterpart should also be investigated. For example, bisimilarity on first-order processes is hopefully the same, and perhaps there is an interesting class of calculi where the canonical higher-order calculus can be encoded. Finally, higher-order calculi should be combined with other extensions of the psi-calculi framework. We have successfully integrated higher-order calculi and ordinary bisimulation with the broadcast extension presented in (BHJ$^+$11). Here the total effort in the formalisation was roughly half a day, mainly to textually combine the proof files. This is a striking advantage of using formal proof repositories. We could also extend our recent work on sort systems (BGP$^+$12) to a higher-order setting.

In the invocation rule, the handle M must be exactly the same in the premise (where it

occurs in $M \Leftarrow P$) and conclusion (where it occurs in **run** M). This means that it is not possible to directly describe extraction of handles from complicated data structures. For example, consider one process defining two clauses $M_i \Leftarrow P_i$, and then sending the pair of the handles $\langle M_1, M_2 \rangle$. A receiving process might want to receive the pair and invoke its first element. Expressing this as $a(x) \,.\, \mathbf{run}\ \pi_1(x)$ will not work. After the communication of $\langle M_1, M_2 \rangle$ this becomes $\mathbf{run}\ \pi_1(\langle M_1, M_2 \rangle)$ but the environment contains $M_1 \Leftarrow P_1$ and not $\pi_1(\langle M_1, M_2 \rangle) \Leftarrow P_1$. What would be necessary here is a rewriting theory of projections, with axioms such as $\pi_1(\langle M_1, M_2 \rangle) \to M_1$, to be used in the entailment relation.

Most cases of simple extractions such as projections can be handled by pattern matching, as in this case $a(\lambda x, y)\langle x, y \rangle \,.\, \mathbf{run}\ x$. In more complicated structures, for example to represent encryption and decryption of handles, pattern matching will not be sufficient and we must include information about the evaluation of handles in the assertions, where scoping can be used to make them local. This device is is already present for communication subjects as the channel equivalence predicate. It remains to be seen if it is feasible to introduce a similar relation for handles.

Acknowledgements We are very grateful to Magnus Johansson and Björn Victor for constructive and inspiring discussions.

References

Johannes Åman Pohjola and Palle Raabjerg. Isabelle proofs for higher-order psi-calculi. Proof scripts for higher-order psi-calculi. Available at `http://www.it.uu.se/research/group/mobility/theorem/hopsi.tar.gz`.

Jesper Bengtson. *Formalising process calculi*. PhD thesis, Uppsala University, 2010.

Johannes Borgström, Ramūnas Gutkovas, Joachim Parrow, Björn Victor, and Johannes Åman Pohjola. Sorted psi-calculi with generalised pattern matching. Unpublished manuscript, 2012.

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, *SEFM*, volume 7041 of *Lecture Notes in Computer Science*, pages 74–89. Springer, 2011.

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. Full version at `http://user.it.uu.se/~joachim/psi-long.pdf`.

Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Weak equivalences in psi-calculi. In *Proceedings of LICS 2010*, pages 322–331. IEEE, 2010.

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.

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 *LNCS*, pages 99–114. Springer Verlag, August 2009.

Romain Demangeon, Daniel Hirschkoff, and Davide Sangiorgi. Termination in higher-order

concurrent calculi. In Farhad Arbab and Marjan Sirjani, editors, *FSEN*, volume 5961 of *Lecture Notes in Computer Science*, pages 81–96. Springer, 2009.

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

Magnus Johansson. *Psi-calculi: a framework for mobile process calculi*. PhD thesis, Uppsala University, May 2010.

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

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.

Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi, and Alan Schmitt. On the expressiveness and decidability of higher-order process calculi. In *LICS*, pages 145–155. IEEE Computer Society, 2008.

Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi, and Alan Schmitt. On the expressiveness of polyadic and synchronous communication in higher-order process calculi. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *ICALP (2)*, volume 6199 of *Lecture Notes in Computer Science*, pages 442–453. Springer, 2010.

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

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.

Davide Sangiorgi. Bisimulation for higher-order process calculi. *Inf. Comput.*, 131(2):141–178, 1996.

Davide Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8(5):447–479, 1998. An extended abstract appeared in the *Proceedings of MFCS '95*, LNCS 969: 479–488.

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

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

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

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

Chapter 4

A sorted semantics framework for applied process calculi

Paper III

A Sorted Semantic Framework for Applied Process Calculi (extended abstract)

Johannes Borgström, Ramūnas Gutkovas, Joachim Parrow, Björn Victor and Johannes Åman Pohjola

Department of Information Technology, Uppsala University, Sweden

Abstract. Applied process calculi include advanced programming constructs such as type systems, communication with pattern matching, encryption primitives, concurrent constraints, nondeterminism, process creation, and dynamic connection topologies. Several such formalisms, e.g. the applied pi calculus, are extensions of the the pi-calculus; a growing number is geared towards particular applications or computational paradigms.
Our goal is a unified framework to represent different process calculi and notions of computation. To this end, we extend our previous work on psi-calculi with novel abstract patterns and pattern matching, and add sorts to the data term language, giving sufficient criteria for subject reduction to hold. Our framework can accommodate several existing process calculi; the resulting transition systems are isomorphic to the originals up to strong bisimulation. We also demonstrate different notions of computation on data terms, including cryptographic primitives and a lambda-calculus with angelic choice. Substantial parts of the metatheory of sorted psi-calculi have been machine-checked using Nominal Isabelle.

1 Introduction

There is today a growing number of high-level constructs in the area of concurrency. Examples include type systems, communication with pattern matching, encryption primitives, concurrent constraints, nondeterminism, and dynamic connection topologies. Combinations of such constructs are included in a variety of application oriented process calculi. For each such calculus its internal consistency, in terms of congruence results and algebraic laws, must be established independently. Our aim is a framework where many such calculi fit and where such results are derived once and for all, eliminating the need for individual proofs about each calculus.

Our effort in this direction is the framework of psi-calculi [1], which provides machine-checked proofs that important meta-theoretical properties, such as compositionality of bisimulation, hold in all instances of the framework. In this paper we introduce a novel generalization of pattern matching, decoupled from

the definition of substitution, and introduce sorts for data terms and names. The generalized pattern matching is a new contribution that holds general interest; here it allows us to directly capture computation on data in advanced process calculi, without elaborate encodings. We evaluate our framework by providing instances that are isomorphic to standard calculi, and by representing several different notions of computation. This is an advance over our previous work, where we had to resort to nontrivial encodings with unclear formal correspondence to the standard calculi.

1.1 Background: Psi-calculi

A psi-calculus has a notion of data terms, ranged over by K, L, M, N, and we write $\overline{M}\ N\,.\,P$ to represent an agent sending the term N along the channel M (which is also a data term), continuing as the agent P. We write $\underline{K}(\lambda\widetilde{x})X\,.\,Q$ to represent an agent that can input along the channel K, receiving some object matching the pattern X, where $\widetilde{x}$ are the variables bound by the prefix. These two agents can interact under two conditions. First, the two channels must be *channel equivalent*, as defined by the channel equivalence predicate $M \leftrightarrows K$. Second, N must match the pattern X.

Formally, a *transition* is of kind $\Psi \rhd P \xrightarrow{\alpha} P'$, meaning that in an environment represented by the *assertion* Ψ the agent P can do an action α to become P'. An assertion embodies a collection of facts used to infer *conditions* such as the channel equivalence predicate $\leftrightarrows$. To continue the example, if $N = X[\widetilde{x} := \widetilde{L}]$ we will have $\Psi \rhd \overline{M}\ N\,.\,P \mid \underline{K}(\lambda\widetilde{x})X\,.\,Q \xrightarrow{\tau} P \mid Q[\widetilde{x} := \widetilde{L}]$ when additionally $\Psi \vdash M \leftrightarrows K$, i.e. when the assertion Ψ entails that M and K represent the same channel. In this way we may introduce a parametrised equational theory over a data structure for channels. Conditions, ranged over by φ, can be tested in the **if** construct: we have that $\Psi \rhd \mathbf{if}\ \varphi\ \mathbf{then}\ P \xrightarrow{\alpha} P'$ when $\Psi \vdash \varphi$ and $\Psi \rhd P \xrightarrow{\alpha} P'$. In order to represent concurrent constraints and local knowledge, assertions can be used as agents: $(\!|\Psi|\!)$ stands for an agent that asserts Ψ to its environment. Assertions may contain names and these can be scoped; for example, in $P \mid (\nu a)((\!|\Psi|\!) \mid Q)$ the agent Q uses all entailments provided by Ψ, while P only uses those that do not contain the name a.

Assertions and conditions can, in general, form any logical theory. Also the data terms can be drawn from an arbitrary set. One of our major contributions has been to pinpoint the precise requirements on the data terms and logic for a calculus to be useful in the sense that the natural formulation of bisimulation satisfies the expected algebraic laws (see Section 2). It turns out that it is necessary to view the terms and logics as *nominal* [2]. This means that there is a distinguished set of names, and for each term a well defined notion of *support*, intuitively corresponding to the names occurring in the term.

1.2 Extension: Generalized pattern matching

In our original definition of psi-calculi [1] (called "the original psi-calculi" below), patterns are just terms and pattern matching is defined by substitution in the usual way: the output object N matches the pattern X with binders $\widetilde{x}$ iff $N = X[\widetilde{x} := \widetilde{L}]$. In order to increase the generality we now introduce a function MATCH which takes a term N, a sequence of names $\widetilde{x}$ and a pattern X, returning a set of sequences of terms; the intuition is that if $\widetilde{L}$ is in $\text{MATCH}(N, \widetilde{x}, X)$ then N matches the pattern X by instantiating $\widetilde{x}$ to $\widetilde{L}$. The receiving agent $\underline{K}(\lambda\widetilde{x})X \,.\, Q$ then continues as $Q[\widetilde{x} := \widetilde{L}]$.

As an example we consider a term algebra with two function symbols: $\mathtt{enc}$ of arity three and $\mathtt{dec}$ of arity two. Here $\mathtt{enc}(N, n, k)$ means encrypting N with the key k and a random nonce n and and $\mathtt{dec}(N, k)$ represents symmetric key decryption, discarding the nonce. Suppose an agent sends an encryption, as in $\overline{M}\ \mathtt{enc}(N, n, k) \,.\, P$. If we allow all terms to act as patterns, a receiving agent can use $\mathtt{enc}(x, y, z)$ as a pattern, as in $\underline{c}(\lambda x, y, z)\mathtt{enc}(x, y, z) \,.\, Q$, and in this way decompose the encryption and extract the message and key. Using the encryption function as a destructor in this way is clearly not the intention of a cryptographic model. With the new general form of pattern matching, we can simply limit the patterns to not bind names in terms at key position. Together with the separation between patterns and terms, this allows to directly represent dialects of the spi-calculus as in Examples 4 and 5 in Section 3.

Moreover, the generalization makes it possible to safely use rewrite rules such as $\mathtt{dec}(\mathtt{enc}(M, N, K), K) \to M$. In the psi-calculi framework such evaluation is not a primitive concept, but it can be part of the substitution function, with the idea that with each substitution all data terms are normalized according to rewrite rules. Such evaluating substitutions are dangerous for two reasons. First, in the original psi-calculi they can introduce ill-formed input prefixes. The input prefix $\underline{M}(\lambda\widetilde{x})N$ is well-formed when $\widetilde{x} \subseteq \text{n}(N)$, i.e. the names $\widetilde{x}$ must all occur in N; a rewrite of the well-formed $\underline{M}(\lambda y)\mathtt{dec}(\mathtt{enc}(N, y, k), k) \,.\, P$ to $\underline{M}(\lambda y)N \,.\, P$ yields an ill-formed agent when y does not appear in N. Such ill-formed agents could also arise from input transitions in some original psi-calculi; with the current generalization preservation of well-formedness is guaranteed.

Second, in the original psi-calculi there is a requirement that a substitution of $\widetilde{L}$ for $\widetilde{x}$ in M must yield a term containing all names in $\widetilde{L}$ whenever $\widetilde{x} \subseteq \text{n}(M)$. The reason is explained at length in [1]; briefly put, without this requirement the scope extension law is unsound. If rewrites such as $\mathtt{dec}(\mathtt{enc}(M, N, K), K) \to M$ are performed by substitutions this requirement is not fulfilled, since a substitution may then erase the names in N and K. However, a closer examination reveals that this requirement is only necessary for some uses of substitution. In the transition

$$\underline{M}(\lambda\widetilde{x})N.P \xrightarrow{\underline{K}\ N[\widetilde{x}:=\widetilde{L}]} P[\widetilde{x} := \widetilde{L}]$$

the non-erasing criterion is important for the substitution above the arrow ($N[\widetilde{x} := \widetilde{L}]$) but unimportant for the substitution after the arrow ($P[\widetilde{x} := \widetilde{L}]$). In the present paper, we replace the former of these uses by the MATCH function,

where a similar non-erasing criterion applies. All other substitutions may safely use arbitrary rewrites, even erasing ones.

1.3 Extension: Sorting

Applied process calculi often make use of a sort system. The applied pi-calculus [3] has a name sort and a data sort; terms of name sort must not appear as subterms of terms of data sort. It also makes a distinction between input-bound variables (which may be substituted) and restriction-bound names (which may not). The pattern-matching spi-calculus [4] uses a sort of patterns and a sort of implementable terms; every implementable term can also be used as a pattern.

To represent such calculi, we admit a user-defined sort system on names, terms and patterns. Substitutions are only well-defined if they conform to the sorting discipline. To specify which terms can be used as channels, and which values can be received on them, we use compatibility predicates on the sorts of the subject and the object in input and output prefixes. The conditions for preservation of sorting by transitions (subject reduction) are very weak, allowing for great flexibility when defining instances.

The restriction to well-sorted substitution also allows to avoid "junk": terms that exist solely to make substitutions total. A prime example is representing the polyadic pi-calculus as a psi-calculus. The terms that can be transmitted between agents are tuples of names. Since a tuple is a term it can be substituted for a name, even if that name is already part of a tuple. The result is that the terms must admit nested tuples of names, which do not occur in the original calculus.

1.4 Related work.

Pattern-matching is in common use in programming languages (e.g. Lisp, ML, Scala, F#). LINDA [5] uses pattern-matching when receiving from a tuple space. The pattern-matching spi-calculus limits which variables may be binding in a pattern in order to match encrypted messages without binding unknown keys (cf. Example 5). In all these cases, the pattern matching is defined by substitution in the usual way.

Sorts for the pi-calculus were first described by Milner [6]. Hüttel's typed psi-calculi [7] admit a family of dependent type systems, capable of capturing a wide range of earlier type systems for pi-like calculi formulated as instances of psi-calculi. However, the term language of typed psi-calculi is required to be a free term algebra (and without name binders); it uses only the standard notions of substitution and matching, and does not admit any computation on terms. The sophisticated type system of typed psi-calculi is intended for fine-grained control of the behaviour of processes, while we focus on an earlier step: the creation of a calculus that is as close to the modeller's intent as possible. Indeed, sorted psi-calculi gives a formal account of the separation between variables and names in typed psi-calculi, and Hüttel's claim that "the set of well-[sorted] terms is closed under well-[sorted] substitutions, which suffices". Furthermore, we

prove meta-theoretical results including preservation of well-formedness under structural equivalence; no such results exist for typed psi-calculi.

In the applied pi-calculus [3] the data language is a term algebra modulo an equational logic, which is suitable for modelling deterministic computation only. ProVerif [8] is a specialised tool for security protocol verification in an extension of applied pi, including a pattern matching construct. Its implementation allows pattern matching of tagged tuples modulo a user-defined rewrite system; this is strictly less general than the psi-calculus pattern matching described in this paper (cf. Example 2).

Fournet et al. [9] add a general authentication logic to a process calculus with destructor matching; the authentication logic is only used to specify program correctness, and do not influence the operational semantics in any way. A comparison of expressiveness to calculi with communication primitives other than binary directed communication, such as the concurrent pattern calculus [10] and the join-calculus [11], would be interesting. We here inherit positive results from the pi calculus, such as the encoding of the join-calculus.

1.5 Results and outline

In Section 2 we define psi-calculi with the above extensions and explain the necessary change to the semantics. A formal account of the whole operational semantics and bisimulations can be found in an appendix. Our results are the usual algebraic properties of bisimilarity, preservation of well-formedness, and subject reduction.

We demonstrate the expressiveness of our generalization in Section 3 by directly representing calculi with advanced data structures and computations on them, even nondeterministic reductions.

2 Definitions

Psi-calculi are based on nominal data types. A nominal data type is similar to a traditional data type, but can also contain binders and identify alpha-variants of terms. Formally, the only requirements are related to the treatment of the atomic symbols called names as explained below. In this paper, we consider sorted nominal datatypes, where names may have different sorts.

We assume a set of sorts $\mathcal{S}$. Given a countable set of sorts for names $\mathcal{S}_\mathcal{N} \subseteq \mathcal{S}$, we assume countably infinite pair-wise disjoint sets of atomic *names* $\mathcal{N}_s$, where $s \in \mathcal{S}_\mathcal{N}$. The set of all names, $\mathcal{N} = \cup_s \mathcal{N}_s$, is ranged over by $a, b, \ldots, x, y, z$. We write $\widetilde{x}$ for a tuple of names $x_1, \ldots, x_n$ and similarly for other tuples, and $\widetilde{x}$ stands for the set of names $\{x_1, \ldots, x_n\}$ if used where a set is expected.

A sorted *nominal set* [2, 12] is a set equipped with *name swapping* functions written $(a\ b)$, for any sort s and names $a, b \in \mathcal{N}_s$, i.e. name swappings must respect sorting. An intuition is that for any member T it holds that $(a\ b) \cdot T$ is T with a replaced by b and b replaced by a. The support of a term, written $\mathrm{n}(T)$, is intuitively the set of names affected by name swappings on T. This definition of

support coincides with the usual definition of free names for abstract syntax trees that may contain binders. We write $a\#T$ for $a \notin \mathrm{n}(T)$, and extend this to finite sets and tuples by conjunction. A function f is equivariant if $(a\ b)(f(T)) = f((a\ b)T)$ always. A *nominal data type* is a nominal set together with some functions on it, for instance a substitution function.

2.1 Original Psi-calculi Parameters

Sorted psi-calculi is an extension of the original psi-calculi framework [1].

Definition 1 (Original psi-calculus parameters). *The psi-calculus parameters from the original psi-calculus include three nominal data types: (data) terms $M, N \in \mathbf{T}$, conditions $\varphi \in \mathbf{C}$, and assertions $\Psi \in \mathbf{A}$; and four equivariant operators: channel equivalence $\leftrightarrow : \mathbf{T} \times \mathbf{T} \to \mathbf{C}$, assertion composition $\otimes : \mathbf{A} \times \mathbf{A} \to \mathbf{A}$, the unit assertion $\mathbf{1}$, and the entailment relation $\vdash \subseteq \mathbf{A} \times \mathbf{C}$.*

The binary functions $\leftrightarrow, \otimes$ and the relation $\vdash$ above will be used in infix form.

Two assertions are equivalent, written $\Psi \simeq \Psi'$, if they entail the same conditions, i.e. for all φ we have that $\Psi \vdash \varphi \Leftrightarrow \Psi' \vdash \varphi$. We impose certain requisites on the sets and operators. In brief, channel equivalence must be symmetric and transitive, the assertions with $(\otimes, \mathbf{1})$ must form an abelian monoid modulo $\simeq$, and $\otimes$ must be compositional w.r.t. $\simeq$ (i.e. $\Psi_1 \simeq \Psi_2 \implies \Psi \otimes \Psi_1 \simeq \Psi \otimes \Psi_2$). For details see [1].

2.2 New parameters for generalized pattern-matching

To the parameters of the original psi-calculi we add patterns X, Y, that are used in input prefixes, a pattern-matching function MATCH, which is used when the input takes place, and a function VARS which yields the possible combinations of binding names in the pattern. Below, we use "variable" for names that can be bound in a pattern.

Definition 2 (Psi-calculus parameters for pattern-matching). *The psi-calculus parameters for pattern-matching include the nominal data type $\mathbf{X}$ of (input) patterns, ranged over by X, Y, and the two equivariant operators*

$$\begin{array}{rll} \textsc{match} : \mathbf{T} \times \mathcal{N}^* \times \mathbf{X} \to \mathcal{P}(\mathbf{T}^*) & & \textit{Pattern matching} \\ \textsc{vars} : \mathbf{X} \to \mathcal{P}(\mathcal{P}(\mathcal{N})) & & \textit{Pattern variables} \end{array}$$

Intuitively, if $\widetilde{L} \in \textsc{match}(N, \widetilde{x}, X)$ then an output of the term N matches an input with the pattern X, binding $\widetilde{x}$, and the receiving agent continues after substituting $\widetilde{L}$ for $\widetilde{x}$.

The VARS operator gives the possible sets of names in a pattern which are bound by an input prefix. For example, an input prefix with a pairing pattern $\langle x, y\rangle$ may bind both x and y, only one of them, or none, so $\textsc{vars}(\langle x, y\rangle) = \{\{x, y\}, \{x\}, \{y\}, \{\}\}$. This way, we can let the input prefix $\underline{c}(\lambda x)\langle x, y\rangle$ only match pairs where the second argument is the name y. To model a calculus

where input patterns cannot be selective in this way, we may instead define $\text{VARS}(\langle x, y\rangle) = \{\{x, y\}\}$. This ensures that input prefixes that use the pattern $\langle x, y\rangle$ must be of the form $\underline{M}(\lambda x, y)\langle x, y\rangle$, where both x and y are bound.

Requirements on VARS and MATCH are given below in Definition 5. Note that the four data types $\mathbf{T}$, $\mathbf{C}$, $\mathbf{A}$ and $\mathbf{X}$ are not required to be disjoint. In most of the examples in this paper, the patterns $\mathbf{X}$ is a subset of the terms $\mathbf{T}$.

2.3 New parameters for sorting

To the parameters defined above we add a sorting function and four sort compatibility predicates.

Definition 3 (Psi-calculus parameters for sorting). *The psi-calculus parameters for sorting include the sorting function* $\text{SORT} : \mathcal{N} \uplus \mathbf{T} \uplus \mathbf{X} \to \mathcal{S}$*, and the four compatibility predicates*

$$\begin{array}{rl} \underline{\propto} \subseteq \mathcal{S} \times \mathcal{S} & \textit{Can be used to receive} \\ \overline{\propto} \subseteq \mathcal{S} \times \mathcal{S} & \textit{Can be used to send} \\ \prec \subseteq \mathcal{S} \times \mathcal{S} & \textit{Can be substituted by} \\ \mathcal{S}_\nu \subseteq \mathcal{S} & \textit{Can be bound by name restriction} \end{array}$$

The SORT operator gives the sort of a name, term or pattern; on names we require that $\text{SORT}(a) = s$ iff $a \in \mathcal{N}_s$. The sort compatibility predicates are used to restrict where terms and names of certain sorts may appear in processes. Terms of sort s can be used to send values of sort t if $s \mathrel{\overline{\propto}} t$. Dually, a term of sort s can be used to receive with a pattern of sort t if $s \mathrel{\underline{\propto}} t$. A name a can be used in a restriction (νa) if $\text{SORT}(a) \in \mathcal{S}_\nu$. If $\text{SORT}(a) \prec \text{SORT}(M)$ we can substitute the term M for the name a. In most of our examples, $\prec$ is a subset of the equality relation. These predicates can be chosen freely, although the set of well-formed substitutions depends on $\prec$, as detailed in Definition 4 below.

2.4 Substitution and Matching

We require that each datatype is equipped with an equivariant substitution function, which intuitively substitutes terms for names. The requisites on substitution differ from the original psi-calculi as indicated in the Introduction. Substitutions must preserve or refine sorts, and bound pattern variables must not be removed by substitutions.

We define a subsorting preorder $\leq$ on $\mathcal{S}$ as $s_1 \leq s_2$ if s_1 can be used as a channel or message whenever s_2 can be: formally $s_1 \leq s_2$ iff $\forall t \in \mathcal{S}.(s_2 \mathrel{\underline{\propto}} t \Rightarrow s_1 \mathrel{\underline{\propto}} t) \wedge (s_2 \mathrel{\overline{\propto}} t \Rightarrow s_1 \mathrel{\overline{\propto}} t) \wedge (t \mathrel{\underline{\propto}} s_2 \Rightarrow t \mathrel{\underline{\propto}} s_1) \wedge (t \mathrel{\overline{\propto}} s_2 \Rightarrow t \mathrel{\overline{\propto}} s_1)$. This relation compares sorts of terms, and so does not have any formal relationship to $\prec$ (which relates the sort of a name to the sort of a term).

Definition 4 (Substitution). *If* $\widetilde{a}$ *is a sequence of distinct names and* $\widetilde{N}$ *is an equally long sequence of terms such that* $\text{SORT}(a_i) \prec \text{SORT}(N_i)$ *for all* i*, we say that* $[\widetilde{a} := \widetilde{N}]$ *is a* substitution. *Substitutions are ranged over by* σ.

For each data type among $\mathbf{T}, \mathbf{A}, \mathbf{C}$ *we define substitution on elements* T *of that data type as follows: we require that* $T\sigma$ *is an element of the same data type, and that if* $(\widetilde{a}\ \widetilde{b})$ *is a (bijective) name swapping such that* $\widetilde{b}\#T, \widetilde{a}$ *then* $T[\widetilde{a} := \widetilde{N}] = ((\widetilde{a}\ \widetilde{b}).T)[\widetilde{b} := \widetilde{N}]$ *(alpha-renaming of substituted variables). For terms we additionally require that* $\text{SORT}(M\sigma) \leq \text{SORT}(M)$.

For substitution on patterns $X \in \mathbf{X}$*, we require that if* $\widetilde{x} \in \text{VARS}(X)$ *and* $\widetilde{x}\#\sigma$ *then* $X\sigma \in \mathbf{X}$ *and* $\text{SORT}(X\sigma) \leq \text{SORT}(X)$ *and* $\widetilde{x} \in \text{VARS}(X\sigma)$ *and alpha-renaming of substituted variables (as above) holds.*

Intuitively, the requirements on substitutions on patterns ensure that a substitution on a pattern with binders $((\lambda\widetilde{x})X)\sigma$ with $\widetilde{x} \in \text{VARS}(X)$ and $\widetilde{x}\#\sigma$ yields a pattern $(\lambda\widetilde{x})Y$ with $\widetilde{x} \in \text{VARS}(Y)$. As an example, consider the pair patterns discussed above with $\mathbf{X} = \{\langle x, y\rangle : x \neq y\}$ and $\text{VARS}(\langle x, y\rangle) = \{\{x, y\}\}$. We can let $\langle x, y\rangle\sigma = \langle x, y\rangle$ when $x, y\#\sigma$. Since $\text{VARS}(\langle x, y\rangle) = \{\{x, y\}\}$ the pattern $\langle x, y\rangle$ in a well-formed agent will always occur directly under the binder $(\lambda x, y)$, i.e. in $(\lambda x, y)\langle x, y\rangle$, and here a substitution for x or y will have no effect. It therefore does not matter what e.g. $\langle x, y\rangle[x := M]$ is, since it will never occur in derivations of transitions of well-formed agents. We could think of substitutions as partial functions which are undefined in such cases; formally, since substitutions are total, the result of this substitution can be assigned an arbitrary value.

Matching must be invariant under renaming of pattern variables, and the substitution resulting from a match must not contain any names that are not from the matched term or the pattern:

Definition 5 (Generalized pattern matching). *For the function* MATCH *we require that if* $\widetilde{x} \in \text{VARS}(X)$ *are distinct and* $\widetilde{N} \in \text{MATCH}(M, \widetilde{x}, X)$ *then it must hold that* $[\widetilde{x} := \widetilde{N}]$ *is a substitution, that* $\text{n}(\widetilde{N}) \subseteq \text{n}(M) \cup (\text{n}(X) \setminus \widetilde{x})$*, and that for all name swappings* $(\widetilde{x}\ \widetilde{y})$ *we have* $\widetilde{N} \in \text{MATCH}(M, \widetilde{y}, (\widetilde{x}\ \widetilde{y})X)$ *(alpha-renaming of matching).*

In the opriginal psi-calculi equivariance of matching is imposed as a requirement on substitution on terms, but there is no requirement that substitutions preserve pattern variables. For this reason, the original psi semantics does not preserve the well-formedness of agents (an input prefix $\underline{M}(\lambda\widetilde{x})N\,.\,P$ is well-formed when $\widetilde{x} \subseteq \text{n}(N)$), although this is assumed by the operational semantics [1]. In contrast, the semantics of pattern-matching psi-calculi does preserve well-formedness, as shown below in Theorem 1.

In many process calculi, and also in the symbolic semantics of psi [13], the input construct binds a single variable. This is a trivial instance of pattern matching where the pattern is a single bound variable, matching any term.

Example 1. Given values for the other requisites, we can take $\mathbf{X} = \mathcal{N}$ with $\text{VARS}(a) = \{a\}$, meaning that the pattern variable must always occur bound, and $\text{MATCH}(M, a, a) = \{M\}$ if $\text{SORT}(a) \prec \text{SORT}(M)$. On patterns we define substitution as $a\sigma = a$ when $a\#\sigma$.

2.5 Agents

Definition 6 (Agents). *The* agents*, ranged over by* $P, Q, \ldots$*, are of the following forms.*

$\overline{M}\ N.P$	Output
$\underline{M}(\lambda\widetilde{x})X.P$	Input
$\mathbf{case}\ \varphi_1 : P_1\ [\!]\ \cdots\ [\!]\ \varphi_n : P_n$	Case
$(\nu a)P$	Restriction
$P \mid Q$	Parallel
$!P$	Replication
$(\!\vert\Psi\vert\!)$	Assertion

In the Input any name in $\widetilde{x}$ *binds its occurrences in both* X *and* P*, and in the Restriction* a *binds in* P*. An assertion is* guarded *if it is a subterm of an Input or Output. An agent is* well-formed *if, for all its subterms, in a replication* $!P$ *there are no unguarded assertions in* P*, and in* $\mathbf{case}\ \varphi_1 : P_1\ [\!]\ \cdots\ [\!]\ \varphi_n : P_n$ *there are no unguarded assertion in any* P_i*. Substitution on agents is defined inductively on their structure, using the substitution function of each datatype based on syntactic position, avoiding name capture.*

In comparison to [1] we restrict the syntax of well-formed agents by imposing requirements on sorts: the subjects and objects of prefixes must have compatible sorts, and restrictions may only bind names of a sort in $\mathcal{S}_\nu$.

Definition 7. *In sorted psi-calculi, an agent is* well-formed *if additionally the following holds for all its subterms. In an Output* $\overline{M}\ N.P$ *we require that* $\text{SORT}(M) \mathrel{\overline{\propto}} \text{SORT}(N)$*. In an Input* $\underline{M}(\lambda\widetilde{x})X.P$ *we require that* $\widetilde{x} \in \text{VARS}(X)$ *is a tuple of distinct names and* $\text{SORT}(M) \mathrel{\underline{\propto}} \text{SORT}(X)$*. In a Restriction* $(\nu a)P$ *we require that* $\text{SORT}(a) \in \mathcal{S}_\nu$*.*

The output prefix $\overline{M}\ N.P$ sends N on a channel that is equivalent to M. Dually, $\underline{M}(\lambda\widetilde{x})X.P$ receives a message matching the pattern X from a channel equivalent to M. A non-deterministic case statement $\mathbf{case}\ \varphi_1 : P_1\ [\!]\ \cdots\ [\!]\ \varphi_n : P_n$ executes one of the branches P_i where the corresponding condition φ_i holds, discarding the other branches. Restriction $(\nu a)P$ scopes the name a in P; the scope of a may be extruded if P communicates a data term containing a. A parallel composition $P \mid Q$ denotes P and Q running in parallel; they may proceed independently or communicate. A replication $!P$ models an unbounded number of copies of the process P. The assertion $(\!\vert\Psi\vert\!)$ contributes Ψ to its environment. We often write $\mathbf{if}\ \varphi\ \mathbf{then}\ P$ for $\mathbf{case}\ \varphi : P$, and nothing or $\mathbf{0}$ for the empty case statement $\mathbf{case}$.

2.6 Semantics and Bisimulation

The semantics of a psi-calculus is defined inductively as a structural operation semantics yielding a labelled transition relation. The full definition can be found in our earlier work [1] and in the appendix of this paper. We here only comment

on the one change necessary to accommodate the generalized pattern matching. The original input rule reads

$$\frac{\Psi \vdash M \leftrightarrow K}{\Psi \rhd \underline{M}(\lambda\widetilde{y})X.P \xrightarrow{\underline{K}\ X[\widetilde{y}:=\widetilde{L}]} P[\widetilde{y} := \widetilde{L}]}$$

and means that the instantiating substitution $[\widetilde{y} := \widetilde{L}]$ is applied both in the transition label and in the agent after the transition. Our new input rule is

$$\frac{\Psi \vdash M \leftrightarrow K \quad \widetilde{L} \in \text{MATCH}(N, \widetilde{y}, X)}{\Psi \rhd \underline{M}(\lambda\widetilde{y})X.P \xrightarrow{\underline{K}\ N} P[\widetilde{y} := \widetilde{L}]}$$

Here the matching with the transition label and the substitution applied to the following agent may be different. The MATCH predicate determines both the former (by designating the term N) and the latter (by designating the substitution), but there is no requirement on how they relate. As explained in Section 1.2 this means we can introduce evaluation of terms in the substitution or in the matching.

Theorem 1 (Preservation of well-formedness). *If P is well-formed, then $P\sigma$ is well-formed, and if $\Psi \rhd P \xrightarrow{\alpha} P'$ then P' is well-formed.*

Note that well-formedness implies conformance to the sorting discipline; therefore this theorem implies a kind of subject reduction.

The definition of strong and weak bisimulation and their algebraic properties are unchanged from our previous work [1] and can be found in the appendix. The results can be summerized as follows:

Theorem 2 (Properties of bisimulation). *All results on bisimulation established in [1] and [14] still hold in sorted psi-calculi with generalized matching.*

The results have been formally verified in Isabelle/Nominal by adapting our existing proof scripts. The main difference is in the input cases of inductive proofs. This represents no more than two days of work, with the bulk of the effort going towards proving a crucial technical lemma stating that transitions do not invent new names with the new pattern matching. Unfortunately, in Isabelle/Nominal there are currently no facilities to reason parametrically over the set of name sorts. Therefore the mechanically checked proofs only apply to psi-calculi with a trivial sorting (a single sort that is admitted everywhere); we complement them with manual proofs to extend these to arbitrary sortings.

3 Examples

Several well-known process algebras can be directly represented as a sorted psi-calculus by instantiating the parameters in the right way. With this we mean that the syntax is isomorphic and that the operational semantics is exactly

preserved in a strong operational correspondence modulo strong bisimulation. There is no need for elaborate coding schemes and the correspondence proofs are straightforward.

Theorem 3 (Process algebra representations). *CCS with value passing [15], the unsorted and the sorted polyadic pi-calculus [6, 16], and the polyadic synchronization pi-calculus [17] can all be directly represented as sorted psi-calculi.*

The list can certainly be made longer, though each process algebra currently has a separate definition and therefore requires a separate formal proof. For example, a version of LINDA [5] can easily be obtained as a variant of the polyadic pi-calculus. To illustrate the technique, the only difference between polyadic pi-calculus and polyadic synchronization pi-calculus is about admitting tuples of names in prefix subjects. Details can be found in the appendix.

More interestingly we demonstrate that we can accommodate a variety of structures for communication channels; in general these can be any kind of data, and substitution can include any kind of computation on these structures. This indicates that the word "substitution" may be a misnomer — a better word may be "effect" — though we keep it to conform with our earlier work. The examples below use default values for the parameters where where $\mathbf{A} = \{\mathbf{1}\}$, $\mathbf{C} = \{\top, \bot\}$ and $M \leftrightarrow N = \top$ iff $M = N$, otherwise $\bot$. We let $\mathbf{1} \vdash \top$ and $\mathbf{1} \nvdash \bot$. We also let $\underline{\alpha} = \overline{\alpha} = \mathcal{S} \times \mathcal{S}$, $\mathcal{S}_\nu = \mathcal{S}_\mathcal{N}$, and let $\prec$ be the identity on $\mathcal{S}$, unless otherwise defined. Finally we let $\text{MATCH}(M, \widetilde{x}, X) = \emptyset$ where not otherwise defined, we write $\preceq$ for the subterm (non-strict) partial order, and we use the standard notion of simultaneous substitution unless otherwise stated.

Example 2 (Convergent rewrite system on terms). We here consider deterministic computations specified using a rewrite system on terms containing names. This example highlights how a notion of substitution restricts the possible choices for $\text{VARS}(X)$; see Example 3 and Example 4 for two concrete instances.

Let Σ be a sorted signature, and $\cdot\Downarrow$ be normalization with respect to a convergent rewrite system on the nominal term algebra over $\mathcal{N}$ generated by Σ. We write ρ for sort-preserving capture-avoiding simultaneous substitutions $\{{}^{\widetilde{M}}/_{\widetilde{a}}\}$ where every M_i is in normal form. A pattern (term) X is stable if for all ρ, $X\rho\Downarrow = X\rho$. The patterns include the stable patterns Y and all instances X thereof (i.e., where $X = Y\rho$); such an X can bind any names occurring in Y but not in ρ.

REWRITE($\Downarrow$)
$\mathbf{T} = \mathbf{X} = \text{range}(\Downarrow)$
$M[\widetilde{y} := \widetilde{L}] = M\{{}^{\widetilde{L}}/_{\widetilde{y}}\}\Downarrow$
$\text{VARS}(X) = \bigcup\{\mathcal{P}(\text{n}(Y) \setminus \text{n}(\rho)) : Y \text{ stable} \wedge X = Y\rho\}$
$\text{MATCH}(M, \widetilde{x}, X) = \{\widetilde{L} : M = X\{{}^{\widetilde{L}}/_{\widetilde{x}}\}\}$

As a simple instance of Example 2, we may consider Peano arithmetic.

Example 3 (Peano arithmetic). Let $\mathcal{S} = \mathcal{S}_{\mathcal{N}} = \{\mathtt{nat}, \mathtt{chan}\}$. We take the signature consisting of the function symbols $\mathtt{zero} : \mathtt{nat}$, $\mathtt{succ} : \mathtt{nat} \to \mathtt{nat}$ and $\mathtt{plus} : \mathtt{nat} \times \mathtt{nat} \to \mathtt{nat}$. The rewrite rules $\mathtt{plus}(K, \mathtt{succ}(M)) \to \mathtt{plus}(\mathtt{succ}(K), M)$ and $\mathtt{plus}(K, \mathtt{zero}) \to K$ induce a convergent rewrite system $\Downarrow^{\mathrm{Peano}}$.

The stable terms are those that do not contain any occurrence of $\mathtt{plus}$. The construction of Example 2 yields that $\widetilde{x} \in \textsc{vars}(X)$ if $\widetilde{x} = \varepsilon$ (which matches only the term X itself), or if $\widetilde{x} = a$ and $X = \mathtt{succ}^n(a)$.

Writing i for $\mathtt{succ}^i(\mathtt{zero})$, the agent $(\nu a)(\overline{a}\,2 \mid \underline{a}(\lambda y)\mathtt{succ}(y) \,.\, \overline{c}\,\mathtt{plus}(3, y))$ of $\mathbf{REWRITE}(\Downarrow^{\mathrm{Peano}})$ has one visible transition, with the label $\overline{c}\,4$. In particular, the object of the label is $\mathtt{plus}(3, y)[y := 1] = \mathtt{plus}(3, y)\{{}^{1}\!/_{y}\}\Downarrow^{\mathrm{Peano}} = 4$.

Example 4 (Symmetric encryption). We can also consider variants on the construction in Example 2, such as a simple Dolev-Yao style [18] cryptographic message algebra for symmetric cryptography, where we ensure that the encryption keys of received encryptions can not be bound in input patterns, in agreement with cryptographic intuition.

Let $\mathcal{S} = \mathcal{S}_{\mathcal{N}} = \{\mathtt{message}, \mathtt{key}\}$, and consider the term algebra over the signature with the two function symbols $\mathtt{enc}, \mathtt{dec}$ of sort $\mathtt{message} \times \mathtt{key} \to \mathtt{message}$. The rewrite rule $\mathtt{dec}(\mathtt{enc}(M, K), K) \to M$ induces a convergent rewrite system $\Downarrow^{\mathrm{enc}}$, where the terms not containing $\mathtt{dec}$ are stable.

The construction of Example 2 yields that $\widetilde{x} \in \textsc{vars}(X)$ if $\widetilde{x} \subseteq \mathrm{n}(X)$ are pair-wise different and no x_i occurs as a subterm of a $\mathtt{dec}$ in X. This construction would permit to bind the keys of an encrypted message upon reception, e.g. $\underline{a}(\lambda m, k)\mathtt{enc}(m, k) \,.\, P$ would be allowed although it does not make cryptographic sense. Therefore we further restrict $\textsc{vars}(X)$ to those sets not containing names that occur in key position in X, thus disallowing the binding of k above.

SYMSPI
As $\mathbf{REWRITE}(\Downarrow^{\mathrm{enc}})$, except
$\textsc{vars}(X) = \mathcal{P}(n(X) \setminus \{a : a \preceq \mathtt{dec}(Y_1, Y_2) \preceq X \vee (a \preceq Y_2 \wedge \mathtt{enc}(Y_1, Y_2) \preceq X)\})$

As an example, the agent
$(\nu a, k)(\overline{a}\ \mathtt{enc}(\mathtt{enc}(M, l), k) \mid \underline{a}(\lambda y)\mathtt{enc}(y, k) \,.\, \overline{c}\ \mathtt{dec}(y, l))$ has a visible transition with label $\overline{c}\ M$.

Example 5 (Pattern-matching spi-calculus). A more advanced version of Example 4 is the treatment of data in the pattern-matching spi-calculus [4], to which we refer for more examples and motivations of the definitions below. Features of the calculus includes a non-homomorphic definition of substitution that does not preserve sorts, and a sophisticated way of computing permitted pattern variables. This example highlights the flexibility of sorted psi-calculi in that such a specialized modelling language can be directly represented, in a form that is very close to the original.

We start from the term algebra T_Σ over the unsorted signature Σ consisting of the function symbols $()$, $(\cdot, \cdot)$, $\mathtt{eKey}(\cdot)$, $\mathtt{dKey}(\cdot)$, $\mathtt{enc}(\cdot, \cdot)$ and $\mathtt{enc}^{-1}(\cdot, \cdot)$. The

operation $\mathtt{enc}^{-1}$ is "encryption with the inverse key", which is only permitted to occur in patterns. We add a sort system on T_Σ where $\mathtt{impl}$ denotes implementable terms not containing $\mathtt{enc}^{-1}$, and $\mathtt{pat}$ those that may only be used in patterns. The sort $\bot$ denotes ill-formed terms, which do not occur in well-formed processes. Substitution is defined homomorphically on the term algebra, except for $\mathtt{enc}^{-1}(M_1, M_2)\sigma$ which is $\mathtt{enc}(M_1\sigma, \mathtt{eKey}(N))$ when $M_2\sigma = \mathtt{dKey}(N)$, and $\mathtt{enc}^{-1}(M_1\sigma, M_2\sigma)$ otherwise. We let $\Vdash \subset \mathcal{P}(T_\Sigma) \times \mathcal{P}(T_\Sigma)$ be deducibility in the Dolev-Yao message algebra (for the precise definition, see [4]). The definition of $\textsc{vars}(X)$ below allows to bind only those names that can be deduced from X and the other names occurring in X. This excludes binding an unknown key, like in Example 4.

PMSPI

$\mathbf{T} = \mathbf{X} = T_\Sigma$
$\mathcal{S}_\mathcal{N} = \{\mathtt{impl}\}$ $\quad$ $\mathcal{S} = \{\mathtt{impl}, \mathtt{pat}, \bot\}$
$\prec = \overline{\propto} = \{(\mathtt{impl}, \mathtt{impl})\}$
$\propto = \{(\mathtt{impl}, \mathtt{impl}), (\mathtt{impl}, \mathtt{pat})\}$
$\textsc{sort}(M) = \mathtt{impl}$ if $\forall N_1, N_2.\ \mathtt{enc}^{-1}(N_1, N_2) \not\preceq M$
$\textsc{sort}(M) = \bot$ if $\exists N_1, N_2.\ \mathtt{enc}^{-1}(N_1, \mathtt{dKey}(N_2)) \preceq M$
$\textsc{sort}(M) = \mathtt{pat}$ otherwise
$\textsc{match}(M, \widetilde{x}, X) = \{\widetilde{L} \ :\ M = X[\widetilde{x} := \widetilde{L}]\}$
$\textsc{vars}(X) = \{S \subseteq \mathrm{n}(X) \ :\ ((\mathrm{n}(X) \setminus S) \cup \{X\}) \Vdash S\}$

As an example, consider the following transitions in **PMSPI**:

$$\begin{aligned}(\nu a, k, l)(\overline{a}\ \mathtt{enc}(\mathtt{dKey}(l), \mathtt{eKey}(k)).\overline{a}\ \mathtt{enc}(M, \mathtt{eKey}(l)) \\ \mid \underline{a}(\lambda y)\mathtt{enc}(y, \mathtt{eKey}(k))\,.\,\underline{a}(\lambda z)\mathtt{enc}^{-1}(z, y)\,.\,\overline{c}\ z) \\ \xrightarrow{\tau} (\nu a, k, l)(\overline{a}\ \mathtt{enc}(M, \mathtt{eKey}(l)) \mid \underline{a}(\lambda z)\mathtt{enc}(z, \mathtt{eKey}(l))\,.\,\overline{c}\ z) \\ \xrightarrow{\tau} (\nu a, k, l)\overline{c}\ M.\end{aligned}$$

Note that $\sigma = [y := \mathtt{dKey}(l)]$ resulting from the first input changed the sort of the second input pattern: $\textsc{sort}(\mathtt{enc}^{-1}(z, y)) = \mathtt{pat}$, but $\textsc{sort}(\mathtt{enc}^{-1}(z, y)\sigma) = \textsc{sort}(\mathtt{enc}(z, \mathtt{eKey}(l))) = \mathtt{impl}$. However, this is permitted by Definition 4, since $\mathtt{impl} \leq \mathtt{pat}$.

Example 6 (Nondeterministic computation). The previous examples considered total deterministic notions of computation on the term language. Here we consider a data term language equipped with partial non-deterministic evaluation: a lambda calculus with the ambiguous choice operator $\mathtt{amb}$ [19]. Due to the nondeterminism and partiality, evaluation cannot be part of the substitution function. Instead, the MATCH function collects all evaluations of the received term, which are non-deterministically selected from by the IN rule. This example also highlights the use of object languages with binders, a common application of nominal logic.

We let substitution on terms be the usual capture-avoiding syntactic replacement, and define reduction contexts $\mathcal{R} ::= [] \mid \mathcal{R}\ M \mid (\boldsymbol{\lambda} x.M)\ \mathcal{R}$. Reduction $\rightarrow$

is the smallest pre-congruence for reduction contexts that contain the rules for β-reduction ($\boldsymbol{\lambda}x.M\ N \to M[x := N]$) and $\mathtt{amb}$ (namely $\mathtt{amb}\ M_1\ M_2 \to M_i$ if $i \in \{1, 2\}$). We use the single-name patterns of Example 1, but include evaluation in matching.

AMBLAM
$\mathcal{S}_{\mathcal{N}} = \mathcal{S} = \{s\}$ $\qquad$ $\mathbf{X} = \mathcal{N}$
$M ::= a \mid M\ M \mid \boldsymbol{\lambda}x.M \mid \mathtt{amb}\ M\ M$
where x binds into M in $\boldsymbol{\lambda}x.M$
$\textsc{match}(M, x, x) = \{N\ :\ M \to^* N \not\to\}$

As an example, the agent $(\nu a)(\underline{a}(y)\,.\,\overline{c}\ y\,.\,\mathbf{0} \mid \overline{a}\ (\mathtt{amb}\ (\boldsymbol{\lambda}x.x\ x)\ (\boldsymbol{\lambda}x.x))\,.\,\mathbf{0})$ has two visible transitions, with labels $\overline{c}\ \boldsymbol{\lambda}x.x\ x$ and $\overline{c}\ \boldsymbol{\lambda}x.x$.

4 Conclusions and further work

We have described two features that taken together significantly improves the precision of applied process calculi: generalised pattern matching and substitution, which allow us to model computations on an arbitrary data term language, and a sort system which allows us to remove spurious data terms from consideration and to ensure that channels carry data of the appropriate sort. The well-formedness of processes is thereby guaranteed to be preserved by transitions. We have given examples of these features, ranging from the simple polyadic pi-calculus to the highly specialized pattern-matching spi-calculus, in the psi-calculi framework.

The meta-theoretic results carry over from the original psi formulations, and many have been machine-checked in Isabelle. We have also developed a tool for sorted psi-calculi [20], the Psi-calculi Workbench (Pwb), which provides an interactive simulator and automatic bisimulation checker. Users of the tool need only implement the parameters of their psi-calculus instances, supported by a core library.

Future work includes developing a symbolic semantics with pattern matching. For this, a reformulation of the operational semantics in the late style, where input objects are not instantiated until communication takes place, is necessary. We also aim to extend the use of sorts and generalized pattern matching to other variants of psi-calculi, including higher-order psi calculi [21] and reliable broadcast psi-calculi [22]. As mentioned in Section 2.6, further developments in Nominal Isabelle are needed for mechanizing theories with arbitrary but fixed sortings.

References

[1] Bengtson, J., Johansson, M., Parrow, J., Victor, B.: Psi-calculi: a framework for mobile processes with nominal data and logic. LMCS **7**(1:11) (2011)

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

[3] Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: Proceedings of POPL '01, ACM (January 2001) 104–115
[4] Haack, C., Jeffrey, A.: Pattern-matching spi-calculus. Information and Computation **204**(8) (2006) 1195–1263
[5] Gelernter, D.: Generative communication in Linda. ACM TOPLAS **7**(1) (January 1985) 80–112
[6] Milner, R.: The polyadic π-calculus: A tutorial. In Bauer, F.L., Brauer, W., Schwichtenberg, H., eds.: Logic and Algebra of Specification. Volume 94 of Series F., NATO ASI, Springer (1993)
[7] Hüttel, H.: Typed psi-calculi. In Katoen, J.P., König, B., eds.: CONCUR 2011 – Concurrency Theory. Volume 6901 of LNCS., Springer (2011) 265–279
[8] Blanchet, B.: Using Horn clauses for analyzing security protocols. In Cortier, V., Kremer, S., eds.: Formal Models and Techniques for Analyzing Security Protocols. Volume 5 of Cryptology and Information Security Series. IOS Press (March 2011) 86–111
[9] Fournet, C., Gordon, A.D., Maffeis, S.: A type discipline for authorization policies. In Sagiv, M., ed.: Proc. of ESOP 2005. Volume 3444 of LNCS., Springer (2005) 141–156
[10] Given-Wilson, T., Gorla, D., Jay, B.: Concurrent pattern calculus. In Calude, C., Sassone, V., eds.: Theoretical Computer Science. Volume 323 of IFIP Advances in Information and Communication Technology. Springer (2010) 244–258
[11] Fournet, C., Gonthier, G.: The reflexive CHAM and the join-calculus. In: Proc. POPL. (1996) 372–385
[12] Gabbay, M.J., Pitts, A.M.: A new approach to abstract syntax with variable binding. Formal Aspects of Computing **13** (2001) 341–363
[13] Johansson, M., Victor, B., Parrow, J.: Computing strong and weak bisimulations for psi-calculi. Journal of Logic and Algebraic Programming **81**(3) (2012) 162–180
[14] Johansson, M., Bengtson, J., Parrow, J., Victor, B.: Weak equivalences in psi-calculi. In: Proc. of LICS 2010, IEEE (2010) 322–331
[15] Milner, R.: Communication and Concurrency. Prentice-Hall, Inc. (1989)
[16] Sangiorgi, D.: Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms. PhD thesis, University of Edinburgh (1993) CST-99-93 (also published as ECS-LFCS-93-266).
[17] Carbone, M., Maffeis, S.: On the expressive power of polyadic synchronisation in π-calculus. Nordic Journal of Computing **10**(2) (2003) 70–98
[18] Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE Transactions on Information Theory **29**(2) (1983) 198–208
[19] McCarthy, J.: A basis for a mathematical theory of computation. Computer Programming and Formal Systems (1963) 33–70
[20] Borgström, J., Gutkovas, R., Rodhe, I., Victor, B.: A parametric tool for applied process calculi. Accepted for publication in ACSD 2013. Available from `http://www.it.uu.se/research/group/mobility/applied/psiworkbench`.
[21] Parrow, J., Borgström, J., Raabjerg, P., Åman Pohjola, J.: Higher-order psi-calculi (2012) Accepted for publication in MSCS. Available from `http://www.it.uu.se/research/group/mobility`.
[22] Pohjola, J.Å., Borgström, J., Parrow, J., Raabjerg, P., Rodhe, I.: Negative premises in applied process calculi. submitted (2013)
[23] Åman Pohjola, J.: Isabelle proof scripts for sorted psi-calculi (2012)

A Complete Definitions

A.1 Frames and transitions

Each agent affects other agents that are in parallel with it via its frame, which may be thought of as the collection of all top-level assertions of the agent. A *frame* F is an assertion with local names, written $(\nu\widetilde{b})\Psi$ where $\widetilde{b}$ is a sequence of names that bind into the assertion Ψ. We use F, G to range over frames, and identify alpha-equivalent frames. We overload $\otimes$ to frame composition defined by $(\nu\widetilde{b_1})\Psi_1 \otimes (\nu\widetilde{b_2})\Psi_2 = (\nu\widetilde{b_1}\widetilde{b_2})(\Psi_1 \otimes \Psi_2)$ where $\widetilde{b_1}\#\widetilde{b_2}, \Psi_2$ and vice versa. We write $\Psi \otimes F$ to mean $(\nu\epsilon)\Psi \otimes F$, and $(\nu c)((\nu\widetilde{b})\Psi)$ for $(\nu c\widetilde{b})\Psi$.

Intuitively a condition is entailed by a frame if it is entailed by the assertion and does not contain any names bound by the frame, and two frames are equivalent if they entail the same conditions. Formally, we define $F \vdash \varphi$ to mean that there exists an alpha variant $(\nu\widetilde{b})\Psi$ of F such that $\widetilde{b}\#\varphi$ and $\Psi \vdash \varphi$. We also define $F \simeq G$ to mean that for all φ it holds that $F \vdash \varphi$ iff $G \vdash \varphi$.

Definition 8 (Frames and Transitions). *The* frame $\mathcal{F}(P)$ of an agent P *is defined inductively as follows:*

$$\begin{array}{ll}
\mathcal{F}(\underline{M}(\lambda\widetilde{x})N \,.\, P) = \mathcal{F}(\overline{M}\ N \,.\, P) = \mathbf{1} & \\
\mathcal{F}(\mathbf{case}\ \widetilde{\varphi} : \widetilde{P}) = \mathcal{F}(!P) = \mathbf{1} & \\
\mathcal{F}((\!|\Psi|\!)) = (\nu\epsilon)\Psi & \mathcal{F}(P \mid Q) = \mathcal{F}(P) \otimes \mathcal{F}(Q) \\
\mathcal{F}((\nu b)P) = (\nu b)\mathcal{F}(P) &
\end{array}$$

The actions *ranged over by α, β are of the following three kinds: Output $\overline{M}\ (\nu\widetilde{a})\ N$ where $\tilde{a} \subseteq \mathrm{n}(N)$, Input $\underline{M}\ N$, and Silent τ. Here we refer to M as the* subject *and N as the* object*. We define $\mathrm{bn}(\overline{M}\ (\nu\tilde{a})\ N) = \tilde{a}$, and $\mathrm{bn}(\alpha) = \emptyset$ if α is an input or τ. We also define $\mathrm{n}(\tau) = \emptyset$ and $\mathrm{n}(\alpha) = \mathrm{n}(M) \cup \mathrm{n}(N)$ for the input and output actions. We write $\overline{M}\langle N\rangle$ for $\overline{M}\ (\nu\varepsilon)\ N$.*

A transition *is written $\Psi \rhd P \xrightarrow{\alpha} P'$, meaning that in the environment Ψ the well-formed agent P can do an α to become P'. The transitions are defined inductively in Table 1. We write $P \xrightarrow{\alpha} P'$ without an assertion to mean $\mathbf{1} \rhd P \xrightarrow{\alpha} P'$.*

The operational semantics is the same as for the original psi-calculi, except for the use of MATCH in rule IN. We identify alpha-equivalent agents and transitions (see [1] for details). In a transition the names in $\mathrm{bn}(\alpha)$ bind into both the action object and the derivative, therefore $\mathrm{bn}(\alpha)$ is in the support of α but not in the support of the transition. This means that the bound names can be chosen fresh, substituting each occurrence in both the action and the derivative.

As shown in the introduction, well-formedness is not preserved by transitions in the original psi-calculi. However, in sorted psi-calculi the usual well-formedness preservation result holds.

$$\textsc{In}\ \frac{\Psi \vdash M \leftrightarrow K \quad \widetilde{L} \in \textsc{match}(N, \widetilde{y}, X)}{\Psi \rhd \underline{M}(\lambda \widetilde{y})X.P \xrightarrow{\underline{K}\ N} P[\widetilde{y} := \widetilde{L}]} \qquad \textsc{Out}\ \frac{\Psi \vdash M \leftrightarrow K}{\Psi \rhd \overline{M}\ N.P \xrightarrow{\overline{K}\langle N\rangle} P}$$

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

$$\textsc{Par}\ \frac{\Psi_Q \otimes \Psi \rhd P \xrightarrow{\alpha} P'}{\Psi \rhd P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \mathrm{bn}(\alpha) \# Q \qquad \textsc{Case}\ \frac{\Psi \rhd P_i \xrightarrow{\alpha} P' \qquad \Psi \vdash \varphi_i}{\Psi \rhd \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \xrightarrow{\alpha} P'}$$

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

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

Symmetric versions of Com and Par are elided. In the rule Com we assume that $\mathcal{F}(P) = (\nu \widetilde{b_P})\Psi_P$ and $\mathcal{F}(Q) = (\nu \widetilde{b_Q})\Psi_Q$ 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 $\mathcal{F}(Q) = (\nu \widetilde{b_Q})\Psi_Q$ where $\widetilde{b_Q}$ is fresh for Ψ, P and α. In Open the expression $\nu \tilde{a} \cup \{b\}$ means the sequence $\tilde{a}$ with b inserted anywhere.

Table 1. Operational semantics.

B Meta-theory

We begin by recollecting the definition of strong labelled bisimulation on well-formed agents from Bengtson et al. [1], to which we refer for examples, intuitions and the exact formulations of theorems.

Definition 9 (Strong bisimulation). *A* strong bisimulation $\mathcal{R}$ *is a ternary relation on assertions and pairs of agents such that* $\mathcal{R}(\Psi, P, Q)$ *implies the following four statements.*

1. *Static equivalence:* $\Psi \otimes \mathcal{F}(P) \simeq \Psi \otimes \mathcal{F}(Q)$.
2. *Symmetry:* $\mathcal{R}(\Psi, Q, P)$.
3. *Extension with arbitrary assertion:* $\forall \Psi'.\ \mathcal{R}(\Psi \otimes \Psi', P, Q)$.
4. *Simulation: for all* α, P' *such that* $\mathrm{bn}(\alpha) \# \Psi, Q$
   and $\Psi \rhd P \xrightarrow{\alpha} P'$*, there exists* Q'
   such that $\Psi \rhd Q \xrightarrow{\alpha} Q'$ *and* $\mathcal{R}(\Psi, P', Q')$.

We define bisimilarity $P \dot{\sim}_\Psi Q$ *to mean that there is a bisimulation* $\mathcal{R}$ *such that* $\mathcal{R}(\Psi, P, Q)$*, and write* $\dot{\sim}$ *for* $\dot{\sim}_{\mathbf{1}}$*. Strong* congruence *is defined by* $P \sim_\Psi Q$ *iff for*

all sequences $\widetilde{\sigma}$ of substitutions it holds that $P\widetilde{\sigma} \mathrel{\dot{\sim}}_\Psi Q\widetilde{\sigma}$. We write $P \sim Q$ for $P \sim_{\mathbf{1}} Q$.

There is also a notion of weak bisimilarity ($\dot{\approx}$) where τ-transitions cannot be observed; see [14] for its precise definition.

We seek to establish the following properties of bisimulation.

Theorem 4 (Congruence properties of $\dot{\sim}$). *For all Ψ:*

$$\begin{array}{rcl}
P \mathrel{\dot{\sim}}_\Psi Q & \Longrightarrow & P \mid R \mathrel{\dot{\sim}}_\Psi Q \mid R \\
a\#\Psi \wedge P \mathrel{\dot{\sim}}_\Psi Q & \Longrightarrow & (\nu a)P \mathrel{\dot{\sim}}_\Psi (\nu a)Q \\
P \mathrel{\dot{\sim}}_\Psi Q & \Longrightarrow & !P \mathrel{\dot{\sim}}_\Psi !Q \\
\forall i. P_i \mathrel{\dot{\sim}}_\Psi Q_i & \Longrightarrow & \mathbf{case}\ [\!]\ \widetilde{\varphi} : \widetilde{P} \mathrel{\dot{\sim}}_\Psi \mathbf{case}\ [\!]\ \widetilde{\varphi} : \widetilde{Q} \\
P \mathrel{\dot{\sim}}_\Psi Q & \Longrightarrow & \overline{M}\ N\,.\,P \mathrel{\dot{\sim}}_\Psi \overline{M}\ N\,.\,Q \\
(\forall \widetilde{L}.\ P[\widetilde{x} := \widetilde{L}] \mathrel{\dot{\sim}}_\Psi Q[\widetilde{x} := \widetilde{L}]) & \Longrightarrow & \\
 & & \underline{M}(\lambda\widetilde{x})X\,.\,P \mathrel{\dot{\sim}}_\Psi \underline{M}(\lambda\widetilde{x})X\,.\,Q
\end{array}$$

Definition 10. *$P \sim_\Psi Q$ means that for all sequences σ of substitutions it holds that $P\sigma \mathrel{\dot{\sim}}_\Psi Q\sigma$, and we write $P \sim Q$ for $P \sim_{\mathbf{1}} Q$.*

We seek to establish the following properties of bisimulation congruence.

Theorem 5. *Strong congruence $\sim_\Psi$ is a congruence for all Ψ.*

Theorem 6 (Structural equivalence). *$a\#Q, \widetilde{x}, M, N, X, \widetilde{\varphi}$ implies*

$$\begin{array}{rcl}
\mathbf{case}\ [\!]\ \widetilde{\varphi} : \widetilde{(\nu a)P} & \sim & (\nu a)\mathbf{case}\ [\!]\ \widetilde{\varphi} : \widetilde{P} \\
\underline{M}(\lambda\widetilde{x})X\,.\,(\nu a)P & \sim & (\nu a)\underline{M}(\lambda\widetilde{x})X\,.\,P \\
\overline{M}\ N\,.\,(\nu a)P & \sim & (\nu a)\overline{M}\ N\,.\,P \\
Q \mid (\nu a)P & \sim & (\nu a)(Q \mid P) \\
(\nu b)(\nu a)P & \sim & (\nu a)(\nu b)P \\
(\nu a)\mathbf{0} & \sim & \mathbf{0} \\
!P & \sim & P \mid !P \\
P \mid (Q \mid R) & \sim & (P \mid Q) \mid R \\
P \mid Q & \sim & Q \mid P \\
P & \sim & P \mid \mathbf{0}
\end{array}$$

B.1 Trivially sorted calculi

A *trivially* sorted psi calculus is one where $\prec = \underline{\propto} = \overline{\propto} = \mathcal{S} \times \mathcal{S}$ and $\mathcal{S}_\nu = \mathcal{S}$, i.e., the sorts do not affect how terms are used in communications and substitutions. For technical reasons we here first establish the expected algebraic properties of bisimilarity and its induced congruence in trivially sorted psi-calculi, and then investigate how these results are lifted to arbitrary sorted calculi.

Theorem 7. *Theorem 4, Theorem 5, and Theorem 6 hold for trivially sorted psi-calculi.*

Additionally, the corresponding results on the algebraic properties of weak bisimilarity and its induced congruence as defined and presented in [14] also hold for trivially sorted psi-calculi.

These results have all been machine-checked in Isabelle [23]. As indicated these proof scripts apply only to trivially sorted calculi, meaning that the only extension to our previous formulation is in the input rule which now uses MATCH. We have also machine-checked Theorem 1 (preservation of well-formedness) in this setting.

The restriction to trivially sorted calculi is a consequence of technicalities in Nominal Isabelle: it requires every name sort to be declared individually, and there are no facilities to reason parametrically over the set of name sorts. There is also a discrepancy in that our definitions in Section 2 considers only well-sorted alpha-renamings, while the mechanisation works with a single sort of names and thus allows for ill-sorted alpha-renamings. This is only a technicality, since every use of alpha-renaming in the formal proofs is to ensure that the bound names in patterns and substitutions avoid other bound names—thus, whenever we may work with an ill-sorted renaming, there would be a well-sorted renaming that suffices for the task.

B.2 Arbitrary sorted psi-calculi

We here extend the results of Theorem 7 to arbitrary sorted psi-calculi. The idea is to introduce an explicit error element $\bot$ corresponding to an ill-sorted term, pattern, condition and assertion. For technical reasons we must also include one extra condition `fail` (in order to ensure the compositionality of $\otimes$) and in the patterns we need different error elements with different support (in order to ensure the preservation of pattern variables under substitution).

Let $I = (\mathbf{T}, \mathbf{X}, \mathbf{C}, \mathbf{A})$ be a sorted psi-calculus. We construct a trivially sorted psi-calculus $U(I)$ with one extra sort, `error`. The parameters of $U(I)$ are defined by $U(I) = (\mathbf{T} \cup \{\bot\}, \mathbf{X} \cup \{(\bot, S) \ : \ S \subset_{\text{fin}} \mathcal{N}\}, \mathbf{C} \cup \{\bot, \texttt{fail}\}, \mathbf{A} \cup \{\bot\})$. Here $\bot$ and `fail` are constant symbols with empty support of sort `error`. $\bot$ is not a channel, never entailed, matches nothing and entails nothing but `fail`. `fail` is entailed only by $\bot$. We define $\Psi \otimes \bot = \bot \otimes \Psi = \bot$ for all Ψ, and otherwise $\otimes$ is as in I. MATCH and $\leftrightarrow$ in $U(I)$ are the same as in I, and for $\Psi \neq \bot$ we let $\Psi \vdash \varphi$ in $U(I)$ iff $\Psi \vdash \varphi$ in I. Substitution is then defined in $U(I)$ as follows:

$$
T[\widetilde{a} := \widetilde{N}]_{U(I)} = \begin{cases} T[\widetilde{a} := \widetilde{N}]_I & \text{if } \text{SORT}(a_i) \prec_I \text{SORT}(N_i) \text{ and} \\ & \quad N_i \neq \bot \text{ for all } i, \text{ and } T \neq (\bot, S) \\ (\bot, S \setminus \widetilde{a}) & \text{if } T = (\bot, S) \text{ is a pattern} \\ (\bot, \bigcup\bigcup \text{VARS}(T)) & \text{otherwise, if } T \text{ is a pattern} \\ \bot & \text{otherwise} \end{cases}
$$

Lemma 1. *Assume that P and Q are well-formed processes in I, and $\Psi \neq \bot$.*

1. *$U(I)$ as defined above is a sorted psi-calculus.*
2. *P and Q are well-formed when considered as processes of $U(I)$.*
3. *$\Psi \rhd P \xrightarrow{\alpha} P'$ in $U(I)$ iff $\Psi \rhd P \xrightarrow{\alpha} P'$ in I.*
4. *$P \dot{\sim}_\Psi Q$ in I iff $P \dot{\sim}_\Psi Q$ in $U(I)$, and $P \dot{\approx}_\Psi Q$ in I iff $P \dot{\approx}_\Psi Q$ in $U(I)$.*

With Lemma 1, we can lift the congruence and the structural equivalence results for trivially sorted psi-calculi to arbitrary sorted calculi:

Theorem 8. *All clauses of Theorem 7 pertaining to strong bisimilarity, strong congruence and weak bisimilarity are valid in all sorted psi-calculi.*

Proof. We show only the proofs for strong congruence and commutativity of the parallel operator. The other proofs are analogous.

Fix a sorted psi-calculus I. Assume $P \mathrel{\dot{\sim}_\Psi} Q$ holds in I. By Lemma 1.4, $P \mathrel{\dot{\sim}_\Psi} Q$ holds in $U(I)$. Theorem 7 thus yields $P \mid R \mathrel{\dot{\sim}_\Psi} Q \mid R$ in $U(I)$, and Lemma 1.4 yields the same in I.

Let P and Q be well-formed in I and $\Psi \neq \bot$. By Theorem 7, $P \mid Q \sim_\Psi Q \mid P$ holds in $U(I)$. By Definition 9, $(P \mid Q)\widetilde{\sigma} \mathrel{\dot{\sim}_\Psi} (Q \mid P)\widetilde{\sigma}$ in $U(I)$ for all $\widetilde{\sigma}$. By Theorem 1, when $\widetilde{\sigma}$ is well-sorted then $(P \mid Q)\widetilde{\sigma}$ and $(Q \mid P)\widetilde{\sigma}$ are well-formed. By Lemma 1.4, $(P \mid Q)\widetilde{\sigma} \mathrel{\dot{\sim}_\Psi} (Q \mid P)\widetilde{\sigma}$ in I. $P \mid Q \sim_\Psi Q \mid P$ follows by definition.

This approach does not yield a similar result for strong congruence, since the closure of bisimilarity under well-sorted substitutions does not imply its closure under ill-sorted substitutions. Consider a well-sorted instance I such that $\mathbf{0} \sim_\Psi (\!|\mathbf{1}|\!)$. The corresponding property of $U(I)$ does not follow: if σ is ill-sorted then $\mathbf{1}\sigma = \bot$, but $\mathbf{0} \mathrel{\dot{\sim}_\Psi} (\!|\bot|\!)$ does not hold since only $\bot$ entails `fail`. Instead, we have performed a direct proof: it is identical, line by line, to the proof in the trivially sorted case.

A direct manual proof would also be necessary to obtain similar results for weak congruence for the same reasons as in the strong case. We have chosen not to pursue this result since the weak case is less straight-forward and a manual proof would be error-prone.

B.3 Pattern matching in original calculi

In many cases we can recover the pattern matching of the original psi-calculi.

Theorem 9. *Suppose* $(\mathbf{T}, \mathbf{C}, \mathbf{A})$ *is an original psi calculus [1] where pattern variables are preserved by substitutions* $(\mathrm{n}(N\sigma) \supseteq \mathrm{n}(N) \setminus \mathrm{n}(\sigma))$. *Let* $\mathbf{X} = \mathbf{T}$ *and* $\textsc{vars}(X) = \mathcal{P}(\mathrm{n}(X))$ *and* $\textsc{match}(M, \widetilde{x}, X) = \{\widetilde{L} : M = X[\widetilde{x} := \widetilde{L}]\}$ *and* $\mathcal{S} = \mathcal{S}_\mathcal{N} = \mathcal{S}_\nu = \{s\}$ *and* $\underline{\propto} = \overline{\propto} = \prec = \{(s, s)\}$ *and* $\textsc{sort} : \mathcal{N} \uplus \mathbf{T} \uplus \mathbf{X} \to \{s\}$*; then* $(\mathbf{T}, \mathbf{X}, \mathbf{C}, \mathbf{A})$ *is a sorted psi calculus.*

This result has been machined-checked in Isabelle for trivally-sorted calculi.

C Process Calculi Examples

We here consider some variants of popular process calculi. One main point of our work is that we can represent them directly as psi-calculi, without elaborate coding schemes. In our original psi-calculi we could in this way directly represent the monadic pi-calculus; for the other calculi presented below an unsorted psi-calculus would contain terms with no counterpart in the represented calculus,

as explained in sections 1.2 and 1.3. We establish that our formulations enjoy a strong operational correspondence with the original calculus, under trivial mappings that merely specialise the original concrete syntax (e.g., the pi-calculus prefix $a(x)$ maps to $\underline{a}(\lambda x)x$ in psi). This correspondence is significantly stronger than standard correspondence results (cf. Gorla, I&C 208(9):1031-1053, 2010). Because of the simplicity of the mapping and the strength of the correspondence we use the phrasing that psi-calculi *represent* other process calculi, in contrast to *encoding* them.

C.1 Unsorted Polyadic pi-calculus

In the polyadic pi-calculus [6] the only values that can be transmitted between agents are tuples of names. Tuples cannot be nested. An input binds a tuple of distinct names and can only communicate with an output of equal length, resulting in a simultaneous substitution of all names. In the unsorted polyadic pi-calculus there are no further requirements on agents, in particular $a(x) \mid \overline{a}\langle y, z\rangle$ is a valid agent. This agent has no communication action since the lengths of the tuples mismatch.

PPI

$\mathbf{T} = \mathcal{N} \cup \{\langle\widetilde{a}\rangle : \widetilde{a} \subset_{\text{fin}} \mathcal{N}\}$
$\mathbf{C} = \{\top\} \cup \{a = b \mid a, b \in \mathcal{N}\}$
$\mathbf{X} = \{\langle\widetilde{a}\rangle : \widetilde{a} \subset_{\text{fin}} \mathcal{N} \wedge \widetilde{a} \text{ distinct}\}$
$\leftrightarrow$ = identity on names $\quad \mathbf{1} \vdash a = a$
$\text{VARS}(\langle\widetilde{a}\rangle) = \{\widetilde{a}\}$
$\text{MATCH}(\langle\widetilde{a}\rangle, \widetilde{x}, \langle\widetilde{x}\rangle) = \{\widetilde{a}\}$ if $|\widetilde{a}| = |\widetilde{x}|$
$\mathcal{S}_{\mathcal{N}} = \{\texttt{chan}\}$ $\quad \mathcal{S} = \{\texttt{chan}, \texttt{tup}\}$
$\text{SORT}(a) = \texttt{chan}$ $\quad \text{SORT}(\langle\widetilde{a}\rangle) = \texttt{tup}$
$\mathcal{S}_{\nu} = \{\texttt{chan}\}$ $\quad \prec = \{(\texttt{chan}, \texttt{chan})\}$
$\overline{\propto} = \underline{\propto} = \{(\texttt{chan}, \texttt{tup})\}$

As an example the agent $\underline{a}(\lambda x, y)\langle x, y\rangle \,.\, \overline{a}\ \langle y\rangle \,.\, \mathbf{0}$ is well-formed, since $\texttt{chan} \underline{\propto} \texttt{tup}$ and $\texttt{chan}\ \overline{\propto}\ \texttt{tup}$, with $\text{VARS}(\langle x, y\rangle) = \{\{x, y\}\}$. This demonstrates that **PPI** disallows anomalies such as nested tuples but does not enforce a sorting discipline to guarantee that names communicate tuples of the same length.

PPI is a direct representation of the polyadic pi-calculus as presented by Sangiorgi [16] (with replication instead of process constants). Let $[\![\cdot]\!]$ be the function that maps the polyadic pi-calculus to **PPI** processes, and analogously for labels: it maps summation to $\top$-guarded **case** statements, matching $[x = y]$ to $x = y$-guarded **case** statements, input prefix $a(\widetilde{x})$ to $\underline{a}(\lambda\widetilde{x})\widetilde{x}$, and is homomorphic over all other operators.

We obtain a strong operational correspondence between the calculi:

Theorem 10. *If P and Q are polyadic pi-calculus processes, then:*

1. *If $P \xrightarrow{\alpha} P'$ then $[\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$*

2. If $[\![P]\!] \xrightarrow{\alpha'} P''$ *then* $P \xrightarrow{\alpha} P'$ *where* $[\![\alpha]\!] = \alpha'$ *and* $[\![P']\!] = P''$

As it turns out, the representation of polyadic pi-calculus is surjective modulo strong bisimilarity. We show this by defining a translation $\overline{P}$ in the other direction: it is defined homomorphically except for **case** statements, which are mapped to a sum of possibly match-guarded processes, and $\overline{(\!|\mathbf{1}|\!)} = \mathbf{0}$.

Theorem 11. *If* P *is a* **PPI** *process, then* $P \mathrel{\dot\sim} [\![\overline{P}]\!]$.

A version of LINDA [5] can be obtained by adding a term $\bullet$ of a new sort `ts` denoting the tuple space, letting $\overline{\propto} = \underline{\propto} = \{(\mathtt{ts}, \mathtt{tup})\}$ and defining MATCH and VARS as in Theorem 9.

C.2 Sorted polyadic pi-calculus

Milner's classic sorting [6] regime for the polyadic pi-calculus ensures that pattern matching in inputs always succeeds, by enforcing that the length of the pattern is the same as the length of the received tuple. This is achieved as follows. Milner assumes a countable set of subject sorts S ascribed to names, and a partial function $\mathtt{ob} : \mathrm{S} \rightharpoonup \mathrm{S}^*$, assigning a sequence of object sorts to each sort. The intuition is that if a has sort s then any communication along a must be a tuple of sort $\mathtt{ob}(s)$. An agent is *well-sorted* if for any input prefix $a(b_1, \ldots b_n)$ it holds that a has some sort s where $\mathtt{ob}(s)$ is the sequence of sorts of $b_1, \ldots, b_n$ and similarly for output prefixes.

SORTEDPPI	
Everything as in **PPI** except:	
$\mathcal{S}_{\mathcal{N}} = \mathcal{S}_{\nu} = \mathrm{S}$	$\mathcal{S} = \mathrm{S}^*$
$\textsc{sort}(a_1, \ldots, a_n) = \textsc{sort}(a_1), \ldots, \textsc{sort}(a_n)$	
$\textsc{match}(\widetilde{a}, \widetilde{x}, \widetilde{x}) = \{\widetilde{a}\}$	if $\textsc{sort}(\widetilde{a}) = \textsc{sort}(\widetilde{x})$
$\prec = \{(s, s) : s \in \mathrm{S}\}$	$\overline{\propto} = \underline{\propto} = \{(s, \mathtt{ob}(s)) : s \in \mathrm{S}\}$

As an example, let $\textsc{sort}(a) = s$ with $\mathtt{ob}(s) = t_1, t_2$ and $\textsc{sort}(x) = t_1$ with $\mathtt{ob}(t_1) = t_2$ and $\textsc{sort}(y) = t_2$ then the agent $\underline{a}(\lambda x, y)(x, y) \,.\, \overline{x}\ y \,.\, \mathbf{0}$ is well-formed, since $s \mathrel{\underline{\propto}} t_1, t_2$ and $t_1 \mathrel{\overline{\propto}} t_2$, with $\textsc{vars}(x, y) = \{\{x, y\}\}$.

A formal comparison with the system in [6] is complicated by the fact that Milner uses so called concretions and abstractions as agents. Restricting attention to agents in the normal sense we have the following result, where $[\![\cdot]\!]$ is the function from the previous example.

Theorem 12. *P is well-sorted iff $[\![P]\!]$ is well-formed.*

Proof. A trivial induction over the structure of P, observing that the requirements are identical.

C.3 Polyadic synchronisation pi-calculus

Carbone and Maffeis [17] explore the so called pi-calculus with polyadic synchronisation, ${}^{e}\pi$, which can be thought of as a dual to the polyadic pi-calculus. Here action subjects are tuples of names, while the objects transmitted are just single names. It is demonstrated that this allows a gradual enabling of communication by opening the scope of names in a subject, results in simple representations of localities and cryptography, and gives a strictly greater expressiveness than standard pi-calculus.

In order to represent ${}^{e}\pi$, only minor modifications to the representation of the polyadic pi-calculus in Example C.1 are necessary. To allow tuples in subject position but not in object position, we invert the relations $\overline{\propto}$ and $\underline{\propto}$. Moreover, ${}^{e}\pi$ does not have name matching conditions $a = b$.

PSPI
Everything as in **PPI** except:
$\mathbf{C} = \{\top, \bot\}$ $\quad \mathbf{X} = \mathcal{N}$ $\quad \mathbf{1} \not\models \bot$
$\widetilde{a} \leftrightarrow \widetilde{b}$ is $\top$ if $\widetilde{a} = \widetilde{b}$, and $\bot$ otherwise
$\textsc{match}(a, \langle x \rangle, x) = \{a\}$
$\overline{\propto} = \underline{\propto} = \{(\texttt{tup}, \texttt{chan})\}$

In showing that this representation is fully abstract, for convenience we will consider a dialect of ${}^{e}\pi$ without the τ prefix. This has no cost in terms of expressiveness since the τ prefix can be encoded using a scoped communication. Let the mapping $[\![\cdot]\!]$ from ${}^{e}\pi$ processes into **PSPI** be homomorphic on all operators except sum, which is mapped to a $\top$-guarded **case** statement.

The proofs are similar to the polyadic pi-calculus case. The main differences are due to the fact that ${}^{e}\pi$ is formulated in terms of a late semantics with a structural congruence rule.

Lemma 2. *Let $\equiv$ be structural congruence on ${}^{e}\pi$ processes. If $P \equiv Q$ then $[\![P]\!] \sim [\![Q]\!]$.*

Proof. The relation $\mathcal{R} = \{(P, Q) : [\![P]\!] \sim [\![Q]\!]\}$ satisfies all the axioms defining $\equiv$ and is also a process congruence. Since $\equiv$ is the least such congruence, $\equiv \subseteq \mathcal{R}$.

Theorem 13.

1. *If $P \xrightarrow{\tilde{x}(y)} P'$ then for all z, $[\![P]\!] \xrightarrow{\langle \tilde{x} \rangle\, z} P''$ where $P'' \mathrel{\dot{\sim}} [\![P']\!][y := z]$.*
2. *If $P \xrightarrow{\alpha} P'$ and α is not an input, then $[\![P]\!] \xrightarrow{[\![\alpha]\!]} P''$ where $P'' \mathrel{\dot{\sim}} [\![P']\!]$.*
3. *If $[\![P]\!] \xrightarrow{\langle \tilde{x} \rangle\, z} P''$ then for all $y \# P$, $P \xrightarrow{\tilde{x}(y)} P'$ where $[\![P'\{z/y\}]\!] = P''$.*
4. *If $[\![P]\!] \xrightarrow{\alpha'} P''$ and α is not an input, then $P \xrightarrow{\alpha} P'$ where $[\![\alpha]\!] = \alpha'$ and $[\![P']\!] = P''$.*

Proof. By induction on the derivation of the transitions, using Lemma 2 in the Struct cases.

Polyadic synchronization pi has only guarded choice. We say that a **PSPI** process is case-guarded if in all its subterms of the form $\mathbf{case}\ \varphi_1 : P_1 \,[\!]\, \cdots \,[\!]\, \varphi_n : P_n$, if $\varphi_i = \top$ then $P_i = \overline{M}\ N.Q$ or $P_i = \underline{M}(\lambda\widetilde{x})X.Q$. The case-guarded **PSPI** processes correspond directly to polyadic synchronization pi processes with guarded choice.

Theorem 14. *For all case-guarded* **PSPI** *processes R there exists an ${}^e\pi$ process $\overline{R}$ such that $R \mathrel{\dot{\sim}} [\![\overline{R}]\!]$.*

C.4 Value-passing CCS

We can also encode value-passing CCS [15], using substitution to perform evaluation of expressions.

Value-passing CCS [15] is an extension of pure CCS to admit arbitrary data from some set $\mathbf{V}$ to be sent along channels; there is no dynamic connectivity so channel names cannot be transmitted. When a value is received in a communication it replaces the input variable everywhere, and where this results in a closed expression it is evaluated, so for example $a(x)\,.\,\overline{c}(x+3)$ can receive 2 along a and become $\overline{c}\ 5$. There are conditional **if** constructs that can test if a boolean expression evaluates to true, as in $a(x)\,.\,\mathbf{if}\ x > 3\ \mathbf{then}\ P$.

To represent this as a psi-calculus we assume an arbitrary set of value expressions $e \in \mathbf{E}$ of sort `exp`, a subset of which is the boolean expressions $b \in \mathbf{E_B}$. The names are either used as channels (and then have the sort `chan`) or expression variables (of sort `exp`); only the latter can appear in expressions and be substituted by values. An expression is closed if it has no name of sort `exp` in its support, otherwise it is open. The values $v \in \mathbf{V}$ have sort `value`, the boolean values are $\{\text{true}, \text{false}\}$. We let E be an evaluation function on expressions, that takes each closed expression to a value and leaves open expressions unchanged. We write $e\{\widetilde{V}/\widetilde{x}\}$ for the result of syntactically replacing all $\widetilde{x}$ simultaneously by $\widetilde{V}$ in the (boolean) expression e, and assume that the result is a valid (boolean) expression. For example $(x+3)\{2/x\} = 2+3$, and $E(2+3) = 5$. Evaluation takes place in substitution: $(x+3)[x := 2] = E((x+3)\{2/x\}) = E(2+3) = 5$. Since patterns are single names we need to add a failure pattern $\bot$ as explained in Example 1.

VPCCS	
$\mathcal{N} = \mathcal{N}_{\text{Ch}} \cup \mathcal{N}_{\text{Var}}$	$\mathcal{S}_{\mathcal{N}} = \{\texttt{chan}, \texttt{exp}\}$
$\mathbf{T} = \mathcal{N} \cup \mathbf{E} \cup \mathbf{V}$	$\mathcal{S} = \mathcal{S}_{\mathcal{N}} \cup \{\texttt{value}\}$
$\mathbf{C} = \mathbf{E_B}$	$e \in \mathbf{E} \Rightarrow \text{SORT}(e) = \texttt{exp}$
$\mathbf{A} = \{\mathbf{1}\}$	$v \in \mathbf{V} \Rightarrow \text{SORT}(v) = \texttt{value}$
$\mathbf{X} = \mathcal{N} \cup \{\bot\}$	$e \in \mathbf{E} \Rightarrow e[\widetilde{x} := \widetilde{M}] = E(e\{\widetilde{M}/\widetilde{x}\})$
$\mathbf{1} \vdash \text{true}, \quad \mathbf{1} \nvdash \text{false}$	$\prec\ = \{(\texttt{exp}, \texttt{value})\}$
$\leftrightarrows$ = identity on names	$\mathcal{S}_\nu = \{\texttt{chan}\}$
$\text{MATCH}(v, a, a) = \{v\}$ if $v \in \mathbf{V}$	$\overline{\propto} = \underline{\propto} = \{(\texttt{chan}, \texttt{exp}), (\texttt{chan}, \texttt{value})\}$
$\text{VARS}(a) = \{a\}$	

Closed value-passing CCS processes correspond to **VPCCS** agents P where all free names are of sort `chan`, i.e., where $\text{SORT}(\text{n}(P)) \subseteq \{\texttt{chan}\}$.

We show full abstraction with regards to value-passing CCS as defined by Milner [15], with the following modifications: We use replication instead of process constants, with the standard semantics. We consider only finite sums. Milner allows for infinite sums without specifying exactly what infinite sets are allowed and how they are represented, making a fully formal comparison difficult. Introducing infinite sums naively in psi-calculi means that agents might exhibit infinite support and exhaust the set of names, rendering crucial operations such as α-converting all bound names to fresh names impossible. We do not consider relabellings at all. Relabelling has fallen out of fashion since the same effect can be obtained by abstracting over channels, and it is not included in the psi-calculi framework. Only a finite number of channels may be restricted by the set L in a restriction $P \setminus L$. With finite sums, this results in no loss of expressivity since agents have finite support.

Milner's restrictions are sets of names, which we represent as a sequence of ν-binders; for that reason we use a total ordering of any set of names (this is always available since the set of names is countable). Formally we assume an injective and support-preserving function $\sigma : \mathcal{P}_{\text{fin}}(\mathcal{N}_{\texttt{chan}}) \to (\mathcal{N}_{\texttt{chan}})^*$. The mapping $[\![\cdot]\!]$ from value-passing CCS into **VPCCS** is defined homomorphically on all operators except the following:

$$\begin{aligned} [\![\Sigma_i\, P_i]\!] &= \mathbf{case}\ \top : [\![P_1]\!]\ [\!]\ \cdots\ [\!]\ \top : P_i \\ [\![\mathbf{if}\ b\ \mathbf{then}\ P]\!] &= \mathbf{case}\ b : [\![P]\!] \\ [\![P \setminus L]\!] &= (\nu\sigma(L))[\![P]\!] \end{aligned}$$

Lemma 3. *If P is a closed **VPCCS** process and $P \xrightarrow{\alpha} P'$, then P' is closed.*

Theorem 15. *If P and Q are closed value-passing CCS processes, then:*

1. *if $P \xrightarrow{\alpha} P'$ then $[\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$.*
2. *if $[\![P]\!] \xrightarrow{\alpha'} P''$ then $P \xrightarrow{\alpha} P'$ where $[\![\alpha]\!] = \alpha'$ and $[\![P']\!] = P''$.*

Proof. By induction on the derivations of P' and P'', respectively. The full proof is given in Appendix D.4.

Theorem 16. *For all closed value-passing CCS processes P and Q, $P \sim Q$ iff $[\![P]\!] \dot{\sim} [\![Q]\!]$*

Proof. $\{(P, Q) : [\![P]\!] \dot{\sim} [\![Q]\!] \wedge P, Q \text{ closed}\}$ is a bisimulation in value-passing CCS by coinduction, using Theorem 15.

Symmetrically, $\{(\mathbf{1}, [\![P]\!], [\![Q]\!]) : P \sim Q \wedge P, Q \text{ closed}\}$ is a bisimulation in **VPCCS**. The static equivalence and extension of arbitrary assertion cases are vacuous since there's only one frame. Symmetry follows from symmetry of $\sim$, and simulation follows by Theorem 15 and the fact that $\sim$ is a bisimulation.

Value-passing pi-calculus To demonstrate the modularity of psi-calculi, assume that we wish a variant of the pi-calculus enriched with values in the same way as value-passing CCS. This is achieved with only a minor change to **VPCCS**:

VPPI
Everything as in **VPCCS** except:
$\text{MATCH}(z, a, a) = \{z\}$ if $z \in \mathbf{V} \cup \mathcal{N}_{ch}$
$\prec = \{(\texttt{exp}, \texttt{value}), (\texttt{chan}, \texttt{chan})\}$
$\overline{\propto} = \underline{\propto} = \{(\texttt{chan}, \texttt{exp}), (\texttt{chan}, \texttt{value}), (\texttt{chan}, \texttt{chan})\}$

Here also channel names can substitute channel names, and they can be sent and received along channel names.

D Full proofs for Appendix C

The following is full proofs of Appendix 3; we present them here, in a seperate section, due to their length.

We will assume that the reader is acquainted with the relevant psi-calculi presented in Section 3, as well as the definitions, notation and terminology of Sangiorgi [16], Carbone and Maffeis [17], and Milner [15], respectively. We will use their notation except as concerns the treatment of bound names, where we will adopt our notation, e.g. we will write $\text{bn}(\alpha)\#Q$ instead of $\text{bn}(\alpha) \cap \text{fn}(Q) = \emptyset$.

D.1 Auxiliary Lemma

The following lemma is used in the isomorphism proofs in the subsequent sections.

Lemma 4 (Flatten Case). *Let I be a Psi instance. Suppose $\mathbf{A}_I = \{\mathbf{1}\}$ and there is an equivariant condition $\top \in \mathbf{C}_I$ such that $\mathbf{1} \vdash \top$. Let $R = \mathbf{case}\ \top : (\mathbf{case}\ \widetilde{\varphi} : \widetilde{P})\ [\!]\ \widetilde{\phi} : \widetilde{Q}$ and $R' = \mathbf{case}\ \widetilde{\varphi} : \widetilde{P}\ [\!]\ \widetilde{\phi} : \widetilde{Q}$; Then $R \sim R'$.*

Proof. Let $\mathcal{R} = \bigcup_\sigma \{(\mathbf{1}, R\sigma, R'\sigma), (\mathbf{1}, R'\sigma, R\sigma)\} \cup \sim$. We first show that $\mathcal{R}$ is a bisimulation. We only consider the simulation cases as the other cases are trivial (there is only one assertion). Suppose a transition from R is derived as follows

$$\textsc{Case}\ \frac{\textsc{Case}\ \dfrac{P_i\sigma \xrightarrow{\alpha} P_i' \qquad \mathbf{1} \vdash \varphi_i\sigma}{\mathbf{case}\ \widetilde{\varphi}\sigma : \widetilde{P}\sigma \xrightarrow{\alpha} P_i'}}{\mathbf{case}\ \top : (\mathbf{case}\ \widetilde{\varphi}\sigma : \widetilde{P}\sigma)\ [\!]\ \widetilde{\phi}\sigma : \widetilde{Q}\sigma \xrightarrow{\alpha} P_i'}$$

then R' can simulate this with the following

$$\textsc{Case}\ \frac{P_i\sigma \xrightarrow{\alpha} P_i' \qquad \mathbf{1} \vdash \varphi_i\sigma}{\mathbf{case}\ \widetilde{\varphi}\sigma : \widetilde{P}\sigma\ [\!]\ \widetilde{\phi} : \widetilde{Q} \xrightarrow{\alpha} P_i'}$$

By reflexivity of $\sim$, we know $P'_i \sim P'_i$. By noting $\sim \subseteq \mathcal{R}$, we can conclude this case $\forall\sigma.(\mathbf{1}, P'_i\sigma, P'_i\sigma) \in \mathcal{R}$.

In case a transition of R is derived by

$$\textsc{Case}\ \frac{Q_i\sigma \xrightarrow{\alpha} Q'_i \qquad \mathbf{1} \vdash \phi_i\sigma}{\mathbf{case}\ \top : (\mathbf{case}\ \widetilde{\varphi}\sigma : \widetilde{P}\sigma)\ [\!]\ \widetilde{\phi}\sigma : \widetilde{Q}\sigma \xrightarrow{\alpha} Q'_i}$$

R' can simulate it with

$$\textsc{Case}\ \frac{Q_i\sigma \xrightarrow{\alpha} Q'_i \qquad \mathbf{1} \vdash \phi_i\sigma}{\mathbf{case}\ \widetilde{\varphi}\sigma : \widetilde{P}\sigma\ [\!]\ \widetilde{\phi}\sigma : \widetilde{Q}\sigma \xrightarrow{\alpha} Q'_i}$$

Again by reflexivity of $\sim$ we find $Q'_i \sim Q'_i$, and thus $\forall\sigma.(\mathbf{1}, Q'_i\sigma, Q'_i\sigma) \in \sim \subseteq \mathcal{R}$.

By a similar argument, we can show that R simulates R'. Since $\mathcal{R}$ is a bisimulation that is closed under all substitutions, $\mathcal{R} \subseteq \sim$.

D.2 Polyadic Pi-Calculus

We follow the exposition of Polyadic Pi-Calculus given by Sangiorgi in [16] with only departure being that we use replication in the labelled operational semantics instead of process constant invocation.

For convenience, we give an explicit definition of the encoding function given in Example C.1.

Definition 11 (Polyadic Pi-Calculus to PPi).
Agents:

$$\begin{aligned}
[\![P + Q]\!] &= \mathbf{case}\ \top : [\![P]\!]\ [\!]\ \top : [\![Q]\!] \\
[\![[x = y]P]\!] &= \mathbf{case}\ x = y : [\![P]\!] \\
[\![x(\tilde{y}).P]\!] &= \underline{x}(\lambda\widetilde{y})\langle\widetilde{y}\rangle.[\![P]\!] \\
[\![\overline{x}\langle\tilde{y}\rangle.P]\!] &= \overline{x}\langle\tilde{y}\rangle.[\![P]\!] \\
[\![0]\!] &= 0 \\
[\![P \mid Q]\!] &= [\![P]\!] \mid [\![Q]\!] \\
[\![\nu x P]\!] &= (\nu x)[\![P]\!] \\
[\![!P]\!] &= ![\![P]\!]
\end{aligned}$$

Actions:

$$\begin{aligned}
[\![(\nu\widetilde{y'})\overline{z}\langle\tilde{y}\rangle]\!] &= \overline{z}\ (\nu\widetilde{y'})\ \langle\widetilde{y}\rangle \\
[\![x\langle\tilde{z}\rangle]\!] &= \underline{x}\ \langle\tilde{z}\rangle \\
[\![\tau]\!] &= \tau
\end{aligned}$$

In output action $\tilde{y'}$ do not bind into z.

Definition 12 (PPi to Polyadic Pi-Calculus).
Process:

$$\begin{aligned}
\overline{(\!|\mathbf{1}|\!)} &= \mathbf{0} \\
\overline{\mathbf{0}} = \overline{\mathbf{case}} &= \mathbf{0} \\
\overline{\mathbf{case}\ \varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n} &= \overline{\varphi_1 : P_1} + \cdots + \overline{\varphi_n : P_n} \\
\overline{!P} &= !\overline{P} \\
\overline{(\nu x)P} &= \nu x\overline{P} \\
\overline{P \mid Q} &= \overline{P} \mid \overline{Q} \\
\overline{\underline{x}(\lambda\widetilde{y})\langle\widetilde{y}\rangle.P} &= x(\tilde{y}).\overline{P} \\
\overline{\overline{x}\langle\tilde{y}\rangle.P} &= \overline{x}\langle\tilde{y}\rangle.\overline{P}
\end{aligned}$$

Case clause:

$$\begin{aligned}
\overline{x = y : P} &= [x = y]\overline{P} \\
\overline{\top : P} &= \overline{P}
\end{aligned}$$

The following is proof of the strong operational correspondence.

Proof (of Theorem 10).

1. We show $P \xrightarrow{\alpha} P' \Rightarrow [\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$ by induction on the derivation of P'.

   Alp:
   Trivial in nominal logic.

   Out:
   We have that $\overline{x}\langle\tilde{y}\rangle.P \xrightarrow{\overline{x}\langle\tilde{y}\rangle} P$. Since $x \leftrightarrow x$, we can derive $\overline{x}\ \langle\tilde{y}\rangle.[\![P]\!] \xrightarrow{\overline{x}\ \langle\tilde{y}\rangle} [\![P]\!]$.

   Inp:
   We have that $x(\tilde{y}).P \xrightarrow{x\langle\tilde{z}\rangle} P\{\tilde{z}/\tilde{y}\}$ with $|\tilde{z}| = |\tilde{x}|$, hence $\widetilde{z} \in \textsc{match}(\langle\widetilde{z}\rangle, \widetilde{y}, \langle\widetilde{y}\rangle)$. Using this and $x \leftrightarrow x$ we can derive $\underline{x}(\lambda\widetilde{y})\langle\widetilde{y}\rangle.[\![P]\!] \xrightarrow{\underline{x}\ \langle\tilde{z}\rangle} [\![P]\!][\widetilde{y} := \widetilde{z}]$. By an easily checkable equality, $[\![P]\!][\widetilde{y} := \widetilde{z}] = [\![P\{\tilde{z}/\tilde{y}\}]\!]$, which completes the proof.

   Sum:
   Here $P \xrightarrow{\alpha} P'$ and by induction, $[\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$. Thus we can derive $\mathbf{case}\ \top : [\![P]\!]\ []\ \top : [\![Q]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$.

   Par:
   Here $P \xrightarrow{\alpha} P'$ and $\mathrm{bn}(\alpha)\#Q$, and by induction, $[\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$. Then we can choose an alpha-variant of the frame of $[\![Q]\!]$ which is sufficiently fresh to allow the derivation $[\![P]\!] \mid [\![Q]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!] \mid [\![Q]\!]$.

   Com:
   Here $P \xrightarrow{(\nu\tilde{y'})\overline{x}\langle\tilde{y}\rangle} P'$, $Q \xrightarrow{x\langle\tilde{y}\rangle} Q'$ with $\tilde{y'} \subseteq \tilde{y}$ and $\tilde{y'}\#Q$. By induction, $[\![P]\!] \xrightarrow{\overline{x}\ (\nu\widetilde{y'})\ \langle\widetilde{y}\rangle} [\![P']\!]$ and $[\![Q]\!] \xrightarrow{\underline{x}\ \langle\tilde{y}\rangle} [\![Q']\!]$. Moreover, we note that $x \leftrightarrow x$. Finally, we can choose alpha-variants of the frames of $[\![P]\!]$ and $[\![Q]\!]$ which are sufficiently fresh to allow the derivation $[\![P]\!] \mid [\![Q]\!] \xrightarrow{\tau} (\nu\widetilde{y'})([\![P']\!] \mid [\![Q']\!])$.

Match:
Here $P \xrightarrow{\alpha} P'$ and by induction, $[\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$. Thus we can derive $\mathbf{case}\ x = x : [\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$.

Rep:
Here $P \mid !P \xrightarrow{\alpha} P'$ and by induction, $[\![P]\!] \mid ![\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$. Thus we can derive $![\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$.

Res:
Here $P \xrightarrow{\alpha} P'$ with $x \# \alpha$, and by induction, $[\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$. Hence we derive $(\nu x)[\![P]\!] \xrightarrow{[\![\alpha]\!]} (\nu x)[\![P']\!]$.

Open:
Here $P \xrightarrow{(\nu \tilde{y'})\overline{z}\langle \tilde{y} \rangle} P'$ with $x \neq z$, $x \# \tilde{y'}$ and $x \in \tilde{y}$. By induction, $[\![P]\!] \xrightarrow{\overline{z}\,(\nu \tilde{y'})\,\langle \tilde{y} \rangle} [\![P']\!]$. Hence we derive $(\nu x)[\![P]\!] \xrightarrow{\overline{z}\,(\nu \tilde{y'} \cup x)\,\langle \tilde{y} \rangle} [\![P']\!]$.

2. We now show that if $[\![P]\!] \xrightarrow{\alpha'} P''$ then $P \xrightarrow{\alpha} P'$ where $[\![\alpha]\!] = \alpha'$ and $[\![P']\!] = P''$. The proof is similar, by induction on the derivation of $[\![P']\!]$. We show only the "interesting" case:

Case:
Here $[\![P]\!] \xrightarrow{\alpha'} P''$ and by induction, $P \xrightarrow{\alpha} P'$ where $[\![\alpha]\!] = \alpha'$ and $[\![P']\!] = P''$. Since $P_C = \mathbf{case}\ \widetilde{\varphi} : \widetilde{P}$ is in the range of $[\![\cdot]\!]$, either $P_C = \top : [\![P]\!] \,[]\, \top : [\![Q]\!]$, $P_C = \top : [\![Q]\!] \,[]\, \top : [\![P]\!]$ or $P_C = \mathbf{case}\ x = y : [\![P]\!]$. We proceed by case analysis:

(a) When $P_C = \top : [\![P]\!] \,[]\, \top : [\![Q]\!]$, we note that $[\![P + Q]\!] = P_C$ and imitate the derivation of P'' from P_C with the derivation $P + Q \xrightarrow{\alpha} P'$, using the SUM rule.

(b) The case when $P_C = \top : [\![Q]\!] \,[]\, \top : [\![P]\!]$ is symmetric to the previous case.

(c) When $P_C = \mathbf{case}\ x = y : [\![P]\!]$, since $1 \vdash x = y$ by the induction hypothesis, $x = y$. we note that $[\![[x = x]P]\!] = P_C$ and imitate the derivation of P'' from P_C with the derivation $[x = x]P \xrightarrow{\alpha} P'$, using the MATCH rule.

Proof (of Theorem 11).
By structural induction on P. We only consider the case of **case** agent as other cases are trivial.

case case $\varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n$**:**
We get an induction hypothesis for every $i \in \{1..n\}$, IH_i: $P_i \sim [\![\overline{P_i}]\!]$.
We proceed by induction on n.

base case $n = 0$**:**
$[\![\overline{\mathbf{case}}]\!] = [\![\mathbf{0}]\!] = \mathbf{0}$. By reflexivity of $\sim$, $\mathbf{0} \sim \mathbf{0}$.

induction step $n+1$:

The IH for this case is

$$[\![\overline{\textbf{case}\ \varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n}]\!] \sim \textbf{case}\ \varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n = P'$$

We need to show that $Q \sim [\![\overline{Q}]\!]$ for $Q = \textbf{case}\ \varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n \,[]\, \varphi_{n+1} : P_{n+1}$.
We compute

$$\begin{aligned}
[\![\overline{Q}]\!] &= [\![\overline{\varphi_1 : P_1} + \cdots + \overline{\varphi_n : P_n} + \overline{\varphi_{n+1} : P_{n+1}}]\!] \\
&= \textbf{case}\ \top : [\![\overline{\varphi_1 : P_1}]\!] \,[]\, \ldots \,[]\, \top : [\![\overline{\varphi_n : P_n}]\!] \,[]\, \top : [\![\overline{\varphi_{n+1} : P_{n+1}}]\!] \\
&\sim \text{(by Lemma 4)} \\
&\quad \textbf{case}\ \top : (\textbf{case}\ \top : [\![\overline{\varphi_1 : P_1}]\!] \,[]\, \ldots \,[]\, \top : [\![\overline{\varphi_n : P_n}]\!]) \,[]\, \top : [\![\overline{\varphi_{n+1} : P_{n+1}}]\!] \\
&\sim \text{(by } IH\text{)} \\
&\quad \textbf{case}\ \top : (\textbf{case}\ \varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n) \,[]\, \top : [\![\overline{\varphi_{n+1} : P_{n+1}}]\!] \\
&= \textbf{case}\ \top : P' \,[]\, \top : [\![\overline{\varphi_{n+1} : P_{n+1}}]\!] \\
&= Q'
\end{aligned}$$

We distinguishe the cases of φ_{n+1}:

case $\varphi_{n+1} = \top$:

$$\begin{aligned}
Q' &= \textbf{case}\ \top : P' \,[]\, \top : [\![\overline{\top : P_{n+1}}]\!] \\
&= \textbf{case}\ \top : P' \,[]\, \top : [\![\overline{P_{n+1}}]\!] \\
&\sim \text{(by } IH_{n+1}\text{)} \\
&\quad \textbf{case}\ \top : P' \,[]\, \top : P_{n+1} \\
&\sim \text{(by Lemma 4)} \\
&\quad \textbf{case}\ \varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n \,[]\, \top : P_{n+1} = Q
\end{aligned}$$

We conclude this case.

case $\varphi_{n+1} = x = y$:

$$\begin{aligned}
Q' &= \textbf{case}\ \top : P' \,[]\, \top : [\![\overline{x = y : P_{n+1}}]\!] \\
&= \textbf{case}\ \top : P' \,[]\, \top : (\textbf{case}\ x = y : [\![\overline{P_{n+1}}]\!]) \\
&\sim \text{(by } IH_{n+1}\text{)} \\
&\quad \textbf{case}\ \top : P' \,[]\, \top : (\textbf{case}\ x = y : P_{n+1}) \\
&\sim \text{(by Lemma 4)} \\
&\quad \textbf{case}\ \varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n \,[]\, \top : (\textbf{case}\ x = y : P_{n+1}) \\
&\sim \text{(by Lemma 4)} \\
&\quad \textbf{case}\ \varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n \,[]\, x = y : P_{n+1} = Q
\end{aligned}$$

By concluding this case, we conclude the proof.

From the strong operational correspondence, we obtain full abstraction.

Theorem 17. *$P \sim_e Q$ iff $[\![P]\!] \dot{\sim} [\![Q]\!]$*

Proof. $\{(P,Q) : [\![P]\!] \dot{\sim} [\![Q]\!]\}$ is an early bisimulation in the polyadic pi-calculus by coinduction, using Theorem 10.

Symmetrically, $\{(\mathbf{1}, [\![P]\!], [\![Q]\!]) : P \sim_e Q\}$ is a bisimulation in **PPi**. The static equivalence and extension of arbitrary assertion cases are vacuous since there's only one frame. Symmetry follows from symmetry of $\sim_e$, and simulation follows by Theorem 10 and the fact that $\sim_e$ is an early bisimulation.

Lemma 5. *$[\![\cdot]\!]$ is injective, that is, for all P, Q, if $[\![P]\!] = [\![Q]\!]$ then $P = Q$.*

Proof. By induction on P and Q while inspecting all the possible cases.

Lemma 6. *$[\![\cdot]\!]$ is surjective up to $\sim$, that is, for every P there is a Q such that $[\![Q]\!] \sim P$.*

Proof. By structural induction on the well formed agent P.

case $\underline{x}(\lambda\widetilde{y})\langle\widetilde{y}\rangle.P'$**:**
IH tells us that, for some Q', $[\![Q']\!] \sim P'$. Let $Q = x(\tilde{y}).Q'$. Then, $[\![Q]\!] = [\![x(\tilde{y}).Q']\!] = \underline{x}(\lambda\widetilde{y})\langle\widetilde{y}\rangle.[\![Q']\!] \sim \underline{x}(\lambda\widetilde{y})\langle\widetilde{y}\rangle.P'$. This is what we needed to derive.

case $\overline{x}\langle\tilde{y}\rangle.P'$**:**
By IH, we have for some Q', $[\![Q']\!] \sim P'$. Let $Q = \overline{x}\langle\tilde{y}\rangle.Q'$. Now $[\![Q]\!] = \overline{x}\langle\tilde{y}\rangle.[\![Q']\!] \sim \overline{x}\langle\tilde{y}\rangle.P'$, which is what we wanted to derive.

case $P \mid P'$**:**
By IH, we have that for some Q', Q'', $[\![Q']\!] \sim P$ and $[\![Q'']\!] \sim P'$. Then let $Q = Q' \mid Q''$, thus $[\![Q]\!] = [\![Q']\!] \mid [\![Q'']\!] \sim P \mid P'$.

case $(\nu x)P$**:**
By IH, for some Q', $[\![Q']\!] \sim P$. Let $Q = \nu x Q'$. Then $[\![Q]\!] = (\nu x)[\![Q']\!] \sim (\nu x)P$.

case $!P$**:**
By IH, for some Q', $[\![Q']\!] \sim P$. Let $Q = !Q'$. Then $[\![Q]\!] = ![\![Q']\!] \sim !P$.

case $(\!|\mathbf{1}|\!)$**:**
Let $Q = \mathbf{0}$. Then $[\![Q]\!] = \mathbf{0} \sim (\!|\mathbf{1}|\!)$.

case case $\widetilde{\varphi} : \widetilde{P'}$**:**
For induction hypothesis IH**case**, we have for every i there is Q'_i such that $[\![Q'_i]\!] \sim P'_i$. The proof goes by induction on the length of $\widetilde{\varphi}$.

base case:
Let $Q = \mathbf{0}$, then $[\![Q]\!] = \mathbf{0} \sim$ **case**.

induction step:
At this step, we get the following IH

$$[\![Q'']\!] \sim \mathbf{case}\ \varphi_1 : P_1\ [] \ldots [] \ \varphi_n : P_n$$

We need to show that there is some $[\![Q]\!]$ such that

$$[\![Q]\!] \sim \mathbf{case}\ \varphi_1 : P_1\ [] \ldots [] \ \varphi_n : P_n\ [] \ \varphi_{n+1} : P_{n+1}$$

First, we note that $\mathrm{IH}_{\mathbf{case}}$ holds for every i and in particular $i = n+1$, thus we get $[\![Q'_{n+1}]\!] \sim P_{n+1}$. Second, we note that φ_{n+1} has two forms, thus we proceed by case analysis on φ_{n+1}.

case $\varphi_{n+1} = \top$**:**
Let $Q = Q'' + Q'_{n+1}$. Then

$$\begin{aligned}
[\![Q]\!] &= \mathbf{case}\ \top : [\![Q'']\!] \,[]\, \top : [\![Q'_{n+1}]\!] \\
&\sim \mathbf{case}\ \top : (\mathbf{case}\ \varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n) \\
&\quad []\, \top : [\![Q'_{n+1}]\!] \\
&\sim \mathbf{case}\ \top : (\mathbf{case}\ \varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n) \\
&\quad []\, \top : P_{n+1} \\
&\sim (\text{by Lemma 4}) \\
&\quad \mathbf{case}\ \varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n \\
&\quad []\, \top : P_{n+1}
\end{aligned}$$

This case is concluded.

case $\varphi_{n+1} = x = y$**:**
Let $Q = Q'' + [x = y]Q'_{n+1}$. Then

$$\begin{aligned}
[\![Q]\!] &= \mathbf{case}\ \top : [\![Q'']\!] \,[]\, \top : [\![[x = y]Q'_{n+1}]\!] \\
&\sim \mathbf{case}\ \top : (\mathbf{case}\ \varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n) \\
&\quad []\, \top : (\mathbf{case}\ x = y : [\![Q'_{n+1}]\!]) \\
&\sim \mathbf{case}\ \top : (\mathbf{case}\ \varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n) \\
&\quad []\, \top : (\mathbf{case}\ x = y : P_{n+1}) \\
&\sim (\text{by Lemma 4}) \\
&\quad \mathbf{case}\ \varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n \\
&\quad []\, \top : (\mathbf{case}\ x = y : P_{n+1}) \\
&\sim (\text{by permuting and applying Lemma 4}) \\
&\quad \mathbf{case}\ \varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n \,[]\, x = y : P_{n+1}
\end{aligned}$$

This is the last part we needed to check, we conclude the proof.

Theorem 18. $[\![\cdot]\!]$ *is an isomorphism up to* $\sim$.

Proof. Directly follows from Lemma 5 and Lemma 6

D.3 Polyadic Synchronisation Pi-Calculus

We follow the exposition of Polyadic Synchronisation Pi-Calculus, ${}^{e}\pi$, of Carbone and Maffeis [17].

We give an explicit definition of encoding function defined in Example C.3.

Definition 13 (Polyadic synchronisation pi-calculus to PSPi).
Agents:

$$\begin{aligned}
[\![\widetilde{x}(y).P]\!] &= \langle\widetilde{x}\rangle(\lambda y)y.[\![P]\!] \\
[\![\widetilde{x}\langle y\rangle.P]\!] &= \overline{\langle\widetilde{x}\rangle}\ y.[\![P]\!] \\
[\![P \mid Q]\!] &= [\![P]\!] \mid [\![Q]\!] \\
[\![(\nu x)P]\!] &= (\nu x)[\![P]\!] \\
[\![!P]\!] &= ![\![P]\!] \\
[\![0]\!] &= 0 \\
[\![\Sigma_i \alpha_i.P_i]\!] &= \mathbf{case}\ \top_i : [\![\alpha_i.P_i]\!]
\end{aligned}$$

Actions:

$$\begin{aligned}
[\![\tilde{x}\langle\nu c\rangle]\!] &= \overline{\langle\tilde{x}\rangle}\ (\nu c)\ c \\
[\![\tilde{x}\langle c\rangle]\!] &= \overline{\langle\tilde{x}\rangle}\ c \\
[\![\tau]\!] &= \tau \\
[\![\tilde{x}(y)]\!] &= \textit{undefined}
\end{aligned}$$

Because in [17] Carbone and Maffeis defines late style laballed semantics for ${}^e\pi$ the input action has no translation.

Definition 14 (PSPi to Polyadic synchronisation pi-calculus).

$$\begin{aligned}
\overline{(\!|\mathbf{1}|\!)} &= \mathbf{0} \\
\overline{\mathbf{0}} &= \mathbf{0} \\
\overline{!P} &= !\overline{P} \\
\overline{(\nu x)P} &= (\nu x)\overline{P} \\
\overline{P \mid Q} &= \overline{P} \mid \overline{Q} \\
\overline{\langle\widetilde{a}\rangle y.P} &= \overline{a}\langle y\rangle.\overline{P} \\
\overline{\underline{\widetilde{x}}(\lambda y)y.P} &= \overline{x}(y).\overline{P} \\
\overline{\tau.P} &= \tau.\overline{P} \\
\overline{\mathbf{case}\ \top : \alpha_i.P_i} &= \Sigma_i\overline{\alpha_i.P_i}
\end{aligned}$$

Lemma 7 (Lemma 2). *If $P \equiv Q$ then $[\![P]\!] \sim [\![Q]\!]$*

Proof. The relation $\mathcal{R} = \{(P, Q) : [\![P]\!] \sim [\![Q]\!]\}$ satisfies all the axioms defining $\equiv$ and is also a process congruence. Since $\equiv$ is the least such congruence, $\equiv \subseteq \mathcal{R}$.

We give proof for the strong operational correspondence.

Proof (of Theorem 13).

1. By induction on the derivation of P', avoiding z.

   Prefix:
   Here $\Sigma_i\tilde{x}_i(y_i).P_i \xrightarrow{\tilde{x}_i(y_i)} P_i$. We have that

$$\begin{aligned}
[\![\Sigma_i\tilde{x}_i(y_i).P_i]\!] = \mathbf{case}\ \top : \underline{\langle\tilde{x}\rangle}(\lambda y_1)y_1.[\![P_1]\!]\ [] \\
\cdots\ [] \ \top : \underline{\langle\tilde{x}\rangle}(\lambda y_i)y_i.[\![P_i]\!]
\end{aligned}$$

Since $\textsc{match}(z, \langle y_i \rangle, y_i) = \{z\}$, we can use the Case and In rules to derive the transition

$$\mathbf{case}\ \top : \underline{\langle \tilde{x}_1 \rangle}(\lambda y_1) y_1.[\![P_1]\!] \,[]\, \cdots \,[]\, \top : \underline{\langle \tilde{x}_i \rangle}(\lambda y_i) y_i.[\![P_i]\!] \xrightarrow{\langle \tilde{x} \rangle\ z} [\![P_i]\!][y_i := z]$$

Finally, we have $P'' = [\![P_i]\!][y_i := z]$ and use reflexivity of $\dot{\sim}$.

Bang:
Here $P \mid !P \xrightarrow{\tilde{x}(y)} P'$ and by induction, $[\![P]\!] \mid ![\![P]\!] \xrightarrow{\langle \tilde{x} \rangle\ z} P''$ with $P'' \dot{\sim} [\![P']\!][y := z]$. By rule Rep, we also have that $![\![P]\!] \xrightarrow{\langle \tilde{x} \rangle\ z} P''$.

Par:
Here $P \xrightarrow{\tilde{x}(y)} P'$, $y \# Q$ and by induction, $[\![P]\!] \xrightarrow{\langle \tilde{x} \rangle\ z} P''$ with $P'' \dot{\sim} [\![P']\!][y := z]$. Using the Par rule we derive $[\![P]\!] \mid [\![Q]\!] \xrightarrow{\langle \tilde{x} \rangle\ z} P' \mid [\![Q]\!]$. Since $\dot{\sim}$ is closed under $\mid$, $P'' \mid [\![Q]\!] \dot{\sim} [\![P']\!][y := z] \mid [\![Q]\!]$. Finally, since $y \# Q$, $[\![P']\!][y := z] \mid [\![Q]\!] = [\![P' \mid Q]\!][y := z]$.

Struct:
Here $P \equiv Q$, $Q \xrightarrow{\tilde{x}(y)} Q'$ and $Q' \equiv P'$. By induction we obtain Q'' such that $[\![Q]\!] \xrightarrow{\langle \tilde{x} \rangle\ z} Q''$ where $Q'' \dot{\sim} [\![Q']\!][y := z]$. By Lemma 2, $[\![P]\!] \sim [\![Q]\!]$ and $[\![Q']\!] \sim [\![P']\!]$, and by definition of $\sim$, $[\![Q']\!][y := z] \sim [\![P']\!][y := z]$. Since $[\![P]\!] \sim [\![Q]\!]$ and $[\![Q]\!] \xrightarrow{\langle \tilde{x} \rangle\ z} Q''$, there exists P'' such that $[\![P]\!] \xrightarrow{\langle \tilde{x} \rangle\ z} P''$ and $Q'' \dot{\sim} P''$. By transitivity of $\dot{\sim}$ and the fact that $\sim \subseteq \dot{\sim}$, $P'' \dot{\sim} [\![P']\!][y := z]$.

Res:
Here $P \xrightarrow{\tilde{x}(y)} P'$, $a \neq y$, $a \neq z$ $a \# \tilde{x}$, and by induction, $[\![P]\!] \xrightarrow{\langle \tilde{x} \rangle\ z} P''$ with $P'' \dot{\sim} [\![P']\!][y := z]$. This gives us sufficient freshness conditions to derive $(\nu a)[\![P]\!] \xrightarrow{\langle \tilde{x} \rangle\ z} (\nu a) P''$. Since $\dot{\sim}$ is closed under restriction, $(\nu a) P'' \dot{\sim} (\nu a)([\![P']\!][y := z])$. Finally, a is sufficiently fresh to so that $(\nu a)([\![P']\!][y := z]) = ((\nu a)[\![P']\!])[y := z]$

2. By induction on the derivation of P'. The cases not shown here are similar to the previous clause of this theorem, where P does an input.

Comm:
Here $P \xrightarrow{\overline{\tilde{x}}\langle y \rangle} P'$ and $Q \xrightarrow{\tilde{x}(z)} Q'$. By induction, $[\![P]\!] \xrightarrow{\overline{\langle \tilde{x} \rangle}\ y} P''$ where $P'' \dot{\sim} [\![P']\!]$ and by the previous clause of this theorem, $[\![Q]\!] \xrightarrow{\langle \tilde{x} \rangle\ y} Q''$ such that $[\![Q']\!][z := y] \dot{\sim} Q''$. The Com rule lets us derive the transition

$$[\![P]\!] \mid [\![Q]\!] \xrightarrow{\tau} P'' \mid Q''$$

To complete the induction case, we note that $(\nu y)(P'' \mid Q'') \dot{\sim} [\![(\nu y)(P' \mid Q'\{y/z\})]\!]$

Close:
Here $P \xrightarrow{\overline{\tilde{x}}\langle \nu y \rangle} P'$ and $Q \xrightarrow{\tilde{x}(y)} Q'$. We assume $y \# Q$; if not, y can be

α-converted so that this holds. By induction, $[\![P]\!] \xrightarrow{\overline{\langle \tilde{x} \rangle}\,(\nu y)\,y} P''$ where $P'' \mathrel{\dot{\sim}} [\![P']\!]$ and by the previous clause of this theorem, $[\![Q]\!] \xrightarrow{\langle \tilde{x} \rangle\, y} Q''$ such that $[\![Q']\!][y := y] = [\![Q']\!] \mathrel{\dot{\sim}} Q''$. The COM rule lets us derive the transition

$$[\![P]\!] \mid [\![Q]\!] \xrightarrow{\tau} (\nu y)(P'' \mid Q'')$$

To complete the induction case, we note that $(\nu y)(P'' \mid Q'') \mathrel{\dot{\sim}} [\![(\nu y)(P' \mid Q')]\!]$

Open:

Here $P \xrightarrow{\overline{\tilde{x}}\langle y \rangle} P'$ with $y \neq x$, and by induction, $[\![P]\!] \xrightarrow{\overline{\langle \tilde{x} \rangle}\, y} P''$ where $P'' \mathrel{\dot{\sim}} [\![P']\!]$. By OPEN, we derive $(\nu y)[\![P]\!] \xrightarrow{\overline{\langle \tilde{x} \rangle}\,(\nu y)\,y} P''$.

3. By induction on the derivation of P", avoiding y.

Par:

Here $[\![P]\!] \xrightarrow{\underline{x}\,\langle \tilde{z} \rangle} P''$, $y \# P, Q$, and by induction $P \xrightarrow{\tilde{x}(y)} P'$ where $[\![P'\{z/y\}]\!] = P''$. By PAR using $y \# Q$, we derive $P \mid Q \xrightarrow{\tilde{x}(y)} P' \mid Q$. Finally, we note that since $y \# Q$, $[\![(P' \mid Q)\{z/y\}]\!] = P'' \mid [\![Q]\!]$.

Case:

Here $P_C \xrightarrow{\underline{\tilde{x}}\,z} P''$, where $P_C = \mathbf{case}\ \widetilde{\varphi} : \widetilde{Q}$ is in the range of $[\![\cdot]\!]$ - hence P_C must be the encoding of some prefix-guarded sum, ie $P_C = [\![\Sigma_i \alpha_i.P_i]\!] = \mathbf{case}\ \top : [\![\alpha_1]\!].[\![P_1]\!]\ [] \ldots [] \top : [\![\alpha_i]\!].[\![P_i]\!]$. By transition inversion we can deduce that for some j, $\alpha_j = \tilde{x}(y)$ and $[\![P_j]\!][y := z] = P''$. By the PREFIX rule, $\Sigma_i \alpha_i.P_i \xrightarrow{\tilde{x}(y)} P_j$.

Out:

A special case of CASE.

Rep:

Here $[\![P]\!] \mid\, ![\![P]\!] \xrightarrow{\underline{x}\,\langle \tilde{z} \rangle} P''$ and by induction $P \mid\, !P \xrightarrow{\tilde{x}(y)} P'$ where $[\![P'\{z/y\}]\!] = P''$. By BANG we derive $!P \xrightarrow{\tilde{x}(y)} P'$.

Scope:

Here $[\![P]\!] \xrightarrow{\underline{x}\,\langle \tilde{z} \rangle} P''$, $y \# P, Q$, $a \# \tilde{x}, y, z$ and by induction $P \xrightarrow{\tilde{x}(y)} P'$ where $[\![P'\{z/y\}]\!] = P''$. Since $a \# \tilde{x}, y, z$, the RES rule admits the derivation $(\nu a)P \xrightarrow{\tilde{x}(y)} (\nu a)P'$, and $[\![((\nu a)P')\{z/y\}]\!] = (\nu a)P''$

4. By induction on the derivation of P". The cases not shown are similar to the previous clause of this theorem.

Com:

Here $[\![P]\!] \xrightarrow{\overline{\langle \tilde{x} \rangle}\,(\nu \tilde{y'})\,y} P''$, $[\![Q]\!] \xrightarrow{\langle \tilde{x} \rangle\, y} Q''$ and $y' \# Q$. Either $\tilde{y'} = \epsilon$ or $\tilde{y'} = y$; we proceed by case analysis.

(a) If $\tilde{y'} = \epsilon$, we have $P \xrightarrow{\overline{\tilde{x}}\langle y \rangle} P'$ where $[\![P']\!] = P''$ by induction and, by the previous clause of this theorem, $Q \xrightarrow{\tilde{x}(z)} Q'$ where $[\![Q'\{y/z\}]\!] = Q''$. The COMM rule then lets us derive $P \mid Q \xrightarrow{\tau} P' \mid Q'\{y/z\}$.

(b) If $\tilde{y'} = y$, we have $P \xrightarrow{\overline{\tilde{x}}\langle \nu y \rangle} P'$ where $[\![P']\!] = P''$ by induction and, by the previous clause of this theorem, $Q \xrightarrow{\tilde{x}(y)} Q'$ where $[\![Q'\{y/y\}]\!] = [\![Q']\!] = Q''$. The CLOSE rule then lets us derive $P \mid Q \xrightarrow{\tau} (\nu y)(P' \mid Q')$.

Open:

Here $[\![P]\!] \xrightarrow{\overline{\langle \tilde{x} \rangle}\, y} P''$ with $y \neq x$. By induction, $P \xrightarrow{\overline{\tilde{x}}\langle y \rangle} P'$ where $[\![P']\!] = P''$. By rule OPEN, $(\nu y)P \xrightarrow{\overline{\tilde{x}}\langle \nu y \rangle} P'$.

We give the full abstraction result for this calculus.

Theorem 19. *For all ${}^e\pi$ processes P and Q, $P \dot{\sim} Q$ iff $[\![P]\!] \dot{\sim} [\![Q]\!]$*

Proof. $\mathcal{R} = \{(P, Q) : [\![P]\!] \dot{\sim} [\![Q]\!]\}$ is an early bisimulation in the polyadic synchronisation pi-calculus; if $P \mathcal{R} Q$ then

1. If $P \xrightarrow{\tilde{x}(y)} P'$ and $[\![P]\!] \dot{\sim} [\![Q]\!]$, since $\mathcal{R}$ is equivariant, we can assume that $y \# P, Q$ without loss of generality. Fix z. By Theorem 13.1, $[\![P]\!] \xrightarrow{\langle \tilde{x} \rangle\, z} P''$ where $P'' \dot{\sim} [\![P']\!][y := z] = [\![P'\{z/y\}]\!]$. Hence, since $[\![P]\!] \dot{\sim} [\![Q]\!]$, $[\![Q]\!] \xrightarrow{\langle \tilde{x} \rangle\, z} Q''$ where $P'' \dot{\sim} Q''$. Hence, by Theorem 13.3 using $y \# Q$, $Q \xrightarrow{\tilde{x}(y)} Q'$ where $[\![Q'\{z/y\}]\!] = Q''$. By transitivity, $[\![P'\{z/y\}]\!] \dot{\sim} [\![Q'\{z/y\}]\!]$.
2. If $P \xrightarrow{\alpha} P'$ and $[\![P]\!] \dot{\sim} [\![Q]\!]$, since $\mathcal{R}$ is equivariant, we can assume that $\mathrm{bn}(\alpha) \# P, Q$ without loss of generality. By Theorem 13.2, we have that $[\![P]\!] \xrightarrow{[\![\alpha]\!]} P''$ with $P'' \dot{\sim} [\![P']\!]$. Hence, since $[\![P]\!] \dot{\sim} [\![Q]\!]$ and $\mathrm{bn}(\alpha) \# Q$, there is a Q'' such that $[\![Q]\!] \xrightarrow{[\![\alpha]\!]} Q''$ and $Q'' \dot{\sim} P''$. By Theorem 13.4, there is Q' such that $Q \xrightarrow{\alpha} Q'$ and $[\![Q']\!] = Q''$. By transitivity, $[\![P']\!] \dot{\sim} [\![Q']\!]$.

Symmetrically, we show that $\mathcal{R} = \{(\mathbf{1}, [\![P]\!], [\![Q]\!]) : P \dot{\sim} Q\}$ is a bisimulation up to $\dot{\sim}$ in **PSPi**:

Static equivalence:

Vacuous since there's only one frame.

Symmetry:

By symmetry of $\dot{\sim}$

Simulation:

Here $[\![P]\!] \xrightarrow{\alpha'} P''$ and $P \dot{\sim} Q$. We proceed by case analysis on α':

1. If $\alpha' = \underline{\langle \tilde{x} \rangle}\, z$, then by Theorem 13.3 and a sufficiently fresh y, $P \xrightarrow{\tilde{x}(y)} P'$ where $[\![P'\{z/y\}]\!] = P''$. Since $P \dot{\sim} Q$, there exists Q' such that $Q \xrightarrow{\tilde{x}(y)} Q'$ and $P'\{z/y\} \dot{\sim} Q'\{z/y\}$. Hence, by Theorem 13.1, $[\![Q]\!] \xrightarrow{\langle \tilde{x} \rangle\, z} Q''$ where $Q'' \dot{\sim} [\![Q']\!][y := z] = [\![Q'\{z/y\}]\!]$. We have that $P'' = [\![P'\{z/y\}]\!] \mathcal{R} [\![Q'\{z/y\}]\!] \dot{\sim} Q''$, which suffices.

2. If α' is not an input, since $\mathcal{R}$ is equivariant, we can assume that $\mathrm{bn}(\alpha')\#P, Q$ without loss of generality. Since $[\![P]\!] \xrightarrow{\alpha'} P''$, by Theorem 13.4 we have that $P \xrightarrow{\alpha} P'$ where $[\![\alpha]\!] = \alpha'$ and $[\![P']\!] = P''$. Since $P \mathrel{\dot{\sim}} Q$, there is Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \mathrel{\dot{\sim}} Q'$. By Theorem 13.2, $[\![Q]\!] \xrightarrow{[\![\alpha]\!]} Q''$, where $Q'' \mathrel{\dot{\sim}} [\![Q']\!]$. Hence $P'' = [\![P']\!] \mathcal{R} [\![Q']\!] \mathrel{\dot{\sim}} Q''$, which suffices.

Extension of arbitrary assertion:
Vacuous since there's only one frame.

Lemma 8. *$[\![\cdot]\!]$ is surjective up to $\sim$ on the set of case-guarded processes, that is, for every case-guarded P there is a Q such that $[\![Q]\!] \sim P$.*

Proof. By induction on a well formed agent P.

case $\underline{\langle\widetilde{x}\rangle}(\lambda y)y.P'$:
It is valid to consider only this form, since $\{y\} \in \textsc{vars}(y)$. The IH is for some Q', $[\![Q']\!] \sim P'$. Let $Q = \widetilde{x}(y).Q'$. Then $[\![Q]\!] = \underline{\langle\widetilde{x}\rangle}(\lambda y)y.[\![Q']\!] \sim \underline{\langle\widetilde{x}\rangle}(\lambda y)y.P'$.

case $\overline{\langle\widetilde{x}\rangle}\, y.P'$:
From IH, we get for some Q', $[\![Q']\!] \sim P'$. Let $Q = \widetilde{x}\langle y\rangle.Q'$. Then $[\![Q]\!] = \overline{\langle\widetilde{x}\rangle}\, y.[\![Q']\!] \sim \overline{\langle\widetilde{x}\rangle}\, y.P'$.

case $P' \mid P''$:
From IH, for some Q', Q'', we have $[\![Q']\!] \sim P'$ and $[\![Q'']\!] \sim P''$. Let $Q = Q' \mid Q''$. Then $[\![Q]\!] = [\![Q']\!] \mid [\![Q'']\!] \sim P' \mid P''$.

case $(\nu x)P'$:
Let $Q = \nu x Q'$, then by induction hypothesis $[\![Q]\!] = (\nu x)[\![Q']\!] \sim (\nu x)P'$.

case $!P'$:
Let $Q = !Q'$ (Q' from IH). $[\![Q]\!] = ![\![Q']\!] \sim !P'$.

case 0:
Then $[\![\mathbf{0}]\!] = \mathbf{0} \sim \mathbf{0}$.

case $(\!|\mathbf{1}|\!)$:
Then $[\![\mathbf{0}]\!] = \mathbf{0} \sim (\!|\mathbf{1}|\!)$.

case case $\widetilde{\varphi} : \widetilde{P'}$:
For induction hypothesis $\mathrm{IH}_{\mathbf{case}}$, we have for every i there is Q'_i such that $[\![Q'_i]\!] \sim P'_i$. The proof goes by induction on the length of $\widetilde{\varphi}$.

base case:
Let $Q = \mathbf{0}$, then $[\![Q]\!] = \mathbf{0} \sim \mathbf{case}$.

induction step:
At this step, we get the following IH

$$[\![Q'']\!] \sim \mathbf{case}\ \varphi_1 : P_1\ [\!]\ \ldots\ [\!]\ \varphi_n : P_n$$

We need to show that there is some $[\![Q]\!]$ such that

$$[\![Q]\!] \sim \mathbf{case}\ \varphi_1 : P_1\ [\!]\ \ldots\ [\!]\ \varphi_n : P_n\ [\!]\ \varphi_{n+1} : P_{n+1} = P$$

First, we note that $\text{IH}_{\mathbf{case}}$ holds for every i and in particular $i = n+1$, thus we get $[\![Q'_{n+1}]\!] \sim P_{n+1}$. Second, we note that φ_{n+1} has two forms, thus we proceed by case analysis on φ_{n+1}.

case $\varphi_{n+1} = \bot$**:**
Let $Q = Q''$. Then

$$\begin{aligned}[\![Q]\!] &= [\![Q'']\!] \\ &\sim \mathbf{case}\ \varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n \\ &\sim \mathbf{case}\ \varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n \,[]\, \bot : P_{n+1}\end{aligned}$$

This case is concluded.

case $\varphi_{n+1} = \top$**:**
From the assumption, we know that P_{n+1} is of form $\alpha.P'_{n+1}$ and that $[\![Q'_{n+1}]\!] \sim \alpha.P'_{n+1}$. By investigating the construction of Q'_{n+1} we can conclude that $Q'_{n+1} = \alpha.Q''_{n+1}$ where $[\![Q''_{n+1}]\!] \sim P'_{n+1}$. The agent from IH Q'' is either $\mathbf{0}$, or prefixed agent, or a mixed sum.
In case $Q'' = \mathbf{0}$, let $Q = Q'_{n+1}$, then $[\![Q]\!] = [\![Q'_{n+1}]\!] \sim P$.
In case Q'' is prefixed agent, let $Q = Q'' + Q'_{n+1}$. Since Q'' and Q'_{n+1} are prefixed, Q is well formed. Then $[\![Q]\!] = \mathbf{case}\ \top : [\![Q'']\!] \,[]\, \top : [\![Q'_{n+1}]\!] \sim \mathbf{case}\ \varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n \,[]\, \top : P_{n+1}$.
In case Q'' is a sum, let $Q = Q'' + Q'_{n+1}$. Since Q'_{n+1} is guarded, Q is well formed. Then

$$\begin{aligned}[\![Q]\!] &= \mathbf{case}\ \top : [\![Q'']\!] \,[]\, \top : [\![Q'_{n+1}]\!] \\ &\sim \mathbf{case}\ \top : (\mathbf{case}\ \varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n) \\ &\quad []\, \top : [\![Q'_{n+1}]\!] \\ &\sim (\text{by Lemma 4}) \\ &\quad \mathbf{case}\ \varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n \\ &\quad []\, \top : [\![Q'_{n+1}]\!] \\ &\sim \mathbf{case}\ \varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n \\ &\quad []\, \top : P'_{n+1}\end{aligned}$$

This concludes the proof.

Lemma 9. *$[\![\cdot]\!]$ is injective, that is, for all P, Q, if $[\![P]\!] = [\![Q]\!]$ then $P = Q$.*

Proof. By induction on P and Q while inspecting all the possible cases.

Theorem 20. *$[\![\cdot]\!]$ is an isomorphism up to $\sim$ between ${}^{e}\pi$ and the case-guarded processes in* **PSPI**.

Proof. Directly follows from Lemma 9 and Lemma 8

D.4 Value-passing CCS

Lemma 10. *If P is a* **VPCCS** *process such that $P \xrightarrow{\overline{M}\,(\nu \widetilde{x})\,N} P''$ then $\widetilde{x} = \epsilon$*

Proof. By induction on the derivation of P'. Obvious in all cases except Open, where we derive a contradiction since only values can be transmitted yet only channels can be restricted - hence the name a is both a name and a value.

Strong operational correspondence:

Proof (of Theorem 15).

1. By induction on the derivation of P'.

   Act:
   We have that $\alpha.P \xrightarrow{\alpha} P$. Since $\alpha.P$ is in the range of $\widehat{\cdot}$, there must be x and v such that either $\alpha = \overline{x}(v)$ (for if α was an input, $\alpha.P$ would be outside the range of $\widehat{\cdot}$). The Out rule then admits the derivation $\overline{x}\ v.[\![P]\!] \xrightarrow{\overline{x}\ v} [\![P]\!]$

   Sum:
   There are two cases to consider: either $\Sigma_i P_i$ is the encoding of an input, or a summation.
   (a) If $\Sigma_i P_i = \Sigma_v x(v).P\{v/y\} = \widehat{x(y).P}$ we have that $\alpha = x(v)$. Then for each v, we can derive $\underline{x}(\lambda y).[\![P]\!] \xrightarrow{\underline{x}\ v} [\![P\{v/y\}]\!]$ using the In rule.
   (b) Otherwise, we have that $P_j \xrightarrow{\alpha} P'$ and by induction,

   $$[\![P_j]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$$

   The Case rule lets us derive

   $$\mathbf{case}\ \top : [\![P_1]\!]\ [\!]\ \cdots\ [\!]\ \top : P_i \xrightarrow{[\![\alpha]\!]} [\![P']\!]$$

   This suffices since $[\![\Sigma_i P_i]\!] = \mathbf{case}\ \top : [\![P_1]\!]\ [\!]\ \cdots\ [\!]\ \top : P_i$.

   Com1:
   Here $P \xrightarrow{\alpha} P'$ and by induction, $[\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$. The Par rule admits derivation of the transition $[\![P]\!] \mid [\![Q]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!] \mid [\![Q]\!]$, using Lemma 10 to discharge the freshness side condition.

   Com2:
   Symmetric to Com1.

   Com3:
   Here $P \xrightarrow{\alpha} P'$, $Q \xrightarrow{\overline{\alpha}} Q'$. Since α is in the range of $\widehat{\cdot}$, there are x and v such that $\alpha = x(v)$ and $\overline{\alpha} = \overline{x}(v)$ (or vice versa, in which case read the next sentence symmetrically). By the induction hypotheses, $[\![P]\!] \xrightarrow{\underline{x}\ v} [\![P']\!]$ and $[\![Q]\!] \xrightarrow{\overline{x}\ v} [\![Q']\!]$ - hence $[\![P]\!] \mid [\![Q]\!] \xrightarrow{\tau} [\![P']\!] \mid [\![Q']\!]$ by the Com rule, using Lemma 10 to discharge the freshness side condition.

   Res:
   Here $P \xrightarrow{\alpha} P'$ with $L\#\alpha$ - hence $\sigma(L)\#\alpha$. By induction, $[\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$. Then we use the Res rule $|L|$ times to derive $(\nu\sigma(L))[\![P]\!] \xrightarrow{[\![\alpha]\!]} (\nu\sigma(L))[\![P']\!]$.

Rep:

Here $P \mid !P \xrightarrow{\alpha} P'$. By induction, $[\![P]\!] \mid ![\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$, and by the REP rule, $![\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$

2. By induction on the derivation of P'.

In:

Here $\underline{x}(\lambda y)y.[\![P]\!] \xrightarrow{\underline{x}\ v} [\![P\{v/y\}]\!]$. We match this by deriving $\widehat{x(y).P} \xrightarrow{x(v)} \widehat{P}\{v/y\}$ using the ACT and SUM rules.

Out:

Here $\overline{x}\ v.[\![P]\!] \xrightarrow{\overline{x}\ v} [\![P]\!]$. We match this by deriving $\widehat{\overline{x}(v).P} \xrightarrow{\overline{x}(v)} \widehat{P}$ using the ACT rule.

Com:

Here $[\![P]\!] \xrightarrow{\overline{x}\ (\nu\tilde{y})\ v} P''$, $[\![Q]\!] \xrightarrow{\underline{x}\ v} Q''$. By Lemma 10, $\tilde{y} = \epsilon$, and by induction, $P \xrightarrow{\overline{x}(v)} P'$ and $Q \xrightarrow{x(v)} Q'$where $[\![P']\!] = P''$ and $[\![Q']\!] = Q''$. Using the COM3 rule we derive $P \mid Q \xrightarrow{\tau} P' \mid Q'$

Par:

Easy.

Case:

Our case statement can either be the encoding of either a summation or an **if** statement. We proceed by case analysis:

(a) Here $[\![P_j]\!] \xrightarrow{\alpha'} P''$. By induction, $P_j \xrightarrow{\alpha} P'$ where $[\![\alpha]\!] = \alpha'$. By SUM, $\Sigma_i P_i \xrightarrow{\alpha} P'$.

(b) Here $[\![P]\!] \xrightarrow{\alpha'} P''$ and $\mathbf{1} \vdash b$. By induction, $P \xrightarrow{\alpha} P'$ where $[\![\alpha]\!] = \alpha'$ and $[\![P']\!] = P''$. Since b evaluates to true, $\widehat{\textbf{if } b \textbf{ then } P} = \widehat{P}$ - hence $\textbf{if } b \textbf{ then } P \xrightarrow{\alpha} P'$.

Rep:

Easy.

Scope:

Here $[\![P]\!] \xrightarrow{\alpha'} P''$ with $x \sharp \alpha'$ and by induction, $P \xrightarrow{\alpha} P'$ where $\alpha' = [\![\alpha]\!]$ and $P'' = [\![P']\!]$. Hence we can derive $P \setminus \{x\} \xrightarrow{\alpha} P' \setminus \{x\}$ by the RES rule.

Open:

Opening is not possible.

Chapter 5

Negative premises in applied process calculi

Paper IV

Negative premises in applied process calculi

Johannes Åman Pohjola, Johannes Borgström, Joachim Parrow,
Palle Raabjerg, and Ioana Rodhe

Department of Information Technology, Uppsala University, Sweden

Abstract. We explore two applications of negative premises to increase the expressive power of psi-calculi: reliable broadcasts and priorities. Together, these can be used to model discrete time, which we illustrate with an example from automotive applications. The negative premises can be encoded by a two-level structural operational semantics without negative premises; we use this fact to prove the standard congruence and structural laws of bisimulation with Nominal Isabelle.

1 Introduction

A *negative premise* in a structural operational semantics (SOS) rule states that in order to derive a transition, the absence of another transition must be proven. Applications include priorities, deadlock detection and sequential composition: an event can only occur if a higher-priority event is *not* available, a deadlock is detected if another transition can *not* be derived, and $P; Q$ may proceed as Q iff P can *not* act. Process algebras with negative premises have been studied for more than 20 years. The main novel contribution of this paper is to use negative premises in a general framework of high-level applied process calculi, where it is easy to define highly specialised modelling languages for particular applications. We show how to capture phenomena such as priorities and reliable broadcasts, and demonstrate the applicability of our framework through a nontrivial example.

Two early studies of negative premises in structural operational semantics are Bloom et al. [4] and Groote [12]; van Glabbeek [28] gives a more recent exposition of the involved challenges. One obvious problem is that it is possible to give an inconsistent set of rules, where a process has a transition if and only if it does not. Groote's solution involves introducing the notion of a *stratification*. Formally, a stratification is a function S from transitions to some ordinal such that for every instance of a rule application that may occur in a derivation tree, the negative premises have smaller S-values than the conclusion, and the positive premises have no larger S-values than the conclusion. A corollary of the existence of a stratification is that there is no transition whose absence is a precondition for its presence. Transitions are constructed stratum by stratum in a bottom-up fashion, where negative premises on a given stratum are true when the corresponding positive premises are not provable on any of the lower strata.

All previous work on negative premises known to us apply only to basic process calculi where there are no high-level data structures or logics in the process

syntax. Psi-calculi [3] is a parametric framework for extensions of the pi-calculus, with arbitrary data structures and logical assertions for facts about data. In earlier papers we have shown how psi-calculi can capture the same phenomena as other proposed extensions of the pi-calculus such as the applied pi-calculus, the spi-calculus, the fusion calculus, the concurrent constraint pi-calculus, and calculi with polyadic communication channels or pattern matching. Psi-calculi can be even more general, for example by allowing structured channels, higher-order formalisms such as the lambda calculus for data structures, and predicate logic for assertions.

In this paper, we extend psi-calculi with two examples of negative premises. First, we introduce a reliable broadcast communication, where the possibility of message loss is prevented by a negative premise in the rule for execution of parallel processes. In other words, it is impossible for a message to miss a process that listens for it. Second, we introduce a priority system, where negative premises ensure that synchronisation over higher priority channels block actions of lower priority. In both cases, we gain generality by making the reliability and priority of channels a dynamic property which may depend on the current process environment. The precise nature of this dependency may vary between different uses of broadcasts and priorities, subject to some natural restrictions.

For reliable broadcast and priorities, the structural operational semantics with negative premises coincides with a two-layer formulation, where the negative premises in the top layer are expressed in terms of the bottom layer, and the bottom layer only uses positive premises — and hence is a formulation that formally has no negative premises. We use this approach when formalising our framework in Nominal Isabelle, allowing us to directly reuse earlier work on the topic by Bengtson [2]. We obtain machine-checked proofs that strong bisimulation satisfies the expected algebraic properties.

We evaluate our framework by showing that a notion of discrete time emerges as a special case of low-priority reliable broadcasts of clock signals. We demonstrate that instances of psi-calculi can capture the broadcast pi-calculus [10] and model the CAN protocol for bus arbitration in automotive vehicles [9].

2 Background on Psi-calculi

The following is a quick recapitulation of the psi-calculi framework. For an in-depth introduction with motivations and examples refer the reader to [3].

We assume a countably infinite set of atomic *names* $\mathcal{N}$ ranged over by $a, b, \ldots, z$. Intuitively, names will represent the symbols that can be scoped, and also represent symbols acting as variables in the sense that they can be subject to substitution. A *nominal set* [21] is a set equipped with a formal notion of what it means for a name a to occur in an element X of the set, written $a \in \mathrm{n}(X)$ (often pronounced as "a is in the support of X"). We write $a\#X$, pronounced "a is fresh for X", for $a \notin \mathrm{n}(X)$, and if A is a set of names we write $A\#X$ to mean $\forall a \in A \,.\, a\#X$. In the following $\tilde{a}$ means a finite sequence of names, $a_1, \ldots, a_n$. The empty sequence is written ϵ and the concatenation of $\tilde{a}$ and $\tilde{b}$ is written $\tilde{a}\tilde{b}$.

When occurring as an operand of a set operator, $\tilde{a}$ means the corresponding set of names $\{a_1, \ldots, a_n\}$. We also use sequences of other nominal sets in the same way. For names, we write $(\widetilde{a}\ \widetilde{b})$ for the *name swapping* that swaps each element of $\widetilde{a}$ with the corresponding element of $\widetilde{b}$; here it is implicit that $\widetilde{a}$ and $\widetilde{b}$ have the same length, and that the names in $\widetilde{a}$ (resp. $\widetilde{b}$) are pair-wise distinct.

A *nominal datatype* is a nominal set together with a set of functions on it. In particular we shall consider substitution functions that substitute elements for names. If X is an element of a datatype, $\tilde{a}$ is a sequence of names without duplicates and $\tilde{Y}$ is an equally long sequence of elements of possibly another datatype, the *substitution* $X[\tilde{a} := \tilde{Y}]$ is an element of the same datatype as X. Substitution is required to satisfy a law akin to alpha-conversion: if $\widetilde{b}\#X, \widetilde{a}$ then $X[\widetilde{a} := \widetilde{T}] = ((\widetilde{b}\ \widetilde{a}) \cdot X)[\widetilde{b} := \widetilde{T}]$.

A psi-calculus is defined by instantiating three nominal data types and four operators:

Definition 1 (Psi-calculus parameters). *A psi-calculus requires the three (not necessarily disjoint) nominal data types: the (data) terms* $\mathbf{T}$*, ranged over by* M, N*, the conditions* $\mathbf{C}$*, ranged over by* φ*, the assertions* $\mathbf{A}$*, ranged over by* Ψ*, and the four equivariant operators:*

$$
\begin{array}{ll}
\leftrightarrow \in \mathbf{T} \times \mathbf{T} \to \mathbf{C} & \textit{(Unicast) Channel Equivalence} \\
\otimes \in \mathbf{A} \times \mathbf{A} \to \mathbf{A} & \textit{Composition} \\
\mathbf{1} : \mathbf{A} & \textit{Unit} \\
\vdash \subseteq \mathbf{A} \times \mathbf{C} & \textit{Entailment}
\end{array}
$$

and substitution functions $[\widetilde{a} := \widetilde{M}]$, substituting terms for names, on each of $\mathbf{T}$, $\mathbf{C}$ and $\mathbf{A}$, where the substitution function on $\mathbf{T}$, in addition to the alpha-conversion-like law above, satisfies the following name preservation laws: if $\widetilde{a} \subseteq \mathsf{n}(M)$ and $b \in \mathsf{n}(\widetilde{N})$ then $b \in \mathsf{n}(M[\widetilde{a} := \widetilde{N}])$; and if $b \in \mathsf{n}(M)$ and $b\#\widetilde{a}, \widetilde{N}$ then $b \in \mathsf{n}(M[\widetilde{a} := \widetilde{N}])$.

The binary functions above will be written in infix. Thus, $M \leftrightarrow N$ is a condition, pronounced "M and N are channel equivalent". We write $\Psi \vdash \varphi$, "Ψ entails φ", for $(\Psi, \varphi) \in \vdash$, and if Ψ and Ψ' are assertions then so is $\Psi \otimes \Psi'$.

We say that two assertions are equivalent, written $\Psi \simeq \Psi'$ if they entail the same conditions, i.e. for all φ we have that $\Psi \vdash \varphi \Leftrightarrow \Psi' \vdash \varphi$. We impose certain requisites on the sets and operators. In brief, channel equivalence must be symmetric and transitive, $\otimes$ must be compositional with regard to $\simeq$, and the assertions with $(\otimes, \mathbf{1})$ form an abelian monoid modulo $\simeq$.

A *frame* F can intuitively be thought of as an assertion with local names: it is of the form $(\nu\widetilde{b})\Psi$ where $\widetilde{b}$ is a sequence of names that bind into the assertion Ψ. We use F, G to range over frames. We overload Ψ to also mean the frame $(\nu\epsilon)\Psi$ and $\otimes$ to composition on frames defined by $(\nu\widetilde{b}_1)\Psi_1 \otimes (\nu\widetilde{b}_2)\Psi_2 = (\nu\widetilde{b}_1\widetilde{b}_2)(\Psi_1 \otimes \Psi_2)$ where $\widetilde{b}_1\#\widetilde{b}_2, \Psi_2$ and vice versa. We write $\Psi \otimes F$ to mean $(\nu\epsilon)\Psi \otimes F$, and $(\nu c)((\nu\widetilde{b})\Psi)$ for $(\nu c\widetilde{b})\Psi$.

Alpha equivalent frames are identified. We define $F \vdash \varphi$ to mean that there exists an alpha variant $(\nu\widetilde{b})\Psi$ of F such that $\widetilde{b}\#\varphi$ and $\Psi \vdash \varphi$. We also de-

fine $F \simeq G$ to mean that for all φ it holds that $F \vdash \varphi$ iff $G \vdash \varphi$. Intuitively a condition is entailed by a frame if it is entailed by the assertion and does not contain any names bound by the frame, and two frames are equivalent if they entail the same conditions.

Definition 2 (Psi-calculus agents). *Given psi-calculus parameters as in Definition 1, the* agents*, ranged over by $P, Q, \ldots$, are of the following forms.*

$\mathbf{0}$	Nil
$\overline{M}N\,.\,P$	Output
$\underline{M}(\lambda\widetilde{x})N\,.\,P$	Input
$\mathbf{case}\ \varphi_1 : P_1 \,[]\, \cdots \,[]\, \varphi_n : P_n$	Case
$(\nu a)P$	Restriction
$P \mid Q$	Parallel
$!P$	Replication
$(\!\lvert \Psi \rvert\!)$	Assertion

Restriction binds a in P and Input binds $\widetilde{x}$ in both N and P. We identify alpha equivalent agents. An occurrence of a subterm in an agent is guarded *if it is a proper subterm of a Prefix form. An agent is* assertion guarded *if it contains no unguarded Assertions. An agent is* well-formed *if in $\underline{M}(\lambda\widetilde{x})N.P$ it holds that $\widetilde{x} \subseteq \mathrm{n}(N)$ is a sequence without duplicates, that in a replication $!P$ the agent P is assertion guarded, and that in $\mathbf{case}\ \varphi_1 : P_1 \,[]\, \cdots \,[]\, \varphi_n : P_n$ the agents P_i are assertion guarded.*

The agent $\mathbf{case}\ \varphi_1 : P_1 \,[]\, \cdots \,[]\, \varphi_n : P_n$ is sometimes abbreviated as $\mathbf{case}\ \widetilde{\varphi} : \widetilde{P}$, or if $n = 1$ as $\mathbf{if}\ \varphi_1\ \mathbf{then}\ P_1$. Input subjects are underlined to facilitate parsing of complicated expressions; in simple cases we often omit the underline. We sometimes write $\underline{M}(x).P$ for $\underline{M}(\lambda x)x.P$. From this point on, we only consider well-formed agents.

The *frame* $\mathcal{F}(P)$ *of an agent* P is defined inductively as follows:

$$\mathcal{F}(\underline{M}(\lambda\widetilde{x})N\,.\,P) = \mathcal{F}(\overline{M}\,N\,.\,P) = \mathcal{F}(\mathbf{0}) = \mathcal{F}(\mathbf{case}\ \widetilde{\varphi} : \widetilde{P}) = \mathcal{F}(!P) = \mathbf{1}$$
$$\mathcal{F}(\!(\Psi)\!) = (\nu\epsilon)\Psi \qquad \mathcal{F}(P \mid Q) = \mathcal{F}(P) \otimes \mathcal{F}(Q) \qquad \mathcal{F}((\nu b)P) = (\nu b)\mathcal{F}(P)$$

The *actions* ranged over by α, β are of the following three kinds: *Output* $\overline{M}(\nu\tilde{a})N$ where $\tilde{a} \subseteq \mathrm{n}(N)$, *input* $\underline{M}\,N$, where $\tilde{a} \subseteq \mathrm{n}(N)$, and *silent* $\tau : p$. Here we refer to M as the *subject* and N as the *object*. We define $\mathrm{bn}(\overline{M}(\nu\tilde{a})N) = \mathrm{bn}(!\overline{M}\,(\nu\tilde{a})N) = \tilde{a}$, and $\mathrm{bn}(\alpha) = \emptyset$ if α is an input, broadcast input or $\tau : p$. We also define $\mathrm{n}(\tau : p) = \emptyset$ and $\mathrm{n}(\alpha) = \mathrm{n}(M) \cup \mathrm{n}(N)$ for the input and output actions. As in the pi-calculus, the output $\overline{M}(\nu\tilde{a})N$ represents an action sending N along M and opening the scopes of the names $\tilde{a}$. Note in particular that the support of this action includes $\tilde{a}$. Thus $\overline{M}(\nu a)a$ and $\overline{M}(\nu b)b$ are different actions.

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

$$\textsc{Case}\ \frac{\Psi \rhd P_i \xrightarrow{\alpha} P' \qquad \Psi \vdash \varphi_i}{\Psi \rhd \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \xrightarrow{\alpha} P'} \qquad \textsc{Par}\ \frac{\Psi_Q \otimes \Psi \rhd P \xrightarrow{\alpha} P'}{\Psi \rhd P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \mathrm{bn}(\alpha)\#Q$$

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

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

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

Table 1. Structured operational semantics. Symmetric versions of Com and Par are elided. In the rule Com we assume that $\mathcal{F}(P) = (\nu\widetilde{b}_P)\Psi_P$ and $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$ where $\widetilde{b}_P$ is fresh for all of $\Psi, \widetilde{b}_Q, Q, M$ and P, and that $\widetilde{b}_Q$ is similarly fresh. In the rule Par we assume that $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$ 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.

Definition 3 (Transitions). *A* transition *is written $\Psi \rhd P \xrightarrow{\alpha} P'$, meaning that in the environment Ψ the well-formed agent P can do an α to become P'. The transitions are defined inductively in Table 1. We write $P \xrightarrow{\alpha} P'$ without an assertion to mean $\mathbf{1} \rhd P \xrightarrow{\alpha} P'$, and $\Psi \rhd P \xrightarrow{\alpha}$ to mean $\exists P'.\Psi \rhd P \xrightarrow{\alpha} P'$, and $(\nu\widetilde{c})\Psi \rhd P \xrightarrow{\alpha} P'$ to mean $\widetilde{c}\#\,\mathrm{subject}(\alpha), P$ and $\Psi \rhd P \xrightarrow{\alpha}$.*

We will sometimes write $F \rhd P \xrightarrow{\alpha} P'$ to mean that there exists $\widetilde{b}_F$ and Ψ_F such that $F = (\nu\widetilde{b}_F)\Psi_F$ and $\Psi_F \rhd P \xrightarrow{\alpha} P'$ where $\widetilde{b}_F$ are fresh in the subject of α and in P.

Agents, frames and transitions are identified by alpha equivalence. In a transition the names in $\mathrm{bn}(\alpha)$ bind into both the action object and the derivative, therefore $\mathrm{bn}(\alpha)$ is in the support of α but not in the support of the transition. This means that the bound names can be chosen fresh, substituting each occurrence in both the object and the derivative.

Definition 4 (Strong bisimulation). *A* strong bisimulation $\mathcal{R}$ *is a ternary relation on assertions and pairs of agents such that $\mathcal{R}(\Psi, P, Q)$ implies*

1. *Static equivalence: $\Psi \otimes \mathcal{F}(P) \simeq \Psi \otimes \mathcal{F}(Q)$; and*

2. *Symmetry:* $\mathcal{R}(\Psi, Q, P)$*; and*
3. *Extension of arbitrary assertion:* $\forall \Psi'.\ \mathcal{R}(\Psi \otimes \Psi', P, Q)$*; and*
4. *Simulation: for all* α, P' *such that* $\Psi \rhd P \xrightarrow{\alpha} P'$ *and* $\mathrm{bn}(\alpha)\#\Psi, Q$*, there exists* Q' *such that* $\Psi \rhd Q \xrightarrow{\alpha} Q'$ *and* $\mathcal{R}(\Psi, P', Q')$*.*

We define $P \mathrel{\dot{\sim}}_\Psi Q$ *to mean that there exists a bisimulation* $\mathcal{R}$ *such that* $\mathcal{R}(\Psi, P, Q)$*, and write* $\mathrel{\dot{\sim}}$ *for* $\mathrel{\dot{\sim}}_{\mathbf{1}}$*.*

Definition 5 (Strong congruence). *We define* $P \sim_\Psi Q$ *to mean that for all substitution sequences* σ*,* $P\sigma \mathrel{\dot{\sim}}_\Psi Q\sigma$ *holds. We write* $P \sim Q$ *to mean* $P \sim_\Psi Q$*.*

3 Reliable Broadcast Psi-calculi

Broadcast psi-calculi [6] is an extension of psi-calculi for synchronous unreliable broadcast. A broadcast output action is written $\overline{!K}\ (\nu\widetilde{a})N$, meaning that N with bound names $\widetilde{a}$ is sent along a broadcast channel represented by K. The corresponding (early) broadcast input action is written $\underline{?K}\ N$. The predicates $\mathrel{\dot{\prec}}$ and $\mathrel{\dot{\succ}}$ regulate how prefix subjects are related to broadcast channels: If $M \mathrel{\dot{\prec}} K$ then an output prefix with subject M will give rise to an action on the broadcast channel K, and similarly $\mathrel{\dot{\succ}}$ is used with broadcast input. For technical reasons related to scope extension, no broadcast channel may have greater support than the prefix subjects connected to it. A special case is to declare $K \mathrel{\dot{\prec}} K$ and $K \mathrel{\dot{\succ}} K$ for all broadcast channels K; these channels are then represented by themselves in the process syntax.

The process syntax is the same as for ordinary psi-calculi, but the transition relation is extended with rules to accommodate broadcast communication, as shown in Table 2. The main difference is that broadcast communication does not result in a silent action. To accommodate multiple receivers, it instead yields a broadcast output action which may be picked up by others: $\Psi \rhd \overline{K}\,N \,.\, P \mid \underline{K}(\lambda\widetilde{x})X \,.\, Q \xrightarrow{\overline{!K}\,N} P \mid Q[\widetilde{x} := \widetilde{L}]$, assuming $N = X[\widetilde{x} := \widetilde{L}]$ and $\Psi \vdash K \mathrel{\dot{\prec}} K$ and $\Psi \vdash K \mathrel{\dot{\succ}} K$. An extensive treatment of broadcast psi-calculi including motivations and examples is in [6].

Here broadcast messages can be non-deterministically lost, a behaviour which is appropriate for the intended application area of wireless networks. However, if we wish to study broadcast communication on a shared bus, message loss is not a concern: any component that listens when a message is sent will receive it.

In this section we now define reliable broadcast psi-calculi, a conservative extension of broadcast psi-calculi where negative premises in the Par rule for parallel composition are used to ensure that messages cannot be lost.

One new psi-calculus parameter is added: the equivariant operator $\texttt{reliable} : \mathbf{T} \to \mathbf{C}$. Intuitively, if $\Psi \vdash \texttt{reliable}(K)$, then broadcasts over the channel K in the environment Ψ will reach all of its potential recipients. We will refer to such broadcasts as *reliable* broadcasts. Hence, it is possible to have both reliable and unreliable broadcast channels in the calculus, and even have channels whose reliability varies dynamically depending on the environment.

$$\textsc{BrOut}\ \frac{\Psi \vdash M \dot{\prec} K}{\Psi \rhd \overline{M}\,N\,.\,P \xrightarrow{\overline{!K}\,N} P} \qquad \textsc{BrIn}\ \frac{\Psi \vdash K \dot{\succ} M}{\Psi \rhd \underline{M}(\lambda\widetilde{y})N\,.\,P \xrightarrow{\underline{?K}\,N[\widetilde{y}:=\widetilde{L}]} P[\widetilde{y}:=\widetilde{L}]}$$

$$\textsc{BrMerge}\ \frac{\Psi_Q \otimes \Psi \rhd P \xrightarrow{\underline{?K}\,N} P' \qquad \Psi_P \otimes \Psi \rhd Q \xrightarrow{\underline{?K}\,N} Q'}{\Psi \rhd P \mid Q \xrightarrow{\underline{?K}\,N} P' \mid Q'}$$

$$\textsc{BrCom}\ \frac{\Psi_Q \otimes \Psi \rhd P \xrightarrow{\overline{!K}\,(\nu\widetilde{a})N} P' \qquad \Psi_P \otimes \Psi \rhd Q \xrightarrow{\underline{?K}\,N} Q'}{\Psi \rhd P \mid Q \xrightarrow{\overline{!K}\,(\nu\widetilde{a})N} P' \mid Q'}\ \widetilde{a}\#Q$$

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

$$\textsc{BrClose}\ \frac{\Psi \rhd P \xrightarrow{\overline{!K}\,(\nu\widetilde{a})N} P'}{\Psi \rhd (\nu b)P \xrightarrow{\tau:p} (\nu b)(\nu\widetilde{a})P'}\ \begin{array}{l} b \in \mathsf{n}(K) \\ b\#\Psi \end{array}$$

Table 2. Additional rules for broadcast psi-calculi. A symmetric version of BrCom is elided. In rules BrCom and BrMerge we assume that $\mathcal{F}(P) = (\nu\widetilde{b}_P)\Psi_P$ and $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$ where $\widetilde{b}_P$ is fresh for $P, \widetilde{b}_Q, Q, K$ and Ψ, and that $\widetilde{b}_Q$ is fresh for $Q, \widetilde{b}_P, P, K$ and Ψ. In BrOpen the expression $\tilde{a} \cup \{b\}$ means the sequence $\tilde{a}$ with b inserted anywhere.

The effects on the semantics are on the Par rule that says, roughly, that any process can always act alone. Eliding the assertions that define the environment, Par means that if $P \xrightarrow{\alpha} P'$ then $P \mid Q \xrightarrow{\alpha} P' \mid Q$. If α is a reliable broadcast that can be received by Q this rule no longer applies. In that case Q must participate in a communication through another rule, involving both P and Q. In order to prevent the Par rule from being applicable in that case we introduce a predicate ADMIT in its premise. Intuitively, $\mathrm{ADMIT}(\alpha, F, Q)$ is true if a process in parallel to Q may perform the action α in frame F without Q having to participate in the action. In other words, it is true unless α is a reliable broadcast that Q can receive in frame F.

$$\mathrm{ADMIT}(\alpha, F, Q) = \begin{cases} \textit{True} \text{ if } \alpha \text{ is not a broadcast action} \\ \neg(F \otimes \mathcal{F}(Q) \vdash \mathtt{reliable}(M) \wedge F \rhd Q \xrightarrow{\underline{?M}\,N}) \\ \qquad \text{if } \alpha = \underline{?M}\,N \text{ or } \alpha = \overline{!M}\,(\nu\widetilde{a})N. \end{cases}$$

The only change to the semantics is that the ADMIT predicate is added as a side-condition to the Par rule, as follows. A symmetric version is elided, and to write out the rule in full with the assertions we assume that $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$ where $\widetilde{b}_Q$ is fresh for Ψ, P and α.

$$\textsc{Par}\ \frac{\Psi_Q \otimes \Psi \rhd P \xrightarrow{\alpha} P' \quad \text{ADMIT}(\alpha, \Psi \otimes \mathcal{F}(P), Q)}{\Psi \rhd P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \text{bn}(\alpha)\#Q$$

Note that the premise of the Par rule is a negative premise since ADMIT requires the absence of transitions.

Finally, a syntactic restriction on processes is imposed: in the agent $!P$, we require that P must be *reliable reception guarded*, which intuitively means that there is no substitution and environment that can make an unguarded prefix into a reliable broadcast input. Formally, an agent is reliable reception guarded if each input is either guarded or its subject M satisfies the following, where σ ranges over substitution sequences:

$$\forall \Psi, N, \sigma. \neg(\Psi \vdash N \mathrel{\dot{\succ}} M\sigma \wedge \Psi \vdash \texttt{reliable}(N))$$

This criterion is imposed in order to ensure that the ruleset is consistent. As an example to demonstrate why it is necessary, consider an agent P with an unguarded reliable broadcast input. The only SOS rule for $!P$ says that it should behave as $P \mid !P$. Now assume that $!P$ has no broadcast input, then by the new Par rule $P \mid !P$ and hence also $!P$ has a broadcast input, leading to a contradiction. On the other hand, if $!P$ has a broadcast input then a shorter inference of that action must have been made from $P \mid !P$; this means that it must have been inferred from P and Par, an impossibility if $!P$ has that action. In other words, it seems that $!P$ has the broadcast action if and only if it has not!

The closure under substitutions in the criterion is needed since $!P$ may occur under an input prefix which, when executing, will give rise to a substitution. Formally, the rule system on agents which are not reliable reception guarded is not stratifiable and thus does not give rise to a meaningful semantics. We do not believe that reliable broadcast input under replication is meaningful — we cannot conceive of a semantics which allows it, while also being faithful to the standard algebraic properties of replication and our intuitive understanding of reliability.

Example 6 (Monadic $b\pi$ calculus). $b\pi$ [10] is an adaptation of the pi-calculus to use broadcast communication in place of point-to-point communication. Here we show a psi-calculus **BPI** that corresponds to monadic $b\pi$. For convenience, we consider only the finite subcalculus, without the τ prefix (since it can be encoded as $(\nu x)\overline{x}\,x$).

BPI	
$\mathbf{T} = \mathcal{N}$	$a \leftrightarrow b = \bot$
$\mathbf{C} = \{\top, \bot\} \cup \{a \doteq b \mid a, b \in \mathcal{N}\} \cup \{\neg\phi \mid \phi \in \mathbf{C}\}$	$a \mathrel{\dot{\succ}} b = a \mathrel{\dot{\prec}} b = a \doteq b$
$\mathbf{A} = \{\mathbf{1}\}$	$\mathbf{1} \vdash \top$
$\texttt{reliable}(M) = \top$	$\mathbf{1} \vdash a \doteq a$
	$\mathbf{1} \vdash \neg\phi$ iff $\neg\mathbf{1} \vdash \phi$

We define representation functions $[\![\cdot]\!]$ from $b\pi$ agents and actions to **BPI** agents and actions, respectively:

Definition 7 ($b\pi$ to PPi).

Agents:

$$\begin{aligned}
[\![P+Q]\!] &= \mathbf{case}\ \top : [\![P]\!]\ []\ \top : [\![Q]\!] \\
[\![\langle x=y\rangle P, Q]\!] &= \mathbf{case}\ x \doteq y : [\![P]\!]\ []\ \neg x \doteq y : [\![Q]\!] \\
[\![x(y).P]\!] &= \underline{x}(\lambda y) y.[\![P]\!] \\
[\![\overline{x}y.P]\!] &= \overline{x}\langle y\rangle.[\![P]\!] \\
[\![\mathrm{nil}]\!] &= \mathbf{0} \\
[\![P \mid Q]\!] &= [\![P]\!] \mid [\![Q]\!] \\
[\![\nu x P]\!] &= (\nu x)[\![P]\!]
\end{aligned}$$

Actions:

$$\begin{aligned}
[\![\nu x \overline{a} x]\!] &= \overline{!a}\ (\nu x) x \\
[\![\overline{a} x]\!] &= \overline{!a}\ x \\
[\![x\langle y\rangle]\!] &= \underline{?x}\ y \\
[\![\tau]\!] &= \tau
\end{aligned}$$

Lemma 8. *If* $P \xrightarrow{a:}$ *then* $\mathrm{ADMIT}(\underline{?a}\, x, \mathbf{1}, [\![P]\!])$.

Proof. By induction on the derivation of $P \xrightarrow{a:}$.

Lemma 9. *If* $\mathrm{ADMIT}(\underline{?a}\, x, \mathbf{1}, [\![P]\!])$ *then* $P \xrightarrow{a:}$.

Proof. By induction on P.

Theorem 10 (Operational correspondence).

1. *If* $P \xrightarrow{\alpha} P'$ *then* $\mathbf{1} \rhd [\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$.
2. *If* $\mathbf{1} \rhd [\![P]\!] \xrightarrow{[\![\alpha]\!]} P'$ *and* $\mathrm{bn}(\alpha) \# P$, *then there exists* P'' *such that* $P \xrightarrow{\alpha} P''$ *and* $P' = [\![P'']\!]$.

Proof. By induction on the derivation of the transitions of P and $[\![P]\!]$, respectively.

There can be no similar correspondence in the polyadic case, since structural equivalence (swapping of outermost restrictions, in particular) is not sound for strong labelled bisimilarity in $b\pi$ (in contradiction to [10, Lemma 34(i)]): $\nu x\, \nu y\, \overline{a}\, x, y.\mathrm{nil} \xrightarrow{\nu x\, \nu y\, \overline{a}\, x, y}$ nil, but the only transition of $\nu y\, \nu x\, \overline{a}\, x, y.\mathrm{nil}$ is $\nu y\, \nu x\, \overline{a}\, x, y.\mathrm{nil} \xrightarrow{\nu y\, \nu x\, \overline{a}\, x, y}$ nil, which has a different label.

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

$$\textsc{Par}\ \frac{\Psi_Q \otimes \Psi \rhd P \xrightarrow{\alpha} P'}{\Psi \rhd P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \begin{array}{l}\mathrm{HIGHEST}(\alpha, \Psi, P \mid Q)\\ \mathrm{bn}(\alpha)\#Q\end{array}$$

$$\textsc{Com}\ \frac{\begin{array}{c}\Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrows K \qquad \Psi \vdash \mathtt{prio}(M) = p\\ \Psi_Q \otimes \Psi \rhd P \xrightarrow{\overline{M}(\nu \widetilde{a})N} P'\\ \Psi_P \otimes \Psi \rhd Q \xrightarrow{K\ N} Q'\end{array}}{\Psi \rhd P \mid Q \xrightarrow{\tau:p} (\nu \widetilde{a})(P' \mid Q')}\ \begin{array}{l}\mathrm{HIGHEST}(\tau : p, \Psi, P \mid Q)\\ \widetilde{a}\#Q\end{array}$$

Table 3. Structured operational semantics with priorities. Symmetric versions of Com and Par are elided. In the rule Com we assume that $\mathcal{F}(P) = (\nu \widetilde{b}_P)\Psi_P$ and $\mathcal{F}(Q) = (\nu \widetilde{b}_Q)\Psi_Q$ where $\widetilde{b}_P$ is fresh for all of $\Psi, \widetilde{b}_Q, Q, M$ and P, and that $\widetilde{b}_Q$ is similarly fresh. In the rule Par we assume that $\mathcal{F}(Q) = (\nu \widetilde{b}_Q)\Psi_Q$ where $\widetilde{b}_Q$ is fresh for Ψ, P and α.

4 Psi-calculi with priorities

In this section, we extend psi-calculi with a priority system. The idea is that for each communication channel M, the assertion environment associates to it a priority level $n \in \mathbb{N}$, where lower values of n denote *higher* priority levels. A process may only interact on M if for all $m < n$, no internal actions at priority m are available. As the assertion environment changes, priority levels may change depending on the particulars of the psi-calculus under consideration.

To achieve this, we add the equivariant operator $\mathtt{has\_prio} \in \mathbf{T} \times \mathbb{N} \to \mathbf{C}$, intuitively $F \vdash \mathtt{has\_prio}(M, p)$ means that the frame F assigns priority p to the term M. If M is a communication channel, i.e., for some K it holds that $\Psi \vdash M \leftrightarrows K$, then we require this p to be unique and to be invariant under channel equivalence. We write $\mathtt{prio}(M) = p$ for $\mathtt{has\_prio}(M, p)$. We tag the silent action with an explicit priority, as in $\tau : p$. We notate the priority of an action α in frame F as $\mathtt{PRIO}(F, \alpha)$, defined to be p if $\alpha = \tau : p$ or if M is the subject of α and $F \vdash \mathtt{prio}(M) = p$.

To define a transition system with priorities, we use a predicate $\mathrm{HIGHEST}(\alpha, \Psi, P)$, that intuitively states that P has no transitions that block α in the current frame. Transitions that block α in frame F are internal actions with higher priority:

$$\mathrm{HIGHEST}(\alpha, \Psi, P) := \neg(\Psi \rhd P \xrightarrow{\tau:n} \wedge\, n < \mathtt{PRIO}(\Psi \otimes \mathcal{F}(P), \alpha))$$

To enforce that transitions respect the priority scheme, HIGHEST is added as a side condition to relevant rules, as in Table 3.

Since HIGHEST requires absence of transitions, it constitutes a negative premise.

Example 11 (Pi-calculus with dynamic priorities). We here define a calculus based on the pi-calculus, with the addition that channels may have one of two priority levels: 0 (high) and 1 (low). Further, it is possible to flip the priority of a channel a dynamically by asserting $\{a\}$. For an example, suppose we want to enforce a fairness scheme such that synchronisations on two channels x and y are guaranteed to interleave. This can be achieved by swapping the priorities of x and y after every such synchronisation, as in the following derivation sequence, where $P_z = (\!|\{z\}|\!) \mid !\overline{x}.(\!|\{x,y\}|\!) \mid !\overline{y}.(\!|\{x,y\}|\!)$.

$$\begin{aligned}\mathbf{1} \rhd\ & P_y \mid x\,.\,x\,.\,x \mid y \xrightarrow{\tau:0} P_x \mid x\,.\,x \mid y \xrightarrow{\tau:0} \\ & P_y \mid x\,.\,x \xrightarrow{\tau:0} P_x \mid x \xrightarrow{\tau:1} P_y\end{aligned}$$

Note that the above τ sequence is the only possible τ sequence — as long as both x and y are available they are guaranteed to be consumed alternatingly. Formally, we define this psi-calculus by $\mathbf{T} = \mathcal{N}$, $\mathbf{C} = \{x = y : x, y \in \mathbf{T}\} \cup \{\mathtt{prio}(M) = n : M \in \mathbf{T} \wedge n \in \mathbb{N}\}$ and $\mathbf{A}$ is the finite sets of names. Moreover, let $\mathbf{1}$ be the empty set and $A \otimes B = (A \cup B) - (A \cap B)$. Entailment is defined so that $\Psi \vdash x = y$ iff $x = y$, $\Psi \vdash \mathtt{prio}(x) = 1$ iff $x \in \Psi$, and $\Psi \vdash \mathtt{prio}(x) = 0$ iff $x \notin \Psi$. Finally, we let channel equivalence be syntactic equality on names.

5 Examples of Reliable Broadcast Priorities

In this section, we combine the extensions for priority and broadcast presented in Section 3 and Section 4 into a single calculus. In order to integrate the two, the following adaptions have to be made. First, high-priority broadcast actions are considered to block lower priority actions, in the same way as high-priority tau actions. The intuition is that both broadcast output and tau are independent actions that can occur without a communication partner. Formally, we let β range over broadcast outputs and τ actions, and amend the definition of HIGHEST as follows:

$$\begin{aligned}\mathrm{HIGHEST}(\alpha, \Psi, P) := \neg\exists\beta.\ & \Psi \rhd P \xrightarrow{\beta} \\ & \wedge \\ & \mathtt{PRIO}\,(\Psi \otimes \mathcal{F}(P), \beta) < \mathtt{PRIO}\,(\Psi \otimes \mathcal{F}(P), \alpha)\end{aligned}$$

This predicate is added as a side condition to the rules for broadcast communication, in the same manner as the rules for unicast communication.

Second, we require that broadcast channels have a well defined priority, in the same way as unicast channels. In other words we require that if $\Psi \vdash M \dot{\prec} K$ or $\Psi \vdash K \dot{\succ} M$, then there is a unique p such that $\Psi \vdash \mathtt{prio}(K) = p$. If $\Psi \vdash M \dot{\prec} K$ we further require that there is a unique p such that $\Psi \vdash \mathtt{prio}(M) = p$, that if $\Psi \vdash \mathtt{prio}(K) = p'$ holds then $p \leq p'$, and that if $\Psi \vdash M \dot{\prec} L$ then $\Psi \vdash \mathtt{prio}(L) = p'$. These requirements ensure that no actions that may arise from an output prefix will block another action from the same prefix. Note

that input prefixes, unlike output prefixes, may be simultaneously connected to broadcast channels with different priorities. Hence, a listener need not care about the priority of messages it receives.

5.1 Discrete Time

There exist several versions of process algebras with discrete time; a common approach is to introduce a special kind of action σ to represent that time passes [15,18]. We claim that in psi-calculi, σ is merely a special case of a reliable broadcast of a clock pulse whose priority is lower than all other actions.

As an example, in the timed broadcasting process calculus aTCWS, due to Macedonio and Merro [18] the intuition is that when no process currently wishes to send anything, a timeout event is propagated through the system. Input prefixes are of the form $\lfloor ?(x).P \rfloor Q$ - this agent may receive a broadcast carrying the message w and evolve to $P\{w/x\}$, or a σ and evolve to Q.

We can easily formulate a psi-calculus to accommodate this. Let σ be a new term such that in all assertion environments $\sigma \mathrel{\dot\prec} \sigma \mathrel{\dot\succ} \sigma$, $\mathtt{reliable}(\sigma)$ and $\forall \Psi, n, n', M. M \neq \sigma \ \wedge \ \Psi \vdash \mathtt{has\_prio}(M, n) \wedge \Psi \vdash \mathtt{has\_prio}(\sigma, n') \Rightarrow n' \geq n$ holds. Let the system contain a timeout factory process $!\overline{\sigma}$. The aTCWS input prefix $\lfloor ?(x).P \rfloor Q$ can then be represented as $?(x).P + \underline{\sigma}.Q$.

As another example consider a more general timeout operator. Nicollin and Sifakis define in their survey [20] a *timeout* for P (the body), Q (the exception), d (the integer time delay) to behave as P if an initial action of P is performed within time d, otherwise it behaves as Q after time d. Varieties of this operator exist in many process algebras; in e.g. TPCCS [14] it is notated $P \triangleright_d Q$.

In a psi-calculus we can represent timeout as follows. Extend the terms to contain $\mathtt{raise}_a$ and $\mathtt{abort}_a$ with support $\{a\}$ for all names a, and extend the broadcast relations such that for all a, Ψ, $\mathtt{raise}_a \mathrel{\dot\prec} \mathtt{raise}_a \mathrel{\dot\succ} \mathtt{raise}_a$, $\mathtt{reliable}(\mathtt{raise}_a)$, and similarly for $\mathtt{abort}_a$, and with no other channel equivalences on these terms. The idea in the following representation is that an action on $\overline{\mathtt{raise}_a}$ will trigger the exception Q, and that P can at any prior time abort the timeout by performing an action $\overline{\mathtt{abort}_b}$, where a and b are distinct fresh names. Let $\mathtt{raise}_a$ carry a priority higher than any other used in P or Q.

Let a timer T^d for any integer d be defined by $T^0 = \overline{\mathtt{raise}_a}.\mathbf{0}$ and $T^d = \underline{\sigma}.T^{d-1} + \underline{\mathtt{abort}_b}.\mathbf{0}$ for $d > 0$. The timeout $P \triangleright_d Q$ then corresponds to

$$(\nu\, a, b)(T^d \mid ((P + \underline{\mathtt{raise}_a}.\mathbf{0}) \mid (\underline{\mathtt{abort}_b}.\mathbf{0} + \underline{\mathtt{raise}_a}.Q)))$$

To examine its behaviour first assume $d > 0$. If $P \xrightarrow{\sigma} P'$, i.e, if P can let time pass to become P', then $P \triangleright_d Q \xrightarrow{\sigma} P' \triangleright_{d-1} Q$. If P instead starts acting by aborting the timeout with $P \xrightarrow{\overline{\mathtt{abort}_b}} P'$ then $P \triangleright_d Q \xrightarrow{\tau} (\nu\, a, b)(\mathbf{0} \mid P' \mid \mathbf{0})$ which will behave as P' since a, b are chosen fresh. If $d = 0$ the system becomes

$$(\nu\, a, b)(\overline{\mathtt{raise}_a}.\mathbf{0} \mid ((P + \underline{\mathtt{raise}_a}.\mathbf{0}) \mid (\underline{\mathtt{abort}_b}.\mathbf{0} + \underline{\mathtt{raise}_a}.Q)))$$

which has a broadcast along $\mathtt{raise}_a$ to become $(\nu\, a, b)(\mathbf{0} \mid \mathbf{0} \mid Q)$, which will behave as Q.

There are several other operators related to time which may merit investigation in the same way, and it would be interesting to explore their algebraic properties. For example, it is reasonable to expect that timeout is associative.

5.2 Controller Area Network Bus

The CAN bus [9] is a communication bus designed for system communication in vehicles. It has also been used in other areas than automobiles, such as automation and aerospace. CAN is a message based protocol where all messages are broadcast to everyone. Each node can transmit at any time on the bus. When several nodes want to transmit at the same time, an arbitration-free algorithm based on priorities is used. The identity of each node is an integer, which is used as the priority of the node. A lower identity has a higher priority.

Using a reliable broadcast psi-calculus, we model the protocol used for deciding who has the highest priority to transmit. In CAN, the binary representation of identities are used to decide who gets priority. A 0 is termed a "dominant" bit, and a 1 is termed a "recessive" bit. A dominant bit will overwrite a recessive bit on the bus. When several nodes want to transmit at the same time, they transmit their identities bit by bit, while listening on the bus to see if their bit is dominated by other nodes. If any node has a dominant bit, all other nodes will see it. The nodes that have a recessive bit will see the dominant bit on the bus, back off, and attempt transmission at a later time.

In order to formulate a model we begin by defining a suitable psi-calculus, by instantiating the parameters to our framework. For our terms, we use names, a set of constants for channels and binary numbers, and a list construction, so that we can represent binary sequences. **bus**, **can**, σ_1 and σ_2 are all reliable broadcast channels. Priorities are set as follows: $(\sigma_1, 3), (\sigma_2, 1), (\mathbf{bus}, 0), (\mathbf{can}, 0)$, and for all names a, $(a, 2)$. The resulting instance is as follows.

Psi-calculus for CAN

$\mathbf{T} = \mathcal{N} \cup \{\mathbf{bus}, \mathbf{can}, \sigma_1, \sigma_2, 0, 1\} \cup \{M\#N : M, N \in T\}$
$\mathbf{C} = \{\top, \bot\}$
$\mathbf{A} = \{\mathbf{1}\}$
$M \leftrightarrow N = \top$ if $M = N \in \mathcal{N}$, otherwise $\bot$
$M \dot{\succ} N = M \dot{\prec} N = \top$ if $M = N \in \{\mathbf{bus}, \mathbf{can}, \sigma_1, \sigma_2\}$, otherwise $\bot$
$\texttt{reliable}(M) = \top$
$\text{hasprio}(M, n) = \top$ iff $(M, n) \in \{(\sigma_1, 3), (\sigma_2, 1), (\mathbf{bus}, 0), (\mathbf{can}, 0)\} \cup \{(a, 2) : a \in \mathcal{N}\}$
$\mathbf{1} \vdash \top$

A node executing the arbitration protocol is written $F(tl, id, M)$. This represents a node with identity id wanting to transmit M on the bus. The process is defined recursively, and tl is what remains of id to be transmitted. σ_1 and σ_2 are clock channels. σ_1 regulates the cycle of arbitration sessions, and σ_2 regulates the cycle

of bit arbitration within a session. **bus** is the channel used for arbitrations, and **can** is the channel for data transmission.

$$\begin{aligned} F(\epsilon, id, M) &= \overline{\mathbf{can}}\, M \\ F(0 :: tl, id, M) &= \overline{\mathbf{bus}}.F(tl, id, M) + \underline{\mathbf{bus}}.F(tl, id, M) \\ F(1 :: tl, id, M) &= \underline{\mathbf{bus}}.\underline{\sigma_1}.F(id, id, M) + \underline{\sigma_2}.F(tl, id, M) \end{aligned}$$

The base case for the recursion is $F(\epsilon, id, M)$. This simulates the case where all arbitration bits have been consumed, and the node has not been dominated. This means that it is the last node standing in the arbitration process, and therefore gets to send its message on the **can** bus. $F(0 :: tl, id, M)$ simulates the case of the node having a dominant arbitration bit. In this case, one of the nodes with a dominant bit will send it on **bus**, and the rest will receive it. A node with a dominant bit will always progress to $F(tl, id, M)$, and thus stay in the arbitration process. $F(1 :: tl, id, M)$ simulates the case of the node having a recessive arbitration bit. Since **bus** has higher priority than σ_2, then if another node is dominant, the recessive node will see the dominant node's transmission on **bus**, back off and wait for a signal on σ_1 to indicate the beginning of the next arbitration session. If there is no dominant node, and thus no transmission on **bus**, σ_2 can synchronise and allow all recessive nodes to enter the next cycle in the arbitration process.

A system where for all $i \leq n$, the node with identity id_i wishes to send M_i can be given as:

$$F(id_0, id_0, M_0) \mid \ \ldots \ \mid F(id_n, id_n, M_n) \mid !\overline{\sigma_2} \mid !\overline{\sigma_1}$$

We show next an example of two nodes $F(1010, 1010, M_{1010})$ and $F(1000, 1000, M_{1000})$ competing for the right to transmit their message on **can**. Priority annotations are included to help readability, but are not part of the action syntax of the framework:

$$\begin{aligned} &\underline{\mathbf{bus}}.\underline{\sigma_1}.F(1010, 1010, M_{1010}) + \underline{\sigma_2}.F(010, 1010, M_{1010}) \\ \mid\ &\underline{\mathbf{bus}}.\underline{\sigma_1}.F(1000, 1000, M_{1000}) + \underline{\sigma_2}.F(000, 1000, M_{1000}) \mid !\overline{\sigma_2} \mid !\overline{\sigma_1} \end{aligned}$$

Since both are recessive, neither receives anything on **bus**. The system instead does a σ_2 to reveal another bit:

$$\begin{aligned} \xrightarrow{!\overline{\sigma_2}:1}\ &\overline{\mathbf{bus}}.F(10, 1010, M_{1010}) + \underline{\mathbf{bus}}.F(10, 1010, M_{1010}) \\ \mid\ &\overline{\mathbf{bus}}.F(00, 1000, M_{1000}) + \underline{\mathbf{bus}}.F(00, 1000, M_{1000}) \mid !\overline{\sigma_2} \mid !\overline{\sigma_1} \end{aligned}$$

Since both are now dominant, either might choose to signal on **bus**, but the other will continue to persist:

$$\begin{aligned} \xrightarrow{!\overline{\mathbf{bus}}:0}\ &\underline{\mathbf{bus}}.\underline{\sigma_1}.F(1010, 1010, M_{1010}) + \underline{\sigma_2}.F(0, 1010, M_{1010}) \\ \mid\ &\overline{\mathbf{bus}}.F(0, 1000, M_{1000}) + \underline{\mathbf{bus}}.F(0, 1000, M_{1000}) \mid !\overline{\sigma_2} \mid !\overline{\sigma_1} \end{aligned}$$

Node 1010 is now recessive and node 1000 dominant. Since the σ_2 channel has lower priority than **bus**, the only thing that can happen is a communication on

bus:

$$\xrightarrow{!\overline{\mathbf{bus}}:0} \underline{\sigma_1}.F(1010, 1010, M_{1010}) \\ \mid \underline{\overline{\mathbf{bus}}}.F(\epsilon, 1000, M_{1000}) + \underline{\mathbf{bus}}.F(\epsilon, 1000, M_{1000}) \mid !\overline{\sigma_2} \mid !\overline{\sigma_1}$$

A reliable broadcast does not need any recipients to synchronise with. Thus, node 1000 dominates the bus on its own and progresses to:

$$\xrightarrow{!\overline{\mathbf{bus}}:0} \underline{\sigma_1}.F(1010, 1010, M_{1010}) \\ \mid \underline{\overline{\mathbf{can}}}\, M_{1000} \mid !\overline{\sigma_2} \mid !\overline{\sigma_1}$$

Finally, since **can** has higher priority than σ_1, M_{1000} is transmitted. After that, a σ_1 transition can proceed to let node 1010 try again.

In [17], Jan and Zdenek models CAN using the graphical formalism of timed automata and verifying properties in UPPAAL. Their model of the arbitration protocol is similar to ours. In [29], Osch and Smolka models and verifies CAN using Murϕ. At a bit over 100 lines, their model of the arbitration protocol is less concise than ours. We have the benefit of modelling the protocol in a bespoke language, defined as an instance of the psi-calculi framework, where the salient features of the protocol have a direct representation. This language may also be extended and formally coexist with arbitrary data structures for data transmitted along the bus, and inherits a machine checked meta-theory.

6 Meta-theory

In this section we discuss the meta-theory of our calculi, including negative premises, algebraic properties of bisimulation, and our Isabelle formalisation.

6.1 Negative Premises

The predicates ADMIT and HIGHEST are used as negative premises. Here, the SOS rules defining the transition relation $\longrightarrow$ are consistent. We show this by constructing a *stratification* S from transition judgments to a well-ordered set such that for every instance of a rule application that may occur in a derivation tree, the negative premises have smaller S-values than the conclusion, and the positive premises have no larger S-values than the conclusion. This guarantees that there are no transitions whose absence is a precondition for their presence.

Definition 12. *We let* $S(\Psi \rhd P \xrightarrow{\alpha} P')$ *be the pair* $(\texttt{PRIO}\,(\Psi \otimes \mathcal{F}(P), \alpha), s)$*, where* s *is the syntactic size of* P *if* α *is a broadcast input with* $\Psi \otimes \mathcal{F}(P) \vdash \texttt{reliable}(\text{subj}(\alpha))$ *and* ω *otherwise, and comparisons use the lexical ordering.*

Theorem 13. *S is a stratification of the transition system $\longrightarrow$.*

Proof. By case analysis, using the fact that HIGHEST only considers transitions on strictly lower strata. In the case of Rep, we note that α cannot be a reliable broadcast input action since P is reliable reception guarded—hence both premise and conclusion are on stratum (p, ω) for some p. The other cases are trivial.

The transition system $\longrightarrow$ is defined by constructing transitions stratum by stratum in a bottom-up fashion, as explained in the introduction. Further, Groote shows that with this method, any two valid stratifications of a transition system specification yield the same transition relation: hence the stratification can be elided. In our Nominal Isabelle formalisation discussed in Section 6.2, we do not directly employ negative premises, in order to avoid defining the semantics by induction over strata.

We give an equivalent alternative definition inspired by the notions of discard relations [11] and initial actions [16].

Definition 14. *Define predicate* HIGHEST_U *as* HIGHEST *with all instances of the transition relation* $\longrightarrow$ *replaced by* $\longrightarrow_\text{U}$*, and similarly for* ADMIT*, where* $\longrightarrow_\text{U}$ *is* $\longrightarrow$ *with all negative premises removed from the rules. We then define a new transition system* $\longrightarrow_\text{I}$ *by replacing all uses of* HIGHEST *in the SOS rules for* $\longrightarrow$ *by* HIGHEST_U *(and similarly for* ADMIT*).*

Note that $\longrightarrow_\text{I}$ does not contain any negative premises in the formal sense, so it is more readily formalized in Isabelle. Moreover, it coincides with $\longrightarrow$.

Lemma 15. $\text{HIGHEST}_\text{U}(\alpha, \Psi, P) \iff \text{HIGHEST}_\text{I}(\alpha, \Psi, P)$

Proof. This lemma has been formally proven in Nominal Isabelle.

Lemma 16. $\text{HIGHEST}_\text{U}(\alpha, \Psi, P \mid Q) \implies$
$(\text{ADMIT}_\text{U}(\alpha, \Psi, P, Q) \iff \text{ADMIT}_\text{I}(\alpha, \Psi, P, Q))$.

Proof. This lemma has been formally proven in Nominal Isabelle.

Theorem 17. $\longrightarrow = \longrightarrow_\text{I}$

Proof (sketch). We consider transitions $\Psi \rhd R \xrightarrow{\alpha}_* R'$ where $*$ is nothing or I. The proof is by strong induction on the priority p of the action α in frame $\mathcal{F}(R) \otimes \Psi$. Here $\text{HIGHEST}(\alpha, \Psi, R)$ iff (by induction hypothesis) $\text{HIGHEST}_\text{I}(\alpha, \Psi, R)$ iff (by Lemma 15) $\text{HIGHEST}_\text{U}(\alpha, \Psi, R)$. We proceed by structural induction on the origin R of the transition. Most cases for R follow by case analysis on the topmost derivation rule; the remaining are as follows.

$R = P \mid Q$ **using** RPar**:** By induction P has the same transitions with $\longrightarrow$ as with $\longrightarrow_\text{I}$. We have $\text{HIGHEST}(\alpha, \Psi, R) \iff \text{HIGHEST}_\text{U}(\alpha, \Psi, R)$ by the outer induction. Remains the side condition ADMIT.
- $\Rightarrow$ Assume that $\text{ADMIT}(\alpha, \Psi, P, Q)$. By induction we get $\text{ADMIT}_\text{I}(\alpha, \Psi, P, Q)$. By Lemma 16 $\text{ADMIT}_\text{U}(\alpha, \Psi, P, Q)$.
- $\Leftarrow$ Assume that $\text{ADMIT}_\text{U}(\alpha, \Psi, P, Q)$. By Lemma 16 $\text{ADMIT}_\text{I}(\alpha, \Psi, P, Q)$. By induction $\text{ADMIT}(\alpha, \Psi, P, Q)$.

$R = !P$ We proceed by induction on the number of uses of the Rep rule to unfold $!P$ in the derivation of the transition.

Base case Here the transition was derived using Rep;RPar. By induction P has the same transitions in the system $\longrightarrow$ as in $\longrightarrow_\text{I}$. Moreover, $\text{ADMIT}(\alpha, \Psi, P, !P)$ and $\text{ADMIT}_\text{U}(\alpha, \Psi, P, !P)$ are always satisfied, since P is BI-guarded and $!P$ thus has no broadcast input transitions.

Induction case By case analysis on the rule used to derive the transition from $P \mid !P$; most cases follow directly from the induction hypotheses. The exception is RPar-R. Here, by induction the transition from $!P$ exists in $\longrightarrow$ iff it exists in $\longrightarrow_{\mathrm{I}}$. Moreover, $\mathrm{ADMIT}(\alpha, \Psi, !P, P)$ and $\mathrm{ADMIT}_{\mathrm{U}}(\alpha, \Psi, !P, P)$ are always satisfied since P is BI-guarded.

6.2 Bisimulation

For any reliable broadcast psi-calculus, labelled bisimilarity respects important algebraic laws. Recall the definitions of strong bisimilarity and congruence are from Section 2. Strong bisimulation is preserved by all operators except input prefix and satisfies the expected algebraic laws such as scope extension:

Theorem 18 (Congruence properties of strong bisimulation). *For all Ψ:*

$$\begin{aligned}
P \mathrel{\dot\sim_\Psi} Q &\implies P \mid R \mathrel{\dot\sim_\Psi} Q \mid R \\
P \mathrel{\dot\sim_\Psi} Q &\implies (\nu a)P \mathrel{\dot\sim_\Psi} (\nu a)Q \qquad \textit{if } a\#\Psi \\
P \mathrel{\dot\sim_\Psi} Q &\implies !P \mathrel{\dot\sim_\Psi} !Q \qquad \textit{if } P, Q \textit{ assertion guarded} \\
\forall i. P_i \mathrel{\dot\sim_\Psi} Q_i &\implies \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \mathrel{\dot\sim_\Psi} \mathbf{case}\ \widetilde{\varphi} : \widetilde{Q} \\
P \mathrel{\dot\sim_\Psi} Q &\implies \overline{M}\,N\,.\,P \mathrel{\dot\sim_\Psi} \overline{M}\,N\,.\,Q \\
(\forall \widetilde{L}.\ P[\widetilde{x} := \widetilde{L}] \mathrel{\dot\sim_\Psi} Q[\widetilde{x} := \widetilde{L}]) &\implies \underline{M}(\lambda \widetilde{x})N\,.\,P \mathrel{\dot\sim_\Psi} \underline{M}(\lambda \widetilde{x})N\,.\,Q
\end{aligned}$$

Theorem 19. *Strong congruence $\sim_\Psi$ is a congruence for all Ψ.*

The standard rules of structural equivalence are sound for bisimilarity congruence.

Theorem 20 (Structural equivalence). *Assume that $a\#Q, \widetilde{x}, M, N, \widetilde{\varphi}$. Then*

$$\begin{array}{cc}
\mathbf{case}\ \widetilde{\varphi} : \widetilde{(\nu a)P} \sim (\nu a)\mathbf{case}\ \widetilde{\varphi} : \widetilde{P} & (\nu a)\mathbf{0} \sim \mathbf{0} \\
\underline{M}(\lambda \widetilde{x})N\,.\,(\nu a)P \sim (\nu a)\underline{M}(\lambda \widetilde{x})(N)\,.\,P & Q \mid (\nu a)P \sim (\nu a)(Q \mid P) \\
\overline{M}\,N\,.\,(\nu a)P \sim (\nu a)\overline{M}\,N\,.\,P & (\nu b)(\nu a)P \sim (\nu a)(\nu b)P \\
P \mid (Q \mid R) \sim (P \mid Q) \mid R & !P \sim P \mid !P \\
P \mid Q \sim Q \mid P & P \sim P \mid \mathbf{0}
\end{array}$$

We have formalised reliable broadcast psi-calculi and the proofs of all theorems in Section 6.2 in Nominal Isabelle [27] — the proof scripts are available online [1]. Since considerable effort has been invested in a proof repository for broadcast psi-calculi [26], we strive to re-use as much of it as possible. However, using negative premises and induction over strata would constitute a major change to the structure of the definition of the operational semantics. Instead, we formalise the two semantics $\longrightarrow_{\mathrm{U}}$ and $\longrightarrow_{\mathrm{I}}$ (Definition 14). Using this strategy, we were able to re-use our repository with only minimal changes to the proofs. The proof scripts constitute 28469 lines of code for $\longrightarrow_{\mathrm{I}}$ and 33823 lines for $\longrightarrow_{\mathrm{U}}$, most of it straightforward adaptations of the existing proof scripts.

The bulk of our adaptation efforts concerns proving various preservation properties of the $\mathrm{ADMIT}_{\mathrm{U}}$ predicate used in the structural congruence proofs, such as

$$\mathrm{ADMIT}_{\mathrm{U}}(\alpha, \Psi, F, Q \mid R) \to \mathrm{ADMIT}_{\mathrm{U}}(\alpha, \Psi, F \otimes \mathcal{F}(R), Q)$$

Other than such details, the proofs are remarkably similar to the case of unreliable broadcast. The same technical lemmas, overarching proof structures and candidate relations are used. Hence, we shall not restate them here. This is a compelling argument for using proof mechanisation: even though the initial effort of developing a proof repository for broadcast psi-calculi was significant, adopting it for the reliable case took only roughly two man-weeks of effort. Adding priorities to unicast psi-calculi took a similar amount of time, as did combining priorities and reliable broadcasts into a single calculus. We consider this a small price to pay for absolute certainty of correctness, at least up to the current state of the art of mechanised proofs.

7 Related work

As early as 1985, Pnueli [22] formulated the semantics of a parallel composition operator with broadcast synchronisation using negative premises. This predates the work on negative premises by Groote, and shows no awareness that introducing such rules may lead to consistency issues. Fortunately, all rules pertaining to it are syntax-directed and hence trivial to stratify. The first account of priorities in process algebra is by Cleaveland and Hennessy [7], who define the semantics using two strata, an idea we also use in defining $\longrightarrow_{\mathrm{U}}$ and $\longrightarrow_{\mathrm{I}}$. A more recent exposition of this area is in [8].

Prasad [23] expresses reliable broadcast in CBS by introducing so-called "lose" actions, alongside the usual input and output, to denote that a process may discard a message. Prasad remarks that "a loss encodes a negative premise", and that "it is possible to formulate CBS too in terms of negative premises". We go further than this by providing both formulations and proving that they correspond.

Prasad also extends CBS with priorities [24]. The priority system is a static one, where every prefix is tagged with an explicit priority level. This yields a very clean operational semantics, with no need for the negative premises or two-stage operational semantics used in the case of unicast communication. Our own formulation is by necessity more complicated since it incorporates features absent in CBS such as dynamic priorities, mobility, unicast communication and the possibility of lossy broadcasts.

The $b\pi$ calculus by Ene and Muntean [10,11] is a version of the pi-calculus featuring reliable broadcast instead of point-to-point communication. In lieu of negative premises, they use an auxiliary "discard" relation to determine whether a process is allowed to discard a message. When compared to Prasad's "lose" actions, this strategy has the benefit of not introducing additional actions into

the semantics. Moraru et al. [19] make an initial effort towards an extension of $b\pi$ with ambients and the possibility of encrypting messages.

Gunter and Yasmeen's work on Secure Broadcast Ambients [13] does not feature reliable broadcast, but is another example of a broadcast calculus implemented in Nominal Isabelle [30]. The authors describe the implementation of the syntax and semantics of their calculus, as well as barbed congruence and equivalence. However, they do not use their framework to prove any theorems.

Jeffrey [16] shows how to translate a variant of timed CCS into prioritised CCS by attaching an explicit time stamp to each action's priority. Crucially, Jeffrey's translation relies on the CSP synchronisation operator, which can be seen as a form of reliable broadcast communication. Prasad achieves a similar feat in [25], where he argues that prioritised CBS is isomorphic to a subset of timed CBS. This, in conjunction with our own findings, leads us to believe that Jeffrey's observation that "time is an example of priority" ought to be clarified: merely having priorities is not sufficient. A one-to-many, non-lossy synchronisation mechanism is also required.

8 Conclusion

Negative premises to formulate reliable broadcast and priorities has a long history. Our contribution is to introduce these ideas in psi-calculi, a general framework where highly specialised modelling languages are easily defined, and where a proof repository for Nominal Isabelle guarantees that important meta-theoretical properties related to bisimulation is sound. We go beyond existing process algebras in that priorities and reliability can vary dynamically as processes execute. We have also investigated the expressivity of reliable broadcast with priorities by modelling discrete timeouts and the arbitration protocols of a vehicular bus.

In future work, we intend to add these features to sorted psi-calculi [5]. We also intend to study the properties of a cache coherence protocol using Nominal Isabelle. It would be interesting to find a more general characterization of which instances of negative premises can be encoded in a similar two-level style. We also want to investigate whether there is a symbolic operational semantics that captures psi-calculi with priorities and reliable broadcast.

References

1. Johannes Åman Pohjola. Reliable broadcast psi-calculus with priorities formalisation. `http://www.it.uu.se/research/group/mobility/theorem/reliablebroadcast.tgz`, April 2013. Isabelle/HOL-Nominal formalisation of the definitions, theorems and proofs.
2. Jesper Bengtson. *Formalising process calculi*. PhD thesis, Uppsala University, June 2010.
3. 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.

4. Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced: preliminary report. In *Department of Computer Science, Cornell University*, pages 229–239, 1988.
5. Johannes Borgström, Ramūnas Gutkovas, Joachim Parrow, Björn Victor, and Johannes Åman Pohjola. A sorted semantic framework for applied process calculi (extended abstract), 2013. Submitted; an early version entitled *Sorted Psi-calculi with Generalised Pattern Matching* was presented at ICE'12.
6. 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, November 2011.
7. Rance Cleaveland and Matthew Hennessy. Priorities in process algebras. In *LICS*, pages 193–202. IEEE Computer Society, 1988.
8. Rance Cleaveland, Gerald Lüttgen, and V. Natarajan. Priority and abstraction in process algebra. *Inf. Comput.*, 205(9):1426–1458, 2007.
9. Steve Corrigan. Introduction to the controller area network (CAN). Technical Report SLOA101A, Texas Instruments, July 2008.
10. Cristian Ene. *Un Modèle formel pour les systèmes mobiles a diffusion*. Phd thesis, Université de la Méditerranée – Marseille, 2001.
11. 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.
12. Jan Friso Groote. Transition system specifications with negative premises. *Theor. Comput. Sci.*, 118(2):263–299, September 1993.
13. 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.
14. Hans Hansson and Bengt Jonsson. A calculus for communicating systems with time and probabitilies. In *RTSS*, pages 278–287. IEEE Computer Society, 1990.
15. Matthew Hennessy and Tim Regan. A process algebra for timed systems. *Inf. Comput.*, 117(2):221–239, 1995.
16. Alan Jeffrey. Translating timed process algebra into prioritized process algebra. In Jan Vytopil, editor, *FTRTFT*, volume 571 of *Lecture Notes in Computer Science*, pages 493–506. Springer, 1992.
17. Jan Krakora and Zdenek Hanzálek. Timed automata approach to CAN verification. In P. Kopacek, C.E. Pereira, and G. Morel, editors, *Information Control Problems in Manufacturing*, pages 147–152. IFAC, 2004.
18. Damiano Macedonio and Massimo Merro. A semantic analysis of wireless network security protocols. In Alwyn Goodloe and Suzette Person, editors, *NASA Formal Methods*, volume 7226 of *Lecture Notes in Computer Science*, pages 403–417. Springer, 2012.
19. Victor Moraru, Traian Muntean, and Emilian Gutuleac. Towards a model for broadcasting secure mobile processes. In *Proceedings of The Fifth International Symposium on Parallel and Distributed Computing*, ISPDC '06, pages 168–172, Washington, DC, USA, 2006. IEEE Computer Society.

20. Xavier Nicollin and Joseph Sifakis. An overview and synthesis on timed process algebras. In Kim Guldstrand Larsen and Arne Skou, editors, *CAV*, volume 575 of *Lecture Notes in Computer Science*, pages 376–398. Springer, 1991.
21. A. M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186:165–193, 2003.
22. Amir Pnueli. Linear and branching structures in the semantics and logics of reactive systems. In Wilfried Brauer, editor, *ICALP*, volume 194 of *Lecture Notes in Computer Science*, pages 15–32. Springer, 1985.
23. K. V. S. Prasad. A calculus of value broadcasts. In *IN PARLE'93*, pages 69–4. Springer Verlag LNCS, 1993.
24. 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.
25. K. V. S. Prasad. Broadcasting in time. In Paolo Ciancarini and Chris Hankin, editors, *COORDINATION*, volume 1061 of *Lecture Notes in Computer Science*, pages 321–338. Springer, 1996.
26. Palle Raabjerg and Johannes Åman Pohjola. Broadcast psi-calculus formalisation. `http://www.it.uu.se/research/group/mobility/theorem/broadcastpsi`, July 2011. Isabelle/HOL-Nominal formalisation of the definitions, theorems and proofs.
27. 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.
28. Rob J. van Glabbeek. The meaning of negative premises in transition system specifications ii. *J. Log. Algebr. Program.*, 60-61:229–258, 2004.
29. M. J. P. van Osch and Scott A. Smolka. Finite-state analysis of the CAN bus protocol. In *HASE*, pages 42–. IEEE Computer Society, 2001.
30. 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.

Chapter 6

Bisimulation up-to techniques for psi-calculi

Paper V

Bisimulation up-to techniques are methods for reducing the size of relations needed in bisimulation proofs. In this chapter, we show how the bisimulation proof method of Sangiorgi [San98] can be adapted to psi-calculi. We formalise all our definitions and theorems in Nominal Isabelle, and show examples where the use of up to-techniques has simplified our proofs of known results. We also prove a few new structural laws about the replication operator.

Showing that a relation $\mathcal{R}$ is a bisimulation essentially boils down to showing that the following diagram commutes for all pairs $(P, Q) \in \mathcal{R}$:

$$\begin{array}{ccc} P & \mathcal{R} & Q \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle\alpha} \\ P' & \mathcal{R} & Q' \end{array}$$

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). A refinement due to Milner is "bisimulation up-to $\dot{\sim}$" [Mil89]:

$$\begin{array}{ccc} P & \mathcal{R} & Q \\ {\scriptstyle\alpha}\downarrow & & \downarrow{\scriptstyle\alpha} \\ P'\dot{\sim} & \mathcal{R} & \dot{\sim}Q' \end{array}$$

This proof technique can significantly reduce the amount of work necessary in bisimulation proofs, since it allows known results about bisimilarity to be re-used when establishing the lower part, without including them in $\mathcal{R}$, hence reducing the number of transitions that must be followed in order to complete the proof. Similar techniques are often referred to collectively as *bisimulation up-to techniques*.

As extensions of psi-calculi grow in complexity, so does the bisimulation up-to techniques used to prove their meta-theory. In the formalisation of psi-calculi by Bengtson [BP09, Ben10], bisimulation up-to $\dot{\sim}$ is the most complicated technique used. In broadcast psi-calculi [BHJ$^+$11], bisimulation up to the transitive closure of a relation is used to prove that binders commute. In higher-order psi-calculi [PBRÅP13], bisimulation up-to the transitive closure of the union with bisimilarity of the closure under binding sequences of a relation is used to show that higher-order bisimilarity is closed under the Par operator.

Of course each new such proof technique must be proven to be sound. Up until now, these soundness proofs have been done on a case-by-case basis. Since the up-to techniques used often share many similar 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.

We improve on this state of affairs by adopting the method of Sangiorgi, which uses a notion of *respectfulness* that provides a systematic way to increase

the size of the target relation without increasing the size of the source relation in a bisimulation proof. Respectfulness is 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.

Sangiorgi has adapted this bisimulation proof method to the pi-calculus [San98]; Hirschkoff implemented it as part of his Coq formalisation of the pi-calculus [Hir97]. The novel feature of psi-calculi with the biggest impact on the bisimulation proof method is the environmental assertions. Since the transition relation is parametrised on an assertion environment, so is bisimulation: recall that a bisimulation relation in psi-calculi is ternary. Further, triples in a bisimulation relation must have statically equivalent frames, and the relation must be closed under arbitrary extensions of the assertion environment. In order to acquire a sound theory of respectful functions, we must impose similar requirements on respectfulness.

6.1 Theory

We will use $\mathcal{R}, \mathcal{S}$ to range over relations in $\mathbf{A} \times \mathbf{P} \times \mathbf{P}$. $\mathcal{F}, \mathcal{G}$ range over functions of type $\mathbf{A} \times \mathbf{P} \times \mathbf{P} \Rightarrow \mathbf{A} \times \mathbf{P} \times \mathbf{P}$. Such functions will be referred to as *first-order functions*. $\mathcal{A}$ ranges over sets of first-order functions. Functions that have first-order functions as both arguments and return values will be called *constructors*.

The notion of *progression* lifts the transition relation from processes to relations between processes:

Definition 9 (Progression)**.** *We say that* $\mathcal{R}$ progresses to $\mathcal{S}$*, denoted* $\mathcal{R} \rightarrowtail \mathcal{S}$*, iff for all* $(\Psi, P, Q) \in \mathcal{R}$*:*

1. *for all* α, Q' *such that* $\mathrm{bn}(\alpha) \sharp \Psi, P$ *and* $\Psi \rhd Q \xrightarrow{\alpha} Q'$*, there exists* P' *such that* $\Psi \rhd P \xrightarrow{\alpha} P'$ *and* $(\Psi, P', Q') \in \mathcal{S}$

2. *for all* α, P' *such that* $\mathrm{bn}(\alpha) \sharp \Psi, Q$ *and* $\Psi \rhd P \xrightarrow{\alpha} P'$*, there exists* Q' *such that* $\Psi \rhd Q \xrightarrow{\alpha} Q'$ *and* $(\Psi, P', Q') \in \mathcal{S}$

The intuition is that if $\mathcal{R} \rightarrowtail \mathcal{R}$ then $\mathcal{R}$ satisfies the simulation clause of bisimulation. However, this is not quite sufficient for $\mathcal{R}$ to be a bisimulation relation in psi-calculi, for two reasons. The first is that bisimulation relations in psi-calculi are required to be symmetric, a requisite which is absent here for reasons that shall be explained later. The second is that it only accounts for the simulation clause of bisimulation, not the clauses pertaining to static equivalence and extensibility of the assertion environment. For this purpose, we lift these notions, as well as the notion of equivariance, to the level of relations.

Definition 10. *We say that a relation* $\mathcal{R}$ *is:*

1. statically equivalent *if for all* $(\Psi, P, Q) \in \mathcal{R}$*,* $\Psi \otimes \mathcal{F}(P) \simeq \Psi \otimes \mathcal{F}(Q)$.

2. extensible *if for all* $(\Psi, P, Q) \in \mathcal{R}$ *and all* Ψ' , $(\Psi \otimes \Psi', P, Q) \in \mathcal{R}$.

3. equivariant *if for all* $(\Psi, P, Q) \in \mathcal{R}$ *and all permutations* p, $(p \cdot \Psi, p \cdot P, p \cdot Q) \in \mathcal{R}$

From now on, we will only be concerned with relations that are statically equivalent, extensible and equivariant. Our reason for only considering equivariant relations will be made clear later.

The main contribution of [San98] is the discovery of a systematic method that given a relation $\mathcal{R}$ makes it possible to construct a larger relation $\mathcal{S}$ such that if $\mathcal{R} \rightarrowtail \mathcal{S}$ then $\mathcal{R} \subseteq \dot{\sim}$. This significantly simplifies the bisimulation proof method by reducing the size of relations and hence the number of pairs from which transitions must be followed. The technique for constructing such an $\mathcal{S}$ is to apply to $\mathcal{R}$ functions that are *respectful*. The notion of respectfulness, adapted to psi-calculi, looks as follows:

Definition 11 (Respectfulness)**.** *We say that a function* $\mathcal{F}$ *is* respectful *iff*

1. *If* $\mathcal{R}$ *is statically equivalent, then so is* $\mathcal{F}(\mathcal{R})$

2. *If* $\mathcal{R}$ *is equivariant and extensible, then* $\mathcal{F}(\mathcal{R})$ *is extensible.*

3. $\mathcal{F}$ *is equivariant.*

4. *If* $\mathcal{R} \subseteq \mathcal{S}$*, and* $\mathcal{R} \rightarrowtail \mathcal{S}$*, and* $\mathcal{R}$ *is equivariant, and* $\mathcal{S}$ *is equivariant, statically equivalent and extensible, then* $\mathcal{F}(\mathcal{R}) \subseteq \mathcal{F}(\mathcal{S})$ *and* $\mathcal{F}(\mathcal{R}) \rightarrowtail \mathcal{F}(\mathcal{S})$

A constructor is respectful *if it preserves respectfulness.*

Clauses 11.1 and 11.2 are necessary since bisimulation relations are required to be statically equivalent and extensible — hence requiring respectful functions to preserve them is natural. Clause 11.3 ensures that respectful functions preserves equivariance. Clause 11.4 states that respectful functions must be monotone and preserve progression, a requirement that should be familiar to readers of [San98].

Theorem 18 (Soundness)**.** *If* $\mathcal{F}$ *is respectful, and* $\mathcal{R}$ *is statically equivalent, extensible and equivariant, and* $\mathcal{R} \rightarrowtail \mathcal{F}(\mathcal{R})$*, then* $\mathcal{R} \subseteq \dot{\sim}$

Proof. We define $\mathcal{R}_n$ where $n \in \mathbb{N}$ to be

$$\mathcal{R}_0 \triangleq \mathcal{R} \qquad\qquad \mathcal{R}_{n+1} \triangleq \mathcal{F}(\mathcal{R}_n) \cup \mathcal{R}_n$$

and show that the symmetric closure of $\bigcup_n \mathcal{R}_n$ is a bisimulation relation that includes $\mathcal{R}$. □

We introduce a few useful respectful functions and constructors. We will use the identity function $\mathcal{I}$, the inverse function $\_^{-1}$, the constant functions $\mathcal{U}$ and $\mathcal{X}$ that maps every relation to bisimilarity and the identity relation, respectively,

and the closure $\mathcal{E}$ of a relation under static equivalence. The main reason respectfulness is attractive is that respectful functions can be combined to form more complex functions which are themselves respectful, using important constructors such as composition $\circ$, union $\cup$ and chaining $\frown$.

Definition 12 (Primitive respectful functions and constructors)**.**

1. $\mathcal{I}(\mathcal{R}) \triangleq \mathcal{R}$
2. $\mathcal{U}(\mathcal{R}) \triangleq \dot{\sim}$
3. $\mathcal{X}(\mathcal{R}) \triangleq \{(\Psi, P, P) : \mathbf{true}\}$
4. $\mathcal{E}(\mathcal{R}) \triangleq \{(\Psi, P, Q) : \exists \Psi'.\Psi' \simeq \Psi \wedge (\Psi', P, Q) \in \mathcal{R}\}$
5. $\mathcal{R}^{-1} \triangleq \{(\Psi, P, Q) : (\Psi, Q, P) \in \mathcal{R}\}$
6. $(\mathcal{G} \circ \mathcal{F})(\mathcal{R}) \triangleq \mathcal{G}(\mathcal{F}(\mathcal{R}))$
7. $(\mathcal{G} \cup \mathcal{F})(\mathcal{R}) \triangleq \mathcal{G}(\mathcal{R}) \cup \mathcal{F}(\mathcal{R})$
8. $(\mathcal{G} \frown \mathcal{F})(\mathcal{R}) \triangleq \{(\Psi, P, Q) : \exists P'.(\Psi, P, P') \in \mathcal{G}(\mathcal{R}) \wedge (\Psi, P', Q) \in \mathcal{F}(\mathcal{R})\}$
9. $(\bigcup \mathcal{A})(\mathcal{R}) \triangleq \bigcup_{\mathcal{F} \in \mathcal{A}} \mathcal{F}(\mathcal{R})$

Most of the above are straightforward adaptations of functions introduced by Sangiorgi, though $\mathcal{X}$ and $\_^{-1}$ do not appear to have been considered as respectful functions in the literature before. The two union operators are defined differently here so as to avoid having a different union operator for every arity. Note that $\cup$ can be defined as a special case of $\bigcup$. The main novelty is $\mathcal{E}$, which is very useful since bisimulation proofs in psi-calculi often rely on being able to rewrite the frame modulo static equivalence. Note that in a pi-calculus setting, $\mathcal{E}$ would collapse to the identity function.

Two design decisions made in adapting the bisimulation proof method to psi-calculi that we mentioned previously are the restriction to equivariant relations, as well as allowing symmetric relations. We will now motivate with examples.

Without equivariance, respectfulness of important constructors such as the chaining operator fail. For a counterexample, consider a psi-calculus where there is only a single assertion, namely the unit assertion (which is elided in the following). Let P_a be a process with no transitions whose support is $\{a\}$. Consider the process $P \triangleq (\nu x)\underline{a}(x).P_x$ and take the relation

$$\mathcal{R} = \{(P, P_b \mid P)\} \cup \{(P_x, P_b \mid P_x) : x \neq b\}$$

Then $\mathcal{R} \rightarrowtail \mathcal{R}$ holds, since bound outputs from P where the object is b need not be considered because of the freshness side conditions on the Par rule and the definition of progression, respectively. However, for

$$\mathcal{S} \triangleq (\mathcal{I} \frown \_^{-1})(\mathcal{R}) = \{(P,P)\} \cup \{(P_x, P_x) : x \neq b\}$$

$\mathcal{S} \rightarrowtail \mathcal{S}$ fails to hold, contradicting respectfulness of chaining: P has the transition $P \xrightarrow{\overline{a}(\nu b)b} P_b$ but $P_b \notin \mathcal{S}$.

Our solution of only considering equivariant relations is similar to Hirschkoff's solution in [Hir97], where only "good" relations are considered. A "good" relation in his setting based on de Bruijn-indices is analogous to an equivariant relation in our nominal logic-based setting. Hirschkoff motivates this restriction with appeals to intuition, with remarks such as

> these transformations ... do not really have a strong significance, since they ... do some kind of 'administrative work' to keep the notation coherent. Therefore, if a pair (P,Q) of processes belongs to a given relation $\mathcal{R}$, there should be no reason for $[(p \cdot P, p \cdot Q)]$ not to be in $\mathcal{R}$.

Square brackets in the above quote demark parts where I have substituted de Bruijn-index specific language for the nominal logic analogue. The counterexample presented here applies to the pi-calculus as well as psi-calculi, and hence demonstrates that Hirschkoff's design decision is not merely a matter of convenience, but a matter of necessity.

Other solutions than only considering equivariant relations might be feasible. Sangiorgi has suggested (in e-mail correspondence) the approach of always closing under all permutations before chaining. We have opted against this since we do not consider the case of non-equivariant relations to be practically significant enough to warrant special treatment: every bisimulation proof we have ever done for psi-calculi has used an equivariant relation.

The reason for allowing non-symmetric relations is likewise that symmetry is not preserved by chaining. Consider the two constant functions $\mathcal{F}$ and $\mathcal{G}$ such that

$$\mathcal{F}(\mathcal{R}) \triangleq \{(\Psi, \mathbf{0}, (\!|\mathbf{1}|\!)), (\Psi, (\!|\mathbf{1}|\!), \mathbf{0})\} \qquad \mathcal{G}(\mathcal{R}) \triangleq \{(\Psi, \mathbf{0} \mid \mathbf{0}, (\!|\mathbf{1}|\!)), (\Psi, (\!|\mathbf{1}|\!), \mathbf{0} \mid \mathbf{0})\}$$

It is easy to check that $\mathcal{F}$ and $\mathcal{G}$ are respectful and preserve symmetry. However, $(\mathcal{F} \frown \mathcal{G})(\mathcal{R})$ is not symmetric[1].

Lemma 19. *The functions $\mathcal{I}$, $\mathcal{U}$, $\mathcal{X}$, $\_^{-1}$ and $\mathcal{E}$, and the constructors $\circ$, $\frown$ and $\cup$ are all respectful. If every $\mathcal{F} \in \mathcal{A}$ is respectful, then $\bigcup \mathcal{A}$ is respectful.*

Using just these primitive functions and constructors, many useful proof techniques can be obtained. For an example, Milner's "bisimulation up-to $\dot{\sim}$" is $\mathcal{U} \frown \mathcal{I} \frown \mathcal{U}$.

Since respectfulness is closed under union, there is a largest respectful function which we denote $\mathcal{F}^\star$ and define to be $\mathcal{F}^\star \triangleq \bigcup\{\mathcal{F} : \mathcal{F} \text{ is respectful}\}$. This

[1] Of course, the processes related by $(\mathcal{F} \frown \mathcal{G})\mathcal{R}$ are nonetheless bisimilar in this example.

construct appears to be novel, and is practically useful when applying the bisimulation proof method since it grants flexibility in the choice of $\mathcal{F}$ until the latest possible moment. The only situation in which choosing an $\mathcal{F}$ other than $\mathcal{F}^\star$ would be meaningful is when it occurs negatively in a goal, such as during an induction.

Corollary 20. *The function $\mathcal{F}^\star$ is respectful.*

The ability to close a relation under contexts is a desirable proof technique. We here consider the closure under each operator of the finite subcalculi separately — the closure under arbitrary contexts can be obtained by composing them using the $\circ$ constructor.

Definition 13 (Closure under contexts)**.**

1. $\mathcal{C}_{\text{RES}}(\mathcal{R}) \triangleq \{(\Psi, (\nu\widetilde{x})P, (\nu\widetilde{x})Q) : \widetilde{x}\sharp\Psi \wedge (\Psi, P, Q) \in \mathcal{R}\}$
2. $\mathcal{C}_{\text{PAR}}(\mathcal{R}) \triangleq \{(\Psi, P \mid R, Q \mid R) : \widetilde{b}_R\sharp\Psi, P, Q \wedge (\Psi \otimes \Psi_R, P, Q) \in \mathcal{R}\}$
3. $\mathcal{C}_{\text{OUT}}(\mathcal{R}) \triangleq \{(\Psi, \overline{M}\,N.P, \overline{M}\,N.Q) : (\Psi, P, Q) \in \mathcal{R}\}$
4. $\mathcal{C}_{\text{IN}}(\mathcal{R}) \triangleq \{(\Psi, \underline{M}(\lambda\widetilde{x})N\,.\,P, \underline{M}(\lambda\widetilde{x})N\,.\,Q) : \forall\widetilde{T}.|\widetilde{x}| = |\widetilde{T}| \to (\Psi, P[\widetilde{x} := \widetilde{T}], Q[\widetilde{x} := \widetilde{T}]) \in \mathcal{R}\}$
5. $\mathcal{C}_{\text{CASE}}(\mathcal{R}) \triangleq \{(\Psi, \mathbf{case}\ \phi : P \,[\!]\, \phi_0 : R_0 \,[\!]\, \ldots \,[\!]\, \phi_n : R_n, \mathbf{case}\ \phi : Q \,[\!]\, \phi_0 : R_0 \,[\!]\, \ldots \,[\!]\, \phi_n : R_n) : (\Psi, P, Q) \in \mathcal{R} \wedge P, Q, R_0, \ldots, R_n \textit{ guarded}\}$

A reader may ask why $\mathcal{C}_{\text{PAR}}$ is not defined using the following much simpler formulation:

$$\mathcal{C}'_{\text{PAR}} \triangleq \{(\Psi, P \mid R, Q \mid R) : (\Psi, P, Q) \in \mathcal{R}\}$$

There are two reasons for this choice. First, we do not know if $\mathcal{C}'_{\text{PAR}}$ is respectful. Second, $\mathcal{C}'_{\text{PAR}}(\mathcal{R}) \subseteq \mathcal{C}_{\text{PAR}}(\mathcal{R})$ for all extensible $\mathcal{R}$, so since we only consider extensible relations $\mathcal{C}_{\text{PAR}}$ is more powerful.

We have only defined the closure under parallel contexts where the common context occurs to the right of the $\mid$ operator. To obtain it to the left instead, we use $\mathcal{U} \frown \mathcal{C}_{\text{PAR}} \frown \mathcal{U}$ and the structural law $\Psi \rhd P \mid Q \mathrel{\dot{\sim}} Q \mid P$. Through a similar technique we may obtain the closure under **case** contexts where P and Q occur in positions other than the first.

Note that $\mathcal{C}_{\text{IN}}$ only requires membership in $\mathcal{R}$ for such substitutions of P and Q as may arise from the particular input prefix under consideration. This constitutes an improvement over [San98, Hir97], where a quantification over all substitution sequences is used.

An attentive reader may notice that we do not consider the closure of a relation under replication. For comments on this omission we refer to Section 6.3.

While not all of these functions are respectful in and of themselves, they are respectful when combined with certain other constructs, as shown in Theorem 21. When using these functions in proofs, this is merely a technicality

since the respectful functions below include the closures under the operators. For an example, $\mathcal{C}_{\text{PAR}}(\mathcal{R}) \subseteq (\mathcal{C}_{\text{RES}} \circ \mathcal{C}_{\text{PAR}} \circ \mathcal{E})(\mathcal{R})$ holds for all $\mathcal{R}$.

Theorem 21. *The following functions are respectful:*

1. $\mathcal{C}_{\text{RES}}$
2. $\mathcal{C}_{\text{RES}} \circ \mathcal{C}_{\text{PAR}} \circ \mathcal{E}$
3. $\mathcal{C}_{\text{OUT}} \cup \mathcal{I}$
4. $\mathcal{C}_{\text{IN}} \cup \mathcal{I}$
5. $\mathcal{C}_{\text{CASE}} \cup \mathcal{I} \cup \mathcal{X}$

Finally we consider an interesting special case of Definition 11, namely when $\mathcal{R} = \mathcal{S} = \dot{\sim}$. From this it follows that bisimilarity is closed under all respectful functions.

Theorem 22. *If $\mathcal{F}$ is respectful, then $\mathcal{F}(\dot{\sim}) \subseteq \dot{\sim}$*

As an immediate consequence, we obtain alternative proofs that bisimilarity is closed under restriction, parallel, output and case.

6.2 Examples

In this section, we apply the proof techniques introduced in Section 6.1 to obtain significantly shorter proofs of familiar results from the psi-calculi literature. We also prove a new structural law about replication.

6.2.1 Bisimilarity is closed under replication

An Isabelle proof that bisimilarity is closed under replication for psi-calculi is due to Bengtson [BP09] — formally, the result is that if $\Psi \rhd P \dot{\sim} Q$ and P, Q are guarded, then $\Psi \rhd !P \dot{\sim} !Q$. A detailed account can be found in [Ben10, p. 388-391].

Bengtson's proof uses bisimulation up-to $\dot{\sim}$, with the candidate relation

$$\{(\Psi, R \mid !P,\ R \mid !Q) : \Psi \rhd P \dot{\sim} Q \wedge P, Q \text{ are guarded}\}$$

This means that in the simulation part of the proof, we must consider all transitions from $R \mid !P$, resulting in a transition inversion where four cases must be analysed (transitions from $R \mid !P$ can be derived via PAR, COM or their symmetric counterparts). Essentially, this inversion re-proves a special case of the more general result that bisimilarity is closed under parallel composition, and seems somewhat besides the point in a proof which is really about the replication operator, not the parallel operator.

We support this intuition by using the techniques introduced in Section 6.1 to give a simpler proof, which does not use a redundant parallel component in

the candidate relation and hence does not feature a rule inversion at all. Our proof constitutes 70 lines of Isabelle code; Bengtson's constitutes 214 lines.

The idea is to pick the candidate relation

$$\mathcal{R} \triangleq \{(\Psi,\ !P,\ !Q) : \Psi \rhd P \mathrel{\dot{\sim}} Q \wedge P, Q \text{ are guarded}\}$$

and the function

$$\mathcal{F} \triangleq \mathcal{U} \frown (\mathcal{C}_{\text{RES}} \circ \mathcal{C}_{\text{PAR}} \circ \mathcal{E}) \frown \mathcal{U}$$

In the simulation part of the proof, we apply the following lemma by Bengtson[2]:

Lemma 23. *(Lemma 28.29 from [Ben10]) If* $\Psi \rhd !P \xrightarrow{\alpha} P'$, $\Psi \rhd P \mathrel{\dot{\sim}} Q$, $\text{bn}(\alpha)\#\Psi, P, Q, \text{subj}(\alpha)$ *and* Q *is guarded, then there exists* Q', R, T *such that* $\Psi \rhd !Q \xrightarrow{\alpha} Q'$, $\Psi \rhd P' \mathrel{\dot{\sim}} R \mid !P$, $\Psi \rhd Q' \mathrel{\dot{\sim}} T \mid !Q$, $\Psi \rhd R \mathrel{\dot{\sim}} T$, $\mathsf{n}(R) \subseteq \mathsf{n}(P')$ *and* $\mathsf{n}(T) \subseteq \mathsf{n}(Q')$.

After applying Lemma 23, all that remains is to show that $(\Psi, P', Q') \in \mathcal{F}(\mathcal{R})$, which is simply a matter of rewriting using the structural congruence laws.

6.2.2 Idempotence of replication

We show that replication is idempotent, a structural congruence law that is new in the setting of psi-calculi, but familiar from the pi-calculus [Mil99, SW01].

Theorem 24. $\Psi \rhd !!P \mathrel{\dot{\sim}} !P$

Proof. By applying Theorem 18, choosing $\mathcal{R} \triangleq \{(\Psi, !!P, !P) : \textbf{true}\}$ and $\mathcal{F} \triangleq \mathcal{U} \frown (\mathcal{C}_{\text{RES}} \circ \mathcal{C}_{\text{PAR}} \circ \mathcal{E}) \frown \mathcal{U}$.

Respectfulness, static equivalence, extension and equivariance are trivial.

The right-to-left direction of the simulation proof goes by applying Lemma 23 to obtain R such that $\Psi \rhd !P \xrightarrow{\alpha} P' \mathrel{\dot{\sim}} R \mid !P$. By a simple derivation, $\Psi \rhd !!P \xrightarrow{\alpha} P' \mid !!P$. From there, it remains to be shown that $(\Psi, P' \mid !!P, P') \in \mathcal{F}(\mathcal{R})$, which follows by structural congruence.

In the left-to-right direction, an induction over the derivation of $\Psi \rhd !!P \xrightarrow{\alpha} P'$ is used. Each case is discharged by applying Lemma 23 and familiar algebraic laws. □

Corollary 25. $\Psi \rhd !P \mid !P \mathrel{\dot{\sim}} !P$

Proof. If P is guarded, $\Psi \rhd !P \mid !P \mathrel{\dot{\sim}} !P \mid !!P \mathrel{\dot{\sim}} !!P \mathrel{\dot{\sim}} !P$[3]. If P is not guarded, neither the LHS or RHS has any transitions or interesting frames. □

[2] Bengton's proof of Lemma 23 is 134 lines long. It is used both in our proof and Bengtson's proof that bisimilarity is closed under replication, and hence we include it in the line count of neither.

[3] This proof idea was suggested in conversation by Borgström and does not seem to occur in the literature.

The Isabelle proof of Theorem 24 is 314 lines long. Strikingly, there does not appear to have been any proof of the corresponding pi-calculus result in the literature until the publication of Sangiorgi and Rutten's recent book [SR11]; it is left as an exercise to the reader in [Mil99, SW01]. Sangiorgi and Rutten's proof technique is similar to ours.

For a comparison of different approaches to this kind of proof, we instead resort to Corollary 25, whose corresponding pi-calculus result has been proven twice by Milner [Mil91, Mil99], and once by Sangiorgi and Walker [SW01].

Milner's first proof uses the candidate relation

$$\{(\nu\widetilde{y})(!P \mid !P \mid Q),\ (\nu\widetilde{y})(!P \mid Q) : \mathbf{true}\}$$

While full details of the proof are not shown, the more complicated relation choice would certainly entail at least an induction over the length of $\widetilde{y}$ and an inversion for each parallel operator before we even get to the replication operators.

Milner's second proof uses the relation $\{(!P \mid Q,\ !P \mid !P \mid Q) : \mathbf{true}\}$ and bisimulation up-to structural congruence. Sangiorgi and Walker's proof instead uses the candidate relation $\{(!P \mid !P,\ !P) : \mathbf{true}\}$ and a choice of $\mathcal{F}$ which closes the relation under contexts and structural congruence, a very similar approach to our proof of Theorem 24.

The shortest of these proofs is Milner's first proof, since it is presented in the least amount of detail. Interestingly, while Milner's second proof is presented at a similar level of abstraction to Sangiorgi and Walker's, the proofs themselves are about the same length despite the simpler relation used by the latter. We conclude that in a pen- and paper setting, the choice of up-to techniques does not have as much impact as in a theorem prover setting. Pen- and paper proofs can often be made easier by using informal arguments such as "the other cases are similar", "as the reader may care to check" et cetera in place of up-to techniques. In a theorem prover, where every single detail must be considered, up-to techniques can provide a much needed substitute for such shortcuts.

6.2.3 Encoding of replication in higher-order calculi

We recall from Chapter 3 the result from higher-order psi-calculi that under certain assumptions on the calculus, replication can be encoded using the **run** construct. More precisely:

$$\Psi \rhd !P \dot{\sim} (\nu a)(\mathbf{run}\ M \mid (\!|\Psi^{M \Leftarrow P \mid \mathbf{run}\ M}|\!))$$

Here $\Psi^{M \Leftarrow P \mid \mathbf{run}\ M}$, pronounced the *characteristic assertion* of a clause $M \Leftarrow P \mid \mathbf{run}\ M$ is an assertion that entails this particular clause but otherwise has no impact on the assertion environment. a is a name that is fresh in P but not in M. For a more detailed account we refer to Chapter 3.

While perhaps not a very surprising result, the Isabelle proof is actually one of the longest proofs in the psi-calculi literature — a theory file consisting

of 8789 lines of code totalling 874.2 kB is dedicated entirely to it. Part of the reason why this proof is so long is the choice of candidate relation, which is the symmetric closure of the following:

$$\mathcal{R} \triangleq \{(\Psi, (\nu\widetilde{x})(Q \mid (\nu a)(\mathbf{run}\ M \mid (\!|\Psi^{M \Leftarrow P} \mid \mathbf{run}\ M|\!))), (\nu\widetilde{x})(Q \mid !P))\}$$

In order to avoid bogging down the presentation, the definition of $\mathcal{R}$ given above is not fully formal since it omits several freshness conditions and requisites on the characteristic assertion. Readers interested in the gory details are referred to the Isabelle sources [ÅP13].

Focusing on the right- to left direction of the simulation proofs, we must follow all transitions from $(\nu\widetilde{x})(Q \mid !P)$. The first step is an induction over the length of the binding sequence $\widetilde{x}$. In the base case, an inversion of the transition from $Q \mid !P$ follows, yielding four cases which must be analysed: Par, Com and their symmetric counterparts. In three of these cases, an induction over the derivation of the transition from $!P$ occurs, yielding five sub-cases: two each of Par and Com, plus Rep.

With such a convoluted proof structure, it should be easy to see why the proof is so long. Fortunately, we can significantly improve upon the situation using bisimulation up-to techniques and the following much simpler relation:

$$\mathcal{S} \triangleq \{(\Psi, (\nu a)(\mathbf{run}\ M \mid (\!|\Psi^{M \Leftarrow P} \mid \mathbf{run}\ M|\!)), !P)\}$$

The proof uses $\mathcal{F} \triangleq \mathcal{U} \frown (\mathcal{C}_{\text{Res}} \circ \mathcal{C}_{\text{Par}} \circ \mathcal{E}) \frown \mathcal{U}$, which should be familiar from the previous examples. Of the steps necessary with $\mathcal{R}$ as the relation choice discussed above, only the innermost induction over the derivation from $!P$ must now be considered. We are also relieved of the obligation to close $\mathcal{S}$ under symmetry. Without otherwise changing anything, this significantly shortens the proof to 3324 lines and 246.9 kB, a size decrease of about 60% and 70% respectively.

6.3 Conclusion

In this chapter, we have adapted Sangiorgi's bisimulation proof method to the setting of psi-calculi in a way that accounts for assertion environments. We have illustrated the usefulness of this proof method by showing how it can make proofs of known results significantly easier, and used it to prove a new result. We have also mechanised all of our definitions and theorems in Nominal Isabelle.

The shorter proofs are worthwhile to obtain even disregarding purely illustrative and aesthetic concerns because they make the proofs more maintainable. Just like program code, proof scripts require maintenance: definitions change, or new versions of Isabelle break backwards-compatibility. For every such change, all proofs must be re-checked, and shorter proofs are naturally easier to check.

For future work, we would like to apply these techniques to the extensions of psi-calculi discussed in this thesis. So far, these proofs have been carried out for the original psi-calculi. While we are able to use the same proof scripts for higher-order psi-calculi without changing a single line of code, different proofs of respectfulness for the up-to context techniques would be necessary for the extensions with broadcasts and negative premises.

We have so far only considered the closure under contexts for the operators of the finite subcalculi. For completeness it would be nice to also provide closure under replication contexts, though this omission is not hugely significant in practice — the most practically interesting context closures are those for restriction and parallel composition, since they are the only operators that occur on the right-hand side of the transition arrow in the operational semantics.

Bisimulation up-to techniques for weak bisimulation represents another interesting area to explore. Soundness is more delicate for weak bisimulation than for the strong case; for an example, weak bisimulation up-to $\dot{\approx}$ is unsound. Solutions have been proposed by Sangiorgi and Milner [SM92] and Pous [Pou05, Pou07] that could constitute a starting point for similar investigations in the field of psi-calculi.

Bibliography

[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.

[ÅP13] Johannes Åman Pohjola. Formalisations of psi-calculi extensions. `http://www.it.uu.se/research/group/mobility/theorem/psiextensions.tgz`, September 2013. Isabelle/HOL-Nominal formalisation of the definitions, theorems and proofs.

[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.

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

[BB90] Gerard Berry and Gerard Boudol. The chemical abstract machine. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '90, pages 81–94, New York, NY, USA, 1990. ACM.

[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.

[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*, May 2012. `http://afp.sf.net/entries/Psi_Calculi.shtml`, Formal proof development.

[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.

[BHJ+11] 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, November 2011.

[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.

[BIM88] Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced: preliminary report. In *Department of Computer Science, Cornell University*, pages 229–239, 1988.

[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.

[Bou89] Gérard Boudol. Towards a lambda-calculus for concurrent and communicating systems. In Josep Díaz and Fernando Orejas, editors, *TAPSOFT, Vol.1*, volume 351 of *Lecture Notes in Computer Science*, pages 149–161. Springer, 1989.

[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.

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

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

[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.

[CHM12] Andrew Cerone, Matthew Hennessy, and Massimo Merro. Modelling mac-layer communications in wireless systems. Technical Report TCD-CS-2012-17, Trinity College Dublin, August 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.

[CLN07] Rance Cleaveland, Gerald Lüttgen, and V. Natarajan. Priority and abstraction in process algebra. *Inf. Comput.*, 205(9):1426–1458, 2007.

[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.

[EM02] Cristian Ene and Traian Muntean. Testing theories for broadcasting processes. *Sci. Ann. Cuza Univ.*, 11:214–230, 2002.

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

[FH92] Matthias Felleisen and Robert Hieb. The revised report on the syntactic theories of sequential control and state. *Theor. Comput. Sci.*, 103(2):235–271, 1992.

[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.

[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.

[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.

[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.

[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.

[Hol93] Uno Holmer. Interpreting broadcast communication in sccs. In Eike Best, editor, *CONCUR*, volume 715 of *Lecture Notes in Computer Science*, pages 188–201. Springer, 1993.

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

[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.

[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.

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

[ISO03] ISO. Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling. ISO 11898-1, International Organization for Standardization, Geneva, Switzerland, 2003.

[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.

[LMS10] Ugo Dal Lago, Simone Martini, and Davide Sangiorgi. Light logics and higher-order processes. In Sibylle B. Fröschle and Frank D. Valencia, editors, *EXPRESS'10*, volume 41 of *EPTCS*, pages 46–60, 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.

[Mil90] Robin Milner. Functions as processes. In *Proceedings of the seventeenth international colloquium on Automata, languages and programming*, pages 167–180, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

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

[Mil99] Robin Milner. *Communicating and mobile systems: the pi-calculus*. Cambridge University Press, New York, NY, USA, 1999.

[MM12] Damiano Macedonio and Massimo Merro. A semantic analysis of wireless network security protocols. In Alwyn Goodloe and Suzette Person, editors, *NASA Formal Methods*, volume 7226 of *Lecture Notes in Computer Science*, pages 403–417. Springer, 2012.

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

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

[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.

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

[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.

[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.

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

[PBRÅP13] Joachim Parrow, Johannes Borgström, Palle Raabjerg, and Johannes Åman Pohjola. Higher-order psi-calculi. *Mathematical Structures in Computer Science*, FirstView:1–37, 6 2013.

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

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

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

[Plo83] Gordon D. Plotkin. An operational semantics for CSP. In A. Salwicki, editor, *Logics of Programs and Their Applications*, volume 148 of *Lecture Notes in Computer Science*, pages 250–252. Springer Berlin Heidelberg, 1983.

[Pou05] Damien Pous. Up-to techniques for weak bisimulation. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP*, volume 3580 of *Lecture Notes in Computer Science*, pages 730–741. Springer, 2005.

[Pou07] Damien Pous. New up-to techniques for weak bisimulation. *Theor. Comput. Sci.*, 380(1-2):164–180, 2007.

[Pra91] Kuchi 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] Kuchi V. S. Prasad. A calculus of value broadcasts. In *IN PARLE'93*, pages 69–4. Springer Verlag LNCS, 1993.

[Pra94] Kuchi 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] Kuchi V. S. Prasad. A calculus of broadcasting systems. *Science of Computer Programming*, 25(2-3):285–327, 1995.

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

[San93a] Davide Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, LFCS, University of Edinburgh, 1993. CST-99-93 (also published as ECS-LFCS-93-266).

[San93b] 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.

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

[SM92] Davide Sangiorgi and Robin Milner. The problem of "weak bisimulation up to". In Rance Cleaveland, editor, *CONCUR*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 1992.

[SR11] Davide Sangiorgi and Jan Rutten. *Advanced Topics in Bisimulation and Coinduction*. Cambridge University Press, New York, NY, USA, 1st edition, 2011.

[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.

[SW01] Davide Sangiorgi and David Walker. *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press, 2001.

[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.

[UK12] Christian Urban and Cezary Kaliszyk. General bindings and alpha-equivalence in nominal isabelle. *Logical Methods in Computer Science*, 8(2), 2012.

[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.

[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.

[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.

[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.

Recent licentiate theses from the Department of Information Technology

2013-003 Daniel Elfverson: *On Discontinuous Galerkin Multiscale Methods*

2013-002 Marcus Holm: *Scientific Computing on Hybrid Architectures*

2013-001 Olov Rosén: *Parallelization of Stochastic Estimation Algorithms on Multicore Computational Platforms*

2012-009 Andreas Sembrant: *Efficient Techniques for Detecting and Exploiting Runtime Phases*

2012-008 Palle Raabjerg: *Extending Psi-calculi and their Formal Proofs*

2012-007 Margarida Martins da Silva: *System Identification and Control for General Anesthesia based on Parsimonious Wiener Models*

2012-006 Martin Tillenius: *Leveraging Multicore Processors for Scientific Computing*

2012-005 Egi Hidayat: *On Identification of Endocrine Systems*

2012-004 Soma Tayamon: *Nonlinear System Identification with Applications to Selective Catalytic Reduction Systems*

2012-003 Magnus Gustafsson: *Towards an Adaptive Solver for High-Dimensional PDE Problems on Clusters of Multicore Processors*

2012-002 Fredrik Bjurefors: *Measurements in Opportunistic Networks*

2012-001 Gunnika Isaksson-Lutteman: *Future Train Traffic Control – Development and deployment of new principles and systems in train traffic control*

UPPSALA
UNIVERSITET

Department of Information Technology, Uppsala University, Sweden