UPPSALA
UNIVERSITET

IT Licentiate theses
2014-007

Advancing Concurrent System Verification

Type based approach and tools

RAMŪNAS GUTKOVAS

UPPSALA UNIVERSITY
Department of Information Technology

Advancing Concurrent System Verification

Type based approach and tools

Ramūnas Gutkovas
ramunas.gutkovas@it.uu.se

October 2014

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

Concurrent systems, i.e. systems of parallel processes, are nearly ubiquitous and verifying the correctness of such systems is becoming an important subject. Many formalisms were invented for such purpose, however, new types of systems are introduced and there is a need for handling larger systems. One examples is wireless sensor networks that are being deployed in increasing numbers in various areas; and in particular safety-critical areas, e.g., bush fire detection. Thus, ensuring their correctness is important.

A process calculus is a formal language for modeling concurrent systems. The pi-calculus is a prominent example of such a language featuring message-passing concurrency. Psi-calculi is a parametric framework that extends the pi-calculus with arbitrary data and logics. Psi-calculi feature a universal theory with its results checked in an automated theorem prover ensuring their correctness.

In this thesis, we extend psi-calculi expressiveness and modeling precision by introducing a sort system and generalised pattern matching. We show that the extended psi-calculi enjoy the same meta-theoretical results.

We have developed the Pwb, a tool for the psi-calculi framework. The tool provides a high-level interactive symbolic execution and automated behavioral equivalence checking. We exemplify the use of the tool by developing a high-level executable model of a data collection protocol for wireless sensor networks.

We are the first to introduce a session types based system for systems with unreliable communication. Remarkably, we do not need to add specific extensions to the types to accommodate such systems. We prove the standard desirable properties for type systems hold also for our type system.

Acknowledgements

First and foremost, I am grateful to my advisors Johannes Borgström and Björn Victor for providing support and reading numerous drafts of this thesis and of course producing papers with me.

I would also like to thank my co-authors Dimitris Kouzapas, Simon J. Gay, Johannes Åman Pohjola, Ioana Rodhe, and Joachim Parrow as without them this thesis would not be possible.

The work is supported by the ProFun project. The work on Paper II was supported by a Short-Term Scientific Mission grant from COST Action IC1201 (Behavioural Types for Reliable Large-Scale Software Systems).

Last but not least, I would like to thank my Mom and especially Mia for support.

Contents

Chapter 1

Introduction

Concurrent systems consist of computer programs running in parallel. They are becoming ubiquitous and consequentially their correctness is becoming a more pressing issue. It has been evident for some time that the computer processor vendors can no longer make processors execute programs faster by simply decreasing the size of their basic building blocks, which has been the standard for many years. Instead, vendors are simply cramming more processors, a.k.a. cores, on the same chip. The only way to make programs run faster and increasingly power efficient is to design them to run concurrently on multiple cores.

New kinds of concurrent systems are emerging. A wireless sensor network (WSN) consists of many small gadgets, called nodes, equipped with a processor, sensor, actuator, and antenna. Nodes sense and control their surrounding environment. They can be used to transmit their collected data to a central base station, and also receive commands. WSN's utility lies in the cheap nodes that can be distributed in large numbers. For example, WSN can be used to control temperature and ventilation in a building, monitor wide geographic areas for natural disasters like bush fires, earthquakes, etc.

Concurrent systems are deployed in increasing numbers and wide areas: they fly planes, drive cars, monitor power stations, and so on. Needless to say, their correct behavior is essential.

Developing correctly functioning computer programs is a difficult task. This is evident, as virtually everyone who has used a computer has experienced 'hanging' programs, wrong results or behavior. The heart of the problem is the complexity of software: the sheer amount of cases a programmer has to consider mentally might be beyond what is humanly possible.

Concurrency adds another dimension of complexity to an already complicated task. A completely new class of bugs arise in such systems, e.g., deadlock, livelock, data race, etc. To take an example, a system deadlocks when at least two parts of it wait mutually for each other's completion. What is more, the inherent discreteness and typically large size of such systems destroy any hope of ensuring the complete correctness by means of testing.

Various concurrent system verification methods have been introduced over the years to deal with complexity and ensure some form of correctness. It is by no means a solved problem. New types of systems are being used where standard methods are not applicable or scale poorly to the sizes of modern systems.

Process calculi are, typically small, formal languages with their syntax and semantics defined using mathematics for precise reasoning about concurrent systems. They have been around since the 1980s; the most notable example is the pi-calculus introduced by Milner, Parrow and Walker [MPW92a, MPW92b]. The pi-calculus is a foundational language for modeling message-passing concurrency with small number of primitives, well developed theory, and a number of tools. It comes with the well-defined notion of the behavioral equivalence allowing for compositional reasoning. That is, it allows us to tell when two processes behave the same and thus can be substituted for one another; for example, we can replace parts of a system with a simpler implementation, that is behaviorally equivalent, and still be guarantied that it will continue working.

The pi-calculus however powerful is not suitable for real world systems and there are many extensions of it with data structures and logics for a particular use case. The psi-calculi framework by Bengtson et al. [BJPV11] is an extension of the pi-calculus providing arbitrary data structures, and logics allowing more concise modeling and unifying framework for many disparate process calculi. Many of the different extensions of the pi-calculus, including the spi-calculus [AG97], the fusion calculus [WG05], and the polyadic synchronisation pi-calculus [CM02], can be directly represented as a psi-calculus.

This thesis is about extending existing frameworks to be more expressive and providing tools, which are crucial for modeling. It is laid out in three papers (listed in Section 1.2 and description of my personal contributions in Section 1.3).

1.1 Outline

In Chapter 2, we introduce the background theory to the contributions chapter (Chapter 3). Section 2.1 is a brief section with essential definitions to nominal techniques for abstract syntax that are used to define psi-calculi syntax and semantics. We give a brief introduction to the pi-calculus and its variants in Section 2.2. We also introduce its behavioral equivalences (Section 2.2.5) and a type system (Section 2.2.6). Readers familiar with the pi-calculus can safely skip Section 2.2. In Section 2.3, we introduce the psi-calculi framework, an extension of the pi-calculus. We define its parameters, syntax, semantics and present the bisimulation theory.

In Chapter 3, we recapitulate the main contributions of this thesis. We developed a tool, the Psi-calculi Workbench (Section 3.1 and Paper I). The tool provides symbolic execution and automated behavioral equivalence checking based on the psi-calculi framework for modeling and verifying concurrent systems. The tool implements both the usual point-to-point and unreliable broadcast semantics. To exhibit the use of the tool, we exploit the high-level abstraction modeling and symbolic execution to obtain an executable and concise model of the TAG data collection protocol [MFHH02] for WSN.

In Section 3.3, we summarise the broadcast session types of Paper II. We are the first to adapt the elegant idea of session types to the unreliable broadcast systems. Session types provide a simple and concise language for specification of wide class of communication protocols and surprisingly the type system is not more complicated than the standard.

In Section 3.4, we extended the psi-calculi framework with the abstract notion of pattern-matching that increases expressivity of psi-calculi, for example, it allows us to express the pattern-matching spi-calculus [HJ06] in the framework. In addition in Section 3.5, we introduce a simple and flexible sort system for psi-calculi; it makes the model more precise by disallowing nonsensical terms. We show that our extensions are well defined: the meta-theoretical results hold such as behavioral equivalence is congruence. These two section is the introduction of Paper III.

Finally, we end the thesis with related work Chapter 4, future work Chapter 5, and conclude in Chapter 6.

1.2 Included Papers

This thesis is a compilation of the following list of papers.

Paper I. J. Borgström, R. Gutkovas, I. Rodhe, and B. Victor. *A parametric tool for applied process calculi*. Accepted to the special issue of "Application of Concurrency to System Design" in the ACM Transactions on Embedded Computing Systems (TECS) journal. To appear.

An abridged version of the paper is published in J. Carmona, M. T. Lazarescu, and M. Pietkiewicz-Koutny, editors, Application of Concurrency to System Design (ACSD), 2013 13th International Conference, pages 180-185, Barcelona, Spain, July 2013. IEEE.

Paper II. D. Kouzapas, R. Gutkovas, and S. J. Gay. *Session types for broadcasting*. In A. F. Donaldson and V. T. Vasconcelos, editors, Proceedings 7th Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software, Grenoble, France, 12 April 2014, volume 155 of Electronic Proceedings in Theoretical Computer Science, pages 25-31. Open Publishing Association, 2014.

Paper III. J. Borgström, R. Gutkovas, J. Parrow, B. Victor, and J. Å. Pohjola. *A sorted semantic framework for applied process calculi*. To be submitted.

An extended abstract version of the paper is published in M. Abadi and A. Lluch Lafuente, editors, Trustworthy Global Computing, Lecture Notes in Computer Science, volume 8358, pages 103-118. Springer International Publishing, 2014.

1.3 My Contributions

Paper I. I am the author of the Psi-calculi Workbench and I did most of the implementation. Also, I provided text in the paper for the examples and tool description.

Paper II. The idea of applying session types to unreliable broadcast systems is mine. I contributed to the definition of the type systems and the translation. I invented the reduction semantics for the unicast and broadcast semantics. I also provided the proofs.

Paper III. My contributions to Paper III are mainly half of the non-mechanised proofs found in the appendix of the paper.

Chapter 2

Background

This chapter presents the background material for the contributions chapter (Chapter 3). In Section 2.1, we describe the nominal techniques used to represent the syntax of the calculi described. We introduce the pi-calculus (Section 2.2), a language for concurrent systems, its syntax (Section 2.2 and Section 2.2.4) and semantics. We introduce its behavioral equivalence (Section 2.2.5) and a rudimentary type system (Section 2.2.6). We later extend this introduction to the full psi-calculi framework in the same vein (Section 2.3).

2.1 Nominal Datatypes

Psi-calculi are based on nominal data types. A nominal data type is used to represent abstract syntax trees with binding constructs. It is similar to a traditional data type, but can contain binders and identifies alpha-variants of terms. Formally, the only requirements are related to the treatment of the atomic symbols called names as explained below. Here, we consider sorted nominal datatypes, where names and elements of the data type may have different sorts.

A sorted *nominal set* [GP99, Pit03] 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 $n(T)$, 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 $a \notin n(T)$, and extend this to finite sets and tuples. A function f is *equivariant* if $(a\ b) \cdot (f(T)) = f((a\ b) \cdot T)$ for every a, b and T; a relation $\mathcal{R}$ is equivariant if $x \mathrel{\mathcal{R}} y$ implies that $(a\ b) \cdot x \mathrel{\mathcal{R}} (a\ b) \cdot y$ holds; and a constant symbol C is equivariant if $(a\ b) \cdot \mathrm{C} = \mathrm{C}$. A *nominal data type* is a nominal set together with some equivariant functions on it, for instance a substitution function.

2.2 Pi-calculus

The reader familiar with the pi-calculus may skip to Section 2.3.

The pi-calculus is a mathematical formalism for modelling concurrent systems introduced in the '90s by Milner, Parrow and Walker [MPW92a, MPW92b]. It is extends CCS [Mil89] with mobility. Since then their work on the pi-calculus garnered numerous citations[1]. The pi-calculus is the basis for many concurrency theories and is typically taken as the point of departure when constructing new process calculi.

The theory has been reformalised to handle different modes of concurrency: asynchrony [HT91], broadcast [EM01], multicast [HYC08]. And, extended with more advanced data structures, e.g., [AG97] and [AF01].

Here, we give a brief introduction to the pi-calculus and its theory, and some of the extensions. For a not so brief introduction, the original papers are still authoritative [Mil92, MPW92b], the textbook on the pi-calculus by Sangiorgi and Walker [SW01] is a comprehensive overview, and also Parrow's introduction to the pi-calculus [Par01] contains a gentle introduction to many variants of the theory.

2.2.1 Basics

In this section we present a sublanguage of the pi-calculus.

Pi-calculus processes represent concurrent agents that can send and receive messages, synchronously. This notion of concurrency is also referred to as message passing. The only data structure that the agents communicate is that of a name. Names represent both channels and variables. Since agents can transmit channels and create names, the interconnections of a process may change as the process evolves.

Formally, the pi-calculus agents have the following forms.

Definition 1 (Pi-calculus processes (agents))**.**

$$\begin{array}{rcll} P, Q & ::= & a(x).P & \text{(Input Prefix)} \\ & \mid & \overline{a}b.P & \text{(Output Prefix)} \\ & \mid & \mathbf{0} & \text{(Inactive process)} \\ & \mid & P \mid Q & \text{(Parallel)} \\ & \mid & (\nu x)P & \text{(Restriction)} \\ & \mid & !P & \text{(Replication)} \end{array}$$

In the processes $a(x).P$ and $(\nu x)P$ are binding structures, that is, the name x binds into P. The set of processes form a nominal set (Section 2.1) and thus they are identified up to α-equivalence.

Pi-calculus processes represent some form of behaviour, similarly to the lambda-calculus for sequential programs.

Intuitively, the behavior, or semantics, of a process $a(x).P$ is that it inputs a name y on channel a which is then substituted for x in P, and continues as $P\{y/x\}$.

[1] According to the Google Scholar service the number of citatations is 4179 for [Mil92] on May 12th, 2014.

Dually, the process $\overline{a}b.P$ sends the name b over the channel a and continues as P. The inactive process $\mathbf{0}$ as the name suggests has no behavior. The process $P \mid Q$ represent process P and Q behaving concurrently and potentially interacting (we will see examples of this). The process $(\nu x)P$ behaves as P, but the scope of the name x is restricted only to P. Here, x represents a secret name which other processes cannot use, but they can get the hold of it if P chooses to send it. Finally, the process $!P$ can spawn an unbounded number of copies of P in parallel, in other words, replicate.

Having this in mind, we can now give the intuition behind the fundamental behavior of message passing concurrent system, namely, communication. For example, in the following the process $\overline{a}b.b(y).\mathbf{0}$ sends the name b to the process $a(x).\overline{x}c.\mathbf{0}$:

$$a(x).\overline{x}c.\mathbf{0} \mid \overline{a}b.b(y).\mathbf{0} \quad \rightarrow \quad \overline{b}c.\mathbf{0} \mid b(y).\mathbf{0},$$

and in turn

$$\overline{b}c.\mathbf{0} \mid b(y).\mathbf{0} \quad \rightarrow \quad \mathbf{0} \mid \mathbf{0}$$

by sending c on channel b.

Another aspect of the pi-calculus, is that restricted names can be transmitted as well. In the following the restricted name b is *extruded* to both agents by communication:

$$a(x).\overline{x}c.\mathbf{0} \mid (\nu b)(\overline{a}b.b(y).\mathbf{0}) \quad \rightarrow \quad (\nu b)(\overline{b}c.\mathbf{0} \mid b(y).\mathbf{0}),$$

where the scope of (νb) follows the name b, and

$$(\nu b)(\overline{b}c.\mathbf{0} \mid b(y).\mathbf{0}) \quad \rightarrow \quad (\nu b)(\mathbf{0} \mid \mathbf{0}).$$

But if were to add $b(y).\mathbf{0}$ in parallel to the source process above, the following is *not* a valid transition

$$b(y).\mathbf{0} \mid (\nu b)(\overline{b}c.\mathbf{0} \mid b(y).\mathbf{0}) \quad \rightarrow \quad \mathbf{0} \mid (\nu b)(\mathbf{0} \mid b(y).\mathbf{0}),$$

because the restricted names are distinct from all other names in an agent. As you can see, intuitively with νb we can create private channels that are invisible to the outside processes.

2.2.2 Reduction Semantics

A popular and perhaps the most straightforward way to formalise the behavior discussed in the previous section is to use term rewriting. The resulting semantics is usually called reduction semantics.

The reductions are defined up to a congruence relation on agents called structural congruence. A structural congruence is usually kept small and includes the immediate laws that we require of agents, e.g., that the parallel composition is commutative.

$$[\textsc{pi-comm-red}]\ \frac{}{a(x).P \mid \overline{a}b.Q \to P\{b/x\} \mid Q}$$

$$[\textsc{pi-par-red}]\ \frac{P \to P'}{P \mid Q \to P' \mid Q} \qquad [\textsc{pi-res-red}]\ \frac{P \to P'}{(\nu x)P \to (\nu x)P'}$$

$$[\textsc{pi-cong-red}]\ \frac{P \equiv P' \to Q' \equiv Q}{P \to Q}$$

Figure 2.1: Reduction Semantics for the pi-calculus

Definition 2 (Structural Congruence)**.** Structural congruence $\equiv$ is the least congruence satisfying the following laws:

$$\begin{array}{rcll}
P \mid Q & \equiv & Q \mid P & & \text{(Parallel Commutativity)}\\
(P \mid Q) \mid R & \equiv & P \mid (Q \mid R) & & \text{(Parallel Associativity)}\\
P \mid \mathbf{0} & \equiv & P & & \text{(Parallel Unit)}\\
!P & \equiv & P \mid !P & & \text{(Replication)}\\
(\nu x)\mathbf{0} & \equiv & \mathbf{0} & & \text{(Restriction Intro)}\\
(\nu x)(\nu y)P & \equiv & (\nu y)(\nu x)P & & \text{(Restriction Commute)}\\
(\nu x)(P \mid Q) & \equiv & P \mid (\nu x)Q & \text{if } x \# P & \text{(Restriction Extrusion)}
\end{array}$$

Definition 3 (Reduction relation)**.** The reduction relation is defined to be the least relation satisfying the rules in Figure 2.1.

For example, let us compute the reduction of the agent

$$P = b(x).\mathbf{0} \mid a(x).\overline{x}c.\mathbf{0} \mid (\nu b)(\overline{a}b.b(y).\mathbf{0}).$$

First, we use the [pi-cong-red] rule to compute the following by using the commutativity and associativity of parallel, and scope extrusion since we have $b \# a(x).\overline{x}c.\mathbf{0}$

$$P \equiv P' = (\nu b)(a(x).\overline{x}c.\mathbf{0} \mid \overline{a}b.b(y).\mathbf{0}) \mid b(x).\mathbf{0}.$$

By applying first [pi-par-red] and then [pi-res-red] we obtain

$$a(x).\overline{x}c.\mathbf{0} \mid \overline{a}b.b(y).\mathbf{0}$$

to which we can apply [pi-comm-red] and derive

$$a(x).\overline{x}c.\mathbf{0} \mid \overline{a}b.b(y).\mathbf{0} \to \overline{b}c.\mathbf{0} \mid b(y).\mathbf{0}.$$

Finally, we derive

$$b(x).\mathbf{0} \mid a(x).\overline{x}c.\mathbf{0} \mid (\nu b)(\overline{a}b.b(y).\mathbf{0}) \to (\nu b)(\overline{b}c.\mathbf{0} \mid b(y).\mathbf{0}) \mid b(x).\mathbf{0}.$$

The reduction relation is perhaps the most intuitive way of giving semantics to a process calculus, however, it is not always adequate. The structural congruence typically adds another level of induction in proofs. This can be a drawback when using automated tools such as proof assistants.

2.2.3 Structural Operational Semantics

In this section we present semantics for pi-calculus based on Plotkin's [Plo81] structured operational semantics. This style of semantics is given by a set of inductive rules on the structure of the processes. This allows for convenient inductive arguments to be used when proving properties on semantics. The rules define a labelled transition system of the form

$$P \xrightarrow{\alpha} P'$$

where P transitions to P' with the observable action α.

It is not only the convenient inductive arguments that make this style of semantics desirable, but also labelled semantics model the notion of an external observer that observes an action α. In this sense, intuitively we may peek at what happened within a process. This is useful when defining observational equivalences (Section 2.2.5).

Also, the observations allow to model an open-ended system. For example, suppose we are modelling a memory architecture in a computer system and we are interested in modeling only the memory unit. In this respect, we do not need to model the CPU sending commands since the external observer conceptually sends commands by observing various actions emanating from the model of a memory unit.

In order to define labelled semantics, we first define the possible observations that one can make on the pi-calculus agents, namely, the actions.

Definition 4 (Actions)**.**

$$\begin{array}{rcll}
\alpha & ::= & \tau & \text{(silent action)} \\
& \mid & ab & \text{(input action)} \\
& \mid & \overline{a}b & \text{(output action)} \\
& \mid & \overline{a}(\nu b)b & \text{(bound output action)}
\end{array}$$

The name b is bound in the bound output action. We also define $bn(\alpha)$ to be $\{b\}$ if $\alpha = \overline{a}(\nu b)b$ and otherwise $\emptyset$.

The silent action τ denotes internal activity, that is, communication. The input action ab denotes an input of the name b over the channel a, dually the output action $\overline{a}b$ denotes an output of the name b over the channel a. The bound output action $\overline{a}(\nu b)b$ denotes the sending of a restricted name b on channel a.

$$[\textsc{Pi-In}]\ \frac{}{a(x).P \xrightarrow{ay} P\{y/x\}} \qquad [\textsc{Pi-Out}]\ \frac{}{\bar{a}b.P \xrightarrow{\bar{a}b} P}$$

$$[\textsc{Pi-Par}]\ \frac{P \xrightarrow{\alpha} P' \qquad bn(\alpha)\ \#\ Q}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad [\textsc{Pi-Par-S}]\ \frac{Q \xrightarrow{\alpha} Q' \qquad bn(\alpha)\ \#\ P}{P \mid Q \xrightarrow{\alpha} P \mid Q'}$$

$$[\textsc{Pi-Open}]\ \frac{P \xrightarrow{\bar{a}x} P'}{(\nu x)P \xrightarrow{\bar{a}(\nu x)x} P'} \qquad [\textsc{Pi-Scope}]\ \frac{P \xrightarrow{\alpha} P' \qquad x\ \#\ \alpha}{(\nu x)P \xrightarrow{\alpha} (\nu x)P'}$$

$$[\textsc{Pi-Com}]\ \frac{P \xrightarrow{\bar{a}b} P' \qquad Q \xrightarrow{ab} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad [\textsc{Pi-Com-S}]\ \frac{P \xrightarrow{ab} P' \qquad Q \xrightarrow{\bar{a}b} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$[\textsc{Pi-Close}]\ \frac{P \xrightarrow{\bar{a}(\nu b)b} P' \qquad Q \xrightarrow{ab} Q'}{P \mid Q \xrightarrow{\tau} (\nu b)(P' \mid Q')} \qquad [\textsc{Pi-Close-S}]\ \frac{P \xrightarrow{ab} P' \qquad Q \xrightarrow{\bar{a}(\nu b)b} Q'}{P \mid Q \xrightarrow{\tau} (\nu b)(P' \mid Q')}$$

$$[\textsc{Pi-Rep}]\ \frac{P \mid !P \xrightarrow{\alpha} P'}{P \xrightarrow{\alpha} P'}$$

Figure 2.2: Structural Operational Semantics of the pi-calculus.

Definition 5 (Transition Relation)**.** The transition relation $\rightarrow$ on the pi-calculus agents P and Q, and an action α is defined as the least relation satisfying the rules in Figure 2.2. We write $P \xrightarrow{\alpha} Q$ for $(P, \alpha, Q) \in \rightarrow$, and say that P transitions to Q on α.

To give some intuition behind the rules, we derive a transition of the following agent (cf. Section 2.2)

$$b(x).\mathbf{0} \mid (a(x).\bar{x}c.\mathbf{0} \mid (\nu b)(\bar{a}b.b(y).\mathbf{0})).$$

First, we apply the [Pi-Par-S] rule to give us the left hand side of the arrow:

$$a(x).\bar{x}c.\mathbf{0} \mid (\nu b)(\bar{a}b.b(y).\mathbf{0}).$$

The above agent looks like it can communicate on a, but the object of the prefix of the agent on the right of $\mid$ is under restriction, thus we we apply the [Pi-Close-S] rule here to get the two agents on the left hand side of the arrow:

$$a(x).\bar{x}c.\mathbf{0}$$

and

$$(\nu b)(\bar{a}b.b(y).\mathbf{0}).$$

To the first one we apply [Pi-In] and derive

$$a(x).\bar{x}c.\mathbf{0} \xrightarrow{ab} \bar{b}c.\mathbf{0}$$

in anticipation that the agent will receive b. To the second we first apply [Pi-Open] to get

$$\bar{a}b.b(y).\mathbf{0},$$

and finally derive by [Pi-Out]

$$\bar{a}b.b(y).\mathbf{0} \xrightarrow{\bar{a}b} b(y).\mathbf{0}.$$

By filling in the right hand side of the arrows,

$$b(x).\mathbf{0} \mid (a(x).\bar{x}c.\mathbf{0} \mid (\nu b)(\bar{a}b.b(y).\mathbf{0})) \xrightarrow{\tau} b(x).\mathbf{0} \mid (\nu b)(\bar{b}c.\mathbf{0} \mid b(y).\mathbf{0}).$$

We explicitly write out the above example as a derivation tree as follows:

$$\text{[Pi-Par-S]}\ \dfrac{\text{[Pi-Close-S]}\ \dfrac{\text{[Pi-In]}\ \dfrac{}{a(x).\bar{x}c.\mathbf{0} \xrightarrow{ab} \bar{b}c.\mathbf{0}} \qquad \text{[Pi-Open]}\ \dfrac{\text{[Pi-Out]}\ \dfrac{}{\bar{a}b.b(y).\mathbf{0} \xrightarrow{\bar{a}b} b(y).\mathbf{0}}}{(\nu b)(\bar{a}b.b(y).\mathbf{0}) \xrightarrow{\bar{a}(\nu b)b} b(y).\mathbf{0}}}{a(x).\bar{x}c.\mathbf{0} \mid (\nu b)(\bar{a}b.b(y).\mathbf{0}) \xrightarrow{\tau} (\nu b)(\bar{b}c.\mathbf{0} \mid b(y).\mathbf{0})}}{b(x).\mathbf{0} \mid (a(x).\bar{x}c.\mathbf{0} \mid (\nu b)(\bar{a}b.b(y).\mathbf{0})) \xrightarrow{\tau} b(x).\mathbf{0} \mid (\nu b)(\bar{b}c.\mathbf{0} \mid b(y).\mathbf{0})}$$

In contrast to the reduction semantics discussed earlier, the scope extension law is part of the derivation that uses the bound output label captured by the rules [Pi-Close] and [Pi-Close-S]. The τ transitions coincide with the reductions up to structural congruence: $P \xrightarrow{\tau} P'$ if and only if $P \rightarrow P'' \equiv P'$.

We defined a structural operational semantics without taking the structural congruence as a primitive. A way to be convinced that the definition is correct is to check that the structural congruence laws are derivable.

One of the benefits of this style of semantics is that we don't need to use a structural congruence. This gives the ability to use single induction on the rules to reason about the semantic properties.

2.2.4 Language Variants

The pi-calculus that we described in Section 2.2.1 is quite small. The more typical presentation of the pi-calculus includes more constructs.

Definition 6 (Pi-calculus Agents with extensions)**.** We extend Definition 1 with the following forms.

$$\begin{array}{ll} \textbf{if } a = b \textbf{ then } P & \text{(Match)} \\ \textbf{if } a \neq b \textbf{ then } P & \text{(Mismatch)} \\ P + Q & \text{(Sum)} \end{array}$$

$$[\textsc{Pi-Match}]\ \frac{P \xrightarrow{\alpha} P'}{\textbf{if } a = a \textbf{ then } P \xrightarrow{\alpha} P'} \qquad [\textsc{Pi-Mismatch}]\ \frac{P \xrightarrow{\alpha} P' \qquad a \neq b}{\textbf{if } a \neq b \textbf{ then } P \xrightarrow{\alpha} P'}$$

$$[\textsc{Pi-Sum}]\ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad [\textsc{Pi-Sum}_2]\ \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

Figure 2.3: Operational semantics of the pi-calculus with extensions. The rules in this figure are added to Figure 2.2.

The intuition behind (Match) and (Mismatch) is that they behave as P whenever the names either match or not. (Sum) implements what could be called global choice. The agent $P+Q$ can behave as either P or Q non-deterministically.

Definition 7 (Extended Transition Relation)**.** The transition relation is defined to be the least relation satisfying the rules in Figure 2.3 and Figure 2.3.

For example, we could implement the more usual if-then-else construct familiar from programming languages as follows

$$\textbf{if } a = b \textbf{ then } P \textbf{ else } Q \overset{\text{def}}{=} \textbf{if } a = b \textbf{ then } P \ + \ \textbf{if } a \neq b \textbf{ then } Q.$$

Having the (Sum) operator is important, since models of systems often contain fair amounts of non-determinism and the sum is very natural way of capturing that. Another reason to include (Sum) is to allow for easy encoding of non-deterministic automata.

2.2.5 Behavioral Equivalence

We have defined what it means to observe a process by observing its actions. This means that we can formally compare the behavior of processes, and, if we cannot distinguish them, we can say that they are behaviorally equivalent. Behavioral equivalence is a powerful way of reasoning about a system: it gives us the ability to replace processes with processes that are behaviorally indistinguishable but perhaps less complex.

The standard notion of behavioral equivalence in the pi-calculus is what it is called strong bisimulation.

Definition 8 ((Strong) Bisimulation)**.** A binary relation R is a bisimulation if it is *symmetric* and for every $(P,Q) \in R$, we have

$$(\forall \alpha, P') \quad P \xrightarrow{\alpha} P' \wedge bn(\alpha) \# Q \implies \exists Q'.Q \xrightarrow{\alpha} Q' \wedge (P',Q') \in R$$

At first it is a quite unusual definition due to it circularity, but it is very natural: one process mimics every actions of another at every step and their continuations are able to do the same, and vice versa. The freshness condition ensures that a newly opened name by P is different from the names in Q and that Q' can only simulate P' using this name by also opening it.

Definition 9 ((Strong) Bisimilarity)**.** P and Q are bisimilar, written $P \dot{\sim} Q$, if and only if there is a bisimulation relation R such that $(P, Q) \in R$.

So two processes are bisimilar if we can find an appropriate relation. As an example, let us prove that the following two processes are bisimilar

$$P = a(x) \mid \overline{b}b \;\; \dot{\sim} \;\; a(x).\overline{b}b + \overline{b}b.a(x) = Q.$$

We first propose a candidate relation R:

$$\begin{array}{rcll} R & = & \{ & (a(x) \mid \overline{b}b,\;\; a(x).\overline{b}b + \overline{b}b.a(x)), \\ & & & (\mathbf{0} \mid \overline{b}b,\;\; \overline{b}b), \\ & & & (a(x) \mid \mathbf{0},\;\; a(x)) \\ & & & (\mathbf{0} \mid \mathbf{0},\;\; \mathbf{0}) \quad \} \cup \text{ symmetric version of this.} \end{array}$$

Note that the elements are obtained essentially by following all possible derivations of P and Q. It is then easy to see that all elements transition into the relation, e.g., the agent $\mathbf{0} \mid \overline{b}b \xrightarrow{\overline{b}b} \mathbf{0} \mid \mathbf{0}$ and its pair $\overline{b}b \xrightarrow{\overline{b}b} \mathbf{0}$ result in R, that is, $(\mathbf{0} \mid \mathbf{0}, \mathbf{0}) \in R$. Thus we proved that P and Q are bisimilar. Q.e.d.

Remark 1. It is easy to see that $\dot{\sim}$ is an equivalence relation, but it is not a congruence. For example, let us take the same two processes,

$$P = a(x) \mid \overline{b}b \;\; \dot{\sim} \;\; a(x).\overline{b}b + \overline{b}b.a(x) = Q,$$

but

$$c(a).P \not\dot{\sim} c(a).Q$$

since we could substitute b for a in both P and Q and then P would have more behavior, namely, an extra τ transitions.

However, the characterisation of the largest congruence is quite simple: it is bisimilarity closed under all substitutions.

Definition 10 (Congruence)**.** Let $P \sim Q$ mean, for all substitutions σ, $P\sigma \dot{\sim} Q\sigma$.

The relation $\sim$ is a congruence.

The strong bisimulation is sometimes too strong for modeling systems, because it is a common case that computation is done by internal communication. Think of a task being spawned to compute a value. What we would like is to ignore all these internal actions. This leads to the notion of weak bisimulation. We write $\rightarrow^*$ for reflexive and transitive closure of $\xrightarrow{\tau}$. Let us define $\overset{\hat{\alpha}}{\Rightarrow}$ as $\rightarrow^* \xrightarrow{\alpha} \rightarrow^*$ if $\alpha \neq \tau$, and as $\rightarrow^*$ otherwise.

Definition 11 (Weak bisimulation)**.** The binary relation R is a *weak* bisimulation if it is *symmetric* and for every $(P, Q) \in R$, we have

$$(\forall \alpha, P') \;\; P \xrightarrow{\alpha} P' \wedge bn(\alpha) \# Q \implies \exists Q'.Q \overset{\hat{\alpha}}{\Rightarrow} Q' \wedge (P', Q') \in R$$

Definition 12 (Weak Bisimilarity)**.** P and Q are weakly bisimilar, written $P \approx Q$, if and only if there is a weak bisimulation R such that $(P, Q) \in R$.

Weak bisimilarity is not congruence for the same reason, and we can obtain a congruence in exactly the same way.

Definition 13 (Weak Congruence)**.** Define $P \approx Q$ as for all substitutions σ, $P\sigma \approx Q\sigma$. The binary relation $\approx$ is a *weak* congruence.

Strong and weak bisimulations are well-behaved equivalences: they have an simple characterisation of a congruence. Also they are quite intuitve. We use them as a basis for the psi-calculi behvariol equivalences (Section 2.3.3) and in the Psi-calculi Workbench (Section 3.1).

2.2.6 Type System

Milner [Mil93] introduced the first type system for the pi-calculus he named it as sort system. He considered an extension of the pi-calculus with polyadic communication.

Definition 14 (Polyadic pi-calculus agents)**.** The *polyadic* pi-calculus agents are the agents in Definition 1 with the input and output prefixes replaced with the following

$$\begin{array}{ll} a(x_1, \ldots, x_n).P & \text{(Polyadic Input Prefix)} \\ \overline{a}\langle b_1, \ldots, b_m\rangle.P & \text{(Polyadic Output Prefix)} \end{array}$$

where $x_1, \ldots, x_n$ are binders and pairwise distinct.

The semantics of communication is an extension of monadic communication.

Definition 15. The reduction relation for the polyadic pi-calculus is the least relation satisfying the rules in Figure 2.1 with [PI-COMM-RED] replaced with the following

$$a(x_1, \ldots, x_n).P \mid \overline{a}\langle b_1, \ldots, b_n\rangle.Q \quad \rightarrow \quad P\{b_1/x_1 \cdots b_n/x_n\} \mid Q.$$

The communication rule can be only applied when the arities of the prefixes match. The following cannot be reduced:

$$a(x).\mathbf{0} \mid \overline{a}\langle b, c\rangle.\mathbf{0} \not\rightarrow .$$

However, the prefixes above use the same name for a channel. This means that we can enforce some sort of interface on the channel name that would tell the arity of a data that the channel supports.

Milner's solution is to define and enforce these interfaces by using a sort system. For this, we parameterise on the set of sorts

$$\mathcal{S} : \text{Set}.$$

We assign a sort $S \in \mathcal{S}$ to every name $a \in \mathcal{N}$, and write

$$a : S$$

$$[\text{T-P-O}\textsc{ut}]\ \frac{a : S \qquad ob(S) = \langle S_1, \ldots, S_n\rangle \qquad b_1 : S_1, \ldots, b_n : S_n \qquad \vdash P}{\vdash \overline{a}\langle b_1, \ldots b_n\rangle.P}$$

$$[\text{T-P-I}\textsc{n}]\ \frac{a : S \qquad ob(S) = \langle S_1, \ldots, S_n\rangle \qquad x_1 : S_1, \ldots, x_n : S_n \qquad \vdash P}{\vdash a(x_1, \ldots x_n).P}$$

$$[\text{T-P-P}\textsc{ar}]\ \frac{\vdash P \qquad \vdash Q}{\vdash P \mid Q} \qquad\qquad [\text{T-P-R}\textsc{es}]\ \frac{\vdash P}{\vdash (\nu x)P}$$

$$[\text{T-P-R}\textsc{ep}]\ \frac{\vdash P}{\vdash\, !P} \qquad\qquad [\text{T-P-N}\textsc{il}]\ \frac{}{\vdash \mathbf{0}}$$

Figure 2.4: Polyadic pi-calculus sort system rules

to mean that the name a has the sort S. We also parameterise on the sorting of channels, that is, to some sorts we assign a sequence of sorts and the assignment is a partial function as follows

$$ob : \mathcal{S} \rightarrow_{\mathrm{p}} \mathcal{S}^*.$$

This function is intuitively an interface of a polyadic channel of sort $\mathcal{S}$. That is, it says how many names it can receive or send and of what sort.

Definition 16 (Polyadic Sort System)**.** The well sorted predicate $\vdash$ on processes is defined as the least predicate satisfying the rules in Figure 2.4.

The rules are quite trivial. The rules [T-P-Out] and [T-P-In] simply ensure that the polyadicity and the sorts that the channel can carry match. And other rules propogate the check.

For example, we can recover the monadicity and uni-sortedness of the pi-calculus by having only one sort $\star$ and define:

$$\begin{array}{rcl} \mathcal{S} & = & \{\star\} \\ ob(\star) & = & \langle\star\rangle \\ a : \star & & \text{for every } a \in \mathcal{N}. \end{array}$$

Under this instantiation of the system the agent $a(x).\mathbf{0} \mid \overline{a}\langle b, c\rangle.\mathbf{0}$ is not well-sorted. Howver, $\vdash a(x).\mathbf{0} \mid \overline{a}\langle b\rangle.\mathbf{0}$.

The sort system has the pleasant property that well-sorted agents do not get stuck because of an arity mismatch. This is not hard to prove, the easiest way is to introduce another agent

$$\mathbf{wrong},$$

the reduction rule

$$a(x_1, \ldots, x_n).P \mid \overline{a}\langle b_1, \ldots, b_m\rangle.Q \;\rightarrow\; \mathbf{wrong} \qquad \text{if } n \neq m$$

that reduces agents to the stuck process **wrong**, and structural congruence that equates every agent to **wrong** that have a subagent **wrong**. The we can state where $\rightarrow^*$ is a reflexive and transitive closure of $\rightarrow$:

Theorem 1 (Well sorted process don't go wrong)**.**

$$\vdash P \implies P \not\rightarrow^* \textbf{wrong}$$

The above property is often called safety. It is a consequence of the following result which is a standard result for a sort or type system.

Theorem 2 (Subject reduction)**.**

$$\vdash P \land P \rightarrow P' \implies \vdash P'$$

2.3 Psi-calculi

The psi-calculi framework is an extension and generalisation of the pi-calculus with arbitrary data structures, logics and logical assertions. The need for extension is the realisation that the pi-calculus is not adequate for modeling realistic systems because they feature more complicated data structures than names, for example, integers, lists, trees, etc. It is, of course, possible to model large systems completely in the pi-calculus framework, however, it is neither feasible nor straightforward.

This need has spawned many pi-calculus extensions over the years: the applied pi-calculus by Abadi and Fournet [AF01], and spi-calculus by Abadi and Gordon [AG97] extend the pi-calculus with primitives for security among many others.

What differentiates psi-calculi from other pi-calculus extensions is that it is also a generic unifying theory that can capture many of those extensions as special cases. In this respect, psi-calculi can also be regarded as a framework of process calculi.

Psi-calculi gives powerful reasoning tools by simply instantiating its parameters. For example, the congruence obtained from bisimulation satisfies the natural laws of structural congruence. Typically, the requisites on parameters is trivial to check; it is certainly much easier to establish the requisites than to conduct an arduous prove that the bisimulation of a custom extensions of the pi-calculus satisfies structural congruence laws.

What is more, the meta-theory of psi-calculi has been mechanised in the proof assistant Nominal Isabelle. Thus, the correctness of psi-calculi meta-theory is guaranteed by the state-of-art theorem proving technology.

This sections is a brief summary of the main definitions and results of psi-calculi. We invite the reader to read the exposition of psi-calculi by Bengtson et al. [BJPV11] where the reader will find detailed explanations and ample of examples of psi-calculi.

2.3.1 Basics

Psi-calculi generalises the pi-calculus in quite straightforward ways. What follows is a comparison between the constructs playing the same role in both calculi.

Instead of drawing elements from just a name set, psi-calculi draws the subject and object of both input and output prefix from an arbitrary user defined set called terms $M, N \in \mathbf{T}$.

pi-calculus	psi-calculi
$a(x).P$	$M(\lambda\widetilde{x})N.P$
$\overline{a}b.P$	$\overline{M}\,N.P$

The psi-calculi input prefix supports pattern matching where $\widetilde{x}$ is a sequence of pattern names bound in both N and P. For example,

$$a(\lambda x)x.P$$

is an encoding of the pi-calculus input $a(x).P$.

Psi-calculi also generalise the matching, mismatching and sum in one construct. Condition in psi-calculi is not restricted to just checking name equality, it is, as with terms, a condition drawn from arbitrary set called conditions $\varphi \in \mathbf{C}$. In the following, sum is encoded if there is (this is not need to be the case) a particular condition that is always enabled $\mathtt{true} \in \mathbf{C}$.

pi-calculus	psi-calculi
$\textbf{if } a = b \textbf{ then } P$	$\textbf{case } \varphi : P$
$P + Q$	$\textbf{case } \mathtt{true} : P \mathbin{[]} \mathtt{true} : Q$
$\textbf{if } a_1 = b_1 \textbf{ then } P_1 + \cdots + \textbf{if } a_n = b_n \textbf{ then } P_n$	$\textbf{case } \varphi_1 : P_1 \mathbin{[]} \ldots \mathbin{[]} \varphi_n : P_n$

Furthermore, psi-calculi introduces a novel construct called assertions (or sometimes environment) $\Psi \in \mathbf{A}$. The following is a psi-calculi process that encloses the assertion Ψ

$$(\!|\Psi|\!).$$

Conditions depend on this environment such as when two channels are equivalent, or when a condition in a case statement is enabled. This is also a process whose names can be restricted or substituted. For example, the condition `is-flag-set?` is enabled whenever `flag-is-set` is in the current environment. Suppose $P \xrightarrow{\alpha} P'$, then

$$(\!|\mathtt{flag\text{-}is\text{-}set}|\!) \mid \textbf{case } \mathtt{is\text{-}flag\text{-}set?} : P \xrightarrow{\alpha} P',$$

but the following process does not have any transitions in an empty environment:

$$\textbf{case } \mathtt{is\text{-}flag\text{-}set?} : P.$$

Formally, the parameters are the following.

Definition 17 (Psi-calculi Parameters)**.**

$$\begin{array}{rcll}
\mathbf{T} & : & \mathrm{NomSet} & \text{(Terms)} \\
\mathbf{C} & : & \mathrm{NomSet} & \text{(Conditions)} \\
\mathbf{A} & : & \mathrm{NomSet} & \text{(Assertions)} \\[4pt]
\mathsf{s}_{\mathbf{T}} & : & \mathcal{N}^* \times \mathbf{T}^* \to \mathbf{T} \to \mathbf{T} & \text{(Term Subsitution)} \\
\mathsf{s}_{\mathbf{C}} & : & \mathcal{N}^* \times \mathbf{T}^* \to \mathbf{C} \to \mathbf{C} & \text{(Condition Subsitution)} \\
\mathsf{s}_{\mathbf{A}} & : & \mathcal{N}^* \times \mathbf{T}^* \to \mathbf{A} \to \mathbf{A} & \text{(Assertion Subsitution)} \\[4pt]
\dot{\leftrightarrow} & : & \mathbf{T} \times \mathbf{T} \to \mathbf{C} & \text{(Channel (Pre)equivalence)} \\
\otimes & : & \mathbf{A} \times \mathbf{A} \to \mathbf{A} & \text{(Assertion Composition)} \\
\mathbf{1} & : & \mathbf{A} & \text{(Assertion Unit)} \\
\vdash & \subseteq & \mathbf{A} \times \mathbf{C} & \text{(Entailment)}
\end{array}$$

We let σ range over $(\mathcal{N} \times \mathbf{T})^*$. For a particular σ we also write $[\widetilde{x} := \widetilde{M}]$ where $x_i \in \mathcal{N}$ and $M_i \in \mathbf{T}$. For σ application to a M we write $M\sigma_{\mathbf{T}}$ to mean $\mathsf{s}_{\mathbf{T}}(\sigma)(M)$, and if the parameter is understood, we simply write $M\sigma$.

Definition 18 (Static equivalence)**.** Two assertions are statically equivalent if and only if they entail the same conditions, formally:

$$\Psi \simeq \Psi' \quad \overset{\text{def}}{\iff} \quad (\forall \varphi \in \mathbf{C})\ \Psi \vdash \varphi \iff \Psi' \vdash \varphi$$

The requisites on parameters are that channel equivalence is symmetric and transitive, and also that assertions form an abelian monoid $(\otimes, \mathbf{1})$ modulo static equivalence.

Definition 19 (Requisites)**.**

$$\begin{array}{ll}
\Psi \vdash M \dot{\leftrightarrow} N \implies \Psi \vdash N \dot{\leftrightarrow} M & \text{(Channel Symmetry)} \\
\Psi \vdash M \dot{\leftrightarrow} N \wedge \Psi \vdash N \dot{\leftrightarrow} L \implies \Psi \vdash M \dot{\leftrightarrow} L & \text{(Channel Transitivity)} \\[4pt]
\Psi \otimes \mathbf{1} \simeq \Psi & \text{(Identity)} \\
\Psi \otimes \Psi' \simeq \Psi' \otimes \Psi & \text{(Commutativity)} \\
(\Psi \otimes \Psi') \otimes \Psi'' \simeq \Psi \otimes (\Psi' \otimes \Psi'') & \text{(Associativity)} \\
\Psi \simeq \Psi' \implies \Psi \otimes \Psi'' \simeq \Psi' \otimes \Psi'' & \text{(Compositionality)}
\end{array}$$

Now we give the syntax of the psi-calculi agents.

Definition 20 (Agents/Processes)**.**

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

We denote the set of precesses as **P**.

Well-formed agents In the (Input) we require that $\widetilde{x} \subseteq n(N)$ and elements of the sequence $\widetilde{x}$ are pairwise distinct. Also $\widetilde{x}$ bind into N and P. An assertion is *guarded* if it is a subterm of (Input) or (Output). In (Replication) there cannot be unguarded assertions in P and likewise for (Case) in P_i.

Definition 21 (Requisites on substitution)**.** The substitutions $\sigma_{\mathbf{T}}, \sigma_{\mathbf{C}}$ and $\sigma_{\mathbf{A}}$ are required to satisfy the following for $X \in \{\mathbf{T}, \mathbf{C}, \mathbf{A}\}$.

$$(\forall \widetilde{a} \subseteq n(X))(\forall b \in n(\widetilde{T}))b \in n(X[\widetilde{a} := \widetilde{T}])$$

and

$$(\forall \widetilde{b} \# X, \widetilde{a})X[\widetilde{a} := \widetilde{T}] = ((\widetilde{b}\, \widetilde{a})X)[\widetilde{b} := \widetilde{T}]$$

The first requirements prevents substitution from erasing names, and the second is an α-conversion for substitutions.

We define substitution on agents by structural recursion.

Definition 22 (Substitution)**.**

$$\begin{array}{rcl}
\widetilde{x} \# \sigma \implies (M(\lambda\widetilde{x})N.P)\sigma & \overset{\text{def}}{=} & M\sigma_{\mathbf{T}}(\lambda\widetilde{x})N\sigma_{\mathbf{T}}.(P\sigma) \\
(M\,N.P)\sigma & \overset{\text{def}}{=} & M\sigma_{\mathbf{T}}\,N\sigma_{\mathbf{T}}.P\sigma \\
(\mathbf{case}\ \varphi_1 : P_1 \ [\!]\ \varphi_n : P_n)\sigma & \overset{\text{def}}{=} & \mathbf{case}\ \varphi_1\sigma_{\mathbf{C}} : P_1\sigma \ [\!]\ \varphi_n\sigma_{\mathbf{C}} : P_n\sigma \\
a \# \sigma \implies ((\nu a)P)\sigma & \overset{\text{def}}{=} & (\nu a)P\sigma \\
(P \mid Q)\sigma & \overset{\text{def}}{=} & P\sigma \mid Q\sigma \\
(!P)\sigma & \overset{\text{def}}{=} & !P\sigma \\
(\mathbf{0})\sigma & \overset{\text{def}}{=} & \mathbf{0} \\
((\!|\Psi|\!))\sigma & \overset{\text{def}}{=} & (\!|\Psi\sigma_{\mathbf{A}}|\!)
\end{array}$$

The notion of frame captures the current environment of a process. It is used to define the operational semantics of psi-calculi.

Definition 23 (Frame)**.** A frame is a tuple of names and assertions of the form

$$(\nu\widetilde{a})\Psi$$

Frame composition is defined for $\widetilde{b} \# \widetilde{a}, \Psi$ and $\widetilde{a} \# \Psi'$ as

$$(\nu\widetilde{a})\Psi \otimes (\nu\widetilde{b})\Psi' = (\nu\widetilde{ab})(\Psi \otimes \Psi').$$

We also define addition of a name to a frame as $(\nu a)(\nu\widetilde{b})\Psi = (\nu a, \widetilde{b})\Psi$. We can extract a frame from a process by using the following function

$$\begin{array}{rcl}
\mathcal{F}((\!|\Psi|\!)) & = & (\nu\epsilon)\Psi \\
\mathcal{F}(P \mid Q) & = & \mathcal{F}(P) \otimes \mathcal{F}(Q) \\
\mathcal{F}((\nu a)P) & = & (\nu a)\mathcal{F}(P)
\end{array}$$

which, for the other cases, is defined as the empty frame, $(\nu\epsilon)\mathbf{1}$.

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

$$[\textsc{Com}]\ \frac{\begin{array}{c}\Psi_Q \otimes \Psi \vartriangleright P \xrightarrow{\overline{M}(\nu\widetilde{a})N} P' \\ \Psi_P \otimes \Psi \vartriangleright Q \xrightarrow{KN} Q' \qquad \Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \mathrel{\dot{\leftrightarrow}} K\end{array}}{\Psi \vartriangleright P \mid Q \xrightarrow{\tau} (\nu\widetilde{a})(P' \mid Q')}\ \widetilde{a} \# Q$$

$$[\textsc{Par}]\ \frac{\Psi_Q \otimes \Psi \vartriangleright P \xrightarrow{\alpha} P'}{\Psi \vartriangleright P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \mathrm{bn}(\alpha) \# Q \qquad [\textsc{Case}]\ \frac{\Psi \vartriangleright P_i \xrightarrow{\alpha} P' \qquad \Psi \vdash \varphi_i}{\Psi \vartriangleright \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \xrightarrow{\alpha} P'}$$

$$[\textsc{Rep}]\ \frac{\Psi \vartriangleright P \mid !P \xrightarrow{\alpha} P'}{\Psi \vartriangleright !P \xrightarrow{\alpha} P'} \qquad [\textsc{Scope}]\ \frac{\Psi \vartriangleright P \xrightarrow{\alpha} P'}{\Psi \vartriangleright (\nu b)P \xrightarrow{\alpha} (\nu b)P'}\ b \# \alpha, \Psi$$

$$[\textsc{Open}]\ \frac{\Psi \vartriangleright P \xrightarrow{\overline{M}(\nu\widetilde{a})N} P'}{\Psi \vartriangleright (\nu b)P \xrightarrow{\overline{M}(\nu\widetilde{a}\cup\{b\})N} P'}\ \begin{array}{c} b \# \widetilde{a}, \Psi, M \\ b \in \mathrm{n}(N)\end{array}$$

Symmetric versions of [Com] and [Par] are elided. In the rule [Com] we assume that $\mathcal{F}(P) = (\widetilde{b_P})\Psi_P$ and $\mathcal{F}(Q) = (\widetilde{b_Q})\Psi_Q$ 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 $\mathcal{F}(Q) = (\widetilde{b_Q})\Psi_Q$ 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.

Figure 2.5: Operational semantics of psi-calculi.

Definition 24 (Transition relation)**.** The transition relation is defined as the least relation satisfying the rules in Figure 2.5.

In Figure 2.5 rules, the premises and conclusions are indexed by an environment assertion indicated by the $\vartriangleright$ symbol. They 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]. In the rules [Par] and [Com], the parallel agents contribute to each others environment via their frames. In a derivation tree for a transition, the composition of environment assertions will therefore increase towards the leafs by application of [Par] and [Com]. 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.

2.3.2 Example Psi-calculi

The pi-calculus is a psi-calculus. We instantiate the psi-calculi framework with the following definitions to obtain the full pi-calculus with summation, matching and mismatching.

The terms are instantiated to be simply the set of names, conditions are set to equality and disequality between names, and the true condition used to encode the sum operator. The pi-calculus does not have environments, so the set of assertions is a singleton set with trivial composition and unit parameters. To be complete, we give the definition of the substation function.

$$
\begin{array}{rcl}
\mathbf{T} & \overset{\text{def}}{=} & \mathcal{N} \\
\mathbf{C} & \overset{\text{def}}{=} & \{a =_c b : a, b \in \mathcal{N}\} \cup \\
 & & \{a \neq_c b : a, b \in \mathcal{N}\} \cup \{\mathtt{true}\} \\
\mathbf{A} & \overset{\text{def}}{=} & \{*\} \\
\mathbf{1} & \overset{\text{def}}{=} & * \\
\mathbf{1} \otimes \mathbf{1} & \overset{\text{def}}{=} & \mathbf{1} \\
\dot{\leftrightarrow} & \overset{\text{def}}{=} & =_c \\
\mathbf{1} \vdash \mathtt{true} & \overset{\text{def}}{\iff} & \text{always} \\
\mathbf{1} \vdash a =_c b & \overset{\text{def}}{\iff} & a = b \\
\mathbf{1} \vdash a \neq_c b & \overset{\text{def}}{\iff} & a \neq b \\
\mathsf{s}_{\mathbf{T}}\ \sigma\ a & \overset{\text{def}}{=} & b \quad \text{if } (a, b) \in \sigma \\
\mathsf{s}_{\mathbf{T}}\ \sigma\ a & \overset{\text{def}}{=} & a \quad \text{otherwise} \\
\mathsf{s}_{\mathbf{C}}\ \sigma\ (a =_c b) & \overset{\text{def}}{=} & \mathsf{s}_{\mathbf{T}}\ \sigma\ a =_c \mathsf{s}_{\mathbf{T}}\ \sigma\ b \\
\mathsf{s}_{\mathbf{C}}\ \sigma\ (a \neq_c b) & \overset{\text{def}}{=} & \mathsf{s}_{\mathbf{T}}\ \sigma\ a \neq_c \mathsf{s}_{\mathbf{T}}\ \sigma\ b \\
\mathsf{s}_{\mathbf{A}}\ \sigma\ \mathbf{1} & \overset{\text{def}}{=} & \mathbf{1}
\end{array}
$$

We give a formally translate the pi-calculus to the psi-calculus defined above.

$$
\begin{array}{rcl}
[\![a(x).P]\!] & \overset{\text{def}}{=} & a(\lambda x)x.[\![P]\!] \\
[\![\overline{a}b.P]\!] & \overset{\text{def}}{=} & a\,b.[\![P]\!] \\
[\![P + Q]\!] & \overset{\text{def}}{=} & \textbf{case}\ \mathtt{true} : [\![P]\!]\ [\!]\ \mathtt{true} :: [\![Q]\!] \\
[\![\textbf{if}\ a = b\ \textbf{then}\ P]\!] & \overset{\text{def}}{=} & \textbf{case}\ a =_c b : [\![P]\!] \\
[\![\textbf{if}\ a \neq b\ \textbf{then}\ P]\!] & \overset{\text{def}}{=} & \textbf{case}\ a \neq_c b : [\![P]\!] \\
[\![P \mid Q]\!] & \overset{\text{def}}{=} & [\![P]\!] \mid [\![Q]\!] \\
[\![!P]\!] & \overset{\text{def}}{=} & ![\![P]\!] \\
[\![\mathbf{0}]\!] & \overset{\text{def}}{=} & \mathbf{0}
\end{array}
$$

The two calculi are the same in the sense that the transition relation correspond [BJPV11].

It is common to extend the pi-calculus with a free algebra parameterised over some signature with function symbols Σ and equational logics Eq over those symbols (e.g., the applied pi-calculus [AF01]). This is straightforward in psi-calculi.

Let Σ be a signature, $T(\Sigma, \mathcal{N})$ a term algebra over the signature Σ generated by the set of names $\mathcal{N}$, $Eq(\Sigma, \mathcal{N})$ equations ranged over by φ, some fixed theory $\mathcal{T} \subseteq_{\text{fin}} Eq(\Sigma, \mathcal{N})$ and finally a provability relation $\vdash_\Sigma$. Then we define

$$\begin{aligned} \mathbf{T} &\overset{\text{def}}{=} T(\Sigma, \mathcal{N}) \\ \mathbf{C} &\overset{\text{def}}{=} Eq(\Sigma, \mathcal{N}) \\ M \leftrightarrow N &\overset{\text{def}}{=} M = N \\ \mathbf{1} \vdash \varphi &\overset{\text{def}}{\iff} \mathcal{T} \vdash_\Sigma \varphi \end{aligned}$$

The assertions are trivial as in the pi-calculus and substitution is defined in the standard way. The defined psi-calculus is also a framework parameterised over Σ and $\mathcal{T}$. With it we can model some aspects of cryptography, for instance, let m, k be names:

$$\begin{aligned} \Sigma &\overset{\text{def}}{=} \{\mathsf{dec}, \mathsf{enc}\} \\ \mathcal{T} &\overset{\text{def}}{=} \{\mathsf{dec}(\mathsf{enc}(m,k),k) = m\}. \end{aligned}$$

Suppose $P \xrightarrow{\alpha} P'$, then the following is a possible trace

$$\begin{aligned} \overline{a}\,\mathsf{enc}(v,k).\mathbf{0} \mid \underline{a}(\lambda x)x.\mathbf{case}\ \mathsf{dec}(x,k) = v : P &\xrightarrow{\tau} \mathbf{0} \mid \mathbf{case}\ \mathsf{dec}(\mathsf{enc}(v,k),k) = v : P \\ &\xrightarrow{\alpha} P'. \end{aligned}$$

The first transition is allowed because $\mathbf{1} \vdash a \leftrightarrow a$ holds. We can see this by expanding the definition to $\mathcal{T} \vdash_\Sigma a = a$, which holds because of reflexivity. The second transition depends on $\mathbf{1} \vdash \mathsf{dec}(\mathsf{enc}(v,k),k) = v$ being true, we can easily see this by expanding the definition and using the sole equation in the equational logic.

We can go even further by allowing local knowledge in form of assertions being sets of equations as follows

$$\begin{aligned} \mathbf{A} &\overset{\text{def}}{=} \{\mathcal{S} : \mathcal{S} \subseteq_{\text{fin}} Eq(\Sigma, \mathcal{N})\} \\ \mathcal{S}_1 \otimes \mathcal{S}_2 &\overset{\text{def}}{=} \mathcal{S}_1 \cup \mathcal{S}_2 \\ \mathbf{1} &\overset{\text{def}}{=} \emptyset \end{aligned}$$

and modifying the entailment relation to

$$\mathcal{S} \vdash \varphi \overset{\text{def}}{\iff} \mathcal{T} \cup \mathcal{S} \vdash_\Sigma \varphi$$

thus we allow local theories. Returning to the example, we can also do

$$(\!|\{\mathsf{enc}(v,k) = y, y = x\}|\!) \mid \mathbf{case}\ \mathsf{dec}(x,k) = v : P \xrightarrow{\alpha} P'.$$

because

$$\{\mathsf{enc}(v,k) = y, y = x, \mathsf{dec}(\mathsf{enc}(m,k),k) = m\} \vdash_\Sigma \mathsf{dec}(x,k) = v.$$

We easily obtained an advanced calculus which is strictly more expressive than the pi-calculus. All of these kind of calculi enjoy the usual meta-theoretical results presented in the next section.

2.3.3 Bisimulation Theory

In this section, we give a brief summary of some of results on psi-calculi meta-theory. All of them have been checked in the Nominal Isabelle theorem prover. The results presented here for the bisimilarity are important because they show that the definition of bisimulation, bisimilarity, congruence are indeed correct. In the sense, that the definitions conform to natural expectations of a process calculi.

Strong bisimulation relation is a generalisation of the pi-calculus relation Definition 8. If we instantiate the psi-calculi framework to obtain a pi-calculus, the bisimulation relations do coincide. The extra requirements below are due to the use of assertions.

Definition 25 ((Strong) Bisimulation)**.** Bisimulation is ternary relation $\mathcal{R}$ such that forall Ψ, P, Q that $\mathcal{R}(\Psi, P, Q)$ implies all of

1. $\mathcal{R}(\Psi, P, Q)$,
2. $\forall \alpha, P'.\mathrm{bn}(\alpha) \# \Psi, Q \wedge \Psi \vartriangleright P \xrightarrow{\alpha} P' \implies \exists Q'.\Psi \vartriangleright Q \xrightarrow{\alpha} Q' \wedge \mathcal{R}(\Psi, P', Q')$
3. $\Psi \otimes \mathcal{F}(P) \simeq \Psi \otimes \mathcal{F}(Q)$,
4. $\forall \Psi'.\mathcal{R}(\Psi \otimes \Psi', P, Q)$,

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

Clauses 1 and 2 are the same as in the pi-calculus (see Definition 8). Clause 1 states that the relations is symmetric and Clause 2 state that the Q can simulate the agent P. Clause 3 asserts that the environments of the agents are equivalent, that is, they entail exactly the same conditions. Clause 3 states that the agents are related even by extending the environments arbitrarily.

The following laws hold for bisimilarity. They state that bisimilarity is almost a congruence. It is exactly the same situation as in the pi-calculus: it fails to be a congruence for the input prefix (see Remark 1 in Section 2.2.5).

Theorem 3. For all Ψ, a, P, Q, R, M, N all the following hold

$$\begin{aligned}
P \dot{\sim}_\Psi Q &\implies P \mid R \dot{\sim}_\Psi Q \mid R \\
P \dot{\sim}_\Psi Q &\implies !P \dot{\sim}_\Psi !Q \\
P \dot{\sim}_\Psi Q &\implies M N.P \dot{\sim}_\Psi M N.Q \\
a \# \Psi \wedge P \dot{\sim}_\Psi Q &\implies (\nu a)P \dot{\sim}_\Psi (\nu a)Q \\
\forall i.P_i \dot{\sim}_\Psi Q_i &\implies \textbf{case}\ \widetilde{\varphi} : \widetilde{P} \dot{\sim}_\Psi \textbf{case}\ \widetilde{\varphi} : \widetilde{Q} \\
(\forall \widetilde{L}.\, P[\widetilde{a} := \widetilde{L}] \dot{\sim}_\Psi Q[\widetilde{a} := \widetilde{L}]) &\implies M(\lambda\widetilde{a})N.P \dot{\sim}_\Psi M(\lambda\widetilde{a})N.Q
\end{aligned}$$

We can use the same 'trick' to obtain a congruence as already seen in the above result. We close bisimilarity under all substitution sequences to obtain the following where $P\widetilde{\sigma}$ means $(((P\sigma_1)\ldots)\sigma_n)$ for $\widetilde{\sigma} = (\sigma_1, \ldots, \sigma_n)$.

Definition 26 ((Strong) Congruence)**.** $P \sim_\Psi Q$ is defined as for all substitution sequences $\widetilde{\sigma}$ it holds that $P\widetilde{\sigma} \mathrel{\dot{\sim}_\Psi} Q\widetilde{\sigma}$. Also, we write $P \sim Q$ for $P \sim_{\mathbf{1}} Q$.

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

Theorem 5 (Structural Laws)**.**

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

The above theory can be formulated for the weak version of bisimulation and congruence, however, due to assertions the theory is quite subtle [JBPV10].

Chapter 3

Contributions

In this chapter, we present the contributions of this thesis (Paper I, Paper II, Paper III). The contributions pertain to verification of concurrent systems. We have developed a tool, the Psi-Calculu Workbench (Section 3.1), based on the psi-calculi framework (Section 2.3), which provides symbolic execution and equivalence checking. We developed a type system based on session types [HVK98] for systems with unreliable communication in Section 3.3, and finally we extend the expressiveness of psi-calculi by adding generalised pattern matching (Section 3.4) and a sort system (Section 3.5).

3.1 Psi-Calculi Workbench

Psi-Calculi Workbench (Pwb) is a parametric tool for modelling concurrent systems. It provides symbolic execution and automated behavioral equivalence checking. It is based on psi-calculi and effectively it is an implementation of a subset of it. The tool is a generic tool: it can model many different applied concurrent systems. And in particular the tool features unreliable synchronous broadcast communication that can be freely mixed with the usual reliable point-to-point communication. In Paper I, we demonstrate a use case for wireless sensor networks among others.

Pwb could be thought as a library for writing process calculi implementations. It is written in the Standard ML programming language and is organised using the powerful module system of Standard ML. The Figure 3.1 summarises the architecture of the tool.

Pwb comes with a library for facilitating implementation of the parameters: nominal data structures, parser combinators, a rudimentary SMT solver, etc. The Pwb is instantiated to a process calculus tool by implementing the psi-calculi parameters, a constraint solver for the (symbolic) transition constraints, a constraint solver for (symbolic) bisimulation constraints, and printers and parsers for command interpreter. The instantiation of these produces a concrete tool with transition enumeration, bisimulation checking, and a command interpreter as user interface.

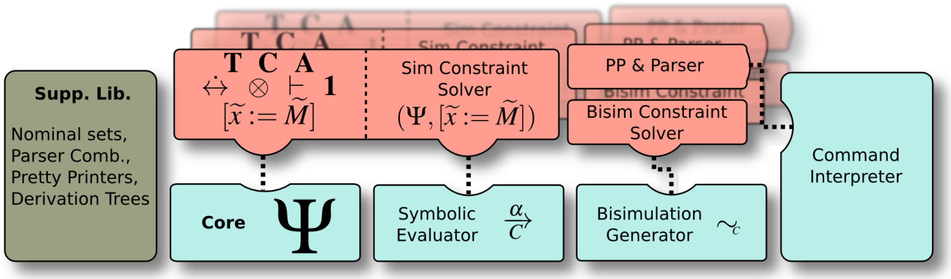

Supp. Lib.
Nominal sets, Parser Comb., Pretty Printers, Derivation Trees
T C A
$\leftrightarrow$ $\otimes$ $\vdash$ **1**
$[\widetilde{x} := \widetilde{M}]$
Sim Constraint Solver
$(\Psi, [\widetilde{x} := \widetilde{M}])$
PP & Parser
Bisim Constraint Solver
Command Interpreter
Core
Ψ
Symbolic Evaluator
$\xrightarrow[C]{\alpha}$
Bisimulation Generator
$\sim_c$

Figure 3.1: The Psi-Calculi Workbench architecture

In other words, broadly speaking, Pwb is a function of the following kind

$$\text{Pwb}(\mathbf{T}, \mathbf{C}, \mathbf{A}, \dots) = \textit{tool}.$$

For example, the Pwb distribution contains an implementation of pi-calculus by instantiating the psi-calculi parameters such that we obtain a tool for it

$$\text{Pwb}(\mathbf{T}_\pi, \mathbf{C}_\pi, \mathbf{A}_\pi, \dots) = \textit{tool}_\pi.$$

In this sense, we obtain a tool similar to the Mobility Workbench [Vic94]. Moreover, we can obtain tools for more powerful process calculi by implementing parameters with structured data, non-trivial logics, and assertions. Thus Pwb is a tool factory for process calculi which possibly can save quite a bit of effort of implementing similar tools from scratch.

Pwb accepts the following syntax for the agents. The prefixes are of the polyadic variety. Prefixes are of two kinds: broadcast and point-to-point. The syntax is parametric: `x` is a name accepted by the instantiation of `parseName` parameter, likewise for terms `M`, `N`, conditions `phi`, and assertions `Psi`.

```
P  ::=  'M < N,...,N > .P              (Polyadic Output)
     |  M(x,...,x).P                   (Polyadic Input)
     |  'M! < N,...,N > .P             (Polyadic Broadcast Output)
     |  M?(x,...,x).P                  (Polyadic Broadcast Input)
     |  case phi : P[]...[]phi : P     (Case)
     |  (new x)P                       (Restriction)
     |  P | P                          (Parallel)
     |  !P                             (Replication)
     |  (|Psi|)                        (Assertion)
     |  A < M,...,M >                  (Invocation of Process Clause)
     |  *tau * .P                      (Silent Prefix)
     |  0                              (Nil)
```

Pwb transitions are indexed with a process clause environment of the form

$$\texttt{A(x,...,x) <= P}$$

Processes can invoke the clauses recursively. There can be more than one clause with the same name. In this case, the clause is chosen non-deterministically.

We have developed a tool with symbolic executable semantics and automated behavioral equivalence checking for an advanced process calculi. This make it more practical to model more complex real-world systems. In addition, we have implemented and distribute with Pwb example instances for the pi-calculus, calculi with structured data: calculus for modelling aggregation protocols in wireless sensor networks, alternating bit protocol and others. We provide a transition constraint solver for each of them and a bisimulation constraint solver for the pi-calculus instance.

3.2 Application of Pwb to Wireless Sensor Networks

Wireless Sensor Networks are quite challenging to model because of usage broadcast communication, need for structured messages, and dynamic connectivity. In Paper I we demonstrate applications of Psi-calculi Workbench to Wireless Sensor Networks (Section 5 and 6).

A wireless sensor network (WSN) consists of a number of nodes with small computational capacity, sensors and short range wireless communication. The purpose of such a network is to sense environmental data and to relay it to a base station.

We model a well-known tree building and data collection protocol TAG [MFHH02] for WSN. It uses multi-hop communication pattern to establish a logical spanning tree of nodes rooted in the base station. The nodes later send and forward data on the basis of this tree.

Our main contribution is that our model is high level and executable. This is due to the strength of our modelling tool: we mix in the model point-to-point and unreliable broadcast communication, and use structured data all in the same framework.

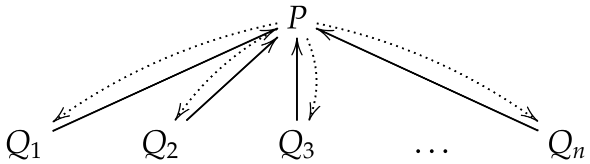
In addition, we show that our framework is flexible. We extend the model of the tree building protocol to handle the dynamic connectivity of nodes (Section 6). This allows to model the mobility of nodes present in some WSNs.

We have showed that it is possible to model WSNs in a single framework of the Psi-Calculi Workbench. It can handle broadcast communication, structured messages, and dynamic connectivity aspects of a WSN.

3.3 Broadcast Session Types

Session types [HVK98] is quite elegant and powerful type system for process calculi, which typically ensures such an important property as deadlock freedom. However, its standard application is to systems with reliable communication. We are the first to propose session types for a system with unreliable communication (Paper II). We consider wireless sensor networks (WSN) as a motivational example of a system with unreliable communication.

In WSNs, the following common pattern arises, for instance, data collection protocol [MFHH02]. In the diagram below P, and $Q_1, \ldots, Q_n$ represent physical nodes of a subnetwork in some wireless sensor network.



The node P first broadcasts a request message (dotted lines) to the neighboring nodes $Q_1, \ldots, Q_n$, and then waits for a response message from each node (solid line) in turn. In other words, *P scatters* a request message, and then iteratively *gathers* responses.

Implementing such protocols in this setting is challenging. Due to the unreliable nature of the communication medium. For example, in WSNs, not all the neighboring nodes may receive the request message, and, in an extreme case, none. So it is hopeless for P to wait for all the nodes to respond, even if P is aware of the number of neighboring nodes.

What is more, the communication medium is not the only source of unreliability. Nodes may fail to respond in various ways. For example, the node's battery may discharge, or a node might be moved away from the radio coverage of P's antenna.

These kinds of unreliability make a system highly non-deterministic. It means that a correct implementation of the protocol needs to be robust to all these kinds of failures. Also a desirable implementation property is deadlock-freedom.

For expressing protocols in such systems, we adopt the view of session types [HVK98]. That is, we take a session type as a protocol specification, a process calculus as an implementation and we use a typing relation to ensure that that a well-typed process is deadlock free and robust to failures.

Our process calculus features both reliable point-to-point and unreliable broadcast communication primitives. Its syntax is fairly standard and includes parallel composition, recursion, restriction, input and output prefix, and nil. For supporting the mentioned systems, we also include session initialisation and acceptance, scatter and gather. Syntactically, initialisation and acceptance are dual as well as scatter and gather in the usual sense, however, semantically session initialisation and scatter uses unreliable communication, and session acceptance and gather are iterative processes that non-deterministically waits for arbitrary number of messages. This models the communication pattern described above. To handle the exceptions arising from the non-determinism, we couple every process with a process for recovery

Perhaps surprisingly we were able to use the standard binary session types by extending the notion of duality. That is, multiple copies of session endpoints at the same stage are dual to the endpoint of the scatterer.

To ensure session fidelity, that is, that the processes follow the prescribed session type and do not diverge due to non-determinism, we introduced semantic support for tracking the active sessions. This is achieved by translating the calculus into a 'runtime' calculus. As runtime calculus, we use a psi-calculus where the sessions are tracked using a shared environment in which the processes count how many prefixes were consumed.

Our system enjoys the usual subject reduction property: a well-typed process reduces to a well-typed process. The other result is that a well-typed process does not reduce to an error process, meaning that a well-typed process is deadlock-free.

3.4 Pattern Matching

Pattern-matching is an convenient construct used in many formal languages. By generalising it in psi-calculi (Paper III), we show that psi-calculi becomes more expressive while retaining the standard meta-theory results.

Formally, in standard psi-calculi (Section 2.3) pattern matching is defined using the available substitution function on terms to instantiate the pattern variables, that is, the pattern M matches the term N where $\widetilde{x}$ are the pattern variables if the following holds for some sequence of terms $\widetilde{L}$:

$$M[\widetilde{x} := \widetilde{L}] = N.$$

To give some intuition behind the definition, consider the example where terms are defined as tuples[1] of names and integers. Then the pattern

$$\langle x, y, 3 \rangle$$

matches

$$\langle 1, 2, 3 \rangle$$

because we can find a substitution function such that

$$\langle x, y, 3 \rangle[x := 1, y := 2] = \langle 1, 2, 3 \rangle.$$

In psi-calculi, pattern matching is defined in terms of substitution. It is a slight generalisation of the way pattern matching is defined in many functional programming languages since the term set is arbitrary and not necessarily inductively defined. This notion of pattern matching is also common in process calculi.

This, however, is not sufficient to capture quite common patterns found in programming languages, for example, record patterns found in ML descendants like Standard ML, Haskell, OCaml among others. As a simple example of this, take the following pattern[2] which intuitively matches the first element in a tuple and binds it to x

$$1\#x$$

[1] Technically, we are forced to define terms to be finitely branching trees of names and integers that is $\mathbf{T} = \mathbb{N} \cup \mathcal{N} \cup \mathbf{T}^*$. This is because terms must be closed under all total substitutions. We address this issue by relaxing the requirement to only well-sorted substitutions. We explain this in Section 3.5.

[2] Formally, we need to extend $\mathbf{T}$ to $\mathbf{T} = \mathbb{N} \cup \mathcal{N} \cup \{1\#M : M \in t\} \cup \mathbf{T}^*$

Clearly we cannot in general define a substitution that would satisfy for any $n > 0$

$$1\#x[x := M_1] = \langle M_1, \ldots, M_n \rangle$$

and in particular

$$1\#x[x := 1] = \langle 1, 2, 3 \rangle$$

Another issue is that every term is a pattern, and every pattern is a term. For example, consider psi instance where $\mathtt{enc}(M, k)$ is a term which intuitively denotes encryption of M with the key k. The term can be received and decomposed since it is also a patter in $a(\lambda m, k)\mathtt{enc}(m, k).P$ revealing both the encrypted message and the key used to encrypt it. Obviously, this is not the intention of our cryptographic model.

The pattern variables are allowed to bind into any name in the pattern term. But if we allowed finer control over the binding, we could remedy the above example. By disallowing k to be a pattern variable, then the key cannot be discovered by simply inputing and so the input $a(\lambda m)\mathtt{enc}(m, k).P$ can occur only if the key k is known.

Yet another issue is that the substitution function is required not to erase names (Definition 21). Perhaps surprisingly, this is a consequence of using pattern matching on the label in the [In] rule (Figure 2.5). Without this requirement, it is possible to capture non-extruded restricted names (see [BJPV11] for justification and examples).

This is a drawback. Since it prevents us from defining interesting computations as part of a substitution function. For instance, symmetric cryptography can be modeled as a term rewriting system with the following rule

$$\mathtt{dec}(\mathtt{enc}(M, k), k) \to M$$

where M is a term and k is a name. But the substitution $\mathtt{dec}(x, k)[x := \mathtt{enc}(M, k)] = M$ violates the name preservation law whenever $k \# M$.

We address the issues by modifying psi-calculi in the following ways. We introduce a new nominal datatype $\mathbf{X}$ (which may or may not overlap with terms $\mathbf{T}$), ranged over by X, called patterns and two operations on the datatype. The $\textsc{Match}(M, \widetilde{x}, X)$ operation matches the term M against the pattern X with the $\widetilde{x}$ being pattern variables. The $\textsc{Vars}(X)$ operation returns a set of sets of names that are allowed to be bound by $\widetilde{x}$ in the pattern X. This operation is used to ensure that the binders $\widetilde{x}$ bind the appropriate names in the input prefix $M(\lambda\widetilde{x})X.P$.

Definition 27 (Pattern Matching Psi-calculi Parameters)**.** We extend the parameters of the Definition 17 with the following

$$\begin{array}{rcll}
\mathbf{X} & : & \text{NomSet} & \text{(Patterns)} \\
\mathsf{s}_{\mathbf{X}} & : & \mathcal{N}^* \times \mathbf{T}^* \to \mathbf{X} \to \mathbf{X} & \text{(Pattern Substitution)} \\
\textsc{Match} & : & \mathbf{T} \times \mathcal{N}^* \times \mathbf{X} \to \mathcal{P}_{\text{fin}}(\mathbf{T}^*) & \text{(Pattern Matching)} \\
\textsc{Vars} & : & \mathbf{X} \to \mathcal{P}_{\text{fin}}(\mathcal{P}_{\text{fin}}(\mathcal{N})) & \text{(Names as Pattern Variables)}
\end{array}$$

$\sigma_{\mathbf{P}}$ is extended with the $\sigma_{\mathbf{X}}$ as the base case for patterns in Definition 22.

Definition 28. The input agent of Definition 20 is replaced with the following form where $X \in \mathbf{X}$

$$M(\lambda\widetilde{x})X.P$$

We also replace the proviso of well-formed input agent (Definition 20) with the following

> If $\widetilde{x} \in \textsc{Vars}(X)$ and P is well formed, then $M(\lambda\widetilde{x})X.P$ is well formed.

Definition 29 (Requisites on pattern matching parameters)**.** We extend psi-calculi requisites Definition 19 with the following

$$\begin{aligned}
&\widetilde{N} \in \textsc{Match}(M, \widetilde{x}, X) \implies \\
&\qquad n(\widetilde{N}) \subseteq n(M) \cup (n(X) \setminus \widetilde{x}) && \text{(Cannot invent names)}\\
&\widetilde{N} \in \textsc{Match}(M, \widetilde{x}, X) \implies \\
&\qquad (\forall \widetilde{y} \# X) \;\; \widetilde{N} \in \textsc{Match}(M, \widetilde{y}, (\widetilde{x}\;\widetilde{y}) \cdot X) && \text{(Invariant under } \alpha\text{-conversion)}\\
&\widetilde{x} \in \textsc{Vars}(X) \wedge \widetilde{x} \# \sigma \implies \\
&\qquad \widetilde{x} \in \textsc{Vars}(X\sigma) && \text{(Substitution cannot erase pattern names)}
\end{aligned}$$

By moving pattern matching out of the label in the [In] rule, we relax the requirements of term substitution by dropping name preservation. We obtain the following transition system.

Definition 30. We define operation semantics with matching by replacing [In] in Figure 2.5 with the rule [In-M] in Figure 3.2.

The new rule increases the expressivity of psi-calculi since it allows for more kinds of behavior. Pattern matching can also be non-deterministic as exemplified by the encoding lambda calculus with erratic choice in the pattern language.

Because substitution can erase pattern variables in non pattern matching psi-calculi, the well-formedness condition $\widetilde{x} \subseteq n(N)$ of a process $M(\lambda\widetilde{x})N.P$ may not be preserved by the transition relation. But by requiring pattern substitution to preserve them (Definition 29), we obtain

Theorem 6 (Subject Reduction)**.** If P is well formed and $\Psi \rhd P \xrightarrow{\alpha} P'$, then P' is well formed.

The extensions described here are sound in the following sense. The result has been checked in the automated theorem prover Isabelle.

Theorem 7. Standard meta-theoretical results hold: (weak) bisimilarity is a (weak) congruence except for input, (weak) bisimilarity closed under substitution sequences is a (weak) congruence, (weak) bisimilarity and congruence satisfies the structural (weak) congruence laws.

We have shown that generalised pattern-matching adds expressivity to the psi-calculi framework. What is more, by introducing the generalised pattern-matching, we do not break the framework, that is, we retain the standard meta-theoretic properties.

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

Figure 3.2: Input Rule with Generic Pattern Matching

3.5 Data Sorting

We extend psi-calculi with a simple type system, which we call sorted psi-calculi (Paper III). The system allows us to remove non-sensical terms arising in the original psi-calculi (Section 2.3) and faithfully capture several well-known processes calculi including polyadic sorted and unsorted pi-calculus [Mil93], polyadic synchornisation pi-calculus [CM02], and value-passing CCS [Mil89]. The resulting parametric calculi enjoys the same meta-theoretical properties as the original psi-calculi (see Section 2.3 and [BJPV11]).

One of the motivations to introduce sorted psi-calculi is capture the notion of well-formedness of a process at the data level. This need arises when considering more structured data than names. For example, Milner [Mil93] uses a type system to enforce that channels have the same arities by sorting the names of the channels, while Abadi and Fournet in their applied pi-calculus [AF01] sort names into channel names and variables (which also double as variables in term a algebra).

The other motivation is technical. It is to help the process calculi designer to capture his or hers intentions more closely when describing the term language of a psi-calculus. Substitution on the psi-calculi parameters is required to be total (Definition 17) and the designer might be forced to expand the set of terms in order to accommodate this requirement.

We already encountered this shortcoming in Section 3.4 when we tried to define sequences of names and integers as data terms $\mathbf{T}_1 \overset{\text{def}}{=} (\mathcal{N} \cup \mathbb{N})^*$, but we were forced to set it to a somewhat more complicated structure of finitely branching trees of names and integers $\mathbf{T}_2 \overset{\text{def}}{=} \mu t.\mathcal{N} \cup \mathbb{N} \cup t^*$. This is because substitution $\mathrm{s}_D : \mathcal{N}^* \times D^* \to D \to X$ (for $D \in \{\mathbf{T}, \mathbf{C}, \mathbf{A}\}$) in Definition 17 is a total function and so it cannot yield an element outside the data term set. For example, $\langle a, b\rangle$ and $\langle c\rangle$ are both in $\mathbf{T}_1$ and $\mathbf{T}_2$, thus, if we define substitution in the obvious way and apply $[a := \langle c\rangle]$, we find that T_1 is too small as follows

$$\langle a, b\rangle[a := \langle c\rangle] = \langle\langle c\rangle, b\rangle \ \begin{matrix} \notin \mathbf{T}_1 \\ \in \mathbf{T}_2 \end{matrix}$$

We could address this issue by defining substitution by sending the above application of substitution to an error term. But this is not satisfactory either, since we still would need to expand $\mathbf{T}_1$ with elements with the sole purpose of makeing the substitution total. We thus in essence introduce spurious terms or simply "junk" to the data term language. It is for the same reason why we cannot directly define polyadic pi-calculus as a psi-calculus.

$$[\text{S-}\textsc{In}]\ \frac{\vdash P \qquad \textsc{Sort}(M) \mathrel{\underline{\propto}} \textsc{Sort}(X)}{\vdash M(\lambda\widetilde{x})X.P} \qquad [\text{S-}\textsc{Out}]\ \frac{\vdash P \qquad \textsc{Sort}(M) \mathrel{\overline{\propto}} \textsc{Sort}(N)}{\vdash \overline{M}N.P}$$

$$[\text{S-}\textsc{Res}]\ \frac{\vdash P \qquad \mathcal{S}_\nu(\textsc{Sort}(a))}{\vdash (\nu a)P} \qquad [\text{S-}\textsc{Rep}]\ \frac{\vdash P}{\vdash !P} \qquad [\text{S-}\textsc{Par}]\ \frac{\vdash P \qquad \vdash Q}{\vdash P \mid Q}$$

$$[\text{S-}\textsc{Nil}]\ \frac{}{\vdash \mathbf{0}} \qquad [\text{S-}\textsc{Case}]\ \frac{\forall i.\vdash P_i}{\vdash \textbf{case}\ \widetilde{\varphi} : \widetilde{P}} \qquad [\text{S-}\textsc{Psi}]\ \frac{}{\vdash (\!|\Psi|\!)}$$

Figure 3.3: Sorting rules

Instead, we sort the names, terms, and patterns, and only consider well-sorted substitutions. We introduce a set of sorts $\mathcal{S}$ as a parameter, and a sort context $\textsc{Sort}$ that gives the sort of names, terms and patterns. We introduce compatibility relations on prefixes that govern which data sorts can be sent or received on particular channel, and a predicate that tells which sort of names can be restricted in a process. We also introduce a relation that specifies formation of well-sorted substitutions.

We also fix a family of countably infinite name sets $\{\mathcal{N}_s\}_{s \in I}$ and denote the their union with $\mathcal{N}$.

Definition 31 (Sorted Psi-Calculi Parameters)**.** We extend the parameters of the Definition 27 with the following:

$$\begin{array}{rcll}
\mathcal{S} & : & \text{Set} & \text{(Sort Set)} \\[1ex]
\overline{\propto} & \subseteq & \mathcal{S} \times \mathcal{S} & \text{(Can Send)} \\
\underline{\propto} & \subseteq & \mathcal{S} \times \mathcal{S} & \text{(Can Receive)} \\
\prec & \subseteq & \mathcal{S} \times \mathcal{S} & \text{(Can Substitute)} \\
\mathcal{S}_\nu & \subseteq & \mathcal{S} & \text{(Can Restrict)} \\[1ex]
\textsc{Sort} & : & \mathcal{N} + \mathbf{T} + \mathbf{X} \to \mathcal{S} & \text{(Sort Context).}
\end{array}$$

We require that the sort context respects the sorting of names, that is, $\textsc{Sort}(a) = s$ iff $s \in \mathcal{N}_s$.

Definition 32 (Well Sorted Agent)**.** A well-sorted agent P is a well-formed agent (Definition 28 and Definition 20) where additionally $\vdash P$ as defined by the rules in Figure 3.3.

Definition 33 (Subsorting pre-order)**.**

$$s_1 \leq s_2 \overset{\text{def}}{\iff} (\forall t \in \mathcal{S}) \begin{pmatrix} s_2 \mathrel{\underline{\propto}} t & \implies & s_1 \mathrel{\underline{\propto}} t & \wedge \\ s_2 \mathrel{\overline{\propto}} t & \implies & s_1 \mathrel{\overline{\propto}} t & \wedge \\ t \mathrel{\underline{\propto}} s_2 & \implies & t \mathrel{\underline{\propto}} s_1 & \wedge \\ t \mathrel{\overline{\propto}} s_2 & \implies & t \mathrel{\overline{\propto}} s_1 & \end{pmatrix}$$

Definition 34 (Well sorted substitution)**.**

$$\begin{array}{ll}
\widetilde{a}\text{ distinct } \wedge |\widetilde{a}| = |\widetilde{N}| \wedge a_i \prec N_i \implies [\widetilde{a} := \widetilde{N}] \text{ w.f.} & \text{(Well formed substitution)} \\
(\forall X \in \{\mathbf{T}, \mathbf{C}, \mathbf{A}, \mathbf{X}\})(\forall T \in X) & \\
\quad \widetilde{b} \# T, \widetilde{a} \implies T[\widetilde{a} := \widetilde{N}] = ((\widetilde{a}\,\widetilde{b}) \cdot T)[\widetilde{b} := \widetilde{N}] & (\alpha\text{-conversion}) \\
(\forall M \in \mathbf{T})\ \textsc{Sort}(M\sigma) \leq \textsc{Sort}(M) & \text{(Subsorting)} \\
(\forall X \in \mathbf{X})\ \widetilde{x} \in \textsc{Vars}(X) \wedge \widetilde{x} \# \sigma \implies \textsc{Sort}(X\sigma) \leq \textsc{Sort}(X) & \text{(Subsorting)}
\end{array}$$

We show that our sorting system enjoys the usual subject reduction property. This depends on that well-sorted substitutions preserve well-formedness of a process.

Theorem 8 (Subject Reduction)**.** If P is well-formed, then

1. $P\sigma$ is well-sorted for any well-sorted σ.

2. If $\Psi \rhd P \xrightarrow{\alpha} P'$ then P' is well-sorted.

Returning to the previous example we can use terms T_1 and restrict substitutions to only substitute name or integer for names by taking $\mathcal{S} \overset{\text{def}}{=} \{\texttt{name}, \texttt{int}, \texttt{seq}\}$, $\prec \overset{\text{def}}{=} \{(\texttt{name}, \texttt{int}), (\texttt{name}, \texttt{name})\}$, and $\textsc{Sort}(a) = \texttt{name}$ for $a \in \mathcal{N}$, $\textsc{Sort}(n) = \texttt{int}$ for $n \in \mathbb{N}$ and $\textsc{Sort}(\langle x_1, \ldots, x_n \rangle) = \texttt{seq}$.

Using this sort system, we can directly represent as sorted psi-calculus both sorted and unsorted pi-calculus, polyadic synchronisation pi-calculus, and value-passing CCS. That is, all of these calculi have strong operational correspondence with the respective psi-calculus up to strong bisimulation, and the syntax is homomorphically translated to psi. In case of polyadic pi-calculus, the syntax is in bijection with the psi-calculus.

Theorem 9. The standard meta-theoretical results hold: bisimilarity is a congruence except for input, bisimilarity closed under substitution sequences is a congruence, bisimilarity and congruence satisfies the structural congruence laws. The same is true for the weak version of bisimilarity and congruence.

For technical reasons, we cannot check the above results in the current version of Nominal Isabelle [Urb08]. We have instead reduced the proofs to the trivially sorted case and for strong and weak congruence we have repeated the proof by hand.

We have introduced a sort system that allows for a more concise modeling in the psi-calculi framework. The transition relation preserves the well-formedness property of the system and the standard meta-theoretical results hold. We have shown with this kind of system we can express directly advanced process calculi.

Chapter 4

Related Work

4.1 Type Systems

Our work on sorted psi-calculi and session types for broadcast mainly falls under two kinds of type systems for process calculi: a type systems ensuring a syntactic well-formedness property and a type system ensuring a correct behaviour of a process with regard to a type, respectively. The latter kind is also known under the moniker of behavioural type systems and we concentrate on a specific class of behavioral types called session types.

Sorts The first type system for the pi-calculus was given as a simple sorting discipline of names by Milner in [Mil93]. His system ensures that all uses of a name in a well-sorted process has the same arity. As a consequence of this a well-sorted process does not result in a communication error due to mismatched arities of polyadic input and output channels.

Later Pierce and Sangiorgi [PS93] generalised Milners sort system to subtyping. Their system allows to restrict the use of a channel to be input or output only. They achieve this by defining a subsort relation $\leq$ on sorts marked with a capability of a channel. For example, $\overline{a}\langle b\rangle$ is well-sorted if the sort S_1 of channel a is a subsort of sort S_2^+ of channel b, i.e., $S_1 \leq S_2^+$. The marker + denotes that the name of this sort is used for sending. The way $\leq$ is defined forces S_1 to have the capability of either sending this sort or both sending and receiving. Similarly for the input $a(b)$ is well sorted if $S_1 \leq S_2^-$.

The sorted psi-calculi can be seen as generalisation of the ideas of sorting names (Milner) and using subsort relations to restrict the use of channels (Pierce and Sangiorgi). We generalise the sorting of names to sorting of arbitrary data structures. We do not stipulate a particular relation on channels, thus any relation can be used including subsorting.

In [Hüt11], Hüttel defines a generic type system for psi-calculi. His system is parameterised on a set of types and a set of type judgments rules. The type judgments may depend on the assertions in the type environment. He also extends the syntax of psi-calculi to include the type in the restriction operator.

Hüttel's system enjoys the usual subject reduction property, that is, a well-typed process always transitions into a well typed process, however this only holds for silent transitions. Like in ours, he also shows that the channels capabilities are respected in typed psi-calculi as one a general safety property ensured by his type system. This general system allows for expressing advanced safety properties, such as, authenticity and secrecy properties that of typed spi-calculus.

The typed psi-calculi parameters are limited to freely generated algebras over names, and the substitution functions are required to be homomorphic, i.e., the distributivity law must hold for every substitution σ, $f(M_1, \ldots, M_n)\sigma = f(M_1\sigma, \ldots, M_n\sigma)$ where f is a function symbol and M_i is a term. Whereas we consider every possible psi-calculi, and in this sense sorted psi-calculi is more general than typed psi-calculi.

The name set of typed psi-calculi is sorted into two sets: names and variable names, where only variable names are affected by substitution. Sorted psi-calculi provides foundation for this kind of sorting where standard bisimulation results hold such as structural laws are satisfied, bisimulation is a congruence. In this sense sorted psi-calculi is a precursor of typed psi-calculi.

In [Hüt14], Hüttel gives another powerful parametric type system for psi-calculi. The type system ensures correct resource use, e.g., channel is used linearly. This type system subsumes the linear type system for pi-calculus by Koboyashi et al. [KPT99] among other resource aware process calculi type systems.

Session Types The line of work on session types was initiated by Honda in [Hon93] and, together with Vasconcelos and Kubo, in [HVK98]. These papers introduce the idea that the type describes the interaction of processes thereby the processes conforming to a type interact in dual fashion, and in the latter paper, within a session. In such a interaction, every input is matched by an output and vice versa by each participating process. This ensures that well-typed processes do not deadlock.

Since these systems allow only two processes to interact, they are commonly referred to as binary session types. The idea of binary session types has been extended to multi-party session types [HYC08] where multiple processes interact within a session using multi-cast communication.

Carbone et al. [CHY08] extend binary session types with exceptions and exception handlers. They introduce the session type $\alpha\{\!|\beta|\!\}$, which denotes that a process may throw an exception while following the protocol α asynchronously to signal the other party that they both must continue with the protocol β. This is different from our approach in three ways: first, in our approach processes do not signal other parties of their exceptions and may choose to recover autonomously; second, recovery processes are typed using shared channels thus we don't introduce a recovery session type; and, third, the their processes use reliable communication.

Capecchi et al. [CGY10] extend the session type system with exceptions to multiparty. While broadcast can be regarded as a particular case of multicast, their system does not deal with unreliability.

4.2 Pattern Matching

Pattern matching is a feature found in many functional programming languages like Standard ML, Haskell, Scala, Erlang, among others. It is essential for proving inductive properties in functional programming languages [Bur69]. Languages like Haskell and Scala use patterns with computation, known as view patterns [Wad87]. Abel et al. [APTS13] have introduced a dual notion of the usual structural patterns, called copatterns, to define computation on infinite data.

In process calculi area, the PICT programming language by Pierce and Turner [PT95] based on the pi-calculus defines pattern-matching function for tuples and record patterns, for example, the pattern "`record` $f_1 = p$ `end`" matches value "`record` $f_1 = v_1, f_2 = v_2$ `end`" if the pattern p matches v_1. This kind of system can be easily defined in our pattern-matching framework.

The pattern-matching spi-calculus of Haack and Jeffrey [HJ06] has a similar distinction between pattern variables (bound) and names allowing to decrypt messages without pattern matching unknown keys. Otherwise, their pattern matching system is standard.

The Kell calculus [SS05] like ours is parameterised on a pattern language and the pattern matching device. Patterns can contain pattern variables disjoint from free names. Unlike pattern matching psi-calculi, patterns are required to preserve user defined structural congruence.

Brown et al. [JLM05] introduce an extension to the pi-calculus `PiDuce`. Their calculus have patterns possibly as either XML schemas or XML markup. Patterns which can be a sub-schema or sub-markup are matched against XML markup.

Both Honda [Hon93] and Given-Wilson et al. [GWGJ10] use bidirectional structured patterns for communication based on unification of prefixes instead of standard pattern-matching.

4.3 Verification Tools

Tool lineage for process calculi trace back to Concurrency Workbench by Cleaveland et al. [CPS90] for Milner's Calculus of Communicating Systems [Mil89]. It includes features such as strong equivalence checking and model checking.

The Mobility Workbench by Victor and Moller [VM94, Vic94] is a tool for the polyadic pi-calculus that includes open bisimulation equivalence checking of both kinds weak and strong. Similarly to Pwb, Mobility Workbench uses symbolic methods to obtain effective operational semantics. Mobility Workbench was also extended with model checking capabilities by Beste [Bes98].

There are many tools designed specifically for equivalence checking in process caluli, however less general than Pwb: Another Bisimulation Chekcer (ABC) by Briais [Bri05] for the pi-calculus, Symbolic Bismulation Checker [BB04] by Borgström and Briais that checks behavioral equivalences in the spi-calculus [AG97], and PiET by Meo [Mio06] for the pi-calculus that checks a plethora of strong behavioral equivalences.

Meyer et al. [MKS09] implement a tool called Petruchio with which they model-check finite control pi-calculus agents. They use a translation from the pi-calulus to Petri nets to exploit model-checking methods from Petri nets.

ProVerif [Bla11] is a specialised tool for security protocol verification. Its accepted language is an extension of the applied pi-calculus of Abadi and Fournet [AF01]. It is parameterised over a first order signature over which messages can be built and pattern matched. The data terms are identified up to a user defined term rewriting system. The main feature of ProVerif is reachability and secrecy analysis using Horn Clauses to represent protocols. Recently, ProVerif was extended with behavioral equivalence checking by Cheval and Blanchet, however the equivalences are quite strong: the processes are required to have the same structure and they can only differ in messages sent.

Another tool for process calculi extended with datatypes is mCRL2 [CGK$^+$13] for the Algebra of Communicating Processes, which allows higher order sorted term algebras and equational logic. mCRL2 features architecture consisting of many tools interfacing via intermediate languages. Among these tools, mCRL2 includes an equivalence checker, a visual simulator and a model checker.

PAT3 [LSD11] which includes a CSP♯ [SLDC09] module where actions built over types like booleans and integers are extended with C♯-like programs.

Chapter 5

Future Work

5.1 Algebra of Psi-calculi

A psi-calculus is a structure (Section 2.3)

$$\mathcal{A} = (\mathbf{T}, \mathbf{C}, \mathbf{A}, \dot{\leftrightarrow}, \otimes, \mathbf{1}, \vdash, \mathsf{s}_{\mathbf{T}}, \mathsf{s}_{\mathbf{C}}, \mathsf{s}_{\mathbf{A}})$$

It is then natural to ask, what are the structure-preserving operations on psi-calculi? And if we group them, what kind of algebras does psi-calculi form? Such a theory could have several applications.

Let us illustrate by considering examples of a possible theory. We first can consider unary operations on psi-calculi

$$\mathsf{op}\mathcal{A}$$

Unary operations could be used as refinements on psi-calculi, i.e. we could build more advanced calculi out of basic ones. Take for instance op to be an operation which adds natural numbers and the usual arithmetic operations on them to the term set, and equations on those expression to the conditions. Thus we could obtain a pi-calculus with arithmetic by lifting the pi-calculus instance **Pi** into $\mathsf{op}\,\mathbf{Pi}$.

We have already met this sort of operation in Paper III. It mapped a sorted and pattern matching psi-calculus to a trivially sorted psi-calculi. We used the mapping to lift some bisimulation results for unsorted psi-calculi to the sorted case. Hence this kind of operations have merit in proof theory.

It is also interesting to consider binary operations on psi-calculi $\mathcal{A}$ and $\mathcal{B}$:

$$\mathcal{A} \odot \mathcal{B}$$

What could merging two process calculi mean? We can also consider universal constructions borrowed from universal algebra such as direct products and co-products.

Such a theory could have uses in tools like the Psi-calculi Workbench if we consider symbolic psi-calculi structures together with constraint solvers. Then we could have a modular language for constructing psi-calculi instances with much less effort than we currently do.

5.2 Nominal Algebras for Transition System Specification

The goal of symbolic semantics for value passing process calculi is to reduce infinite transition graphs into finite transition graphs. In other words, to make derivation of transitions computable. This was first proposed by Hennessy and Lin [HL95] for CCS in order to make bisimulation checking feasible.

This technique has been extended to the pi-calculus by Boreale and Nicola [BdN96], and to many other calculi: spi-calculus by Borgström et al. [BBN04], by Chen et al. [CHL05] for a the pi-calculus using a more general notion of symbolic transition graph, and psi-calculi by Johansson et al. [JVP10, JVP12] with an amended version by Borgström et al. (cf. Paper I) used in the Psi-calculi workbench.

All the extensions mentioned above are ad-hoc, case by case considerations. They all share the same idea.

The main source of infinity in the pi-calculus in the early semantics is the input rule:

$$a(x).P \xrightarrow{\underline{a}\,y} P\{y/x\}$$

Which is of course satisfied by every $y \in \mathcal{N}$, that is, y ranges over an infinite domain. The solution for making this branching finite is to use a single name to represent all the possible values received, which in the pi-calculus means using a late semantics, or the following rule

$$a(x).P \xrightarrow{\underline{a}(x)} P$$

This is the choice made by Johansson et al. [JVP10], however, for advanced calculi like psi-calculi this approach does not scale well and it is hard to accommodate extensions defined for early style semantics such as pattern-matching (Section 3.4).

Because a name could represent an uninstantiated value, it is also necessary to record the conditions which could possible be enabled by a particular instantiation. For example, the symbolic version of the match rule in the pi-calculus is decorated with the match and the condition is left to be checked after the derivation is complete.

$$\frac{P \xrightarrow{C,\alpha} P'}{[a = b]P \xrightarrow{[a=b]\wedge C,\alpha} P'}$$

We could formalise this kind of 'know-how' knowledge of lifting structural operational semantics to symbolic versions thereof. Instead of doing this lifting case-by-case basis for each SOS, one could have a transition system specification of a particular calculus and obtain a symbolic transition system which is sound and complete with regard to the original. There are a number of psi-calculi extensions (priorities, reliable broadcast, higher order) that we then would be able to implement in the Psi-calculi Workbench without designing a symbolic semantics for each case.

In summary, following diagram commutes where TSS is a transition system specification, TS is a transition system, and similarly STSS is symbolic transition system specification, and STS is symbolic transition system.

$$\begin{tikzcd}
\text{TSS} \arrow[r, "gen_1"] \arrow[d, "lift"'] & \text{TS} \\
\text{STSS} \arrow[r, "gen_2"'] & \text{STS} \arrow[u, "inst"']
\end{tikzcd}$$

To put it differently, one can obtain the same transition system by lifting the transition system specification to a symbolic one, and then generating symbolic transitions. The obtained symbolic transitions could be initialised to the required transition system, and vice versa (*inst* is one-to-one correspondence), as follows

$$gen_1 = inst \circ gen_2 \circ lift$$

As shown by Bengtson [Ben10], nominal techniques are perfectly suited for name-passing calculi and advanced calculi like psi-calculi. We could express transition system specifications in nominal term algebras [UPG04], and represent the abstractions of quantified values by using the notion of unknown due to Dowek et al. [DGM10].

5.3 Models of Psi-calculi

Psi-calculi could be regarded as a meta model of process calculi due to its powerful theory which captures many calculi. There are other such theories proposed for modelling of concurrency. It is interesting to explore the connection between those theories and psi-calculi.

Coalgebra is a uniform way of expressing great variety of dynamic state based systems among which are transition systems, automata, and process calculi. It is a dual notion of algebra with many of algebraic notions dualised, e.g., congruence in algebra is dual to bisimulation equivalence. Rutten has developed [Rut00] universal coalgebra theory in very same sense as universal algebra theory with general results, e.g., the existence of a final coalgebra. The generality of coalgebras allows to transport results between different systems.

Montanari and Pistone [MP98] have introduced an extension of finite-state automata called history-dependent automata which allowed them to model the pi-calculus and capture its behavioral equivalence. Ferrari et al. [FMT05] used coalgebraic techniques to develop verification methods for the pi-calculus by using the fact that both history dependent automata and pi-calculus are coalgebras. These results could potentially be reused and extended for the psi-calculi and the Psi-calculi workbench given if there is a coalgebra of psi-calculi.

Milner has developed the theory of bi-graphs [Mil01] for what he calls ubiquitous computing. Bi-graphs are combination of multi-graphs and trees capturing the notion of connected components in a hierarchical system. It is a more concrete theory than either psi-calculi or coalgebra, and does not give universal constructions for the bi-graphs themselves. Nevertheless, it is a general theory capturing several well known calculi, e.g., the polyadic pi-calculus [BS06]. It is then interesting to see whether the theory of psi-calculi with the assertion environments could be expressed in bi-graphs. Again transporting results would be of interest, e.g., inferring labelled transition system from bigraphical reductions.

5.4 Logics for Psi-calculi

Models of large systems are usually complex and their correctness is not at all obvious. In order to reason about their correctness, one is concerned with establishing so called liveness and safety properties of a model. Examples of which are absence of livelocks and deadlocks which are of interest in any concurrent system.

The modal μ-calculus is a successful formalism for expressing such properties. The modal μ-calculus is a modal logic with least and greatest fixed-point operators. Modal logic is a logic which allows reasoning about the future states of a system, e.g., the formula $\langle a \rangle \phi$ express the fact in some next state of a system ϕ holds. The fixpoint operators of μ-calculus represent the least and greatest sets of states that satisfy a formula. This allows naturally express liveness and safety properties.

The μ-calculus has been extended to handle the polyadic pi-calculus with a model-checking algorithm by Dam [Dam96], and Beste has implemented a model checker in the Mobility Workbench [Bes98].

Psi-calculi allows expressing models of concurrent systems at a more natural abstraction level in a parametric way. What we propose is to have a family of modal μ-calculi and verification frameworks for psi-calculi. We think that the application of psi-calculi to real-world systems would be greatly helped by such a framework. We plan to develop and use a variant of modal μ-calculus for psi-calculi to verify the cache coherence protocol VIPS [RK12].

Chapter 6

Conclusion

In this thesis, we have presented the results of three papers. First, we have developed the tool Psi-calculi Workbench based on the semantic framework of psi-calculi (Section 2.3), 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. We have investigated the applicability of it to several concurrent systems such as wireless sensor networks by devising abstract executable models for future analysis.

Second, we have defined a system of session types for a calculus based on unreliable broadcast communication. This is the first time that session types have been generalised beyond reliable point-to-point communication. We defined the operational semantics of our calculus by translation into an instantiation of broadcast psi-calculi, and proved subject reduction and safety results. The use of the psi-calculi framework opens the possibility of exploiting its general theory of bisimulation for reasoning about session-typed unreliable broadcasting systems.

Third, we present generalised pattern matching and a sort system for psi-calculi. These two features significantly improve the precision of modelling in psi-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 preserved by the transition system. The meta-theoretic results carry over from the original psi formulations, and many have been machine-checked in the theorem prover Isabelle.

Bibliography

[AF01] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '01, pages 104–115, New York, NY, USA, 2001. ACM.

[AG97] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: the spi calculus. In *Proceedings of the 4th ACM conference on Computer and communications security*, CCS '97, pages 36–47, New York, NY, USA, 1997. ACM.

[APTS13] Andreas Abel, Brigitte Pientka, David Thibodeau, and Anton Setzer. Copatterns: Programming infinite structures by observations. In *Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '13, pages 27–38, New York, NY, USA, 2013. ACM.

[BB04] Sébastien Briais and Johannes Borgström. SBC: Symbolic Bisimulation Checker, 2004.

[BBN04] Johannes Borgström, Sébastien Briais, and Uwe Nestmann. Symbolic bisimulation in the spi calculus. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR 2004 - Concurrency Theory*, volume 3170 of *Lecture Notes in Computer Science*, pages 161–176. Springer Berlin Heidelberg, 2004.

[BdN96] Michele Boreale and Rocco de Nicola. A symbolic semantics for the π-calculus. *Information and Computation*, 126(1):34 – 52, 1996.

[Ben10] Jesper Bengtson. *Formalising process calculi*. PhD thesis, Uppsala University, Division of Computer Systems, 2010.

[Bes98] Fredrick B. Beste. The model prover - a sequent-calculus based modal μ-calculus model checker tool for finite control π-calculus agents. Master's thesis, Department of Computer Systems, Uppsala University, Sweden, March 1998. Available as report DoCS 98/97.

[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(1:11), 01 2011.

[Bla11] Bruno Blanchet. Using Horn clauses for analyzing security protocols. In Véronique Cortier and Steve Kremer, editors, *Formal Models and Techniques for Analyzing Security Protocols*, volume 5 of *Cryptology and Information Security Series*, pages 86–111. IOS Press, March 2011.

[Bri05] Sébastien Briais. Abc: Another bisimulation checker. `http://sbriais.free.fr/tools/abc/`, 2005. Retrieved Sep 1, 2014.

[BS06] Mikkel Bundgaard and Vladimiro Sassone. Typed polyadic pi-calculus in bigraphs. In *Proceedings of the 8th ACM SIGPLAN international conference on Principles and practice of declarative programming*, PPDP '06, pages 1–12, New York, NY, USA, 2006. ACM.

[Bur69] Rodney M. Burstall. Proving properties of programs by structural induction. *The Computer Journal*, 12(1):41–48, 1969.

[CGK$^+$13] Sjoerd Cranen, Jan Friso Groote, Jeroen J. A. Keiren, Frank P. M. Stappers, Erik P. Vink, Wieger Wesselink, and Tim A. C. Willemse. An overview of the mCRL2 toolset and its recent advances. In Nir Piterman and Scott A. Smolka, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 7795 of *Lecture Notes in Computer Science*, pages 199–213. Springer Berlin Heidelberg, 2013.

[CGY10] Sara Capecchi, Elena Giachino, and Nobuko Yoshida. Global Escape in Multiparty Sessions. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 338–351, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[CHL05] Taolue Chen, Tingting Han, and Jian Lu. A modal logic for π-calculus and model checking algorithm. *Electronic Notes in Theoretical Computer Science*, 123(0):19 – 33, 2005.

[CHY08] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured interactional exceptions in session types. In Franck Breugel and Marsha Chechik, editors, *CONCUR 2008 - Concurrency Theory*, volume 5201 of *Lecture Notes in Computer Science*, pages 402–417. Springer Berlin Heidelberg, 2008.

[CM02] Marco Carbone and Sergio Maffeis. On the expressive power of polyadic synchronisation in pi-calculus. *Electronic Notes in Theoretical Computer Science*, 68(2):15 – 32, 2002. EXPRESS'02, 9th International Workshop on Expressiveness in Concurrency.

[CPS90] Rance Cleaveland, Joachim Parrow, and Bernhard Steffen. The concurrency workbench. In Joseph Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 24–37. Springer Berlin Heidelberg, 1990.

[Dam96] Mads Dam. Model checking mobile processes. *Information and Computation*, 129(1):35 – 51, 1996.

[DGM10] Gilles Dowek, Murdoch J. Gabbay, and Dominic P. Mulligan. Permissive nominal terms and their unification: an infinite, co-infinite approach to nominal techniques. *Logic Journal of IGPL*, 18(6):769–822, 2010.

[EM01] Christian Ene and Traian Muntean. A broadcast-based calculus for communicating systems. *Parallel and Distributed Processing Symposium, International*, 3:30149b, 2001.

[FMT05] Gianluigi Ferrari, Ugo Montanari, and Emilio Tuosto. Coalgebraic minimization of hd-automata for the π-calculus using polymorphic types. *Theoretical Computer Science*, 331(2–3):325 – 365, 2005. Formal Methods for Components and Objects.

[GP99] Murdoch Gabbay and Andrew Pitts. A new approach to abstract syntax involving binders. In *Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science*, LICS '99, Washington, DC, USA, 1999. IEEE Computer Society.

[GWGJ10] Thomas Given-Wilson, Daniele Gorla, and Barry Jay. Concurrent pattern calculus. In CristianS. Calude and Vladimiro Sassone, editors, *Theoretical Computer Science*, volume 323 of *IFIP Advances in Information and Communication Technology*, pages 244–258. Springer Berlin Heidelberg, 2010.

[HJ06] Christian Haack and Alan Jeffrey. Pattern-matching spi-calculus. *Information and Computation*, 204(8), 2006.

[HL95] Matthew Hennessy and Huimin Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138(2):353 – 389, 1995.

[Hon93] Kohei Honda. Types for dyadic interaction. In Eike Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993.

[HT91] Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. In Pierre America, editor, *ECOOP'91 European Conference on Object-Oriented Programming*, volume 512 of *Lecture Notes in Computer Science*, pages 133–147. Springer Berlin Heidelberg, 1991.

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

[Hüt14] Hans Hüttel. Types for resources in psi-calculi. In Martín Abadi and Alberto Lluch Lafuente, editors, *Trustworthy Global Computing*, Lecture Notes in Computer Science, pages 83–102. Springer International Publishing, 2014.

[HVK98] Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In Chris Hankin, editor, *Programming Languages and Systems*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer Berlin Heidelberg, 1998.

[HYC08] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *Proc. of POPL'08*, pages 273–284. ACM Press, 2008.

[JBPV10] Magnus Johansson, Jesper Bengtson, Joachim Parrow, and Björn Victor. Weak equivalences in psi-calculi. In *Logic in Computer Science (LICS), 2010 25th Annual IEEE Symposium on*, pages 322–331, July 2010.

[JLM05] Allen L. Brown Jr., Cosimo Laneve, and L. Gregory Meredith. PiDuce: A Process Calculus with Native XML Datatypes. In Mario Bravetti, Leïla Kloul, and Gianluigi Zavattaro, editors, *Formal Techniques for Computer Systems and Business Processes*, volume 3670 of *Lecture Notes in Computer Science*, pages 18–34. Springer Berlin Heidelberg, 2005.

[JVP10] Magnus Johansson, Björn Victor, and Joachim Parrow. A fully abstract symbolic semantics for psi-calculi. In *Proc. 6th Workshop on Structural Operational Semantics : SOS 2009*, number 18 in Electronic Proceedings in Theoretical Computer Science, 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, 4 2012.

[KPT99] Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the pi-calculus. *ACM Trans. Program. Lang. Syst.*, 21(5):914–947, September 1999.

[LSD11] Yang Liu, Jun Sun, and Jin Song Dong. PAT 3: An extensible architecture for building multi-domain model checkers. In *Proc. of ISSRE '11*, pages 190–199, Los Alamitos, CA, USA, 2011. IEEE.

[MFHH02] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131–146, December 2002.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[Mil92] Robin Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2:119–141, 6 1992.

[Mil93] Robin Milner. The polyadic π-calculus: A tutorial. In Friedrich L. Bauer, Wilfried Brauer, and Helmut Schwichtenberg, editors, *Logic and Algebra of Specification*, volume 94 of *Series F*. NATO ASI, Springer, 1993.

[Mil01] Robin Milner. Bigraphical reactive systems. In KimG. Larsen and Mogens Nielsen, editors, *CONCUR 2001 — Concurrency Theory*, volume 2154 of *Lecture Notes in Computer Science*, pages 16–35. Springer Berlin Heidelberg, 2001.

[Mio06] Matteo Mio. Piet: Pi calculus equivalences tester. `http://piet.sourceforge.net`, 2006. Retrieved Sep 1, 2014.

[MKS09] Roland Meyer, Victor Khomenko, and Tim Strazny. A practical approach to verification of mobile systems using net unfoldings. *Fundamenta Informaticae*, 94(3):439–471, 01 2009.

[MP98] Ugo Montanari and Marco Pistore. An introduction to history dependent automata. *Electronic Notes in Theoretical Computer Science*, 10(0):170 – 188, 1998. HOOTS II, Second Workshop on Higher-Order Operational Techniques in Semantics.

[MPW92a] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, September 1992.

[MPW92b] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, II. *Inf. Comput.*, 100(1):41–77, September 1992.

[Par01] Joachim Parrow. *An Introduction to the pi-Calculus*, volume 19, pages 8–10. Elsevier, 2001.

[Pit03] Andrew M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186(2):165 – 193, 2003. Theoretical Aspects of Computer Software (TACS 2001).

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

[PS93] Benjamin Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. In *Logic in Computer Science, 1993. LICS '93., Proceedings of Eighth Annual IEEE Symposium on*, pages 376–385, Jun 1993.

[PT95] Benjamin C. Pierce and David N. Turner. Concurrent objects in a process calculus. In Takayasu Ito and Akinori Yonezawa, editors, *Theory and Practice of Parallel Programming*, volume 907 of *Lecture Notes in Computer Science*, pages 187–215. Springer Berlin Heidelberg, 1995.

[RK12] Alberto Ros and Stefanos Kaxiras. Complexity-effective multicore coherence. In *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, PACT '12, pages 241–252, New York, NY, USA, 2012. ACM.

[Rut00] Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3 – 80, 2000.

[SLDC09] Jun Sun, Yang Liu, Jin Song Dong, and Chunqing Chen. Integrating specification and programs for system modeling and verification. In *Proc. TASE '09*, pages 127–135. IEEE, 2009.

[SS05] 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 Berlin Heidelberg, 2005.

[SW01] Davide Sangiorgi and David Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[UPG04] Christian Urban, Andrew M. Pitts, and Murdoch J. Gabbay. Nominal unification. *Theoretical Computer Science*, 323(1–3):473 – 497, 2004.

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

[Vic94] Björn Victor. *A Verification Tool for the Polyadic pi-Calculus*. Licentiate thesis, Department of Computer Systems, Uppsala University, Sweden, May 1994. Available as report DoCS 94/50.

[VM94] Björn Victor and Faron Moller. The mobility workbench – a tool for the π-calculus. In DavidL. Dill, editor, *Computer Aided Verification*, volume 818 of *Lecture Notes in Computer Science*, pages 428–440. Springer Berlin Heidelberg, 1994.

[Wad87] Philip Wadler. Views: A way for pattern matching to cohabit with data abstraction. In *Proceedings of the 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '87, pages 307–313, New York, NY, USA, 1987. ACM.

[WG05] Lucian Wischik and Philippa Gardner. Explicit fusions. *Theoretical Computer Science*, 304(3):606–630, 2005.

Paper I

The Psi-Calculi Workbench: a Generic Tool for Applied Process Calculi

Submitted to Special Issue on Application of Concurrency to System Design

Johannes Borgström, Ramūnas Gutkovas, Ioana Rodhe and Björn Victor, Uppsala University

Psi-calculi is a parametric framework for extensions of the pi-calculus with arbitrary data, and logic. All instances of the framework inherit machine-checked proofs of the meta-theory such as compositionality and bisimulation congruence. We present a generic analysis tool for psi-calculus instances, enabling symbolic execution and (bi)simulation checking for both unicast and broadcast communication. The tool also provides a library for implementing new psi-calculus instances. We provide examples from traditional communication protocols and wireless sensor networks. We also describe the theoretical foundations of the tool, including an improved symbolic operational semantics, with additional support for scoped broadcast communication.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Protocol Verification*; D.2.2 [**Software Engineering**]: Design tools and techniques; I.1.4 [**Symbolic and Algebraic Manipulation**]: Applications

General Terms: Design, Theory, Verification

Additional Key Words and Phrases: Wireless sensor networks, process calculi, symbolic semantics

ACM Reference Format:
Johannes Borgström, Ramūnas Gutkovas, Ioana Rodhe and Björn Victor, 2014. The Psi-Calculi Workbench: a Generic Tool for Applied Process Calculi. *ACM Trans. Embedd. Comput. Syst.* 999, 9999, Article 99999 (Month 2014), 25 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

The development of concurrent systems is greatly helped by the use of precise and formal models of the system. There are many different formalisms for concurrent systems, often in specialised versions for particular application areas. For each formalism, tool support is necessary for constructing and reasoning about models of non-trivial systems. This paper describes such tool support for a generic semantic framework for process calculi with mobility. Thus, instead of developing a separate tool for each separate process calculus, we develop one single generic tool for a whole family of process calculi.

Psi-calculi [Bengtson et al. 2011] is a parametric semantic framework based on the pi-calculus [Milner et al. 1992a], adding the possibility to tailor the data language and logic for each application. The framework provides a variety of features, such as lexically scoped local names for resources, communication channels as data, both unicast

This work has been supported by the ProFun project. Author's addresses: Uppsala University, Dept. of IT, Box 337, 751 05 Uppsala, Sweden.

DOI:http://dx.doi.org/10.1145/0000000.0000000

and broadcast communication [Borgström et al. 2011], and both first- and higher-order communication [Parrow et al. 2013].

Many of the different extensions of the pi-calculus, including the spi-calculus [Abadi and Gordon 1997], the fusion calculus [Wischik and Gardner 2005], the concurrent constraint pi-calculus [Buscemi and Montanari 2007], and the polyadic synchronisation pi-calculus [Carbone and Maffeis 2003], can be directly represented as instances of the psi-calculi framework. A major advantage is that all meta-theoretical results, including algebraic laws and congruence properties of bisimilarity, apply to any valid instantiation of the framework. Additionally, most of these results have been proved with certainty, using the Nominal Isabelle theorem prover [Urban and Tasson 2005]. These features of psi-calculi save a lot of effort for anyone using it — psi-calculi is a reusable framework.

This paper describes the Psi-Calculi Workbench (PWB), a generic tool for implementing psi-calculus instances, and for analysing processes in the resulting instances. While there are several other tools, specialised for particular process calculi and particular application areas, our tool is generic and reusable. It has a wider scope than previous works, and also allows experimentation with new process calculi with a relatively low effort. Like psi-calculi, our tool is parametric: it provides functionality for bisimulation equivalence checking and symbolic simulation (or execution) of processes in any psi instance, and a base library for implementing new psi-calculi instances. PWB thus has two types of users: the user analysing systems in an existing instance of the framework, and the instance implementor.

We illustrate both uses of the tool in three steps: In Section 2 we introduce the framework of psi-calculi semiformally, relating an instance corresponding to the pi-calculus and showing symbolic simulation of agents. After describing the design of PWB and how to implement an instance in Section 3, we show how to add data and computation in Section 4 by modelling the traditional alternating bit protocol for reliable communication. In Section 5 we model a data aggregation protocol for wireless sensor networks, incorporating specialised data structures and logics, and both unicast and broadcast communication. Section 6 extends the previous example with a dynamic topology.

In Section 7 we describe the symbolic semantics implemented in PWB. The symbolic operational semantics of Section 7.1 simplifies previous symbolic semantics for psi-calculi [Johansson et al. 2012], and adds rules for wireless (synchronous and unreliable) broadcast [Borgström et al. 2011]. To our knowledge, this is the first symbolic semantics for lexically scoped broadcast communication.

In Section 8 we discuss related work. An abridged version of this article was published as [Borgström et al. 2013].

2. INTRODUCING PSI-CALCULI

In this section we introduce the psi-calculi parametric semantic framework semiformally, and defer some precise definitions and the operational semantics to Section 7. For a more extensive treatment of psi-calculi, including motivations of the requisites and examples of other instances see [Bengtson et al. 2011; Borgström et al. 2011; Johansson et al. 2012; Johansson et al. 2010]. We show more complex examples in Sections 4, 5 and 6.

A psi-calculus instance is specified by three data types: the (data) terms $\mathbf{T}$, ranged over by M, N, the conditions $\mathbf{C}$, ranged over by φ, and the assertions $\mathbf{A}$, ranged over by Ψ. The terms, conditions and assertions can be any sets where the elements may contain names (from the set $\mathcal{N}$ of names) and name permutations are admitted (so-called *nominal sets* [Pitts 2003]). In particular, every element X has a finite set of free names $\mathsf{n}(X) \subseteq \mathcal{N}$, and we write $a\#X$ for $a \notin \mathsf{n}(X)$.

Terms are used both as communication channels, and for the data sent and received in communication. They can be structured, and so permit standard constructs as lists and sets, numbers and booleans, as well as more advanced structures. Assertions are used to model "facts" about terms and relations between them, for instance by giving values to variables or by constraining their values. The minimal assertion is the unit, written $\mathbf{1}$, and assertions are composed by the $\otimes$ operator. Conditions are used to perform tests on terms. Their outcome depends on the current assertion environment, through an entailment relation (Ψ entails φ, written $\Psi \vdash \varphi$) which is also part of the psi instance specification.

In the **Pi** instance, corresponding to the polyadic pi-calculus, terms are simply names $a, b, c \ldots$ and the conditions are equality tests on names. (Name equality is used in the match construct $[a = b]P$, which behaves as P if $a = b$ holds.) In the pi-calculus there are no assertions, but the psi-calculi framework requires at least the trivial unit assertion. Later examples will show how assertions can be exploited for modelling advanced features.

Given the psi-calculus parameters $\mathbf{T}, \mathbf{C}, \mathbf{A}$, the *agents*, ranged over by $P, Q, \ldots$, are of the following forms:

$\overline{M}\,\tilde{N}\,.\,P$	Output prefix
$\underline{M}(\tilde{x})\,.\,P$	Input prefix
$\overline{M}!\tilde{N}\,.\,P$	Broadcast output prefix
$\underline{M}?\,(\tilde{x})\,.\,P$	Broadcast input prefix
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
$A\langle\widetilde{M}\rangle$	Invocation

We write $\widetilde{M}$ for the tuple $M_1, \ldots M_n$. The output and input prefixes denote polyadic (unicast) output and input, while the broadcast prefixes denote (synchronous) broadcast output and input, which is unreliable (as in wireless systems) in the sense that transmissions might not be received. The case construct can act as any P_i such that the corresponding condition φ_i is true; the other cases are discarded. Restriction binds a in P and input prefixes bind $\tilde{x}$ in the suffix; we identify alpha-equivalent agents. The Invocation form invokes a process A, defined by the form $A(\tilde{y}) \Leftarrow P$; the behaviour is that of $P\{\widetilde{M}/\tilde{y}\}$.

In the **Pi** instance, the output and input prefixes are the usual $\overline{a}\,\tilde{x}\,.\,P$ and $\underline{a}(\tilde{x})\,.\,P$; the match construct $[a = b]P$ corresponds to **case** $a = b : P$. If we have a condition **true** which is always true, we can model nondeterministic choice (traditionally written $P + Q$) as **case true** $: P \;[]\;$ **true** $: Q$.

The semantics for psi-calculi is defined by a labelled transition relation written $\Psi \rhd P \xrightarrow{\alpha} P'$, meaning that in environment Ψ agent P can do an action α to become P'. In the pi-calculus instance, the environment Ψ is always the trivial $\mathbf{1}$, but in general it represents the assertions of the environment, including parallel agents.

The semantics is defined only for well-formed agents. An occurrence of a subterm in an agent is *guarded* if it is a proper subterm of a prefix form. An agent is *well-formed* if in $\underline{M}(\widetilde{x}).P$ and $\underline{M}?\,(\widetilde{x}).P$ it holds that $\widetilde{x}$ is a sequence without duplicates, that in **case** $\varphi_1 : P_1 \;[]\; \cdots \;[]\; \varphi_n : P_n$ the agents P_i have no unguarded assertions, and that in a replication $!P$ the agent P has no unguarded assertions or broadcast input prefixes. For process definitions a similar requirement as for replication applies.

The actions are input $\underline{M}\ (\tilde{x})$ denoting the reception of data bound to $\tilde{x}$ over the channel denoted by M, output $\overline{M}\ (\nu\tilde{x})\tilde{N}$ denoting the sending of $\tilde{N}$ over M and additionally opening the scopes of the names $\tilde{x}$, the corresponding broadcast actions $\underline{M}?(\tilde{x})$ and $\overline{M}!\,(\nu\tilde{x})\tilde{N}$, and the silent action τ which is the result of communication between an input and an output. When $\tilde{x}$ is empty, we often omit $(\nu\tilde{x})$ and $(\tilde{x})$.

The connectivity predicates used for communication are also defined by the instantiation. The conditions include the *channel equivalence* predicate $M \dot{\leftrightarrow} N$ which is used to define which terms denote the same unicast channel, and the broadcast connectivity predicates $M \dot{\prec} K$ and $K \dot{\succ} M$ for sending and receiving on broadcast channels: a term M can be used to send a broadcast message on the channel K only if $M \dot{\prec} K$ in the current assertion environment, and similar for broadcast reception (see Section 5 for an example).

As an example, the **Pi** agent

$$\overline{b}\,c.Q \mid \underline{b}(a).\mathbf{case}\ a = b : \underline{a}(z).R$$

has transitions labelled $\overline{b}c$, $\underline{b}\ (x)$ for all names x, and τ. The input prefix can generate infinitely many input actions (here one for each x). To avoid this infinite branching, we use a *symbolic* semantics in the tool (see Section 7.1), where the actual values are abstracted by variables. Instead each transition has a *transition constraint*, which must be satisfied for the corresponding non-symbolic transitions to be possible. Formally these transitions are written $P \xrightarrow[C]{\alpha} P'$ where C is a transition constraint.

The input transitions of the agent above can be represented by a single transition in the symbolic semantics. For simplicity we show the first two transitions of the input prefix subagent:

$$P = \underline{b}(a)\,.\,\mathbf{case}\ a = b : \underline{a}(z).R \xrightarrow[\{\!|\mathbf{1}\vdash b\dot{\leftrightarrow}w|\!\}]{\underline{w}\ (a)} \mathbf{case}\ a = b : \underline{a}(z).R \xrightarrow[\{\!|\mathbf{1}\vdash a\dot{\leftrightarrow}v|\!\}\wedge\{\!|\mathbf{1}\vdash a=b|\!\}]{\underline{v}\ (z)} R$$

where w and v are fresh (see Section 7 for the formal semantics). The constraint of the first transition intuitively says that the channel w is equivalent to b (there may not always be such a w!); for the second transition a similar constraint appears in addition to the condition of the case construct.

We can use the PWB to simulate the transitions of P. The tool uses an ASCII representation of agents, where non-alphanumeric terms and conditions must be in double quotes, ν is written new, output objects are written between angular brackets and the overline in outputs is written by a preceding single quote. For example, $\overline{b}\,\mathrm{f}(a,c)\,.\,(\nu x)Q$ is written 'b<"f(a,c)">.(new x)Q.

The first transition of the agent P above:

```
--|gna(a)|-->

  Source:
    b(a). case "a = b" : a(x). R<>
  Constraint:
    {| "b = gna" |}
  Solution:
    ([gna := b], 1)
  Derivative:
    case "a = b" : a(x). R<>
```

When printing the constraint, the trivial $\mathbf{1}\vdash$ is elided. The “gna” here represents a fresh name, corresponding to w above: the subject of the symbolic input action.

The derivative **case** "a = b" : a(x).R does not have a non-symbolic transition since a is not the same name as b, but the symbolic semantics does have a transition *under the constraint* that a=b.

```
−−|gnb(x)|−−>

    Source:
      case "a = b" : a(x). R<>
    Constraint:
      {| "a = gnb" |} ∧  {| "a = b" |}
    Solution:
      ([b := a, gnb := a], 1)
    Derivative:
      R<>
```

The constraint {| "a = b" |} can be solved by substituting a for b, as stated by the Solution line above. The solution is generated by a constraint solver module in the PWB, which for the pi-calculus instance performs name unification (see Section 3.2), similar to earlier tools for pi-related calculi (e.g. MWB). After applying the solution to the agent, there is a corresponding non-symbolic transition.

In addition to symbolic execution, the PWB also includes a symbolic checker that computes a minimal sufficient constraint for one agent to be (bi)similar to another, plus a witnessing relation. The two agents are non-symbolically related after applying a solution to the constraint (if there is one).

3. IMPLEMENTATION

The Psi-Calculi Workbench (PWB) is implemented in the Standard ML programming language and compiles under the Poly/ML compiler [PolyML 2013] version 5.4. PWB is open source and freely available online from [Gutkovas and Borgström 2013].

PWB is a modular implementation of psi-calculi, and can be viewed both as a modelling tool and as a library for building tools for particular instances of psi-calculi. Used as a modelling tool, the user interacts with a command interpreter that provides commands for process definitions (manually or from files), manipulation of the process environment, stepping through symbolic (strong and weak) transitions of a process, and symbolic bisimilarity checking (strong and weak). Examples of such use are given in sections 4 and 5. Below we describe the implementation of PWB and the modules which need to be provided when creating an instance of psi-calculi.

3.1. Psi-Calculus instantiation

PWB implements a number of helper libraries for the instance implementor. We show the architecture of PWB in Figure 1. In this figure, dependencies between components go from right to left: each component may depend only on components that are above it or to its left. All components build on the supporting library that provides the basic data structures and core algorithms for psi-calculi. The instance implementor provides definitions for the parameters of an instance, constraint solvers, and parsing and pretty-printing code. These user-implemented components are then called by the different algorithms implemented by the tool and by the command interpreter. Not all components are required to be implemented: for instance, the bisimulation constraint solver is only needed for bisimilarity checking.

The parameters of an instance consist of the types name, term, condition and assertion, and three classes of functions: those defining the logics, the substitutions, and the connectivity. As an example of the types, here are the declarations for the pi-calculus instance mentioned in Section 2. All SML code presented is written by the instance implementor.

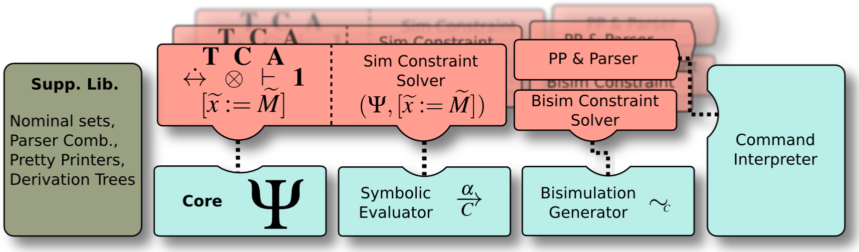

Supp. Lib.
Nominal sets,
Parser Comb.,
Pretty Printers,
Derivation Trees
T C A
$\dot{\leftrightarrow}$ $\otimes$ $\vdash$ 1
$[\widetilde{x} := \widetilde{M}]$
Sim Constraint Solver
$(\Psi, [\widetilde{x} := \widetilde{M}])$
PP & Parser
Bisim Constraint Solver
Command Interpreter
Core Ψ
Symbolic Evaluator $\xrightarrow[C]{\alpha}$
Bisimulation Generator $\sim_c$

Fig. 1: Psi-Calculi Workbench Architecture

```
type term          = name
datatype condition = Eq of term * term | T
datatype assertion = Unit
```

We need three functions to define the logic of the instance: entailment (entails, or $\vdash$) that describes which conditions are true given an assertion, a composition operator (compose, or $\otimes$) that composes two assertions, and a unit assertion (unit, or **1**). We require that assertion composition forms a commutative monoid (modulo entailment), and that all functions are equivariant, meaning that they treat all names equally. The bisimulation algorithm and the weak symbolic semantics also require weakening to hold, meaning that $\Psi \vdash \varphi$ implies $\Psi \otimes \Psi' \vdash \varphi$ for all Ψ'.

```
val entails : assertion * condition -> bool
val compose : assertion * assertion -> assertion
val unit    : assertion
```

We also need equivariant substitution functions, substituting terms for names in each of term, condition and assertion.

```
type subst = (name * term) list
val substT : subst -> term      -> term
val substC : subst -> condition -> condition
val substA : subst -> assertion -> assertion
```

Finally, we have three equivariant functions that describe the connectivity of the calculus: chaneq (for unicast connectivity), brTransmit and brReceive (for broadcast). Typically, these functions are simple injections into the conditions type (e.g., **fun** chaneq (M,N) = ChanEq (M,N) where ChanEq is a data constructor of condition) leaving the definition of connectivity to either the entailment relation or the constraint solver.

Channel equivalence chaneq is required to be commutative and transitive (for every Ψ). brTransmit is broadcast output connectivity $\dot{\prec}$ and brReceive is broadcast input connectivity $\dot{\succ}$; these functions are exemplified in Section 5. If Ψ entails $M \dot{\prec} K$ or Ψ entails $K \dot{\succ} M$, then we require all names that occur in K to also occur in M.

```
val chaneq     : term * term -> condition
val brTransmit : term * term -> condition
val brReceive  : term * term -> condition
```

All of the functions above are further required to commute with substitution, in the sense that $f(X\sigma) = f(X)\sigma$.

The user also needs to implement parsers for each of the data types, that are called by the parser for process terms.

3.2. Symbolic execution

PWB provides symbolic execution of processes by the sstep command. This is a useful tool to explore the properties of a process, or indeed the model itself. Here values input by the process are represented by variables, and constraints are collected along the derivation of a transition. The constraints show under which conditions transitions are possible, deferring instantiation of variables as long as possible. Both strong and weak (ignoring τ-transitions) symbolic semantics are available (presented in Section 7).

In psi-calculi, parallel contexts that contain an assertion, such as $(\!|x = 3|\!)$, can enable additional transitions. Therefore, a solution (σ, Ψ) to a constraint consists of a substitution σ (representing earlier inputs) and an assertion Ψ (representing the parallel context). Intuitively, every solution (σ, Ψ) solves true, there is no solution to false, every solution to both C and C' is a solution to $C \wedge C'$, and the solutions to $(\nu\tilde{a})\{\!|\Psi' \vdash \varphi|\!\}$ are the pairs (σ, Ψ) where $\Psi \otimes \Psi'\sigma \vdash \varphi\sigma$ and the names in $\tilde{a}$ do not occur in Ψ, σ.

The instance implementor may provide a constraint solver for the transition constraints. The solver should return either a string describing the unsatisfiability of a constraint, or a solution consisting of a substitution and assertion. Since transition constraints are simply a conjunction of atomic constraints, a simple unification-based solver often suffices. The type of the solver is the following:

```
val solve : constraint -> (string, (name * term) list * assertion) either
```

As an example, the solver for the pi-calculus instance of Section 2 performs unification, implemented by the transition relation below. The nodes in the transition system are either a pair (C, σ), or the failed state $\varnothing$.

$$\begin{array}{rcll}
(\nu\tilde{a})\{\!|\mathbf{1} \vdash \mathrm{T}\ |\!\} \wedge C, \sigma & \rightarrow & C, \sigma & \\
(\nu\tilde{a})\{\!|\mathbf{1} \vdash a = a|\!\} \wedge C, \sigma & \rightarrow & C, \sigma & \\
(\nu\tilde{a})\{\!|\mathbf{1} \vdash a = b|\!\} \wedge C, \sigma & \rightarrow & \varnothing & \text{if } a \neq b \wedge (a \in \tilde{a} \vee b \in \tilde{a}) \\
(\nu\tilde{a})\{\!|\mathbf{1} \vdash a = b|\!\} \wedge C, \sigma & \rightarrow & C[b := a], \sigma[b := a] & \text{otherwise}
\end{array}$$

3.3. Symbolic (bi)simulation

PWB can also be used to check simulation relations on processes. As an example, the command P ~ Q attempts to construct a bisimulation relation relating agents P and Q. To this end, we implement a symbolic bisimulation algorithm based on [Johansson et al. 2012] (with some corrections and optimisations). This algorithm takes two processes and yields a constraint in an extended constraint language; the two processes are bisimilar under all solutions to the constraint. A simple variation of the algorithm is used for simulation checking.

The language for bisimulation constraints additionally includes conjunction, disjunction and implication, as well as constraints for term equality $\{\!|M = N|\!\}$, freshness $\{\!|a \# X|\!\}$ (with the intuition "a is not free in X"), and static implication. In order to simplify the development of a constraint solver for this richer language, PWB contains an SMT solver library with suitable helper functions. Unless the assertion language is trivial (only the unit assertion), most of the additional effort in extending a solver for transition constraints to one for bisimulation constraints lies in properly treating static implication constraints.

4. THE ALTERNATING BIT PROTOCOL

In this section, we describe the modelling in PWB of the classical Alternating Bit Protocol. We demonstrate that the PWB allows to define a tailor-made process calculus for a particular problem or problem domain. We also give an example of symbolic weak transition generation in PWB.

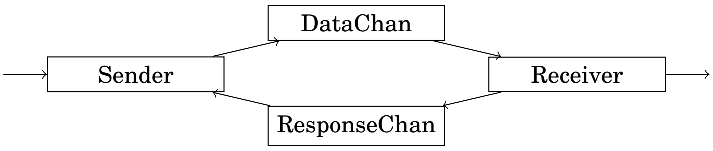


Fig. 2: Alternating Bit Protocol scheme

4.1. Introduction to the Alternating Bit Protocol

The Alternating Bit Protocol (ABP) [Bartlett et al. 1969] is a simple network protocol for reliable data transmission through lossy channels. Reliable here means that all data fragments are received exactly once and in the right order at the receiver. Consider a sender Sender, a receiver Receiver and two communication channels between them: DataChan, over which data fragments are sent, and ResponseChan, over which acknowledgements are sent. We show this situation in Figure 2: the arrows denote the direction of the data being transmitted. ABP assumes reliable error detection, but no error correction.

To ensure that Receiver receives every fragment despite lossy communication channels, Sender repeatedly sends the same fragment until it receives a corresponding acknowledgment, at which point the sender starts transmitting the next fragment. Since the receiver should not accept the same fragment twice, a protocol is needed for distinguishing between packets. In ABP, each data packet has a one-bit flag attached to it. The flag 0 is attached to the first packet sent; the acknowledgment of the receiver for this packet will also have flag bit 0. When Sender receives an acknowledgment with flag 0, it knows that Receiver has correctly received the fragment, and Sender will then start sending the next packet with flag bit 1, and so on. Thus, sequences of sent or received packages resp. acknowledgments with the same flag bit all refer to the same data fragment.

4.2. A Psi-calculus Instance for ABP

To define a psi-calculus instance where ABP can be expressed, we start with the data terms. Since the behaviour of the protocol does not depend on the data being transmitted, we simply represent each fragment as a name. However, the protocol itself needs some data values and structures.

In the set of terms we include the channels DataChan and ResponseChan, and the value ERR to signify that an error has been detected. We also have 0 and 1 bits and a negation operation $\sim\cdot$ on them with the expected equalities $\sim 0 = 1$ and $\sim 1 = 0$.

Our account of ABP is untyped, so these term constructors yield terms which are not intended to be part of the model, such as $\sim$ERR. Such spurious terms yield the invalid value $\bot$. In summary, we define the data terms $\mathbf{T}$ as follows:

Notation	*SML*	Pwb
$Val \triangleq \{\mathrm{ERR}, 0, 1\}$ $\mathbf{T} \triangleq Val \cup \{\bot\} \cup \mathcal{N}$ $\cup \{\sim M : M \in \mathbf{T}\}$	**datatype** term = Error | Zero | One | Bottom | Name **of** name | Neg **of** term	M ::= ERR | 0 | 1 | _|_ | *Name* | ~M

Here and in subsequent displays, the column *Notation* is the mathematical notation, *SML* is the code written by the instance implementor, and Pwb is the ASCII syntax used in the tool by the user of the instance.

Next we define the conditions. In the protocol, we need to compare the sender's or receiver's bit with a transmitted bit, and to see whether an error occurred while transmitting data. To do this, we use equality $=$ on values.

We add a condition True which always holds, and a False condition that never holds. Lastly, we include a channel equivalence condition for unicast communication (ABP does not use broadcast, so we let the broadcast connectivity predicates yield False).

Notation	*SML*	PWB

$\mathbf{C} \triangleq \{\mathrm{True}, \mathrm{False}\}$
$\cup \ \{M = N : M, N \in \mathbf{T}\}$
$\cup \ \{M \dot{\leftrightarrow} N : M, N \in \mathbf{T}\}$
	`datatype condition`
	`= CTrue | CFalse`
	`| Equal of term * term`
	`| ChEq of term * term`
		φ ::= True | False
		| M = M
		| M <-> M

We do not need assertions to model the ABP, so we let $\mathbf{A} = \{\mathrm{Unit}\}$ as in Section 2.

As the last step we define the substitution functions on terms and conditions. They are standard capture avoiding substitutions, followed by normalisation with respect to a term rewriting system given below. We use rewriting after substitutions in order to accurately detect loops of τ-transitions when computing weak transitions. This also significantly simplifies the constraint solver, since the normal forms are simpler to handle than arbitrary terms.

Below, we give the rewrite system for terms for reduction context $\mathcal{R} ::= [] | {\sim}\mathcal{R}$. It evaluates the ${\sim}\cdot$ operator, cancels out double negation of variables, and identifies the spurious terms. In particular, the term ${\sim\sim}$ERR is spurious, and is rewritten to $\bot$.

$$\begin{array}{rcl} {\sim}\mathrm{ERR} & \to & \bot \\ {\sim}\bot & \to & \bot \end{array} \qquad \begin{array}{rcl} {\sim}0 & \to & 1 \\ {\sim}1 & \to & 0 \end{array} \qquad {\sim\sim}x \ \to \ x \ \text{ if } x \in \mathcal{N}$$

The following is the term rewriting system for the conditions. Equalities involving spurious terms $\bot$ are rewritten to False. Note that we only consider equality conditions where the constituent terms are already in normal form; this suffices since the substitution function on conditions is defined in terms of substitution function on terms.

$$\begin{array}{rcl} {\sim}x = {\sim}y & \to & x = y \\ {\sim}x = x & \to & \mathrm{False} \\ x = {\sim}x & \to & \mathrm{False} \end{array} \qquad \begin{array}{rcl} M = N & \to & \mathrm{True} \ \text{ if } M = N \text{ and } \{M, N\} \subseteq \mathit{Val} \cup \mathcal{N} \\ M = N & \to & \mathrm{False} \ \text{ if } M \neq N \text{ and } \{M, N\} \subseteq \mathit{Val} \\ M = N & \to & \mathrm{False} \ \text{ if } \bot \in \{M, N\} \end{array}$$

Finally, we need to define entailment. For conditions in normal form we define

$$\mathrm{Unit} \vdash a \dot{\leftrightarrow} b \text{ iff } a = b \qquad \mathrm{Unit} \vdash M = N \text{ iff } M = N \qquad \mathrm{Unit} \vdash \mathrm{True},$$

and otherwise we let $\mathrm{Unit} \vdash \varphi$ iff $\varphi \to^{+} \varphi' \not\to$ and $\mathrm{Unit} \vdash \varphi'$

4.3. Constraint Solver for ABP Transition Constraints

The ABP constraint solver is a standard unification algorithm defined as a transition system. The design is greatly simplified by the fact that the conditions in the constraints are in normal form.

The following is the unification transition system. The first two rules are trivial. The rules concerning the channel equivalence $\dot{\leftrightarrow}$ condition are the classic unification on names as seen in the pi-calculus solver. The last rules concern the equality condition $=$. Because the terms are in the normal form, we know that one of the sides is a name, and thus we do elimination, or swapping in order to allow elimination. Below, $\tilde{a}\#X$

denotes that names $\tilde{a}$ don't occur freely in X; we omit $\mathbf{1} \vdash$ in front of every condition.

$$\begin{array}{rcll}
(\nu\tilde{a})\{\!|\mathrm{True}|\!\} \wedge C, \sigma & \to & C, \sigma & \\
(\nu\tilde{a})\{\!|\mathrm{False}|\!\} \wedge C, \sigma & \to & \varnothing & \\
(\nu\tilde{a})\{\!|a \dot{\leftrightarrow} b|\!\} \wedge C, \sigma & \to & C, \sigma & \text{if } a = b \text{ and } a, b \in \mathcal{N} \\
(\nu\tilde{a})\{\!|a \dot{\leftrightarrow} b|\!\} \wedge C, \sigma & \to & C[b := a], \sigma[b := a] & \text{if } \tilde{a}\#a, b \text{ and } a \neq b \\
(\nu\tilde{a})\{\!|a \dot{\leftrightarrow} b|\!\} \wedge C, \sigma & \to & \varnothing & \text{otherwise} \\
(\nu\tilde{a})\{\!|a = M|\!\} \wedge C, \sigma & \to & C[a := M], \sigma[a := M] & \text{if } \tilde{a}\#a, M \text{ and } a \in \mathcal{N} \\
(\nu\tilde{a})\{\!|M = N|\!\} \wedge C, \sigma & \to & (\nu\tilde{a})\{\!|N = M|\!\} \wedge C, \sigma & \text{otherwise}
\end{array}$$

4.4. The ABP as a Process

Here we present the process modelling the ABP in the ABP psi-calculus instance defined above. We give the definition in PWB syntax, which is used by the user of the psi instance.

We model the components Sender and Receiver of ABP shown in Figure 2 as psi-calculus processes. The behaviour of components DataChan and ResponseChan are captured implicitly in our model. For composing the system, components have input and output channels inp and out, respectively. The Receiver and Sender each have one additional channel for output o resp. input i to the application that uses the protocol.

The sender is modelled as follows: first it inputs data on input channel i and then recursively outputs the data together with the current bit b on the channel out. Then the sender receives the acknowledgment bit on input channel inp: if it matches b, the sender flips b and returns to waiting for data, otherwise (if the bit did not match or an error occurred) the sender attempts to send the data and b until it receives an acknowledgment with flag b.

```
Sender(i,inp,out,b) <= i(data).SenderSend<i,inp,out,data,b>;

SenderSend(i,inp,out,data,b) <= 'out<data, b>. inp(ackBit).
    case "b = ackBit"   : Sender<i,inp,out,"~b">
      [] "b = ~ackBit"  : SenderSend<i,inp,out,data,b>
      [] "ERR = ackBit" : SenderSend<i,inp,out,data,b> ;
```

The receiver works in a dual fashion.

```
Receiver(o, inp, out, b) <= inp(data, bit).
    case "b = bit"   : 'o<data>.'out<b>.Receiver<o,inp,out,"~b">
      [] "b = ~bit"  : 'out<"~bit"> . Receiver<o,inp,out,b>
      [] "ERR = bit" : 'out<"~b"> . Receiver<o,inp,out,b> ;
```

An error might occur at any time on each of the channels. This kind of unreliable process is modelled implicitly by treating names (representing bits) as variables. Since transmitted names are variables the constraint solver may enable any **case** clause in either Sender or Receiver by finding a suitable term to substitute them for.

Hiding the internal channels, the ABP system can be described as follows:

```
ABP(i,o,sb,rb) <= (new RcSn, SnRc) (
    Sender<i,RcSn,SnRc,sb> | Receiver<o,SnRc,RcSn,rb>);
```

4.5. A Sample Weak Transition

When studying the ABP, it is interesting to see when the protocol communicates with the outside system, ignoring τ-transitions. We here show such a "weak" transition, where the sender receives data and transmits it to the receiver via the data channel. We use the wsstep command on ABP<i,o,sb,rb> to obtain the following transition, among others.

```
1  ==|gen2(data1)|==>
2    Source:
3      ABP<i, o, sb, rb>
4    Constraint:
5      (new RcSn, SnRc){| ”i <–> gen2” |} ∧
6      (new RcSn, SnRc){| ”SnRc <–> SnRc” |} ∧
7      (new RcSn, SnRc){| ”RcSn <–> RcSn” |} ∧
8      (new RcSn, SnRc){| ”rb = ˜sb” |}
9    Solution:
10     ([rb := ”˜sb”, gen2 := i], 1)
11   Derivative:
12     (new SnRc, RcSn)(
13       (case
14         False : Sender<i, RcSn, SnRc, ”˜sb”> []
15         True  : SenderSend<i, RcSn, SnRc, data1, sb> []
16         False : SenderSend<i, RcSn, SnRc, data1, sb>
17       ) |
18         (Receiver<o, SnRc, RcSn, rb>)
19     )
```

After the transition, the sender (lines 13-16) is in a state where it has received an acknowledgment bit which does not match its own bit (constraint on line 8) reducing the condition ”b = ˜ackBit” (at this state it is ”sb = ˜rb”) of SenderSend to true (on line 15).

This transition is among the seven transitions produced by PWB. Since there is always a possibility that both sender and receiver will detect an error ERR, there are infinitely many weak transitions following a cycle between them. The occurrence of such cycles are detected (modulo alpha-equivalence) by the wsstep command. Since the terms occurring in agents are in normal form, wsstep terminates on ABP.

We have shown the development of a tailor-made psi-calculus instance in PWB. (The full code listing is available online [Gutkovas and Borgström 2013].) Doing so, we have expressed bits and bit operations directly, and we have shown that it is possible and useful to use computation in the substitution functions, which departs from traditional calculi. We have also shown the symbolic simulation of a weak transition, which is useful for applications.

5. DATA COLLECTION IN A WIRELESS SENSOR NETWORK

In this example we study a data collection protocol for wireless sensor networks (WSNs) by modelling it in a custom psi-calculus that we implement in PWB.

A wireless sensor network consists of numerous sensor nodes that sense environmental data. A special node, called the sink, is used to collect data from the network. Collection often uses multi-hop communication, building a routing tree rooted at the sink [Madden et al. 2002]. As wireless communication is unreliable, different trees may be built in each protocol run.

We present a simple algorithm to build a routing tree: the sink starts the tree building by broadcasting a special init message containing its identifier $Sink$. When a node n first receives an init message, it sets its parent $parent_n$ to the sender of the message, and broadcasts a new init message containing its own identifier to continue building the next level of the tree. After the building of a tree is complete, each node sends a data message containing its data to its parent. Moreover, each node forwards received data messages to its parent, ensuring that it eventually reaches the sink.

5.1. Psi-calculus instance for WSN data collection

We first define and implement a custom Psi-calculus instance suitable for modelling the tree building and data collection protocol described above. We use structured chan-

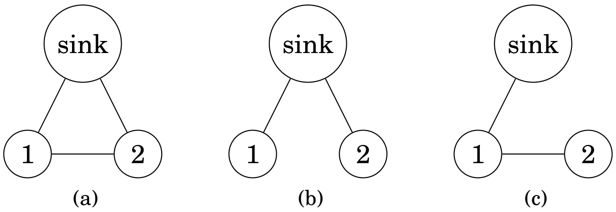


Fig. 3: A simple topology with a sink and two sensor nodes where (a) shows the connectivity and (b)-(c) show some possible routing trees.

nels, of two kinds: broadcast channels $\mathsf{init}(M)$ and unicast channels $\mathsf{data}(M)$. The broadcast connectivity between nodes is given by an undirected topology graph. We first assume a static topology top; the topology in Figure 3(a) would be represented by $\mathsf{top} = \{(0,1),(0,2),(1,2)\}$ where the sink has id 0. The corresponding psi-calculi parameters are defined as follows.

Notation	*SML*	PWB
	datatype term	
$\mathbf{T} \triangleq \{\mathsf{init}(M), \mathsf{data}(N)$	= Init **of** term | Data **of** term	$\mathrm{M} ::= \mathrm{init(M)} \mid \mathrm{data(M)}$
$: M, N \in \mathbf{T}\} \cup \mathcal{N} \cup \mathbb{N}$	| Name **of** name | Int **of** **int**	$\mid \; \mathit{Name} \mid N$
$\mathbf{C} \triangleq \{M \dot{\prec} N, M \dot{\succ} N,$	**datatype** condition	$\varphi ::= \mathrm{M} < \mathrm{M} \mid \mathrm{M} > \mathrm{M}$
$M \dot{\leftrightarrow} N : M, N \in \mathbf{T}\}$	= OutputConn **of** term∗term	$\mid \; \mathrm{M} <\!-\!> \mathrm{M}$
	| InputConn **of** term∗term	
$\mathbf{A} \triangleq \{\mathsf{top}\}$	| ChEq **of** term∗term	$\Psi ::= 1$
$\mathbf{1} \triangleq \mathsf{top}$	**datatype** assertion = Unit	$\mathrm{N} ::= [0-9]^{+}$
	val unit = Unit	

Since we consider a static topology, we implement assertions as a unit type. A broadcast output prefix with subject $\mathsf{init}(i)$ can broadcast on the broadcast channel $\mathsf{init}(i)$, while an input prefix with the same subject can receive from any connected broadcast channel as given by the topology. Two unicast prefixes may communicate iff their subjects are the same name. Thus, we define $\vdash$ as follows.

$$\begin{array}{ll}
\Psi \vdash \mathsf{init}(M) \dot{\prec} \mathsf{init}(N) & \text{iff } M = N \in \mathbb{N} \\
\Psi \vdash \mathsf{init}(M) \dot{\succ} \mathsf{init}(N) & \text{iff } M, N \in \mathbb{N} \text{ and either } (M,N) \in \Psi \text{ or } (N,M) \in \Psi \\
\Psi \vdash \mathsf{data}(a) \dot{\leftrightarrow} \mathsf{data}(b) & \text{iff } a = b \in \mathcal{N}
\end{array}$$

5.2. Constraint Solver for Symbolic Transitions

We describe the implementation of the transition constraint solver. We write $\varnothing$ for no solution. Transition constraints are conjunctions of conditions. The constraints are solved in two phases, corresponding to the unicast connectivity constraints and the broadcast connectivity constraints, respectively. To simplify the solver, we treat all free names in the processes as distinct (cf. distinctions [Milner et al. 1992b]). For unicast constraints, the solver thus fails (returning $\varnothing$) if the constraint is not satisfied.

$$\begin{array}{lll}
(\nu\tilde{a})\{\!|\mathsf{data}(a) \dot{\leftrightarrow} \mathsf{data}(b)|\!\} \wedge C & \rightarrow C & \text{if } a = b \\
(\nu\tilde{a})\{\!|\mathsf{data}(a) \dot{\leftrightarrow} \mathsf{data}(b)|\!\} \wedge C & \rightarrow \varnothing & \text{otherwise}
\end{array}$$

The constraint solver then checks for broadcast connectivity in the given topology. Let O be the output constraints $\{\!|\mathsf{init}(n) \dot{\prec} a|\!\}$ and I the input constraints $\{\!|a \dot{\succ} \mathsf{init}(n)|\!\}$. We distinguish four different cases:

(1) if $I = \emptyset$ and $O = \{\{\!|\mathsf{init}(n) \dot{\prec} a|\!\}\}$, then the solution is $[a := \mathsf{init}(n)]$.

(2) if $I \neq \emptyset$ and $O = \{\{\!|\mathsf{init}(n) \dot{\prec} a|\!\}\}$, and we have $(n,m) \in \mathsf{top}$ for every constraint $\{\!|a \dot{\succ} \mathsf{init}(m)|\!\}$ in I, then the solution is $[a := \mathsf{init}(n)]$. Otherwise the constraint is unsatisfiable, i.e. $\varnothing$.
(3) if $I \neq \emptyset$ and $O = \emptyset$, then the constraint solver finds n such that for every $\{\!|a \dot{\succ} \mathsf{init}(m)|\!\} \in I$ we have $(n,m) \in \mathsf{top}$. For each such n, $[a := \mathsf{init}(n)]$ is a possible solution.
(4) if $I = \emptyset$ and $O = \emptyset$, then the broadcast part of the constraint is trivially true.

5.3. Tree building model

Once the instance is implemented, we can define processes modelling the tree building algorithm in PWB syntax. The sink broadcasts its own channel and then goes into data collection mode, that is, it listens on its unicast channel repeatedly.

```
Sink(nodeId, bsChan) <=
    '"init(nodeId)"!<bsChan> .
    ! "data(bsChan)"(x) ;
```

A node listens on its broadcast channel for a channel of a parent to which it will send data to. Then, similarly to the sink, it broadcasts its own unicast channel on which it expects data to receive in order to forward it to the parent. After completing the broadcast, it sends its data to the parent and goes into mode of forwarding data.

```
Node(nodeId, nodeChan, datum) <=
    "init(nodeId)"?(pChan) .
    '"init(nodeId)"!<nodeChan> .
    '"data(pChan)"<datum> .
    NodeForwardData<nodeChan, pChan> ;

NodeForwardData(nodeChan, pChan) <=
    ! "data(nodeChan)"(x). '"data(pChan)"<x>   ;
```

5.4. Example Strong Transitions

We here study the (symbolic) transition system generated by a small WSN with a sink and two sensor nodes. Each node has a unique channel for response messages.

```
System3(d1,d2) <=
    (new chanS) Sink<0,chanS>        |
    (new chan1)  Node<1, chan1, d1>  |
    (new chan2)  Node<2, chan2, d2>
```

We will show a possible transition sequence in PWB, using the topology shown in Figure 3a. Below, we only consider transitions labelled with broadcast output and unicast communication actions.

The following initial transition is obtained by executing the symbolic simulator of PWB on System3<d1,d2>. The resulting system is in configuration where both sensor nodes have obtained the parent's channel, in this case the sink's. The nodes would then be able to communicate their data to the sink. The unicast channel connectivity corresponds to the routing tree shown in Figure 3b. It is one of seven possible initial transitions produced by PWB, of which three represent broadcast reception from the environment, and the other three situations where not all nodes receive the broadcast message. The transition label gna!(new bsChan)bsChan, represents the channel with a fresh name gna. The generated constraint requires $\{\!|\mathsf{init}(0) \dot{\prec} \mathsf{gna}|\!\} \wedge \{\!|\mathsf{gna} \dot{\succ} \mathsf{init}(1)|\!\} \wedge \{\!|\mathsf{gna} \dot{\succ} \mathsf{init}(2)|\!\}$, meaning node 0 is output connected to some channel gna which is input connected to nodes 1 and 2. The constraint solver finds a solution to the constraint, which substitutes init(0) for gna.

```
−−|gna!(new bsChan)bsChan|−−>
Source:
  System3<d1, d2>
Constraint:
  (new chan1, chan2, chanS){| "init(0)<gna" |} ∧
  (new chanS, chan2, chan1){| "gna>init(1)" |} ∧
  (new chanS, chan1, chan2){| "gna>init(2)" |}
Solution:
  ([gna := "init(0)"], 1)
Derivative:
  (!("data(chanS)"(x))) |
    (((new chan1)(
      '"init(1)"!<chan1>.
        '"data(chanS)"<d1>.
          NodeForwardData<chan1, chanS>
    )) |
      ((new chan2)(
        '"init(2)"!<chan2>.
          '"data(chanS)"<d2>.
            NodeForwardData<chan2, chanS>
      )))
```

In the derivative the Sink successfully communicated its unicast channel chanS to both nodes.

From this point the system can evolve in two symmetrical ways: either of the nodes broadcasts an init message, but since no node in the (closed) system is listening on a broadcast channel, the message is not received. The following transition is for node 1.

```
−−|gna!(new chan1)chan1|−−>
Source:
   The same as the above derivative
Constraint:
  (new chan2, chan1){| "init(1)<gna" |}
Solution:
  ([gna := "init(1)"], 1)
Derivative:
  (!("data(chanS)"(x))) |
    (('"data(chanS)"<d1>.
      NodeForwardData<chan1, chanS>) |
     ((new chan2)(
       '"init(2)"!<chan2>.
         '"data(chanS)"<d2>.
           NodeForwardData<chan2, chanS>
      )))
```

The system is now in the state where node 1 can send data to the sink. By following the analogous transition for node 2, we get the system where both nodes are ready to communicate the data.

```
−−|gna!(new chan2)chan2|−−>
Source:
   The same as the above derivative
Constraint:
  (new chan2){| "init(2)<gna" |}
Solution:
  ([gna := "init(2)"], 1)
Derivative:
  (!("data(chanS)"(x))) |
(('"data(chanS)"<d1>.
  NodeForwardData<chan1, chanS>) |
 ('"data(chanS)"<d2>.
   NodeForwardData<chan2, chanS>))
```

We have demonstrated the use of advanced features in PWB such as the use of structured channels with different modes of communication (point-to point vs broadcast). The broadcast connectivity graph (topology) was formalised as an assertion; this allows us to potentially extend the model, for instance to dynamic or localised connectivity. We used the symbolic execution to simulate strong symbolic transitions of the system. All of this shows the versatility and utility of PWB for use in modelling and studying WSN algorithms.

6. DYNAMIC TOPOLOGY IN WIRELESS SENSOR NETWORK

We here extend the example of Section 5 with dynamic topology. We first allow adding edges to the connectivity graph, and then add the dual operation of removing edges.

Let the parameters be as in the example in Section 5 except for the assertions, which is now a finite set of tuples representing edges in a topology.

Notation	*SML*	PWB
$\mathbf{A} \triangleq \mathcal{P}_{\text{fin}}(\mathbf{T} \times \mathbf{T})$	**datatype** assertion	$\Psi ::= \epsilon$
	= Top **of** (term∗term)**list**	$\mid$ $(\mathrm{M},\mathrm{N})(,(\mathrm{M},\mathrm{N}))^*$
$\mathbf{1} \triangleq \emptyset$	**val** unit = Top []	

The entailment relation is left unchanged, and the constraint solver for the unicast constraints is the same. To enable broadcast connectivity, if the necessary edge is not present, the solver simply attempts to add it to the solution (as is common in process calculi models for WSNs [Ghassemi et al. 2008; Godskesen 2010]). For example, the solution of the constraint of the first transition in Section 5.4 with an empty topology is ([gna := "init(0)"], " (0,2),(0,1) ").

In the following we add the ability for agents to also remove edges from the environment. In the assertions we model edges as binary toggles, so if the same edge occurs twice this is equivalent to it not appearing at all ($i.e.$, $\{(M,N)\} \otimes \{(M,N)\} \simeq \mathbf{1}$). The parameters are extended by adding conditions corresponding to whether an edge is present or not, and the assertions are finite multisets.

Notation	*SML*	PWB
$\mathbf{C} \triangleq \cdots \cup \{\mathsf{conn}(M,N),$	**datatype** condition = ...	$\varphi ::= \ldots \mid \mathrm{conn(M,N)}$
$\mathsf{disconn}(M,N)$	| Conn **of** term∗term	$\mid$ disconn(M,N)
$: M,N \in \mathbf{T}\}$	| Disconn **of** term∗term	
$\mathbf{A} \triangleq \mathbf{T} \times \mathbf{T} \rightarrow_{\text{fin}} \mathbb{N}$	**datatype** assertion	$\Psi ::= \epsilon$
	= Top **of** (term∗term)**list**	$\mid$ $(\mathrm{M},\mathrm{N})(,(\mathrm{M},\mathrm{N}))^*$
$\mathbf{1} \triangleq \emptyset$	**val** unit = Top []	

An odd number of edge tuples in the environment denote that the edge is present; an even number denotes absence. Thus adding a tuple to the environment might add or remove an edge. We capture this with the following entailment definition

$$\begin{array}{ll} \Psi \vdash \mathsf{conn}(M,N) & \text{iff } M,N \in \mathbb{N} \text{ and } |\Psi(M,N)| + |\Psi(N,M)| \text{ is odd} \\ \Psi \vdash \mathsf{disconn}(M,N) & \text{iff } M,N \in \mathbb{N} \text{ and } \Psi \nvdash \mathsf{conn}(M,N) \\ \Psi \vdash \mathsf{init}(M) \dot{\succ} \mathsf{init}(N) & \text{iff } \mathsf{conn}(M,N) \end{array}$$

For the protocol in Section 5 we may reuse the same constraint solver, keeping in mind that it does not handle the case where a disconn condition guards a broadcast input. We can also express the alteration of the topology with the following two agents:

```
Connect(a,b) <=
case "conn(a,b)"    :*tau*.0
  [] "disconn(a,b)":*tau*.(|"(a,b)"|) ;
```

```
Disconnect(a,b) <=
case "conn(a,b)"    :*tau*.(|"(a,b)"|)
  [] "disconn(a,b)":*tau*.0 ;
```

The agent Disconnect<1, 2> | (|"(1,2)"|) has two transitions: first (|"(1,2)" |)|(| "(1,2)"|) with trivially solvable constraint {|"(1,2)" |– "conn(1,2)"|}, and second 0 | (|"(1,2)"|) with the solution ([], "(1,2)"). In both transitions, the environment was extended with an extra tuple $(1, 2)$, effectively removing an edge from the topology. Intuitively, the agents Connect and Disconnect allow to set and unset bits in a global table.

7. SYMBOLIC SEMANTICS

In this section we describe a symbolic operational semantics for broadcast psi-calculi, that is sound (Theorem 7.11) and complete (Theorem 7.12) with respect to the concrete broadcast semantics [Borgström et al. 2011; Borgström et al. 2013]. This semantics is the one that is implemented in the PWB, and it extends, simplifies, and corrects the original symbolic semantics [Johansson et al. 2012].

7.1. Symbolic Operational Semantics

As we have seen, transitions in the symbolic operational semantics are of the form $P \xrightarrow[C]{\alpha} Q$, where C is a constraint that needs to be satisfied for the transition to be enabled. Each PWB instance implements a solver, that computes solutions for the transition constraints of that instance.

Definition 7.1 (*Constraints and Solutions*). A *solution* is a pair (σ, Ψ) where σ is a substitution sequence of terms for names, and Ψ is an assertion. The *transition constraints*, ranged over by C, C_t, and their corresponding solutions $\mathrm{sol}(C)$ are defined by:

$$
\begin{array}{rcl}
 & \text{Constraint} & \text{Solutions} \\
C, C' ::= & \mathbf{true} & \{(\sigma, \Psi) \ : \ \sigma \text{ is a subst. sequence } \wedge \ \Psi \in \mathbf{A}\} \\
| & \mathbf{false} & \emptyset \\
| & (\nu a)C & \{(\sigma, \Psi) \ : \ b \# \sigma, \Psi, C \wedge (\sigma, \Psi) \in \mathrm{sol}((a\ b) \cdot C)\} \\
| & \{\!|\Psi' \vdash \varphi|\!\} & \{(\sigma, \Psi) \ : \ \Psi'\sigma \otimes \Psi \vdash \varphi\sigma\} \\
| & \exists x.C & \{(\sigma, \Psi) \ : \ y \# \sigma, \Psi, C \wedge ([y := M]\sigma, \Psi) \in \mathrm{sol}((x\ y) \cdot C)\} \\
| & a \in \mathsf{n}(M) & \{(\sigma, \Psi) \ : \ a \in \mathsf{n}(M\sigma)\} \\
| & C \wedge C' & \mathrm{sol}(C) \cap \mathrm{sol}(C')
\end{array}
$$

Above, $(a\ b) \cdot C$ stands for the simultaneous replacement of a for b and b for a in C ("swapping"). In $(\nu a)C$, a is binding into C; and in $\exists x.C$, x is binding into C. We write $\exists^b x.C$ for $(\nu b)\exists x.(b \in \mathsf{n}(x) \wedge C)$; the only uses of $\exists$ and $\cdot \in \mathsf{n}(\cdot)$ will be in this restricted form (which is itself only used in rule SBRCLOSE in Table I). We adopt the notation $(\sigma, \Psi) \models C$ to say that $(\sigma, \Psi) \in \mathrm{sol}(C)$, and write $C \leftrightarrow D$ to say that $\mathrm{sol}(C) = \mathrm{sol}(D)$.

A transition constraint C defines a set of solutions $\mathrm{sol}(C)$, namely those where the formula becomes true by applying the substitution and adding the assertion. For example, the transition constraint $\{\!|\mathbf{1} \vdash x = 3|\!\}$ has solutions $([x := 3], \mathbf{1})$ and $([],\ x = 3)$, where $[]$ is the identity substitution.

Restriction distributes over logical conjunction, and logical conjunction has $\mathbf{true}$ as unit and is associative. We thus consider constraints modulo the equations below.

LEMMA 7.2. $(\nu a)(C_1 \wedge C_2) \leftrightarrow (\nu a)C_1 \wedge (\nu a)C_2$ *and* $C_1 \wedge (C_2 \wedge C_3) \leftrightarrow (C_1 \wedge C_2) \wedge C_3$ *and* $C \wedge \mathbf{true} \leftrightarrow C$.

The concept of *frame of an agent* $\mathcal{F}(P)$ is used in the semantics: intuitively it is the top-level assertions of an agent, including the top-level binders. Frames are of the form

$F ::= \Psi \mid (\nu\, a)\Psi$ where a is bound in $(\nu\, a)\Psi$. The frame of a process denotes its contribution to parallel agents. For example, the frame $\mathcal{F}((\nu a)((\!|\Psi_1|\!) \mid \overline{M}\,\widetilde{N}\,.\,(\!|\Psi_3|\!) \mid (\!|\Psi_2|\!)))$ is $(\nu a)(\Psi_1 \otimes \Psi_2)$. Note that Ψ_3 is not included in the frame, since it occurs under a prefix. In order to define the symbolic operational semantics, we need a way to add the frame of a parallel process to the current transition constraint.

Definition 7.3 (*Adding frames to constraints*). We define $F \otimes C$ as follows:

$$\begin{array}{rcll}
F \otimes (\nu a)C & = & (\nu a)(F \otimes C) & \text{where } a\#F \\
(\nu\widetilde{a})\Psi \otimes \{\!|\Psi' \vdash \varphi|\!\} & = & (\nu\widetilde{a})\{\!|\Psi \otimes \Psi' \vdash \varphi|\!\} & \text{where } \widetilde{a}\#\Psi',\varphi \\
(\nu\widetilde{a})\Psi \otimes \exists x.C & = & (\nu\widetilde{a})\exists x.(\Psi \otimes C) & \text{where } \widetilde{a}\#C \text{ and } x\#\widetilde{a},\Psi \\
F \otimes (C \wedge D) & = & (F \otimes C) \wedge (F \otimes D) & \\
F \otimes C & = & C & \text{otherwise.}
\end{array}$$

For the symbolic semantics to be able to pick out the original channel to be used to send a message, we require partial injectivity of channel connectivity in its left argument: we require that for all names a, the function $x \mapsto (x \dot{\leftrightarrow} a)$ is injective.

A process P is said to be assertion guarded if every occurrence of a $(\!|\Psi|\!)$ in P is a subterm of an input or an output. We require that processes are well-formed: P is well-formed if in every subterm of P of the form **case** $\widetilde{\varphi} : \widetilde{Q}$ every Q_i is assertion guarded, and in every subterm of P of the form $!Q$ we have that Q is assertion guarded.

We let the subject (or channel) of an action α be $\mathrm{subj}(\underline{x}?(\widetilde{y})) = \mathrm{subj}(\underline{x}(\widetilde{y})) = \mathrm{subj}(\overline{x}!\ (\nu\widetilde{a})\widetilde{N}) = \mathrm{subj}(\overline{x}\ (\nu\widetilde{a})\widetilde{N}) = x$ and $\mathrm{subj}(\tau) = \emptyset$. We also define the bound names (i.e., the private names) of a label as bn$(\underline{x}?(\widetilde{y}))$ = bn$(\underline{x}(\widetilde{y}))$ $= \widetilde{y}$ and bn$(\overline{x}!\ (\nu\widetilde{a})\widetilde{N})$ = bn$(\overline{x}\ (\nu\widetilde{a})\widetilde{N}) = \widetilde{a}$ and bn$(\tau) = \emptyset$.

The structured symbolic operational semantics preserves well-formedness, and is defined in Tables I, II and III. We first describe the broadcast rules in Table I. First consider the sBrOut rule: $\overline{M}\,\widetilde{N}.P \xrightarrow[\{\!|\mathbf{1} \vdash M \dot{\prec} y|\!\}]{\overline{y}!\,\widetilde{N}} P$. The solutions to its transition constraint are those that enable the subject M of the output prefix to broadcast on the fresh channel variable y. Similarly, in sBrIn we can receive a broadcast from any channel x that the subject M of the input prefix can listen to. In sBrMerge, two inputs with the same labels are merged into one. In sBrCom, a broadcast of P is received by Q, substituting the message $\widetilde{N}$ for the input variables $\widetilde{y}$. The names $\widetilde{a}$ are restricted in P, so they must be fresh for Q. In both sBrMerge and sBrCom, each transition constraint is extended with the frame of the other process. In sBrOpen, the scope of the new name b that occurs in the message $\widetilde{N}$ is opened; we remember in the transition constraint that b is fresh. In sBrClose, a broadcast that has reached its lexical scope turns into an internal τ action. The scoping of the new names $\widetilde{a}$ is reestablished.

The other symbolic rules in Tables II and III are similar to the broadcast rules, with two exceptions. In the sCase rule in Table III we add the constraint that φ_i must hold to the transition constraint. In the sCom rule in Table II we partially deconstruct the transition constraints of the input and the output transition, picking out the first conjunct. We then recombine the remainder of the transition constraints, adding the constraint that their channels are equivalent (i.e., $\Psi_1 \otimes \Psi_2 \vdash M_1 \dot{\leftrightarrow} M_2$), yielding C_{com}. Here the partial injectivity of $\dot{\leftrightarrow}$ is used to guarantee that M_1 is the channel that originated the transition.

7.2. Comparison with the Original Symbolic Operational Semantics

The symbolic semantics used in this paper differs from the original semantics [Johansson et al. 2012] in four significant ways:

Table I: Symbolic transition rules for broadcast communication. A symmetric version of $\textsc{sBrCom}$ is elided. In $\textsc{sBrOpen}$ the expression $\nu\tilde{a} \cup \{b\}$ means the sequence $\tilde{a}$ with b inserted anywhere.

$$\textsc{sBrOut}\ \frac{x\#\Psi, M, \widetilde{N}, P}{\overline{M}\,\widetilde{N} \,.\, P \xrightarrow[\mathbf{1}\vdash M \dot{\prec} x]{\overline{x}!\ \widetilde{N}} P} \qquad \textsc{sBrIn}\ \frac{x, \widetilde{y}\#\Psi, M, P \quad x\#\widetilde{y}}{\underline{M}(\widetilde{y}) \,.\, P \xrightarrow[\mathbf{1}\vdash x \dot{\succ} M]{\underline{x}?(\widetilde{y})} P}$$

$$\textsc{sBrMerge}\ \frac{P \xrightarrow[C_1]{\underline{x}?(\widetilde{y})} P' \qquad Q \xrightarrow[C_2]{\underline{x}?(\widetilde{y})} Q'}{P \mid Q \xrightarrow[(\mathcal{F}(Q)\otimes C_1)\wedge(\mathcal{F}(P)\otimes C_2)]{\underline{x}?(\widetilde{y})} P' \mid Q'}$$

$$\textsc{sBrCom}\ \frac{P \xrightarrow[C_1]{\overline{x}!\ (\nu\widetilde{a})\widetilde{N}} P' \qquad Q \xrightarrow[C_2]{\underline{x}?(\widetilde{y})} Q'}{P \mid Q \xrightarrow[(\mathcal{F}(Q)\otimes C_1)\wedge(\mathcal{F}(P)\otimes C_2)]{\overline{x}!\ (\nu\widetilde{a})\widetilde{N}} P' \mid Q'[\widetilde{y} := \widetilde{N}]}\ \begin{array}{l} |\widetilde{y}| = |\widetilde{N}| \\ \widetilde{a}\#Q \end{array}$$

$$\textsc{sBrOpen}\ \frac{P \xrightarrow[C]{\overline{y}!\ (\nu\widetilde{a})\widetilde{N}} P'}{(\nu b)P \xrightarrow[(\nu b)C]{\overline{y}!\ (\nu\widetilde{a}\cup\{b\})\widetilde{N}} P'}\ \begin{array}{l} b \in \mathsf{n}(\widetilde{N}) \\ b\#\widetilde{a}, y \end{array} \qquad \textsc{sBrClose}\ \frac{P \xrightarrow[C]{\overline{x}!\ (\nu\widetilde{a})\widetilde{N}} P'}{(\nu b)P \xrightarrow[\exists^b x.C]{\tau} (\nu b)(\nu\widetilde{a})P'}$$

Table II: Revised symbolic transition rules for binary communication. The symmetric version of $\textsc{sCom}$ is elided. In $\textsc{sCom}$, we assume that $\widetilde{c_1}\#y, \widetilde{c_2}, \Psi_2, M_2$ and $\widetilde{c_2}\#z, \Psi_1, M_1$ and let $C_{\text{com}} = ((\nu\widetilde{c_1}\widetilde{c_2})\{\!|\Psi_1 \otimes \Psi_2 \vdash M_1 \dot{\leftrightarrow} M_2|\!\}) \wedge (((\nu\widetilde{c_2})\Psi_2) \otimes C_1) \wedge (((\nu\widetilde{c_1})\Psi_1) \otimes C_2)$. In $\textsc{sOpen}$ the expression $\nu\tilde{a} \cup \{b\}$ means the sequence $\tilde{a}$ with b inserted anywhere.

$$\textsc{sOut}\ \frac{y\#M, \widetilde{N}, P}{\overline{M}\,\widetilde{N} \,.\, P \xrightarrow[\{\!|\mathbf{1}\vdash M \dot{\leftrightarrow} y|\!\}]{\overline{y}\widetilde{N}} P} \qquad \textsc{sIn}\ \frac{y\#M, P, \widetilde{x}}{\underline{M}(\widetilde{x}) \,.\, P \xrightarrow[\{\!|\mathbf{1}\vdash M \dot{\leftrightarrow} y|\!\}]{\underline{y}(\widetilde{x})} P}$$

$$\textsc{sCom}\ \frac{\begin{array}{c} P \xrightarrow[(\nu\widetilde{c_1})\{\!|\Psi_1\vdash M_1 \dot{\leftrightarrow} y|\!\}\wedge C_1]{\overline{y}\ (\nu\widetilde{a})\widetilde{N}} P' \\ Q \xrightarrow[(\nu\widetilde{c_2})\{\!|\Psi_2\vdash M_2 \dot{\leftrightarrow} z|\!\}\wedge C_2]{\underline{z}(\widetilde{x})} Q' \end{array}}{P \mid Q \xrightarrow[C_{\text{com}}]{\tau} (\nu\widetilde{a})(P' \mid Q'[\widetilde{x} := \widetilde{N}])}\ \begin{array}{l} |\widetilde{x}| = |\widetilde{N}| \\ \widetilde{a}\#Q \end{array} \qquad \textsc{sOpen}\ \frac{P \xrightarrow[C]{\overline{y}\ (\nu\widetilde{a})\widetilde{N}} P'}{(\nu b)P \xrightarrow[(\nu b)C]{\overline{y}\ (\nu\widetilde{a}\cup\{b\})\widetilde{N}} P'}\ \begin{array}{l} b \in \mathsf{n}(\widetilde{N}) \\ b\#\widetilde{a}, y \end{array}$$

(1) support for broadcast communication (Table I);
(2) support for polyadic communication (sending of multiple message terms at once);
(3) no dependence on an external assertion environment ($\Psi \rhd$ below); and
(4) a new $\textsc{sCom}$ rule, for reasons explained below.

The original version of the communication rule was as follows (omitting its freshness side conditions). Below, the assertion environment "$\ldots\Psi\ \rhd$ " collects the assertions of

Table III: Revised symbolic transition rules common to broadcast and binary communication. A symmetric version of sPAR is elided.

$$\textsc{sCase}\ \frac{P_i \xrightarrow[C]{\alpha} P'}{\textbf{case}\ \widetilde{\varphi} : \widetilde{P} \xrightarrow[C \wedge \{\!|\mathbf{1} \vdash \varphi_i|\!\}]{\alpha} P'}\ \mathrm{subj}(\alpha)\#\varphi_i \qquad \textsc{sRep}\ \frac{P \mid !P \xrightarrow[C]{\alpha} P'}{!P \xrightarrow[C]{\alpha} P'}$$

$$\textsc{sPar}\ \frac{P \xrightarrow[C]{\alpha} P'}{P \mid Q \xrightarrow[\mathcal{F}(Q) \otimes C]{\alpha} P' \mid Q}\ \begin{array}{l}\mathrm{bn}(\alpha)\#Q \\ \alpha = \tau \vee \mathrm{subj}(\alpha)\#Q\end{array} \qquad \textsc{sScope}\ \frac{P \xrightarrow[C]{\alpha} P'}{(\nu b)P \xrightarrow[(\nu b)C]{\alpha} (\nu b)P'}\ b\#\alpha$$

$$\textsc{sInv}\ \frac{P[\widetilde{x} := \widetilde{M}] \xrightarrow[C]{\alpha} P'}{\mathbf{A}\langle \tilde{M} \rangle \xrightarrow[C]{\alpha} P'}\ \begin{array}{l}\mathbf{A}\langle \tilde{x} \rangle \Leftarrow P \\ |\widetilde{x}| = |\widetilde{M}|\end{array}$$

the context of the current process, and can be ignored.

$$\textsc{Old-sCom}\ \frac{\begin{array}{l}\Psi_Q \otimes \Psi \rhd P \xrightarrow[(\nu\widetilde{b}_P)\{\!|\Psi_1 \vdash M_1 \dot{\leftrightarrow} y|\!\} \wedge C_1]{\overline{y}\,(\nu\widetilde{a})\widetilde{N}} P' \\ \Psi_P \otimes \Psi \rhd Q \xrightarrow[(\nu\widetilde{b}_Q)\{\!|\Psi_2 \vdash M_2 \dot{\leftrightarrow} z|\!\} \wedge C_2]{\underline{z}(\widetilde{x})} Q'\end{array}}{\Psi \rhd P \mid Q \xrightarrow[C_{\text{old}}]{\tau} (\nu\widetilde{a})(P' \mid Q'[\widetilde{x} := \widetilde{N}])}\ \begin{array}{c}|\widetilde{x}| = |\widetilde{N}| \\ \mathcal{F}(P) = (\nu\widetilde{b}_P)\Psi_P \\ \mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q \\ \Psi_1 = \Psi_2 = \Psi \otimes \Psi_P \otimes \Psi_Q\end{array}$$

In order to derive a transition with OLD-SCOM, we need to compute the frames of P and Q, equate the bound names in the frames with the ones appearing in the transition constraints such that $\mathcal{F}(P) = (\nu\widetilde{b}_P)\Psi_P$ and $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$, and then check that $\Psi_1 = \Psi_2 = \Psi \otimes \Psi_P \otimes \Psi_Q$. However, these equalities fail in certain situations where we would expect them to hold.

Example 7.4. This example shows issues related to restrictions under process constructors **case** and replication (!). We use replication as an example; the issue when using **case** is analogous. Consider the process $P = !(\nu\, b)\overline{c}\, b.Q$ in the pi-calculus instance. In the original semantics, the symbolic output transition of P has the constraint $(\nu\, b)\{\!|\mathbf{1} \vdash c \dot{\leftrightarrow} x|\!\}$ since the frame of $(\nu\, b)\overline{c}\, b.Q$ (which is $(\nu\, b)\mathbf{1}$) is used in the derivation. When attempting to derive a communication between P and the process $\underline{c}(x).R$, the side condition $\mathcal{F}(P) = (\nu\widetilde{b}_P)\Psi_P$ of OLD-SCOM is impossible to satisfy: $\mathcal{F}(P) = (\nu\varepsilon)\mathbf{1}$ while the transition constraint of P is $(\nu\, b)\{\!|\mathbf{1} \vdash c \dot{\leftrightarrow} x|\!\}$, and the number of bound names thus differ.

A similar issue, related to the ordering of restrictions in the frame, applies when an inactive parallel process has top-level restrictions.

Example 7.5. Let $P = (\nu\, b)\overline{c}\, b.Q \mid (\nu\, a)\underline{c}(x).R$. In the original semantics, the symbolic output transition of P has the constraint $(\nu\, a)(\nu\, b)\{\!|\mathbf{1} \vdash c \dot{\leftrightarrow} x|\!\}$ but $\mathcal{F}(P) = (\nu\, b)(\nu\, a)\mathbf{1}$ where the order of the bound names is different.

Both these issues could be avoided if the binders of frames were so-called `set+` binders [Huffman and Urban 2010] where order does not matter and redundant binders are ignored. However, such a notion of binders is not available in the version of Nominal Isabelle [Urban and Tasson 2005] that is used for the formalization of psi-calculi [Bengtson and Parrow 2009].

Table IV: General requirements on substitution

$$\begin{array}{rcll} X[x := x] & = & X & \\ x[x := M] & = & M & \\ X[x := M] & = & X & \text{if } x\#X \\ X[x := L][y := M] & = & X[y := M][x := L] & \text{if } x\#y, M \text{ and } y\#L \\ X[\tilde{x} := \tilde{T}] & = & ((\tilde{y}\ \tilde{x}) \cdot X)[\tilde{y} := \tilde{T}] & \text{if } \tilde{y}\#X, \tilde{x} \end{array}$$

Table V: Requirements for specific data types

$$\begin{array}{rcl} \mathrm{n}(M\sigma) & \supseteq & \mathrm{n}(M) \setminus \mathrm{n}(\sigma) \\ \mathrm{n}(M[\tilde{a} := \tilde{L}]) & \supseteq & \mathrm{n}(\tilde{L}) \text{ when } \mathrm{n}(M) \supseteq \tilde{a} \\ (M \dot{\prec} N)\sigma & = & M\sigma \dot{\prec} N\sigma \\ (N \dot{\succ} M)\sigma & = & N\sigma \dot{\succ} M\sigma \\ (M \dot{\leftrightarrow} N)\sigma & = & M\sigma \dot{\leftrightarrow} N\sigma \\ \mathbf{1}\sigma & = & \mathbf{1} \end{array}$$

$$\begin{array}{rcl} \Psi \otimes \mathbf{1} & \simeq_{\mathcal{N}} & \Psi \\ \Psi \otimes \Psi' & \simeq_{\mathcal{N}} & \Psi' \otimes \Psi \\ \Psi_1 \otimes (\Psi_2 \otimes \Psi_3) & \simeq_{\mathcal{N}} & (\Psi_1 \otimes \Psi_2) \otimes \Psi_3 \\ (\Psi \otimes \Psi')\sigma & \simeq_{\mathcal{N}} & \Psi\sigma \otimes \Psi'\sigma \\ \Psi \otimes \Psi_1 & \simeq_{\mathcal{N}} & \Psi \otimes \Psi_2 \text{ when } \Psi_1 \simeq_{\mathcal{N}} \Psi_2 \end{array}$$

Example 7.6. This example show issues related to situations where assertion composition is non-commutative. Let the assertions be tuples $\widetilde{a}$ of names, composed using concatenation $\widetilde{a;b}$. Consider the premises of OLD-SCOM: in the original semantics Ψ_1 will have a prefix $\Psi_Q;\Psi$ and Ψ_2 will have a prefix $\Psi_P;\Psi$. Since concatenation is non-commutative, the side condition $\Psi_1 = \Psi_2 = \Psi \otimes \Psi_P \otimes \Psi_Q$ of OLD-COM cannot hold if Ψ_P and Ψ_Q are non-empty and $\mathrm{n}(\Psi_P) \neq \mathrm{n}(\Psi_Q)$. This makes it impossible for the two processes $(\!|a|\!) \mid c$ and $(\!|b|\!) \mid \overline{c}$ to communicate using OLD-SCOM.

These examples show that the OLD-SCOM rule makes too strong assumptions on the syntactic form of the constraints of the transitions in its premise. The original symbolic semantics still corresponds to the concrete semantics [Bengtson et al. 2011] in certain instances, such as when communicating processes do not contain restrictions and assertion composition satisfies the commutative monoid laws (not only modulo assertion equivalence). In contrast to OLD-SCOM, rule SCOM in Table II does not make any assumptions about the number of bound names nor on the structure of the assertion, and the corresponding broadcast rules SBRCOM and SBRMERGE in Table I do not make any assumptions at all about the form of their constraints.

7.3. Correctness of the Symbolic Operational Semantics

The proofs for the soundness and completeness of the symbolic semantics with respect to the concrete broadcast semantics [Borgström et al. 2011] mainly follow [Johansson et al. 2012]. The main exception is that their counterpart of Lemma 7.10, which describes the shape of transition constraints, does not hold in all cases, as seen in Examples 7.4, 7.5 and 7.6. We here instead prove a weaker result by considering assertions and frames modulo redundant restrictions (cf. Example 7.4), restriction ordering (cf. Example 7.5) and commutative monoid laws for assertion composition (cf. Example 7.6).

As to technical preliminaries, we assume the general properties of substitution in Table IV, and the homomorphism and name preservation laws in Table V. As an example, the standard notion of substitution in (nominal) term algebras satisfies all of these properties. We write $\Psi \simeq_{\mathcal{N}} \Psi'$ iff $\mathrm{n}(\Psi) = \mathrm{n}(\Psi')$ and for all φ it holds that $\Psi \vdash \varphi$ iff $\Psi' \vdash \varphi$. We then assume the equivalences in Table V. As an example, they are satisfied when assertions are finite sets of equations on terms, with standard substitution.

The main difference to the original proofs is the introduction of an auxiliary relation on frames (Definition 7.7) in order to accurately describe the shape of transition constraints (Lemma 7.10) such that they can always be decomposed in the SCOM rule, unlike the case for OLD-SCOM.

Definition 7.7 (*AC-equivalence*). Associative/Commutative equivalence (AC-equivalence) of assertions is the smallest equivalence relation such that

(1) $\mathbf{1} \otimes \Psi \equiv_{\mathrm{AC}} \Psi$; and
(2) $\Psi_1 \otimes \Psi_2 \equiv_{\mathrm{AC}} \Psi_2 \otimes \Psi_1$; and
(3) $\Psi_1 \otimes (\Psi_2 \otimes \Psi_3) \equiv_{\mathrm{AC}} (\Psi_1 \otimes \Psi_2) \otimes \Psi_3$; and
(4) $\Psi_1 \equiv_{\mathrm{AC}} \Psi_2 \implies \Psi \otimes \Psi_1 \equiv_{\mathrm{AC}} \Psi \otimes \Psi_2$.

Frames $(\nu\widetilde{a})\Psi_1$ and $(\nu\widetilde{c})\Psi_2$ are AC-equivalent, written $(\nu\widetilde{a})\Psi_1 \equiv_{\mathrm{AC}} (\nu\widetilde{c})\Psi_2$, if $\Psi_1 \equiv_{\mathrm{AC}} \Psi_2$ and $\{\widetilde{a}\} \cap \mathrm{n}(\Psi_1) = \{\widetilde{c}\} \cap \mathrm{n}(\Psi_2)$.

LEMMA 7.8. *AC-equivalence is an equivalence relation on frames, and whenever $F_1 \equiv_{\mathrm{AC}} F_2$ we also have $\mathrm{n}(F_1) = \mathrm{n}(F_2)$ and $(\nu a)F_1 \equiv_{\mathrm{AC}} (\nu a)F_2$ and $G \otimes F_1 \equiv_{\mathrm{AC}} G \otimes F_2$.*

PROOF. Straightforward from the definitions, using the laws in Table V. □

As an example, guarded processes have frames that are AC-equivalent to the unit frame $\mathbf{1}$.

LEMMA 7.9. *If P is assertion guarded, then $\mathcal{F}(P) \equiv_{\mathrm{AC}} \mathbf{1}$.*

PROOF. By induction on P. □

The following lemma characterises the shape of the constraints of point-to-point input and output transitions. The first conjunct in the constraint is always a channel equivalence constraint (between the object M of the original prefix and the transition object variable y) that must hold under a frame $(\nu\widetilde{c})\Psi$ that is AC-equivalent to that of the original process P. The lemma is used in the proof of Theorem 7.12 to show that the precondition on the shape of the transitions in the SCOM rule always holds.

LEMMA 7.10 (FORM OF CONSTRAINT). *Let $\alpha = \overline{y}\,(\nu\widetilde{a})\widetilde{N}$ or $\alpha = \underline{y}(\widetilde{x})$. If $P \xrightarrow[C]{\alpha} P'$ and $y\#P$ then there exist $\widetilde{c}, \Psi, M$ and D such that $\mathcal{F}(P) \equiv_{\mathrm{AC}} (\nu\widetilde{c})\Psi$ and $y\#\widetilde{c}, \Psi, M, D$ and $C = (\nu\widetilde{c})\{\!|\Psi \vdash M \mathrel{\dot{\leftrightarrow}} y|\!\} \wedge D$.*

PROOF. By induction on the derivation of $P \xrightarrow[C]{\alpha} P'$. A base case is as follows.

SOUT. In this case the transition is derived by

$$\textsc{sOut}\ \frac{}{\overline{K}\widetilde{N}\,.\,P \xrightarrow[\{\!|\mathbf{1}\vdash K \dot{\leftrightarrow} y|\!\}]{\overline{y}\widetilde{N}} P}\ y\#K, \widetilde{N}, P$$

Here $\widetilde{c} = \epsilon$, $\Psi = \mathbf{1}$, $M = K$, and $D = \mathbf{true}$, where $\mathcal{F}(\overline{K}\,\widetilde{N}\,.\,P) = \mathbf{1}$.

The cases that require the use of AC-equivalence are the following.

SCASE. In this case the transition is derived by

$$\textsc{sCase}\ \frac{P_i \xrightarrow[C]{\alpha} P'}{\mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \xrightarrow[C \wedge \{\!|\mathbf{1}\vdash\varphi_i|\!\}]{\alpha} P'}\ \mathrm{bn}(\alpha)\#\varphi_i$$

By induction we get $M, D', \Psi, \widetilde{c}$ such that $C = (\nu\widetilde{c})\{\!|\Psi \vdash M \mathrel{\dot{\leftrightarrow}} y|\!\} \wedge D'$ with $y\#D'$ and $\mathcal{F}(P_i) \equiv_{\mathrm{AC}} (\nu\widetilde{c})\Psi$. Let $D = D' \wedge \{\!|\mathbf{1} \vdash \varphi_i|\!\}$; since $y\#\mathbf{case}\ \widetilde{\varphi} : \widetilde{P}$ we also have that $y\#D$. By well-formedness, P_i is guarded, so by Lemma 7.9 $\mathcal{F}(P_i) \equiv_{\mathrm{AC}} \mathbf{1}$. By transitivity $\mathcal{F}(P) = \mathbf{1} \equiv_{\mathrm{AC}} (\nu\widetilde{c})\Psi$.

sPar. In this case the transition is derived by

$$\textsc{sPar}\ \frac{P \xrightarrow[C]{\alpha} P'}{P \mid Q \xrightarrow[\mathcal{F}(Q)\otimes C]{\alpha} P' \mid Q}\ \begin{array}{c}\mathrm{bn}(\alpha)\#Q\\ \alpha = \tau \vee \mathrm{subj}(\alpha)\#Q\end{array}$$

By induction there are $M, D', \Psi, \widetilde{c}$ such that $C = (\nu\widetilde{c})\{\!\!|\Psi \vdash M \dot{\leftrightarrow} y|\!\!\} \wedge D'$ with $y\#D'$ and $\mathcal{F}(P) \equiv_{\mathrm{AC}} (\nu\widetilde{c})\Psi$. Let $D = \mathcal{F}(Q) \otimes D'$; since $y\#P|Q$ we also have that $y\#D$. By Lemma 7.8 $\mathcal{F}(P \mid Q) \equiv_{\mathrm{AC}} ((\nu\widetilde{c})\Psi) \otimes \mathcal{F}(Q) \equiv_{\mathrm{AC}} \mathcal{F}(Q) \otimes (\nu\widetilde{c})\Psi$.

sScope. In this case the transition is derived by

$$\textsc{sScope}\ \frac{P \xrightarrow[C]{\alpha} P'}{(\nu b)P \xrightarrow[(\nu b)C]{\alpha} (\nu b)P'}\ b\#\alpha$$

By induction there exist $\widetilde{c}, \Psi, M$ and D' such that $C = (\nu\widetilde{c})(\Psi \vdash M \dot{\leftrightarrow} y) \wedge D'$ with $y\#M, D$ and $\mathcal{F}(P) \equiv_{\mathrm{AC}} (\nu\widetilde{c})\Psi$. Let $D = (\nu b)D'$; a fortiori $y\#(\nu b)D$. By Lemma 7.8 $\mathcal{F}((\nu b)P) \equiv_{\mathrm{AC}} (\nu b)(\nu\widetilde{c})\Psi$.

sOpen. As sScope.

sRep. In this case the transition is derived by

$$\textsc{sRep}\ \frac{P \mid !P \xrightarrow[C]{\alpha} P'}{!P \xrightarrow[C]{\alpha} P'}$$

By induction there exist $\widetilde{c}, \Psi, M$ and D such that $C = (\nu\widetilde{c})(\Psi \vdash M \dot{\leftrightarrow} y) \wedge D$ with $y\#M, D$ and $\mathcal{F}(P|!P) \equiv_{\mathrm{AC}} (\nu\widetilde{c})\Psi$. By well-formedness, P is guarded, so by Lemma 7.9 $\mathcal{F}(P|!P) \equiv_{\mathrm{AC}} \mathbf{1}$. By transitivity $\mathcal{F}(!P) = \mathbf{1} \equiv_{\mathrm{AC}} (\nu\widetilde{c})\Psi$. □

We prove soundness and completeness of the symbolic semantics of this paper with respect to a polyadic version of the concrete semantics of broadcast psi-calculi [Borgström et al. 2011], which we show in Table VI.

The soundness theorem and its proof follow [Johansson et al. 2012], apart from the weaker preconditions of the sCom rule (compared to Old-sCom), and the new cases for broadcast actions.

Theorem 7.11 (Soundness of symbolic transitions).
If $P \xrightarrow[C]{\alpha} P'$ *and* $(\sigma, \Psi) \models C$ *and* $\mathrm{bn}(\alpha)\#\sigma$ *then* $\Psi \rhd P\sigma \xrightarrow{\alpha\sigma} P'\sigma$.

Proof. By induction on the inference of $P \xrightarrow[C]{\alpha} P'$. □

The proof of the completeness theorem follows [Johansson et al. 2012], apart from new cases for the broadcast rules. In the cCom case of the proof, Lemma 7.10 is used to show that the symbolic transitions obtained from the induction hypothesis are of the right form to apply rule sCom.

Theorem 7.12 (Completeness of symbolic transitions).

—*If* $\Psi \rhd P\sigma \xrightarrow{\tau} P'$ *then* $\exists C, Q\,.\ P \xrightarrow[C]{\tau} Q$, $Q\sigma = P'$ *and* $(\sigma, \Psi) \models C$.

—*If* $\Psi \rhd P\sigma \xrightarrow{\alpha} P'$, $\alpha \neq \tau$, $y\#P, \mathrm{bn}(\alpha), \sigma$, *and* $\mathrm{bn}(\alpha)\#\sigma, P$ *then* $\exists C, \alpha', Q.\ P \xrightarrow[C]{\alpha'} Q$, $Q\sigma = P'$, $\mathrm{subj}(\alpha') = y$, $\alpha'\sigma' = \alpha$, *and* $(\sigma', \Psi) \models C$ *where* $\sigma' = \sigma[y := \mathrm{subj}(\alpha)]$.

Proof. By induction on the inference of $\Psi \rhd P\sigma \xrightarrow{\alpha} P'\sigma$. □

Table VI: Concrete semantics. Symmetric versions of cBrCom, cCom and cPar are elided. In rules cBrCom and cBrMerge and cCom 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 $P, \widetilde{b_Q}, Q$ and Ψ, and that $\widetilde{b_Q}$ is fresh for $Q, \widetilde{b_P}, P$ and Ψ. In the rule cPar we assume that $\mathcal{F}(Q) = (\nu\widetilde{b_Q})\Psi_Q$ where $\widetilde{b_Q}$ is fresh for Ψ, P and α. In cOpen and cBrOpen the expression $\tilde{a} \cup \{b\}$ means the sequence $\tilde{a}$ with b inserted anywhere.

$$\textsc{cBrOut}\ \dfrac{\Psi \vdash M \,\dot{\prec}\, K}{\Psi \,\rhd\, \overline{M}!\widetilde{N}\,.\,P \xrightarrow{\overline{K}!\,\widetilde{N}} P} \qquad \textsc{cBrIn}\ \dfrac{\Psi \vdash K \,\dot{\succ}\, M \qquad |\widetilde{x}| = |\widetilde{N}|}{\Psi \,\rhd\, \underline{M}?\,(\widetilde{x})\,.\,P \xrightarrow{\underline{K}?\,\widetilde{N}} P[\widetilde{x} := \widetilde{N}]}$$

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

$$\textsc{cBrCom}\ \dfrac{\Psi_Q \otimes \Psi \,\rhd\, P \xrightarrow{\overline{K}!\,(\nu\widetilde{a})\widetilde{N}} P' \qquad \Psi_P \otimes \Psi \,\rhd\, Q \xrightarrow{\underline{K}?\,\widetilde{N}} Q'}{\Psi \,\rhd\, P \mid Q \xrightarrow{\overline{K}!\,(\nu\widetilde{a})\widetilde{N}} P' \mid Q'}\ \begin{array}{c}\widetilde{b_P}\widetilde{b_Q}\#K \\ \widetilde{a}\#Q\end{array}$$

$$\textsc{cBrOpen}\ \dfrac{\Psi \,\rhd\, P \xrightarrow{\overline{K}!\,(\nu\widetilde{a})\widetilde{N}} P'}{\Psi \,\rhd\, (\nu b)P \xrightarrow{\overline{K}!\,(\nu\widetilde{a}\cup\{b\})\widetilde{N}} P'}\ \begin{array}{c}b\#\widetilde{a}, \Psi, K \\ b \in \mathrm{n}(N)\end{array} \qquad \textsc{cBrClose}\ \dfrac{\Psi \,\rhd\, P \xrightarrow{\overline{K}!\,(\nu\widetilde{a})\widetilde{N}} P'}{\Psi \,\rhd\, (\nu b)P \xrightarrow{\tau} (\nu b)(\nu\widetilde{a})P'}\ \begin{array}{c}b \in \mathsf{n}(K) \\ b\#\Psi\end{array}$$

$$\textsc{cOut}\ \dfrac{\Psi \vdash M \,\dot{\leftrightarrow}\, K}{\Psi \,\rhd\, \overline{M}\,\widetilde{N}\,.\,P \xrightarrow{\overline{K}\widetilde{N}} P} \qquad \textsc{cIn}\ \dfrac{\Psi \vdash M \,\dot{\leftrightarrow}\, K \qquad |\widetilde{x}| = |\widetilde{N}|}{\Psi \,\rhd\, \underline{M}(\widetilde{x})\,.\,P \xrightarrow{\underline{K}\ \widetilde{N}} P[\widetilde{x} := \widetilde{N}]}$$

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

$$\textsc{cOpen}\ \dfrac{\Psi \,\rhd\, P \xrightarrow{\overline{M}\ (\nu\widetilde{a})N} P'}{\Psi \,\rhd\, (\nu b)P \xrightarrow{\overline{M}\ (\nu\widetilde{a}\cup\{b\})N} P'}\ \begin{array}{c}b\#\widetilde{a}, \Psi, M \\ b \in \mathrm{n}(N)\end{array} \qquad \textsc{cCase}\ \dfrac{\Psi \,\rhd\, P_i \xrightarrow{\alpha} P' \qquad \Psi \vdash \varphi_i}{\Psi \,\rhd\, \textbf{case}\ \widetilde{\varphi} : \widetilde{P} \xrightarrow{\alpha} P'}$$

$$\textsc{cRep}\ \dfrac{\Psi \,\rhd\, P \mid !P \xrightarrow{\alpha} P'}{\Psi \rhd !P \xrightarrow{\alpha} P'} \qquad \textsc{cPar}\ \dfrac{\Psi_Q \otimes \Psi \,\rhd\, P \xrightarrow{\alpha} P'}{\Psi \,\rhd\, P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \mathrm{bn}(\alpha)\#Q$$

$$\textsc{cScope}\ \dfrac{\Psi \,\rhd\, P \xrightarrow{\alpha} P'}{\Psi \,\rhd\, (\nu b)P \xrightarrow{\alpha} (\nu b)P'}\ b\#\alpha, \Psi \qquad \textsc{cInv}\ \dfrac{\Psi \,\rhd\, P[\widetilde{x} := \widetilde{M}] \xrightarrow{\alpha} P'}{\Psi \,\rhd\, \mathrm{A}\langle\tilde{M}\rangle \xrightarrow{\alpha} P'}\ \begin{array}{c}\mathrm{A}\langle\tilde{x}\rangle \Leftarrow P \\ |\widetilde{x}| = |\widetilde{M}|\end{array}$$

8. RELATED WORK

Our previous work [Borgström et al. 2011] presented the broadcast extension of psi-calculi, and a model of a routing protocol for ad-hoc networks. In the present paper we have given a corresponding symbolic semantics, and several new example models.

The precursors of the PWB are the Concurrency Workbench [Cleaveland et al. 1993] for CCS, and the Mobility Workbench [Victor and Moller 1994] for pi-calculus. The tool mCRL2 [Cranen et al. 2013] for ACP allows higher order sorted free algebras and equational logics. PAT3 [Liu et al. 2011] includes a CSP♯ [Sun et al. 2009] module where actions built over types like booleans, integers are extended with C♯ like programs. ProVerif [Blanchet 2011] is a verification tool for the applied pi-calculus [Abadi and Fournet 2001], an extension of the pi-calculus that is specialised for security protocol verification. The tool is parametric in a term language equipped with equations and unidirectional rewrite rules, but works in a fixed logic (predicate logic with equality). ProVerif does not include a symbolic simulator or a general bisimulation checker.

Our symbolic semantics and bisimulation generation algorithm (slight variations of our previous work [Johansson et al. 2012]) are to a large extent based on the pioneering work by Hennessy and Lin [Hennessy and Lin 1995] for value-passing CCS, later specialised for the pi-calculus by Boreale and De Nicola [Boreale and De Nicola 1996] and independently by Lin [Lin 1996; Lin 2000].

9. FUTURE WORK

It would be interesting to investigate other notions of bisimulation for wireless communication [Merro 2007], including machine-checked proofs of their meta-theoretical properties. We have performed initial work [Åman Pohjola et al. 2013] on modelling discrete time, and are considering extensions to other quantitative aspects of wireless networks, including probabilities, distance, and energy.

ACKNOWLEDGMENTS

We thank the anonymous referees of ACSD 2013 for their comments on an earlier version of this paper.

REFERENCES

Martín Abadi and Cédric Fournet. 2001. Mobile Values, New Names, and Secure Communication. In *Proc. of POPL '01*. ACM Press, New York, NY, USA, 104–115. DOI:http://dx.doi.org/10.1145/373243.360213

Martín Abadi and Andrew D. Gordon. 1997. A Calculus for Cryptographic Protocols: The Spi Calculus. In *Fourth ACM Conference on Computer and Communications Security*. ACM Press, 36–47. DOI:http://dx.doi.org/10.1145/266420.266432

Johannes Åman Pohjola, Johannes Borgström, Joachim Parrow, Palle Raabjerg, and Ioana Rodhe. 2013. *Negative Premises in Applied Process Calculi*. Technical Report 2013-014. Dept of Information Technology, Uppsala University.

K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson. 1969. A Note on Reliable Full-Duplex Transmission over Half-Duplex links. *Commun. ACM* 12, 5 (May 1969), 260–261. DOI:http://dx.doi.org/10.1145/362946.362970

Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. 2011. Psi-calculi: A framework for mobile processes with nominal data and logic. *Logical Methods in Computer Science* 7, 1, Article 11 (2011), 44 pages. DOI:http://dx.doi.org/10.2168/LMCS-7(1:11)2011

Jesper Bengtson and Joachim Parrow. 2009. Psi-calculi in Isabelle. In *Proc. of TPHOLs 2009 (LNCS)*. Springer, 99–114. DOI:http://dx.doi.org/10.1007/978-3-642-03359-9_9

Bruno Blanchet. 2011. Using Horn Clauses for Analyzing Security Protocols. In *Formal Models and Techniques for Analyzing Security Protocols*, Véronique Cortier and Steve Kremer (Eds.). Vol. 5. IOS Press, 86–111. DOI:http://dx.doi.org/10.3233/978-1-60750-714-7-86

Michele Boreale and Rocco De Nicola. 1996. A Symbolic Semantics for the π-Calculus. *Information and Computation* 126, 1 (1996), 34–52. DOI:http://dx.doi.org/10.1006/inco.1996.0032

Johannes Borgström, Ramūnas Gutkovas, Ioana Rodhe, and Björn Victor. 2013. A Parametric Tool for Applied Process Calculi. In *Proc. of ACSD'13*. IEEE, Los Alamitos, CA, USA, 187–192. DOI:http://dx.doi.org/10.1109/ACSD.2013.22

Johannes Borgström, Shuqin Huang, Magnus Johansson, Palle Raabjerg, Björn Victor, Johannes Åman Pohjola, and Joachim Parrow. 2011. Broadcast Psi-calculi with an Application to Wireless Protocols. In *Proc. of SEFM '11 (LNCS)*. Springer, 74–89. DOI:http://dx.doi.org/10.1007/978-3-642-24690-6_7

Johannes Borgström, Shuqin Huang, Magnus Johansson, Palle Raabjerg, Björn Victor, Johannes Åman Pohjola, and Joachim Parrow. 2013. Broadcast Psi-calculi with an Application to Wireless Protocols. *Software and Systems Modeling* (2013). In press.

Maria Grazia Buscemi and Ugo Montanari. 2007. CC-Pi: A Constraint-Based Language for Specifying Service Level Agreements. In *Proceedings of ESOP 2007 (LNCS)*, Rocco De Nicola (Ed.), Vol. 4421. Springer, 18–32.

Marco Carbone and Sergio Maffeis. 2003. On the Expressive Power of Polyadic Synchronisation in π-calculus. *Nordic Journal of Computing* 10, 2 (2003), 70–98.

Rance Cleaveland, Joachim Parrow, and Bernhard Steffen. 1993. The Concurrency Workbench: a Semantics-Based Tool for the Verification of Concurrent Systems. *ACM Trans. Program. Lang. Syst.* 15, 1 (1993), 36–72. DOI:http://dx.doi.org/10.1145/151646.151648

Sjoerd Cranen, Jan Friso Groote, Jeroen J A Keiren, Frank P M Stappers, Erik P Vink, Wieger Wesselink, and Tim A C Willemse. 2013. An Overview of the mCRL2 Toolset and Its Recent Advances. In *Proc. of TACAS '13 (LNCS)*, Vol. 7795. Springer, 199–213. DOI:http://dx.doi.org/10.1007/978-3-642-36742-7_15

F. Ghassemi, W. Fokkink, and A. Movaghar. 2008. Restricted Broadcast Process Theory. In *Software Engineering and Formal Methods, 2008. SEFM '08. Sixth IEEE International Conference on*. 345–354. DOI:http://dx.doi.org/10.1109/SEFM.2008.25

Jens Chr. Godskesen. 2010. Observables for Mobile and Wireless Broadcasting Systems. In *Coordination Models and Languages*, Dave Clarke and Gul Agha (Eds.). LNCS, Vol. 6116. Springer, 1–15. DOI:http://dx.doi.org/10.1007/978-3-642-13414-2_1

Ramūnas Gutkovas and Johannes Borgström. 2013. The Psi-Calculi Workbench web page. (2013). http://www.it.uu.se/research/group/mobility/applied/psiworkbench

Matthew Hennessy and Huimin Lin. 1995. Symbolic Bisimulations. *Theoretical Computer Science* 138, 2 (1995), 353–389. DOI:http://dx.doi.org/10.1016/0304-3975(94)00172-F

Brian Huffman and Christian Urban. 2010. A New Foundation for Nominal Isabelle. In *Proc. of ITP'10*. Springer, 35–50. DOI:http://dx.doi.org/10.1007/978-3-642-14052-5_5

Magnus Johansson, Jesper Bengtson, Joachim Parrow, and Björn Victor. 2010. Weak Equivalences in Psi-calculi. In *Proc. of LICS 2010*. IEEE, 322–331. DOI:http://dx.doi.org/10.1109/LICS.2010.30

Magnus Johansson, Björn Victor, and Joachim Parrow. 2012. Computing strong and weak bisimulations for psi-calculi. *Journal of Logic and Algebraic Programming* 81, 3 (2012), 162–180. DOI:http://dx.doi.org/10.1016/j.jlap.2012.01.001

Huimin Lin. 1996. Symbolic Transition Graph with Assignment. In *Proc. of CONCUR '96 (LNCS)*, Vol. 1119. Springer, 50–65. DOI:http://dx.doi.org/10.1007/3-540-61604-7_47

Huimin Lin. 2000. Computing Bisimulations for Finite-Control pi-Calculus. *Journal of Computer Science and Technology* 15, 1 (2000), 1–9. DOI:http://dx.doi.org/10.1007/BF02951922

Yang Liu, Jun Sun, and Jin Song Dong. 2011. PAT 3: An Extensible Architecture for Building Multi-domain Model Checkers. In *Proc. of ISSRE '11*. IEEE, Los Alamitos, CA, USA, 190–199. DOI:http://dx.doi.org/10.1109/ISSRE.2011.19

Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. 2002. TAG: a Tiny AGgregation Service for Ad-hoc Sensor Networks. *SIGOPS Oper. Syst. Rev.* 36, SI (Dec. 2002), 131–146. DOI:http://dx.doi.org/10.1145/844128.844142

Massimo Merro. 2007. An Observational Theory for Mobile Ad Hoc Networks. *Electronical Notes in Theoretical Computer Science* 173 (April 2007), 275–293. DOI:http://dx.doi.org/10.1016/j.entcs.2007.02.039

Robin Milner, Joachim Parrow, and David Walker. 1992a. A Calculus of Mobile Processes, I. *Inf. Comput.* 100, 1 (1992), 1–40. DOI:http://dx.doi.org/10.1016/0890-5401(92)90008-4

Robin Milner, Joachim Parrow, and David Walker. 1992b. A Calculus of Mobile Processes, II. *Inf. Comput.* 100, 1 (1992), 41–77. DOI:http://dx.doi.org/10.1016/0890-5401(92)90009-5

Joachim Parrow, Johannes Borgström, Palle Raabjerg, and Johannes Åman Pohjola. 2013. Higher-order psi-calculi. *Mathematical Structures in Computer Science* FirstView (June 2013), 1–37. DOI:http://dx.doi.org/10.1017/S0960129513000170

Andrew M. Pitts. 2003. Nominal logic, a first order theory of names and binding. *Inf. Comput.* 186, 2 (2003), 165–193. DOI:http://dx.doi.org/10.1016/S0890-5401(03)00138-X

PolyML 2013. Poly/ML. (2013). http://www.polyml.org A full implementation of Standard ML.

Jun Sun, Yang Liu, Jin Song Dong, and Chunqing Chen. 2009. Integrating Specification and Programs for System Modeling and Verification. In *Proc. TASE '09*. IEEE, 127–135. DOI:http://dx.doi.org/10.1109/TASE.2009.32

Christian Urban and Christine Tasson. 2005. Nominal Techniques in Isabelle/HOL.. In *CADE (LNCS)*, Robert Nieuwenhuis (Ed.), Vol. 3632. Springer, 38–53. DOI:http://dx.doi.org/10.1007/11532231_4

Björn Victor and Faron Moller. 1994. The Mobility Workbench — A Tool for the π-Calculus. In *Proc. of CAV '94 (LNCS)*, David Dill (Ed.), Vol. 818. Springer, 428–440. DOI:http://dx.doi.org/10.1007/3-540-58179-0_73

Lucian Wischik and Philippa Gardner. 2005. Explicit fusions. *Theoretical Computer Science* 304, 3 (2005), 606–630. DOI:http://dx.doi.org/10.1016/j.tcs.2005.03.017

Received November 2013; revised ; accepted

Online Appendix to: The Psi-Calculi Workbench: a Generic Tool for Applied Process Calculi

Submitted to Special Issue on Application of Concurrency to System Design

Johannes Borgström, Ramūnas Gutkovas, Ioana Rodhe and Björn Victor, Uppsala University

A. CORRECTNESS PROOFS FOR THE SYMBOLIC SEMANTICS

A.1. Correctness Proofs for the Symbolic Operational Semantics

The proofs for the soundness and completeness of the symbolic semantics with respect to the concrete broadcast semantics [Borgström et al. 2011] mainly follow [Johansson et al. 2012].

We begin by enumerating the additional axioms for substitution.

AXIOM 1. $\mathrm{n}(M\sigma) \supseteq \mathrm{n}(M) \setminus \mathrm{n}(\sigma)$.

AXIOM 2. $\mathrm{n}((\Psi \otimes \Psi')\sigma) = \mathrm{n}(\Psi\sigma \otimes \Psi'\sigma)$.

AXIOM 3. $(\Psi \otimes \Psi')\sigma \simeq \Psi\sigma \otimes \Psi'\sigma$.

AXIOM 4. $\mathbf{1}\sigma = \mathbf{1}$.

We then present a number of lemmas used in the proofs.

LEMMA A.1 (WEAKENING). $(\sigma, \Psi) \models C \Longrightarrow \forall \Psi' : (\sigma, \Psi \otimes \Psi') \models C$

PROOF. By induction over the structure of C.

$\mathbf{true}$ Trivial since all solutions satisfies $\mathbf{true}$.

$\mathbf{false}$ Trivial since false has no solutions.

$\{\!|\Psi'' \vdash \varphi|\!\}$ We have that $(\sigma, \Psi) \models \{\!|\Psi'' \vdash \varphi|\!\}$, so $\Psi''\sigma \otimes \Psi \vdash \varphi\sigma$. Let Ψ' be any assertion. By weakening $\Psi''\sigma \otimes \Psi \otimes \Psi' \vdash \varphi\sigma$, or in other words $(\sigma, \Psi \times \Psi') \models \{\!|\Psi'' \vdash \varphi|\!\}$.

$M = N$ Trivial.

$a\#X$ Trivial.

$a \in \mathsf{n}(M)$ Trivial.

$\exists x.C$ Here there exists $y\#\sigma, \Psi$ such that $(\sigma[y := M], \Psi) \models (x\ y).C$. By induction $(\sigma[y := M], \Psi \otimes \Psi') \models (x\ y).C$. By equivariance of $\vdash$ we may assume that $b\#\Psi'$, so by definition $(\sigma, \Psi \otimes \Psi') \models \exists x.C$.

$(\nu a)C$ We have that $(\sigma, \Psi) \models (\nu a)C$. By Definition 7.1 this means that $\exists b.b\#\sigma, \Psi, C$ such that $(\sigma, \Psi) \models (a\ b)C$. By equivariance of $\vdash$ we may assume that $b\#\Psi'$. By induction $(\sigma, \Psi \otimes \Psi') \models (a\ b)C$, so by definition $(\sigma, \Psi \times \Psi') \models (\nu a)C$.

$C \wedge C'$ By induction $(\sigma, \Psi \otimes \Psi') \models C$ and $(\sigma, \Psi \otimes \Psi') \models C'$, thus $(\sigma, \Psi \otimes \Psi') \models C \wedge C'$.

$C \vee C'$ By induction $(\sigma, \Psi \otimes \Psi') \models C$ or $(\sigma, \Psi \otimes \Psi') \models C'$, thus $(\sigma, \Psi \otimes \Psi') \models C \vee C'$.

$C \Rightarrow C'$ We have that $(\sigma, \Psi) \models C \Rightarrow C'$, i.e. $\forall \Psi''.(\sigma, \Psi \otimes \Psi'') \models C$ implies $(\sigma, \Psi \otimes \Psi'') \models C'$. We must check that $(\sigma, \Psi \otimes \Psi') \models C \Rightarrow C'$, i.e. $\forall \Psi'''.(\sigma, \Psi \otimes \Psi' \otimes \Psi''') \models C$ implies $(\sigma, \Psi \otimes \Psi' \otimes \Psi''') \models C'$, which holds since $\forall \Psi''.(\sigma, \Psi \otimes \Psi'') \models C$ implies $(\sigma, \Psi \otimes \Psi'') \models C'$, and in particular it holds for any $\Psi'' = \Psi' \otimes \Psi'''$ □

LEMMA A.2 (OPENING). *If* $a\#\sigma, \Psi$ *then* $(\sigma, \Psi) \models (\nu a)C$ *iff* $(\sigma, \Psi) \models C$.

DOI : http://dx.doi.org/10.1145/0000000.0000000

PROOF. Immediate from the definition of solutions for $(\nu a)C$. □

LEMMA A.3 (CHANNEL SUBSTITUTION). *When* $(\sigma, \Psi) \models (\nu\widetilde{b})\{\!|\Psi' \vdash M \dot{\leftrightarrow} N|\!\} \wedge C$ *and* $y\#\sigma, \Psi, \widetilde{b}, \Psi', M, N, C$ *and* $\widetilde{b}\#\sigma, \Psi$ *then* $(\sigma \cdot [y := M\sigma], \Psi) \models (\nu\widetilde{b})\{\!|\Psi' \vdash N \dot{\leftrightarrow} y|\!\} \wedge C$.

PROOF. By expanding the definitions involved, using the freshness assumptions and the partial invertibility of $\dot{\leftrightarrow}$. □

AXIOM 5. $\Psi \equiv_{\mathrm{AC}} \Psi' \implies \mathrm{n}(\Psi) = \mathrm{n}(\Psi')$.

LEMMA A.4. $\Psi \equiv_{\mathrm{AC}} \Psi' \implies \Psi\sigma \simeq \Psi'\sigma$ *and* $\mathrm{n}(\Psi\sigma) = \mathrm{n}(\Psi'\sigma)$.

PROOF. By induction on the derivation of $\Psi \equiv_{\mathrm{AC}} \Psi'$, using the symmetry, transitivity and reflexivity of $\simeq$ at the symmetry, transitivity and reflexivity cases, Axiom 3 at the base cases and the induction case, and Axiom 4 at the unit case. □

LEMMA A.5. *If* $\Psi\sigma \simeq \Psi'\sigma$ *and* $\mathrm{n}(\Psi\sigma) = \mathrm{n}(\Psi'\sigma)$*, then* $(\sigma, \Psi) \models C$ *iff* $(\sigma, \Psi') \models C$.

PROOF. Straightforward from the definition of $\models$. □

LEMMA A.6. *If* $\widetilde{c}\#\sigma$ *then*

(1) $\mathcal{F}(P) = (\nu\widetilde{c})\Psi \implies \exists\Psi' \,.\, \mathcal{F}(P\sigma) = (\nu\widetilde{c})\Psi'$ *and* $\Psi' \simeq \Psi\sigma$ *and* $\mathrm{n}(\Psi') = \mathrm{n}(\Psi\sigma)$.
(2) $\mathcal{F}(P\sigma) = (\nu\widetilde{c})\Psi' \implies \exists\Psi \,.\, \mathcal{F}(P) = (\nu\widetilde{c})\Psi$ *and* $\Psi' \simeq \Psi\sigma$ *and* $\mathrm{n}(\Psi') = \mathrm{n}(\Psi\sigma)$.

PROOF. By induction on the derivation of $\mathcal{F}(P)$ (resp $\mathcal{F}(P\sigma)$), using Axiom 3 at the parallel induction case and Axiom 4 at the trivial base cases. □

LEMMA A.7. *If* $\widetilde{a}\#C, \sigma, \Psi'$ *then* $(\sigma, \Psi') \models ((\nu\widetilde{a})\Psi) \otimes C$ *iff* $(\sigma, \Psi' \otimes \Psi\sigma) \models C$.

PROOF. By induction on C. The interesting case is $C = (\nu\widetilde{c})\{\!|\Psi'' \vdash \varphi|\!\}$ where $(\sigma, \Psi') \models ((\nu\widetilde{a})\Psi) \otimes C \iff \Psi' \otimes \Psi\sigma \otimes \Psi''\sigma \vdash \varphi\sigma \iff (\sigma, \Psi' \otimes \Psi\sigma) \models C$ using Axiom 3. □

LEMMA A.8. *If* $F \equiv_{\mathrm{AC}} G$ *and* $(\sigma, \Psi) \models F \otimes C$ *then* $(\sigma, \Psi) \models G \otimes C$.

PROOF. By Lemma A.4 and Lemma A.7. □

The following key lemma characterises the shape of the constraints of point-to-point input and output transitions. The first conjunct in the constraint is always a channel equivalence constraint (between the object M of the original prefix and the transition object variable y) that must hold under a frame $(\nu\widetilde{c})\Psi$ that is AC-equivalent to that of the original process P. The lemma is used in the proof of Theorem 7.11, to show that the frames of the transition constrints correspond to the frames of the originating processes, in the SCOM case. It is also used in the proof of Theorem 7.12, to show that the precondition on the shape of the transitions in the SCOM rule always holds.

LEMMA A.9 (LEMMA 7.10). *Let* $\alpha = \overline{y}\,(\nu\widetilde{a})\widetilde{N}$ *or* $\alpha = \underline{y}(\widetilde{x})$*. If* $P \xrightarrow[C]{\alpha} P'$ *and* $y\#P$ *then there exist* $\widetilde{c}, \Psi, M$ *and* D *such that* $\mathcal{F}(P) \equiv_{\mathrm{AC}} (\nu\widetilde{c})\Psi$ *and* $y\#\widetilde{c}, \Psi, M, D$ *and* $C = (\nu\widetilde{c})\{\!|\Psi \vdash M \dot{\leftrightarrow} y|\!\} \wedge D$.

PROOF. By induction on the derivation of $P \xrightarrow[C]{\alpha} P'$.

Case SIN. In this case the transition is derived like

$$\textsc{sIn}\;\frac{}{\underline{K}(\widetilde{x}) \,.\, P \xrightarrow[\{\!|\mathbf{1} \vdash K \dot{\leftrightarrow} y|\!\}]{\underline{y}(\widetilde{x})} P}\; y\#K, P, \widetilde{x}$$

Here $\widetilde{c} = \epsilon$, $\Psi = \mathbf{1}$, $M = K$, and $D = \mathbf{true}$, where $\mathcal{F}(\underline{K}(\widetilde{x}) \,.\, P) = \mathbf{1}$.

Case sOUT. In this case the transition is derived like

$$\textsc{sOut}\ \dfrac{}{\overline{M}\widetilde{N}\,.\,P \xrightarrow[\{\!|\mathbf{1}\vdash M\leftrightarrow y|\!\}]{\overline{y}\widetilde{N}} P}\ y\#M,\widetilde{N},P$$

Here $\widetilde{c}=\epsilon$, $\Psi=\mathbf{1}$, $M=M$, and $D=\mathbf{true}$, where $\mathcal{F}(\overline{M}\,\widetilde{N}\,.\,P)=\mathbf{1}$.

Case sCASE. In this case the transition is derived like

$$\textsc{sCase}\ \dfrac{P_i \xrightarrow[C]{\alpha} P'}{\mathbf{case}\ \widetilde{\varphi}:\widetilde{P} \xrightarrow[C\wedge\{\!|\mathbf{1}\vdash\varphi_i|\!\}]{\alpha} P'}\ \mathrm{bn}(\alpha)\#\varphi_i$$

By induction we get $M,D',\Psi,\widetilde{c}$ such that $C=(\nu\widetilde{c})\{\!|\Psi\vdash M\leftrightarrow y|\!\}\wedge D'$ with $y\#D'$ and $\mathcal{F}(P_i)\equiv_{\mathrm{AC}}(\nu\widetilde{c})\Psi$. Let $D=D'\wedge\{\!|\mathbf{1}\vdash\varphi_i|\!\}$; since $y\#\mathbf{case}\ \widetilde{\varphi}:\widetilde{P}$ we also have that $y\#D$. By well-formedness, P_i is guarded, so by Lemma 7.9 $\mathcal{F}(P_i)\equiv_{\mathrm{AC}}\mathbf{1}$. By transitivity $\mathcal{F}(P)=\mathbf{1}\equiv_{\mathrm{AC}}(\nu\widetilde{c})\Psi$.
Case sPAR. In this case the transition is derived like

$$\textsc{sPar}\ \dfrac{P \xrightarrow[C]{\alpha} P'}{P\mid Q \xrightarrow[\mathcal{F}(Q)\otimes C]{\alpha} P'\mid Q}\ \begin{array}{c}\mathrm{bn}(\alpha)\#Q\\ \alpha=\tau\vee\mathrm{subj}(\alpha)\#Q\end{array}$$

By induction there are $M,D',\Psi,\widetilde{c}$ such that $C=(\nu\widetilde{c})\{\!|\Psi\vdash M\leftrightarrow y|\!\}\wedge D'$ with $y\#D'$ and $\mathcal{F}(P)\equiv_{\mathrm{AC}}(\nu\widetilde{c})\Psi$. Let $D=\mathcal{F}(Q)\otimes D'$; since $y\#P|Q$ we also have that $y\#D$. By Lemma 7.8 $\mathcal{F}(P\mid Q)\equiv_{\mathrm{AC}}((\nu\widetilde{c})\Psi)\otimes\mathcal{F}(Q)\equiv_{\mathrm{AC}}\mathcal{F}(Q)\otimes(\nu\widetilde{c})\Psi$.
Case sSCOPE. In this case the transition is derived like

$$\textsc{sScope}\ \dfrac{P \xrightarrow[C]{\alpha} P'}{(\nu b)P \xrightarrow[(\nu b)C]{\alpha} (\nu b)P'}\ b\#\alpha$$

By induction there exist $\widetilde{c},\Psi,M$ and D' such that $C=(\nu\widetilde{c})(\Psi\vdash M\leftrightarrow y)\wedge D'$ with $y\#M,D$ and $\mathcal{F}(P)\equiv_{\mathrm{AC}}(\nu\widetilde{c})\Psi$. Let $D=(\nu b)D'$; a fortiori $y\#(\nu b)D$. By Lemma 7.8 $\mathcal{F}((\nu b)P)\equiv_{\mathrm{AC}}(\nu b)(\nu\widetilde{c})\Psi$.
Case sOPEN. As sSCOPE.
Case sREP. In this case the transition is derived like

$$\textsc{sRep}\ \dfrac{P\mid !P \xrightarrow[C]{\alpha} P'}{!P \xrightarrow[C]{\alpha} P'}$$

By induction there exist $\widetilde{c},\Psi,M$ and D such that $C=(\nu\widetilde{c})(\Psi\vdash M\leftrightarrow y)\wedge D$ with $y\#M,D$ and $\mathcal{F}(P|!P)\equiv_{\mathrm{AC}}(\nu\widetilde{c})\Psi$. By well-formedness, P is guarded, so by Lemma 7.9 $\mathcal{F}(P|!P)\equiv_{\mathrm{AC}}\mathbf{1}$. By transitivity $\mathcal{F}(!P)=\mathbf{1}\equiv_{\mathrm{AC}}(\nu\widetilde{c})\Psi$. □

LEMMA A.10 (CHANGE SUBJECT). *Let B be any finite set of names. If $\alpha=\overline{y}\,(\nu\widetilde{a})\widetilde{N}$ (resp. $\alpha=y(\widetilde{x})$)*

and $P \xrightarrow[(\nu\widetilde{b})\{\!|\Psi\vdash M\leftrightarrow y|\!\}\wedge C]{\alpha} P'$ then $\exists z$ such that $z\#\Psi,\widetilde{b},P,B,C$ and $P \xrightarrow[(\nu\widetilde{b})\{\!|\Psi\vdash M\leftrightarrow z|\!\}\wedge C]{\alpha'} P'$

with $\alpha'=\overline{z}\,(\nu\widetilde{a})\widetilde{N}$ (resp. $\alpha'=z(\widetilde{x})$).

PROOF. By induction on the derivation of the transition. The set of names B is necessary to be able to use the induction hypothesis in some of the induction cases. □

LEMMA A.11. *If* $P \xrightarrow[C]{\alpha} P'$ *and* $a\#P, \mathrm{bn}(\alpha)$ *then* $a\#P'$.

PROOF. The proof is by induction on the derivation of the transition.

Case sIN. In this case the transition is derived like

$$\textsc{sIn}\ \frac{}{\underline{M}(\widetilde{x})\,.\,P \xrightarrow[\{\!|\mathbf{1}\vdash M \dot{\leftrightarrow} y|\!\}]{\underline{y}(\widetilde{x})} P}\ y\#\Psi, M, P, \widetilde{x}$$

We know that $a\#\underline{M}(\widetilde{x})\,.\,P, \widetilde{x}$. Then also $a\#P$.
Case sOUT. In this case the transition is derived like

$$\textsc{sOut}\ \frac{}{\overline{M}\widetilde{N}\,.\,P \xrightarrow[\{\!|\mathbf{1}\vdash M \dot{\leftrightarrow} y|\!\}]{\overline{y}\widetilde{N}} P}\ y\#\Psi, M, \widetilde{N}, P$$

We know that $a\#\overline{M}\widetilde{N}\,.\,P$. Then also $a\#P$.
Case sCASE. In this case the transition is derived like

$$\textsc{sCase}\ \frac{P_i \xrightarrow[C]{\alpha} P'}{\mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \xrightarrow[C\wedge\{\!|\mathbf{1}\vdash\varphi_i|\!\}]{\alpha} P'}\ \mathrm{bn}(\alpha)\#\varphi_i$$

We know that $a\#\mathbf{case}\ \widetilde{\varphi} : \widetilde{P}, \mathrm{bn}(\alpha)$. Then also $a\#P_i$. By induction we get that $a\#P'$.
Case sCOM. In this case the transition is derived like

$$\textsc{sCom}\ \frac{P \xrightarrow[C_P]{\overline{y}\,(\nu\widetilde{a})\widetilde{N}} P' \qquad Q \xrightarrow[C_Q]{\underline{z}(\widetilde{x})} Q'}{P \mid Q \xrightarrow[C]{\tau} (\nu\widetilde{a})(P' \mid Q'[\widetilde{x} := \widetilde{N}])}\ \begin{array}{l}\widetilde{a}\#Q,\\ y\#z\end{array}$$

We know that $a\#P \mid Q$. Let $p \subseteq \widetilde{a} \times (p \cdot \widetilde{a})$ be a permutation such that $a\#p \cdot \widetilde{a}$. By α-conversion we write the transition from P as $P \xrightarrow[p\cdot C_P]{\overline{y}\,(\nu p\cdot\widetilde{a})\widetilde{p\cdot\widetilde{N}}} p \cdot P'$. By induction we get that $a\#p \cdot P'$. Let $q \subseteq \{\widetilde{x}\} \times (q \cdot \{\widetilde{x}\})$ be a permutation such that $a, p \cdot \widetilde{a}\#q \cdot \widetilde{x}$. By α-conversion we write the transition from Q as $Q \xrightarrow[q\cdot C_Q]{\underline{z}(\widetilde{q\cdot x})} q \cdot Q'$. By induction we get that $a\#q \cdot Q'$ and that $p \cdot \widetilde{a}\#q \cdot Q'$. Since $a\#P, p \cdot \widetilde{a}$ we also have that $a\#p \cdot \widetilde{N}$. This means that $a\#(q \cdot Q')[q \cdot \widetilde{x} := p \cdot \widetilde{N}]$ by one of the requirements on substitution. All together we get that $a\#(\nu p \cdot \widetilde{a})(p \cdot P' \mid (q \cdot Q')[q \cdot \widetilde{x} := p \cdot \widetilde{N}])$. By the substitution law for α-conversion we get that $a\#(\nu p \cdot \widetilde{a})(p \cdot P' \mid Q'[\widetilde{x} := p \cdot \widetilde{N}])$. Finally, by α-converting we get that $a\#(\nu\widetilde{a})(P' \mid Q'[\widetilde{x} := \widetilde{N}])$.
Case sPAR. In this case the transition is derived like

$$\textsc{sPar}\ \frac{P \xrightarrow[C]{\alpha} P'}{P \mid Q \xrightarrow[\mathcal{F}(Q)\otimes C]{\alpha} P' \mid Q}\ \begin{array}{c}\mathrm{bn}(\alpha)\#Q\\ \alpha = \tau \vee \mathrm{subj}(\alpha)\#Q\end{array}$$

We know that $a\#P \mid Q, \mathrm{bn}(\alpha)$. By induction we get that $a\#P'$. Then also $a\#P' \mid Q$.

Case sSCOPE. In this case the transition is derived like

$$\textsc{sScope}\ \frac{P \xrightarrow[C]{\alpha} P'}{(\nu b)P \xrightarrow[(\nu b)C]{\alpha} (\nu b)P'}\ b\#\alpha,\Psi$$

We know that $a\#(\nu b)P, \mathrm{bn}(\alpha)$. Let $p = (b\ c)$ such that $a\#c, p\cdot P, p\cdot\mathrm{bn}(\alpha)$. By equivariance the premise is rewritten to $p \cdot \left(p \cdot P \xrightarrow[p\cdot C]{p\cdot\alpha} p \cdot P'\right)$. By induction we get that $a\#p\cdot P'$. Then also $a\#(\nu p\cdot b)(p\cdot P')$. By α-equivalence we get that $a\#(\nu b)P'$.
Case sOPEN. In this case the transition is derived like

$$\textsc{sOpen}\ \frac{P \xrightarrow[C]{\overline{y}\ (\nu\widetilde{a})\widetilde{N}} P'}{(\nu b)P \xrightarrow[(\nu b)C]{\overline{y}\ (\nu\widetilde{a}\cup\{b\})\widetilde{N}} P'}\ \begin{array}{l} b\in\mathrm{n}(\widetilde{N}) \\ b\#\widetilde{a},\Psi,y \end{array}$$

We know that $a\#(\nu b)P,\widetilde{a},b$. This gives us that $a\#P$. By induction we get that $a\#P'$.
Case sREP. In this case the transition is derived like

$$\textsc{sRep}\ \frac{P \mid\, !P \xrightarrow[C]{\alpha} P'}{!P \xrightarrow[C]{\alpha} P'}$$

We know that $a\#!P, \mathrm{bn}(\alpha)$. This gives us that $a\#P \mid\, !P$. By induction we get that $a\#P'$.
Case sBROUT. Here the transition is derived by

$$\textsc{sBrOut}\ \frac{x\#M,\widetilde{N},P}{\overline{M}\,\widetilde{N}\,.\,P \xrightarrow[\mathbf{1}\vdash M\dot{\prec}x]{\overline{x}!\ \widetilde{N}} P}$$

We know that $a\#P$.
Case sBRIN. Here the transition is derived by

$$\textsc{sBrIn}\ \frac{x,\widetilde{y}\#\Psi,M,P \quad x\neq\widetilde{y}}{\underline{M}(\widetilde{y})\,.\,P \xrightarrow[\mathbf{1}\vdash x\dot{\succ}M]{\underline{x}?(\widetilde{y})} P}$$

We know that $a\#\underline{M}(\widetilde{y})\,.\,P,\widetilde{y}$. This gives us that $a\#P$.
Case sBRMERGE. Here the transition is derived by

$$\textsc{sBrMerge}\ \frac{P \xrightarrow[C_P]{\underline{x}?(\widetilde{y})} P' \qquad Q \xrightarrow[C_Q]{\underline{x}?(\widetilde{y})} Q'}{P \mid Q \xrightarrow[(\mathcal{F}(Q)\otimes C_P)\wedge(\mathcal{F}(P)\otimes C_Q)]{\underline{x}?(\widetilde{y})} P' \mid Q'}$$

By induction $a\#P',Q'$, so $a\#P' \mid Q'$.
Case sBRCOM. Here the transition is derived by

$$\textsc{sBrCom}\ \frac{P \xrightarrow[C_P]{\overline{x}\ (\nu\widetilde{a})\widetilde{N}} P' \qquad Q \xrightarrow[C_Q]{\underline{x}?(\widetilde{y})} Q'}{P \mid Q \xrightarrow[(\mathcal{F}(Q)\otimes C_P)\wedge(\mathcal{F}(P)\otimes C_Q)]{\overline{x}\ (\nu\widetilde{a})\widetilde{N}} P' \mid Q'[\widetilde{y} := \widetilde{N}]}\ \widetilde{a}\#Q$$

By induction $a\#P',Q'$. Here $\mathrm{n}(\widetilde{N}) \subseteq \mathrm{n}(P)\cup\{\widetilde{a}\}$, so since $a\#P,\widetilde{a}$ we have $a\#\widetilde{N}$. Thus $a\#P' \mid Q'[\widetilde{y} := \widetilde{N}]$.

Case sBrClose. Here the transition is derived by

$$\textsc{sBrClose}\ \frac{P \ \xrightarrow[C]{\overline{x}!\,(\nu\widetilde{a})\widetilde{N}}\ P'}{(\nu b)P \ \xrightarrow[\exists^b x.C]{\tau}\ (\nu b)(\nu\widetilde{a})P'}$$

Assume that $a\#b$. By induction $a\#P'$, so $a\#(\nu b)(\nu\widetilde{a})P'$. □

Lemma A.12.

$$\begin{aligned}\mathcal{F}((\nu a)P) = (\nu\widetilde{b}_{(\nu a)P})\Psi_{(\nu a)P} \implies\ & \exists\widetilde{b}_P, \Psi_P \textit{ such that}\\ & \mathcal{F}(P) = (\nu\widetilde{b}_P)\Psi_P\\ \wedge\ & \widetilde{b}_{(\nu a)P} = a\widetilde{b}_P\\ \wedge\ & \Psi_{(\nu a)P} = \Psi_P\end{aligned}$$

Proof. Just use the definitions involved. □

Lemma A.13.

$$\begin{aligned}\mathcal{F}(P \mid Q) = (\nu\widetilde{b}_{P \mid Q})\Psi_{P \mid Q} \implies\ & \exists\widetilde{b}_P, \widetilde{b}_Q, \Psi_P, \Psi_Q \textit{ such that}\\ & \mathcal{F}(P) = (\nu\widetilde{b}_P)\Psi_P\\ \wedge\ & \mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q\\ \wedge\ & \widetilde{b}_{P \mid Q} = \widetilde{b}_P\widetilde{b}_Q\\ \wedge\ & \Psi_{P \mid Q} = \Psi_P \otimes \Psi_Q\end{aligned}$$

Proof. Just use the definitions involved. □

Lemma A.14 (Change frame).
If $\Psi \rhd P \xrightarrow{\alpha} P'$, $\Psi \simeq \Psi'$, *and* $\mathrm{n}(\Psi) = \mathrm{n}(\Psi')$, *then* $\Psi' \rhd P \xrightarrow{\alpha} P'$.

Proof. By induction on the derivation of the transition. □

Lemma A.15 (Names are fresh in the constraint).
If $P \xrightarrow[C]{\alpha} P'$ *with* $\alpha = \underline{y}(\widetilde{x})$ *or* $\alpha = \underline{y}?(\widetilde{x})$, *and* $\widetilde{x}, z\#P, y$ *then* $\widetilde{x}, z\#C$.

Proof. By induction on the derivation of the transition.

Case sIn. In this case the transition is derived like

$$\textsc{sIn}\ \frac{}{\underline{M}(\widetilde{x})\,.\,P \ \xrightarrow[\{\!|\mathbf{1}\vdash M \dot{\leftrightarrow} y|\!\}]{\alpha}\ P}\ y\#M, P, \widetilde{x}$$

We know that $\widetilde{x}, z\#y, \underline{M}(\widetilde{x})\,.\,P$, so $\widetilde{x}, z\#\{\!|\mathbf{1} \vdash M \dot{\leftrightarrow} y|\!\}$.
Case sCase. In this case the transition is derived like

$$\textsc{sCase}\ \frac{P_i \ \xrightarrow[C]{\alpha}\ P'}{\textbf{case}\ \widetilde{\varphi} : \widetilde{P} \ \xrightarrow[C \wedge \{\!|\mathbf{1}\vdash\varphi_i|\!\}]{\alpha}\ P'}$$

By induction we get that $\widetilde{x}, z\#C$. From $\widetilde{x}, z\#\varphi_i$ we get that $\widetilde{x}, z\#\{\!|\mathbf{1} \vdash \varphi_i|\!\}$.
Case sPar. In this case the transition is derived like

$$\textsc{sPar}\ \frac{P \ \xrightarrow[C]{\alpha}\ P'}{P \mid Q \ \xrightarrow[\mathcal{F}(Q)\otimes C]{\alpha}\ P' \mid Q}\ \begin{array}{l}\widetilde{x}\#Q\\ y\#Q\end{array}$$

By induction $\widetilde{x}, z\#C$, and since $\widetilde{x}, z\#Q$ we also have $\widetilde{x}, z\#\mathcal{F}(Q)$. By equivariance of $\otimes$ we get $\widetilde{x}, z\#\mathcal{F}(Q) \otimes C$.

Case sSCOPE. In this case the transition is derived like

$$\text{sSCOPE}\ \frac{P \xrightarrow[C]{\alpha} P'}{(\nu b)P \xrightarrow[(\nu b)C]{\alpha} (\nu b)P'}\ b\#\alpha,\Psi$$

We may assume that $b\#z$. By induction we get that $\widetilde{x}, z\#C$, so a fortiori $\widetilde{x}, z\#(\nu b)C$.
Case sREP. In this case the transition is derived like

$$\text{sREP}\ \frac{P \mid !P \xrightarrow[C]{\alpha} P'}{!P \xrightarrow[C]{\alpha} P'}$$

The desired result follows directly from induction.
Case sBRIN. Here the transition is derived by

$$\text{sBRIN}\ \frac{\widetilde{x}, y\#\Psi, M, P \quad y\#\widetilde{x}}{\underline{M}(\widetilde{x})\,.\,P \xrightarrow[\mathbf{1}\vdash y \dot{\succ} M]{\underline{y}?(\widetilde{x})} P}$$

We know that $\widetilde{x}, z\#y, \underline{M}(\widetilde{x})\,.\,P$, so $\widetilde{x}, z\#\{\!\mid \mathbf{1} \vdash y \dot{\succ} M \mid\!\}$.
Case sBRMERGE. Here the transition is derived by

$$\text{sBRMERGE}\ \frac{P \xrightarrow[C_P]{\underline{y}?(\widetilde{x})} P' \qquad Q \xrightarrow[C_Q]{\underline{y}?(\widetilde{x})} Q'}{P \mid Q \xrightarrow[(\mathcal{F}(Q)\otimes C_P)\wedge(\mathcal{F}(P)\otimes C_Q)]{\underline{y}?(\widetilde{x})} P' \mid Q'}$$

By induction $\widetilde{x}, z\#C_P, C_Q$. By assumption $\widetilde{x}, z\#P, Q$, so $\widetilde{x}, z\#\mathcal{F}(P), \mathcal{F}(Q)$. By equivariance of $\otimes$ and $\wedge$ we then get $\widetilde{x}, z\#\mathcal{F}(Q) \otimes C_P \wedge \mathcal{F}(P) \otimes C_Q$. □

LEMMA A.16 (CONGRUENCE OF CONSTRAINT EQUIVALENCE).
If $\forall\sigma,\Psi.(\sigma,\Psi) \models C \Leftrightarrow (\sigma,\Psi) \models D$ *then* $\forall\sigma,\Psi.(\sigma,\Psi) \models (\nu a)C \Leftrightarrow (\sigma,\Psi) \models (\nu a)D$.

PROOF. Adding a restriction of a to a constraint amounts to removing the solutions involving a from the set of all solutions. In this case we remove the same solutions from both C and D, so the resulting sets of all substitutions will still be equal. □

LEMMA A.17. $\forall\sigma,\Psi.(\sigma,\Psi) \models (\nu a)(\nu b)C \Leftrightarrow (\sigma,\Psi) \models (\nu b)(\nu a)C$

PROOF. Both $(\nu a)(\nu b)$ and $(\nu b)(\nu a)$ remove the same set of solutions from C. □

A.2. Proof of Soundness Theorem

We prove soundness and completeness of the symbolic semantics of this paper with respect to the concrete semantics of broadcast psi-calculi [Borgström et al. 2011] (Table VI). The soundness theorem and its proof follow [Johansson et al. 2012], apart from the weaker preconditions of the sCOM rule (compared to OLD-COM), and the new cases for broadcast actions.

THEOREM A.18 (THEOREM 7.11).
If $P \xrightarrow[C]{\alpha} P'$ *and* $(\sigma,\Psi) \models C$ *and* $\mathrm{bn}(\alpha)\#\sigma$ *then* $\Psi \rhd P\sigma \xrightarrow{\alpha\sigma} P'\sigma$.

PROOF. By induction on the inference of $P \xrightarrow[C]{\alpha} P'$.

Case sIn. In this case the inference looks like

$$\textsc{sIn}\ \frac{}{\underline{M}(\widetilde{x}).P \xrightarrow[\{\!|\mathbf{1}\vdash M \dot\leftrightarrow y|\!\}]{\underline{y}(\widetilde{x})} P}\ y\#M,P,\widetilde{x}$$

Since $\widetilde{x}\#\sigma$ we have that $(\underline{M}(\widetilde{x}).P)\sigma = \underline{M\sigma}(\widetilde{x}).P\sigma$ and that $(\underline{y}(\widetilde{x}))\sigma = \underline{y\sigma}(\widetilde{x})$. We then do the following derivation:

$$\textsc{cIn}\ \frac{\Psi \vdash M\sigma \dot\leftrightarrow y\sigma}{\Psi \ \rhd\ \underline{M\sigma}(\widetilde{x}).P\sigma \ \xrightarrow{\underline{y\sigma}(\widetilde{x})}\ P\sigma}$$

Case sOut. In this case the inference looks like

$$\textsc{sOut}\ \frac{}{\overline{M}\widetilde{N}.P \xrightarrow[\{\!|\mathbf{1}\vdash M \dot\leftrightarrow y|\!\}]{\overline{y}\widetilde{N}} P}\ y\#M,\widetilde{N},P$$

We then have a concrete transition

$$\textsc{cOut}\ \frac{\Psi \vdash M\sigma \dot\leftrightarrow y\sigma}{\Psi \ \rhd\ (\overline{M}\widetilde{N}.P)\sigma \ \xrightarrow{(\overline{y}\widetilde{N})\sigma}\ P\sigma}$$

Case sCase. In this case the inference looks like

$$\textsc{sCase}\ \frac{P_i \xrightarrow[C]{\alpha} P'}{\textbf{case}\ \widetilde{\varphi} : \widetilde{P} \xrightarrow[C\wedge\{\!|\mathbf{1}\vdash\varphi_i|\!\}]{\alpha} P'}$$

Take (σ,Ψ) such that $(\sigma,\Psi) \models C \wedge \{\!|\mathbf{1} \vdash \varphi_i|\!\}$ and $\mathrm{bn}(\alpha)\#\sigma$. We must find a transition $\Psi \ \rhd\ (\textbf{case}\ \widetilde{\varphi} : \widetilde{P})\sigma \ \xrightarrow{\alpha\sigma}\ P'\sigma$.
We then have that $\Psi \vdash \varphi_i\sigma$ and that (σ,Ψ) is also a solution to C. By induction we get that $\Psi \ \rhd\ P_i\sigma \ \xrightarrow{\alpha\sigma}\ P'\sigma$. We can now do the following derivation:

$$\textsc{cCase}\ \frac{\Psi \ \rhd\ P_i\sigma \ \xrightarrow{\alpha\sigma}\ P'\sigma \qquad \Psi \vdash \varphi_i\sigma}{\Psi \ \rhd\ (\textbf{case}\ \widetilde{\varphi} : \widetilde{P})\sigma \ \xrightarrow{\alpha\sigma}\ P'\sigma}$$

Case sCom. In this case the inference looks like

$$\textsc{sCom}\ \frac{P \xrightarrow[C'_P]{\overline{y}\,(\nu\widetilde{a})\widetilde{N}} P' \qquad Q \xrightarrow[C'_Q]{\underline{z}(\widetilde{x})} Q'}{P \mid Q \xrightarrow[C]{\tau} (\nu\widetilde{a})(P' \mid Q'[\widetilde{x} := \widetilde{N}])}\ \begin{array}{c}\widetilde{a}\#Q\\ y\#z\\ y,z\#\widetilde{b}_P,P,\widetilde{b}_Q,Q,\widetilde{N},\widetilde{a}\end{array}$$

where $C'_P = (\nu\widetilde{c}_P)\{\!|\Psi'_P \vdash M_P \dot\leftrightarrow y|\!\} \wedge C_P$, $C'_Q = (\nu\widetilde{c}_Q)\{\!|\Psi'_Q \vdash M_Q \dot\leftrightarrow z|\!\} \wedge C_Q$ and $C = (\nu\widetilde{c}_P\widetilde{c}_Q)\{\!|\Psi_P \otimes \Psi_Q \vdash M_P \dot\leftrightarrow M_Q|\!\} \wedge ((\nu\widetilde{c}_Q)\Psi'_Q \otimes C_P) \wedge ((\nu\widetilde{c}_Q)\Psi'_P \otimes C_Q)$.
We assume that $y,z\#\sigma,\Psi',C_P,C_Q$. If that is not the case we can use Lemma A.10 to find subjects for which it is true. We further assume that $\widetilde{a},\widetilde{x}\#\sigma$ (bound names are fresh). Let $\mathcal{F}(P) = (\nu\widetilde{b}_P)\Psi_P$ and $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$. We assume that that $\widetilde{b}_P,\widetilde{b}_Q\#(\sigma,\Psi),P,Q,\widetilde{a}$.
By Lemma 7.10 $\mathcal{F}(P) \equiv_{\mathrm{AC}} (\nu\widetilde{c}_P)\Psi'_P$ and $\mathcal{F}(Q) \equiv_{\mathrm{AC}} (\nu\widetilde{c}_Q)\Psi'_Q$.
We know that $(\sigma,\Psi) \models (\nu\widetilde{c}_P)(\nu\widetilde{c}_Q)\{\!|\Psi'_P \otimes \Psi'_Q \vdash M_P \dot\leftrightarrow M_Q|\!\} \wedge (\nu\widetilde{c}_Q)\Psi'_Q \otimes C_P$, so by Lemma A.17 and Lemma A.2 $(\sigma,\Psi) \models (\nu\widetilde{c}_P)\{\!|\Psi'_P \otimes \Psi'_Q \vdash M_P \dot\leftrightarrow M_Q|\!\} \wedge \Psi'_Q \otimes C_P$.

By Lemma A.8 $(\sigma, \Psi) \models \Psi_Q \otimes ((\nu\widetilde{c_P})\{\!|\Psi'_P \vdash M_P \dot\leftrightarrow M_Q|\!\} \wedge C_P)$. so by Lemma A.7 $(\sigma, \Psi \otimes \Psi_Q\sigma) \models (\nu\widetilde{c_P})\{\!|\Psi'_P \vdash M_P \dot\leftrightarrow M_Q|\!\} \wedge C_P$. By substitutivity $(\sigma \cdot [y := M_Q\sigma], \Psi \otimes \Psi_Q\sigma) \models C'_P$, so by induction $\Psi \otimes \Psi_Q\sigma \rhd P\sigma \xrightarrow{(\overline{y}\ (\nu\widetilde{a})\widetilde{N})\sigma'} P'\sigma'$. By Lemma A.6 we get that $\mathcal{F}(Q\sigma) = (\nu\widetilde{b_Q})\Psi_{Q\sigma}$ such that $\Psi_{Q\sigma} \simeq \Psi_Q\sigma$ and $\mathrm{n}(\Psi_{Q\sigma}) = \mathrm{n}(\Psi_Q\sigma)$. By Lemma A.14 $\Psi \otimes \Psi_{Q\sigma} \rhd P\sigma \xrightarrow{(\overline{y}\ (\nu\widetilde{a})\widetilde{N})\sigma'} Q'\sigma'$.

Similarly $(\sigma \cdot [z := M_P\sigma], \Psi \otimes \Psi_P\sigma) \models C'_Q$, so by induction $\Psi \otimes \Psi_P\sigma \rhd Q\sigma \xrightarrow{(\underline{z}(\widetilde{x}))\sigma'} Q'\sigma'$. By Lemma A.6 we get that $\mathcal{F}(P\sigma) = (\nu\widetilde{b_P})\Psi_{P\sigma}$ such that $\Psi_{P\sigma} \simeq \Psi_P\sigma$ and $\mathrm{n}(\Psi_{P\sigma}) = \mathrm{n}(\Psi_P\sigma)$. By Lemma A.14 $\Psi \otimes \Psi_{P\sigma} \rhd Q\sigma \xrightarrow{(\underline{z}(\widetilde{x}))\sigma'} Q'\sigma'$.

Applying Lemma A.2 to $(\sigma, \Psi) \models \Psi_Q \otimes ((\nu\widetilde{c_P})\{\!|\Psi'_P \vdash M_P \dot\leftrightarrow M_Q|\!\})$ we get that $(\sigma, \Psi) \models \Psi_Q \otimes \{\!|\Psi'_P \vdash M_P \dot\leftrightarrow M_Q|\!\}$, By Lemma A.7 we have $(\sigma, \Psi \otimes \Psi_Q\sigma) \models \{\!|\Psi'_P \vdash M_P \dot\leftrightarrow M_Q|\!\}$, so by Lemma A.8 we get $(\sigma, \Psi \otimes \Psi_Q\sigma) \models \{\!|\Psi_P \vdash M_P \dot\leftrightarrow M_Q|\!\}$. Thus $\Psi \otimes \Psi_P\sigma \otimes \Psi_Q\sigma \vdash M_P\sigma \dot\leftrightarrow M_Q\sigma$, so $\Psi \otimes \Psi_{P\sigma} \otimes \Psi_{Q\sigma} \vdash M_P\sigma \dot\leftrightarrow M_Q\sigma$.

We then have the following derivation (remember that y and z are fresh for basically everything but themselves):

$$\textsc{cCom}\ \frac{\begin{array}{c}\Psi \otimes \Psi_{Q\sigma} \rhd P\sigma \xrightarrow{\overline{M_Q\sigma}\ (\nu\widetilde{a})\widetilde{N\sigma}} P\sigma \\ \Psi \otimes \Psi_{P\sigma} \rhd Q\sigma \xrightarrow{M_P\sigma(\widetilde{x})} Q'\sigma \qquad \Psi \otimes \Psi_{P\sigma} \otimes \Psi_{Q\sigma} \vdash M_P\sigma \dot\leftrightarrow M_Q\sigma\end{array}}{\Psi \rhd P\sigma \mid Q\sigma \xrightarrow{\tau} (\nu\widetilde{a})(P'\sigma \mid Q'\sigma[\widetilde{x} := \widetilde{N}\sigma])}\ \widetilde{a}\#Q\sigma$$

Since $\widetilde{a}, \widetilde{x}\#\sigma$ we have that $(\nu\widetilde{a})(P'\sigma \mid Q'\sigma[\widetilde{x} := \widetilde{N}\sigma]) = (\nu\widetilde{a})(P' \mid Q'[\widetilde{x} := \widetilde{N}])\sigma$.

Case sPar. In this case the inference looks like

$$\textsc{sPar}\ \frac{\Psi \otimes \Psi_Q \rhd P \xrightarrow[C]{\alpha} P'}{P \mid Q \xrightarrow[\mathcal{F}(Q)\otimes C]{\alpha} P' \mid Q}\ \begin{array}{c}\mathrm{bn}(\alpha)\#Q \\ \alpha = \tau \vee \mathrm{subj}(\alpha)\#Q\end{array}$$

We can assume that $\mathrm{subj}(\alpha)\#P$ (if not, use Lemma A.10 to find another subject). Assume that $\widetilde{x}\#\sigma, P \mid Q$.

Let $\mathcal{F}(Q) = (\nu\widetilde{b_Q})\Psi_Q$ with $\widetilde{b_Q}\#\alpha, C, \Psi, \sigma$. By Lemma A.7 $(\sigma, \Psi \otimes \Psi_Q\sigma) \models C$. By induction we then get that $P \xrightarrow[C]{\alpha} P'$ has a matching transition $\Psi \otimes \Psi_Q\sigma \rhd P\sigma \xrightarrow{\alpha\sigma} P'\sigma$. By Lemma A.6 we get that $\mathcal{F}(Q\sigma) = (\nu\widetilde{b_Q})\Psi_{Q\sigma}$ such that $\Psi_{Q\sigma} \simeq \Psi_Q\sigma$ and $\mathrm{n}(\Psi_{Q\sigma}) = \mathrm{n}(\Psi_Q\sigma)$. By Lemma A.14 $\Psi \otimes \Psi_{Q\sigma} \rhd P\sigma \xrightarrow{\alpha\sigma} P'\sigma$, so we can do the following concrete inference:

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

Case sScope. In this case the inference looks like

$$\textsc{sScope}\ \frac{P \xrightarrow[C]{\alpha} P'}{(\nu a)P \xrightarrow[(\nu a)C]{\alpha} (\nu a)P'}\ a\#\alpha, \Psi$$

We assume that $\mathrm{subj}(\alpha)\#(\nu a)P, \widetilde{x}$ (if not, use Lemma A.10 to find a new subject). We also assume $a\#\sigma, \Psi$ (bound names are fresh). By Lemma A.2 we then have that $(\sigma, \Psi) \models C$.

By induction we get that $P \xrightarrow[C]{\alpha} P'$ has a corresponding transition $\Psi \rhd P\sigma \xrightarrow{\alpha\sigma} P'\sigma$. We can then do the following concrete inference:

$$\textsc{cScope}\ \frac{\Psi\ \rhd\ P\sigma\ \xrightarrow{\alpha\sigma}\ P'\sigma}{\Psi\ \rhd\ (\nu a)(P\sigma)\ \xrightarrow{\alpha\sigma}\ (\nu a)(P'\sigma)}\ a\#\alpha\sigma,\Psi$$

Since $a\#\sigma$ we have that $(\nu a)(P\sigma) = ((\nu a)P)\sigma$ and $(\nu a)(P'\sigma) = ((\nu a)P')\sigma$.
Case sOpen. In this case the inference looks like

$$\textsc{sOpen}\ \frac{P\ \xrightarrow[C]{\overline{y}\ (\nu\widetilde{a})\widetilde{N}}\ P'}{(\nu a)P\ \xrightarrow[(\nu a)C]{\overline{y}\ (\nu\widetilde{a}\cup\{a\})\widetilde{N}}\ P'}\ \begin{array}{c} a\in \mathrm{n}(\widetilde{N}) \\ a\#\widetilde{a},y\end{array}$$

We can assume that $y\#P,\widetilde{a},a$ (if not, use Lemma A.10 to find another subject). Since $(\sigma,\Psi)\models(\nu a)C$ we also have that $(\sigma,\Psi)\models C$.

By induction we get that $P\ \xrightarrow[C]{\overline{y}\ (\nu\widetilde{a})\widetilde{N}}\ P'$ has a corresponding transition $\Psi\ \rhd\ P\sigma\ \xrightarrow{(\overline{y}\ (\nu\widetilde{a})\widetilde{N})\sigma}\ P'\sigma$.
We assume that $a\#\sigma$ (bound names are fresh). By Axiom 1 $a\in\mathrm{n}(\widetilde{N}\sigma)$, so we have the following concrete inference.

$$\textsc{cOpen}\ \frac{\Psi\ \rhd\ P\sigma\ \xrightarrow{(\overline{y}\ (\nu\widetilde{a})\widetilde{N})\sigma}\ P'\sigma}{\Psi\ \rhd\ (\nu a)P\sigma\ \xrightarrow{(\overline{y}\ (\nu\widetilde{a}\cup\{a\})\widetilde{N})\sigma}\ P'\sigma}\ \begin{array}{c} a\in\mathrm{n}(\widetilde{N}\sigma) \\ a\#\widetilde{a},\Psi\sigma,y\sigma\end{array}$$

Case sRep. In this case the inference looks like

$$\textsc{sRep}\ \frac{P\mid !P\ \xrightarrow[C]{\alpha}\ P'}{!P\ \xrightarrow[C]{\alpha}\ P'}$$

By induction $\Psi\ \rhd\ (P\mid !P)\sigma\ \xrightarrow{\alpha\sigma}\ P'\sigma$, so we can do the following derivation.

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

Case sBrOut. Here the transition is derived by

$$\textsc{sBrOut}\ \frac{x\#,M,\widetilde{N},P}{\overline{M}\,\widetilde{N}\,.\,P\ \xrightarrow[\mathbf{1}\vdash M\dot{\prec}x]{\overline{x}!\ \widetilde{N}}\ P}$$

We then have the corresponding concrete transition

$$\textsc{cBrOut}\ \frac{\Psi\vdash M\sigma\ \dot{\prec}\ x\sigma}{\Psi\ \rhd\ (\overline{M}\widetilde{N}.P)\sigma\ \xrightarrow{(\overline{x}\widetilde{N})\sigma}\ P\sigma}$$

Case sBrIn. Here the transition is derived by

$$\textsc{sBrIn}\ \frac{x,\widetilde{y}\#\Psi,M,P\quad x\#\widetilde{y}}{\underline{M}(\widetilde{y})\,.\,P\ \xrightarrow[\mathbf{1}\vdash x\dot{\succ}M]{\underline{x}?(\widetilde{y})}\ P}$$

Since $y\#\sigma$ we have that $(\underline{M}(\widetilde{y})\,.\,P)\sigma = \underline{M\sigma}(\widetilde{y})\,.\,P\sigma$ and that $(\underline{x}?(\widetilde{y}))\sigma = \underline{x\sigma}?(\widetilde{y})$. We then do the following derivation:

$$\text{cBrIn}\ \frac{\Psi \vdash x\sigma \dot{\succ} M\sigma}{\Psi \rhd \underline{M\sigma}(\widetilde{y}).P\sigma \xrightarrow{\underline{x\sigma}?(\widetilde{y})} P\sigma}$$

Case sBrMerge. Here the transition is derived by

$$\text{sBrMerge}\ \frac{P \xrightarrow[C_P]{\underline{x}?(\widetilde{y})} P' \qquad Q \xrightarrow[C_Q]{\underline{x}?(\widetilde{y})} Q'}{P \mid Q \xrightarrow[(\mathcal{F}(Q)\otimes C_P)\wedge(\mathcal{F}(P)\otimes C_Q)]{\underline{x}?(\widetilde{y})} P' \mid Q'}$$

By induction $\Psi \rhd P\sigma \xrightarrow{\underline{x\sigma}?(\widetilde{y})} P'\sigma$ and $\Psi \rhd Q\sigma \xrightarrow{\underline{x\sigma}?(\widetilde{y})} Q'\sigma$. By cBrMerge $\Psi \rhd P\sigma \mid Q\sigma \xrightarrow{\underline{x\sigma}?(\widetilde{y})} P'\sigma \mid Q'\sigma$.
Case sBrCom. Here the transition is derived by

$$\text{sBrCom}\ \frac{P \xrightarrow[C_P]{\overline{x}\,(\nu\widetilde{a})\widetilde{N}} P' \qquad Q \xrightarrow[C_Q]{\underline{x}?(\widetilde{y})} Q'}{P \mid Q \xrightarrow[(\mathcal{F}(Q)\otimes C_P)\wedge(\mathcal{F}(P)\otimes C_Q)]{\overline{x}\,(\nu\widetilde{a})\widetilde{N}} P' \mid Q'[\widetilde{y} := \widetilde{N}]}\ \widetilde{a}\#Q$$

By induction $\Psi \rhd P\sigma \xrightarrow{\overline{x\sigma}\,(\nu\widetilde{a})\widetilde{N\sigma}} P'\sigma$ and $\Psi \rhd Q\sigma \xrightarrow{\underline{x\sigma}?(\widetilde{y})} Q'\sigma$. By cBrCom $\Psi \rhd P\sigma \mid Q\sigma \xrightarrow{\overline{x\sigma}\,(\nu\widetilde{a})\widetilde{N\sigma}} P'\sigma \mid Q'[\widetilde{y} := \widetilde{N}]\sigma$.
Case sBrClose. Here the transition is derived by

$$\text{sBrClose}\ \frac{P \xrightarrow[C]{\overline{x}!\,(\nu\widetilde{a})\widetilde{N}} P'}{(\nu b)P \xrightarrow[\exists^b x.C]{\tau} (\nu b)(\nu\widetilde{a})P'}$$

By assumption there is K such that $b \in \mathrm{n}(K)$ and $(\sigma[x := K], \Psi) \models C$. We assume that $b\#\sigma$. Since $x\#P, \widetilde{a}$ we have $x\#P', \widetilde{N}$, so by induction $\Psi \rhd P\sigma \xrightarrow{\overline{K}!\,(\nu\widetilde{a})\widetilde{N\sigma}} P'\sigma$. We then have the following derivation.

$$\text{cBrClose}\ \frac{\Psi \rhd P\sigma \xrightarrow{\overline{K}!\,(\nu\widetilde{a})\widetilde{N\sigma}} P'\sigma}{\Psi \rhd (\nu b)P \xrightarrow{\tau} (\nu b)(\nu\widetilde{a})P'\sigma}\ b \in \mathrm{n}(K) \qquad \square$$

A.3. Proof of Completeness Theorem

The proof of the completeness theorem follows [Johansson et al. 2012], apart from the new cases for the broadcast rules, and the updated sCom rule.

Theorem A.19 (Theorem 7.12).

—*If* $\Psi \rhd P\sigma \xrightarrow{\tau} P'$ *then* $\exists C, Q \,.\, P \xrightarrow[C]{\tau} Q$, $Q\sigma = P'$ *and* $(\sigma, \Psi) \models C$.

—*If* $\Psi \rhd P\sigma \xrightarrow{\alpha} P'$, $\alpha \neq \tau$, $y\#P, \mathrm{bn}(\alpha), \sigma$, *and* $\mathrm{bn}(\alpha)\#\sigma, P$ *then* $\exists C, \alpha', Q.\ P \xrightarrow[C]{\alpha'} Q$, $Q\sigma = P'$, $\mathrm{subj}(\alpha') = y$, $\alpha'\sigma' = \alpha$, *and* $(\sigma', \Psi) \models C$ *where* $\sigma' = \sigma[y := \mathrm{subj}(\alpha)]$.

Proof. By induction on the inference of $\Psi \rhd P\sigma \xrightarrow{\alpha} P'\sigma$.

Case cIn. In this case the inference looks like

$$\text{cIn}\ \frac{\Psi \vdash M'\sigma \dot{\leftrightarrow} M}{\Psi \rhd (\underline{M'}(\widetilde{x}).P)\sigma \xrightarrow{\underline{M}(\widetilde{x})} P\sigma}$$

We know that $y\#\underline{M'}(\widetilde{x})\,.\,P, \widetilde{x}, \sigma$ and that $\widetilde{x}\#\sigma, \underline{M'}(\widetilde{x})\,.\,P$.
We let $Q = P$, and do the following derivation:

$$\textsc{sIn}\ \frac{}{\underline{M'}(\widetilde{x}).P \xrightarrow[\{\!|\mathbf{1}\vdash M' \dot{\leftrightarrow} y|\!\}]{\underline{y}(\widetilde{x})} P}\ y\#M', P, \widetilde{x}$$

Since $\Psi \vdash M'\sigma \dot{\leftrightarrow} M$ we have that $(\sigma[y := M], \Psi) \models \{\!|\mathbf{1} \vdash M' \dot{\leftrightarrow} y|\!\}$.
Case cOut. In this case the inference looks like

$$\textsc{cOut}\ \frac{\Psi \vdash M'\sigma \dot{\leftrightarrow} M}{\Psi \rhd (\overline{M'}\widetilde{N}\,.\,P)\sigma \xrightarrow{\overline{M}\widetilde{N\sigma}} P\sigma}$$

We know that $y\#M', \widetilde{N}, P$. We must find a constraint C such that $\overline{M'}\widetilde{N}\,.\,P \xrightarrow[C]{\overline{y}\widetilde{N}} P$ and $(\sigma[y := M], \Psi) \models C$. We let $Q = P$, $\widetilde{K} = \widetilde{N}$, and derive such a transition with

$$\textsc{sOut}\ \frac{}{\overline{M'}\widetilde{N}\,.\,P \xrightarrow[\{\!|\mathbf{1}\vdash M' \dot{\leftrightarrow} y|\!\}]{\overline{y}\widetilde{N}} P}\ y\#M', \widetilde{N}, P$$

Since $\Psi \vdash M'\sigma \dot{\leftrightarrow} M$ we have that $(\sigma[y := M], \Psi) \models \{\!|\mathbf{1} \vdash M' \dot{\leftrightarrow} y|\!\}$.
Case cCase. In this case the inference looks like

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

$\alpha = \tau$. By induction we know that $\Psi \rhd P_i\sigma \xrightarrow{\tau} P'$ has a matching transition $P_i \xrightarrow[C]{\tau} Q$ such that $(\sigma, \Psi) \models C$ and $Q\sigma = P'$. We also have that $(\sigma, \Psi) \models \{\!|\mathbf{1} \vdash \varphi_i|\!\}$. Together this gives us that $(\sigma, \Psi) \models C \wedge \{\!|\mathbf{1} \vdash \varphi_i|\!\}$.
$\alpha \neq \tau$. Since $y\#\mathbf{case}\ \widetilde{\varphi} : \widetilde{P}$ we have in particular that $y\#\varphi_i, P_i$. By induction we know that $\Psi \rhd P_i\sigma \xrightarrow{\alpha} P'$ has a matching transition $P_i \xrightarrow[C]{\alpha'} Q$ such that $(\sigma', \Psi) \models C$ and $Q\sigma = P'$. Since $\Psi \vdash \varphi_i\sigma$ we have that $(\sigma, \Psi) \models \{\!|\mathbf{1} \vdash \varphi_i|\!\}$, and since $y\#\varphi_i$ we also have that $(\sigma', \Psi) \models \{\!|\mathbf{1} \vdash \varphi_i|\!\}$. Together this gives us that $(\sigma', \Psi) \models C \wedge \{\!|\mathbf{1} \vdash \varphi_i|\!\}$.

We can then do the following derivation:

$$\textsc{sCase}\ \frac{P_i \xrightarrow[C]{\alpha_s} Q}{\mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \xrightarrow[C\wedge\{\!|\mathbf{1}\vdash\varphi_i|\!\}]{\alpha_s} Q}$$

Case cCom. The interesting case is the cCom case, where the inference looks like

$$\textsc{cCom}\ \frac{\begin{array}{c}\Psi \otimes \Psi_{P\sigma} \otimes \Psi_{Q\sigma} \vdash M \dot{\leftrightarrow} K \\ \Psi \otimes \Psi_{Q\sigma} \rhd P\sigma \xrightarrow{\overline{M}\,(\nu\widetilde{a})\widetilde{N\sigma}} P'\sigma \qquad \Psi \otimes \Psi_{P\sigma} \rhd Q\sigma \xrightarrow{\underline{K}(\widetilde{x})} Q'\sigma\end{array}}{\Psi \rhd (P \mid Q)\sigma \xrightarrow{\tau} (\nu\widetilde{a})(P' \mid Q'[\widetilde{x} := \widetilde{N}])\sigma}\ \widetilde{a}\#Q\sigma$$

Here $\mathcal{F}(P\sigma) = (\nu\widetilde{b}_{P\sigma})\Psi_{P\sigma}$ and $\mathcal{F}(Q\sigma) = (\nu\widetilde{b}_{Q\sigma})\Psi_{Q\sigma}$. We know that $\widetilde{b}_{P\sigma}\#\Psi, P\sigma, Q\sigma, \widetilde{b}_{Q\sigma}$ and $\widetilde{b}_{Q\sigma}\#\Psi, P\sigma, Q\sigma, \widetilde{b}_{P\sigma}, P$. We assume that $\widetilde{a}\#P, \widetilde{b}_{Q\sigma}, \sigma$ and $\widetilde{x}\#\widetilde{b}_{P\sigma}$. Let $y, z\#\ldots$. By Lemma A.11 we also have that $\widetilde{b}_{Q\sigma}\#P'\sigma$. By Lemma A.6 we get $\mathcal{F}(P) = (\nu\widetilde{b}_{P\sigma})\Psi_P$ with $\Psi_P\sigma \simeq \Psi_{P\sigma}$ and $\mathcal{F}(Q) = (\nu\widetilde{b}_{Q\sigma})\Psi_Q$ with $\Psi_Q\sigma \simeq \Psi_{Q\sigma}$.

By induction, $P \xrightarrow[C_P]{\overline{y}\,(\nu\widetilde{a})\widetilde{N'}} P'$ such that $\widetilde{N} = \widetilde{N}'\sigma$ and $(\sigma[y := M], \Psi \otimes \Psi_{Q\sigma}) \models C_P$.
By Lemma 7.10 $C_P = ((\nu\widetilde{c}_P)(\Psi'_P \vdash M_P \dot{\leftrightarrow} y) \wedge C'_P$ with $(\nu\widetilde{c}_P)\Psi'_P \equiv_{\mathrm{AC}} (\nu\widetilde{b}_P)\Psi_P$.
In the same way, by induction $Q \xrightarrow[C_Q]{\underline{z}(\widetilde{x})} Q'$ such that $(\sigma[z := K], \Psi \otimes \Psi_{P\sigma}) \models C_Q$.
By Lemma 7.10 $C_Q = ((\nu\widetilde{c}_Q)(\Psi'_Q \vdash M_Q \dot{\leftrightarrow} z) \wedge C'_Q$ with $(\nu\widetilde{c}_Q)\Psi'_Q \equiv_{\mathrm{AC}} (\nu\widetilde{b}_Q)\Psi_Q$.
We can then do the following inference:

$$\textsc{sCom}\ \frac{P \xrightarrow[(\nu\widetilde{c}_P)\{\!|\Psi'_P \vdash M_P \dot{\leftrightarrow} y|\!\}\wedge C'_P]{\overline{y}\,(\nu\widetilde{a})\widetilde{N'}} P' \qquad Q \xrightarrow[(\nu\widetilde{c}_Q)\{\!|\Psi'_Q \vdash M_Q \dot{\leftrightarrow} z|\!\}\wedge C'_Q]{\underline{z}(\widetilde{x})} Q'}{P \mid Q \xrightarrow[C]{\tau} (\nu\widetilde{a})(P' \mid Q'[\widetilde{x} := \widetilde{N}'])}\ \widetilde{a}\#Q$$

where $C = (\nu\widetilde{c}_P\widetilde{c}_Q)\{\!|\Psi'_P \otimes \Psi'_Q \vdash M_P \dot{\leftrightarrow} M_Q|\!\} \wedge ((\nu\widetilde{c}_Q)\Psi'_Q \otimes C'_P) \wedge ((\nu\widetilde{c}_P)\Psi'_P \otimes C'_Q)$. It remains to show that $(\sigma, \Psi) \models C$. We consider each conjunct in turn.
By Lemma A.8 $(\sigma[z := K], \Psi \otimes \Psi_{P\sigma} \models (\nu\widetilde{b}_Q)(\Psi_Q \vdash M_Q \dot{\leftrightarrow} z)$ and $(\sigma[y := M], \Psi \otimes \Psi_{Q\sigma}) \models (\nu\widetilde{b}_P)(\Psi_P \vdash M_P \dot{\leftrightarrow} y)$. Thus $\Psi \otimes \Psi_{P\sigma} \otimes \Psi_Q\sigma \vdash M_Q\sigma \dot{\leftrightarrow} K$ and $\Psi \otimes \Psi_{Q\sigma} \otimes \Psi_P\sigma \vdash M_P\sigma \dot{\leftrightarrow} M$. By AC of entailment of $\dot{\leftrightarrow}$ and $\otimes$ modulo $\simeq$ $\Psi \otimes \Psi_P\sigma \otimes \Psi_Q\sigma \vdash M_P\sigma \dot{\leftrightarrow} M_Q\sigma$. By Lemma A.4 $\Psi \otimes \Psi'_P\sigma \otimes \Psi'_Q\sigma \vdash M_P\sigma \dot{\leftrightarrow} M_Q\sigma$, so $(\sigma, \Psi) \models \Psi'_P\sigma \otimes \Psi'_Q \vdash M_P\sigma \dot{\leftrightarrow} M_Q\sigma$. By Lemma A.2 $(\sigma, \Psi) \models ((\nu\widetilde{c}_P\widetilde{c}_Q)\Psi'_P \otimes \Psi'_Q \vdash M_P \dot{\leftrightarrow} M_Q)$, which is the first conjunct.
By Lemma A.6 $(\sigma[y := M], \Psi \otimes \Psi_Q\sigma) \models C_P$ so by Lemma A.7 $(\sigma[y := M], \Psi) \models \mathcal{F}(Q) \otimes C'_P$. Since $y\#Q, C'_P$ we have $(\sigma, \Psi) \models \mathcal{F}(Q) \otimes C'_P$. By Lemma A.8 $(\sigma, \Psi) \models ((\nu\widetilde{c}_Q)\Psi'_Q) \otimes C'_P$.
In the same way, by Lemma A.6 $(\sigma[z := K], \Psi \otimes \Psi_P\sigma) \models C_Q$ so by Lemma A.7 $(\sigma[z := K], \Psi) \models \mathcal{F}(P) \otimes C'_Q$, Since $z\#P, C'_Q$ we have $(\sigma, \Psi) \models \mathcal{F}(P) \otimes C'_Q$. By Lemma A.8 $(\sigma, \Psi) \models ((\nu\widetilde{c}_P)\Psi'_P) \otimes C'_Q$.

Case cPar. In this case the inference looks like

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

where $\mathcal{F}(Q\sigma) = (\nu\widetilde{b}_{Q\sigma})\Psi_{Q\sigma}$ with $\widetilde{b}_{Q\sigma}\#\Psi, \Psi\sigma, P\sigma, \alpha, y$. By Lemma A.6 $\mathcal{F}(Q) = (\nu\widetilde{b}_{Q\sigma})\Psi_Q$ such that $\Psi_{Q\sigma} \simeq \Psi_Q\sigma$ and $\mathrm{n}(\Psi_{Q\sigma}) = \mathrm{n}(\Psi_Q\sigma)$.
By Lemma A.11 we also have that $\widetilde{b}_{Q\sigma}\#P'\sigma$. Since $y\#\widetilde{b}_{Q\sigma}, Q, \sigma$ we get that $y\#\Psi_Q\sigma$. Together with $y\#\sigma$ this gives us that $y\#\Psi_Q$.
By induction we know that $\Psi \otimes \Psi_{Q\sigma} \rhd P\sigma \xrightarrow{\alpha} P'\sigma$ has a matching transition $P \xrightarrow[C]{\alpha'} P'$ such that $(\sigma', \Psi \otimes \Psi_{Q\sigma}) \models C$.
We can then do the following symbolic inference:

$$\textsc{sPar}\ \frac{P \xrightarrow[C]{\alpha'} P'}{P \mid Q \xrightarrow[\mathcal{F}(Q)\otimes C]{\alpha'} P' \mid Q}\ \begin{array}{l}\widetilde{x}\#Q\\ y\#Q\end{array}$$

Lemma A.6 yields that $(\sigma', \Psi \otimes \Psi_Q\sigma) \models C$, so by Lemma A.7 $(\sigma', \Psi) \models \mathcal{F}(Q) \otimes C$.
Case cScope. In this case the transition is

$$\Psi \rhd ((\nu a)P)\sigma \xrightarrow{\alpha} ((\nu a)P')\sigma$$

Let b be a sufficiently fresh name, and let $p = (a\ b)$. By applying the substitution and using α-conversion to avoid capture, this transition is equivalent to

$$\Psi \rhd (\nu b)((p \cdot P)\sigma) \xrightarrow{\alpha} (\nu b)((p \cdot P')\sigma)$$

This transition is inferred like

$$\textsc{cScope}\ \frac{\Psi \rhd (p \cdot P)\sigma \xrightarrow{\alpha} (p \cdot P')\sigma}{\Psi \rhd (\nu b)((p \cdot P)\sigma) \xrightarrow{\alpha} (\nu b)((p \cdot P')\sigma)}\ b\#\alpha, \Psi$$

We know that $y\#(\nu a)P$. Since $y\#(\nu b)(p \cdot P)$ and $b\#y$ we have that $y\#p \cdot P$.

By induction we have that $\Psi \rhd (p \cdot P)\sigma \xrightarrow{\alpha} (p \cdot P')\sigma$ has a matching transition $p \cdot P \xrightarrow[p\cdot C]{\alpha'} p \cdot P'$ such that $(\sigma', \Psi) \models C$.

We pick $b\#\sigma, \alpha$, so $b\#\alpha'$ by Axiom 1.

We then do the following symbolic inference:

$$\textsc{sScope}\ \frac{p \cdot P \xrightarrow[p\cdot C]{\alpha'} p \cdot P'}{(\nu b)(p \cdot P) \xrightarrow[(\nu b)p\cdot C]{\alpha'} (\nu b)(p \cdot P')}\ b\#\alpha$$

Since $(\sigma', \Psi) \models C$ and $b\#\sigma, \Psi, y, \mathrm{subj}(\alpha)$ we also have that $(\sigma', \Psi) \models (\nu b)p \cdot C$.

By α-converting the final transition we get that

$$(\nu a)(P) \xrightarrow[(\nu a)C]{\alpha'} (\nu a)(P')$$

Case $\textsc{cOpen}$. In this case the transition looks like

$$(\nu a)P\sigma \xrightarrow{\overline{M}\ (\nu\widetilde{a} \cup \{a\})\widetilde{N\sigma}} P'\sigma$$

Let b be a sufficiently fresh name, and let $p = (a\ b)$. By applying the substitution and using α-conversion to avoid capture, this transition is equivalent to

$$(\nu b)((p \cdot P)\sigma) \xrightarrow{\overline{M}\ (\nu\widetilde{a} \cup \{b\})\widetilde{(p\cdot\widetilde{N})\sigma}} (p \cdot P')\sigma$$

This transition is inferred like

$$\textsc{cOpen}\ \frac{\Psi \rhd (p \cdot P)\sigma \xrightarrow{\overline{M}\ (\nu\widetilde{a})\widetilde{(p\cdot\widetilde{N})\sigma}} (p \cdot P')\sigma}{\Psi \rhd (\nu b)((p \cdot P)\sigma) \xrightarrow{\overline{M}\ (\nu\widetilde{a} \cup \{b\})\widetilde{(p\cdot\widetilde{N})\sigma}} (p \cdot P')\sigma}\ \begin{array}{c} b \in \mathrm{n}((p \cdot \widetilde{N})\sigma) \\ b\#\widetilde{a}, \Psi\sigma, M \end{array}$$

We know that $y\#(\nu a)P, x,\ x\#\sigma, (\nu a)P$. Since $y\#(\nu b)(p \cdot P)$ and $b\#y$ we have that $y\#p \cdot P$, and similarly we get that $x\#p \cdot P$

By induction we have that $\Psi \rhd (p \cdot P)\sigma \xrightarrow{\overline{M}\ (\nu\widetilde{a})\widetilde{(p\cdot\widetilde{N})\sigma}} (p \cdot P')\sigma$ has a matching transition $p \cdot P \xrightarrow[p\cdot C]{\overline{y}\ (\nu\widetilde{a})\widetilde{(p\cdot\widetilde{N})}} p \cdot P'$ such that $(\sigma[y := M], \Psi) \models p \cdot C$.

We then infer:

$$\textsc{sOpen}\ \frac{p \cdot P \xrightarrow[p\cdot C]{\overline{y}\ (\nu\widetilde{a})\widetilde{p\cdot\widetilde{N}}} p \cdot P'}{(\nu b)(p \cdot P) \xrightarrow[(\nu b)p\cdot C]{\overline{y}\ (\nu\widetilde{a} \cup \{b\})\widetilde{p\cdot\widetilde{N}}} p \cdot P'}\ \begin{array}{c} b \in \mathrm{n}(p \cdot \widetilde{N}) \\ b\#\widetilde{a}, y \end{array}$$

Since $b\#\sigma, \Psi, M, y$ and we have that $(\sigma', \Psi) \models p \cdot C$ we also have that $(\sigma', \Psi) \models (\nu b)p \cdot C$.

By α-converting the final transition we get:

$$(\nu a)P \xrightarrow[(\nu a)C]{\overline{y}\ (\nu\widetilde{a}\cup\{a\})\widetilde{N}} P'$$

Case cRep. In this case the inference looks like

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

If $\alpha \neq \tau$*:.* We have that $y\#!P, \mathrm{bn}(\alpha)$, which gives us that $y\#P, !P, \mathrm{bn}(\alpha)$. By induction we get that $P \mid !P \xrightarrow[C]{\alpha'} P'$ and that $(\sigma', \Psi) \models C$.
We do the following derivation

$$\textsc{sRep}\ \frac{P \mid !P \xrightarrow[C]{\alpha'} P'}{!P \xrightarrow[C]{\alpha'} P'}$$

Case cBrIn. Here the transition is derived by

$$\textsc{cBrIn}\ \frac{\Psi \vdash K \mathrel{\dot{\succ}} M\sigma \qquad \widetilde{x}\#\Psi, M\sigma, P\sigma}{\Psi \ \rhd\ \underline{M\sigma}(\widetilde{x}) \,.\, P\sigma \ \xrightarrow{\underline{K}?(\widetilde{x})}\ P\sigma}$$

We know that $y\#\underline{M}(\widetilde{x}) \,.\, P, \widetilde{x}, \sigma$ and assume that $\widetilde{x}\#\sigma, M$.
We let $Q = P$, and do the following derivation:

$$\textsc{sBrIn}\ \frac{\widetilde{x}, y\#M, P \quad y\#\widetilde{x}}{\underline{M}(\widetilde{x}) \,.\, P \ \xrightarrow[\mathbf{1}\vdash y\dot{\succ} M]{\underline{y}?(\widetilde{x})}\ P}$$

Since $\Psi \vdash K \mathrel{\dot{\succ}} M\sigma$ we get $(\sigma[y := K], \Psi) \models \mathbf{1} \vdash y \mathrel{\dot{\succ}} M$.
Case cBrOut. Here the transition is derived by

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

We know that $y\#\overline{M}\,\widetilde{N} \,.\, P, \sigma$. We let $Q = P$, and do the following derivation:

$$\textsc{sBrOut}\ \frac{x\#M, N, P}{\overline{M}\,\widetilde{N} \,.\, P \ \xrightarrow[\mathbf{1}\vdash M\dot{\prec} x]{\overline{x}!\ \widetilde{N}}\ P}$$

Case cBrMerge. Here the transition is derived by

$$\textsc{cBrMerge}\ \frac{\Psi \otimes \Psi_{Q\sigma} \ \rhd\ P\sigma \ \xrightarrow{\underline{K}?(\widetilde{y})}\ P' \qquad \Psi \otimes \Psi_{P\sigma} \ \rhd\ Q\sigma \ \xrightarrow{\underline{K}?(\widetilde{y})}\ Q'}{\Psi \ \rhd\ P\sigma \mid Q\sigma \ \xrightarrow{\underline{K}?(\widetilde{y})}\ P' \mid Q'}$$

Here $\mathcal{F}(P\sigma) = (\nu\widetilde{b}_{P\sigma})\Psi_{P\sigma}$ and $\mathcal{F}(Q\sigma) = (\nu\widetilde{b}_{Q\sigma})\Psi_{Q\sigma}$. We know that $\widetilde{b}_{P\sigma}\#\Psi, P\sigma, Q\sigma, \widetilde{b}_{Q\sigma}$ and $\widetilde{b}_{Q\sigma}\#\Psi, P\sigma, Q\sigma, \widetilde{b}_{P\sigma}, P$. We assume that $x\#\widetilde{b}_{P\sigma}$.
By induction $P \xrightarrow[C_P]{\underline{x}(\widetilde{y})} P''$ and $Q \xrightarrow[C_Q]{\underline{x}(\widetilde{y})} Q''$ such that $(\sigma[x := K], \Psi \otimes \Psi_{Q\sigma}) \models C_P$ and $(\sigma[x := K], \Psi \otimes \Psi_{P\sigma}) \models C_Q$ and $P' = P''\sigma$ and $Q' = Q''\sigma$.

We then have the following derivation.

$$\textsc{sBrMerge}\ \frac{P \xrightarrow[C_P]{\underline{x}?(\widetilde{y})} P'' \qquad Q \xrightarrow[C_Q]{\underline{x}?(\widetilde{y})} Q''}{P \mid Q \xrightarrow[(\mathcal{F}(Q)\otimes C_P)\wedge(\mathcal{F}(P)\otimes C_Q)]{\underline{x}?(\widetilde{y})} P'' \mid Q''}$$

By Lemma A.6 we get $\mathcal{F}(P) = (\nu\widetilde{b}_{P\sigma})\Psi_P$ with $\Psi_P\sigma \simeq \Psi_{P\sigma}$ and $\mathcal{F}(Q) = (\nu\widetilde{b}_{Q\sigma})\Psi_Q$ with $\Psi_Q\sigma \simeq \Psi_{Q\sigma}$. By Lemma A.7 $(\sigma[x := K], \Psi) \models \mathcal{F}(Q) \otimes C_P$ and $(\sigma[x := K], \Psi) \models \mathcal{F}(P) \otimes C_Q$.
Case $\textsc{cBrCom}$. Here the transition is derived by

$$\textsc{cBrCom}\ \frac{\begin{array}{c}\Psi \otimes \Psi_{Q\sigma} \vartriangleright P\sigma \xrightarrow{\overline{K}\ (\nu\widetilde{a})\widetilde{N}} P' \\ \Psi \otimes \Psi_{P\sigma} \vartriangleright Q\sigma \xrightarrow{\underline{K}?(\widetilde{x})} Q'\end{array}}{\Psi \vartriangleright P\sigma \mid Q\sigma \xrightarrow{\overline{y}\ (\nu\widetilde{a})\widetilde{N}} P' \mid Q'[y := N]}\ \widetilde{a}\#Q\sigma$$

Here $\mathcal{F}(P\sigma) = (\nu\widetilde{b}_{P\sigma})\Psi_{P\sigma}$ and $\mathcal{F}(Q\sigma) = (\nu\widetilde{b}_{Q\sigma})\Psi_{Q\sigma}$. We know that $\widetilde{b}_{P\sigma}\#\Psi, P\sigma, Q\sigma, \widetilde{b}_{Q\sigma}$ and $\widetilde{b}_{Q\sigma}\#\Psi, P\sigma, Q\sigma, \widetilde{b}_{P\sigma}, P$. We assume that $\widetilde{x}\#P, \widetilde{b}_{P\sigma}$.
By induction $P \xrightarrow[C_P]{\overline{y}\ (\nu\widetilde{a})\widetilde{M}} P''$ and $Q \xrightarrow[C_Q]{\underline{y}(\widetilde{x})} Q''$ such that $(\sigma[y := K], \Psi \otimes \Psi_{Q\sigma}) \models C_P$ and $(\sigma[y := K], \Psi \otimes \Psi_{P\sigma}) \models C_Q$ and $P' = P''\sigma$ and $\widetilde{M}\sigma = \widetilde{N}$ and $Q' = Q''\sigma$.
We then have the following derivation.

$$\textsc{sBrCom}\ \frac{P \xrightarrow[C_P]{\overline{y}\ (\nu\widetilde{a})\widetilde{M}} P'' \qquad Q \xrightarrow[C_Q]{\underline{y}?(\widetilde{x})} Q''}{P \mid Q \xrightarrow[(\mathcal{F}(Q)\otimes C_P)\wedge(\mathcal{F}(P)\otimes C_Q)]{\overline{y}\ (\nu\widetilde{a})\widetilde{M}} P'' \mid Q''[\widetilde{y} := \widetilde{N}]}\ \widetilde{a}\#Q$$

By Lemma A.6 we get $\mathcal{F}(P) = (\nu\widetilde{b}_{P\sigma})\Psi_P$ with $\Psi_P\sigma \simeq \Psi_{P\sigma}$ and $\mathcal{F}(Q) = (\nu\widetilde{b}_{Q\sigma})\Psi_Q$ with $\Psi_Q\sigma \simeq \Psi_{Q\sigma}$. By Lemma A.7 $(\sigma[y := K], \Psi) \models \mathcal{F}(Q) \otimes C_P$ and $(\sigma[y := K], \Psi) \models \mathcal{F}(P) \otimes C_Q$.
Case $\textsc{cBrClose}$. Here the transition is derived by

$$\textsc{cBrClose}\ \frac{\Psi \vartriangleright P\sigma \xrightarrow{\overline{K}!\ (\nu\widetilde{a})\widetilde{N}} P'}{\Psi \vartriangleright (\nu b)P\sigma \xrightarrow{\tau} (\nu b)(\nu\widetilde{a})P'}\ b \in \mathrm{n}(K)$$

By induction $P \xrightarrow[C]{\overline{y}!\ (\nu\widetilde{a})\widetilde{M}} P''$ such that $\widetilde{M}\sigma = \widetilde{N}$ and $P''\sigma = P'$ and $(\sigma[y := K], \Psi) \models C$. We then do

$$\textsc{sBrClose}\ \frac{P \xrightarrow[C]{\overline{x}!\ (\nu\widetilde{a})\widetilde{M}} P'}{(\nu b)P \xrightarrow[\exists^b x.C]{\tau} (\nu b)(\nu\widetilde{a})P'}$$

where $(\sigma, \Psi) \models \exists^b x.C$. □

Paper II

Session Types for Broadcasting

Dimitrios Kouzapas
University of Glasgow
dimitrios.kouzapas@glasgow.ac.uk

Ramūnas Gutkovas
Uppsala University
ramunas.gutkovas@it.uu.se

Simon J. Gay
University of Glasgow
simon.gay@glasgow.ac.uk

Up to now session types have been used under the assumptions of point to point communication, to ensure the linearity of session endpoints, and reliable communication, to ensure send/receive duality. In this paper we define a session type theory for broadcast communication semantics that by definition do not assume point to point and reliable communication. Our session framework lies on top of the parametric framework of broadcasting ψ-calculi, giving insights on developing session types within a parametric framework. Our session type theory enjoys the properties of soundness and safety. We further believe that the solutions proposed will eventually provide a deeper understanding of how session types principles should be applied in the general case of communication semantics.

1 Introduction

Session types [5, 7, 6] allow communication protocols to be specified as types and verified by type-checking. Up to now, session type systems have assumed reliable, point to point message passing communication. Reliability is important to maintain send/receive duality, and point to point communication is required to ensure session endpoint linearity.

In this paper we propose a session type system for unreliable broadcast communication. Developing such a system was challenging for two reasons: (i) we needed to extend binary session types to handle unreliability as well as extending the notion of session endpoint linearity, and (ii) the reactive control flow of a broadcasting system drove us to consider typing patterns of communication interaction rather than communication prefixes. The key ideas are (i) to break the symmetry between the s^+ and s^- endpoints of channel s, allowing s^+ (uniquely owned) to broadcast and gather, and s^- to be shared; (ii) to implement (and type) the gather operation as an iterated receive. We retain the standard binary session type constructors.

We use ψ-calculi [1] as the underlying process framework, and specifically we use the extension of the ψ-calculi family with broadcast semantics [2]. ψ-calculi provide a parametric process calculus framework for extending the semantics of the π-calculus with arbitrary data structures and logical assertions. Expressing our work in the ψ-calculi framework allows us to avoid defining a new operational semantics, instead defining the semantics of our broadcast session calculus by translation into a broadcast ψ-calculus. Establishing a link between session types and ψ-calculi is therefore another contribution of our work.

Intuition through Demonstration. We demonstrate the overall intuition by means of an example. For the purpose of the demonstration we imply a set of semantics, which we believe are self explanatory. Assume types $S = !T; ?T; \mathtt{end}$, $\overline{S} = ?T; !T; \mathtt{end}$ for some data type T, and typings $s^+ : S$, $s^- : \overline{S}$, $a : \langle S \rangle$, $v : T$. The session type prefix $!T$ means *broadcast* when used by s^+, and *single destination send* when used by s^-. Dually, $?T$ means *gather* when used by s^+, and *single origin receive* when used by s^-.

Session Initiation through broadcast, creating an arbitrary number of receiving endpoints:
$\overline{a}s^-.P_0 \mid ax.P_1 \mid ax.P_2 \mid ax.P_3 \longrightarrow P_0 \mid P_1\{s^-/x\} \mid P_2\{s^-/x\} \mid ax.P_3$

Submitted to:

Due to unreliability, $ax.P_3$ did not initiate the session. We denote the initiating and accepting session endpoint as s^+ and s^- respectively.

Session Broadcast from the s^+ endpoint results in multiple s^- endpoints receiving:

$$s^+!\langle v\rangle;P_0 \mid s^-?(x);P_1 \mid s^-?(x);P_2 \mid s^-?(x);P_3 \longrightarrow P_0 \mid P_1\{v/x\} \mid P_2\{v/x\} \mid s^-?(x);P_3$$

Due to unreliability, a process (in the above reduction, process $s^-?(x);P_3$) might not receive a message. In this case the session endpoint that belongs to process $s^-?(x);P_3$ is considered broken, and later we will introduce a recovery mechanism.

Gather: The next challenge is to achieve the sending of values from the s^- endpoints to the s^+ endpoint. The gather prefix $s^+?(x);P_0$ is translated (in Section 4) into a process that iteratively receives messages from the s^- endpoints, non-deterministically stopping at some point and passing control to P_0.

$$s^+?(x);P_0 \mid s^-!\langle v_1\rangle;P_1 \mid s^-!\langle v_2\rangle;P_2 \mid s^-!\langle v_3\rangle;P_3 \longrightarrow^* P_0' \mid P_1 \mid s^-!\langle v_2\rangle;P_2 \mid P_3$$

with $P_0\{\{v_1,v_3\}/x\} \longrightarrow P_0'$.

After two reductions the messages from processes $s^-!\langle v_1\rangle;P_1$ and $s^-!\langle v_3\rangle;P_3$ had been received by the s^+ endpoint. On the third reduction the s^+ endpoint decided not to wait for more messages and proceeded with its session non-deterministically, resulting in a broken sending endpoint ($s^-!\langle v_2\rangle;P_2$), which is predicted by the unreliability of the broadcast semantics. The received messages, v_1 and v_2, were delivered to P_0 as a set.

Prefix Enumeration: The above semantics, although capturing broadcast session initiation and interaction, still violate session type principles due to the unreliability of communication:

$$\begin{array}{lll} s^+!\langle v_1\rangle;s^+!\langle v_2\rangle;\mathbf{0} \mid s^-?(x);s^-?(y);\mathbf{0} \mid s^-?(x);s^-?(y);\mathbf{0} & \longrightarrow & \\ \qquad s^+!\langle v_2\rangle;\mathbf{0} \mid s^-?(y);\mathbf{0} \mid s^-?(x);s^-?(y);\mathbf{0} & \longrightarrow & \mathbf{0} \mid \mathbf{0} \mid s^-?(y);\mathbf{0} \end{array}$$

The first reduction produced a broken endpoint, $s^-?(x);s^-?(y);\mathbf{0}$, while the second reduction reduces the broken endpoint. This situation is not predicted by session type principles. To solve this problem we introduce an enumeration on session prefixes:

$$(s^+,1)!\langle v_1\rangle;(s^+,2)!\langle v_2\rangle;\mathbf{0} \mid (s^-,1)?(x);(s^-,2)?(y);\mathbf{0} \mid (s^-,1)?(x);(s^-,2)?(y);\mathbf{0} \longrightarrow$$
$$(s^+,2)!\langle v_2\rangle;\mathbf{0} \mid (s^-,2)?(y);\mathbf{0} \mid (s^-,1)?(x);(s^-,2)?(y);\mathbf{0} \longrightarrow \mathbf{0} \mid \mathbf{0} \mid (s^-,1)?(x);(s^-,2)?(y);\mathbf{0}$$

The intuitive semantics described in this example are encoded in the ψ-calculi framework. From this it follows that all the operational semantics, typing system and theorems are stated using the ψ-calculus framework.

Contributions. This paper is the first to propose session types as a type meta-theory for the ψ-calculi. Applying session semantics in such a framework meets the ambition that session types can effectively describe general communication semantics. A step further is the development of a session type framework for broadcast communication semantics. It is the first time that session types escape the assumptions of point to point communication and communication reliability. We also consider as a contribution the fact that we use enumerated session prefixes in order to maintain consistency of the session communication. We believe that this technique will be applied in future session type systems that deal with unreliable and/or unpredictable communication semantics.

Related Work. Carbone *et al.* [4] extended binary session types with exceptions, allowing both parties in a session to collaboratively handle a deviation from the standard protocol. Capecchi *et al.* [3] generalized a similar approach to multi-party sessions. In contrast, our recovery processes allow a broadcast sender or receiver to autonomously handle a failure of communication. Although it might be possible to represent broadcasting in multi-party session type systems, by explicitly specifying separate messages from a single source to a number of receivers, all such systems assume reliable communication for every message.

2 Broadcast Session Calculus

We define an intuitive syntax for our calculus. The syntax below will be encoded in the ψ-calculi framework so that it will inherit the operational semantics.

$$\begin{array}{rcl} P,R & ::= & \bar{a}s^-.P \mid ax.P \mid s^+!\langle v\rangle;P \mid s^-!\langle v\rangle;P \mid s^+?(x);P \mid s^-?(x);P \\ & \mid & s^+\oplus l;P \mid s^-\&\{l_i:P_i\} \mid P \bowtie R \mid \mathbf{0} \mid \mu X.P \mid X \mid P \mid P \mid (\nu\, n)P \end{array}$$

Processes $\bar{a}s^-.P$, $ax.P$ are prefixed with session initiation operators that interact following the broadcast semantics. Processes $s^+!\langle v\rangle;P$, $s^-!\langle v\rangle;P$ define two different sending patterns. For the s^+ endpoint we have a broadcast send. For the s^- endpoint we have a unicast send. Processes $s^+?(x);P$, $s^-?(x);P$ assume gather (i.e. the converse of broadcast send) and unicast receive, respectively. We allow selection and branching $s^+\oplus l;P$, $s^-\&\{l_i:P_i\}$ only for broadcast semantics from the s^+ to the s^- endpoint. Each process can carry a recovery process R with the operator $P \bowtie R$. The process can proceed non-deterministically to recovery if the session endpoint is broken due to the unreliability of the communication. Process R is carried along as process P reduces its prefixes. The rest of the processes are standard π-calculus processes.

Structural congruence is defined over the abelian monoid defined by the parallel operator (|) and the inactive process ($\mathbf{0}$) and additionally satisfies the rules:

$$(\nu\, n)\mathbf{0} \equiv \mathbf{0} \qquad P \mid (\nu\, n)Q \equiv (\nu\, n)(P \mid Q) \text{ if } n \notin \mathtt{fn}(P)$$

3 Broadcast ψ-Calculi

Here we define the parametric framework of ψ-calculi for broadcast. For a detailed description of ψ-calculi we refer the reader to [1].

We fix a countably infinite set of names $\mathcal{N}$ ranged over by a,b,x. ψ-calculi are parameterised over three nominal sets: terms ($\mathbf{T}$ ranged over by M,N,L), conditions ($\mathbf{C}$ ranged over by φ), and assertions ($\mathbf{A}$ ranged over by Ψ); and operators: channel equivalence, broadcast output and input connectivity $\dot{\leftrightarrow}, \dot{\prec}, \dot{\succ}$: $\mathbf{T}\times\mathbf{T}\to\mathbf{C}$, assertion composition $\otimes : \mathbf{A}\times\mathbf{A}\to\mathbf{A}$, unit $\mathbf{1}\in\mathbf{A}$, entailment relation $\vdash \mathbf{A}\times\mathbf{C}$, and a substitution function substituting terms for names for each set. The channel equivalence is required to be symmetric and transitive, and assertion composition forms abelian monoid with $\mathbf{1}$ as the unit element. We do not require output and input connectivity be symmetric, i.e., $\Psi \vdash M \dot{\prec} N$ is not equivalent to $\Psi \vdash N \dot{\succ} M$, however for technical reasons require that the names of L should be included in N and M whenever $\Psi \vdash N \dot{\prec} L$ or $\Psi \vdash L \dot{\succ} M$. The agents are defined as follows

$$P,Q \;::=\; M(\lambda\tilde{a})N.P \mid \overline{M}N.P \mid \mathbf{case}\,\varphi_1 : P_1 \,[]\, \ldots \,[]\, \varphi_n : P_n \mid (\!|\Psi|\!) \mid (\nu a)P \mid P \mid Q \mid\; !P$$

where $\tilde{a}$ bind into N and P. The assertions in the case and replicated agents are required to be guarded. We abbreviate the case agent as $\mathbf{case}\,\tilde{\varphi} : \tilde{P}$; we write $\mathbf{0}$ for $(\!|\mathbf{1}|\!)$, we also write $a\#X$ to intuitively mean that name a does not occur freely in X.

We give a brief intuition behind the communication parameters: Agents unicast whenever their subject of their prefixes are channel equivalent, to give an example, $\overline{M}L.P$ and $N(\lambda\tilde{a})K.Q$ communicate whenever $\Psi \vdash M \dot{\leftrightarrow} N$. In contrast, broadcast communication is mediated by a broadcast channel, for example, the agents $\overline{M}N.P$ and $M_i(\lambda\tilde{a}_i)N_i.P_i$ (for $i > 0$) communicate if they can broadcast and receive from the same channel $\Psi \vdash M \dot{\prec} K$ and $\Psi \vdash K \dot{\succ} M_i$.

In addition to the standard structural congruence laws of pi-calculus we define the following, with the assumption that $a \,\#\, \tilde{\varphi}, M, N, \tilde{x}$ and π is permutation of a sequence.

$$\begin{array}{cc} (\nu a)\mathbf{case}\,\tilde{\varphi} : \tilde{P} \equiv_\Psi \mathbf{case}\,\tilde{\varphi} : \widetilde{(\nu a)P} & \mathbf{case}\,\tilde{\varphi} : \tilde{P} \equiv_\Psi \mathbf{case}\,\pi\cdot(\tilde{\varphi} : \tilde{P}) \\ \overline{M}N.(\nu a)P \equiv_\Psi (\nu a)\overline{M}N.P & M(\lambda\tilde{x})N.(\nu a)P \equiv_\Psi (\nu a)M(\lambda\tilde{x})N.P \end{array}$$

The following is a reduction context with two types of numbered holes (condition hole $\hat{[]}$ and process hole $[]$) such that no two holes of the same type have the same number.

$$C \;::=\; (\mathbf{case}\,\hat{[]}_j : C \;[]\; \widetilde{\varphi} : \widetilde{P}) \mid C \quad\mid\quad \textstyle\prod_{k>0}[]_{i_k}$$

The filling of the holes is defined in the following way: filling a process (resp. condition) hole with a assertion guarded process (resp. condition) taken from the number position of a given sequence. We denote filling of holes as $C[(\varphi_i)_{i\in I};(P_j)_{j\in J};(Q_k)_{k\in K}]$ where the first component is for filling the condition holes and the other two are for filling process holes.

We require that I is equal to the numbering set of condition holes and furthermore J and K are disjoint and their union is equal to the numbering set of context for the process holes. We also require that every J numbered hole is either in parallel with any of the K holes or is parallel to **case** where recursively a K numbered hole can be found. When the numbering is understood we simply write $C[\widetilde{\varphi};\widetilde{P};\widetilde{Q}]$.

In the following we define reduction semantics of ψ-calculi, in addition to the standard labelled transition semantics [1]. The two rules describe unicast and broadcast semantics. We identify agents up to structural congruence, that is, we also assume the rule such that two agents reduce if their congruent versions reduce. In the broadcast rule, if for some $a \in \widetilde{a}$, $a \in n(K)$, then $\widetilde{b} = \widetilde{a}$, otherwise $\widetilde{b} = \widetilde{a} \setminus n(N)$. To simplify the presentation we abbreviate $\prod \widetilde{(\!|\Psi|\!)}$ as $(\!|\hat{\Psi}|\!)$ and $\otimes_i \Psi_i$ as $\hat{\Psi}$. We prove that reductions correspond to silent and broadcast transitions.

$$\frac{N' = N[\widetilde{x} := \widetilde{L}] \text{ and } \hat{\Psi} \vdash M \dot{\leftrightarrow} M' \text{ and } \forall i.\hat{\Psi} \vdash \varphi_i}{(\nu\widetilde{a})(C[\widetilde{\varphi};\ \widetilde{R};\ M(\lambda\widetilde{x})N.P,\ \overline{M'}N'.Q] \mid (\!|\hat{\Psi}|\!)) \quad\to\quad (\nu\widetilde{a})(P[\widetilde{x} := \widetilde{L}] \mid Q \mid \prod\widetilde{R} \mid (\!|\hat{\Psi}|\!))}$$

$$\frac{\hat{\Psi} \vdash M \dot{\prec} K \text{ and } \forall i.\hat{\Psi} \vdash K \dot{\succ} M'_i \text{ and } N'_i[\widetilde{x}_i := \widetilde{L}_i] = N \text{ and } \forall j.\hat{\Psi} \vdash \varphi_j}{(\nu\widetilde{a})(C[\widetilde{\varphi};\ \widetilde{R};\ \overline{M}N.P, (M'\widetilde{(\lambda\widetilde{x})}N'.Q)] \mid (\!|\hat{\Psi}|\!)) \quad\to\quad (\nu\widetilde{b})(P \mid \prod_i Q_i[\widetilde{x}_i := \widetilde{L}_i] \mid \prod\widetilde{R} \mid (\!|\hat{\Psi}|\!))}$$

Theorem 3.1. *Let α be either a silent or broadcast output action. Then, $\mathbf{1} \rhd P \xrightarrow{\alpha} P'$ iff $P \to P'$*

Proof Sketch. The complicated direction is $\Longrightarrow$. One needs to prove similar results for the other actions, and then demonstrate that they in parallel have the right form. □

4 Translation of Broadcast Calculus to Broadcast ψ-Calculus

The semantics for the broadcast session calculus are given as an instance of the ψ-calculi with broadcast [2]. To achieve this effect we define a translation between the syntax of § 2 and a particular instance of the ψ-calculi. Operational semantics are then inherited by the ψ-calculi framework.

We fix the set of labels $\mathscr{L}$ and ranged over by $l, l_1, l_2 \ldots$. The following are the nominal sets

$$\begin{array}{lcl}
\mathbf{T} & = & \mathscr{N} \cup \{*\} \cup \{(n^p,k),(n^p,i),(n^p,k,\mathsf{u}),(n^p,l,k), n\cdot k \mid n,k \in \mathbf{T} \wedge i \in \mathbb{N} \wedge l \in \mathscr{L} \wedge p \in \{+,-\}\} \\
\mathbf{C} & = & \{t_1 \dot{\leftrightarrow} t_2, t_1 \dot{\prec} t_2, t_1 \dot{\succ} t_2 \mid t_1,t_2 \in \mathbf{T}\} \cup \{\mathsf{true}\} \\
\mathbf{A} & = & \mathbf{T} \to \mathbb{N}
\end{array}$$

We define the $\otimes$ operator (here defined as multiset union) and the $\vdash$ relation:

$$(f\otimes g)(n) = \begin{cases} f(n)+g(n) & \text{if } n \in dom(f) \cap dom(g) \\ f(n) & \text{if } n \in dom(f) \\ g(n) & \text{if } n \in dom(g) \\ \text{undefined} & \text{otherwise} \end{cases}
\quad\left|\quad
\begin{array}{lll}
\Psi & \vdash & (s^{p_1},k,\mathsf{u}) \dot{\leftrightarrow} (s^{p_2},j,\mathsf{u}) \text{ iff } \Psi(k) = \Psi(j) \\
\Psi & \vdash & (s^+,k) \dot{\prec} (s^+,i) \text{ iff } \Psi(k) = i \\
\Psi & \vdash & (s^+,i) \dot{\succ} (s^-,k) \text{ iff } \Psi(k) = i \\
\Psi & \vdash & \mathsf{true} \qquad \Psi \vdash a \dot{\leftrightarrow} a \in \mathscr{N}
\end{array}\right.$$

It can be easily checked that the definition is indeed a broadcast ψ-calculus. We write $\Sigma_{i\in I}P$ as a shorthand for **case** $\widetilde{\texttt{true}} : \widetilde{P}$, and $P+Q$ for **case** $\texttt{true} : P \,[]\, \texttt{true} : Q$

The translation is parameterised by ρ, which tracks the enumeration of session prefixes, represented by multisets of asserted names $\llparenthesis k \rrparenthesis$. The replication in $s^+?(x,u^i);P$ implements the iterative broadcast receive. We annotated the prefixes $s^+?(x,u^i);^b P$ and $\mu X^b.P \bowtie R$ with $b \in \{0,1\}$ to capture their translation as a two step (0 and 1) iterative process. The recovery process can be chosen in a non-deterministic way instead of a s^- prefix. Otherwise it is pushed in the continuation of the translation.

$$\begin{aligned}
&\llbracket \overline{a}s^-.P \bowtie R \rrbracket_\rho = (\nu k)(\overline{a}s^-.\llbracket P \bowtie R \rrbracket_{\rho\cup\{s^+:k\}}) \qquad \llbracket ax.P \bowtie R \rrbracket_\rho = (\nu k)(a(\lambda x)x.\llbracket P \bowtie R \rrbracket_{\rho\cup\{s^-:k\}})\\
&\llbracket s^+!\langle v\rangle;P \bowtie R \rrbracket_{\rho\cup\{s^+:k\}} = \overline{(s^+,k)}v.(\llbracket P \bowtie R \rrbracket_{\rho\cup\{s^+:k\}} \mid \llparenthesis k \rrparenthesis)\\
&\llbracket s^-!\langle v\rangle;P \bowtie R \rrbracket_{\rho\cup\{s^-:k\}} = \overline{(s^-,k,\mathsf{u})}v.(\llbracket P \bowtie R \rrbracket_{\rho\cup\{s^-:k\}} \mid \llparenthesis k \rrparenthesis) + \llbracket R \rrbracket_{\rho\cup\{s^-:k\}}\\
&\quad \llbracket s^+?(x,u);^0 P \bowtie R \rrbracket_{\rho\cup\{s^+:k\}} = \quad (\nu\, n)(\overline{n}u.\mathbf{0} \mid !(n(\lambda x)x.((s^+,k,\mathsf{u})(\lambda y)y.\overline{n}(x\cdot y).\mathbf{0})\\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad +\tau.(\llbracket P \bowtie R \rrbracket_{\rho\cup\{s^+:k\}} \mid \llparenthesis k \rrparenthesis)))\\
&\quad \llbracket s^+?(x,u);^1 P \bowtie R \rrbracket_{\rho\cup\{s^+:k\}} = \quad (\nu\, n)(((s^+,k,\mathsf{u})(\lambda y)y.\overline{n}(u\cdot y).\mathbf{0}) + \tau.(\llbracket P \bowtie R \rrbracket_{\rho\cup\{s^+:k\}}[x := u] \mid \llparenthesis k \rrparenthesis)\\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \mid !(n(\lambda x)x.((s^+,k,\mathsf{u})(\lambda y)y.\overline{n}(x\cdot y).\mathbf{0}) + \tau.(\llbracket P \bowtie R \rrbracket_{\rho\cup\{s^+:k\}} \mid \llparenthesis k \rrparenthesis)))\\
&\llbracket s^-?(x);P \bowtie R \rrbracket_{\rho\cup\{s^-:k\}} = (s^-,k)(\lambda x)x.(\llbracket P \bowtie R \rrbracket_{\rho\cup\{s^-:k\}} \mid \llparenthesis k \rrparenthesis) + \llbracket R \rrbracket_{\rho\cup\{s^-:k\}}\\
&\llbracket s^+ \oplus l;P \bowtie R \rrbracket_{\rho\cup\{s^-:k\}} = \overline{(s^+,l,k)}*.(\llbracket P \bowtie R \rrbracket_{\rho\cup\{s^+:k\}} \mid \llparenthesis k \rrparenthesis) \qquad \llbracket \llparenthesis k \rrparenthesis \rrbracket_{\rho\cup\{s^p:k\}} = \llparenthesis k \rrparenthesis\\
&\llbracket s^-\&\{l_i : P_i\}_{i\in I} \bowtie R \rrbracket_{\rho\cup\{s^-:k\}} = \Sigma_{i\in I}(s^-,l_i,k)(\lambda)*.(\llbracket P_i \bowtie R \rrbracket_{\rho\cup\{s^-:k\}} \mid \llparenthesis k \rrparenthesis) + \llbracket R \rrbracket_{\rho\cup\{s^-:k\}}\\
&\llbracket \mu X^0.P \bowtie R \rrbracket_\rho = (\nu\, n)(!(n(\lambda)*.\llbracket P \bowtie R \rrbracket_{\rho\cup\{X:n\}}) \mid \overline{n}*.\mathbf{0})\\
&\llbracket \mu X^1.P \bowtie R \rrbracket_\rho = (\nu\, n)(\llbracket P \bowtie R \rrbracket_{\rho\cup\{X:n\}} \mid !(n(\lambda)*.\llbracket P \bowtie R \rrbracket_{\rho\cup\{X:n\}}))\\
&\llbracket X \rrbracket_{\rho\cup\{X:n\}} = \overline{n}*.\mathbf{0} \quad \llbracket \mathbf{0} \rrbracket_\rho = \mathbf{0} \quad \llbracket \mathbf{0} \bowtie R \rrbracket_\rho = \mathbf{0} \quad \llbracket P \mid Q \rrbracket_\rho = \llbracket P \rrbracket_\rho \mid \llbracket Q \rrbracket_\rho \quad \llbracket (\nu\, n)P \rrbracket_\rho = (\nu n)\llbracket P \rrbracket_\rho
\end{aligned}$$

The encoding respects the following desirable properties.

Lemma 4.1 (Encoding Properties)**.** Let P be a session broadcast process.

1. $\llbracket P[x := v] \rrbracket = \llbracket P \rrbracket[x := v]$
2. $\llbracket P \rrbracket \to Q$ implies that for a session broadcast process $P', Q \equiv_\Psi \llbracket P' \rrbracket$.

5 Broadcast Session Types

Broadcast session types syntax is identical to classic binary session type syntax (cf. [7]), with the exception that we do not allow session channel delegation. We assume the duality relation as defined in [7]. Note that we do not need to carry the session prefix enumeration in the session type system or semantics. Session prefix enumeration is used operationaly only to avoid communication missmatch.

$$\begin{aligned}
S \quad &::= \quad !U;S \;\mid\; ?U;S \;\mid\; \oplus\{l_i : S_i\}_{i\in I} \;\mid\; \&\{l_i : S_i\}_{i\in I} \;\mid\; \texttt{end} \;\mid\; \mathsf{X} \;\mid\; \mu\mathsf{X}.S\\
U \quad &::= \quad \langle S\rangle \;\mid\; [U]
\end{aligned}$$

Typing judgements are: $\Gamma \vdash P$ read as P is typed under environment Γ, with

$$\Delta \;::=\; \emptyset \;\mid\; \Delta\cdot s^p : S \qquad\qquad \Gamma \;::=\; \emptyset \;\mid\; \Gamma\cdot a : \langle S\rangle \;\mid\; \Gamma\cdot s^p : S \;\mid\; \Gamma\cdot \mathsf{X} : \Delta$$

Δ environments map only session names to session types, while Γ maps shared names to shared types, session names to session types and process variables to Δ mappings.

The rules below define the broadcast session type system:

$$\Gamma\cdot n : U \vdash n : U \;\text{[Name]} \qquad \frac{\Gamma \vdash P \quad s^p \notin \texttt{fn}(P)}{\Gamma\cdot s^p : \texttt{end} \vdash P}\;\text{[Weak]} \qquad \frac{s \notin \texttt{dom}(\Gamma)}{\Gamma \vdash \mathbf{0}\;\text{[Inact]}} \qquad \frac{\Gamma \vdash R \quad s^p \notin \texttt{dom}(\Gamma)}{\Gamma \vdash \mathbf{0} \bowtie R}\;\text{[Recov]}$$

$$\frac{\Gamma \vdash a : \langle S \rangle \quad \Gamma \vdash s^{+} : S \quad \Gamma \vdash P}{\Gamma \cdot s^{-} : \overline{S} \vdash \overline{a} s^{-}.P}\;[\text{BInit}] \qquad \frac{\Gamma \vdash a : \langle S \rangle \quad \Gamma \cdot x : \overline{S} \vdash P}{\Gamma \vdash ax.P}\;[\text{BAcc}]$$

$$\frac{\Gamma \cdot s^{+} : S \vdash P \bowtie R \quad \Gamma \vdash v : \langle S' \rangle}{\Gamma \cdot s^{+} :!\langle S' \rangle ; S \vdash s^{+}!\langle v \rangle ; P \bowtie R}\;[\text{BSend}] \qquad \frac{\Gamma \cdot s^{-} : S \vdash P \bowtie R \quad \Gamma \vdash v : \langle S' \rangle \quad s^{-} \notin \mathtt{dom}(\Gamma)}{\Gamma \cdot s^{-} :!\langle S' \rangle ; S \vdash s^{-}!\langle v \rangle ; P \bowtie R}\;[\text{USend}]$$

$$\frac{\Gamma \cdot s^{+} : S \cdot x : \langle S' \rangle \vdash P \bowtie R \quad \Gamma \vdash u : [\langle S' \rangle]}{\Gamma \cdot s^{+} :?\langle S' \rangle ; S \vdash s^{+}?(x,u);^{b} P \bowtie R}[\text{URcv}] \qquad \frac{\Gamma \cdot s^{-} : S \cdot x : \langle S' \rangle \vdash P \bowtie R \quad s^{-} \notin \mathtt{dom}(\Gamma)}{\Gamma \cdot s^{-} :?\langle S' \rangle ; S \vdash s^{-}?(x) ; P \bowtie R}\;[\text{BRcv}]$$

$$\frac{\Gamma \cdot s^{+} : S_k \vdash P \bowtie R \quad k \in I}{\Gamma \cdot s^{+} : \oplus\{l_i : S_i\}_{i \in I} \vdash s^{+} \oplus l_k ; P \bowtie R}\;[\text{Sel}] \qquad \frac{\Gamma \cdot s^{-} : S_i \vdash P_i \bowtie R \quad s^{-} \notin \mathtt{dom}(\Gamma)}{\Gamma \cdot s^{-} : \&\{l_i : S_i\}_{i \in I} \vdash s^{-}\&\{l_i : P_i\}_{i \in I} \bowtie R}\;[\text{Bra}]$$

$$\frac{\Gamma_1 \vdash P_1 \quad \Gamma_2 \vdash P_2 \quad s^{+} \notin \mathtt{dom}(\Gamma_1) \cap \mathtt{dom}(\Gamma_2)}{\Gamma_1 \cup \Gamma_2 \vdash P_1 \mid P_2}\;[\text{Par}] \qquad \frac{\Gamma \cdot s^{+} : S \cdot \{s^{-} : \overline{S_i}\}_{i \in I} \vdash P \quad S = S_i}{\Gamma \vdash (\nu\ s)P}\;[\text{SRes}]$$

$$\frac{\Gamma \cdot a : \langle S \rangle \vdash P}{\Gamma \vdash (\nu\ a)P}\;[\text{ShRes}] \qquad \frac{\Gamma \cup \Delta \cdot X : \Delta \vdash P \quad s^{p} \notin \mathtt{dom}(\Gamma)}{\Gamma \cup \Delta \vdash \mu X^{b}.P}\;[\text{Rec}] \qquad \Gamma \cup \Delta \cdot X : \Delta \vdash X\ [\text{RVar}]$$

Rule [Recov] types the recovery process. We expect no free session names in a recover process. Rules [BInit], [BAcc], [BSend], [Usend], [BRcv], [BRcv], [Sel] and [Bra] type prefixes in the standard way, i.e. check for object and the subject type match. Rule [URcv] types both binary instances of the unicast receive prefix with the same type. We require that the recovery process is carried and typed inductively in the structure of a process. A recovery process must not (re)use any session endpoints ([Recov]). Also we require the s^{-} to be the only one in Γ. Multiple s^{-} endpoints are collected using the [Par] rule. The [Par] rule expects that there is no duplicate s^{+} endpoint present inside a process. When restricting a session name we check endpoint s^{+} and the set of endpoints s^{-} to have dual types. The rest of the rules are standard.

5.1 Soundness and Safety

We use the standard notion of a context $\mathscr{C}$ on session types S with a single hole denoted as $[]$. We write $\mathscr{C}[S]$ for filling a hole in C with the type S. We define the set of non-live sessions in a context as $d(\Gamma) = \{s^{-} : S \mid s^{+} : S' \in \Gamma \text{ and } \overline{S} = C[S'] \text{ with } C \neq []\}$ and live $l(\Gamma) = \Gamma \setminus d(\Gamma)$. We say that Γ is well typed iff $\forall s^{+} : S \in l(\Gamma)$ then $\{s^{-} : \overline{S_i}\}_{i \in I} \subset l(\Gamma)$ with $S = S_i$ or $S = ?U; S_i$.

Theorem 5.1 (Subject Congruence)**.** If $\Gamma \vdash P$ with Γ well typed and $P \equiv P'$ then $\Gamma \vdash P'$.

Theorem 5.2 (Subject Reduction)**.** If $\Gamma \vdash P$ with Γ well typed, $dom(\rho) \subseteq dom(\Gamma)$ and $[\![P]\!]_{\rho} \to Q$, then there is P' such that $[\![P']\!]_{\rho} \equiv_{\Psi} Q$, $\Gamma' \vdash P'$ and Γ' well typed with either $\Gamma' = d(\Gamma) \cup l(\Gamma')$ or $\Gamma' = d(\Gamma) \setminus \{s^{-} : S\} \cup l(\Gamma')$ or $\Gamma' = d(\Gamma) \cup \{s^{-} : S\} \cup l(\Gamma')$.

Definition 5.1 (Error Process)**.** Let s-prefix processes to have the following form:

1. $s^{+}!\langle v \rangle ; P$ 2. $s^{+} \oplus l ; P$ 3. $s^{+}?(x) ; P$ 4. $\prod_{i \in I} s^{-}?(x) ; P_i \mid \prod_{j \in J} C_j[s^{-}?(x) ; P_j]$
5. $\prod_{i \in I} s^{-}!\langle v_i \rangle ; P_i \mid \prod_{k \in K} P_k \mid \prod_{j \in J} C_j[s^{-}?(x) ; P_j]$
where $\prod_{i \in I} P_i \mid \prod_{k \in K} P_k \mid \prod_{j \in J} C_j[s^{-}?(x) ; P_j]$ forms an s-redex.
6. $\prod_{i \in I} s^{-}\&\{l_k : P_k\}_{k \in K_i} \mid \prod_{j \in J} C_j[s^{-}\&\{l_k : P_k\}_{k \in K_j}]$

with $C_j[]$ being a context that contains s^{-} prefixes.

A valid s-redex is a parallel composition of either s-prefixes 1 and 4, s-prefixes 2 and 6, or s-prefixes 3 and 5. Every other combination of s-prefixes is invalid. An error process is a process of the form $P \equiv (\nu\ \tilde{n})(R \mid Q)$ where R is an invalid s-redex and Q does not contain any other s-prefixes.

Theorem 5.3 (Type Safety)**.** A well typed process will never reduce into an error process.

Proof. The proof is a direct consequence of the Subject Reduction Theorem (5.2) since error process are not well typed. □

6 Conclusion

We have defined a system of session types for a calculus based on unreliable broadcast communication. This is the first time that session types have been generalised beyond reliable point-to-point communication. We defined the operational semantics of our calculus by translation into an instantiation of broadcast ψ-calculi, and proved subject reduction and safety results. The use of the ψ-calculi framework means that we can try to use its general theory of bisimulation for future work on reasoning about session-typed broadcasting systems. The definition of a session typing system is also a new direction for the ψ-calculi framework.

Acknowledgements Kouzapas and Gay are supported by the UK EPSRC project "From Data Types to Session Types: A Basis for Concurrency and Distribution" (EP/K034413/1). This research was supported by a Short-Term Scientific Mission grant from COST Action IC1201 (Behavioural Types for Reliable Large-Scale Software Systems).

References

[1] Jesper Bengtson, Magnus Johansson, Joachim Parrow & Björn Victor (2009): *Psi-calculi: Mobile Processes, Nominal Data, and Logic*. In: *LICS*, pp. 39–48, doi:10.1109/LICS.2009.20.

[2] Johannes Borgström, Shuqin Huang, Magnus Johansson, Palle Raabjerg, Björn Victor, Johannes Åman Pohjola & Joachim Parrow (2011): *Broadcast Psi-calculi with an Application to Wireless Protocols.* In Gilles Barthe, Alberto Pardo & Gerardo Schneider, editors: *SEFM*, *Lecture Notes in Computer Science* 7041, Springer, pp. 74–89, doi:10.1007/978-3-642-24690-6_7.

[3] Sara Capecchi, Elena Giachino & Nobuko Yoshida (2014): *Global Escape in Multiparty Sessions*. *Mathematical Structures in Computer Science*. To appear.

[4] Marco Carbone, Kohei Honda & Nobuko Yoshida (2008): *Structured Interactional Exceptions in Session Types*. In: *CONCUR*, *LNCS* 5201, Springer, pp. 402–417, doi:10.1007/978-3-540-85361-9_32.

[5] Kohei Honda, Vasco T. Vasconcelos & Makoto Kubo (1998): *Language Primitives and Type Disciplines for Structured Communication-based Programming*. In: *ESOP'98*, *LNCS* 1381, Springer, pp. 22–138, doi:10.1007/BFb0053567.

[6] Kohei Honda, Nobuko Yoshida & Marco Carbone (2008): *Multiparty Asynchronous Session Types*. In: *POPL'08*, ACM, pp. 273–284, doi:10.1145/1328897.1328472.

[7] Nobuko Yoshida & Vasco Thudichum Vasconcelos (2007): *Language Primitives and Type Discipline for Structured Communication-Based Programming Revisited: Two Systems for Higher-Order Session Communication*. *Electr. Notes Theor. Comput. Sci.* 171(4), pp. 73–93, doi:10.1016/j.entcs.2007.02.056.

Paper III

A SORTED SEMANTIC FRAMEWORK FOR APPLIED PROCESS CALCULI

JOHANNES BORGSTRÖM, RAMŪNAS GUTKOVAS, JOACHIM PARROW, BJÖRN VICTOR, AND JOHANNES ÅMAN POHJOLA

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 erratic choice. Finally, we prove standard congruence and structural properties of bisimulation; substantial parts of the proof 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 [BJPV11], which provides machine-checked proofs that important meta-theoretical properties, such as compositionality of bisimulation, hold in all instances of the framework. We claim that the theoretical development is more robust than that of other calculi of comparable complexity, since we use a structural operational semantics given by a single inductive definition, and since we have checked most results in the theorem prover Nominal Isabelle [Urb08].

LOGICAL METHODS
IN COMPUTER SCIENCE

DOI:10.2168/LMCS-???

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 correspond to standard calculi, and use several different notions of computation. We define strong criteria for a psi-calculus to *represent* another process calculus, meaning that they are for all practical purposes one and the same. Representation is stronger than the standard *encoding* correspondences e.g. by Gorla[Gor10], which define criteria for one language to encode the behaviour of another. The representations that we provide of other calculi advance our previous work, where we had to resort to nontrivial encodings with unclear formal correspondence to the standard calculi.

1.1. **Background: 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. Full definitions can be found in the references above, and for a reader not acquainted with our work we recommend the first few sections of [BJPV11] for an introduction.

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 $M \leftrightarrow K$. 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 $\dot{\leftrightarrow}$. 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 $\Psi \vdash M \dot{\leftrightarrow} K$, 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 $\Psi \rhd \textbf{if}\ \varphi\ \textbf{then}\ P \xrightarrow{\alpha} P'$ 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* [Pit03]. 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. In addition, we impose straight-forward requirements on the combination of assertions, on channel equivalence, and on substitution. Our requirements are quite general, and therefore our framework accommodates a wide variety of applied process calculi.

1.2. **Extension: Generalized pattern matching.** In our original definition of psi-calculi ([BJPV11], 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 $\textsc{match}(N, \widetilde{x}, X)$ 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: `enc` of arity three and `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 **??** and **??** in Section 5.

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 $\widetilde{x} \subseteq \mathrm{n}(N)$, 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 $\widetilde{x} \subseteq \mathrm{n}(M)$. The reason is explained at length in [BJPV11]; 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.

In this paper, we address these three issues by introducing explicit notions of patterns, pattern variables and matching. This allows us to control precisely which parts of messages can be bound by pattern-matching and how messages can be deconstructed. It also ensures that well-formedness is preserved by transitions and admits computations such as $\mathtt{dec}(\mathtt{enc}(M, N, K), K) \to M$ in substitutions.

1.3. **Extension: Sorting.** Applied process calculi often make use of a sort system. The applied pi-calculus [AF01] 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 [HJ06] 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. Such anomalies disappear when introducing an appropriate sort system; cf. Section 4.1.

1.4. **Related work.** Pattern-matching is in common use in functional programming languages. Scala admits pattern-matching of objects [EOW07] using a method `unapply` that turns the receiving object into a matchable value (e.g. a tuple). F# admits the definition of pattern cases independently of the type that they should match [SNM07], facilitating interaction with third-party and foreign-language code. Turning to message-passing systems, LINDA [Gel85] uses pattern-matching when receiving from a tuple space. Similarly, in Erlang, message reception from a mailbox is guarded by a pattern.

These notions of patterns, with or without computation, are easily supported by the MATCH construct. However, the standard first-match policy needs to be accomodated by extending the pattern language, as is usual for core calculi [Kri09].

Pattern matching in process calculi. The pattern-matching spi-calculus [HJ06] limits which variables may be binding in a pattern in order to match encrypted messages without binding unknown keys (cf. Example **??**). The Kell calculus [SS05] also uses pattern languages equipped with a match function. However, in the Kell calculus the channels are single names and appear as part of the pattern in the input prefix, patterns may match multiple communications simultaneously (à la join calculus), and first-order pattern variables only match names (not composite messages) making forwarding and partial decomposition impossible.

The applied pi-calculus [AF01] models deterministic computation by using for data language a term algebra modulo an equational logic. ProVerif [Bla11] 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. Section 5.1).

Other tools for process calculi extended with datatypes include mCRL2 [CGK$^+$13] for ACP, which allows higher order sorted term algebras and equational logic, and PAT3 [LSD11] which includes a CSP♯ [SLDC09] module where actions built over types like booleans and integers are extended with C♯-like programs. In all these cases, the pattern matching is defined by substitution in the usual way.

A comparison of expressiveness to calculi with non-binary (e.g., join-calculus [FG96]) or bidirectional (e.g., dyadic interaction terms [Hon93] or the concurrent pattern calculus [GWGJ10]) communication primitives would be interesting. We here inherit positive results from the pi calculus, such as the encoding of the join-calculus.

Sort systems for mobile processes. Sorts for the pi-calculus were first described by Milner [Mil93], and were developed in order to remove nonsensical processes using polyadic communication, similar to the motivation for the present work.

In contrast, Hüttel's dependently typed psi-calculi [Hüt11] is intended for a more fine-grained control of the behaviour of processes. Typed psi-calculi are capable of capturing a wide range of earlier sort systems for pi-like calculi formulated as instances of psi-calculi. However, 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 can be seen as a foundation for typed psi-calculi: we give a formal account of the separation between variables and names used in typed psi-calculi, and substantiate that Hüttel's claim that "the set of well-[sorted] terms is closed under well-[sorted] substitutions, which suffices" does not cause problems for the meta-theory of the language. Typed psi-caluli are also less general than sorted psi-calculi in some ways: the term language of typed psi-calculi is required to be a free term algebra (without name binders); it uses only the standard notions of substitution and matching, and does not admit any computation on terms. Furthermore, we prove meta-theoretical results including congruence results and structural equivalence laws for well-sorted bisimulation, and the preservation of well-sortedness under structural equivalence; no such results exist for typed psi-calculi.

The state-of-the art report [HV13] of WG1 of the BETTY project (EU COST Action IC1201) is a comprehensive guide to behavioural types for process calculi.

Fournet et al. [FGM05] add type-checking for a general authentication logic to a process calculus with destructor matching; there the authentication logic is only used to specify program correctness, and does not influence the operational semantics in any way.

1.5. **Results and outline.** In Section 2 we define psi-calculi with the above extensions and prove preservation of well-formedness. In Section 3 we prove the usual algebraic properties of bisimilarity. The proof is in two steps: a machine-checked proof for single-sorted calculi, followed by a manual proof based on the translation of a multi-sorted psi calculus instance to a corresponding single-sorted instance. We demonstrate the expressiveness of our generalization in Section 4 where we directly represent standard calculi, and in Section 5 where we give examples of 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 and elements of the data type 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 $\widetilde{x}$ also stands for the set of names $\{x_1, \ldots, x_n\}$ if used where a set is expected. We let π range over permutations of tuples of names: $\pi \cdot \widetilde{x}$ is a tuple of names of the same length as $\widetilde{x}$, containing the same names with the same multiplicities.

A sorted *nominal set* [Pit03, GP01] 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 $(a\ b) \cdot (f(T)) = f((a\ b) \cdot T)$ always holds; a relation $\mathcal{R}$ is equivariant if $x\ \mathcal{R}\ y$ implies that $(a\ b) \cdot x\ \mathcal{R}\ (a\ b) \cdot y$ holds; and a constant symbol C is equivariant if $(a\ b) \cdot C = C$. A *nominal data type* is a nominal set together with some equivariant 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 [BJPV11], which are given by three nominal datatypes (data terms, conditions and assertions) as discussed in the introduction.

Definition 2.1 (Original psi-calculus parameters)**.** The psi-calculus parameters from the original psi-calculus are the following nominal data types: (data) terms $M, N \in \mathbf{T}$, conditions $\varphi \in \mathbf{C}$, and assertions $\Psi \in \mathbf{A}$; equipped with the following four equivariant operators: channel equivalence $\dot{\leftrightarrow} : \mathbf{T} \times \mathbf{T} \to \mathbf{C}$, assertion composition $\otimes : \mathbf{A} \times \mathbf{A} \to \mathbf{A}$, the unit assertion $\mathbf{1} \in \mathbf{A}$, and the entailment relation $\vdash \subseteq \mathbf{A} \times \mathbf{C}$.

The binary functions $\dot{\leftrightarrow}$ and $\otimes$ and the relation $\vdash$ above will be used in infix form. Two assertions are said to be 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 modulo entailment, 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 [BJPV11].

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 function VARS which yields the possible combinations of binding names in the pattern, and a pattern-matching function MATCH, which is used when the input takes place. Intuitively, an input pattern $(\lambda\widetilde{x})X$ matches a message N if there are $\widetilde{L} \in \textsc{match}(N, \widetilde{x}, X)$; the receiving agent then continues after substituting $\widetilde{L}$ for $\widetilde{x}$. If $\textsc{match}(N, \widetilde{x}, X) = \emptyset$ then $(\lambda\widetilde{x})X$ does not match N; if $|\textsc{match}(N, \widetilde{x}, X)| > 1$ then one of the matches will be non-deterministically chosen. Below, we use "variable" for names that can be bound in a pattern.

Definition 2.2 (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

$$\begin{array}{rcll} \textsc{match} & : & \mathbf{T} \times \mathcal{N}^* \times \mathbf{X} \to \mathcal{P}_{\text{fin}}(\mathbf{T}^*) & \text{Pattern matching} \\ \textsc{vars} & : & \mathbf{X} \to \mathcal{P}_{\text{fin}}(\mathcal{P}_{\text{fin}}(\mathcal{N})) & \text{Pattern variables} \end{array}$$

The VARS operator gives the possible (finite) 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 $\textsc{vars}(\langle x, y\rangle) = \{\{x, y\}\}$. 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. Another use for VARS is to exclude the binding of terms in certain positions, such as the keys of cryptographic messages (cf. Example **??**).

Requisites on VARS and MATCH are given below in Definition 2.5. 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 2.3 (Psi-calculus parameters for sorting)**.** The psi-calculus parameters for sorting include the sorting function $\textsc{sort} : \mathcal{N} \uplus \mathbf{T} \uplus \mathbf{X} \to \mathcal{S}$, and the four compatibility predicates

$$\begin{array}{rcll} \underline{\propto} & \subseteq & \mathcal{S} \times \mathcal{S} & \text{can be used to receive,} \\ \overline{\propto} & \subseteq & \mathcal{S} \times \mathcal{S} & \text{can be used to send,} \\ \prec & \subseteq & \mathcal{S} \times \mathcal{S} & \text{can be substituted by,} \\ \mathcal{S}_\nu & \subseteq & \mathcal{S} & \text{can be bound by name restriction.} \end{array}$$

The SORT operator gives the sort of a name, term or pattern; on names we require that $\textsc{sort}(a) = s$ 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 $\textsc{sort}(a) \in \mathcal{S}_\nu$. If $\textsc{sort}(a) \prec \textsc{sort}(M)$ 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 2.4 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 $\forall t \in \mathcal{S}.(s_2 \underline{\propto} t \Rightarrow s_1 \underline{\propto} t) \wedge (s_2 \overline{\propto} t \Rightarrow s_1 \overline{\propto} t) \wedge (t \underline{\propto} s_2 \Rightarrow t \underline{\propto} s_1) \wedge (t \overline{\propto} s_2 \Rightarrow t \overline{\propto} s_1)$. This relation compares the 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 2.4 (Requisites on substitution)**.** If $\widetilde{a}$ is a sequence of distinct names and $\widetilde{N}$ is an equally long sequence of terms such that $\textsc{sort}(a_i) \prec \textsc{sort}(N_i)$ 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 $T[\widetilde{a} := \widetilde{N}] = ((\widetilde{a}\ \widetilde{b}) \cdot T)[\widetilde{b} := \widetilde{N}]$ (alpha-renaming of substituted variables). For terms we additionally require that $\textsc{sort}(M\sigma) \leq \textsc{sort}(M)$.

For substitution on patterns $X \in \mathbf{X}$, we require that $X\sigma \in \mathbf{X}$, and if $\widetilde{x} \in \textsc{vars}(X)$ and $\widetilde{x} \# \sigma$ then $\textsc{sort}(X\sigma) \leq \textsc{sort}(X)$ and $\widetilde{x} \in \textsc{vars}(X\sigma)$ and alpha-renaming of substituted variables (as above) holds for σ and X.

Intuitively, the requirements on substitutions on patterns ensure that a substitution on a pattern with binders $((\lambda\widetilde{x})X)\sigma$ with $\widetilde{x} \in \textsc{vars}(X)$ and $\widetilde{x} \# \sigma$ yields a pattern $(\lambda\widetilde{x})Y$ with $\widetilde{x} \in \textsc{vars}(Y)$. As an example, consider the pair patterns discussed above with $\mathbf{X} = \{\langle x, y\rangle : x \neq y\}$ and $\textsc{vars}(\langle x, y\rangle) = \{\{x, y\}\}$. We can let $\langle x, y\rangle\sigma = \langle x, y\rangle$ when $x, y \# \sigma$. Since $\textsc{vars}(\langle x, y\rangle) = \{\{x, y\}\}$ 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.

In the original psi-calculi there is no requirement that substitutions on terms preserve names used as pattern variables (i.e., $\mathrm{n}(N\sigma) \supseteq \mathrm{n}(N) \setminus \mathrm{n}(\sigma)$). For this reason, the original psi semantics does not always preserve the well-formedness of agents (an input prefix $\underline{M}(\lambda\widetilde{x})N \,.\, P$ is well-formed when $\widetilde{x} \subseteq \mathrm{n}(N)$), although this is assumed by the operational semantics [BJPV11]. In pattern-matching psi-calculi, the operational semantics does preserve well-formedness, as shown below in Theorem 2.11.)

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 2.5 (Requisites on pattern matching)**.** For the function $\textsc{match}$ we require that if $\widetilde{x} \in \textsc{vars}(X)$ are distinct and $\widetilde{N} \in \textsc{match}(M, \widetilde{x}, X)$ then it must hold that $[\widetilde{x} := \widetilde{N}]$ is a substitution, that $\mathrm{n}(\widetilde{N}) \subseteq \mathrm{n}(M) \cup (\mathrm{n}(X) \setminus \widetilde{x})$, and that for all name swappings $(\widetilde{x}\ \widetilde{y})$ with $\widetilde{y} \# X$ we have $\widetilde{N} \in \textsc{match}(M, \widetilde{y}, (\widetilde{x}\ \widetilde{y}) \cdot X)$ (alpha-renaming of matching).

In many process calculi, and also in the symbolic semantics of psi [JVP12], 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 2.6. Given values for the other requisites, we can take $\mathbf{X} = \mathcal{N}$ with $\textsc{vars}(a) = \{a\}$, meaning that the pattern variable must always occur bound, and $\textsc{match}(M, a, a) = \{M\}$ if $\textsc{sort}(a) \prec \textsc{sort}(M)$. On patterns we define substitution as $a\sigma = a$ when $a\#\sigma$.

When all substitutions on terms preserve names, we can recover the pattern matching of the original psi-calculi. Such psi-calculi also enjoy well-formedness preservation (Theorem 2.11).

Theorem 2.7. *Suppose* $(\mathbf{T}, \mathbf{C}, \mathbf{A})$ *is an original psi-calculus* [BJPV11] *where* $\mathrm{n}(N\sigma) \supseteq \mathrm{n}(N) \setminus \mathrm{n}(\sigma)$ *for all* N, σ. *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.*

Proof. Straightforward; this result has been checked in Isabelle. □

2.5. **Agents.**

Definition 2.8 (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
$\llparenthesis \Psi \rrparenthesis$	Assertion

In the Input all names in $\widetilde{x}$ bind their occurrences in both X and P, and in the Restriction a binds in P. Substitution on agents is defined inductively on their structure, using the substitution function of each datatype based on syntactic position, avoiding name capture.

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 $\llparenthesis \Psi \rrparenthesis$ contributes Ψ to its environment. We often write **if** φ **then** P for **case** $\varphi : P$, and nothing or $\mathbf{0}$ for the empty case statement **case**.

In comparison to [BJPV11] we additionally 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 2.9. An assertion is *guarded* if it is a subterm of an Input or Output. An agent is *well-formed* if, for all its subterms,

(1) in a replication $!P$ there are no unguarded assertions in P; and
(2) in **case** $\varphi_1 : P_1 \;[\!]\; \cdots \;[\!]\; \varphi_n : P_n$ there is no unguarded assertion in any P_i; and
(3) in an Output $\overline{M}\ N.P$ we require that $\textsc{sort}(M) \mathrel{\overline{\propto}} \textsc{sort}(N)$; and
(4) in an Input $\underline{M}(\lambda\widetilde{x})X.P$ we require that
  (a) $\widetilde{x} \in \textsc{vars}(X)$ is a tuple of distinct names and
  (b) $\textsc{sort}(M) \mathrel{\underline{\propto}} \textsc{sort}(X)$; and
(5) in a Restriction $(\nu a)P$ we require that $\textsc{sort}(a) \in \mathcal{S}_\nu$.

Requirements 3, 4b and 5 are new for sorted psi-calculi.

2.6. **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 2.10 (Frames and Transitions)**.** The *frame* $\mathcal{F}(P)$ *of an agent* P is defined inductively as follows:

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

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, defined in Table 1, is the same as for the original psi-calculi, except for the use of $\textsc{match}$ in rule $\textsc{In}$. We identify alpha-equivalent agents and transitions (see [BJPV11] 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.

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

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

$$\textsc{Par}\ \frac{\Psi_Q \otimes \Psi \rhd P \xrightarrow{\alpha} P'}{\Psi \rhd P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \mathrm{bn}(\alpha)\#Q \qquad \textsc{Case}\ \frac{\Psi \rhd P_i \xrightarrow{\alpha} P' \qquad \Psi \vdash \varphi_i}{\Psi \rhd \textbf{case}\ \widetilde{\varphi} : \widetilde{P} \xrightarrow{\alpha} P'}$$

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

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

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.

Theorem 2.11 (Preservation of well-formedness)**.** *If P is well-formed, then*

(1) *$P\sigma$ is well-formed; and*

(2) *if $\Psi \rhd P \xrightarrow{\alpha} P'$ then P' is well-formed.*

Proof. The first part is by induction on P. The output prefix case uses the sort preservation property of substitution on terms (Definition 2.4). The interesting case is input prefix $\underline{M}(\lambda\widetilde{x})X.Q$: assume that Q is well-formed, that $\widetilde{x} \in \textsc{vars}(X)$, that $\textsc{sort}(M) \underline{\propto} \textsc{sort}(X)$ and that $\widetilde{x}\#\sigma$. By induction $Q\sigma$ is well-formed. By sort preservation we get $\textsc{sort}(M\sigma) \leq \textsc{sort}(M)$, so $\textsc{sort}(M\sigma) \underline{\propto} \textsc{sort}(X)$. By preservation of patterns by non-capturing substitutions we have that $\widetilde{x} \in \textsc{vars}(X\sigma)$ and $\textsc{sort}(X\sigma) \leq \textsc{sort}(X)$, so $\textsc{sort}(M\sigma) \underline{\propto} \textsc{sort}(X\sigma)$.

The second part is by induction on the transition rules, using part 1 in the In rule. □

3. Meta-theory

As usual, the labelled operational semantics gives rise to notions of labelled bisimilarity. Similarly to the applied pi-calculus [AF01], the standard definition of bisimilarity needs to be adapted to take assertions into account. In this section, we show that both strong and weak bisimilarity satisfy the expected structural congruence laws and the standard congruence properties of name-passing process calculi. We first prove these results for calculi with a single sort (Theorem 3.12) supported by Nominal Isabelle, and then extend the result to all sorted psi-caluli (Theorem 3.16) by a manual proof. We start by recollecting the required definitions, beginning with the definition of strong labelled bisimulation on well-formed agents by Bengtson et al. [BJPV11], to which we refer for examples and more intuitions.

Definition 3.1 (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: for all Ψ' it holds that $\mathcal{R}(\Psi \otimes \Psi', P, Q)$.
(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* $P \mathrel{\dot{\sim}_\Psi} Q$ to mean that there is a bisimulation $\mathcal{R}$ such that $\mathcal{R}(\Psi, P, Q)$, and write $\dot{\sim}$ for $\dot{\sim}_{\mathbf{1}}$.

Above, (1) corresponds to the capability of a parallel observer to test the truth of a condition using **case**, while (3) models an observer taking a step and adding a new assertion Ψ' to the current environment.

We close strong bisimulation under substitutions to obtain a congruence in the usual way:

Definition 3.2 (Strong bisimulation congruence)**.** $P \sim_\Psi Q$ means that 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 $P \sim_{\mathbf{1}} Q$.

To illustrate the definitions of bisimulation and bisimulation congruence, we here prove a result about the **case** statement, to be used in Section 4.

Lemma 3.3 (Flatten Case)**.** *Suppose that there exists a condition $\top \in \mathbf{C}$ such that $\Psi \vdash \top\widetilde{\sigma}$ for all Ψ and substitution sequences $\widetilde{\sigma}$. 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. We let $\mathcal{I} := \bigcup_{\Psi,P}\{(\Psi, P, P)\}$ be the identity relation, and

$$\mathcal{S} := \bigcup_{\Psi,\widetilde{P},\widetilde{Q},\widetilde{\phi},\widetilde{\varphi}} \{(\Psi, \mathbf{case}\ \varphi_\top : (\mathbf{case}\ \widetilde{\varphi} : \widetilde{P})\ [\!]\ \widetilde{\phi} : \widetilde{Q}, \mathbf{case}\ \varphi_\top : \mathbf{case}\ \widetilde{\varphi} : \widetilde{P}\ [\!]\ \widetilde{\phi} : \widetilde{Q}) : \varphi_\top \in \mathbf{C} \wedge \forall \Psi' \in \mathbf{A}.\ \Psi' \vdash \varphi_\top\}.$$

We prove that $\mathcal{T} := \mathcal{S} \cup \mathcal{S}^{-1} \cup \mathcal{I}$ is a bisimulation, where $\mathcal{S}^{-1} := \{(\Psi, Q, P) : (\Psi, P, Q) \in \mathcal{S}\}$. Then, $\mathcal{T}(\mathbf{1}, R\widetilde{\sigma}, R'\widetilde{\sigma})$ for all $\widetilde{\sigma}$, so $R \sim R'$ by the definition of $\sim$. The proof that $\mathcal{T}$ is a bisimulation is straightforward:

Static equivalence: The frame of a **case** agent is always $\mathbf{1}$, hence static equivalence follows by reflexivity of $\simeq$.

Symmetry: Follows by definition of $\mathcal{T}$.

Extension with arbitrary assertion: Trivial by the choice of candidate relation, since the Ψ in $\mathcal{S}$ and $\mathcal{I}$ are universally quantified.

Simulation: Trivially, any process P simulates itself. Fix $(\Psi, R, R') \in \mathcal{S}$, such that $R = \mathbf{case}\ \varphi_\top : (\mathbf{case}\ \widetilde{\varphi} : \widetilde{P})\ [\!]\ \widetilde{\phi} : \widetilde{Q}$ and $R' = \mathbf{case}\ \widetilde{\varphi} : \widetilde{P}\ [\!]\ \widetilde{\phi} : \widetilde{Q}$. Here $\Psi \vdash \varphi_\top$ follows by definition of $\mathcal{S}$. Since $\mathcal{T}$ includes both $\mathcal{S}$ and $\mathcal{S}^{-1}$, we must follow transitions from both R and R'.

- A transition from R via P_i can be derived as follows:

$$\textsc{Case}\ \dfrac{\textsc{Case}\ \dfrac{\Psi \rhd P_i \xrightarrow{\alpha} P_i' \qquad \Psi \vdash \varphi_i}{\Psi \rhd \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \xrightarrow{\alpha} P_i'} \qquad \Psi \vdash \varphi_\top}{\Psi \rhd \mathbf{case}\ \varphi_\top : (\mathbf{case}\ \widetilde{\varphi} : \widetilde{P})\ [\!]\ \widetilde{\phi} : \widetilde{Q} \xrightarrow{\alpha} P_i'}$$

  Then R' can simulate this with the following derivation:

$$\textsc{Case}\ \dfrac{\Psi \rhd P_i \xrightarrow{\alpha} P_i' \qquad \Psi \vdash \varphi_i}{\Psi \rhd \mathbf{case}\ \widetilde{\varphi} : \widetilde{P}\ [\!]\ \widetilde{\phi} : \widetilde{Q} \xrightarrow{\alpha} P_i'}$$

  By reflexivity of $\dot{\sim}_\Psi$, we get that $P_i' \dot{\sim}_\Psi P_i'$.
- A transition from R' via Q_i can be derived as follows:

$$\textsc{Case}\ \dfrac{\Psi \rhd Q_i \xrightarrow{\alpha} Q_i' \qquad \Psi \vdash \phi_i}{\Psi \rhd \mathbf{case}\ \widetilde{\varphi} : \widetilde{P}\ [\!]\ \widetilde{\phi} : \widetilde{Q} \xrightarrow{\alpha} Q_i'}$$

  The process R can simulate this with the following derivation:

$$\textsc{Case}\ \dfrac{\Psi \rhd Q_i \xrightarrow{\alpha} Q_i' \qquad \Psi \vdash \phi_i}{\Psi \rhd \mathbf{case}\ \varphi_\top : (\mathbf{case}\ \widetilde{\varphi} : \widetilde{P})\ [\!]\ \widetilde{\phi} : \widetilde{Q} \xrightarrow{\alpha} Q_i'}$$

  By reflexivity of $\dot{\sim}_\Psi$ we get $Q_i' \dot{\sim}_\Psi Q_i'$.
- Symmetrically, R' can simulate transitions derived from R via Q_i, and R can simulate transitions derived from R' via P_i. □

Psi-calculi are also equipped with a notion of weak bisimilarity ($\dot{\approx}$) where τ-transitions cannot be observed, introduced by Bengtson et al. [JBPV10]. We here restate its definition, but refer to the original publication for examples and more motivation.

The definition of weak transitions is standard.

Definition 3.4 (Weak transitions)**.** $\Psi \rhd P \implies P'$ means that either $P = P'$ or there exists P'' such that $\Psi \rhd P \xrightarrow{\tau} P''$ and $\Psi \rhd P'' \implies P'$.

For weak bisimulation we use static implication (rather than static equivalence) to compare the frames of the process pair under consideration.

Definition 3.5 (Static implication)**.** P *statically implies* Q in the environmental assertion Ψ, written $P \leq_\Psi Q$, if

$$\forall \varphi.\ \Psi \otimes \mathcal{F}(P) \vdash \varphi \ \Rightarrow\ \Psi \otimes \mathcal{F}(Q) \vdash \varphi$$

Definition 3.6 (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: for all Ψ' there exist Q', Q'' such that

$$\Psi \rhd Q \implies Q' \quad \wedge \quad \Psi \otimes \Psi' \rhd Q' \implies Q'' \quad \wedge \quad P \leq_\Psi Q' \quad \wedge \quad \mathcal{R}(\Psi \otimes \Psi', P, Q'')$$

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

(3) Extension of arbitrary assertion: for all Ψ' it holds that $\mathcal{R}(\Psi \otimes \Psi', P, Q)$

(4) Weak simulation: for all P',

  (a) if $\Psi \rhd P \xrightarrow{\tau} P'$ then $\exists Q'.\ \Psi \rhd Q \implies Q' \wedge \mathcal{R}(\Psi, P', Q')$; and

(b) for all $\Psi', \alpha \neq \tau$ such that $\mathrm{bn}(\alpha)\#\Psi, Q$, there exist Q', Q'', Q''' such that

$$\begin{array}{l}\Psi \rhd Q \Longrightarrow Q' \quad \wedge \quad \Psi \rhd Q' \xrightarrow{\alpha} Q'' \quad \wedge \quad \Psi \otimes \Psi' \rhd Q'' \Longrightarrow Q''' \\ \wedge \quad P \leq_\Psi Q' \quad \wedge \quad \mathcal{R}(\Psi \otimes \Psi', P', Q''')\end{array}$$

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

Above, (1) allows Q to take τ-transitions before and after enabling at least those conditions that hold in the frame of P, as per Definition 3.5. Moreover, when testing these conditions, the observer may also add an assertion Ψ' to the environment. In (4b), the observer may test the validity of conditions when matching a visible transition, and may also add an assertion as above.

To obtain a congruence from weak bisimulation, we must require that every τ-transition are simulated by a weak transition containing at least one τ-transition.

Definition 3.7. A *weak τ-bisimulation* $\mathcal{R}$ is a ternary relation between assertions and pairs of agents such that $\mathcal{R}(\Psi, P, Q)$ implies all conditions of a weak bisimulation (Definition 3.6) with 4a replaced by

(4a$'$) if $\Psi \rhd P \xrightarrow{\tau} P'$ then $\exists Q', Q''.\, \Psi \rhd Q \xrightarrow{\tau} Q' \wedge \Psi \rhd Q' \Longrightarrow Q'' \wedge \mathcal{R}(\Psi, P', Q'')$.

We then let $P \approx_\Psi Q$ mean that for all sequences $\widetilde{\sigma}$ of substitutions there is a weak τ-bisimulation $\mathcal{R}$ such that $\mathcal{R}(\Psi, P\widetilde{\sigma}, Q\widetilde{\sigma})$. We write $P \approx Q$ for $P \approx_{\mathbf{1}} Q$.

Lemma 3.8 (Comparing bisimulations)**.** *For all relations $\mathcal{R} \subseteq \mathbf{A} \times \mathbf{P} \times \mathbf{P}$,*

- *if $\mathcal{R}$ is a strong bisimulation then $\mathcal{R}$ is a weak τ-bisimulation.*
- *if $\mathcal{R}$ is a weak τ-bisimulation then $\mathcal{R}$ is a weak bisimulation.*

Corollary 3.9 (Comparing congruences)**.** *If $P \sim_\Psi Q$ then $P \approx_\Psi Q$.*

We seek to establish the following standard congruence and structural properties properties of strong and weak bisimulation:

Definition 3.10 (Congruence relation)**.** A relation $\mathcal{R} \subseteq \mathbf{A} \times \mathbf{P} \times \mathbf{P}$, where $(\Psi, P, Q) \in \mathcal{R}$ is written $P \mathrel{\mathcal{R}_\Psi} Q$, is a *congruence* iff for all Ψ, $\mathcal{R}_\Psi$ is an equivalence relation, and the following hold:

$$\begin{array}{lrcl}
\textsc{CPar} & P \mathrel{\mathcal{R}_\Psi} Q & \Longrightarrow & (P \mid R) \mathrel{\mathcal{R}_\Psi} (Q \mid R) \\
\textsc{CRes} & a\#\Psi \wedge P \mathrel{\mathcal{R}_\Psi} Q & \Longrightarrow & (\nu a)P \mathrel{\mathcal{R}_\Psi} (\nu a)Q \\
\textsc{CBang} & P \mathrel{\mathcal{R}_\Psi} Q & \Longrightarrow & !P \mathrel{\mathcal{R}_\Psi} !Q \\
\textsc{CCase} & \forall i.P_i \mathrel{\mathcal{R}_\Psi} Q_i & \Longrightarrow & \mathbf{case} \mathbin{[]} \widetilde{\varphi} : \widetilde{P} \;\mathcal{R}_\Psi\; \mathbf{case} \mathbin{[]} \widetilde{\varphi} : \widetilde{Q} \\
\textsc{COut} & P \mathrel{\mathcal{R}_\Psi} Q & \Longrightarrow & \overline{M}\, N \,.\, P \mathrel{\mathcal{R}_\Psi} \overline{M}\, N \,.\, Q \\
\textsc{CIn} & P \mathrel{\mathcal{R}_\Psi} Q & \Longrightarrow & \underline{M}(\lambda\widetilde{x})X \,.\, P \mathrel{\mathcal{R}_\Psi} \underline{M}(\lambda\widetilde{x})X \,.\, Q
\end{array}$$

A relation that satisfies all of the above implications except CIn is called an *open congruence* if it also satisfies the following:

$$\textsc{CIn-2} \quad (\forall \widetilde{L}.\, P[\widetilde{x} := \widetilde{L}] \mathrel{\mathcal{R}_\Psi} Q[\widetilde{x} := \widetilde{L}]) \implies \underline{M}(\lambda\widetilde{x})X \,.\, P \mathrel{\mathcal{R}_\Psi} \underline{M}(\lambda\widetilde{x})X \,.\, Q$$

A relation that does not satisfy rule CCase but is otherwise an open congruence is called a *weak open congruence*.

Definition 3.11 (Structural congruence)**.** *Structural congruence*, denoted $\equiv \in \mathbf{P} \times \mathbf{P}$, is the smallest relation such that $\{(\mathbf{1}, P, Q) \;:\; P \equiv Q\}$ is a congruence relation, and that satisfies the following clauses whenever $a\#Q, \widetilde{x}, M, N, X, \widetilde{\varphi}$:

$$\begin{array}{rclrcl}
\mathbf{case} \; [] \; \widetilde{\varphi} : \widetilde{(\nu a)P} & \equiv & (\nu a)\mathbf{case} \; [] \; \widetilde{\varphi} : \widetilde{P} & !P & \equiv & P \,|\, !P \\
\underline{M}(\lambda\widetilde{x})X \,.\, (\nu a)P & \equiv & (\nu a)\underline{M}(\lambda\widetilde{x})X \,.\, P & P \,|\, (Q \,|\, R) & \equiv & (P \,|\, Q) \,|\, R \\
\overline{M}\; N \,.\, (\nu a)P & \equiv & (\nu a)\overline{M}\; N \,.\, P & P \,|\, Q & \equiv & Q \,|\, P \\
Q \,|\, (\nu a)P & \equiv & (\nu a)(Q \,|\, P) & P & \equiv & P \,|\, \mathbf{0} \\
(\nu b)(\nu a)P & \equiv & (\nu a)(\nu b)P & (\nu a)\mathbf{0} & \equiv & \mathbf{0}
\end{array}$$

A relation $\mathcal{R} \subseteq \mathbf{P} \times \mathbf{P}$ is *complete with respect to structual congruence* if $\equiv \subseteq \mathcal{R}$.

Our goal is to establish that for all Ψ the relations $\dot{\sim}_\Psi$, $\sim_\Psi$, $\dot{\approx}_\Psi$ and $\approx_\Psi$ are complete with respect to structural congruence; that $\dot{\sim}$ is an open congruence; that $\sim$ is a congruence; that $\dot{\approx}$ is a weak open congruence; and that $\approx$ is a congruence.

3.1. **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 3.12. *For trivially sorted psi-calculi, $\dot{\sim}_\Psi$, $\sim_\Psi$, $\dot{\approx}_\Psi$ and $\approx_\Psi$ are complete wrt. structural congruence for all Ψ, $\dot{\sim}$ is an open congruence, $\sim$ is a congruence, $\dot{\approx}$ is a weak open congruence, and $\approx$ is a congruence.*

These results have all been machine-checked in Isabelle [ÅP14]. The proof scripts are adapted from Bengtson's formalisation of psi calculi [Ben10]. They constitute 30579 lines of Isabelle code; Bengtson's code is 28414 lines. The same technical lemmas hold and the proof scripts are essentially identical, save for the input cases of inductive proofs and a more detailed treatment of structural congruence. This represents no more than three 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 matching construct. 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 2.11 (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.

3.2. **Arbitrary sorted psi-calculi.** We here extend the results of Theorem 3.12 to arbitrary sorted psi-calculi. The idea is to introduce an explicit error element $\bot$, resulting from application of ill-sorted substitutions. For technical reasons we must also include one extra condition $\mathtt{fail}$ (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 $I = (\mathbf{T}_I, \mathbf{X}_I, \mathbf{C}_I, \mathbf{A}_I, \dots)$ be a sorted psi-calculus. We construct a trivially sorted psi-calculus $U(I)$ with one extra sort, $\mathtt{error}$, and constant symbols $\bot$ and $\mathtt{fail}$, with empty support of sort $\mathtt{error}$, where $\bot$ is not a channel, never entailed, matches nothing and entails nothing but $\mathtt{fail}$.

The parameters of $U(I)$ are defined by $U(I) = (\mathbf{T}_I \cup \{\bot\}, \mathbf{X}_I \cup \{(\bot, S) \;:\; S \subset_{\text{fin}} \mathcal{N}\}, \mathbf{C}_I \cup \{\bot, \mathtt{fail}\}, \mathbf{A}_I \cup \{\bot\})$. We define $\Psi \otimes \bot = \bot \otimes \Psi = \bot$ for all Ψ, and otherwise $\otimes$ is as in I. $\textsc{match}$ is the same in $U(I)$ as in I, plus $\textsc{match}(M, \widetilde{x}, (\bot, S)) = \emptyset$. Channel equivalence $\dot{\leftrightarrow}$ is the same in $U(I)$ as in I, plus $M \dot{\leftrightarrow} \bot = \bot \dot{\leftrightarrow} M = \bot \dot{\leftrightarrow} \bot = \bot$. For $\Psi \neq \bot$ we let $\Psi \vdash \varphi$ in $U(I)$ iff $\Psi \vdash \varphi$ in I, and we let $\bot \vdash \varphi$ iff $\varphi = \mathtt{fail}$. Substitution is then defined in $U(I)$ as follows:

$$T[\widetilde{a} := \widetilde{N}]_{U(I)} := \begin{cases} T[\widetilde{a} := \widetilde{N}]_I & \text{if } \textsc{sort}(a_i) \prec_I \textsc{sort}(N_i) \text{ and} \\ & \quad N_i \neq \bot \text{ for all } i, \text{ and } T \neq (\bot, S) \\ (\bot, S \setminus \widetilde{a}) & \text{if } T = (\bot, S) \text{ is a pattern} \\ (\bot, \bigcup \textsc{vars}(T)) & \text{otherwise, if } T \text{ is a pattern} \\ \bot & \text{otherwise} \end{cases}$$

Lemma 3.13. *$U(I)$ as defined above is a sorted psi-calculus, and any well-formed process P in I is well-formed in $U(I)$.*

Proof. A straight-forward application of the definitions. □

Processes in I have the same transitions in $U(I)$.

Lemma 3.14. *If P is well-formed in I and $\Psi \neq \bot$, then $\Psi \rhd P \xrightarrow{\alpha} P'$ in $U(I)$ iff $\Psi \rhd P \xrightarrow{\alpha} P'$ in I.*

Proof. By induction on the derivation of the transitions. The cases In, Out, Case and Com use the fact that $\textsc{match}$, $\vdash$ and $\dot{\leftrightarrow}$ are the same in I and $U(I)$, and that substitutions in I have the same effect when considered as substitutions in $U(I)$. □

Bisimulation in $U(I)$ coincides with bisimulation in I for processes in I.

Lemma 3.15. *Assume that P and Q are well-formed processes in I. Then $P \mathrel{\dot{\sim}_\Psi} Q$ in I iff $P \mathrel{\dot{\sim}_\Psi} Q$ in $U(I)$, and $P \mathrel{\dot{\approx}_\Psi} Q$ in I iff $P \mathrel{\dot{\approx}_\Psi} Q$ in $U(I)$.*

Proof. We show only the proof for the strong case; the weak case is similar. Let $\mathcal{R}$ be a bisimulation in $U(I)$. Then $\{(\Psi, P', Q') \in \mathcal{R} \;:\; \Psi \neq \bot \wedge P', Q' \text{ well-formed in } I\}$ is a bisimulation in I: the proof is by coinduction, using Lemma 3.14 and Theorem 2.11 in the simulation case.

Symmetrically, let $\mathcal{R}'$ be a bisimulation in I, and let $\mathcal{R}'_\bot = \{(\bot, P, Q) \;:\; \exists \Psi. (\Psi, P, Q) \in \mathcal{R}'\}$. Then $\mathcal{R}' \cup \mathcal{R}'_\bot$ is a bisimulation in $U(I)$: simulation steps from $\mathcal{R}'$ lead back to $\mathcal{R}'$ by Lemma 3.14. From $\mathcal{R}'_\bot$ there are no transitions, since $\bot$ entails no channel equivalence clauses. The other parts of Definition 3.1 are straightforward; when applying clause 3 with $\Psi' = \bot$ the resulting triple is in $\mathcal{R}'_\bot$. □

With Lemma 3.15, we can lift the congruence and the structural congruence results for trivially sorted psi-calculi to arbitrary sorted calculi:

Theorem 3.16. *All clauses of Theorem 3.12 are valid in all sorted psi-calculi.*

Proof. Fix a sorted psi-calculus I. For strong and weak bisimilarity, we show only the proofs for commutativity and congruence of the parallel operator. The other cases are analogous.

For commutativity of parallel composition, let P and Q be well-formed in I and $\Psi \neq \bot$. By Theorem 3.12, $P \mid Q \sim_\Psi Q \mid P$ holds in $U(I)$. By Definition 3.1, $(P \mid Q)\widetilde{\sigma} \mathrel{\dot\sim}_\Psi (Q \mid P)\widetilde{\sigma}$ in $U(I)$ for all $\widetilde{\sigma}$. By Theorem 2.11, when $\widetilde{\sigma}$ is well-sorted then $(P \mid Q)\widetilde{\sigma}$ and $(Q \mid P)\widetilde{\sigma}$ are well-formed. By Lemma 3.15, $(P|Q)\widetilde{\sigma} \mathrel{\dot\sim}_\Psi (Q|P)\widetilde{\sigma}$ in I for all well-formed $\widetilde{\sigma}$. $P|Q \sim_\Psi Q|P$ follows by definition. $P \mid Q \approx_\Psi Q \mid P$ follows by Corollary 3.9.

For congruence of parallel composition for bisimulation, assume $P \mathrel{\dot\sim}_\Psi Q$ holds in I. By Lemma 3.15, $P \mathrel{\dot\sim}_\Psi Q$ holds in $U(I)$. Theorem 3.12 thus yields $P \mid R \mathrel{\dot\sim}_\Psi Q \mid R$ in $U(I)$, and Lemma 3.15 yields the same in I. The same argument shows that $P \mathrel{\dot\approx}_\Psi Q$ implies $P \mid R \mathrel{\dot\approx}_\Psi Q \mid R$ in I.

This approach does not work for proving congruence properties for $\sim$ or $\approx$, since the closure of bisimilarity under well-sorted substitutions does not imply its closure under ill-sorted substitutions: consider a sorted psi-calculus I such that $\mathbf{0} \sim (\!|\mathbf{1}|\!)$. This equation does not hold in $U(I)$: if σ is ill-sorted then $\mathbf{1}\sigma = \bot$, but $\mathbf{0} \mathrel{\dot\sim} (\!|\bot|\!)$ does not hold since only $\bot$ entails `fail`. Instead, we have performed direct proofs: they are identical, line by line, to the proofs in the trivially sorted case (cf. [Ben10]).

□

4. Representing Standard Process Calculi

We here consider psi-calculi corresponding to some variants of popular process calculi. One main point of our work is that we can represent other calculi directly as psi-calculi, without elaborate coding schemes. In the original psi-calculi we could in this way directly represent the monadic pi-calculus, but for the other calculi presented below a corresponding unsorted psi-calculus would contain terms with no counterpart in the represented calculus, as explained in Section 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).

Because of the simplicity of the mapping and the strength of the correspondence we say that psi-calculi *represent* other process calculi, in contrast to *encoding* them. A representation is significantly stronger than standard correspondences, such as the approach to encodability proposed by Gorla [Gor10]. Gorla's criteria aim to capture the property that one language can encode the behaviour of another using some (possibly elaborate) protocol, while our criteria aim to capture the property that two languages are for all practical purposes one and the same.

Definition 4.1. A psi-calculus is a *representation* of a process calculus with processes $P \in \mathcal{P}$ and labelled transition system $\rightarrow \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$, if there exist an equivariant map $[\![\cdot]\!]$ from $\mathcal{P}$ to psi-calculus processes and an equivariant relation $\cong$ between $\mathcal{A}$ and psi-calculus actions that preserves the kind (input, output, tau) and subject of actions, such that

(1) $[\![\cdot]\!]$ is a simple homomorphism, i.e., for each process constructor f of $\mathcal{P}$ there is an equivariant psi-calculus context C such that $[\![f(P_1, \ldots, P_n)]\!] = C[[\![P_1]\!], \ldots, [\![P_n]\!]]$.

(2) $[\![\cdot]\!]$ is a strong operational correspondence (modulo structural equivalence), i.e.,
   (a) whenever $P \xrightarrow{\beta} Q$ then $[\![P]\!] \xrightarrow{\alpha} P'$ such that $[\![Q]\!] \equiv P'$ and $\beta \approxeq \alpha$; and
   (b) whenever $[\![P]\!] \xrightarrow{\alpha} P'$ then $P \xrightarrow{\beta} Q$ such that $[\![Q]\!] \equiv P'$ and $\beta \approxeq \alpha$.

A representation is *complete* if it additionally satisfies

(3) $[\![\cdot]\!]$ is surjective modulo strong bisimulation congruence, i.e., for each psi process P there is $Q \in \mathcal{P}$ such that $P \sim [\![Q]\!]$.

Briefly, the differences to Gorla's criteria are as follows:

- In Gorla's approach, the contexts that process constructors are translated to may fix certain names, or translate one name into several names, in accordance with a renaming policy. Our approach admits no such special treatment of names.
- Gorla requires the translation function to be name invariant up-to the renaming policy. We require equivariance, which corresponds to name invariance up-to the policy of renaming every name to itself.
- Gorla uses three criteria for semantic correspondence: weak operational correspondence modulo some equivalence for silent transitions, that the translation does not introduce divergence, and that reducibility to a success process in the source and target processes coincides. Clearly strong operational correspondence modulo structural equivalence implies all of these criteria.
- Our surjectivity requirement implies that the target language cannot express more behaviours than the source language, something that is not considered in Gorla's approach.

Our use of structural equivalence in the operational correspondence allows to admit representations of calculi that use a structural congruence rule to define a labelled semantics (cf. Section 4.4).

Below, for simplicity we let the assertions be the singleton $\{\mathbf{1}\}$ in all examples, with $\mathbf{1} \vdash \top$ and $\mathbf{1} \not\vdash \bot$. We use the standard notion of simultaneous substitution, and let $\textsc{match}(M, \widetilde{x}, X) = \emptyset$ where not otherwise defined. Proofs of lemmas and theorems can be found in Appendix A.

4.1. **Unsorted Polyadic pi-calculus.** In the polyadic pi-calculus [Mil93] the only values that can be transmitted between agents are tuples of names. Tuples cannot be nested. The processes are defined as follows

$$\boxed{P, Q \quad ::= \quad \mathbf{0} \;\mid\; x(\tilde{y}).P \;\mid\; \overline{x}\langle\tilde{y}\rangle.P \;\mid\; [a = b]P \;\mid\; \nu x\, P \;\mid\; !P \;\mid\; P \,|\, Q \;\mid\; P + Q}$$

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 $a(x).P \mid \overline{a}\langle y, z\rangle.Q$ is a valid agent. This agent has no communication action since the lengths of the tuples mismatch.

We now present the psi-calculus **PPI**, which we will show represents the polyadic pi-calculus.

PPI	
$\mathbf{T} = \mathcal{N} \cup \{\langle\widetilde{a}\rangle : \widetilde{a} \subset_{\mathrm{fin}} \mathcal{N}\}$	$\mathcal{S} = \{\mathtt{chan}, \mathtt{tup}\}$
$\mathbf{C} = \{\top\} \cup \{a = b \mid a, b \in \mathcal{N}\}$	$\mathcal{S}_{\mathcal{N}} = \{\mathtt{chan}\}$
$\mathbf{X} = \{\langle\widetilde{a}\rangle : \widetilde{a} \subset_{\mathrm{fin}} \mathcal{N} \wedge \widetilde{a} \text{ distinct}\}$	$\textsc{sort}(a) = \mathtt{chan}$
$\dot{\leftrightarrow}$ = identity on names	$\textsc{sort}(\langle\widetilde{a}\rangle) = \mathtt{tup}$
$\mathbf{1} \vdash a = a$	$\mathcal{S}_{\nu} = \{\mathtt{chan}\}$
$\textsc{vars}(\langle\widetilde{a}\rangle) = \{\widetilde{a}\}$	$\prec = \{(\mathtt{chan}, \mathtt{chan})\}$
$\textsc{match}(\langle\widetilde{a}\rangle, \widetilde{x}, \langle\widetilde{y}\rangle) = \{\pi \cdot \widetilde{a}\}$ if $\lvert\widetilde{a}\rvert = \lvert\widetilde{y}\rvert$ and $\widetilde{x} = \pi \cdot \widetilde{y}$	$\overline{\propto} = \underline{\propto} = \{(\mathtt{chan}, \mathtt{tup})\}$

This being our first substantial example, we give a detailed explanation of the new instance parameters. Patterns $\mathbf{X}$ are finite vectors of distinct names. The sorts $\mathcal{S}$ are `chan` for channels and `tup` for tuples (of names); the only sort of names $\mathcal{S}_{\mathcal{N}}$ is channels, as is the sort of restricted names. The only sort of substitutions ($\prec$) are channels for channels; the only sort of sending ($\overline{\propto}$) and receiving ($\underline{\propto}$) is tuples over channels. In an input prefix all names in the tuple must be bound (VARS) and a vector of names $\widetilde{a}$ matches a pattern $\widetilde{y}$ if the lengths match and all names in the pattern are bound (in some arbitrary order).

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 $\mathtt{chan} \mathrel{\underline{\propto}} \mathtt{tup}$ and $\mathtt{chan} \mathrel{\overline{\propto}} \mathtt{tup}$, with $\textsc{vars}(\langle x, y\rangle) = \{\{x, y\}\}$. 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.

To prove that **PPI** is a psi-calculus, we need to check the requisites on the parameters (data types and operations) defined above. Clearly the parameters are all equivariant, since no names appear free in their definitions. For the original psi-calculus parameters (Definition 2.1), the requisites are symmetry and transitivity of channel equivalence, which hold because of the same properties of (entailment of) name equality, and abelian monoid laws and compositionality for assertion composition, which trivially hold since $\mathbf{A} = \{\mathbf{1}\}$. The standard notion of simultaneous substitution of names for names preserves sorts, and also satisfies the other requirements of Definition 2.4. To check the requisites on pattern matching (Definition 2.5), it is easy to see that MATCH generates only well-sorted substitutions (of names for names), and that $\mathrm{n}(\widetilde{b}) = \mathrm{n}(\langle\widetilde{a}\rangle)$ whenever $\widetilde{b} \in \textsc{match}(\langle\widetilde{a}\rangle, \widetilde{x}, \langle\widetilde{y}\rangle)$ Finally, for all name swappings $(\widetilde{x}\ \widetilde{y})$ we have $\textsc{match}(\langle\widetilde{a}\rangle, \widetilde{x}, \langle\widetilde{z}\rangle) = \textsc{match}(\langle\widetilde{a}\rangle, \widetilde{y}, (\widetilde{x}\ \widetilde{y}) \cdot \langle\widetilde{z}\rangle)$.

PPI is a direct representation of the polyadic pi-calculus as presented by Sangiorgi [San93] (with replication instead of process constants).

Definition 4.2 (Polyadic Pi-Calculus to **PPI**).
Let $[\![\cdot]\!]$ be the function that maps the polyadic pi-calculus to **PPI** processes as follows. The function $[\![\cdot]\!]$ is homomorphic for $\mathbf{0}$, restriction, replication and parallel composition, and is otherwise defined as follows:

$$\begin{array}{rcl}
[\![P + Q]\!] & = & \mathbf{case}\ \top : [\![P]\!]\ [\!]\ \top : [\![Q]\!] \\
[\![[x = y]P]\!] & = & \mathbf{case}\ x = y : [\![P]\!] \\
[\![x(\tilde{y}).P]\!] & = & \underline{x}(\lambda\widetilde{y})\langle\widetilde{y}\rangle.[\![P]\!] \\
[\![\overline{x}\langle\tilde{y}\rangle.P]\!] & = & \overline{x}\langle\tilde{y}\rangle.[\![P]\!]
\end{array}$$

Similarly, we also translate the actions of polyadic pi-calculus. Here each action corresponds to a set of psi actions, since in a pi-calculus output label “the order of the bound names is immaterial" [SW01, p. 129], which is not the case in psi-calculi.

$$\begin{array}{rcl} [\![(\nu\tilde{y})\overline{x}\langle\tilde{z}\rangle]\!] & = & \{\overline{x}\ (\nu\widetilde{y'})\ \langle\widetilde{z}\rangle\ :\ \widetilde{y}' = \pi\cdot\widetilde{y}\} \\ [\![x\langle\tilde{z}\rangle]\!] & = & \{\underline{x}\ \langle\tilde{z}\rangle\} \\ [\![\tau]\!] & = & \{\tau\} \end{array}$$

Although the binders in bound output actions are ordered in psi-calculi, they can be arbitrarily reordered.

Lemma 4.3. *If* $\Psi \rhd P \xrightarrow{\overline{M}\ (\nu\widetilde{a})\ N} Q$ *then* $\Psi \rhd P \xrightarrow{\overline{M}\ (\nu\pi\cdot\widetilde{a})\ N} Q$

Proof. By induction on the derivation of the transition. The base case is trivial. In the Open rule, we use the induction hypothesis to reorder the bound names in the premise as desired; we can then add the opened name at any position in the action in the conclusion of the rule. The other induction cases are trivial. □

We can now show that $[\![\cdot]\!]$ is a strong operational correspondence.

Theorem 4.4. *If P and Q are polyadic pi-calculus processes, then:*

(1) *If $P \xrightarrow{\beta} P'$ then for all $\alpha \in [\![\beta]\!]$ we have $[\![P]\!] \xrightarrow{\alpha} [\![P']\!]$*

(2) *If $[\![P]\!] \xrightarrow{\alpha} P''$ then $P \xrightarrow{\beta} P'$ such that $\alpha \in [\![\beta]\!]$ and $[\![P']\!] = P''$*

Proof. By induction on the length of derivation of the transitions, using Lemma 4.3 in the **OPEN** case of (1). □

We have now shown that the polyadic pi-calculus can be embedded in **PPI**, with an embedding $[\![\cdot]\!]$ that is a strong operational correspondence.

In order to investigate surjectivity properties of the embedding $[\![\cdot]\!]$, we also define a translation $\overline{P}$ in the other direction.

Definition 4.5 (**PPi** to Polyadic Pi-Calculus)**.** The translation $\overline{\cdot}$ is homomorphic for **0**, restriction, replication and parallel composition, and is otherwise defined as follows:

$$\begin{array}{rcl} \overline{(\!|\mathbf{1}|\!)} & = & \mathbf{0} \\ \overline{\mathbf{case}\ \varphi_1 : P_1\ [\!]\ \ldots\ [\!]\ \varphi_n : P_n} & = & \overline{\varphi_1 : P_1} + \cdots + \overline{\varphi_n : P_n} \\ \overline{\underline{x}(\lambda\widetilde{y})\langle\widetilde{z}\rangle.P} & = & x(\tilde{z}).\overline{P} \\ \overline{\overline{x}\langle\tilde{y}\rangle.P} & = & \overline{x}\langle\tilde{y}\rangle.\overline{P} \end{array}$$

where condition-guarded processes are translated as

$$\begin{array}{rcl} \overline{x = y : P} & = & [x = y]\overline{P} \\ \overline{\top : P} & = & \overline{P}. \end{array}$$

Above, note that the order of the binders in input prefixes is ignored. To show that the reverse translation is an inverse of $[\![\cdot]\!]$ modulo bisimilarity, we need to prove that their order does not matter.

Lemma 4.6. *In* **PPI**, $\underline{x}(\lambda\widetilde{y})\langle\widetilde{z}\rangle.P \sim \underline{x}(\lambda\widetilde{z})\langle\widetilde{z}\rangle.P$.

Proof. Straightforward from the definitions of MATCH and substitution on patterns. □

We now show that the embeddings $\overline{\cdot}$ and $[\![\cdot]\!]$ are inverses, modulo bisimilarity.

Theorem 4.7. *If P is a* **PPI** *process, then* $P \sim [\![\overline{P}]\!]$.

Proof. By structural induction on P. The input case uses Lemma 4.6. For **case** agents, we use an inner induction on the number of branches, with Lemma 3.3 applied in the induction case. □

Let the relation $\sim_e^c$ be an early congruence of polyadic pi-calculus agents as defined in [San93]. Then we have

Corollary 4.8. *If P is a polyadic pi-calculus process, then $P \sim_e^c \overline{[\![P]\!]}$.*

We also have

Corollary 4.9. *If P and Q are polyadic pi-calculus process, then $P \sim_e^c Q$ (i.e., P and Q are early labelled congruent) iff $[\![P]\!] \sim [\![Q]\!]$.*

Proof. Follows from the strong operational correspondence of Theorem 4.4, and $[\![\cdot]\!]$ commuting with substitutions. □

This shows that every **PPI** process corresponds to a polyadic pi-calculus process, modulo strong bisimulation congruence, since $\overline{\cdot}$ is surjective on the bisimulation classes of polyadic pi-calculus, and the inverse of $[\![\cdot]\!]$. In other words, **PPI** is a *representation.*

Theorem 4.10. **PPI** *is a complete representation of the polyadic pi-calculus.*

Proof. We let $\beta \approxeq \alpha$ iff $\alpha \in [\![\beta]\!]$.

(1) $[\![\cdot]\!]$ is a simple homomorphism by definition.
(2) $[\![\cdot]\!]$ is a strong operational correspondence by Theorem 4.4.
(3) $[\![\cdot]\!]$ is surjective modulo strong bisimulation congruence by Theorem 4.7. □

4.2. **LINDA** [Gel85].

A process calculus with LINDA-like pattern matching can easily be obtained from the **PPI** calculus, by modifying the possible binding names in patterns.

LINDA

Everything as in **PPI** except:
$\mathbf{X} = \{\langle\widetilde{a}\rangle : \widetilde{a} \subset_{\text{fin}} \mathcal{N}\}$
$\textsc{vars}(\langle\widetilde{a}\rangle) = \mathcal{P}(\widetilde{a})$
$\textsc{match}(\langle\widetilde{a}\rangle, \widetilde{x}, \langle\widetilde{y}\rangle) = \{\widetilde{c} : \langle\widetilde{a}\rangle = \langle\widetilde{y}\rangle[\widetilde{x} := \widetilde{c}]\}$

Here, any subset of the names occurring in a pattern may be bound in the input prefix; this allows to only receive messages with particular values at certain positions (sometimes called "structured names" [Gel85]) We also do not require patterns to be linear, i.e., the same variable may occur more than once in a pattern, and the pattern only matches a tuple if each occurrence of the variable corresponds to the same name in the tuple.

As an example, $\underline{a}(\lambda x)\langle x,x,z\rangle.P \mid \overline{a}\langle c,c,z\rangle.Q \xrightarrow{\tau} P[x := c] \mid Q$ while the agent $\underline{a}(\lambda x)\langle x,x,z\rangle.P \mid \overline{a}\langle c,d,z\rangle.Q$ has no τ transition.

To prove that **LINDA** is a psi-calculus, the interesting case is the preservation of variables of substitution on patterns in Definition 2.4, i.e., that $\widetilde{x} \in \textsc{vars}(\langle\widetilde{y}\rangle)$ and $\widetilde{x}\#\sigma$ implies $\widetilde{x} \in \textsc{vars}(\langle\widetilde{y}\rangle\sigma)$. This holds because standard substitution preserves names and structure: if $x \in \widetilde{y}$ and $x\#\sigma$, then there is $\widetilde{z}$ such that $\langle\widetilde{y}\rangle\sigma = \langle\widetilde{z}\rangle$ and $x \in \widetilde{z}$.

4.3. **Sorted polyadic pi-calculus.** Milner's classic sorting [Mil93] 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 in its domain. 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 $a(b_1, \dots b_n)$ it holds that a has some sort s where $\mathtt{ob}(s)$ is the sequence of sorts of $b_1, \dots, b_n$ and similarly for output prefixes.

SORTEDPPI

Everything as in **PPI** except:

$$\mathcal{S}_{\mathcal{N}} = \mathcal{S}_{\nu} = \mathrm{S} \qquad \mathcal{S} = \mathrm{S}^*$$

$$\prec\, = \{(s,s) : s \in \mathrm{S}\} \qquad \overline{\propto} = \underline{\propto} = \{(s, \mathtt{ob}(s)) : s \in \mathrm{S}\}$$

$$\textsc{sort}(\langle a_1, \dots, a_n\rangle) = \textsc{sort}(a_1), \dots, \textsc{sort}(a_n)$$

$$\textsc{match}(\langle \widetilde{a}\rangle, \widetilde{x}, \langle \widetilde{y}\rangle) = \{\pi \cdot \widetilde{a}\} \quad \text{if } \widetilde{x} = \pi \cdot \widetilde{y} \text{ and } \textsc{sort}(\langle \widetilde{a}\rangle) = \textsc{sort}(\langle \widetilde{y}\rangle)$$

We need to show that MATCH always generates well-sorted substitutions: this holds since whenever $\widetilde{c} \in \textsc{match}(\langle \widetilde{a}\rangle, \widetilde{x}, \langle \widetilde{y}\rangle)$ we have that $[\widetilde{x} := \widetilde{c}] = [\pi \cdot \widetilde{y} := \pi \cdot \widetilde{a}]$ and $\textsc{sort}(y_i) = \textsc{sort}(a_i)$ for all i.

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 $s \;\underline{\propto}\; t_1, t_2$ and $t_1 \;\overline{\propto}\; t_2$, with $\textsc{vars}(x, y) = \{\{x, y\}\}$.

A formal comparison with the system in [Mil93] 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 4.11. *P is well-sorted iff $[\![P]\!]$ is well-formed.*

Proof. A trivial induction over the structure of P, observing that the requirements are identical. □

Theorem 4.12. SORTEDPPI *is a complete representation of the sorted polyadic pi-calculus.*

Proof. The operational correspondence in Theorem 4.4 still holds when restricted to well-formed agents. The inverse translation $\bar{\cdot}$ maps well-formed agents to well-sorted processes, so the surjectivity result in Theorem 4.7 still applies. □

4.4. **Polyadic synchronisation pi-calculus.** Carbone and Maffeis [CM03] 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. The processes of ${}^{e}\pi$ is defined as follows.

$$\begin{array}{rcl} P, Q & ::= & \mathbf{0} \mid \Sigma_i \alpha_i.P_i \mid P \,|\, Q \mid (\nu a)P \mid\; !P \\ \alpha & ::= & \tilde{a}(x) \mid \tilde{a}\langle b\rangle \end{array}$$

In order to represent ${}^{e}\pi$, only minor modifications to the representation of the polyadic pi-calculus in Section 4.1 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 $a = b$, since they can be encoded (see [CM03]).

PSPI

Everything as in **PPI** except:

$\mathbf{C} = \{\top, \bot\}$	$\widetilde{a} \leftrightarrow \widetilde{b}$ is $\top$ if $\widetilde{a} = \widetilde{b}$, and $\bot$ otherwise
$\mathbf{X} = \mathcal{N}$	$\textsc{vars}(x) = \{\{x\}\}$
$\overline{\propto} = \underline{\propto} = \{(\texttt{tup}, \texttt{chan})\}$	$\textsc{match}(a, x, x) = \{a\}$

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 communication over a restricted fresh name. The ${}^{e}\pi$ calculus also uses an operational semantics with late input, unlike psi-calculi. In order to yield a representation, we consider an early version $\longrightarrow^{e}$ of the semantics, obtained by turning bound input actions into free input actions at top-level.

$$\textsc{eIn}\ \frac{P \xrightarrow{\tilde{x}(y)} P'}{P \xrightarrow{\tilde{x}\ z}{}^{e} P'\{z/y\}} \qquad \textsc{Out}\ \frac{P \xrightarrow{\tilde{x}\langle c\rangle} P'}{P \xrightarrow{\tilde{x}\langle c\rangle}{}^{e} P'} \qquad \textsc{BOut}\ \frac{P \xrightarrow{\tilde{x}\langle \nu c\rangle} P'}{P \xrightarrow{\tilde{x}\langle \nu c\rangle}{}^{e} P'} \qquad \textsc{Tau}\ \frac{P \xrightarrow{\tau} P'}{P \xrightarrow{\tau}{}^{e} P'}$$

Definition 4.13 (Polyadic synchronisation pi-calculus to **PSPI**)**.** $[\![\cdot]\!]$ is homomorphic for $\mathbf{0}$, restriction, replication and parallel composition, and is otherwise defined as follows:

$$\begin{array}{rcl} [\![\Sigma_i \alpha_i.P_i]\!] & = & \textbf{case}\ \top_i : [\![\alpha_i.P_i]\!] \\ [\![\widetilde{x}(y).P]\!] & = & \underline{\langle\widetilde{x}\rangle}(\lambda y)y.[\![P]\!] \\ [\![\widetilde{x}\langle y\rangle.P]\!] & = & \overline{\langle\widetilde{x}\rangle}\ y.[\![P]\!] \end{array}$$

We translate bound and free output, free input, and tau actions in the following way.

$$\begin{array}{rcl} [\![\tilde{x}\langle \nu c\rangle]\!] & = & \overline{\langle\tilde{x}\rangle}\,(\nu c)\,c \\ [\![\tilde{x}\langle c\rangle]\!] & = & \overline{\langle\tilde{x}\rangle}\ c \\ [\![\tilde{x}\ y]\!] & = & \langle\widetilde{x}\rangle\ y \\ [\![\tau]\!] & = & \tau \end{array}$$

The transition system in ${}^{e}\pi$ is given up to structural congruence, i.e., for all α we have $\xrightarrow{\alpha} = (\equiv\xrightarrow{\alpha}\equiv)$.

Definition 4.14. $\equiv$ is the least congruence satisfying alpha conversion, the commutative monoidal laws with respect to both (|,0) and (+,0) and the following axioms[1]:

$$(\nu x)P \mid Q \equiv (\nu x)(P \mid Q) \text{ if } x\#Q \qquad\qquad (\nu x)P \equiv P \text{ if } x\#P$$

The proofs of operational correspondence are similar to the polyadic pi-calculus case. We have the following initial results for late input actions.

Lemma 4.15.

(1) *If* $P \xrightarrow{\tilde{x}(y)} P'$ *then for all* z, $[\![P]\!] \xrightarrow{\langle\tilde{x}\rangle\ z} P''$ *where* $P'' \equiv [\![P']\!][y := z]$.

(2) *If* $[\![P]\!] \xrightarrow{\langle\tilde{x}\rangle\ z} P''$ *then for all* $y\#P$, $P \xrightarrow{\tilde{x}(y)} P'$ *where* $[\![P'\{z/y\}]\!] = P''$.

[1] The original definition of $\equiv$ [CM03] includes an additional axiom $[x = x]P \equiv P$ allowing to contract successful matches, but this axiom is omitted here since the ${}^{e}\pi$ calculus does not include the match construct. Unusually, the definition of $\equiv$ does not admit commuting restrictions, i.e., $(\nu x)(\nu y)P \not\equiv (\nu y)(\nu x)P$.

Proof. By induction on the derivation of the transitions. □

This in turn yields the desired operational correpondence.

Theorem 4.16.

(1) *If* $P \xrightarrow{\alpha}{}^{e} P'$ *and* $\alpha \neq \tilde{x}(y)$, *then* $[\![P]\!] \xrightarrow{[\![\alpha]\!]} P''$ *where* $P'' \equiv [\![P']\!]$.

(2) *If* $[\![P]\!] \xrightarrow{\alpha'} P''$, *then* $P \xrightarrow{\alpha}{}^{e} P'$ *where* $[\![\alpha]\!] = \alpha'$ *and* $[\![P']\!] = P''$.

Proof. By induction on the derivation of the transitions. □

Again, these results lead us to say that the polyadic synchronization pi-calculus can be *represented* as a psi-calculus.

Theorem 4.17. PSPI *is a representation of the polyadic synchronization pi-calculus.*

Proof. We let $\beta \approxeq \alpha$ iff $\alpha = [\![\beta]\!]$.

(1) $[\![\cdot]\!]$ is a simple homomorphism by definition.

(2) $[\![\cdot]\!]$ is a strong operational correspondence by Theorem 4.4. □

To investigate the surjectivity properties of $[\![\cdot]\!]$, we need to consider the fact that polyadic synchronization pi has only mixed (i.e., prefix-guarded) choice.

Definition 4.18 (Case-guarded)**.** A **PSPI** process is case-guarded if in all its subterms of the form **case** $\varphi_1 : P_1 \mathbin{[\,]} \cdots \mathbin{[\,]} \varphi_n : P_n$, for all $i \in \{1, \ldots, n\}$, $\varphi_i = \top$ implies $P_i = \overline{M}\, N.Q$ or $P_i = \underline{M}(\lambda\tilde{x})X.Q$.

We define the translation $\overline{R}$ from case-guarded **PSPI** processes to ${}^{e}\pi$ as the translation with the same name from **PPI**, except that $\bot$-guarded branches of **case** statements are discarded.

Theorem 4.19. *For all case-guarded* **PSPI** *processes* R *we have* $R \sim [\![\overline{R}]\!]$.

Proof. By structural induction on R. For **case** agents, we use an inner induction on the number of branches, with Lemma 3.3 applied in the induction case. □

Corollary 4.20. *If* P *is a polyadic synchronization pi-calculus process, then* $P \underset{\cdot}{\sim} \overline{[\![P]\!]}$.

Corollary 4.21. *For all* ${}^{e}\pi$ *processes* P, Q, $P \underset{\cdot}{\sim} Q$ *(i.e.,* P *and* Q *are early labelled congruent) iff* $[\![P]\!] \sim [\![Q]\!]$.

Proof. By strong operational correspondence 4.16, and $[\![\cdot]\!]$ commuting with substitutions. □

We thus have that the case-guarded **PSPI** processes correspond to polyadic synchronization pi, modulo flattening and structural congruence.

4.5. **Value-passing CCS.** Value-passing CCS [Mil89] 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 $a(x)\,.\,\textbf{if}\ x > 3\ \textbf{then}\ P$. Formally, the value-passing CCS processes are defined by the following grammar with x, y ranging over names, v over values, b over boolean expressions, and L over set of names.

$$\boxed{P, Q \ ::= \ x(y).P \ \mid \ \overline{x}(v).P \ \mid \ \Sigma_i \ P_i \ \mid \ \mathbf{if}\ b\ \mathbf{then}\ P \ \mid \ P \setminus L \ \mid \ P \,|\, Q \ \mid \ !P \ \mid \ \mathbf{0}}$$

To represent this as a psi-calculus we assume an arbitrary set of expressions $e \in \mathbf{E}$ including at least the values $\mathbf{V}$. A subset of $\mathbf{E}$ is the boolean expressions $b \in \mathbf{E_B}$. 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 $v \in \mathbf{V}$ are closed and have sort `value`; all other expressions have sort `exp`. The boolean values are $\mathbf{V_B} := \mathbf{V} \cap \mathbf{E_B} = \{\top, \bot\}$, and $\mathbf{1} \vdash \top$ but $\neg(\mathbf{1} \vdash \bot)$. 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 $E(2+3) = 5$. We define substitution on expressions to use evaluation, i.e. $e[\widetilde{x} := \widetilde{V}] = E(e\{\widetilde{V}/\widetilde{x}\})$. As an example, $(x+3)[x := 2] = E((x+3)\{2/x\}) = E(2+3) = 5$. We use the single-variable patterns of Example 2.6.

VPCCS	
$\mathbf{T} = \mathcal{N} \cup \mathbf{E}$	$\mathcal{S}_\mathcal{N} = \{\texttt{chan}, \texttt{exp}\}$
$\mathbf{C} = \mathbf{E_B}$	$\mathcal{S} = \mathcal{S}_\mathcal{N} \cup \{\texttt{value}\}$
$\mathbf{A} = \{\mathbf{1}\}$	$v \in \mathbf{V} \Rightarrow \textsc{sort}(v) = \texttt{value}$
$\mathbf{X} = \mathcal{N}$	$e \in \mathbf{E} \setminus \mathbf{V} \Rightarrow \textsc{sort}(e) = \texttt{exp}$
$a \dot{\leftrightarrow} a = \top$	$e \in \mathbf{E} \Rightarrow e[\widetilde{x} := \widetilde{M}] = E(e\{\widetilde{M}/\widetilde{x}\})$
$e \dot{\leftrightarrow} e' = \bot$ otherwise	$\prec = \{(\texttt{exp}, \texttt{value})\}$
$\textsc{match}(v, a, a) = \{v\}$ if $v \in \mathbf{V}$	$\mathcal{S}_\nu = \{\texttt{chan}\}$
$\textsc{vars}(a) = \{a\}$	$\overline{\propto} = \underline{\propto} = \{(\texttt{chan}, \texttt{exp}), (\texttt{chan}, \texttt{value})\}$

Closed value-passing CCS processes correspond to **VPCCS** agents P where all free names are of sort `chan`. To prove that **VPCCS** is a psi-calculus, the interesting case is when the sort of a term is changed by substitution: let e be an open term, and σ a substitution such that $\mathrm{n}(e) \subseteq \mathrm{dom}(\sigma)$. Here $\textsc{sort}(e) = \texttt{exp}$ and $\textsc{sort}(e\sigma) = \texttt{value}$; this satisfies Definition 2.4 since $\texttt{value} \leq \texttt{exp}$ in the subsorting preorder (here $\texttt{exp} \leq \texttt{value}$ also holds, but is immaterial since there are no names of sort `value`).

We show that **VPCCS** represents value-passing CCS as defined by Milner [Mil89], with the following modifications:

- We use replication instead of process constants.
- 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 cofinite 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 the relabelling construct $P[f]$ of CCS 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.

- We only allow finite sets L in restrictions $P \setminus L$. With finite sums, this results in no loss of expressivity since agents have finite support.

Milner's restrictions are of sets of names, which we represent as a sequence of ν-binders. To create a unique such sequence from L, we assume an injective and support-preserving function $\overrightarrow{\cdot} : \mathcal{P}_{\mathrm{fin}}(\mathcal{N}_{\mathtt{chan}}) \to (\mathcal{N}_{\mathtt{chan}})^*$. For instance, $\overrightarrow{L}$ may be defined as sorting the names in L according to some total order on $\mathcal{N}_{\mathtt{chan}}$, which is always available since $\mathcal{N}_{\mathtt{chan}}$ is countable.

The mapping $[\![\cdot]\!]$ from value-passing CCS into **VPCCS** is defined homomorphically on parallel composition, output and $\mathbf{0}$, and otherwise as follows.

$$\begin{array}{rcl}
[\![x(y).P]\!] & = & \underline{x}(\lambda y)y.[\![P]\!] \\
[\![\Sigma_i\, P_i]\!] & = & \mathbf{case}\ \top : [\![P_1]\!]\ [\!]\ \cdots\ [\!]\ \top : [\![P_i]\!] \\
[\![\mathbf{if}\ b\ \mathbf{then}\ P]\!] & = & \mathbf{case}\ b : [\![P]\!] \\
[\![P \setminus L]\!] & = & (\nu \overrightarrow{L})[\![P]\!]
\end{array}$$

We translate the value-passing CCS actions as follows

$$\begin{array}{rcl}
[\![x(v)]\!] & = & \underline{x}\, v \\
[\![\overline{x}(v)]\!] & = & \overline{x}\, v \\
[\![\tau]\!] & = & \tau
\end{array}$$

As an example, in a version of **VPCCS** where the expressions **E** include natural numbers and operations on those,

$$\begin{array}{ll}
\multicolumn{2}{l}{\underline{a}(\lambda y)x\,.\,\mathbf{case}\ x > 3 : \overline{c}(x+3)} \\
\xrightarrow{\underline{a}\,4} & (\mathbf{case}\ x > 3 : \overline{c}(x+3))[x := 4] \\
= & \mathbf{case}\ E((x > 3)\{4/x\}) : \overline{c}(E((x+3)\{4/x\})) \\
= & \mathbf{case}\ E(4 > 3) : \overline{c}(E(4+3)) \\
= & \mathbf{case}\ \top : \overline{c}7 \\
\xrightarrow{\overline{c}\,7} & \mathbf{0}
\end{array}$$

In our psi semantics, expressions in processes are evaluated when they are closed by reception of variables (e.g. in the first transition above), while Milner simply identifies closed expressions with their values [Mil89, p55f].

Lemma 4.22. *If P is a closed* **VPCCS** *process and $P \xrightarrow{\alpha} P'$, then P' is closed.*

Theorem 4.23. *If P and Q are closed value-passing CCS processes, then*

(1) *if $P \xrightarrow{\alpha} P'$ then $[\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$; and*
(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 A.3. □

As before, this yields a representation theorem.

Theorem 4.24. **VPCCS** *is a representation of the closed agents of value-passing CCS (modulo the modifications described above).*

Proof. We let $\beta \approxeq \alpha$ iff $\alpha = [\![\beta]\!]$.

(1) $[\![\cdot]\!]$ is a simple homomorphism by definition.
(2) $[\![\cdot]\!]$ is a strong operational correspondence by Theorem 4.23. □

To investigate the surjectivity of the encoding, we let $\mathcal{P} = \{P : \textsc{sort}(\mathrm{n}(P)) \subseteq \{\texttt{chan}\}\}$ be the **VPCCS** processes where all fre names are of channel sort.

Lemma 4.25. *If $P \in \mathcal{P}$, then there is a CCS process Q such that $P \sim [\![Q]\!]$.*

Proof. As before, we define an inverse translation $\overline{\cdot}$, that is homomorphic except for

$$\overline{\textbf{case}\ b_1 : P_1 \ [\!]\ \cdots\ [\!]\ b_i : P_i} = (\textbf{if}\ b_1\ \textbf{then}\ \overline{P_1}) + \cdots + (\textbf{if}\ b_i\ \textbf{then}\ \overline{P_i})$$

Using Lemma 3.3, we get $P \sim [\![\overline{P}]\!]$. □

Example 4.26 (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:
$\textsc{match}(z, a, a) = \{z\}$ 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 be substituted for other channel names, and they can be sent and received along channel names.

5. Advanced Data Structures

We here demonstrate that we can accommodate a variety of term structures for data and 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. We focus on our new contribution in the patterns and sorts, and therefore make the following definitions that are common to all the examples (unless explicitly otherwise defined).

$\mathbf{A} = \{\mathbf{1}\}$	$\mathbf{1} \otimes \mathbf{1} = \mathbf{1}$
$\mathbf{C} = \{\top, \bot\}$	$\vdash = \{(\mathbf{1}, \top)\}$
$M \dot{\leftrightarrow} M = \top$	$M \dot{\leftrightarrow} N = \bot$ if $M \neq N$
$\textsc{match}(M, \widetilde{x}, X) = \emptyset$	$\prec = \{(s, s) : s \in \mathcal{S}\}$
$\underline{\propto} = \overline{\propto} = \mathcal{S} \times \mathcal{S}$	$\mathcal{S}_\nu = \mathcal{S}_{\mathcal{N}} = \mathcal{S}$

If t and u are from some term algebra, we write $t \preceq u$ when t is a (non-strict) subterm of u.

5.1. Convergent rewrite systems on terms.

In Example 4.26, the value language consisted of closed terms, with an opaque notion of evaluation. We can instead work with terms containing names and consider deterministic computations specified by a convergent rewrite system. The interesting difference is in which terms are admissible as patterns, and which choices of $\textsc{vars}(X)$ are valid. We first give a general definition and then give a concrete instance in Example 5.1.

Let Σ be a sorted signature with sorts $\mathcal{S}$, and $\cdot\Downarrow$ be normalization with respect to a convergent sort-preserving rewrite system on the nominal term algebra over $\mathcal{N}$ generated by the signature Σ. We let terms M range over the range of $\Downarrow$, i.e., the normal forms. We write ρ for sort-preserving capture-avoiding simultaneous substitutions $\{\widetilde{M}/\widetilde{a}\}$ where every M_i is in normal form; here $n(\rho) = \mathrm{n}(\widetilde{M}, \widetilde{a})$. A term M is stable if for all ρ, $M\rho{\Downarrow} = M\rho$. The patterns are all instances of stable terms, i.e., $X = M\rho$ where M is stable. Such a pattern X can bind any combination of names occurring in M but not in ρ. As an example, any term M is a pattern (since any name x is stable and $M = x\{M/x\}$) that can be used to match the term M itself (since $\emptyset \subseteq \mathrm{n}(x) \setminus \mathrm{n}(M, x) = \emptyset$).

REWRITE($\Downarrow$)

$$\begin{array}{ll} \mathbf{T} = \mathbf{X} = \mathrm{range}(\Downarrow) & \textsc{match}(M, \widetilde{x}, X) = \{\widetilde{L} : M = X\{\widetilde{L}/\widetilde{x}\}\} \\ M[\widetilde{y} := \widetilde{L}] = M\{\widetilde{L}/\widetilde{y}\}{\Downarrow} & \textsc{vars}(X) = \bigcup\{\mathcal{P}(\mathrm{n}(M) \setminus \mathrm{n}(\rho)) : M \text{ stable} \wedge X = M\rho\} \end{array}$$

We need to show that the patterns are closed under substitution, including preservation of $\textsc{vars}$ (cf. Definition 2.4), and that matching satisfies the criteria of Definition 2.5. Since any term is a pattern, the patterns are closed under substitution. Since term substitution $\{\cdot/\cdot\}$ and normalization $\Downarrow$ are both sort-preserving, term and pattern substitution $[\cdot := \cdot]$ is also sort-preserving.

To show preservation of pattern variables, assume that $\widetilde{x} \in \textsc{vars}(X)$ is a tuple of distinct names. By definition there are M and ρ such that $X = M\rho$ with M stable and $\widetilde{x} \subseteq \mathrm{n}(M) \setminus \mathrm{n}(\rho)$. Assume that $\widetilde{x}\#\sigma$; then $X\sigma = (M\rho)\sigma = M(\sigma \circ \rho)$ with $\widetilde{x}\#\sigma \circ \rho$, so $\widetilde{x} \in \textsc{vars}(X\sigma)$.

For the criteria of Definition 2.5, additionally assume that $\widetilde{L} \in \textsc{match}(N, \widetilde{x}, X)$ and let $\sigma = [\widetilde{x} := \widetilde{L}]$. Since $\{\widetilde{L}/\widetilde{x}\}$ is well-sorted, so is $[\widetilde{x} := \widetilde{L}]$. We also immediately have $\mathrm{n}(\widetilde{L}) = \mathrm{n}(N) \cup (\mathrm{n}(X) \setminus \widetilde{x})$, and alpha-renaming of matching follows from the same property for term substitution.

Example 5.1 (Peano arithmetic)**.** As a simple instance of **REWRITE**($\Downarrow$), we may consider Peano arithmetic. The rewrite rules for addition (below) induce a convergent rewrite system $\Downarrow^{\mathrm{Peano}}$, where the stable terms are those that do not contain any occurrence of `plus`.

PEANO

Everything as in **REWRITE($\Downarrow$)** except:
$\mathcal{S} = \{\mathtt{nat}, \mathtt{chan}\}$
$\Sigma = \{\mathtt{zero} : \mathtt{nat}, \quad \mathtt{succ} : \mathtt{nat} \to \mathtt{nat} \quad \mathtt{plus} : \mathtt{nat} \times \mathtt{nat} \to \mathtt{nat}\}$
$\mathtt{plus}(K, \mathtt{zero}) \to K \qquad \mathtt{plus}(K, \mathtt{succ}(M)) \to \mathtt{plus}(\mathtt{succ}(K), M)$
$\textsc{vars}(\mathtt{succ}^n(a)) = \{\emptyset, \{a\}\} \qquad \textsc{vars}(M) = \{\emptyset\}$ otherwise

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 **REWRITE**($\Downarrow^{\mathrm{Peano}}$) has one visible transition, with the label $\overline{c}\,4$. In particular, the object of the label is $\mathtt{plus}(3, y)[y := 1] = \mathtt{plus}(3, y)\{1/y\}{\Downarrow^{\mathrm{Peano}}} = 4$.

5.2. **Symmetric cryptography.** We can also consider variants of **REWRITE**($\Downarrow$), such as a simple Dolev-Yao style [DY83] 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.

The rewrite rule describing decryption $\mathtt{dec}(\mathtt{enc}(M,K),K) \to M$ induces a convergent rewrite system $\Downarrow^{\mathrm{enc}}$, where the terms not containing $\mathtt{dec}$ are stable. The construction of **REWRITE**($\Downarrow$) 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 still 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. Below we give the formal definition (recall that $\preceq$ is the subterm preorder).

SYMSPI

Everything as in **REWRITE($\Downarrow^{\mathrm{enc}}$)** except:
$\mathcal{S} = \{\mathtt{message}, \mathtt{key}\}$
$\Sigma = \{\mathtt{enc} : \mathtt{message} \times \mathtt{key} \to \mathtt{message},\quad \mathtt{dec} : \mathtt{message} \times \mathtt{key} \to \mathtt{message}\}$
$\mathtt{dec}(\mathtt{enc}(M,K),K) \to M$
$\textsc{vars}(X) = \mathcal{P}(n(X) \setminus \{a : a \preceq \mathtt{dec}(Y_1,Y_2) \preceq X \vee (a \preceq Y_2 \wedge \mathtt{enc}(Y_1,Y_2) \preceq X)\})$

The proof of the conditions of Definition 2.4 and Definition 2.5 for patterns is the same as for **REWRITE**($\cdot$) in Section 5.1 above.

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 $\overline{c}\;M$: the subagent

$$\underline{a}(\lambda y)\mathtt{enc}(y,k)\,.\,\overline{c}\;\mathtt{dec}(y,l) \xrightarrow{\underline{a}\;\mathtt{enc}(\mathtt{enc}(M,l),k)} \overline{c}\;\mathtt{dec}(y,l)[y := \mathtt{enc}(M,l)]$$

since $\mathtt{enc}(M,l) \in \textsc{match}(\mathtt{enc}(\mathtt{enc}(M,l),k), y, \mathtt{enc}(y,k))$. The resulting process is

$$\overline{c}\;\mathtt{dec}(y,l)[y := \mathtt{enc}(M,l)] = \overline{c}\;\mathtt{dec}(y,l)\{{}^{\mathtt{enc}(M,l)}\!/_{y}\}\Downarrow = \overline{c}\;\mathtt{dec}(\mathtt{enc}(M,l),l)\Downarrow = \overline{c}\;M.$$

5.3. **Asymmetric cryptography.** A more advanced version of Section 5.2 is the treatment of data in the pattern-matching spi-calculus [HJ06], to which we refer for more examples and motivations of the definitions below. The calculus uses asymmetric encryption, and 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 specialized modelling features can be presented in a form that is very close to the original.

We start from the term algebra T_Σ over the unsorted signature

$$\Sigma = \{(),\ (\cdot,\cdot),\ \mathtt{eKey}(\cdot),\ \mathtt{dKey}(\cdot),\ \mathtt{enc}(\cdot,\cdot)\ \mathtt{enc}^{-1}(\cdot,\cdot)\}$$

The $\mathtt{eKey}(M)$ and $\mathtt{dKey}(M)$ constructions represent the encryption and decryption parts of the key pair M, respectively. The operation $\mathtt{enc}^{-1}(M,N)$ is encryption of M with the inverse of the decryption key N, which is not an implementable operation but only permitted to occur in patterns. We add a sort system on T_Σ with sorts $\mathcal{S} = \{\mathtt{impl}, \mathtt{pat}, \bot\}$, 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. Names stand for implementable terms, so we let $\mathcal{S}_{\mathcal{N}} = \{\mathtt{impl}\}$. Substitution is defined homomorphically on the term algebra, except to avoid unimplementable subterms on the form $\mathtt{enc}^{-1}(M, \mathtt{dKey}(N))$.

In order to define $\textsc{vars}(X)$, we write $\widetilde{M} \Vdash \widetilde{N}$ if all $N_i \in \widetilde{N}$ can be deduced from $\widetilde{M}$ in the Dolev-Yao message algebra (i.e., using cryptographic operations such as encryption and decryption). For the precise definition, see [HJ06]. The definition of $\textsc{vars}(X)$ below allows to bind a set S of names only if all names in S can be deduced from the message term X using the other names occurring in X. This excludes binding an unknown key, like in Example **??**.

PMSPI

$$\begin{array}{lll}
\mathbf{T} = \mathbf{X} = T_\Sigma & \mathcal{S} = \{\mathtt{impl}, \mathtt{pat}, \bot\} & \mathcal{S}_\mathcal{N} = \{\mathtt{impl}\} \\
\prec\, = \overline{\propto} = \{(\mathtt{impl}, \mathtt{impl})\} & \underline{\propto} = \{(\mathtt{impl}, \mathtt{impl}), (\mathtt{impl}, \mathtt{pat})\} &
\end{array}$$

$$\begin{array}{l}
\textsc{sort}(M) = \mathtt{impl} \text{ if } \forall N_1, N_2.\ \mathtt{enc}^{-1}(N_1, N_2) \not\preceq M \\
\textsc{sort}(M) = \bot \text{ if } \exists N_1, N_2.\ \mathtt{enc}^{-1}(N_1, \mathtt{dKey}(N_2)) \preceq M \\
\textsc{sort}(M) = \mathtt{pat} \text{ otherwise} \\
\textsc{match}(M, \widetilde{x}, X) = \{\widetilde{L}\ :\ M = X[\widetilde{x} := \widetilde{L}]\} \\
\textsc{vars}(X) = \{S \subseteq \mathrm{n}(X)\ :\ ((\mathrm{n}(X) \setminus S) \cup \{X\}) \Vdash S\}
\end{array}$$

$$\begin{array}{rcll}
x[\widetilde{y} := \widetilde{L}] &=& L_i & \text{if } y_i = x \\
x[\widetilde{y} := \widetilde{L}] &=& x & \text{otherwise.} \\
\mathtt{enc}^{-1}(M_1, M_2)[\widetilde{y} := \widetilde{L}] &=& \mathtt{enc}(M_1[\widetilde{y} := \widetilde{L}], \mathtt{eKey}(N)) & \text{when } M_2[\widetilde{y} := \widetilde{L}] = \mathtt{dKey}(N) \\
f(M_1, \ldots, M_n)[\widetilde{y} := \widetilde{L}] &=& f(M_1[\widetilde{y} := \widetilde{L}], \ldots, M_n[\widetilde{y} := \widetilde{L}]) & \text{otherwise.}
\end{array}$$

As an example, consider the following transitions in **PMSPI**:

$$\begin{array}{l}
(\nu a, k, l)(\ \overline{a}\ \mathtt{enc}(\mathtt{dKey}(l), \mathtt{eKey}(k)).\overline{a}\ \mathtt{enc}(M, \mathtt{eKey}(l)) \\
\qquad\qquad \mid \underline{a}(\lambda y)\mathtt{enc}(y, \mathtt{eKey}(k)) \,.\, \underline{a}(\lambda z)\mathtt{enc}^{-1}(z, y) \,.\, \overline{c}\ z) \\
\qquad\qquad \xrightarrow{\tau} (\nu a, k, l)(\overline{a}\ \mathtt{enc}(M, \mathtt{eKey}(l)) \mid \underline{a}(\lambda z)\mathtt{enc}(z, \mathtt{eKey}(l)) \,.\, \overline{c}\ z) \\
\qquad\qquad \xrightarrow{\tau} (\nu a, k, l)\overline{c}\ M.
\end{array}$$

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 2.4 (Substitution), since $\mathtt{impl} \leq \mathtt{pat}$ (implementable terms can be used as channels or messages whenever patterns can be).

Terms (and patterns) are trivially closed under substitution. All terms in the domain of a well-sorted substitution have sort $\mathtt{impl}$, so well-sorted substitutions cannot introduce subterms of the forms $\mathtt{enc}^{-1}(N_1, N_2)$ or $\mathtt{enc}^{-1}(N_1, \mathtt{dKey}(N_2))$ where none existed; thus $\textsc{sort}(M\sigma) \leq \textsc{sort}(M)$ as required by Definition 2.4.

To show preservation of pattern variables, we have that $((\mathrm{n}(X) \setminus \widetilde{x}) \cup \{X\}) \Vdash \widetilde{x}$ implies that $((\mathrm{n}(X\sigma) \setminus \widetilde{x}) \cup \{X\sigma\}) \Vdash \widetilde{x}$ whenever $x \# \sigma$, by induction on $\Vdash$. Add definition, of $\Vdash$, give IH? The requisites on matching (Definition 2.5) follow from those on substitution.

5.4. **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 extended with the erratic choice operator $\cdot \,[\!]\, \cdot$ and the reduction rule $M_1 \,[\!]\, M_2 \to M_i$ if $i \in \{1, 2\}$. Due to non-determinism and partiality, evaluation cannot be part of the substitution function. Instead, we define the $\textsc{match}$ function to collect all evaluations of the received term, which are non-deterministically selected from by the $\textsc{In}$ 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 $\mathcal{R} ::= [\,] \mid \mathcal{R}\ M \mid (\boldsymbol{\lambda} x.M)\ \mathcal{R}$ (we here use the boldface $\boldsymbol{\lambda}$ rather than the λ used in input prefixes). Reduction $\rightarrow$ is the smallest pre-congruence for reduction contexts that contain the rules for β-reduction ($\boldsymbol{\lambda} x.M\ N \rightarrow M[x := N]$) and $\cdot [\!] \cdot$ (see above). We use the single-name patterns of Example 2.6, but include evaluation in matching.

NDLAM

$\mathcal{S} = \{s\}$ $\qquad \mathbf{X} = \mathcal{N}$

$M ::= a \mid M\ M \mid \boldsymbol{\lambda} x.M \mid M \mathbin{[\!]} M$ $\qquad$ where x binds into M in $\boldsymbol{\lambda} x.M$

$\textsc{match}(M, x, x) = \{N \ :\ M \rightarrow^* N \not\rightarrow\}$

As an example, the agent $P \overset{def}{=} (\nu a)(\underline{a}(y)\,.\,\overline{c}\ y\,.\,\mathbf{0} \mid \overline{a}\ ((\boldsymbol{\lambda} x.x\ x) \mathbin{[\!]} (\boldsymbol{\lambda} x.x))\,.\,\mathbf{0})$ has the following transitions:

$$P \xrightarrow{\tau} (\nu a)(\overline{c}\ \boldsymbol{\lambda} x.xx\,.\,\mathbf{0} \mid \mathbf{0}) \xrightarrow{\overline{c}\ \boldsymbol{\lambda} x.xx} \mathbf{0}$$
$$P \xrightarrow{\tau} (\nu a)(\overline{c}\ \boldsymbol{\lambda} x.x\,.\,\mathbf{0} \mid \mathbf{0}) \xrightarrow{\overline{c}\ \boldsymbol{\lambda} x.x} \mathbf{0}.$$

6. Conclusions and further work

We have described two features that taken together significantly improve 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. Using these features we have provided representations of other process calculi, ranging from the simple polyadic pi-calculus to the spi-calculus and non-deterministic computations, in the psi-calculi framework. The critera for representation (rather than encoding) are stronger than standard correspondences e.g. by Gorla, and mean that the psi-calculus and the calculus represented by it are for all practical purposes one and the same.

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 [BGRV13], 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 [PBRÅP13] and reliable broadcast psi-calculi [ÅPBP$^+$13]. As mentioned in Section 3.1, further developments in Nominal Isabelle are needed for mechanizing theories with arbitrary but fixed sortings.

References

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

[ÅP14] Johannes Åman Pohjola. Isabelle proof scripts for sorted psi-calculi. Available at `http://www.it.uu.se/research/group/mobility/theorem/sortedPsi.tar.gz`, 2014.

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

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

[BGRV13] Johannes Borgström, Ramūnas Gutkovas, Ioana Rodhe, and Björn Victor. A parametric tool for applied process calculi. In *Proc. 13th International Conference on Application of Concurrency to System Design (ACSD'13).* IEEE, 2013.

[BJPV11] Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Psi-calculi: a framework for mobile processes with nominal data and logic. *LMCS*, 7(1:11), 2011.

[Bla11] Bruno Blanchet. Using Horn clauses for analyzing security protocols. In Véronique Cortier and Steve Kremer, editors, *Formal Models and Techniques for Analyzing Security Protocols*, volume 5 of *Cryptology and Information Security Series*, pages 86–111. IOS Press, March 2011.

[CGK+13] Sjoerd Cranen, Jan Friso Groote, Jeroen J. A. Keiren, Frank P. M. Stappers, Erik P. de Vink, Wieger Wesselink, and Tim A. C. Willemse. An overview of the mCRL2 toolset and its recent advances. In Nir Piterman and Scott A. Smolka, editors, *TACAS*, volume 7795 of *Lecture Notes in Computer Science*, pages 199–213. Springer, 2013.

[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.

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

[EOW07] Burak Emir, Martin Odersky, and John Williams. Matching objects with patterns. In *Proceedings of the 21st European Conference on Object-Oriented Programming*, ECOOP'07, pages 273–298, Berlin, Heidelberg, 2007. Springer-Verlag.

[FG96] Cédric Fournet and Georges Gonthier. The reflexive CHAM and the join-calculus. In *Proc. POPL*, pages 372–385, 1996.

[FGM05] Cédric Fournet, Andrew D. Gordon, and Sergio Maffeis. A type discipline for authorization policies. In Mooly Sagiv, editor, *Proc. of ESOP 2005*, volume 3444 of *LNCS*, pages 141–156. Springer, 2005.

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

[Gor10] Daniele Gorla. Towards a unified approach to encodability and separation results for process calculi. *Information and Computation*, 208(9):1031–1053, 2010.

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

[GWGJ10] Thomas Given-Wilson, Daniele Gorla, and Barry Jay. Concurrent pattern calculus. In Cristian Calude and Vladimiro Sassone, editors, *Theoretical Computer Science*, volume 323 of *IFIP Advances in Information and Communication Technology*, pages 244–258. Springer, 2010.

[HJ06] Christian Haack and Alan Jeffrey. Pattern-matching spi-calculus. *Information and Computation*, 204(8):1195–1263, 2006.

[Hon93] Kohei Honda. Types for dyadic interaction. In Eike Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993.

[Hüt11] Hans Hüttel. Typed psi-calculi. In Joost-Pieter Katoen and Barbara König, editors, *CONCUR 2011 – Concurrency Theory*, volume 6901 of *LNCS*, pages 265–279. Springer, 2011.

[HV13] Hans Hüttel and Vasco T Vasconcelos. The foundations of behavioural types. State-of-the art report of WG1 of the BETTY project (EU COST Action IC1201). To appear, 2013.

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

[Kri09] Neelakantan R. Krishnaswami. Focusing on pattern matching. In *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '09, pages 366–378, New York, NY, USA, 2009. ACM.

[LSD11] Yang Liu, Jun Sun, and Jin Song Dong. PAT 3: An extensible architecture for building multi-domain model checkers. In Tadashi Dohi and Bojan Cukic, editors, *ISSRE '11*, pages 190–199. IEEE, 2011.

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

[Mil93] Robin Milner. The polyadic π-calculus: A tutorial. In Friedrich L. Bauer, Wilfried Brauer, and Helmut Schwichtenberg, editors, *Logic and Algebra of Specification*, volume 94 of *Series F*. NATO ASI, Springer, 1993.

[PBRÅP13] Joachim Parrow, Johannes Borgström, Palle Raabjerg, and Johannes Åman Pohjola. Higher-order psi-calculi. *Mathematical Structures in Computer Science*, FirstView, June 2013.

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

[San93] Davide Sangiorgi. *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).

[SLDC09] Jun Sun, Yang Liu, Jin Song Dong, and Chunqing Chen. Integrating specification and programs for system modeling and verification. In *TASE '09*, pages 127–135. IEEE Computer Society, 2009.

[SNM07] Don Syme, Gregory Neverov, and James Margetson. Extensible pattern matching via a lightweight language extension. In *Proceedings of the 12th ACM SIGPLAN International Conference on Functional Programming*, ICFP '07, pages 29–40, New York, NY, USA, 2007. ACM.

[SS05] 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 *LNCS*, pages 146–178. Springer Berlin Heidelberg, 2005.

[SW01] Davide Sangiorgi and David Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

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

Appendix A. Full proofs for Section 4

The following is full proofs of Section 4; 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 4, as well as the definitions, notation and terminology of Sangiorgi [San93], Carbone and Maffeis [CM03], and Milner [Mil89], 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 $\mathrm{bn}(\alpha)\#Q$ instead of $\mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset$.

A.1. **Polyadic Pi-Calculus.**

We follow the exposition of Polyadic Pi-Calculus given by Sangiorgi in [San93] 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 4.1.

Definition A.1 (Polyadic Pi-Calculus to **PPi**).
Agents:

$$\begin{array}{rcl}
[\![P + Q]\!] & = & \mathbf{case}\ \top : [\![P]\!]\ [\,]\ \top : [\![Q]\!] \\
[\![[x = y]P]\!] & = & \mathbf{case}\ x = y : [\![P]\!] \\
[\![x(\tilde{y}).P]\!] & = & \underline{x}(\lambda\widetilde{y})\langle\widetilde{y}\rangle.[\![P]\!] \\
[\![\overline{x}\langle\tilde{y}\rangle.P]\!] & = & \overline{x}\langle\tilde{y}\rangle.[\![P]\!] \\
[\![0]\!] & = & 0 \\
[\![P \mid Q]\!] & = & [\![P]\!] \mid [\![Q]\!] \\
[\![\nu x P]\!] & = & (\nu x)[\![P]\!] \\
[\![!P]\!] & = & ![\![P]\!]
\end{array}$$

Actions:

$$\begin{array}{rcl}
[\![(\nu\widetilde{y'})\overline{z}\langle\tilde{y}\rangle]\!] & = & \overline{z}\ (\nu\widetilde{y'})\ \langle\widetilde{y}\rangle \\
[\![x\langle\tilde{z}\rangle]\!] & = & \underline{x}\ \langle\tilde{z}\rangle \\
[\![\tau]\!] & = & \tau
\end{array}$$

In output action $\widetilde{y'}$ do not bind into z.

Definition A.2 (**PPi** to Polyadic Pi-Calculus).
Process:

$$\begin{array}{rcl}
\overline{(\!|\mathbf{1}|\!)} & = & \mathbf{0} \\
\overline{\mathbf{0}} = \overline{\mathbf{case}} & = & \mathbf{0} \\
\overline{\mathbf{case}\ \varphi_1 : P_1\ [\,]\ \ldots\ [\,]\ \varphi_n : P_n} & = & \overline{\varphi_1 : P_1} + \cdots + \overline{\varphi_n : P_n} \\
\overline{!P} & = & !\overline{P} \\
\overline{(\nu x)P} & = & \nu x\overline{P} \\
\overline{P \mid Q} & = & \overline{P} \mid \overline{Q} \\
\overline{\underline{x}(\lambda\widetilde{y})\langle\widetilde{y}\rangle.P} & = & x(\tilde{y}).\overline{P} \\
\overline{\overline{x}\langle\tilde{y}\rangle.P} & = & \overline{x}\langle\tilde{y}\rangle.\overline{P}
\end{array}$$

Case clause:

$$\begin{array}{rcl}
\overline{x = y : P} & = & [x = y]\overline{P} \\
\overline{\top : P} & = & \overline{P}
\end{array}$$

We prove that substitution function distributes over the encoding function. We use this auxiliary result in some of the following theorems.

Lemma A.3. $\llbracket P \rrbracket[\tilde{y} := \tilde{z}] = \llbracket P\{\tilde{z}/\tilde{y}\} \rrbracket$

Proof. By induction on P. We consider only the agents where $bn(P) \cap fn(P)\{\tilde{z}/\tilde{y}\} = \emptyset$ as in Definition 2.1.1 in [San93] on page 21. We show the interesting cases of the substitution application as others are just homomorphic.

- case $P = P' + Q$.

$$\begin{array}{rcll}
\llbracket P' + Q \rrbracket[\tilde{y} := \tilde{z}] & = & \textbf{case}\ \top[\tilde{y} := \tilde{z}] : \llbracket P' \rrbracket[\tilde{y} := \tilde{z}] \mathbin{[]} \top[\tilde{y} := \tilde{z}] : \llbracket Q \rrbracket[\tilde{y} := \tilde{z}] & \\
 & = & \textbf{case}\ \top : \llbracket P' \rrbracket[\tilde{y} := \tilde{z}] \mathbin{[]} \top : \llbracket Q \rrbracket[\tilde{y} := \tilde{z}] & \\
 & = & \textbf{case}\ \top : \llbracket P'\{\tilde{z}/\tilde{y}\} \rrbracket \mathbin{[]} \top : \llbracket Q\{\tilde{z}/\tilde{y}\} \rrbracket & \text{(IH)} \\
 & = & \llbracket P'\{\tilde{z}/\tilde{y}\} + Q\{\tilde{z}/\tilde{y}\} \rrbracket & \\
 & = & \llbracket (P' + Q)\{\tilde{z}/\tilde{y}\} \rrbracket &
\end{array}$$

- case $P = [x = y]Q$.

$$\begin{array}{rcll}
\llbracket [x = y]Q \rrbracket[\tilde{y} := \tilde{z}] & = & \textbf{case}\ x[\tilde{y} := \tilde{z}] = y[\tilde{y} := \tilde{z}] : \llbracket Q \rrbracket[\tilde{y} := \tilde{z}] & \\
 & = & \textbf{case}\ x[\tilde{y} := \tilde{z}] = y[\tilde{y} := \tilde{z}] : \llbracket Q\{\tilde{z}/\tilde{y}\} \rrbracket & \text{(IH)} \\
 & = & [x\{\tilde{z}/\tilde{y}\} = y\{\tilde{z}/\tilde{y}\}]\llbracket Q\{\tilde{z}/\tilde{y}\} \rrbracket & \\
 & = & \llbracket ([x = y]Q)\{\tilde{z}/\tilde{y}\} \rrbracket &
\end{array}$$

- case $P = a(\tilde{x}).Q$

$$\begin{array}{rcll}
\llbracket a(\tilde{x}).Q \rrbracket[\tilde{y} := \tilde{z}] & = & \underline{a[\tilde{y} := \tilde{z}]}(\lambda\widetilde{x})\langle\widetilde{x}\rangle.\llbracket Q \rrbracket[\tilde{y} := \tilde{z}] & \text{(From assumption } \tilde{x}\#[\tilde{y} := \tilde{z}]) \\
 & = & \underline{a[\tilde{y} := \tilde{z}]}(\lambda\widetilde{x})\langle\widetilde{x}\rangle.\llbracket Q\{\tilde{z}/\tilde{y}\} \rrbracket & \text{(IH)} \\
 & = & a\{\tilde{z}/\tilde{y}\}(\tilde{x}).\llbracket Q\{\tilde{z}/\tilde{y}\} \rrbracket & \\
 & = & \llbracket (a(\tilde{x}).Q)\{\tilde{z}/\tilde{y}\} \rrbracket &
\end{array}$$

□

The following is proof of the strong operational correspondence. The labeled semantics of polyadic pi-calculus can be found on page 30 of [San93].

Proof of Theorem 4.4.

(1) By induction on the length of the derivation of P'. We have the following cases to check by considering the last rule applied to derive P'.

ALP:
Trivial since in Psi-calculi agents are identified up to alpha equivalence.

OUT:
Assume $\overline{x}\langle\tilde{y}\rangle.P \xrightarrow{\overline{x}\langle\tilde{y}\rangle} P$ and $\alpha \in \{\overline{x}\ \langle\tilde{y}\rangle\} = \llbracket \overline{x}\langle\tilde{y}\rangle \rrbracket$. Since $\mathbf{1} \vdash x \dot{\leftrightarrow} x$ and $\llbracket \overline{x}\ \langle\tilde{y}\rangle.P \rrbracket = \overline{x}\ \langle\tilde{y}\rangle.\llbracket P \rrbracket$ and $\alpha = \overline{x}\ \langle\tilde{y}\rangle$, we can derive $\overline{x}\ \langle\tilde{y}\rangle.\llbracket P \rrbracket \xrightarrow{\overline{x}\ \langle\tilde{y}\rangle} \llbracket P \rrbracket$.

INP:
Assume $x(\tilde{y}).P \xrightarrow{x\langle\tilde{z}\rangle} P\{\tilde{z}/\tilde{y}\}$ with $\tilde{z} : \tilde{y}$ and $\alpha \in \llbracket \beta \rrbracket = \{\underline{x}\ \langle\tilde{z}\rangle\}$. We compute that $\llbracket x(\tilde{y}).P \rrbracket = \underline{x}(\lambda\widetilde{y})\langle\widetilde{y}\rangle.\llbracket P \rrbracket$ and $\widetilde{z} \in \textsc{match}(\langle\widetilde{z}\rangle, \widetilde{y}, \langle\widetilde{y}\rangle)$. Using this and $\mathbf{1} \vdash x \dot{\leftrightarrow} x$ we can derive $\underline{x}(\lambda\widetilde{y})\langle\widetilde{y}\rangle.\llbracket P \rrbracket \xrightarrow{\underline{x}\ \langle\tilde{z}\rangle} \llbracket P \rrbracket[\widetilde{y} := \widetilde{z}]$ with the In rule. By applying Lemma A.3 completes the proof.

SUM:
Assume $P + Q \xrightarrow{\beta} P'$ and $\alpha \in \llbracket \beta \rrbracket$, and also $P \xrightarrow{\beta} P'$. From induction hypothesis we have that for every $\alpha \in \llbracket \beta \rrbracket$, $\llbracket P \rrbracket \xrightarrow{\alpha} \llbracket P' \rrbracket$. Thus we can derive $\textbf{case}\ \top : \llbracket P \rrbracket \mathbin{[]} \top : \llbracket Q \rrbracket \xrightarrow{\alpha} \llbracket P' \rrbracket$ with the Case rule for every $\alpha \in \llbracket \beta \rrbracket$.

PAR:

Assume $P \mid Q \xrightarrow{\beta} P' \mid Q$ and $\alpha \in \llbracket\beta\rrbracket$. We also assume $P \xrightarrow{\beta} P'$ with $bn(\beta) \cap fn(Q) = \emptyset$. From induction hypothesis, we get that for every $\alpha \in \llbracket\beta\rrbracket$, $\llbracket P\rrbracket \xrightarrow{\alpha} \llbracket P'\rrbracket$. From assumption follows that $bn(\alpha)\#\llbracket Q\rrbracket$ for any $\alpha \in \llbracket\beta\rrbracket$. By applying the PAR rule, we obtain the required transition $\llbracket P\rrbracket \mid \llbracket Q\rrbracket \xrightarrow{\alpha} \llbracket P'\rrbracket \mid \llbracket Q\rrbracket$.

COM:

Assume $P \mid Q \xrightarrow{\tau} \nu\tilde{y}'(P' \mid Q')$ with $\tilde{y}' \cap fn(Q) = \emptyset$. We also assume $P \xrightarrow{(\nu\tilde{y'})\overline{x}\langle\tilde{y}\rangle} P'$ and $Q \xrightarrow{x\langle\tilde{y}\rangle} Q'$. From induction hypothesis, we have that for every $\alpha' \in \llbracket(\nu\tilde{y'})\overline{x}\langle\tilde{y}\rangle\rrbracket$ and $\alpha'' \in \llbracket x\langle\tilde{y}\rangle\rrbracket$, $\llbracket P\rrbracket \xrightarrow{\alpha'} \llbracket P'\rrbracket$ and $\llbracket Q\rrbracket \xrightarrow{\alpha''} \llbracket Q'\rrbracket$ Moreover, we note that $\mathbf{1} \vdash x \dot{\leftrightarrow} x$ and $\tilde{y}'\#\llbracket Q\rrbracket$. Finally, we choose α' and α'' and choose alpha-variants of the frames of $\llbracket P\rrbracket$ and $\llbracket Q\rrbracket$ which are sufficiently fresh to allow the derivation $\llbracket P\rrbracket \mid \llbracket Q\rrbracket \xrightarrow{\tau} (\nu\widetilde{y'})(\llbracket P'\rrbracket \mid \llbracket Q'\rrbracket)$ with the COM rule.

MATCH:

Assume $[x = x]P \xrightarrow{\beta} P'$ and $\alpha \in \llbracket\beta\rrbracket$. We also assume $P \xrightarrow{\beta} P'$. From induction hypothesis we acquire that $\llbracket P\rrbracket \xrightarrow{\alpha} \llbracket P'\rrbracket$. Since $\mathbf{1} \vdash x = x$ and **case** $x = x : \llbracket P\rrbracket = \llbracket[x = x]P\rrbracket$, we derive **case** $x = x : \llbracket P\rrbracket \xrightarrow{\alpha} \llbracket P'\rrbracket$ with the CASE rule.

REP:

Assume $!P \xrightarrow{\beta} P'$ and $\alpha \in \llbracket\beta\rrbracket$. Moreover, assume $P \mid\, !P \xrightarrow{\beta} P'$ and hence from induction hypothesis $\llbracket P \mid\, !P\rrbracket \xrightarrow{\alpha} \llbracket P'\rrbracket$. We compute $\llbracket P\rrbracket \mid\, !\llbracket P\rrbracket = \llbracket P \mid\, !P\rrbracket$ and apply the REP rule to obtain $!\llbracket P\rrbracket \xrightarrow{\alpha} \llbracket P'\rrbracket$.

RES:

Assume $\nu xP \xrightarrow{\beta} \nu xP'$ where $x \notin n(\beta)$ and $\alpha \in \llbracket\beta\rrbracket$. We also assume $P \xrightarrow{\beta} P'$, to acquire from induction hypothesis $\llbracket P\rrbracket \xrightarrow{\alpha} \llbracket P'\rrbracket$. Now by obtaining $x\#\alpha$ from assumption and computing $\llbracket\nu xP\rrbracket = (\nu x)\llbracket P\rrbracket$, we derive $(\nu x)\llbracket P\rrbracket \xrightarrow{\alpha} (\nu x)\llbracket P'\rrbracket$ with the SCOPE rule.

OPEN:

Let $\beta = (\nu x, \tilde{y'})\overline{z}\langle\tilde{y}\rangle$. Assume $\nu xP \xrightarrow{\beta} P'$ and $x \neq z, x \in \tilde{y} - \tilde{y'}$ and $\alpha \in \llbracket\beta\rrbracket = \{\overline{z}\,(\nu\widetilde{y''})\,\langle\tilde{y}\rangle \;:\; \tilde{y}'' = \pi \cdot x, \tilde{y}'\}$. From induction hypothesis, we get that for every $\alpha' \in \llbracket(\nu\tilde{y}')\overline{z}\langle\tilde{y}\rangle\rrbracket = \{\overline{z}\,(\nu\widetilde{y''})\,\langle\tilde{y}\rangle \;:\; \tilde{y}'' = \pi \cdot \tilde{y'}\}$ we can derive $\llbracket P\rrbracket \xrightarrow{\alpha'} \llbracket P'\rrbracket$. We choose $\alpha' = \overline{z}\,(\nu\widetilde{y'})\,\tilde{y}$ and by having $\llbracket\nu xP\rrbracket = (\nu x)\llbracket P\rrbracket$ derive, $(\nu x)\llbracket P\rrbracket \xrightarrow{\overline{z}\,(\nu x,\tilde{y'})\,\langle\tilde{y}\rangle} \llbracket P'\rrbracket$ with the OPEN rule. The side conditions of OPEN, $x\#\tilde{y}', z$ and $x \in n(\tilde{y})$, follow from assumptions.

From the assumption $\alpha \in \llbracket\beta\rrbracket$, it follows that, for any permutation π, α is of the form $\overline{z}\,(\nu\pi \cdot x, \widetilde{y'})\,\langle\tilde{y}\rangle$. By applying Lemma 4.3, we get the required α and transition $(\nu x)\llbracket P\rrbracket \xrightarrow{\alpha} \llbracket P'\rrbracket$. And this concludes this proof case.

(2) We now show that if $[\![P]\!] \xrightarrow{\alpha} P''$ then $P \xrightarrow{\beta} P'$ where $\alpha \in [\![\beta]\!]$ and $[\![P']\!] = P''$. We proceed by by induction on the length of the derivation of P''. We only write down the interesting cases:

Case:

Assume $[\![P]\!] \xrightarrow{\alpha} P''$. Because P'' is derived with the Case rule, $[\![P]\!]$ is of the form $\mathbf{case}\ \tilde{\varphi} : \tilde{P}$. Since $P_C = \mathbf{case}\ \widetilde{\varphi} : \widetilde{P}$ is in the range of $[\![\cdot]\!]$, either $P_C = \top : [\![P]\!] \mathbin{[\,]} \top : [\![Q]\!]$, $P_C = \top : [\![Q]\!] \mathbin{[\,]} \top : [\![P]\!]$ or $P_C = \mathbf{case}\ x = y : [\![P]\!]$. We proceed by case analysis:

(a) When $P_C = \top : [\![P]\!] \mathbin{[\,]} \top : [\![Q]\!]$, we note that $[\![P + Q]\!] = P_C$ and imitate the derivation of P'' from P_C with the derivation $P + Q \xrightarrow{\beta} P'$, using the **SUM** rule and the fact obtained from induction hypothesis $\alpha \in [\![\beta]\!]$.

(b) The case when $P_C = \top : [\![Q]\!] \mathbin{[\,]} \top : [\![P]\!]$ is symmetric to the previous case.

(c) When $P_C = \mathbf{case}\ x = y : [\![P]\!]$, since $\mathbf{1} \vdash x = y$ 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 $[x = x]P \xrightarrow{\beta} P'$, using the **MATCH** rule and the fact obtained from induction hypothesis $\alpha \in [\![\beta]\!]$.

Open:

Assume $[\![P]\!] \xrightarrow{\overline{z}\,(\nu\tilde{y} \cup \{x\})\,\langle\tilde{y}'\rangle} P''$. Because P'' is derived with the Open rule, $[\![P]\!]$ is of the form $(\nu x)R$. Since $(\nu x)R$ is in the range of $[\![\cdot]\!]$, $P = \nu x R'$ where $R = [\![R']\!]$. From induction hypothesis, we have that $R \xrightarrow{\overline{z}\,(\nu\tilde{y})\,\langle\tilde{y}'\rangle} P''$ and $\overline{z}\,(\nu\tilde{y})\,\langle\tilde{y}'\rangle \in [\![\beta']\!]$ and $R' \xrightarrow{\beta'} P'$ and lastly $[\![P']\!] = P''$. Thus we use $\beta' = (\nu\tilde{y})\overline{z}\langle\tilde{y}'\rangle$ as it gives us $\overline{z}\,(\nu\tilde{y})\,\langle\tilde{y}'\rangle \in [\![\beta']\!]$ to derive using the rule **OPEN**, $\nu x R' \xrightarrow{(\nu x, \tilde{y})\overline{z}\langle\tilde{y}'\rangle} P'$. Clearly $\overline{z}\,(\nu\tilde{y} \cup \{x\})\,\langle\tilde{y}'\rangle \in [\![(\nu x, \tilde{y})\overline{z}\langle\tilde{y}'\rangle]\!]$ for every insertion of x.

From the strong operational correspondence, we obtain full abstraction. We use Sangiorgi's the definition of bisimulation and congruence of polyadic pi-calculus which can be found in [San93] on page 42.

Theorem A.4. *For polyadic-pi calculus agents P and Q we have $P \sim_e^c Q$ iff $[\![P]\!] \sim [\![Q]\!]$*

Proof. Direction $\Leftarrow$. Assume $[\![P]\!] \sim [\![Q]\!]$. We claim that the relation $\mathcal{R} = \{(P, Q) : [\![P]\!] \sim [\![Q]\!]\}$ is an *early congruence* in the polyadic pi-calculus.

For simulation, assume $P \xrightarrow{\beta} P'$. We need to show that for some Q', s.t. $Q \xrightarrow{\beta} Q'$ and $(P', Q') \in \mathcal{R}$. By Theorem 4.4 (1), we get $[\![P]\!] \xrightarrow{\alpha} [\![P']\!]$ for any $\alpha \in [\![\beta]\!]$. By Theorem 4.4 (2) and using the assumption $\alpha \in [\![\beta]\!]$ as well as the fact $[\![P]\!] \sim [\![Q]\!]$, we derive $[\![Q]\!] \xrightarrow{\alpha} [\![Q']\!]$. From simulation clause and that $[\![P]\!]$ and $[\![Q]\!]$ are congruent follows that $[\![P']\!] \sim [\![Q']\!]$ and hence $(P', Q') \in \mathcal{R}$. Symmetry case follows from the symmetry of $\sim$. Hence $\mathcal{R}$ is an early bisimulation. Since $\mathcal{R}$ is closed under all substitutions by Lemma A.3, it is an early congruence.

We prove direction $\Rightarrow$, assume $P \sim_e^c Q$. We claim the the relation $\{(\mathbf{1}, [\![P]\!], [\![Q]\!]) : P \sim_e^c Q\}$ is a congruence in **PPI**. The static equivalence and extension of arbitrary assertion cases are trivial since there is unit assertion only. Symmetry follows from symmetry of $\sim_e^c$, and simulation follows by Theorem 4.4 and the fact that $\sim_e^c$ is an early congruence.

□

Proof of Theorem 4.7. By structural induction on P. We only consider the case of **case** agent as other cases are trivial.

case case $\varphi_1 : P_1 \;[]\; \dots \;[]\; \varphi_n : P_n$**:**
We get an induction hypothesis for every $i \in \{1..n\}$, IH_i: $P_i \sim \llbracket \overline{P_i} \rrbracket$.
We proceed by induction on n.

base case $n = 0$**:**
$\llbracket \overline{\mathbf{case}} \rrbracket = \llbracket \mathbf{0} \rrbracket = \mathbf{0}$. By reflexivity of $\sim$, $\mathbf{0} \sim \mathbf{0}$.

induction step $n + 1$**:**
The IH for this case is

$$\llbracket \overline{\mathbf{case}\ \varphi_1 : P_1 \;[]\; \dots \;[]\; \varphi_n : P_n} \rrbracket \sim \mathbf{case}\ \varphi_1 : P_1 \;[]\; \dots \;[]\; \varphi_n : P_n = P'$$

We need to show that $Q \sim \llbracket \overline{Q} \rrbracket$ for $Q = \mathbf{case}\ \varphi_1 : P_1 \;[]\; \dots \;[]\; \varphi_n : P_n \;[]\; \varphi_{n+1} : P_{n+1}$.
We compute

$$\begin{array}{rcl}
\llbracket \overline{Q} \rrbracket & = & \llbracket \overline{\varphi_1 : P_1} + \cdots + \overline{\varphi_n : P_n} + \overline{\varphi_{n+1} : P_{n+1}} \rrbracket \\
 & = & \mathbf{case}\ \top : \llbracket \overline{\varphi_1 : P_1} \rrbracket \;[]\; \dots \;[]\; \top : \llbracket \overline{\varphi_n : P_n} \rrbracket \;[]\; \top : \llbracket \overline{\varphi_{n+1} : P_{n+1}} \rrbracket \\
 & \sim & \text{(by Lemma 3.3)} \\
 & & \mathbf{case}\ \top : (\mathbf{case}\ \top : \llbracket \overline{\varphi_1 : P_1} \rrbracket \;[]\; \dots \;[]\; \top : \llbracket \overline{\varphi_n : P_n} \rrbracket) \;[]\; \top : \llbracket \overline{\varphi_{n+1} : P_{n+1}} \rrbracket \\
 & \sim & \text{(by } IH\text{)} \\
 & & \mathbf{case}\ \top : (\mathbf{case}\ \varphi_1 : P_1 \;[]\; \dots \;[]\; \varphi_n : P_n) \;[]\; \top : \llbracket \overline{\varphi_{n+1} : P_{n+1}} \rrbracket \\
 & = & \mathbf{case}\ \top : P' \;[]\; \top : \llbracket \overline{\varphi_{n+1} : P_{n+1}} \rrbracket \\
 & = & Q'
\end{array}$$

We distinguish the cases of φ_{n+1}:

case $\varphi_{n+1} = \top$**:**

$$\begin{array}{rcl}
Q' & = & \mathbf{case}\ \top : P' \;[]\; \top : \llbracket \overline{\top : P_{n+1}} \rrbracket \\
 & = & \mathbf{case}\ \top : P' \;[]\; \top : \llbracket \overline{P_{n+1}} \rrbracket \\
 & \sim & \text{(by } IH_{n+1}\text{)} \\
 & & \mathbf{case}\ \top : P' \;[]\; \top : P_{n+1} \\
 & \sim & \text{(by Lemma 3.3)} \\
 & & \mathbf{case}\ \varphi_1 : P_1 \;[]\; \dots \;[]\; \varphi_n : P_n \;[]\; \top : P_{n+1} = Q
\end{array}$$

We conclude this case.

case $\varphi_{n+1} = x = y$**:**

$$\begin{array}{rcl}
Q' & = & \mathbf{case}\ \top : P' \;[]\; \top : \llbracket \overline{x = y : P_{n+1}} \rrbracket \\
 & = & \mathbf{case}\ \top : P' \;[]\; \top : (\mathbf{case}\ x = y : \llbracket \overline{P_{n+1}} \rrbracket) \\
 & \sim & \text{(by } IH_{n+1}\text{)} \\
 & & \mathbf{case}\ \top : P' \;[]\; \top : (\mathbf{case}\ x = y : P_{n+1}) \\
 & \sim & \text{(by Lemma 3.3)} \\
 & & \mathbf{case}\ \varphi_1 : P_1 \;[]\; \dots \;[]\; \varphi_n : P_n \;[]\; \top : (\mathbf{case}\ x = y : P_{n+1}) \\
 & \sim & \text{(by Lemma 3.3)} \\
 & & \mathbf{case}\ \varphi_1 : P_1 \;[]\; \dots \;[]\; \varphi_n : P_n \;[]\; x = y : P_{n+1} = Q
\end{array}$$

By concluding this case, we conclude the proof. □

Lemma A.5. $\llbracket \cdot \rrbracket$ *is injective, that is, for all* P, Q, *if* $\llbracket P \rrbracket = \llbracket Q \rrbracket$ *then* $P = Q$.

Proof. By induction on P and Q while inspecting all the possible cases. □

Lemma A.6. $\llbracket \cdot \rrbracket$ *is surjective up to* $\sim$, *that is, for every* P *there is a* Q *such that* $\llbracket Q \rrbracket \sim P$.

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', $\llbracket Q' \rrbracket \sim P'$. Let $Q = x(\tilde{y}).Q'$. Then, $\llbracket Q \rrbracket = \llbracket x(\tilde{y}).Q' \rrbracket = \underline{x}(\lambda\widetilde{y})\langle\widetilde{y}\rangle.\llbracket Q' \rrbracket \sim \underline{x}(\lambda\widetilde{y})\langle\widetilde{y}\rangle.P'$. This is what we needed to derive.

case $\overline{x}\langle\widetilde{y}\rangle.P'$**:**
By IH, we have for some Q', $\llbracket Q' \rrbracket \sim P'$. Let $Q = \overline{x}\langle\tilde{y}\rangle.Q'$. Now $\llbracket Q \rrbracket = \overline{x}\langle\widetilde{y}\rangle.\llbracket Q' \rrbracket \sim \overline{x}\langle\tilde{y}\rangle.P'$, which is what we wanted to derive.

case $P \mid P'$**:**
By IH, we have that for some Q', Q'', $\llbracket Q' \rrbracket \sim P$ and $\llbracket Q'' \rrbracket \sim P'$. Then let $Q = Q' | Q''$, thus $\llbracket Q \rrbracket = \llbracket Q' \rrbracket \mid \llbracket Q'' \rrbracket \sim P \mid P'$.

case $(\nu x)P$**:**
By IH, for some Q', $\llbracket Q' \rrbracket \sim P$. Let $Q = \nu x Q'$. Then $\llbracket Q \rrbracket = (\nu x)\llbracket Q' \rrbracket \sim (\nu x)P$.

case $!P$**:**
By IH, for some Q', $\llbracket Q' \rrbracket \sim P$. Let $Q = !Q'$. Then $\llbracket Q \rrbracket = !\llbracket Q' \rrbracket \sim !P$.

case $\llparenthesis \mathbf{1} \rrparenthesis$**:**
Let $Q = \mathbf{0}$. Then $\llbracket Q \rrbracket = \mathbf{0} \sim \llparenthesis \mathbf{1} \rrparenthesis$.

case $\mathbf{case}\ \widetilde{\varphi} : \widetilde{P'}$**:**
For induction hypothesis $\text{IH}_{\mathbf{case}}$, we have for every i there is Q'_i such that $\llbracket Q'_i \rrbracket \sim P'_i$. The proof goes by induction on the length of $\widetilde{\varphi}$.

base case:
Let $Q = \mathbf{0}$, then $\llbracket Q \rrbracket = \mathbf{0} \sim \mathbf{case}$.

induction step:
At this step, we get the following IH

$$\llbracket Q'' \rrbracket \sim \mathbf{case}\ \varphi_1 : P_1 \ [\,]\ \ldots\ [\,]\ \varphi_n : P_n$$

We need to show that there is some $\llbracket Q \rrbracket$ such that

$$\llbracket Q \rrbracket \sim \mathbf{case}\ \varphi_1 : P_1 \ [\,]\ \ldots\ [\,]\ \varphi_n : P_n \ [\,]\ \varphi_{n+1} : P_{n+1}$$

First, we note that $\text{IH}_{\mathbf{case}}$ holds for every i and in particular $i = n + 1$, thus we get $\llbracket Q'_{n+1} \rrbracket \sim P_{n+1}$. 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

$$\begin{array}{rcl}
[\![Q]\!] & = & \textbf{case}\ \top : [\![Q'']\!] \mathrel{[]} \top : [\![Q'_{n+1}]\!] \\
& \sim & \textbf{case}\ \top : (\textbf{case}\ \varphi_1 : P_1 \mathrel{[]} \dots \mathrel{[]} \varphi_n : P_n) \\
& & [] \ \top : [\![Q'_{n+1}]\!] \\
& \sim & \textbf{case}\ \top : (\textbf{case}\ \varphi_1 : P_1 \mathrel{[]} \dots \mathrel{[]} \varphi_n : P_n) \\
& & [] \ \top : P_{n+1} \\
& \sim & \text{(by Lemma 3.3)} \\
& & \textbf{case}\ \varphi_1 : P_1 \mathrel{[]} \dots \mathrel{[]} \varphi_n : P_n \\
& & [] \ \top : P_{n+1}
\end{array}$$

This case is concluded.

case $\varphi_{n+1} = x = y$**:**

Let $Q = Q'' + [x = y]Q'_{n+1}$. Then

$$\begin{array}{rcl}
[\![Q]\!] & = & \textbf{case}\ \top : [\![Q'']\!] \mathrel{[]} \top : [\![[x = y]Q'_{n+1}]\!] \\
& \sim & \textbf{case}\ \top : (\textbf{case}\ \varphi_1 : P_1 \mathrel{[]} \dots \mathrel{[]} \varphi_n : P_n) \\
& & [] \ \top : (\textbf{case}\ x = y : [\![Q'_{n+1}]\!]) \\
& \sim & \textbf{case}\ \top : (\textbf{case}\ \varphi_1 : P_1 \mathrel{[]} \dots \mathrel{[]} \varphi_n : P_n) \\
& & [] \ \top : (\textbf{case}\ x = y : P_{n+1}) \\
& \sim & \text{(by Lemma 3.3)} \\
& & \textbf{case}\ \varphi_1 : P_1 \mathrel{[]} \dots \mathrel{[]} \varphi_n : P_n \\
& & [] \ \top : (\textbf{case}\ x = y : P_{n+1}) \\
& \sim & \text{(by permuting and applying Lemma 3.3)} \\
& & \textbf{case}\ \varphi_1 : P_1 \mathrel{[]} \dots \mathrel{[]} \varphi_n : P_n \mathrel{[]} x = y : P_{n+1}
\end{array}$$

This is the last part we needed to check, we conclude the proof. □

Theorem A.7. $[\![\cdot]\!]$ *is an isomorphism up to* $\sim$.

Proof. Directly follows from Lemma A.5 and Lemma A.6. □

A.2. **Polyadic Synchronisation Pi-Calculus.** We follow the exposition of Polyadic Synchronisation Pi-Calculus, ${}^e\pi$, of Carbone and Maffeis [CM03].

We give an explicit definition of encoding function defined in Example 4.4.

Definition A.8 (Polyadic synchronisation pi-calculus to **PSPi**)**.**
Agents:

$$\begin{array}{rcl}
[\![\widetilde{x}(y).P]\!] & = & \underline{\langle\widetilde{x}\rangle}(\lambda y)y.[\![P]\!] \\
[\![\widetilde{x}\langle y\rangle.P]\!] & = & \overline{\langle\widetilde{x}\rangle}\ y.[\![P]\!] \\
[\![P \mid Q]\!] & = & [\![P]\!] \mid [\![Q]\!] \\
[\![(\nu x)P]\!] & = & (\nu x)[\![P]\!] \\
[\![!P]\!] & = & ![\![P]\!] \\
[\![0]\!] & = & 0 \\
[\![\Sigma_i \alpha_i.P_i]\!] & = & \textbf{case}\ \top_i : [\![\alpha_i.P_i]\!]
\end{array}$$

Actions:

$$\begin{array}{rcl}
\llbracket \tilde{x}\langle \nu c\rangle \rrbracket & = & \overline{\langle \tilde{x}\rangle}\,(\nu c)\,c \\
\llbracket \tilde{x}\langle c\rangle \rrbracket & = & \overline{\langle \tilde{x}\rangle}\,c \\
\llbracket \tau \rrbracket & = & \tau \\
\llbracket \tilde{x}(y) \rrbracket & = & \text{undefined}
\end{array}$$

Because in [CM03] Carbone and Maffeis defines late style laballed semantics for ${}^{e}\pi$ the input action has no translation.

Definition A.9 (**PSPi** to Polyadic synchronisation pi-calculus)**.**

$$\begin{array}{rcl}
\overline{(\!|\mathbf{1}|\!)} & = & \mathbf{0} \\
\overline{\mathbf{0}} & = & \mathbf{0} \\
\overline{!P} & = & !\overline{P} \\
\overline{(\nu x)P} & = & (\nu x)\overline{P} \\
\overline{P \mid Q} & = & \overline{P} \mid \overline{Q} \\
\overline{\langle \widetilde{a}\rangle y.P} & = & \overline{a}\langle y\rangle.\overline{P} \\
\overline{\underline{\widetilde{x}}(\lambda y)y.P} & = & \overline{x}(y).\overline{P} \\
\overline{\tau.P} & = & \tau.\overline{P} \\
\overline{\mathbf{case}\ \top : \alpha_i.P_i} & = & \Sigma_i\overline{\alpha_i.P_i}
\end{array}$$

Lemma A.10. *If* $P \equiv Q$ *then* $\llbracket P \rrbracket \sim \llbracket Q \rrbracket$

Proof. The relation $\mathcal{R} = \{(P, Q) : \llbracket P \rrbracket \sim \llbracket Q \rrbracket\}$ 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 4.16.

(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

$$\begin{array}{rcl}
\llbracket \Sigma_i\tilde{x}_i(y_i).P_i \rrbracket & = & \mathbf{case}\ \top : \underline{\langle \tilde{x}\rangle}(\lambda y_1)y_1.\llbracket P_1 \rrbracket\ [] \\
 & & \cdots\ []\ \top : \underline{\langle \tilde{x}\rangle}(\lambda y_i)y_i.\llbracket P_i \rrbracket
\end{array}$$

Since $\textsc{match}(z, \langle y_i\rangle, y_i) = \{z\}$, we can use the Case and In rules to derive the transition

$$\mathbf{case}\ \top : \underline{\langle \tilde{x}_1\rangle}(\lambda y_1)y_1.\llbracket P_1 \rrbracket\ []\ \cdots\ []\ \top : \underline{\langle \tilde{x}_i\rangle}(\lambda y_i)y_i.\llbracket P_i \rrbracket \xrightarrow{\langle \tilde{x}\rangle\ z} \llbracket P_i \rrbracket[y_i := z]$$

Finally, we have $P'' = \llbracket P_i \rrbracket[y_i := z]$ and use reflexivity of $\sim$.

Bang:

Here $P \mid !P \xrightarrow{\tilde{x}(y)} P'$ and by induction, $\llbracket P \rrbracket \mid !\llbracket P \rrbracket \xrightarrow{\langle \tilde{x}\rangle\ z} P''$ with $P'' \sim \llbracket P' \rrbracket[y := z]$. By rule Rep, we also have that $!\llbracket P \rrbracket \xrightarrow{\langle \tilde{x}\rangle\ z} P''$.

Par:

Here $P \xrightarrow{\tilde{x}(y)} P'$, $y \# Q$ and by induction, $\llbracket P \rrbracket \xrightarrow{\langle \tilde{x}\rangle\ z} P''$ with $P'' \sim \llbracket P' \rrbracket[y := z]$. Using the Par rule we derive $\llbracket P \rrbracket \mid \llbracket Q \rrbracket \xrightarrow{\langle \tilde{x}\rangle\ z} P' \mid \llbracket Q \rrbracket$. Since $\sim$ is closed under $\mid$, $P'' \mid \llbracket Q \rrbracket \sim \llbracket P' \rrbracket[y := z] \mid \llbracket Q \rrbracket$. Finally, since $y \# Q$, $\llbracket P' \rrbracket[y := z] \mid \llbracket Q \rrbracket = \llbracket P' \mid Q \rrbracket[y := z]$.

Struct:
Here $P \equiv Q$, $Q \xrightarrow{\tilde{x}(y)} Q'$ and $Q' \equiv P'$. By induction we obtain Q'' such that $\llbracket Q \rrbracket \xrightarrow{\langle\tilde{x}\rangle\ z} Q''$ where $Q'' \sim \llbracket Q' \rrbracket[y := z]$. By Lemma A.10, $\llbracket P \rrbracket \sim \llbracket Q \rrbracket$ and $\llbracket Q' \rrbracket \sim \llbracket P' \rrbracket$, and by definition of $\sim$, $\llbracket Q' \rrbracket[y := z] \sim \llbracket P' \rrbracket[y := z]$. Since $\llbracket P \rrbracket \sim \llbracket Q \rrbracket$ and $\llbracket Q \rrbracket \xrightarrow{\langle\tilde{x}\rangle\ z} Q''$, there exists P'' such that $\llbracket P \rrbracket \xrightarrow{\langle\tilde{x}\rangle\ z} P''$ and $Q'' \sim P''$. By transitivity of $\sim$, $P'' \sim \llbracket P' \rrbracket[y := z]$.

Res:
Here $P \xrightarrow{\tilde{x}(y)} P'$, $a \neq y$, $a \neq z$ $a\#\tilde{x}$, and by induction, $\llbracket P \rrbracket \xrightarrow{\langle\tilde{x}\rangle\ z} P''$ with $P'' \sim \llbracket P' \rrbracket[y := z]$. This gives us sufficient freshness conditions to derive $(\nu a)\llbracket P \rrbracket \xrightarrow{\langle\tilde{x}\rangle\ z} (\nu a)P''$. Since $\sim$ is closed under restriction, $(\nu a)P'' \sim (\nu a)(\llbracket P' \rrbracket[y := z])$. Finally, a is sufficiently fresh to so that $(\nu a)(\llbracket P' \rrbracket[y := z]) = ((\nu a)\llbracket P' \rrbracket)[y := z]$

(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 $P \xrightarrow{\bar{\tilde{x}}\langle y\rangle} P'$ and $Q \xrightarrow{\tilde{x}(z)} Q'$. By induction, $\llbracket P \rrbracket \xrightarrow{\overline{\langle\tilde{x}\rangle}\ y} P''$ where $P'' \sim \llbracket P' \rrbracket$ and by the previous clause of this theorem, $\llbracket Q \rrbracket \xrightarrow{\langle\tilde{x}\rangle\ y} Q''$ such that $\llbracket Q' \rrbracket[z := y] \sim Q''$. The Com rule lets us derive the transition

$$\llbracket P \rrbracket \mid \llbracket Q \rrbracket \xrightarrow{\tau} P'' \mid Q''$$

To complete the induction case, we note that $(\nu y)(P'' \mid Q'') \sim \llbracket (\nu y)(P' \mid Q'\{y/z\}) \rrbracket$

Close:
Here $P \xrightarrow{\bar{\tilde{x}}\langle \nu y\rangle} P'$ and $Q \xrightarrow{\tilde{x}(y)} Q'$. We assume $y\#Q$; if not, y can be α-converted so that this holds. By induction, $\llbracket P \rrbracket \xrightarrow{\overline{\langle\tilde{x}\rangle}\ (\nu y)\ y} P''$ where $P'' \sim \llbracket P' \rrbracket$ and by the previous clause of this theorem, $\llbracket Q \rrbracket \xrightarrow{\langle\tilde{x}\rangle\ y} Q''$ such that $\llbracket Q' \rrbracket[y := y] = \llbracket Q' \rrbracket \sim Q''$. The Com rule lets us derive the transition

$$\llbracket P \rrbracket \mid \llbracket Q \rrbracket \xrightarrow{\tau} (\nu y)(P'' \mid Q'')$$

To complete the induction case, we note that $(\nu y)(P'' \mid Q'') \sim \llbracket (\nu y)(P' \mid Q') \rrbracket$

Open:
Here $P \xrightarrow{\bar{\tilde{x}}\langle y\rangle} P'$ with $y \neq x$, and by induction, $\llbracket P \rrbracket \xrightarrow{\overline{\langle\tilde{x}\rangle}\ y} P''$ where $P'' \sim \llbracket P' \rrbracket$. By Open, we derive $(\nu y)\llbracket P \rrbracket \xrightarrow{\overline{\langle\tilde{x}\rangle}\ (\nu y)\ y} P''$.

(3) By induction on the derivation of P”, avoiding y.

Par:
Here $\llbracket P \rrbracket \xrightarrow{\underline{x}\ \langle\tilde{z}\rangle} P''$, $y\#P, Q$, and by induction $P \xrightarrow{\tilde{x}(y)} P'$ where $\llbracket P'\{z/y\} \rrbracket = P''$. By Par using $y\#Q$, we derive $P \mid Q \xrightarrow{\tilde{x}(y)} P' \mid Q$. Finally, we note that since $y\#Q$, $\llbracket (P' \mid Q)\{z/y\} \rrbracket = P'' \mid \llbracket Q \rrbracket$.

Case:
Here $P_C \xrightarrow{\underline{\tilde{x}}\ z} P''$, where $P_C = \textbf{case}\ \widetilde{\varphi} : \widetilde{Q}$ is in the range of $\llbracket\cdot\rrbracket$ - hence P_C must be the encoding of some prefix-guarded sum, ie $P_C = \llbracket \Sigma_i \alpha_i.P_i \rrbracket = \textbf{case}\ \top : \llbracket \alpha_1 \rrbracket.\llbracket P_1 \rrbracket \;[]\; \ldots \;[]\; \top : \llbracket \alpha_i \rrbracket.\llbracket P_i \rrbracket$. By transition inversion we can deduce

that for some j, $\alpha_j = \tilde{x}(y)$ and $\llbracket P_j \rrbracket[y := z] = P''$. By the Prefix rule, $\Sigma_i \alpha_i.P_i \xrightarrow{\tilde{x}(y)} P_j$.

Out:
A special case of Case.

Rep:
Here $\llbracket P \rrbracket \mid !\llbracket P \rrbracket \xrightarrow{x\ \langle\tilde{z}\rangle} P''$ and by induction $P \mid !P \xrightarrow{\tilde{x}(y)} P'$ where $\llbracket P'\{z/y\} \rrbracket = P''$. By Bang we derive $!P \xrightarrow{\tilde{x}(y)} P'$.

Scope:
Here $\llbracket P \rrbracket \xrightarrow{x\ \langle\tilde{z}\rangle} P''$, $y\#P, Q$, $a\#\tilde{x}, y, z$ and by induction $P \xrightarrow{\tilde{x}(y)} P'$ where $\llbracket P'\{z/y\} \rrbracket = P''$. Since $a\#\tilde{x}, y, z$, the Res rule admits the derivation $(\nu a)P \xrightarrow{\tilde{x}(y)} (\nu a)P'$, and $\llbracket ((\nu a)P')\{z/y\} \rrbracket = (\nu a)P''$

(4) By induction on the derivation of P”. The cases not shown are similar to the previous clause of this theorem.

Com:
Here $\llbracket P \rrbracket \xrightarrow{\overline{\langle\tilde{x}\rangle}\ (\nu\tilde{y'})\ y} P''$, $\llbracket Q \rrbracket \xrightarrow{\langle\tilde{x}\rangle\ y} Q''$ 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 $\llbracket P' \rrbracket = P''$ by induction and, by the previous clause of this theorem, $Q \xrightarrow{\tilde{x}(z)} Q'$ where $\llbracket Q'\{y/z\} \rrbracket = Q''$. 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 $\llbracket P' \rrbracket = P''$ by induction and, by the previous clause of this theorem, $Q \xrightarrow{\tilde{x}(y)} Q'$ where $\llbracket Q'\{y/y\} \rrbracket = \llbracket Q' \rrbracket = Q''$. The Close rule then lets us derive $P \mid Q \xrightarrow{\tau} (\nu y)(P' \mid Q')$.

Open:
Here $\llbracket P \rrbracket \xrightarrow{\overline{\langle\tilde{x}\rangle}\ y} P''$ with $y \neq x$. By induction, $P \xrightarrow{\overline{\tilde{x}}\langle y\rangle} P'$ where $\llbracket P' \rrbracket = P''$. By rule Open, $(\nu y)P \xrightarrow{\overline{\tilde{x}}\langle\nu y\rangle} P'$. □

We give the full abstraction result for this calculus. The definition of congruence for polyadic synchronisation pi-calculus can be found in [CM03] on page 6.

Theorem A.11. *For all ${}^e\pi$ processes P and Q, $P \sim Q$ iff $\llbracket P \rrbracket \sim \llbracket Q \rrbracket$*

Proof. $\mathcal{R} = \{(P, Q) : \llbracket P \rrbracket \sim \llbracket Q \rrbracket\}$ is an early congruence in the polyadic synchronisation pi-calculus; if $P\ \mathcal{R}\ Q$ then

(1) If $P \xrightarrow{\tilde{x}(y)} P'$ and $\llbracket P \rrbracket \sim \llbracket Q \rrbracket$, since $\mathcal{R}$ is equivariant, we can assume that $y\#P, Q$ without loss of generality. Fix z. By Theorem 4.16 (1), $\llbracket P \rrbracket \xrightarrow{\langle\tilde{x}\rangle\ z} P''$ where $P'' \sim \llbracket P' \rrbracket[y := z] = \llbracket P'\{z/y\} \rrbracket$. Hence, since $\llbracket P \rrbracket \sim \llbracket Q \rrbracket$, $\llbracket Q \rrbracket \xrightarrow{\langle\tilde{x}\rangle\ z} Q''$ where $P'' \sim Q''$. Hence, by Theorem 4.16.3 using $y\#Q$, $Q \xrightarrow{\tilde{x}(y)} Q'$ where $\llbracket Q'\{z/y\} \rrbracket = Q''$. By transitivity, $\llbracket P'\{z/y\} \rrbracket \sim \llbracket Q'\{z/y\} \rrbracket$.

(2) If $P \xrightarrow{\alpha} P'$ and $\llbracket P \rrbracket \sim \llbracket Q \rrbracket$, since $\mathcal{R}$ is equivariant, we can assume that $\mathrm{bn}(\alpha)\#P, Q$ without loss of generality. By Theorem 4.16.2, we have that $\llbracket P \rrbracket \xrightarrow{\llbracket\alpha\rrbracket} P''$ with $P'' \sim \llbracket P' \rrbracket$. Hence, since $\llbracket P \rrbracket \sim \llbracket Q \rrbracket$ and $\mathrm{bn}(\alpha)\#Q$, there is a Q'' such that $\llbracket Q \rrbracket \xrightarrow{\llbracket\alpha\rrbracket} Q''$

and $Q'' \sim P''$. By Theorem 4.16.4, there is Q' such that $Q \xrightarrow{\alpha} Q'$ and $[\![Q']\!] = Q''$. By transitivity, $[\![P']\!] \sim [\![Q']\!]$.

Symmetrically, we show that $\mathcal{R} = \{(\mathbf{1}, [\![P]\!], [\![Q]\!]) : P \sim Q\}$ is a congruence in **PSPI**:

Static equivalence:
Trivial since there is only a unit assertion.

Symmetry:
By symmetry of $\sim$

Simulation:
Here $[\![P]\!] \xrightarrow{\alpha'} P''$ and $P \sim Q$. We proceed by case analysis on α':

(1) If $\alpha' = \underline{\langle\tilde{x}\rangle}\ z$, then by Theorem 4.16 (3) and a sufficiently fresh y, $P \xrightarrow{\tilde{x}(y)} P'$ where $[\![P'\{z/y\}]\!] = P''$. Since $P \sim Q$, there exists Q' such that $Q \xrightarrow{\tilde{x}(y)} Q'$ and $P'\{z/y\} \sim Q'\{z/y\}$. Hence, by Theorem 4.16 (1), $[\![Q]\!] \xrightarrow{\underline{\langle\tilde{x}\rangle}\ z} Q''$ where $Q'' \sim [\![Q']\!][y := z] = [\![Q'\{z/y\}]\!]$. We have that $P'' = [\![P'\{z/y\}]\!]\ \mathcal{R}\ [\![Q'\{z/y\}]\!] \sim Q''$, 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 4.16 (4) we have that $P \xrightarrow{\alpha} P'$ where $[\![\alpha]\!] = \alpha'$ and $[\![P']\!] = P''$. Since $P \sim Q$, there is Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \sim Q'$. By Theorem 4.16 (2), $[\![Q]\!] \xrightarrow{[\![\alpha]\!]} Q''$, where $Q'' \sim [\![Q']\!]$. Hence $P'' = [\![P']\!]\ \mathcal{R}\ [\![Q']\!] \sim Q''$, which suffices.

Extension of arbitrary assertion:
Trivial since there is only a unit assertion. □

Lemma A.12. $[\![\cdot]\!]$ *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 $\mathbf{0}$**:**
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 $\mathrm{IH}_{\mathbf{case}}$ 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 $\varphi_{n+1} = \bot$**:**
Let $Q = Q''$. Then

$$\begin{array}{rcl} [\![Q]\!] & = & [\![Q'']\!] \\ & \sim & \mathbf{case}\,\varphi_1 : P_1 \;[\!]\; \ldots \;[\!]\; \varphi_n : P_n \\ & \sim & \mathbf{case}\,\varphi_1 : P_1 \;[\!]\; \ldots \;[\!]\; \varphi_n : P_n \;[\!]\; \bot : P_{n+1} \end{array}$$

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

$$\begin{array}{rcl} [\![Q]\!] & = & \mathbf{case}\,\top : [\![Q'']\!] \;[\!]\; \top : [\![Q'_{n+1}]\!] \\ & \sim & \mathbf{case}\,\top : (\mathbf{case}\,\varphi_1 : P_1 \;[\!]\; \ldots \;[\!]\; \varphi_n : P_n) \\ & & [\!]\; \top : [\![Q'_{n+1}]\!] \\ & \sim & \text{(by Lemma 3.3)} \\ & & \mathbf{case}\,\varphi_1 : P_1 \;[\!]\; \ldots \;[\!]\; \varphi_n : P_n \\ & & [\!]\; \top : [\![Q'_{n+1}]\!] \\ & \sim & \mathbf{case}\,\varphi_1 : P_1 \;[\!]\; \ldots \;[\!]\; \varphi_n : P_n \\ & & [\!]\; \top : P'_{n+1} \end{array}$$

This concludes the proof. □

Lemma A.13. *$[\![\cdot]\!]$ 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 A.14. *$[\![\cdot]\!]$ is an isomorphism up to $\sim$ between ${}^{e}\pi$ and the case-guarded processes in* **PSPI**.

Proof. Directly follows from Lemma A.13 and Lemma A.12. □

A.3. **Value-passing CCS.**

Lemma A.15. *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. □

We assume a reverse translation $\widehat{\cdot}$ from **VPCCS** to value-passing CCS. We prove strong operational correspondence.

Proof of Theorem 4.23.

(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 $\underline{x}(\lambda y)y.[\![P]\!] \xrightarrow{x\ v} [\![P\{v/y]\!]$ 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

$$\textbf{case}\ \top : [\![P_1]\!]\ [\!]\ \cdots\ [\!]\ \top : P_i \xrightarrow{[\![\alpha]\!]} [\![P']\!]$$

This suffices since $[\![\Sigma_i P_i]\!] = \textbf{case}\ \top : [\![P_1]\!]\ [\!]\ \cdots\ [\!]\ \top : P_i$.

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 A.15 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 A.15 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 A.15, $\tilde{y} = \epsilon$, 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, **if** $\widehat{b\ \textbf{then}}\ P = \widehat{P}$ - hence **if** b **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. □

Recent licentiate theses from the Department of Information Technology

2014-006 Per Mattsson: *Pulse-modulated Feedback in Mathematical Modeling and Estimation of Endocrine Systems*

2014-005 Thomas Lind: *Change and Resistance to Change in Health Care: Inertia in Sociotechnical Systems*

2014-004 Anne-Kathrin Peters: *The Role of Students' Identity Development in Higher Education in Computing*

2014-003 Liang Dai: *On Several Sparsity Related Problems and the Randomized Kaczmarz Algorithm*

2014-002 Johannes Nygren: *Output Feedback Control - Some Methods and Applications*

2014-001 Daniel Jansson: *Mathematical Modeling of the Human Smooth Pursuit System*

2013-007 Hjalmar Wennerström: *Meteorological Impact and Transmission Errors in Outdoor Wireless Sensor Networks*

2013-006 Kristoffer Virta: *Difference Methods with Boundary and Interface Treatment for Wave Equations*

2013-005 Emil Kieri: *Numerical Quantum Dynamics*

2013-004 Johannes Åman Pohjola: *Bells and Whistles: Advanced Language Features in Psi-Calculi*

2013-003 Daniel Elfverson: *On Discontinuous Galerkin Multiscale Methods*

2013-002 Marcus Holm: *Scientific Computing on Hybrid Architectures*

UPPSALA
UNIVERSITET

Department of Information Technology, Uppsala University, Sweden