UPPSALA
UNIVERSITET

IT Licentiate theses
2012-008

Extending Psi-calculi and their Formal Proofs

PALLE RAABJERG

UPPSALA UNIVERSITY
Department of Information Technology

Extending Psi-calculi and their Formal Proofs

Palle Raabjerg
palle.raabjerg@it.uu.se

November 2012

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

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

Dissertation for the degree of Licentiate of Philosophy in Computer Science

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

Abstract

Psi-calculi is a parametric framework for extensions of the pi-calculus, with arbitrary data structures and logical assertions for facts about data. This thesis presents broadcast psi-calculi and higher-order psi-calculi, two extensions of the psi-calculi framework, allowing respectively one-to-many communications and the use of higher-order process descriptions through conditions in the parameterised logic. Both extensions preserve the purity of the psi-calculi semantics; the standard congruence and structural properties of bisimilarity are proved formally in Isabelle. The work going into the extensions show that depending on the specific extension, working out the formal proofs can be a work-intensive process. We find that some of this work could be automated, and implementing such automation may facilitate the development of future extensions to the psi-calculi framework.

Acknowledgements

I would like to thank my advisor, Joachim Parrow, and my co-advisor, Björn Victor for all their support, help, and advice.

I would like to thank all the co-authors; Johannes Borgström, Shuqin Huang, Magnus Johansson, Joachim Parrow, Johannes Åman Pohjola, and Björn Victor.

I would also like to thank Philochoros for providing me with a social life outside of university.

This work was supported by the Swedish Research Council and carried out within the Linnaeus centre of excellence UPMARC, Uppsala Programming for Multicore Architectures Research Center.

List of Papers

This thesis is based on the following two papers.

I Johannes Borgström, Shuqin Huang, Magnus Johansson, Palle Raabjerg, Björn Victor, Johannes Åman Pohjola, and Joachim Parrow. Broadcast Psi-calculi with an Application to Wireless Protocols. *Proceedings of Software Engineering and Formal Methods*, pages 74-89, 2011.
Contributions: Mechanical proofs of correctness of standard congruence and structural properties of bisimilarity.

II Joachim Parrow, Johannes Borgström, Palle Raabjerg, and Johannes Åman Pohjola. Higher-order psi-calculi. Submitted to Mathematical Structures in Computer Science.
Contributions: Mechanical proofs of correctness of standard congruence and structural properties of bisimilarity.

Contents

Chapter 1

Introduction

The Universe runs in parallel. As hydrogen fuses into helium in the heart of the Sun, the same process happens in about 300 billion other stars in the Milky Way, multiplied by 125 billion galaxies in the universe, each star generating light visible from other star systems. As the Earth hurtles around the Sun it acts in parallel with 7 other planets[1], all affecting each other through the force of gravity. On Earth, weather patterns move about in parallel, bumping into each other, creating new weather patterns, lightning storms, tornadoes, hurricanes and heat waves. On the ground, 7 billion humans walk around, in parallel, communicating and thus affecting each other. And human communication is no longer restricted entirely by geography and location. Conversations move by electrons in copper cables, by light in optical cables, by radio waves through the air. We have even developed computers that can automate and organise much of this communication for us. Weather stations all over the world read weather patterns which are used as variables in predictive simulations performed by parallel computations on distributed super-computers. The results of those simulations are then made available as weather reports to millions of people around the world, through the use of messaging protocols on a massively parallel communication network. And as you read this text, the data of which has likely passed through that network as a bundle of messages more than once, images from your retinae are interpreted by parts of your brain in parallel with sound, smell, touch and taste.

But this is where it stops. Because while your brain collects and interprets that data in parallel, it compiles it into a hierarchy of patterns which at the top becomes a single pattern of the world, and these patterns are then understood in time-dependent sequences [Haw07]. As many neurologists will tell you, the ability to multitask as a human is an illusion at best. The brain is geared for prediction, and it does this by remembering the world through sequences and recalling stored sequences through pattern recognition. So in this way we are pretty good at understanding causality: Events tend to happen as causes of

[1] Pluto was recently demoted, and no longer counts as a planet

other events, and we use this to constantly predict what happens next around us. But because we tend to conflate parallel events into sequences we often derive causality where there is none. If we see two events happen after each other we sometimes assume a causality that does not exist: We perform a dance, and it starts to rain. Suddenly we call that dance a "rain dance" and expect rain to come, or at least be more probable when we perform the dance.

So at a very basic level, our intuitive understanding of the world around us is a heavily optimised sequential approximation of a parallel reality. It assumes some connections where there are none, and misses some connections where they do exist. And while it is an approximation that has served us well for most of human history, the progress of science and technology increasingly presents us with questions and problems that require us to understand much better the intricacies and consequences of parallel interaction.

One of the defining characteristics of humans is that we build and use tools to make up for our shortcomings, both physically and intellectually. We cannot chop down trees with our bare hands, and so we invent the axe. We cannot move very fast, and so we invent the bicycle. We cannot see very well, and so we invent glasses and telescopes.

We cannot intuitively understand the universe, and so we invent mathematics. The distances that the brain really understands can be measured in metres. But the distances we can express and perform calculations with in mathematics are literally boundless. Even if we may never completely grasp just how far "10 billion light-years" is, we can still work with such numbers on paper and in simulations. It seems perhaps trite, but it is the difference between believing in dots of light attached to a firmament, and knowing about other stars existing at literally unfathomable distances.

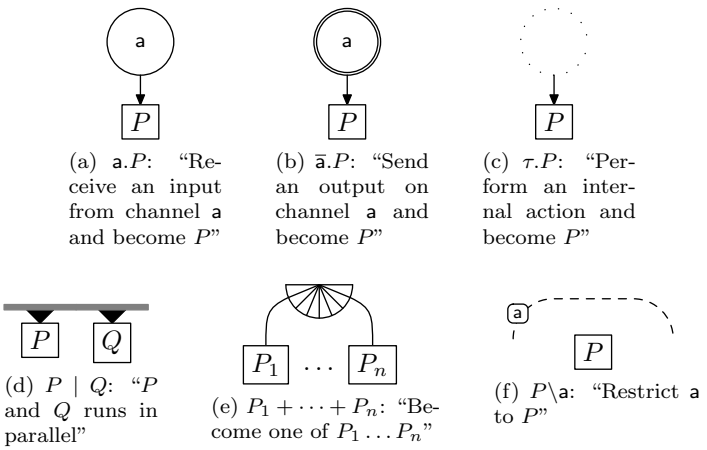
And finally, getting to the point of this introduction: We cannot intuitively understand parallel interactions, and so we invent process calculi.

1.1 Process Calculi

Process calculi were developed in many ways to facilitate the simplest possible mathematical models of parallel interaction. The word "process" bears witness to the fact that these calculi were developed in a computer science environment where the process abstraction is common. But depending on what you wish to model, you could easily replace the word "process" with "human", "cell", "aeroplane", "car" or any other object or entity that could be thought to communicate in parallel with other such entities.

1.1.1 CCS

One of the simplest process calculi is the Calculus of Communicating Systems (CCS), developed by Robin Milner [Mil80]. It is a simple modelling language that allows us to describe and simulate communication in parallel systems.



(a) $\mathtt{a}.P$: "Receive an input from channel a and become P"

(b) $\bar{\mathtt{a}}.P$: "Send an output on channel a and become P"

(c) $\tau.P$: "Perform an internal action and become P"

(d) $P \mid Q$: "P and Q runs in parallel"

(e) $P_1 + \cdots + P_n$: "Become one of $P_1 \ldots P_n$"

(f) $P\backslash\mathtt{a}$: "Restrict a to P"

Figure 1.1: Recursive definition of CCS syntax. A process P or Q can be any of the above. a is some channel name from an infinite collection of names.

In Figure 1.1 we show the CCS syntax together with informal descriptions of what the constructs mean. In fact, we show two versions of the syntax: A textual version and a graphical version. The textual syntax is how Milner originally presented the calculus. The graphical syntax is one developed as an experiment for this thesis.

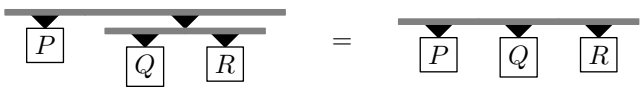
The CCS syntax provides a way of describing a process in an environment where processes run in parallel and can communicate with each other over communication channels. Names in the syntax work as identifiers for those channels. Since the definition is recursive, parallel compositions will allow any number of processes to run in parallel. The structural congruences in Figure 1.2a and 1.2b ensure that we can describe such parallel processes in any order and nesting that we like. To illustrate this, we also introduce a bit of syntactic sugar in Figure 1.2c.

With the syntax comes a system of operational semantics. This semantics describes precisely what a CCS process may do in any give state. The semantics will not be shown here, but to give you some idea of how it works, let us consider a classic example of CCS in Figure 1.3 (in this case borrowed from the book "Reactive Systems" [AILS07]).

Figure 1.3a describes a coffee machine: It inputs a coin, outputs some coffee and starts over. 1.3b is a computer scientist: She outputs a publication, outputs a coin and inputs coffee before starting over with a new publication. Finally, in 1.3c we use parallel composition (Figure 1.1d) to compose the two into the same process in parallel: A small university, `SmUni`. In this process, the names coin and coffee are restricted. This means that they are hidden from

(a) Commutativity of parallel composition: $P \mid Q \equiv Q \mid P$

(b) Associativity of parallel composition: $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$



(c) Syntactic sugar for parallel composition: $P \mid (Q \mid R) = P \mid Q \mid R$

Figure 1.2: Structural congruences and syntactic sugar for parallel composition

any process that might be running outside the `SmUni` process.

A process description can be seen as a state in a labelled transition system (LTS). From that description, a process may then perform actions in accordance with the operational semantics, and thus make transitions to other states. The input, output and tau constructs are commonly known as prefixes, referring to the fact that they are always followed by (prefixed to) a subprocess. Individually, input and output prefixes can perform input and output actions, signifying external communication. An input and an output on the same channel can synchronise and thus communicate with each other internally, performing a tau action. Tau actions can also be performed explicitly by the tau prefix (Figure 1.1c).A prefixed subprocess is referred to as being guarded.

Take the `SmUni` example. If we expand the `CM` and `CS` references, we have the process shown in Figure 1.4a. `CM` and `CS` are running in parallel under the parallel composition at the top. coin and coffee are restricted, meaning only pub can synchronise with outside processes. When executing, the outermost (or unguarded) prefixes indicate the readiness of the process: What it is prepared to do in its current state. In effect, `CM` is ready for an input on coin and `CS` is ready for an output on pub. Since coin is restricted and has no process to synchronise with, the only action the system can do is an output on pub. Thus, the process transitions to 1.4b. The coin channel is restricted, but can now synchronise internally and cause a tau action, becoming 1.4c. Similarly, the coffee channel is restricted, but can now synchronise internally and do another tau action to transition to the original state, 1.4d.

It seems that the `SmUni` process has no choice in how it behaves at any point. At each stage, its behaviour is restricted to a single possible action. This is rarely the case in real specifications. Behavioural choices will occur implicitly whenever there is more than one opportunity for prefixes to synchronise. They can also occur explicitly through use of the nondeterministic choice syntax

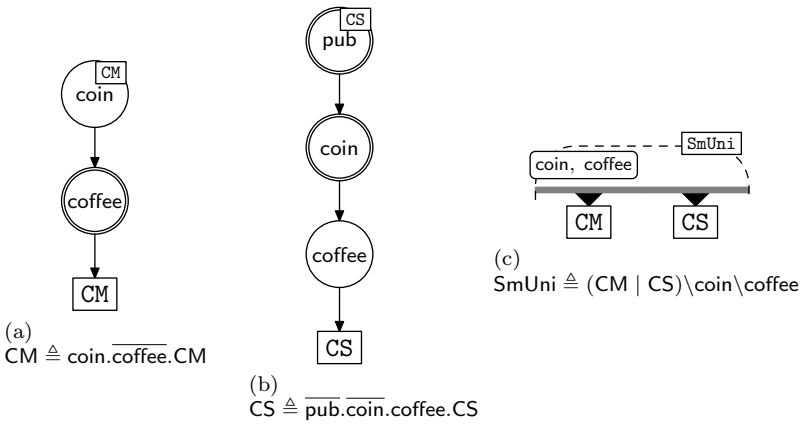


Figure 1.3: Classic CCS university example: (a) represents a coffee machine, (b) a computer scientist and (c) a small university consisting of a coffee machine and a computer scientist running in parallel.

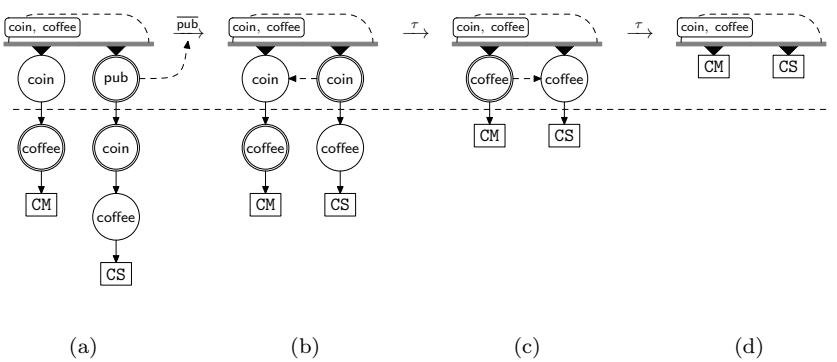


Figure 1.4: Transitions of the `SmUni` process.

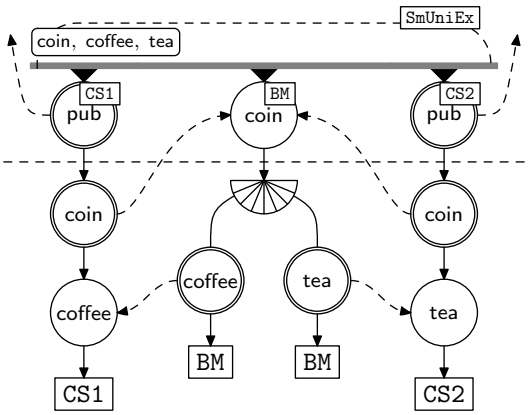


Figure 1.5: `SmUniEx` is the `SmUni` process extended with another computer scientist who drinks tea, and a beverage machine which will produce either coffee or tea.

seen in Figure 1.1e. If we create a simple extension to the `SmUni` process as in Figure 1.5, we can see the effects of nondeterministic choice. The process has been annotated with dashed arrows to show every possible synchronisation in the life of the process. Now it already has a choice in the very beginning: Either scientist could output a publication. If one outputs a publication, that scientist's coin output prefix becomes unguarded and thus an option. But the other scientist might also decide to publish something. If she does, the other coin output prefix becomes unguarded, and it becomes a race for the beverage machine. With two outputs and a single input, only one of the outputs can synchronise with the input. For an alternative chain of events, one scientist could hold off on publishing until the other has input a coin and is waiting for his tea.

Such nondeterminism caused by parallel interaction is one of the primary difficulties encountered today in distributed and multicore programming. Process calculi provide a way of modelling and reasoning about such problems in a formal, theoretical framework.

Strong Bisimilarity

Consider now the processes `PUB` and `PUB2`, shown in Figure 1.6. How does `PUB` differ from `SmUni` and `SmUniEx`? Behaviourally, not much if at all as it turns out. It depends on how exactly we define the equivalence.

The examples of processes depicted here all have a finite number of states they can be in. The transitions we observed in Figure 1.4 enumerate all three possible states the `SmUni` process can traverse. The number of states for `SmUniEx` is significantly larger because of the nondeterministic elements of that process. Each process thus defines a labelled state transition system, the

labels being the actions a process can take to transition to another state. A transition from P to P' using action α is written as $P \xrightarrow{\alpha} P'$.

Definition 1 (Strong bisimulation)**.** *A binary relation $\mathcal{R}$ of processes is a* strong bisimulation *if whenever $(P, Q) \in \mathcal{R}$ and α is $\mathtt{a}$, $\overline{\mathtt{a}}$ or τ:*

- *if $P \xrightarrow{\alpha} P'$ then $Q \xrightarrow{\alpha} Q'$ for some Q' such that $(P', Q') \in \mathcal{R}$, and*
- *if $Q \xrightarrow{\alpha} Q'$ then $P \xrightarrow{\alpha} P'$ for some P' such that $(P', Q') \in \mathcal{R}$.*

Definition 2 (Strong bisimilarity)**.** *Two processes P and Q are* strongly bisimilar *($P \dot{\sim} Q$) if there is a strong bisimulation relation $\mathcal{R}$ such that $(P, Q) \in \mathcal{R}$.*

Informally, this means that in a strong bisimulation, processes can simulate each other's actions, including τ actions. At every point in a strong bisimulation, if one process can perform an action, the other process will be able to imitate it. And if two processes are contained in such a relation, they are *strongly bisimilar*.

This means that `SmUni` (1.3) and `PUB2` (1.6b) are strongly bisimilar. They are both constrained to the exact same sequence of actions: $\xrightarrow{\overline{\mathsf{pub}}} \xrightarrow{\tau} \xrightarrow{\tau} \xrightarrow{\overline{\mathsf{pub}}} \cdots$. For `SmUniEx` (1.5) however, it becomes far more hairy. While everything but the `pub` channel is restricted, the nondeterminism allows many different interleavings of $\overline{\mathsf{pub}}$ actions and τ actions.

Weak bisimilarity

Since τ actions represent internal actions, they should have no bearing on observable behaviour, and so it is often useful to be able to ignore them in behavioural equivalences. This idea gives rise to the notion of weak bisimilarity.

Definition 3 ($\Longrightarrow$ transitions)**.** *We write $P \stackrel{\epsilon}{\Longrightarrow} Q$ iff $P \xrightarrow{\tau} \ldots \xrightarrow{\tau} Q$. We write $P \stackrel{\alpha}{\Longrightarrow} Q$ iff there are processes P', Q' such that $P \stackrel{\epsilon}{\Longrightarrow} P' \xrightarrow{\alpha} Q' \stackrel{\epsilon}{\Longrightarrow} Q$.*

Definition 4 ($\hat{\alpha}$)**.** *If $\alpha = \tau$, then $\hat{\alpha} = \epsilon$. Otherwise, $\hat{\alpha} = \alpha$.*

Informally then, $\Longrightarrow$ and $\hat{\alpha}$ allow us to ignore τ transitions. And so we can succinctly define weak bisimulation.

Definition 5 (Weak bisimulation)**.** *A binary relation $\mathcal{R}$ of processes is a* weak bisimulation *if whenever $(P, Q) \in \mathcal{R}$:*

- *if $P \xrightarrow{\alpha} P'$ then $Q \stackrel{\hat{\alpha}}{\Longrightarrow} Q'$ for some Q' such that $(P', Q') \in \mathcal{R}$, and*
- *if $Q \xrightarrow{\alpha} Q'$ then $P \stackrel{\hat{\alpha}}{\Longrightarrow} P'$ for some P' such that $(P', Q') \in \mathcal{R}$.*

Definition 6 (Weak bisimilarity)**.** *Two processes P and Q are* weakly bisimilar *($P \dot{\approx} Q$) if there is a weak bisimulation relation $\mathcal{R}$ such that $(P, Q) \in \mathcal{R}$.*

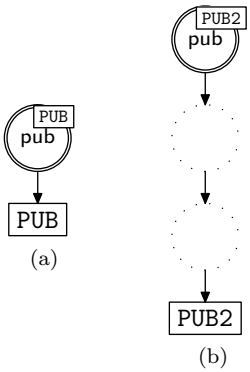


Figure 1.6: Simple processes that do nothing but output on pub.

SmUni, SmUniEx, PUB and PUB2 are all weakly bisimilar. These particular processes will never deadlock, and they will all endlessly produce $\overline{\mathsf{pub}}$ transitions interspersed with τ transitions. Ignoring the τ transitions, they all exhibit the same behaviour.

Bisimilarity can be useful as an abstraction when working with large, complicated processes. Being able to replace parts of a process with simpler, bisimilar counterparts can be useful when performing analyses.

1.1.2 pi

The pi-calculus is a successor to CCS, and adds message passing of channel names and scope extension to the basic calculus. Thus, processes in the pi-calculus can exchange channel names and extend the scopes of those names whenever necessary. This allows pi-calculus processes to model for example the exchange of contact information, like IP addresses or process IDs.

Message Passing

In the pi-calculus, processes may send and receive messages in the form of names. Additionally, those names identify channels. So when a process receives a name through communication, it can use the name for further communication. To accommodate message passing, the input and output syntax must be changed slightly.

The output syntax in Figure 1.7b now contains two names. The name of the channel for the output (m), and the name we wish to output (n). Similarly in Figure 1.7a, the input syntax now contains two names, the name of the channel for the input (m) and the name to substitute for the incoming name (n). For the input, n binds into P, so when an input and an output synchronises in pi, every instance of n is substituted with the incoming message in the continuation of

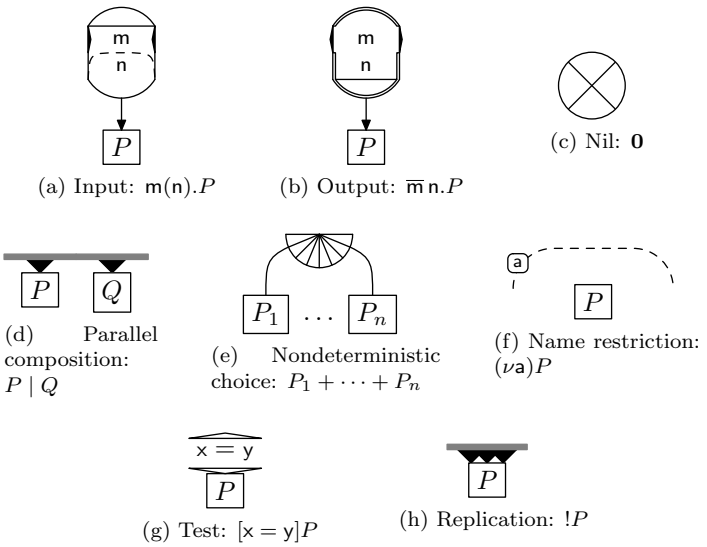


Figure 1.7: Pi syntax. A process P or Q can be any of the above.

the input.

In Figure 1.8 we see how a is sent on b and replaces x. It is then used to send back hello as a reply. Q becomes $Q[\mathsf{x} := \mathsf{a}]$ (x substituted for a) in the first transition, and P becomes $P[\mathsf{x} := \mathsf{hello}]$ (x substituted for hello) in the second transition.

Scope Extension

One of the defining properties of the pi-calculus is scope extension. When a process attempts to send a channel name outside its scope, that scope will expand to include the receiving process. Figure 1.9 shows what happens in

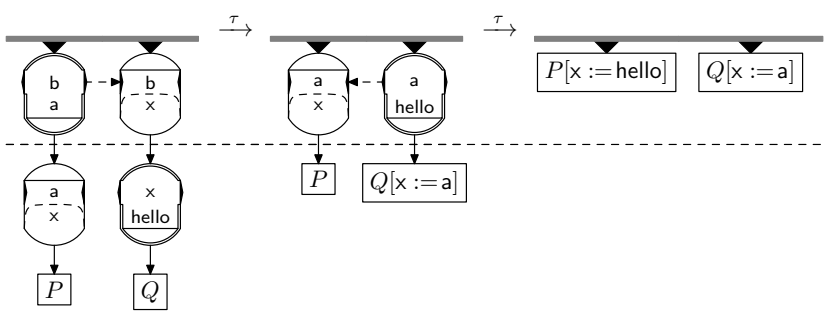


Figure 1.8: Names are channels and can be sent through other channels

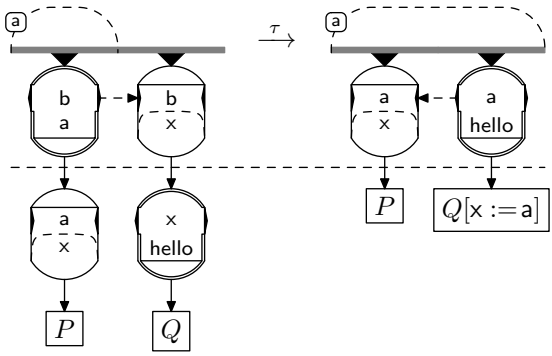


Figure 1.9: Scopes will extend when names try to escape

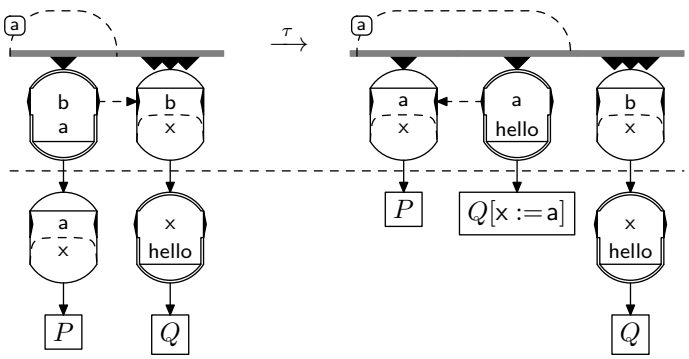


Figure 1.10: Replicated processes spawn a new copy every time it interacts with something

the first transition of the process from 1.8 if the scope of a is restricted to the left-hand process.

In the pi-calculus, scopes restrict names that move around in a system of parallel processes. So when writing pi-calculus processes, it is sometimes better to think of the scopes as representing knowledge of information. In cryptographic versions of the calculus for example, scopes are often used to reason about which processes know which keys and which plaintext messages. In that context, proofs can be worked out to show that information will never be known outside a certain process, or never be known by some specific process.

Tests, Nil and Replication

Apart from message passing, you will note a few other additions in the pi syntax in Figure 1.7. Nil (Figure 1.7c) is simply the process that does nothing. Nil is not necessarily specific to the pi-calculus and could be introduced as an addition to basic CCS. Replication (Figure 1.7g) is also not specific to pi and

could be used with basic CCS. Interaction with a process under replication will simply cause it to spawn a new copy. Continuing with the handshake sample, adding the replication operator as in Figure 1.10 will let the process perform an arbitrary number of handshakes.

Tests (Figure 1.7g) is a construction made useful by the introduction of message passing. They can be used to check the contents of a message. A test that does not evaluate to true will simply become inert, like the nil process. Refer to Figure 1.11 to see how the addition of tests to replication will allow any number of processes to try a handshake, but only those that send the name hey will receive a reply.

Calculus Extensions

The pi-calculus has become quite popular, to the extent that the original papers have accumulated thousands of citations. In many cases, the calculus is used with specific purposes in mind that the basic calculus does not quite accommodate. And so, much of this attention comes in the form of a great number of extensions to the calculus: Applied pi [AF01], spi-calculus [AG97], stochastic pi [Pri95], polyadic pi [CM03], etc.

Some extensions expand the notion of a message to include for example pairs or cryptographic primitives (polyadic pi, spi calculus). Some add new process syntax for certain functionalities. Others again try to change the effects of communication (fusion calculus [PV98]).

The pi-calculus comes with a number of useful results regarding its properties and its notion of bisimilarity. So every time someone changes or extends the basic syntax and semantics, those proofs should be checked. This is a process that tends to be error-prone.

A good example of this is the applied pi calculus [AF01]. Applied pi introduces the concept of active substitutions and encrypted messages. An active substitution $\{^{M}/_{\mathsf{x}}\}$ has an effect on parallel processes defined by the structural congruence in Figure 1.12. A distinction made in this version of applied pi is that of names and variables: Names $(\mathsf{a}, \mathsf{b}, \mathsf{c})$ can be sent on channels, but not substituted by an active substitution. Variables $(\mathsf{x}, \mathsf{y}, \mathsf{z})$ can be substituted by an active substitution, but not sent on channels themselves.

A useful property to have is compositionality of bisimulation, namely that if $P \mathrel{\dot{\sim}} Q$, then $P \mid R \mathrel{\dot{\sim}} Q \mid R$. But as was shown in [BJPV09], that property fails in the applied pi calculus. Consider two processes (where we have omitted the objects for simplicity) $A \triangleq (\nu \mathsf{a})(\{^{\mathsf{a}}/_{\mathsf{x}}\} \mid \mathsf{x}.\mathsf{b}.\mathbf{0})$ and $B \triangleq (\nu \mathsf{a})(\{^{\mathsf{a}}/_{\mathsf{x}}\} \mid \mathbf{0})$. Neither A nor B can take any action. A has no transitions because x is a variable and the substitution will replace it with a, which is a restricted name. Thus $A \mathrel{\dot{\sim}} B$. Consider then $R \triangleq \bar{\mathsf{x}}.\mathbf{0}$, and from R the processes $A \mid R$ (Figure 1.14) and $B \mid R$ (Figure 1.13). In $B \mid R$ the scope of a could legally extend itself to R, letting the substitution replace x with a. $B \mid R$ would still be an inert process because of the restriction of a.

Not so for $A \mid R$. The equivalence caused by the active substitution means

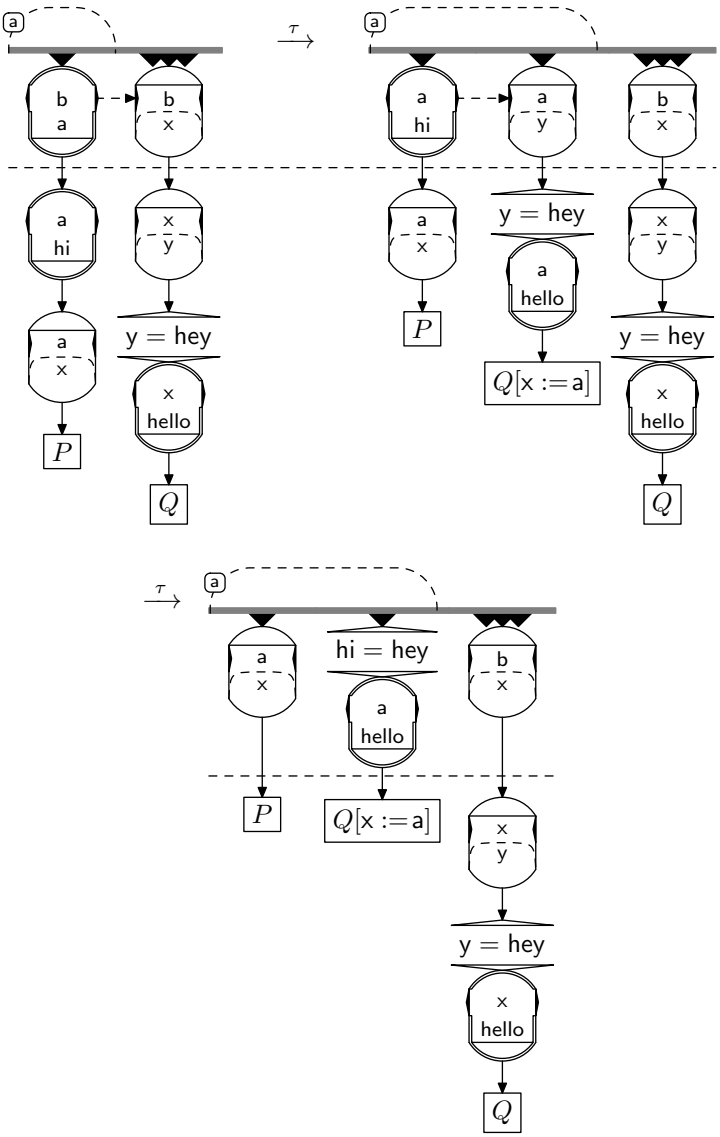


Figure 1.11: A test halts a process if the test does not evaluate to true

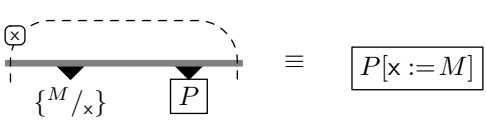


Figure 1.12: Active substitutions in applied pi

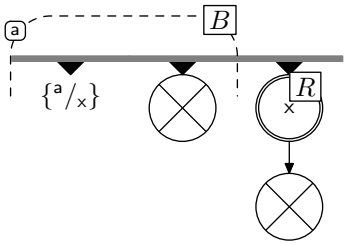


Figure 1.13: $B \mid R$

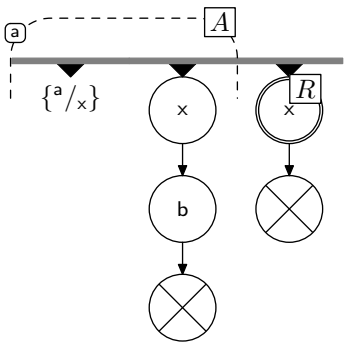


Figure 1.14: $A \mid R$

that if the scope is extended to R, it will become equivalent to a process that can do a tau action and then accept an input on b (Figure 1.15). Thus, even though $A \mathrel{\dot{\sim}} B$, we have that $A \mid R \mathrel{\dot{\not\sim}} B \mid R$.

1.2 The psi-calculi Framework

The psi-calculi framework was developed to unify many pi-calculus extensions. The goal is to have a generalised framework that can accommodate both the pi-calculus and as many as possible of its extensions, making the work of developing them less cumbersome and error-prone. The result is a syntax and semantics where some constructions are left as parameters to be specified by the developer of a new calculus. To make a new calculus from the framework, those parameters simply have to be filled in. The syntax is shown in Figure 1.16, and some syntactic sugar is applied in Figure 1.17.

The parameters consist of three data types:

T	the message (data) terms, ranged over by M, N
C	the conditions, ranged over by φ
A	the assertions, ranged over by Ψ

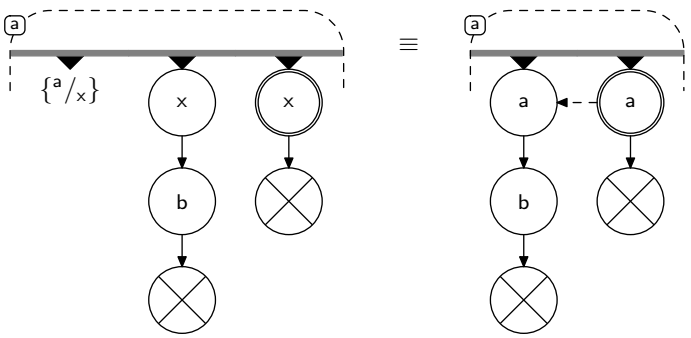


Figure 1.15: Because of the combination of scope extension and active substitution, $A \mid R$ is now equivalent to a process that can do a τ action and then accept an input on b

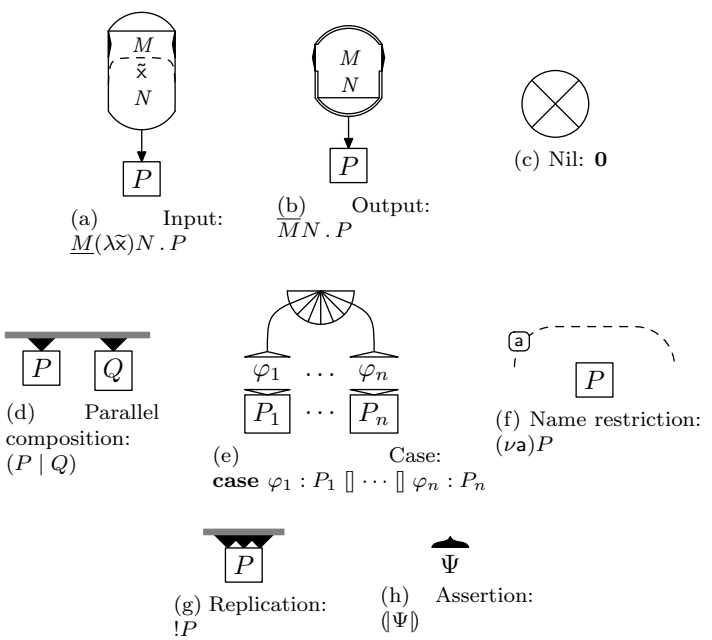
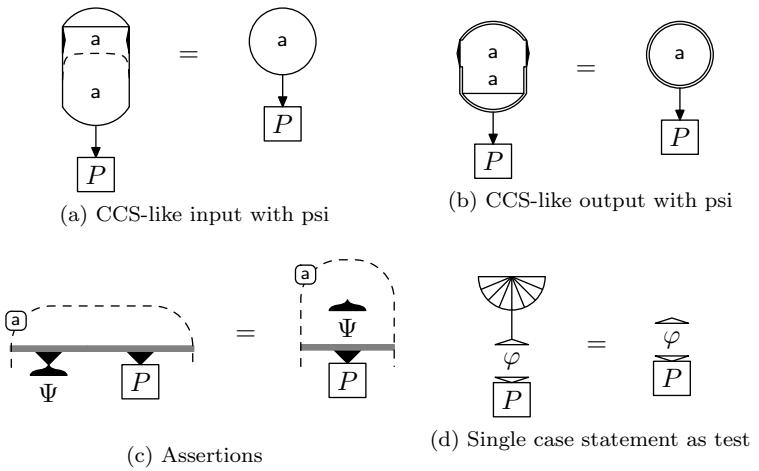


Figure 1.16: Psi syntax. A process P or Q can be any of the above.



(a) CCS-like input with psi

(b) CCS-like output with psi

(c) Assertions

(d) Single case statement as test

Figure 1.17: Syntactic sugar for graphical psi syntax

and four operators:

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

One of the most common types of pi-calculus extension is to expand the message and channel syntax beyond just names. Thus, the set of message terms in psi is one of the parameters. Another interesting parameter relates to the tests found in case statements (Figure 1.16e). The psi-calculi framework uses conditions instead of just the equality tests of the pi-calculus. Conditions are entailed by a logic which has an entailment operator, assertions, and a composition operator for the assertions. Conditions, entailment, composition and assertions are all parameters of the psi-calculi framework.

Assertions (Figure 1.16h) are similar to the active substitutions of applied pi. For the graphical syntax, we apply some syntactic sugar to unguarded (and thus active) assertions (Figure 1.17c).

Imagine as a simple example that we want to use a notion of **ok**-ness on message terms, where at any time in a process we can decide that a name is **ok**. Further, we want to say that a message term is only **ok** if all the names it contains are **ok**, and then at any time we want to be able to check for **ok**-ness of terms through the conditions. In a psi-calculus we can define **ok**-ness for message terms:

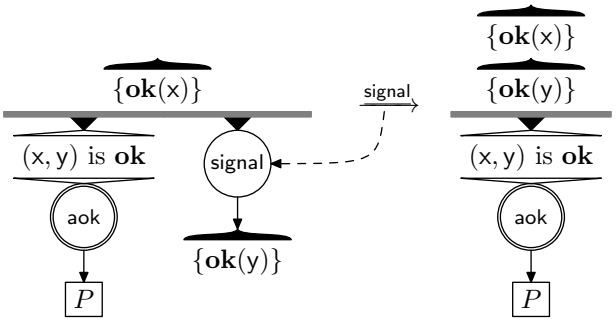


Figure 1.18: Simple example with **ok**-ness assertions

$$\mathbf{A} \supseteq \{\{\mathbf{ok}(a_1), \ldots, \mathbf{ok}(a_n)\} : a_i \in \mathcal{N}\}$$
$$\mathbf{C} \supseteq \{M \text{ is } \mathbf{ok} : M \in \mathbf{T}\}$$
$$\otimes = \cup$$
$$\mathbf{1} = \emptyset$$
$$\vdash \supseteq \{(\Psi, M \text{ is } \mathbf{ok}) : \forall a \in n(M).\mathbf{ok}(a) \in \Psi\}$$

Here, assertions are defined as sets of **ok** statements on names. Conditions are single statements of **ok**-ness. The composition operator for assertions is set union, the unit assertion is the empty set, and the entailment says that if all names of a term are **ok**, the term itself is **ok**.

Figure 1.18 shows a simple example of how **ok**-ness might be used. The aok output is behind a test of **ok**-ness for the term (x, y). The truth value of a condition is derived from the composition of all assertions that are unguarded and in scope with the process. To derive that (x, y) is **ok**, and thus pass the test in the example, both $\mathbf{ok}(\mathsf{x})$ and $\mathbf{ok}(\mathsf{y})$ must be contained in this assertion. In the initial state, only $\mathbf{ok}(\mathsf{x})$ is available. The $\mathbf{ok}(\mathsf{y})$ assertion exists, but is guarded behind the the signal input. A signal input later, both assertions are unguarded. When the condition is tested, the composition operator (in this case $\cup$) is used to compose the two assertions into the assertion $\{\mathbf{ok}(\mathsf{x}), \mathbf{ok}(\mathsf{y})\}$, which can then be used with the entailment operator to derive that (x, y) is **ok**.

Aside from the parameterised conditions, psi-calculi comes with required conditions of channel equivalence included in the framework. Channel equivalence conditions decide which terms identify the same channel: $M \leftrightarrows K$, where M and K are terms. Channel equivalences can be derived through the entailment operator. Then, if $M \leftrightarrows K$ holds in a particular context M and K are considered to be the same channel. In the pi-calculus, processes can only communicate by using the same name in input and output statements. Thus, two different names can never refer to the same channel.

As an example, to emulate the pi-calculus, the following can be included in

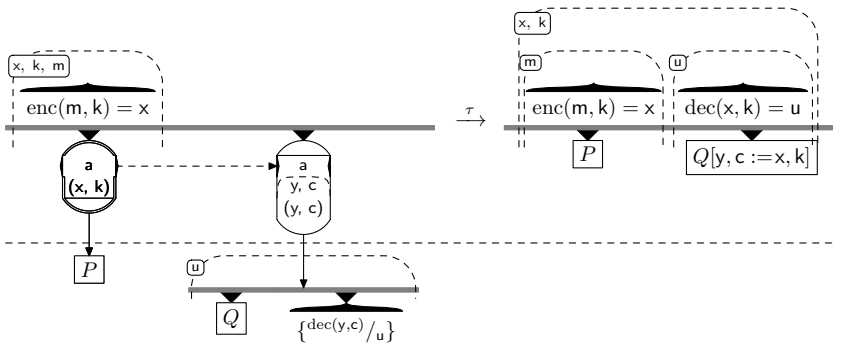


Figure 1.19: Cryptography with psi

the entailment relation:

$$\vdash \supseteq \{\mathbf{1}, \mathsf{a} \leftrightarrows \mathsf{a} : \mathsf{a} \in \mathcal{N}\}$$

Thus, two terms only refer to the same channel when they both consist of the same single name. Another useful way of defining channel equivalence would be from term equality:

$$\vdash \supseteq \{\mathbf{1}, M \leftrightarrows M : M \in \mathbf{T}\}$$

We could also make channel equivalence depend on the assertions, like the **ok**-ness conditions. For the rest of our examples though, we will assume the definition from term equality.

With the use of arbitrarily defined message structures for both objects and subjects comes also the use of pattern matching on input, causing the slightly different input syntax (Figure 1.16a). Formally, a message K matches the pattern $(\lambda\widetilde{x})N$ if $K = N[\widetilde{\mathsf{x}} := \widetilde{T}]$ for some sequence of terms $\widetilde{T}$. The idea of a pattern match is that a synchronisation can only happen if the incoming message on M matches the pattern defined by $(\lambda\widetilde{\mathsf{x}})N$ in the syntax. Here, $\widetilde{\mathsf{x}}$ is a list of names that bind into N and P. In N, the names $\widetilde{\mathsf{x}}$ work as placeholders that will be replaced in P by parts of the incoming message, if the pattern matches the message. As an example, the pattern $(\lambda\mathsf{a}, \mathsf{b})(\mathsf{a}, \mathsf{b})$ will only accept pairs for the input, and a and b will be substituted in P for anything the sender decides to construct that pair from. The pattern $(\lambda\mathsf{a})\mathsf{a}$ will accept any message as the substitute for a. In such cases we may omit the λ-binder from the syntax for brevity.

Let us consider in Figure 1.19 a small example from a psi-calculus that deals with cryptography in a similar way to the applied pi-calculus.

In this psi-calculus we introduce two kinds of message terms: $\mathrm{enc}(\mathsf{m}, \mathsf{k})$ and $\mathrm{dec}(\mathsf{m}, \mathsf{k})$, where $\mathrm{enc}(\mathsf{m}, \mathsf{k})$ is the message m encrypted by key k and where $\mathrm{dec}(\mathrm{enc}(\mathsf{m}, \mathsf{k}), \mathsf{k}) = m$.

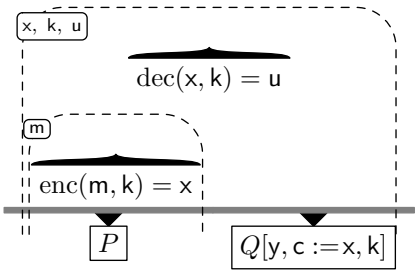


Figure 1.20: In P's environment, $\mathsf{m} = \mathsf{u}$ can now be deduced.

The example illustrates how encryption and decryption primitives can work in a psi-calculus. The encrypted message x is sent together with the key k on channel a. Anyone who receives it will be able to decrypt it. This is what the right-hand process decides to do by the assertion $\mathrm{dec}(\mathsf{y}, \mathsf{c}) = \mathsf{u}$. It receives a pair on (y, c) and then asserts the decryption of y with c. In this case it receives x and k, and so the active assertion becomes $\mathrm{dec}(\mathsf{x}, \mathsf{k}) = \mathsf{u}$, which means that $\mathsf{u} = \mathrm{dec}(\mathsf{x}, \mathsf{k})$. After this transition, m and u exist in different scopes, as do the respective assertions. If Q were to send u back to P, the scope of u would extend as seen in Figure 1.20, and the logic environment of P would be able to deduce from the two assertions that $\mathsf{m} = \mathsf{u}$. This condition could then be checked with a case statement, for example.

Thus, using scopes and assertions it is possible to model both local and global knowledge, and the custom logic system decides the possible consequences of that knowledge.

1.3 Extending psi-calculi

The psi-calculi framework supports very similar concepts to those implemented by many pi extensions, if not always the exact same semantics. The psi-calculi were developed to mitigate many of the issues often encountered when extending pi. Since then, two extensions to the framework have been developed: Higher-order psi and broadcast psi. It is likely that more will follow.

While a priority of psi-calculi is to be a framework that encompasses as many pi extensions as possible, another priority is that the framework should be as simple as possible. Since the semantics itself is one of the immutable parts of the framework, its definitions have been carefully chosen as a sensible minimum to make many basic extensions work. Thus, there are concepts from more elaborate pi extensions that cannot be directly represented in standard psi-calculi. Two such useful extensions are higher-order pi-calculus [Tho89, Tho93, San93] and broadcast pi-calculus [EM99, EM01](preceded by CBS in [Pra95]). Corresponding extensions have now been developed for the psi-calculi.

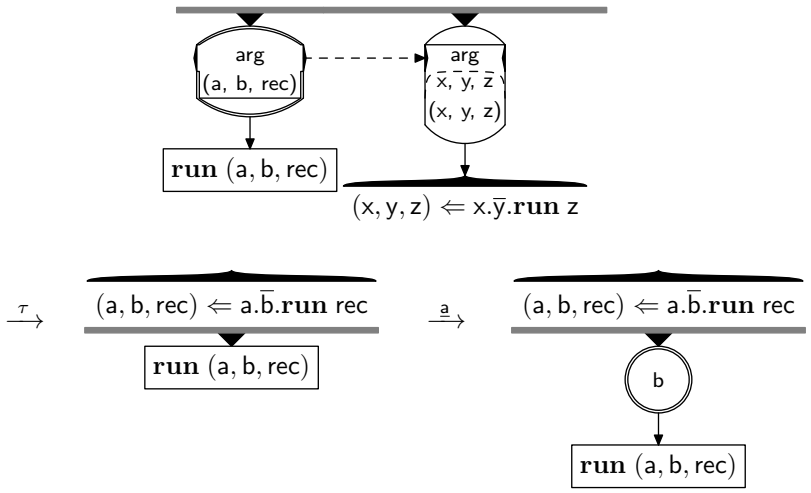


Figure 1.21: Recursive higher-order process

1.3.1 Higher-order Psi

The term "higher-order" refers to the ability of a concept to operate on itself. Thus, "higher-order thinking" refers to the practice of thinking about thinking, and "higher-order functions" refers to functions that can operate on functions. Higher-order concepts come with a great amount of power and flexibility, therefore most common programming languages include some forms of higher-order constructions. Higher-order constructions have also been attempted for the pi-calculus with some success [Tho89, Tho93, San93]. The idea in that case is to have processes that can send and receive not only names or message terms, but also process descriptions that may then be executed by the receiving process.

To some extent, higher-order constructions are already possible in psi. We can use the message terms to describe processes and thus send them around between other processes. There is no explicit construction for invoking such process descriptions however.

In higher-order psi, just as we use conditions to decide channel equivalences, we use conditions to specify higher-order processes. Higher-order process conditions are defined as $M \Leftarrow P$, where the term M works as an identifier for the process P. Then **run** M is new process syntax for invocation of P. For the purpose of the example in Figure 1.21, entailment is simply defined as:

$$\vdash \supseteq \{(\Psi, M \Leftarrow P) : M \Leftarrow P \in \Psi\}$$

In the example, the right-hand process has a guarded assertion with a higher-order process whose names are bound to the input. The left-hand process starts by sending the names it wants the higher-order process to use. Then the higher-order process assertion becomes active. The assertion now allows the **run** $(\mathsf{a}, \mathsf{b}, \mathsf{rec})$ process to act like the higher-order process.

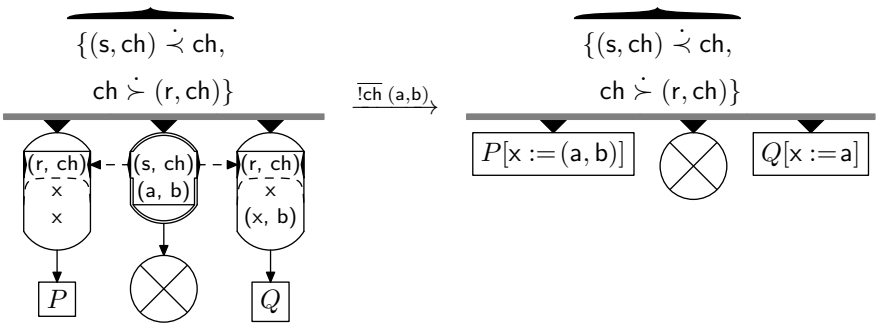


Figure 1.22: Broadcast process

1.3.2 Broadcast Psi

The standard psi-calculi framework implements only unicast, meaning one-to-one synchronisations. But many practical situations of communicating processes involve broadcasting on some level, such as wireless communication and message passing in computer clusters and multicore processors.

In broadcast psi, we make it possible for any number of processes receiving on some channel N to simultaneously synchronise with a single output on some channel M. For this to happen, the conditions $M \dot{\prec} K$ and $K \dot{\succ} N$ must hold for some K. K in this case acts a a proxy channel for the broadcast. The $M \dot{\prec} K$ (broadcast out) condition connects the channel M to some proxy channel K and allows broadcast output on M. The $K \dot{\succ} N$ (broadcast in) condition connects the proxy channel K to N and allows broadcast input on N. Broadcast in and broadcast out conditions serve much the same purpose as the channel equivalence conditions, but for broadcast connections.

In the example of Figure 1.22, we again use a very simple entailment relation for straightforward inference of the conditions:

$$\vdash \supseteq \{(\Psi, M \dot{\prec} K) : M \dot{\prec} K \in \Psi\} \cup \{(\Psi, M \dot{\succ} K) : M \dot{\succ} K \in \Psi\}$$

Note that the action of the transition is $\overline{\mathsf{!ch}}\,(\mathsf{a}, \mathsf{b})$. Two new actions were added for broadcast psi, broadcast input ($\underline{?M}\,N$) and broadcast output ($\overline{!M}\,N$). In the example, apart from synchronising with the two internal processes, the action signifies that any number of external processes may have synchronised with it as well.

In the version of broadcast implemented here, synchronisation is both unreliable and nondeterministic. An output does not have to synchronise with every possible input. In fact, the output of the example might just as well have been received by no inputs, causing a tau action instead.

1.4 Proof Mechanisation

Isabelle is a mechanical proof assistant [NPW02]. In the field of mathematics and theoretical computer science it is still common practice to conduct mathematical proofs on paper, and it is a practice that has served us well for many years. Unfortunately, it is also often fraught with pitfalls. The bigger the proof, the greater the risk that some important detail has been overlooked. Furthermore it is very easy, in fact sometimes essential, to skip less interesting parts of a proof to keep the size down in a paper proof so that there is a chance that others will at least read the ostensibly interesting parts.

Mechanical proof assistance has the potential to change this. With a proof assistant like Isabelle, it suffices to trust the same small core of code for any conducted proof. In the past this field may have been held back somewhat by a lack of computing power and good heuristics for automation, but today computing power, algorithms and proof libraries have come far enough that the use of mechanical proof assistants is often a viable improvement on the development of mathematical proofs.

The psi-calculi frameworks, their definitions and results, have been implemented using the Isabelle proof assistant.

1.4.1 Nominal Isabelle

The specific choice of using Isabelle for the psi-calculi proofs was made because of the existence of the HOL-nominal library for Isabelle [UT05]. Nominal theory [Pit03] deals with the effects and consequences of binders in term constructions. The HOL-nominal library contains lemmas and methods to help deal with the existence of names in terms (support and freshness) and the use of α-conversion. Further, HOL-nominal automates the generation of many lemmas needed to deal with functions and inductive datatypes containing binders.

1.4.2 Locales

Locales [Bal04] are also an important feature of Isabelle in the context of the psi-calculi theories. Locales are what makes it possible to easily implement the abstraction of the parameters in psi-calculi. A locale can be seen as a proof environment with certain parameters that we can make assumptions about within that environment. Outside the environment, you the locale can then be instantiated with concrete instances of those parameters, adhering to the assumptions of the locale. This will give all the lemmas of the locale for those specific parameters.

In psi-calculi, the terms, assertions, conditions and entailment parameters are all implemented using locales.

1.4.3 Psi-calculi formalisation

The formalisation of the fundamental psi-calculi framework was worked out by Jesper Bengtson as outlined in his PhD thesis, Formalising process calculi [Ben10].

Both the paper on broadcast psi (Chapter 2) and the paper on higher-order psi (Chapter 3) contain one or more notes to the effect that the proofs have been checked in Isabelle. The amount of exposition given to the Isabelle formalisation does not always do justice to the amount of work required to be able to write such statements. We will be coming back to this point in Chapter 4.

The Isabelle proofs for the psi-calculi framework consists of a number of theory files, the most significant of which are `agent.thy` (definition of syntax), `frame.thy` (definition of frames), `semantics.thy` (definition of semantics), `simulation.thy` and `bisimulation.thy` (definition and properties of bisimulation), `simStructCong.thy` and `bisimStructCong.thy` (congruence results). Again, Chapter 4 will further elaborate on this.

As mentioned in the conclusion of the higher-order psi paper, the two extensions were even shown to be compatible. After both extensions were completed, merging the Isabelle formalisations to a higher-order broadcast psi-calculi framework turned out to require only a day's work by Johannes Åman Pohjola, including the standard results (bisimulation properties and congruence results, outlined in the following section).

1.5 Summary of Papers

This section contains the noteworthy definitions and theorems of both the broadcast and higher-order psi papers. Section 1.5.1 contains definitions and theorems shared by the two frameworks. Section 1.5.2 and 1.5.3 contains definitions and theorems specific to broadcast psi and higher-order psi respectively.

1.5.1 Shared

In the beginning of Section 2 in both papers, the relevant definitions of nominal sets, support, freshness ($\#$), assertion equivalence, psi-calculus parameters ($\mathbf{T}$, $\mathbf{C}$, $\mathbf{A}$, $\leftrightarrows$, $\otimes$, $\mathbf{1}$, $\vdash$), requisites on the parameters, frames ($(\nu\widetilde{b}_F)\Psi_F$), frame equivalence and frame derivation ($\mathcal{F}(P)$) are recapitulated.

Definition 7 (Psi-calculus agents)**.** *Given valid psi-calculus parameters as in Definition 1 of either paper, the psi-calculus* agents*, ranged over by* $P, Q, \ldots$*,*

are of the following forms.

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

In the Input $\underline{M}(\lambda\widetilde{x})N.P$ *we require that* $\widetilde{x} \subseteq \mathrm{n}(N)$ *is a sequence without duplicates, and any name in* $\widetilde{x}$ *binds its occurrences in both* N *and* P*. Restriction binds* a *in* P*. An assertion is* guarded *if it is a subterm of an Input or Output. In a replication* $!P$ *there may be no unguarded assertions in* P*, and in* **case** $\varphi_1 : P_1 \,[]\, \cdots \,[]\, \varphi_n : P_n$ *there may be no unguarded assertion in any* P_i*. The data type for processes is* $\mathbf{P}$*.*

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

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

We write $\overline{M}N$ *for output actions with no binders. For actions we refer to* M *as the subject and* N *as the object. We define* $\mathrm{bn}(\overline{M}\,(\nu\tilde{a})N) = \tilde{a}$*, and* $\mathrm{bn}(\alpha) = \emptyset$ *if* α *is an input or* τ*.*

Definition 9 (Transitions)**.** *A* transition *is written* $\Psi \rhd P \xrightarrow{\alpha} P'$*, meaning that in the environment* Ψ *the well-formed agent* P *can do an* α *to become* P'*. We write* $P \xrightarrow{\alpha} P'$ *without an assertion to mean* $\mathbf{1} \rhd P \xrightarrow{\alpha} P'$*.*

Definition 10 (Strong bisimulation)**.** *A* strong bisimulation $\mathcal{R}$ *is a ternary relation on assertions and pairs of agents such that* $\mathcal{R}(\Psi, P, Q)$ *implies*

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

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

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

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

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

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

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

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

Table 1.1: Structured operational semantics. Symmetric versions of Com and Par are elided. In the rule Com we assume that $\mathcal{F}(P) = (\nu\widetilde{b}_P)\Psi_P$ and $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$ where $\widetilde{b}_P$ is fresh for all of $\Psi, \widetilde{b}_Q, Q, M$ and P, and that $\widetilde{b}_Q$ is similarly fresh. In the rule Par we assume that $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$ where $\widetilde{b}_Q$ is fresh for Ψ, P and α. In Open the expression $\tilde{a} \cup \{b\}$ means the sequence $\tilde{a}$ with b inserted anywhere.

We sometimes refer to "the standard results" in the context of a psi-calculi framework. The standard results refer, in short, to the basic congruence and structural properties we derive from the notion of strong bisimilarity ($\dot{\sim}_\Psi$, where we write $\dot{\sim}$ for $\dot{\sim}_{\mathbf{1}}$). The standard results are generally considered necessary results to have for a useful psi-calculi framework. They are presented here in Theorem 11 and 12.

Theorem 11. *Bisimilarity is preserved by operators. For all Ψ:*

1. $P \dot{\sim}_\Psi Q \Longrightarrow P \mid R \dot{\sim}_\Psi Q \mid R$.
2. $P \dot{\sim}_\Psi Q \Longrightarrow (\nu\mathsf{a})P \dot{\sim}_\Psi (\nu\mathsf{a})Q$.
3. $P \dot{\sim}_\Psi Q \Longrightarrow !P \dot{\sim}_\Psi !Q$.

4. $\forall i.P_i \mathrel{\dot{\sim}}_\Psi Q_i \Longrightarrow \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \mathrel{\dot{\sim}}_\Psi \mathbf{case}\ \widetilde{\varphi} : \widetilde{Q}$.

5. $P \mathrel{\dot{\sim}}_\Psi Q \Longrightarrow \overline{M}\ N.P \mathrel{\dot{\sim}}_\Psi \overline{M}\ N.Q$.

6. $(\forall \widetilde{L}.\ P[\widetilde{a} := \widetilde{L}] \mathrel{\dot{\sim}}_\Psi Q[\widetilde{a} := \widetilde{L}]) \implies \underline{M}(\lambda\widetilde{a})N.P \mathrel{\dot{\sim}}_\Psi \underline{M}(\lambda\widetilde{a})N.Q$

Theorem 12 (Congruence). *$P \sim_\Psi Q$ means that for all $\widetilde{x}, \widetilde{M}$ it holds $P[\widetilde{x} := \widetilde{M}] \mathrel{\dot{\sim}}_\Psi Q[\widetilde{x} := \widetilde{M}]$, and we write $P \sim Q$ for $P \sim_{\mathbf{1}} Q$. $\sim_\Psi$ is a congruence for all Ψ, and $\sim$ satisfies the following structural laws:*

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

1.5.2 Broadcast psi

The additions of broadcast psi to psi-calculi are shown in Definitions 13, 14 and 15, and the new semantic rules in Table 1.2. The mechanical proofs have been extended accordingly, showing that the standard results hold for the extension.

Definition 13 (Extra predicates for broadcast).

$$
\begin{array}{rcll}
\dot{\prec} & : & \mathbf{T} \times \mathbf{T} \to \mathbf{C} & \textit{Output Connectivity} \\
\dot{\succ} & : & \mathbf{T} \times \mathbf{T} \to \mathbf{C} & \textit{Input Connectivity}
\end{array}
$$

Definition 14 (Requirements for broadcast).

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

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

Definition 15 (Transitions of Broadcast psi). *To the actions of psi-calculi we add broadcast input, written $\underline{?K}\ N$ for a reception of N on K, and broadcast output, written $\overline{!K}\ (\nu\widetilde{a})N$ for a broadcast of N on K, with names $\widetilde{a}$ fresh in K. As before, we omit $(\nu\widetilde{a})$ when $\widetilde{a}$ is empty, and in examples we omit N when it is not relevant. The transitions of well-formed agents are defined inductively in Tables 1.2 and 1.1, where we let α range over both unicast and broadcast actions.*

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

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

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

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

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

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

Table 1.2: Operational broadcast semantics. A symmetric version of BrCom is elided. In rules BrCom and BrMerge we assume that $\mathcal{F}(P) = (\nu\widetilde{b}_P)\Psi_P$ and $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$ where $\widetilde{b}_P$ is fresh for $P, \widetilde{b}_Q, Q, K$ and Ψ, and that $\widetilde{b}_Q$ is fresh for $Q, \widetilde{b}_P, P, K$ and Ψ.

1.5.3 Higher-order psi

The basic additions of higher-order psi to psi-calculi are shown in Definitions 16, 17, 18 and 19. From these, the standard results are derived. Beyond the basic additions, the paper also extends the theory with two additional notions: Canonical higher-order psi-calculi (Definition 20), a way of lifting any psi-calculus instance to a higher-order psi instance, and higher-order bisimulation (Definition 22), a more inclusive notion of bisimulation in the context of higher-order psi. The standard results are adapted for higher-order bisimulation in Theorem 26 and 29.

Definition 16 (Extra predicate for higher-order assignments)**.** *In a higher-order psi-calculus we use one particular nominal datatype of clauses:*

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

Definition 17 (Extended entailment relation)**.** *The entailment relation is extended to* $\vdash \subseteq \mathbf{A} \times (\mathbf{C} \uplus \mathbf{Cl})$*, where we write* $\Psi \vdash \varphi$ *for* $\Psi \vdash (0, \varphi)$ *and* $\Psi \vdash M \Leftarrow P$ *for* $\Psi \vdash (1, M \Leftarrow P)$*.*

Definition 18 (Higher-order agents)**.** *The higher-order agents in a psi-calculus extend those of an ordinary calculus with one new kind of agent:*

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

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

Definition 19 (Higher-order transitions)**.** *The transitions in a higher-order psi-calculus are those that can be derived from the rules in Table 1.1 plus the one additional rule*

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

Definition 20 (Canonical higher-order psi-calculi)**.** *Let a psi-calculus* $\mathcal{C}$ *be defined by the parameters* $\mathbf{T}, \mathbf{C}, \mathbf{A}, \leftrightarrows, \otimes, \mathbf{1}, \vdash$*. Let* $\mathbf{S}$ *be the set of finite sets of parametrised clauses as defined above. The canonical higher-order psi-calculus* $\mathcal{H}(\mathcal{C})$ *extends* $\mathcal{C}$ *by adding the* $\mathbf{run}\ M$ *agent and its semantic rule, and is defined by the parameters* $\mathbf{T}_\mathcal{H}, \mathbf{C}_\mathcal{H}, \mathbf{A}_\mathcal{H}, \leftrightarrows_\mathcal{H}, \otimes_\mathcal{H}, \mathbf{1}_\mathcal{H}, \vdash_\mathcal{H}$ *where*

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

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

Theorem 21. *For all* $\mathcal{C}$ *and* $\bullet\langle\bullet\rangle$*,* $\mathcal{H}(\mathcal{C})$ *is a higher-order psi-calculus.*

The only difference between Definition 10 and 22 is the definition of static equivalence, which has been extended for higher-order bisimulation.

Definition 22 (HO-Bisimulation)**.** *A* strong HO-bisimulation $\mathcal{R}$ *is a ternary relation between assertions and pairs of agents such that* $(\Psi, P, Q) \in \mathcal{R}$ *implies all of*

1. *Static equivalence:*

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

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

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

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

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

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

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

Theorem 23. *In a higher-order psi-calculus, for all assertion guarded* P, Q *and terms* M *with* $\mathrm{n}(P, Q) \subseteq \mathrm{n}(M)$ *with characteristic assertions* $\Psi^{M \Leftarrow P}$ *and* $\Psi^{M \Leftarrow Q}$, *it holds that*

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

Theorem 24. $\Psi \rhd P \mathrel{\dot{\sim}} Q \implies \Psi \rhd P \mathrel{\dot{\sim}}^{\text{HO}} Q$

Corollary 25. $\mathrel{\dot{\sim}}^{\text{HO}}$ *satisfies all structural laws of Theorem 12.*

Theorem 26. *For all* Ψ*:*

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

Definition 27. $\Psi \rhd P \sim^{\text{HO}} Q$ *iff for all sequences* σ *of substitutions it holds that* $\Psi \rhd P\sigma \mathrel{\dot{\sim}}^{\text{HO}} Q\sigma$. *We write* $P \sim^{\text{HO}} Q$ *for* $\mathbf{1} \rhd P \sim^{\text{HO}} Q$.

Theorem 28. *For every* Ψ, *the binary relation* $\{(P, Q) : \Psi \rhd P \sim^{\text{HO}} Q\}$ *is a congruence.*

Theorem 29. $\sim^{\text{HO}}$ *satisfies the following structural laws:*

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

1.6 Related Work

This is a summary of related works of the papers. We start with a brief summary of related works of broadcast psi. That first summary is derived directly from the comprehensive overview found in Section 6 of the paper in Chapter 2. We continue with related works of higher-order psi. The higher-order psi paper of Chapter 3 is not as comprehensive on related works as the broadcast psi paper, so we have chosen to expand on a few of the citations used therein. Next comes related works of the graphical syntax used in the introduction, and last, we summarise on the related works of Chapter 4.

1.6.1 Broadcast psi

The beginnings of process calculi with broadcast traces back to the early 1980's. Milner developed SCCS [Mil83] as a generalisation of CCS [Mil80] to include multiway communication, of which broadcast can be seen as a special case. The first process calculus to seriously consider broadcast with an asynchronous parallel composition was CBS [Pra95]. CBS was later extended to the pi-calculus in the $b\pi$ formalism [EM99].

The first process calculus created with wireless networks in mind was probably CBS$^\sharp$ [NH06]. CBS$^\sharp$ has been followed by several similar calculi: CWS [MS06, LS10] focuses on modelling low level interference. CMAN [God07] is a high level formalism extended with data types, in the same way as the applied pi-calculus extends the original pi-calculus. In the ω-calculus [SRS10], emphasis is on expressing connectivity using sets of group names. RBPT [GFM08] is similar and uses an alternative technique to represent topology changes, leading to smaller state spaces.

$bA\pi$ [God10] is an extension of the applied pi-calculus [AF01] with broadcast, where connectivity information appears explicitly in the process terms and can change non-deterministically during execution. It suffers however, from the same problem as the applied pi-calculus, in that labelled bisimilarity is not compositional.

1.6.2 Higher-order psi

In [Tho93], Thomsen made the first attempt at encoding a higher-order process calculus into π-calculus. He used Plain CHOCS, and showed their equivalence in expressive power by showing that the two calculi can simulate each other. He also uses the Plain CHOCS to give a semantics to a small object oriented language, demonstrating the use of higher-order constructions in connecting process calculi more closely to programming languages. In [San93], Sangiorgi shows a similar result by defining HOπ, probably the first actual extension of the pi-calculus with higher-order constructs, and encoding it into the pi-calculus. This strengthens the expressiveness result of [Tho93], as Plain CHOCS is a second-order process calculus, while HOπ is ω-order. In this terminology, the higher-order psi calculi framework allows the instantiation of ω-order pi-calculus extensions.

More recent work includes development of the Kell Calculus [SS04], which in some respects is similar to higher-order psi. It is a parameterised framework encompassing a family of higher-order calculi. It has just one parameter though, in the language of input patterns. Where higher-order psi-calculi aim to be as general as possible in a higher-order context, the Kell Calculus appears more specifically geared towards the modelling of distributed systems.

1.6.3 Graphical Syntax

The idea of having a graphical syntax for process calculi is not new. In fact, there appears to be a variety of very different ways of doing it. The graphical syntax presented in this thesis most resembles the one presented in [PCC06], and indeed, some inspiration is taken from that paper, which deals specifically with the Stochastic pi-calculus. The tile formats presented in [FM00] deal with an alternative way of defining semantics for calculi. The motivation behind the interaction diagrams of [Par95] is perhaps similar to my own, though the intuition behind them is very different. The bigraphs of [Mil01] is a generalisation applicable to a variety of process calculi, and the intuition behind bigraphs is again quite different.

1.6.4 Extending psi-calculi

Psi-calculi use nominal theory [Pit03, UT05] to represent terms with binders. De Bruijn indices [dB72] is another common way of presenting such terms. Choosing either of the two appears in many respects to be a tradeoff between the difficulties of handling either α-conversion or terms that change. Additionally, nominal theory presents a more human-readable format than de Bruijn indices.

Recent work by Whiteside, Aspinall, Dixon and Grov [WADG11] introduce proof refactoring for Isabelle, which may provide ways to alleviate practices of copying and pasting often found in our own proofs.

On the subject of extending large proofs, it is also worth mentioning the seL4 microkernel project, where Matichuk and Murray make use of a technique they call Extensible Specifications [MM12] for writing specifications with multiple levels of abstractions.

1.7 Conclusion

1.7.1 Summary of Results of Papers

In the papers that follow, we have defined both broadcast psi and higher-order psi as clean extensions to the psi-calculi framework, in the sense that the original syntax and semantics of the framework remain, and no capabilities are restricted or taken away in either extension. Using Isabelle/Nominal, we have proved that the standard congruence and structural properties of bisimilarity hold also in the extensions.

For broadcast psi-calculi, we have also demonstrated its expressive power by modelling a simplified version of the LUNAR protocol for route discovery in wireless ad-hoc networks, and verified a basic correctness property of the protocol.

For higher-order psi-calculi, we have also developed a more inclusive notion of higher order bisimulation to complement the standard notion, with mechanical proofs of the usual structural properties.

1.7.2 Contributions

This section delineates my specific contributions to the papers in Chapter 2 and 3:

- Johannes Borgström, Shuqin Huang, Magnus Johansson, Palle Raabjerg, Björn Victor, Johannes Åman Pohjola, and Joachim Parrow. Broadcast Psi-calculi with an Application to Wireless Protocols. *Proceedings of Software Engineering and Formal Methods*, pages 74-89, 2011.
- Joachim Parrow, Johannes Borgström, Palle Raabjerg, and Johannes Åman Pohjola. Higher-order psi-calculi. Submitted to Mathematical Structures in Computer Science.

For broadcast psi I was responsible for most of the work in proving mechanically the correctness of the aforementioned standard congruence and structural properties of bisimilarity.

For higher-order psi I was also involved in most of the work in proving mechanically those same standard results.

1.7.3 Impact

The broadcast psi and higher-order psi extensions are the first two contributions to what may be called an actual family of psi-calculi frameworks. They neatly

expand the range of pi-calculus variants encompassed by psi-calculi theory, and they demonstrate another benefit of having a mechanical proof repository, apart from affording more confidence: reusability. Not only is it possible to expand proofs as needed without having to worry that details are overlooked in modifying the existing proof, it is also often possible to modify proofs to work in similar contexts.

Contexts where the broadcast psi extension may be useful includes for example wireless communication, as demonstrated in the paper. Another interesting use may turn out to be that of systems biology. Variants of the pi-calculus have already been demonstrated to be useful in this context [PPQ05], and biological systems are certainly not strangers to information broadcasting. Work is also ongoing in development of a reliable version of the broadcast psi framework, allowing us to model constructions in reliable broadcast settings, like multicore communication.

Contexts where the higher-order psi extension seems most immediately useful may include subjects such as the modelling of mobile code and perhaps simply the more accurate modelling of programs with recursive primitives.

1.8 Future Work

1.8.1 GUI for Psi Workbench

The motivation for the graphical syntax used in the introductory chapter is readability, and when it comes to reading order, it appears that showing descriptions as abstract syntax trees is actually a good choice. Every branching of the tree is either a parallelisation or a case statement. So reading the tree top-down, any processes standing side-by-side are either parallel definitions or parts of a case statement, and reading downwards is a linear progression through the future behaviour of the processes.

The Psi Workbench [Gut11] would likely benefit from a graphical user interface, and the graphical representation shown in this thesis lends itself well to animation. Plans for future work include a graphical user interface for the Workbench which may include usage of the graphical syntax presented here, possibly in combination with interaction diagram and bigraph representations for variety.

1.8.2 Verification of Multicore Algorithms

With the higher-order psi and perhaps particularly the broadcast psi extension, we are getting closer to having tools that may be applicable to multicore algorithms and protocols, such as work balancing, memory sharing protocols and parallelised algorithms. Development of a reliable broadcast psi is another currently ongoing step in that direction. With this, we shall start looking seriously at applications to such multicore constructions.

1.8.3 Translating Psi-calculi to Erlang

Another interesting subject is the question of mechanical psi-calculi translation to a real programming language. This kind of translation can be imagined as a more reliable way of implementing network or multicore protocols modelled in psi. Mechanical translation would at least be more reliable than manually translating such models. Just as when we write proofs manually, we are prone to mistakes when converting manually descriptions of protocols to code. Mechanical translations could therefore be beneficial. It may even be possible to verify the translation to some degree, though as one of the points of modelling is that it allows for abstractions, it seems unlikely that we could translate models to complete, working code. Higher-order psi may contribute to better translations of such constructs as functions, allowing for very similar abstractions and the possibility of recursion. Erlang [Arm07] in particular seems like a good candidate for such translation, as it is developed for message passing algorithms and uses the same abstractions for both network and multicore communication.

Chapter 2

Broadcast Psi Calculi

Paper I

Broadcast Psi-calculi with an Application to Wireless Protocols

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

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

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

1 Introduction

Psi-calculi is a parametric framework for extensions of the pi-calculus, with arbitrary data structures and logical assertions for facts about data. In psi-calculi (described in Section 2) the purity of the semantics is on par with the original pi-calculus, the generality and expressiveness exceeds many earlier extensions of the pi-calculus, and the meta-theory is proved correct once and for all using the interactive theorem prover Isabelle/Nominal [26].

In order to model wireless communication used in WSN (Wireless Sensor Network) and MANET (Mobile Ad-hoc Network) applications, the concept of broadcast communication is needed, where one transmission can be received by several processes. Broadcast communication cannot be encoded in the pi-calculus [5]; we extend the psi-calculi framework with broadcast primitives (Section 3). The broadcast primitives are added using new operational actions and rules, and new connectivity predicates. We formally prove the congruence properties of bisimilarity and the soundness of structural equivalence laws using the Isabelle/Nominal theorem prover.

The connectivity predicates allow us to model systems with limited reachability, for instance where a transmitter only reaches nodes within a certain range, and systems with changing reachability, for instance due to physical mobility of nodes. In Section 4, we present a technique for treating different generations of connectivity information. Broadcast channels can be globally visible or have limited scope. Scoped channels can be protected from externally imposed connectivity changes, while permitting connectivity changes by processes within the scope of the channel.

We demonstrate the expressive power of the resulting framework in Section 5, where we provide a model of the LUNAR protocol for routing in ad-hoc wireless networks [24]. The model follows the specification closely, and demonstrates several features of the psi-calculi framework: both unicast and broadcast communication, application-specific data structures and logics, classic unstructured channels as well as pairs corresponding to MAC address and port selector. Our model is significantly more succinct than earlier work [28,27] (ca 30 vs 250 lines). We show an expected basic reachability property of the model: if two network nodes, a sender and a receiver, are both in range of a third node, but not within range of each other, the LUNAR protocol can find a route and transparently handle the delivery of a packet from the sender to the receiver.

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

2 Psi-calculi

This section is a brief recapitulation of psi-calculi; for a more extensive treatment including motivations and examples see [3,4].

We assume a countably infinite set of atomic *names* $\mathcal{N}$ ranged over by $a, b, \ldots, z$. Intuitively, names will represent the symbols that can be scoped, and also represent symbols acting as variables in the sense that they can be subject to substitution. A *nominal set* [18,6] is a set equipped with a formal notion of what it means for a name a to occur in an element X of the set, written $a \in \mathrm{n}(X)$ (often pronounced as "a is in the support of X"). We write $a\#X$, pronounced "a is fresh for X", for $a \notin \mathrm{n}(X)$, and if A is a set of names we write $A\#X$ to mean $\forall a \in A \,.\, a\#X$. In the following $\tilde{a}$ means a finite sequence of names, $a_1, \ldots, a_n$. The empty sequence is written ϵ and the concatenation of $\tilde{a}$ and $\tilde{b}$ is written $\tilde{a}\tilde{b}$. When occurring as an operand of a set operator, $\tilde{a}$ means the corresponding set of names $\{a_1, \ldots, a_n\}$. We also use sequences of other nominal sets in the same way.

A *nominal datatype* is a nominal set together with a set of functions on it. In particular we shall consider substitution functions that substitute elements for names. If X is an element of a datatype, the *substitution* $X[\tilde{a} := \tilde{Y}]$ is an element of the same datatype as X. There is considerable freedom in the choice of functions and substitutions; see [3,4] for details.

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

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

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

and substitution functions $[\widetilde{a} := \widetilde{M}]$, substituting terms for names, on each of $\mathbf{T}$, $\mathbf{C}$ and $\mathbf{A}$.

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

We say that two assertions are equivalent, written $\Psi \simeq \Psi'$ if they entail the same conditions, i.e. for all φ we have that $\Psi \vdash \varphi \Leftrightarrow \Psi' \vdash \varphi$. We impose certain requisites on the sets and operators. In brief, channel equivalence must be symmetric and transitive, $\otimes$ must be compositional with regard to $\simeq$, and the assertions with $(\otimes, \mathbf{1})$ form an abelian monoid modulo $\simeq$. For details see [3].

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

Alpha equivalent frames are identified. We define $F \vdash \varphi$ to mean that there exists an alpha variant $(\nu\widetilde{b})\Psi$ of F such that $\widetilde{b}\#\varphi$ and $\Psi \vdash \varphi$. We also define $F \simeq G$ to mean that for all φ it holds that $F \vdash \varphi$ iff $G \vdash \varphi$.

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

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

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

The agent $\mathbf{case}\ \varphi_1 : P_1 \,[]\, \cdots \,[]\, \varphi_n : P_n$ is sometimes abbreviated as $\mathbf{case}\ \widetilde{\varphi} : \widetilde{P}$, or if $n = 1$ as $\mathbf{if}\ \varphi_1\ \mathbf{then}\ P_1$.

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

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

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

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

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

Table 1. Structured operational semantics. Symmetric versions of Com and Par are elided. In the rule Com we assume that $\mathcal{F}(P) = (\nu\widetilde{b}_P)\Psi_P$ and $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$ where $\widetilde{b}_P$ is fresh for all of $\Psi, \widetilde{b}_Q, Q, M$ and P, and that $\widetilde{b}_Q$ is similarly fresh. In the rule Par we assume that $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$ where $\widetilde{b}_Q$ is fresh for Ψ, P and α. In Open the expression $\tilde{a} \cup \{b\}$ means the sequence $\tilde{a}$ with b inserted anywhere.

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

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

The *actions* ranged over by α, β are of the following three kinds: Output $\overline{M}(\nu\tilde{a})N$ where $\alpha \subseteq \mathrm{n}(N)$, Input $\underline{M}\, N$, and Silent τ. Here we refer to M as the *subject* and N as the *object*. We define $\mathrm{bn}(\overline{M}(\nu\tilde{a})N) = \tilde{a}$, and $\mathrm{bn}(\alpha) = \emptyset$ if α is an input or τ. We also define $\mathrm{n}(\tau) = \emptyset$ and $\mathrm{n}(\alpha) = \mathrm{n}(M) \cup \mathrm{n}(N)$ for the input and output actions.

Definition 3 (Transitions).

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

Agents, frames and transitions are identified by alpha equivalence. In a transition the names in $\mathrm{bn}(\alpha)$ bind into both the action object and the derivative, therefore

$\mathrm{bn}(\alpha)$ is in the support of α but not in the support of the transition. This means that the bound names can be chosen fresh, substituting each occurrence in both the object and the derivative.

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

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

$$\Psi \rhd P \xrightarrow{\alpha} P' \Longrightarrow \Psi \rhd Q \xrightarrow{\alpha} Q' \wedge \mathcal{R}(\Psi, P', Q')$$

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

Strong bisimulation is preserved by all operators except input prefix and satisfies the expected algebraic laws such as scope extension, for details see [3,4].

3 Broadcast semantics

In this section we extend the unicast psi-calculi of the previous section with a broadcast semantics that models wireless (i.e., synchronous and unreliable) broadcast. As an example, assume that the connectivity information Ψ allows receivers M_1 and M_2 to listen to channel K. We would then expect the following transition: $\Psi \rhd \overline{K}\,N.P \mid \underline{M_2}(x).Q \mid \underline{M_3}(y).R \xrightarrow{\overline{K}\,N} P \mid Q[x := N] \mid R[y := N]$.

To allow connectivity to depend on assertions, and to permit broadcast channels to be computed at run-time, we assume a psi-calculus with the following extra predicates:

Definition 5 (Extra predicates for broadcast).

$$\begin{array}{ll} \dot{\prec} : \mathbf{T} \times \mathbf{T} \to \mathbf{C} & \textit{Output Connectivity} \\ \dot{\succ} : \mathbf{T} \times \mathbf{T} \to \mathbf{C} & \textit{Input Connectivity} \end{array}$$

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

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

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

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

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

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

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

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

Definition 6 (Requirements for broadcast).

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

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

The rule BrOut, allows transmission on a broadcast channel K that the subject M of an output prefix is out-connected to. Similarly, the rule BrIn allows input from a broadcast channel K that the subject M of an input prefix is in-connected to. When two parallel processes both receive a broadcast on the same channel, the rule BrMerge combines the two actions. This rule is necessary to ensure the associativity of parallel composition. After a broadcast communication using BrCom, the resulting action is the original transmission. This is different from the unicast Com rule, where a communication yields an internal action τ. Finally, rule BrClose states that a broadcast transmission does not reach beyond its scope. This allows for broadcasting on restricted channels. Dually, the Res rule (of Table 1) ensures that broadcast receivers on restricted channels cannot proceed unless a message is sent. We allow the Open rule to also apply to broadcast output actions, in order to communicate scoped data. The Par rule allows for broadcasts to bypass a process, as in most other broadcast calculi for wireless systems.

We have developed a meta-theory for broadcast psi-calculi. In the following we restrict attention to well-formed agents. The expected compositionality properties of strong bisimilarity hold:

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

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

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

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

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

The standard structural laws hold for strong congruence.

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

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

Theorems 8, 10 and 11 give us assurance that any broadcast psi-calculus has a compositional labelled bisimilarity that respects important structural laws. The proofs [21] are formally verified in the interactive theorem prover Isabelle/Nominal. The full formalisation of broadcast psi-calculi amounts to ca 33000 lines of Isabelle code, of which about 21000 lines are re-used from our earlier work [4]. The fact that the BrComm rule defers the closing of the communication to BrClose causes most of the added complications.

4 Modelling network topology changes

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

The idea is to allow for different generations of assertions by tagging each part of an assertion with a generation number. Only the most recent generation

is used; a generation is made obsolete by adding an assertion from a later generation. We here consider broadcast connectivity, but this technique can also be used in other scenarios where there is a need to retract assertions.

In the following we assume a set of broadcast terms $\mathbf{B} \subseteq \mathbf{T}$; we let B, B' range over elements of $\mathbf{B}$. For simplicity, we assume that no rewriting happens in broadcast output, i.e., that $\dot{\prec}$ is the equality relation of $\mathbf{B}$. Assertions are finite sets of connectivity information, labelled with a generation, with set union as assertion composition $\otimes$ and the empty set as the unit assertion. Formally,

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

$\Psi \vdash \mathsf{currentGeneration}(i)$ if $\forall \langle j, *\rangle \in \Psi \,.\, j \leq i$ and $\exists \langle j, *\rangle \in \Psi \,.\, i = j$
where $*$ is $B \mathrel{\dot{\succ}} B'$ or 0
$\Psi \vdash B \mathrel{\dot{\prec}} B'$ if $B = B'$
$\Psi \vdash B \mathrel{\dot{\succ}} B'$ if $\langle i, B \mathrel{\dot{\succ}} B'\rangle \in \Psi$ and $\mathsf{n}(B) \subseteq \mathsf{n}(B')$ and $\Psi \vdash \mathsf{currentGeneration}(i)$

The condition $\mathsf{currentGeneration}(i)$ is used to test if i is the most recent generation. The assertion $\{\langle i, B \mathrel{\dot{\succ}} B'\rangle\}$ states that B' is in-connected to B in generation i if $\mathsf{n}(B) \subseteq \mathsf{n}(B')$, while the assertion $\{\langle i, 0\rangle\}$ states that nothing is connected in generation i.

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

$$T = (\!|\{\langle 1, 0\rangle\}|\!) \mid \tau \,.\, \big((\!|\{\langle 2, K \mathrel{\dot{\succ}} M\rangle, \langle 2, K \mathrel{\dot{\succ}} N\rangle\}|\!) \mid \tau \,.\, ((\!|\{\langle 3, K \mathrel{\dot{\succ}} M\rangle\}|\!))\big)$$

In the process $P \mid T$, P can broadcast on K while T manages the topology. Initially $\mathcal{F}(T) = \{\langle 1, 0\rangle\}$ and the broadcast is disconnected; after $T \xrightarrow{\tau} T'$ then $\mathcal{F}(T') = \{\langle 1, 0\rangle, \langle 2, K \mathrel{\dot{\succ}} M\rangle, \langle 2, K \mathrel{\dot{\succ}} N\rangle\}$ and a broadcast on K can be received on both M and N, and after $T' \xrightarrow{\tau} T''$ then a broadcast can be received only on M, since $\mathcal{F}(T'') = \{\langle 1, 0\rangle, \langle 2, K \mathrel{\dot{\succ}} M\rangle, \langle 2, K \mathrel{\dot{\succ}} N\rangle, \langle 3, K \mathrel{\dot{\succ}} M\rangle\}$.

5 The LUNAR protocol in Psi

In this section we present a model of the LUNAR routing protocol for mobile ad-hoc networks [24,25]. LUNAR is intended for small wireless networks, ca 15 nodes, with a network diameter of 3 hops. It does not handle route reparation, caching etc, and routes must be re-established every few seconds. It is reasonably simple in comparison to many other ad-hoc routing protocols, and allows us to focus on properties such as dynamic connectivity and broadcasting. It has previously been verified in [28,27] using SPIN and UPPAAL; our model is significantly shorter and at an abstraction level closer to the specification.

The LUNAR protocol is at "layer 2.5", between the link and network layers in the Internet protocol stack. Addressing is by pairs of MAC/Ethernet addresses and 64-bit selectors, similarly to the IP address and port number used

in UDP/TCP. The selectors are used to find the appropriate packet handler through the FIB (Forwarding Information Base) table.

Below, we define a psi-calculus for modelling the LUNAR protocol. In an effort to keep our model simple we abstract from details such as TTL fields in messages, optional protocol fields, globally unique host identifiers, etc. We do not deal with time at all.

Channels are of two kinds: broadcast channels are terms node_i with (for simplicity) empty support, whose connectivity is given by the $\dot{\succ}$ and $\dot{\prec}$ predicates as defined in Section 4, and unicast channels which are pairs $\langle sel, mac\rangle$ where *sel* is a selector name and *mac* is a MAC address name. The *mac* part can also be a $\mathsf{RouteOf}(node, ip)$ construction, which looks up the route of an IP address *ip* in the routing table of the node *node*. Special channels $\langle \mathsf{delivered}, \mathsf{node}_i\rangle$ are used to signal delivery of a packet to the IP layer. Assertions record requests originated at the local node using $\mathsf{Redirected}(node, sel)$ and specify found routes using $\mathsf{HaveRoute}(node, destip, hops, sel)$. The conditions contain predicates for testing if a route has been found ($\mathsf{HaveRoute}(node, ip)$), if a selector has been used for a request originating at the local node ($\mathsf{Redirected}(node, sel)$), and to extract the forwarder of a route ($\langle x, \mathsf{RouteOf}(node, ip)\rangle \leftrightarrow \langle x, ip\rangle$).

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

The parameters of the psi-calculus for LUNAR extend the general topology psi-calculus in Section 4 as follows. The sets $\mathbf{T}, \mathbf{C}$ and $\mathbf{A}$ recursively include terms in order to be closed under substitution of terms for names.

$$
\begin{aligned}
\mathbf{T} &\triangleq \mathcal{N} \cup \{\mathsf{node}_i : i \in \mathbb{N}\} \cup \{\mathsf{delivered}\} \cup \\
&\quad \{\mathsf{RREQ}(Ser,\ TargIp, Rep) : Ser,\ TargIp,\ Rep \in \mathbf{T}\} \cup \\
&\quad \{\mathsf{RREP}(i, Fwd) : i,\ Fwd \in \mathbf{T}\} \cup \\
&\quad \{\mathsf{RouteOf}(Node, Ip) : Node,\ Ip \in \mathbf{T}\} \cup \\
&\quad \{\langle Sel, N\rangle : Sel, N \in \mathbf{T}\} \cup \{N + 1 : N \in \mathbf{T}\} \cup \{0\} \\
\mathbf{C} &\triangleq \{M = N, \mathsf{HaveRoute}(M, N), \mathsf{Redirected}(M, N) : M, N \in \mathbf{T}\} \\
\mathbf{A} &\triangleq \mathcal{P}_{\mathrm{fin}}(\{\mathsf{HaveRoute}(M, N_1, i, N_2) : i,\ M, N_1, N_2 \in \mathbf{T}\} \cup \\
&\qquad \{\mathsf{Redirected}(M, N) : M, N \in \mathbf{T}\})
\end{aligned}
$$

$$
\begin{aligned}
\Psi &\vdash a = a,\ a \in \mathcal{N} \\
\Psi &\vdash \langle a, b\rangle \leftrightarrow \langle a, b\rangle,\ a, b \in \mathcal{N} \\
\Psi &\vdash \langle \mathsf{delivered}, \mathsf{node}_i\rangle \leftrightarrow \langle \mathsf{delivered}, \mathsf{node}_i\rangle,\ i \in \mathbb{N} \\
\Psi \cup \{\mathsf{HaveRoute}(\mathsf{node}_i, a, j, b)\} &\vdash \langle \mathsf{RouteOf}(\mathsf{node}_i, a), x\rangle \leftrightarrow \langle b, x\rangle \\
\Psi \cup \{\mathsf{HaveRoute}(\mathsf{node}_i, a, j, b)\} &\vdash \mathsf{HaveRoute}(\mathsf{node}_i, a) \\
\Psi \cup \{\mathsf{Redirected}(\mathsf{node}_i, s)\} &\vdash \mathsf{Redirected}(\mathsf{node}_i, s) \\
\Psi &\vdash \neg\varphi \text{ if } \neg(\Psi \vdash \varphi)
\end{aligned}
$$

Figures 1-7 describe our psi-calculus model of the LUNAR protocol. We use process identifiers to improve the readability of the model. Process identifiers and recursion can be encoded in a standard fashion using replication, see e.g. [22]. In this section we use process declarations of the form $M(\widetilde{N}) \Leftarrow P$, where M is a process identifier (and also a term, implicitly included in $\mathbf{T}$), $\widetilde{N}$ a list of terms where occurrences of names are binding, and P is a process s.t. $\mathsf{n}(P) \subseteq \mathsf{n}(\widetilde{N})$. In a process, we write $M(\widetilde{N})$ for invoking a process declaration $M(\widetilde{K}) \Leftarrow P$ such that $\widetilde{N} = \widetilde{K}[\widetilde{x} := \widetilde{L}]$ with $\widetilde{x} = \mathsf{n}(\widetilde{K})$, resulting in the process $P[\widetilde{x} := \widetilde{L}]$. We write **if** φ **then** P **else** Q for **case** $\varphi : P \,[]\, \neg\varphi : Q$, and assume a suitable encoding of the τ prefix.

Our model of the protocol closely follows the informal protocol description in [25, Section 4]. Each figure in our model corresponds quite directly to one or more of part 0-5 of the protocol description. To allocate a selector, we simply bind a name; to associate (or bind) a selector to a packet handler we use a replicated process which receives on the unicast channel described by the pair of the selector and our MAC address (see e.g. the second line of the LunARP process declaration in Figure 1). In the informal protocol description [25], the FIB is "abused" by installing a null packet handler for the selector created when sending a route request. This FIB entry is only used to detect and avoid circular forwarding of route requests. We model this by an explicit assertion Redirected and a matching condition. The routing table is modelled using assertions, to show how these can be used as a global data structure. For simplicity we do not model route timeouts and the deletion of routes, but this could be done using the mechanism in Section 4.

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

We show the basic correctness of the model by the following theorem, which in essence corresponds to the correct operation of an ad-hoc routing protocol [28, Definition 1]: if there is a path between two nodes, the protocol finds it, and it is possible to send packets along the path to the destination node.

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

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

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

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

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

Fig. 3. Part 3: Intermediate RREP packet handler

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

Fig. 4. Part 4: Source RREP packet handler

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

Fig. 5. Part 5: IP delivery

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

Fig. 6. Broadcast handler

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

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

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

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

Theorem 12. *If Ψ connects node_0 and node_n via a node node_i (i.e. $\Psi \vdash \mathsf{node}_0 \dot{\succ} \mathsf{node}_i$ and $\Psi \vdash \mathsf{node}_i \dot{\succ} \mathsf{node}_n$), then*

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

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

Proof. By following transitions.

Our analysis is limited to a two-hop configuration due to the labour of manually following transitions in a non-trivial specification. We anticipate this can be automated using a future extension of our symbolic semantics for psi-calculi [10,11].

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

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

When the input prefix is over a broadcast channel, as is the case here, the two are not equivalent since a single communication with BrdHandler′ may result in arbitrarily many RreqHandler processes, while BrdHandler only results in one.

6 Related work

Process calculi with broadcast communication go back to the early 1980's. Milner developed SCCS [16] as a generalisation of CCS [15] to include multiway communication, of which broadcast can be seen as a special case. At the same time Austry and Boudol presented MEIJE [2] as a semantic basis for high-level hardware definition languages.

The first process calculus to seriously consider broadcast with an asynchronous parallel composition was CBS [19,20]. Its development is recorded in a series of papers, examining it from many perspectives. The main focus is on employing broadcast as a high level programming paradigm. CBS was later extended to the pi-calculus in the bπ formalism [5]. Here the broadcast communication channels are names that can be scoped and transmitted between agents.

The main point of this work is to establish a separation result in expressiveness: in the pi-calculus, broadcast cannot be uniformly encoded by unicast.

Recent advances in wireless networks have created a renewed interest in the broadcast paradigm. The first process calculus with this in mind was probably CBS$^\sharp$ [17]. This is a development of CBS to include varying interconnection topologies. Input and output is performed on a universal ether and transitions are indexed with topologies which are sets of connectivity graphs; the connectivity graph matters for the input rule (reception is possible from any connected location). Main applications are on cryptography and routing protocols in mobile ad hoc wireless networks. CBS$^\sharp$ has been followed by several similar calculi. In CWS [14,12] the focus is on modelling low level interference. Communication actions have distinct beginnings and endings, and two actions may interfere if one begins before another has ended. The main result is an operational correspondence between a labelled semantics and a reduction semantics. CMAN [8] is a high level formalism extended with data types, just as the applied pi-calculus extends the original pi-calculus. Data can contain constructors and destructors. There are results on properties of weak bisimulation and an analysis of a cryptographic routing protocol. In the ω-calculus [23] emphasis is on expressing connectivity using sets of group names. An extension also includes separate unicast channels, making this formalism the first to accommodate both multicast and unicast. There are results about strong bisimulation and a verification of a mobile ad hoc network leader election protocol through weak bisimulation. RBPT [7] is similar and uses an alternative technique to represent topology changes, leading to smaller state spaces, and is also different in that it can accommodate an asymmetric neighbour relation (to model the fact that A can send to B but not the other way).

$bA\pi$ [9] is an extension of the applied pi-calculus [1] with broadcast, where connectivity information appears explicitly in the process terms and can change non-deterministically during execution. The claimed result of the paper is proving that a weak labelled bisimulation, for which connectivity is irrelevant, coincides with barbed equivalence. However, for the same reasons as in the applied pi-calculus (cf. [3]), labelled bisimilarity is not compositional in $bA\pi$, so the correspondence does not hold. A suggested fix is to remove unicast channel mobility from the calculus. We would finally mention CMN [13]. The claimed result is to compare two different kinds of semantics for a broadcast operation, but it is in error. The labelled transition semantics contains no rule for merging two inputs as in our BrMerge. As a consequence parallel composition fails to be associative. Consider the situation where P does an output and Q and R both do inputs. A broadcast communication involving all three agents can be derived from $(P|Q) \mid R$ but not from $P \mid (Q|R)$, since in the latter agent the component $Q|R$ cannot make an input involving both Q and R.

It is interesting to compare these formalisms and our broadcast psi from a few important perspectives. Firstly, the broadcast channels are explicitly represented in ω, bπ, CWS and CMN; they are mobile (in the sense that they can be transmitted) only in bπ. In ω, only unicast channels can be communicated.

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

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

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

Table 3. Comparison of some process algebras for wireless broadcast.

Finally, broadcast psi is different from the other calculi for wireless broadcast in that there is no stratification of the syntax into processes and networks. There is just the one kind of agent, suitable for expressing both processes operating in nodes and behaviours of entire networks. In contrast, the other calculi has one set of constructs to express processes and another to express networks, sometimes leading to duplication of effort (for example, there can be a parallel composition operator both at the process and network level). Our conclusion is that broadcast psi is conceptually simpler and more efficient for rigorous proofs, and yet more expressive.

7 Conclusion

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

The model of LUNAR is simplified for clarity and to make manual analysis more manageable. The simplifications are similar to those in the SPIN model by Wibling et al. [28], although we do not model timeouts. Their model [27] is ca 250 lines of SPIN code (excluding comments) while ours is approximately 30 lines. Our model could be improved at the cost of added complexity. For example, allowing broadcast channels to have non-empty support would let us hide broadcast actions, routing tables could be made local by including a scoped name per node, and route deletions could be modelled using generational mechanisms similar to Section 4.

We intend to extend the symbolic semantics for psi-calculi [10,11] with broadcast, and implement the semantics in a tool for automatic verification. We also plan to study weak bisimulation for the broadcast semantics. In order to model more aspects of wireless protocols, we would like to add general resource awareness (e.g. energy or time) to psi-calculi.

References

1. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proceedings of POPL '01*, pages 104–115. ACM, 2001.
2. D. Austry and G. Boudol. Algèbre de processus et synchronisation. *Theor. Comput. Sci.*, 30:91–131, 1984.
3. J. Bengtson, M. Johansson, J. Parrow, and B. Victor. Psi-calculi: Mobile processes, nominal data, and logic. In *Proceedings of LICS 2009*, pages 39–48. IEEE, 2009.
4. J. Bengtson, M. Johansson, J. Parrow, and B. Victor. Psi-calculi: A framework for mobile processes with nominal data and logic. *Logical Methods in Computer Science*, 2011. Accepted for publication. This is an extended version of [3].
5. C. Ene and T. Muntean. Expressiveness of point-to-point versus broadcast communications. In G. Ciobanu and G. Paun, editors, *FCT*, volume 1684 of *LNCS*, pages 258–268. Springer, 1999.
6. M. Gabbay and A. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 13:341–363, 2001.
7. F. Ghassemi, W. Fokkink, and A. Movaghar. Restricted broadcast process theory. In A. Cerone and S. Gruner, editors, *SEFM*, pages 345–354. IEEE Computer Society, 2008.
8. J. C. Godskesen. A calculus for mobile ad hoc networks. In A. L. Murphy and J. Vitek, editors, *COORDINATION*, volume 4467 of *LNCS*, pages 132–150. Springer, 2007.

9. J. C. Godskesen. Observables for mobile and wireless broadcasting systems. In *Proc. of COORDINATION 2010*, volume 6116 of *LNCS*, pages 1–15. Springer, 2010.
10. M. Johansson, B. Victor, and J. Parrow. A fully abstract symbolic semantics for psi-calculi. In *Proceedings of SOS 2009*, volume 18 of *EPTCS*, pages 17–31, 2010.
11. M. Johansson, B. Victor, and J. Parrow. Computing strong and weak bisimulations for psi-calculi. Submitted for publication, 2011.
12. I. Lanese and D. Sangiorgi. An operational semantics for a calculus for wireless systems. *Theor. Comp. Sci.*, 411(19):1928–1948, 2010.
13. M. Merro. An observational theory for mobile ad hoc networks (full version). *Inf. Comput.*, 207(2):194–208, 2009.
14. N. Mezzetti and D. Sangiorgi. Towards a calculus for wireless systems. *Electr. Notes Theor. Comput. Sci.*, 158:331–353, 2006.
15. R. Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980.
16. R. Milner. Calculi for synchrony and asynchrony. *Theor. Comput. Sci.*, 25:267–310, 1983.
17. S. Nanz and C. Hankin. A framework for security analysis of mobile wireless networks. *Theor. Comp. Sci.*, 367(1-2):203–227, 2006.
18. A. M. Pitts. Nominal logic, a first order theory of names and binding. *Information and Computation*, 186:165–193, 2003.
19. K. V. S. Prasad. A calculus of broadcasting systems. In S. Abramsky and T. S. E. Maibaum, editors, *TAPSOFT, Vol.1*, volume 493 of *LNCS*, pages 338–358. Springer, 1991.
20. K. V. S. Prasad. A calculus of broadcasting systems. *Sci. Comput. Program.*, 25(2-3):285–327, 1995.
21. P. Raabjerg and J. Åman Pohjola. Broadcast psi-calculus formalisation. `http://www.it.uu.se/research/group/mobility/theorem/broadcastpsi`, July 2011. Isabelle/HOL-Nominal formalisation of the definitions, theorems and proofs.
22. D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.
23. A. Singh, C. R. Ramakrishnan, and S. A. Smolka. A process calculus for mobile ad hoc networks. *Sci. Comput. Program.*, 75(6):440–469, 2010.
24. C. Tschudin, R. Gold, O. Rensfelt, and O. Wibling. LUNAR: a lightweight underlay network ad-hoc routing protocol and implementation. In *Proc of NEW2AN'04*, St. Petersburg, Feb. 2004.
25. C. F. Tschudin. Lightweight underlay network ad hoc routing (LUNAR) protocol. Internet Draft, Mobile Ad Hoc Networking Working Group, Mar. 2004.
26. C. Urban and C. Tasson. Nominal techniques in Isabelle/HOL. In R. Nieuwenhuis, editor, *Proceedings of CADE 2005*, volume 3632 of *LNCS*, pages 38–53. Springer, 2005.
27. O. Wibling. SPIN and UPPAAL ad hoc routing protocol models. `http://www.it.uu.se/research/group/mobility/adhoc/gbt/other_examples`, 2004. Models of LUNAR scenarios used in [28].
28. O. Wibling, J. Parrow, and A. Pears. Automatized verification of ad hoc routing protocols. In D. de Frutos-Escrig and M. Núñez, editors, *FORTE 2004*, volume 3235 of *LNCS*, pages 343–358. Springer, 2004.

Chapter 3

Higher-order Psi Calculi

Paper II

Higher-order psi-calculi

Joachim Parrow Johannes Borgström Palle Raabjerg
Johannes Åman Pohjola

June 14, 2012

Abstract

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

Robin Milner in memoriam

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

1 Introduction

Psi-calculi is a parametric framework for extensions of the pi-calculus to accommodate applications with complex data structures and high-level logics in a single general and parametric framework with machine-checked proofs. In earlier papers [BJPV09, BP09, JVP10, BJPV10] we have shown how psi-calculi can capture a range of phenomena such as cryptography and concurrent constraints, investigated strong and weak bisimulation, and provided a symbolic semantics. We claim that the theoretical development is more robust than in other calculi of comparable complexity, since we use a single inductive definition in the semantics and since we have checked most results in the theorem prover Isabelle/Nominal [Urb08].

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

1.1 Psi-calculi

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

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

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

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

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

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

agent that asserts Ψ to its environment. For example, in

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

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

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

1.2 Higher-order psi-calculi

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

1.3 Exposition

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

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

2 Psi-calculi

This section recapitulates the relevant parts of [BJPV09, BJPV11].

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

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

A *nominal datatype* is a nominal set together with a set of equivariant functions on it. In particular we shall consider substitution functions that substitutes elements for names. If X is an element of a datatype, the *substitution* $X[\tilde{a} := \tilde{Y}]$ is an element of the same datatype as X. Substitution is required to satisfy a kind of alpha-conversion law: if $\tilde{b}\#X, \tilde{a}$ then $X[\tilde{a} := \tilde{T}] = ((\tilde{b}\ \tilde{a}) \cdot X)[\tilde{b} := \tilde{T}]$; here it is implicit that $\tilde{a}$ and $\tilde{b}$ have the same length, and $(\tilde{a}\ \tilde{b})$ swaps each element of $\tilde{a}$ with the corresponding element of $\tilde{b}$. The name preservation law

$\tilde{a} \subseteq \mathrm{n}(N) \wedge b \in \mathrm{n}(\tilde{M}) \implies b \in \mathrm{n}(N[\tilde{a} := \tilde{M}])$ will be important for some substitutions. Apart from these laws we do not require any particular behaviour of substitution.

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

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

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

and the four equivariant operators:

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

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

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

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

The requisites on valid psi-calculus parameters are:

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

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

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

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

We use F, G to range over frames. Since we identify alpha variants we can choose the bound names arbitrarily. Notational conventions: We write just Ψ for $(\nu\epsilon)\Psi$ when there is no risk of confusing a frame with an assertion, and $\otimes$ to mean composition on frames defined by $(\nu\widetilde{b_1})\Psi_1 \otimes (\nu\widetilde{b_2})\Psi_2 = (\nu\widetilde{b_1}\widetilde{b_2})\Psi_1 \otimes \Psi_2$ where $\widetilde{b_1} \# \widetilde{b_2}, \Psi_2$ and vice versa. We write $(\nu c)((\nu\widetilde{b})\Psi)$ to mean $(\nu c\widetilde{b})\Psi$.

Definition 5. *We define $F \vdash \varphi$ to mean that there exists an alpha variant $(\nu\widetilde{b})\Psi$ of F such that $\widetilde{b}\#\varphi$ and $\Psi \vdash \varphi$. We also define $F \simeq G$ to mean that for all φ it holds that $F \vdash \varphi$ iff $G \vdash \varphi$.*

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

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

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

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

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

Input subjects are underlined to facilitate parsing of complicated expressions; in simple cases we often omit the underline. In the traditional pi-calculus terms are just names and its input construct $a(x)\,.\,P$ can be represented as $\underline{a}(\lambda x)x.P$.

In some of the examples to follow we shall use the simpler notation $a(x)\,.\,P$ for this input form, and sometimes we omit a trailing $\mathbf{0}$, writing just $\overline{M}N$ for $\overline{M}N\,.\,\mathbf{0}$.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Table 1: Operational semantics. Symmetric versions of Com and Par are elided. In the rule Com we assume that $\mathcal{F}(P) = (\nu\widetilde{b}_P)\Psi_P$ and $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$ where $\widetilde{b}_P$ is fresh for all of $\Psi, \widetilde{b}_Q, Q, M$ and P, and that $\widetilde{b}_Q$ is correspondingly fresh. In the rule Par we assume that $\mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q$ where $\widetilde{b}_Q$ is fresh for Ψ, P and α. In Open the expression $\tilde{a} \cup \{b\}$ means the sequence $\tilde{a}$ with b inserted anywhere.

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

3 Higher-order Psi-calculi

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

3.1 Basic definitions

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

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

and the entailment relation is extended to $\vdash \subseteq \mathbf{A} \times (\mathbf{C} \uplus \mathbf{Cl})$, where we write $\Psi \vdash \varphi$ for $\Psi \vdash (0, \varphi)$ and $\Psi \vdash M \Leftarrow P$ for $\Psi \vdash (1, M \Leftarrow P)$. We amend the definition of assertion equivalence to mean that the assertions entail the same conditions *and clauses*. This extension is not formally necessary since we could instead adjoin **Cl** to the conditions, but calling $M \Leftarrow P$ a "condition" is a misnomer we want to avoid.

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

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

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

Finally there is the new transition rule:

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

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

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

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

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

This corresponds to sending the program code. A recipient can both execute it and use it as data. For example R can be **if** $x = P'$ **then** ..., checking if the

received P is syntactically the same as some other agent P'. To prevent the latter, instead send a handle M to represent P:

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

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

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

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

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

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

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

A derivation of this transition uses the Invocation rule

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

Through Par and Scope we get

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

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

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

and finally through Par and Scope again we get the desired transition.

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

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

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

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

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

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

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

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

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

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

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

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

3.2 Process abstractions and parameters

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

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

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

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

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

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

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

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

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

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

and thus

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.3 Canonical higher-order instances

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

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

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

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

by the parameters $\mathbf{T}_{\mathcal{H}}, \mathbf{C}_{\mathcal{H}}, \mathbf{A}_{\mathcal{H}}, \leftrightarrow_{\mathcal{H}}, \otimes_{\mathcal{H}}, \mathbf{1}_{\mathcal{H}}, \vdash_{\mathcal{H}}$ *where*

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

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

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

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

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

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

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

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

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

4 Algebraic theory

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

4.1 Bisimulation

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

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

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

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

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

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

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

Theorem 17. *For all* Ψ*:*

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

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

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

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

The usual structural laws hold for strong congruence:

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

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

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

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

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

4.2 Encoding operators

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

As an example of the encoding of Replication, consider a transition from $(\nu a)(\mathbf{run}\ M \mid (\!|\Psi^{M \Leftarrow P \mid \mathbf{run}\ M}|\!))$. It can only be by invocation where $P \mid \mathbf{run}\ M$ has a transition leading to $P' \mid \mathbf{run}\ M$ and results in

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

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

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

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

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

4.3 Higher-order bisimulation

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

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

In a higher-order process calculus we can instead transmit the possibility to invoke a process, as in $(\nu b)\overline{a}M_b \,.\, (\!|\{M_b \Leftarrow P\}|\!)$. A recipient of M_b has no other

use for this handle than to invoke P. Therefore, if $P \mathrel{\dot{\sim}} P'$ it is reasonable to expect the two processes

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

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

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

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

1. *Static equivalence:*

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

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

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

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

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

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

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

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

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

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

The proof boils down to showing that

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

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

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

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

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

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

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

Theorem 31. *For all* Ψ*:*

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

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

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

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

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

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

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

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

4.4 Informal proofs

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

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

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

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

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

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

The up-to techniques of Definition 35 are sound.

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

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

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

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

$$\begin{aligned} T_{\text{a}}(\mathcal{R}) := \{&(B, \Psi, P, Q) : B\#P, Q \wedge \Psi \triangleright P \; \mathcal{R} \; Q\} \cup \\ \{&(B, \Psi, P, R) : (B, \Psi, P, Q) \in T_{\text{a}}(\mathcal{R}) \wedge (B, \Psi, Q, R) \in T_{\text{a}}(\mathcal{R}))\} \end{aligned}$$

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

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

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

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

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

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

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

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

1. *if $\Psi \simeq \Psi' \otimes \Psi_R$, $\mathcal{F}(R) = (\nu \widetilde{b_R})\Psi_R$, $\widetilde{b_R} \subseteq B$, $B\#R$ and $\widetilde{b_R}\#\Psi', R$ then $\Psi' \triangleright_{B \setminus \widetilde{b_R}} (P \mid R) \; \overline{\mathcal{S}} \; (Q \mid R)$; and*
2. *if $a\#\Psi$ then $\Psi \triangleright_B (\nu a)P \; \overline{\mathcal{S}} \; (\nu a)Q$; and*
3. *if $\Psi \triangleright P' \dot{\sim} P$ and $\Psi \triangleright Q \dot{\sim} Q'$ and $B\#P', Q'$ then $\Psi \triangleright_B P' \; \overline{\mathcal{S}} \; Q'$.*
4. *$\Psi \otimes \Psi' \triangleright_B P \; \overline{\mathcal{S}} \; Q$, and $\Psi \triangleright_{B \setminus B'} P \; \overline{\mathcal{S}} \; Q$, and if $\Psi \simeq \Psi_2$ then $\Psi_2 \triangleright_B P \; \overline{\mathcal{S}} \; Q$.*

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

We recall three lemmas used in the compositionality proof for first-order bisimilarity [BJPV11]. These lemmas have been proven to hold also for higher-order psi-calculi. The first lemma states that when performing a non-tau transition, the frame of the process grows such that the bound names in the frame and the action can be chosen fresh for an arbitrary set of names B.

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

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

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

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

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

$$\begin{array}{rl} & B \subset_{\text{fin}} \mathcal{N} \wedge B \# P \wedge \mathcal{F}(P) = (\nu \widetilde{b}_P)\Psi_P \\ \wedge & \Psi \rhd P \xrightarrow{\alpha} P' \text{ where } \alpha \neq \tau \\ \wedge & \widetilde{b}_P \# \Psi, P, \text{subj}(\alpha), B \\ \Longrightarrow & \exists K.\ B \# K \wedge \Psi \otimes \Psi_P \vdash K \leftrightarrow \text{subj}(\alpha) \end{array}$$

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

Lemma 42 (Subject rewriting)**.**

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

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

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

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

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

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

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

$$
\textsc{Invocation}\ \frac{\Psi \otimes \Psi_P \vdash M \Leftarrow R_1 \qquad \Psi \otimes \Psi_P \rhd R_1 \xrightarrow{\alpha} R_P}{\Psi \otimes \Psi_P \rhd \mathbf{run}\ M \xrightarrow{\alpha} R_P}
$$

By induction, there is R' such that $\Psi \otimes \Psi_Q \rhd R_1 \xrightarrow{\alpha} R'$ and $\Psi \rhd_B (P|R_P) \mathrel{\overline{\mathcal{S}}} (Q|R')$.

Since $\Psi \otimes \mathbf{1} \rhd P \mathrel{\dot{\sim}^{\text{HO}}} Q$ there is R_2 such that $\Psi \otimes \Psi_Q \vdash M \Leftarrow R_2$ and $\mathbf{1} \rhd R_1 \mathrel{\dot{\sim}^{\text{HO}}} R_2$. Then $\Psi \otimes \Psi_Q \rhd R_1 \mathrel{\dot{\sim}^{\text{HO}}} R_2$, so there is R_Q such that $\Psi \otimes \Psi_Q \rhd R_2 \xrightarrow{\alpha} R_Q$ and $\Psi \otimes \Psi_Q \rhd R' \mathrel{\dot{\sim}^{\text{HO}}} R_Q$. By the definition of $\mathcal{S}$, we then get $\Psi \rhd (Q|R') \mathrel{\mathcal{S}} (Q|R_Q)$.

Since $B \# R, \text{obj}(\alpha)$ we get $B \# R_Q$, so $\Psi \rhd_B (Q|R') \mathrel{\overline{\mathcal{S}}} (Q|R_Q)$. By transitivity $\Psi \rhd_B (P|R_P) \mathrel{\overline{\mathcal{S}}} (Q|R_Q)$.

Scope In this case we have that $\mathcal{F}((\nu b)R) = (\nu b)\mathcal{F}(R)$ where $\mathcal{F}(R) = (\nu \widetilde{b}_R)\Psi_R$, so $\mathcal{F}((\nu b)R) = (\nu b \widetilde{b}_R)\Psi_R$. We assume that $b \# \widetilde{b}_R$; since $b \widetilde{b}_R \# (\nu b)R$ we then have $\widetilde{b}_R \# R$. The transition is derived like

$$
\textsc{Scope}\ \frac{\Psi \otimes \Psi_P \rhd R \xrightarrow{\alpha} R_P}{\Psi \otimes \Psi_P \rhd (\nu b)R \xrightarrow{\alpha} (\nu b)R_P}\ b \# \alpha, \Psi \otimes \Psi_P
$$

By induction we get that there exists R_Q such that $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\alpha} R_Q$ and $\Psi \rhd_{B \cup \{b\}} (P|R_P) \mathrel{\overline{\mathcal{S}}} (Q|R_Q)$. Using Scope, $\Psi \otimes \Psi_Q \rhd (\nu b)R \xrightarrow{\alpha} (\nu b)R_Q$. Using Lemma 39 we get $\Psi \rhd_B (P|(\nu b)R_P) \mathrel{\overline{\mathcal{S}}} (Q|(\nu b)R_Q)$.

Par Here $\mathcal{F}(R_1 \mid R_2) = (\nu \widetilde{b}_{R_1} \widetilde{b}_{R_2})\Psi_{R_1} \otimes \Psi_{R_2}$ with $\widetilde{b}_{R_1} \# \widetilde{b}_{R_2}, \Psi_{R_2}$ and $\widetilde{b}_{R_2} \# \widetilde{b}_{R_1}, \Psi_{R_1}$. The transition is derived like

$$\textsc{Par}\ \frac{\Psi_{R_2} \otimes \Psi \otimes \Psi_P \rhd R_1 \xrightarrow{\alpha} R_P}{\Psi \otimes \Psi_P \rhd R_1 \mid R_2 \xrightarrow{\alpha} R_P \mid R_2}\ \mathrm{bn}(\alpha) \# R_2$$

We know that $\widetilde{b}_P \# \Psi, R_1 \mid R_2$ and that $\widetilde{b}_{R_1} \widetilde{b}_{R_2} \# R_1 \mid R_2, \widetilde{b}_P$. This gives us that also $\widetilde{b}_P \# \Psi_{R_2} \otimes \Psi, R_1$ and that $\widetilde{b}_{R_1} \# \Psi_{R_2} \otimes \Psi, R_1$.

By induction we get R_Q such that $\Psi_{R_2} \otimes \Psi \otimes \Psi_Q \rhd R_1 \xrightarrow{\alpha} R_Q$ and $\Psi_{R_2} \otimes \Psi \rhd_{B \cup \widetilde{b}_{R_2}} (P|R_P)\ \overline{\mathcal{S}}\ (Q|R_Q)$. We then derive

$$\textsc{Par}\ \frac{\Psi_{R_2} \otimes \Psi \otimes \Psi_Q \rhd R_1 \xrightarrow{\alpha} R_Q}{\Psi \otimes \Psi_Q \rhd R_1 \mid R_2 \xrightarrow{\alpha} R_Q \mid R_2}\ \mathrm{bn}(\alpha) \# R_2$$

Using Lemma 39 we get that $\Psi \rhd_B (P|R_P|R_2)\ \overline{\mathcal{S}}\ (Q|R_Q|R_2)$.

Com Here $\mathcal{F}(R_1 \mid R_2) = (\nu \widetilde{b}_{R_1} \widetilde{b}_{R_2})\Psi_{R_1} \otimes \Psi_{R_2}$ with $\widetilde{b}_{R_1} \# \widetilde{b}_{R_2}, \Psi_{R_2}$ and vice versa. The transition is derived like

$$\textsc{Com}\ \frac{\begin{array}{c}\Psi_{R_2} \otimes \Psi \otimes \Psi_P \rhd R_1 \xrightarrow{\overline{M}\,(\nu \widetilde{a}) N} R_{P1} \\ \Psi_{R_1} \otimes \Psi \otimes \Psi_P \rhd R_2 \xrightarrow{K\,N} R_{P2} \qquad \Psi \otimes \Psi_P \otimes \Psi_{R_1} \otimes \Psi_{R_2} \vdash M \leftrightarrow K\end{array}}{\Psi \otimes \Psi_P \rhd R_1 \mid R_2 \xrightarrow{\tau} (\nu \widetilde{a})(R_{P1} \mid R_{P2})}$$

We assume that $\widetilde{b}_P \# \widetilde{a}$ (otherwise α-convert $\widetilde{a}$ as necessary). Since $\widetilde{b}_P \# R_1 \mid R_2$ we get $\widetilde{b}_P \# N$. However, we cannot use the induction hypothesis directly since we do not know that $\widetilde{b}_P \# M$ and $\widetilde{b}_P \# K$, respectively.

Let $B_1 = \widetilde{b}_P \cup \widetilde{b}_{R_2}$. We have that $\widetilde{b}_{R_1} \# \Psi_{R_2}, \Psi, \Psi_P, R_1, M, B'$. By Lemma 41 we get that there exists M' such that $B_1 \# M'$ and $\Psi \otimes \Psi_P \otimes \Psi_{R_1} \otimes \Psi_{R_2} \vdash M \leftrightarrow M'$. Similarly, by applying Lemma 41 to the transition of R_2 we get K' such that $\widetilde{b}_P \widetilde{b}_{R_1} \# K'$ and $\Psi \otimes \Psi_P \otimes \Psi_{R_1} \otimes \Psi_{R_2} \vdash K \leftrightarrow K'$.

By symmetry and transitivity of $\leftrightarrow$ we then get that $\Psi \otimes \Psi_P \otimes \Psi_{R_1} \otimes \Psi_{R_2} \vdash M' \leftrightarrow K'$. By Lemma 42 we get that $\Psi_{R_2} \otimes \Psi \otimes \Psi_P \rhd R_1 \xrightarrow{\overline{K'}\,(\nu \widetilde{a}) N} R_{P1}$ and that $\Psi_{R_1} \otimes \Psi \otimes \Psi_P \rhd R_2 \xrightarrow{M'\,N} R_{P2}$. By induction we learn that $\Psi_{R_2} \otimes \Psi \otimes \Psi_Q \rhd R_1 \xrightarrow{\overline{K'}\,(\nu \widetilde{a}) N} R_{Q1}$ and $\Psi_{R_1} \otimes \Psi \otimes \Psi_Q \rhd R_2 \xrightarrow{M'\,N} R_{Q2}$ such that $\Psi_{R_2} \otimes \Psi \rhd_{B \cup \widetilde{b}_{R_2}} (P|R_{P1})\ \overline{\mathcal{S}}\ (Q|R_{Q1})$ and $\Psi_{R_1} \otimes \Psi \rhd_{B \cup \widetilde{b}_{R_1}} (P|R_{P2})\ \overline{\mathcal{S}}\ (Q|R_{Q2})$.

Since $\widetilde{b}_P \# K', M'$ we get that $\Psi \otimes \mathcal{F}(P) \otimes \Psi_{R_1} \otimes \Psi_{R_2} \vdash M' \leftrightarrow K'$. From $\Psi \otimes \Psi_{R_1} \otimes \Psi_{R_2} \rhd P \dot{\sim}^{\text{HO}} Q$ we then get that $\Psi \otimes \mathcal{F}(Q) \otimes \Psi_{R_1} \otimes \Psi_{R_2} \vdash M' \leftrightarrow K'$. We finally get that $\Psi \otimes \Psi_Q \otimes \Psi_{R_1} \otimes \Psi_{R_2} \vdash M' \leftrightarrow K'$, permitting the

following derivation:

$$\textsc{Com}\ \frac{\begin{array}{c}\Psi_{R_2}\otimes\Psi\otimes\Psi_Q \rhd R_1 \xrightarrow{\overline{K'}(\nu\widetilde{a})N} R_{Q1} \\ \Psi_{R_1}\otimes\Psi\otimes\Psi_Q \rhd R_2 \xrightarrow{M'\,N} R_{Q2} \qquad \Psi\otimes\Psi_Q\otimes\Psi_{R_1}\otimes\Psi_{R_2} \vdash M' \leftrightarrow K'\end{array}}{\Psi\otimes\Psi_Q \rhd R_1 \mid R_2 \xrightarrow{\tau} (\nu\widetilde{a})(R_{Q1} \mid R_{Q2})}$$

Assume that $\mathcal{F}(R_{P2}) = (\nu\widetilde{b}_{R_{P2}})\Psi_{R_{P2}}$ and $\mathcal{F}(R_{Q1}) = (\nu\widetilde{b}_{R_{Q1}})\Psi_{R_{Q1}}$ with $\widetilde{b}_{R_{Q1}}\#\widetilde{b}_{R_{P2}}$ and $\widetilde{b}_{R_{Q1}}\widetilde{b}_{R_{P2}}\#\Psi, P, Q, R, N$. Since $\Psi_{R_{P2}} \simeq \Psi_{R_2} \otimes \Psi_2$ for some Ψ_2 we have $\Psi_{R_{P2}} \otimes \Psi \rhd_{B\cup\widetilde{b}_{R_2}} (P|R_{P1})\ \overline{\mathcal{S}}\ (Q|R_{Q1})$ by Lemma 39.4. Then $\Psi \rhd_B (P|R_{P1} \mid R_{P2})\ \overline{\mathcal{S}}\ (Q|R_{Q1} \mid R_{P2})$ by Lemma 39.1. Similarly, $\Psi \rhd_B (P|R_{Q1} \mid R_{P2})\ \overline{\mathcal{S}}\ (Q|R_{Q1} \mid R_{Q2})$.

By symmetry of $\dot{\sim}$, we have $\Psi \otimes \Psi_R \rhd Q \dot{\sim}^{\text{HO}} P$, and by extension of arbitrary assertion we get $\Psi \otimes \Psi_{R_{Q1}} \otimes \Psi_{R_{P2}} \rhd Q \dot{\sim}^{\text{HO}} P$. By the definition of $\mathcal{S}$ we get $\Psi \rhd (Q \mid R_{Q1} \mid R_{P2})\ \mathcal{S}\ (P \mid R_{Q1} \mid R_{P2})$. Since $B\#R_1, N$ we then have $\Psi \rhd_B (Q \mid R_{Q1} \mid R_{P2})\ \overline{\mathcal{S}}\ (P \mid R_{Q1} \mid R_{P2})$. By transitivity of $\overline{\mathcal{S}}$ we then get $\Psi \rhd_B (P \mid R_{P1} \mid R_{P2})\ \overline{\mathcal{S}}\ (Q \mid R_{Q1} \mid R_{Q2})$. Using Lemma 39 we finally have $\Psi \rhd_B P \mid (\nu\widetilde{a})(R_{P1} \mid R_{P2})\ \overline{\mathcal{S}}\ Q \mid (\nu\widetilde{a})(R_{Q1} \mid R_{Q2})$. □

A variant of Lemma 43 treats the case where R makes a transition in the environment of P that can communicate with a transition of P in environment of R. The processes R and Q can then perform matching transitions, leading to $\overline{\mathcal{S}}$-related derivatives.

Lemma 44 (subject switching lemma)**.**

$$\begin{array}{rl} & \Psi \otimes \Psi_R \rhd P \dot{\sim}^{\text{HO}} Q \\ \wedge & \mathcal{F}(P) = (\nu\widetilde{b}_P)\Psi_P \\ \wedge & \mathcal{F}(Q) = (\nu\widetilde{b}_Q)\Psi_Q \\ \wedge & \mathcal{F}(R) = (\nu\widetilde{b}_R)\Psi_R \\ \wedge & \Psi \otimes \Psi_R \rhd P \xrightarrow{\overline{M}(\nu\widetilde{a})N} P' \\ \wedge & \Psi \otimes \Psi_P \rhd R \xrightarrow{M\,N} R_P \\ \wedge & \Psi \otimes \Psi_P \otimes \Psi_R \vdash K \leftrightarrow M \\ \wedge & \widetilde{b}_P\#R, M, N, \Psi, P, Q \\ \wedge & \widetilde{b}_Q\#R, M, N, \Psi, P, Q \\ \wedge & \widetilde{b}_R\#K, N, \Psi, P, \widetilde{b}_P, \Psi_P, Q, \widetilde{b}_Q, \Psi_Q, R \\ \wedge & \widetilde{a}\#M, \Psi, P, Q, R, \widetilde{b}_P, \widetilde{b}_Q \\ \wedge & B\#P, Q, R, N, \widetilde{a}, \widetilde{b}_P, \widetilde{b}_Q, P' \\ \Longrightarrow & \exists M', R_Q, Q'. \\ & \widetilde{b}_R, B\#M' \\ \wedge & \Psi \otimes \Psi_Q \otimes \Psi_R \vdash K \leftrightarrow M' \\ \wedge & \Psi \otimes \Psi_Q \rhd R \xrightarrow{M'\,N} R_Q \\ \wedge & \Psi \otimes \Psi_R \rhd Q \xrightarrow{\overline{M}(\nu\widetilde{a})N} Q' \\ \wedge & \Psi \otimes \Psi_P \rhd_B (P' \mid R_P)\ \overline{\mathcal{S}}\ (Q' \mid R_Q) \end{array}$$

Proof. By induction on the transition of R, similar to Lemma 43. □

The statement of Lemma 44 also needs to hold for output transitions of R, mutatis mutandis. We can then show the desired result.

Theorem 45. *$\mathcal{S}$ is a HO-bisimulation up to $T \circ U \circ R$.*

Proof sketch. Assume that $\Psi \rhd P|R \; \mathcal{S} \; Q|R$, i.e. that $\Psi \otimes \mathcal{F}(R) \rhd P \mathrel{\dot{\sim}^{\text{HO}}} Q$. Symmetry, extension with arbitrary assertion, and static equivalence of conditions follow from the same properties of $\dot{\sim}^{\text{HO}}$. Static equivalence of clauses up to $T \circ U \circ R$ follows from the static equivalence of clauses of $\Psi \otimes \mathcal{F}(R) \rhd P \mathrel{\dot{\sim}^{\text{HO}}} Q$.

We prove simulation up to $T \circ U \circ R$ by case analysis on the derivation of the transition of $P \mid R$. Recall that $\Psi \rhd P' \; (T(\mathcal{R})) \; Q'$ iff $\Psi \rhd_\emptyset P' \; (T_{\text{a}}(\mathcal{R})) \; Q'$.

Par-L By bisimilarity of P and Q.

Par-R Here $\Psi \otimes \Psi_P \rhd R \xrightarrow{\alpha} R_P$. By Lemma 43 $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\alpha} R_Q$ with $\Psi \rhd_\emptyset P \mid R_P \; \overline{\mathcal{S}} \; Q \mid R_Q$.

Com-L Here $\Psi \otimes \Psi_R \rhd P \xrightarrow{\overline{K}\,(\nu \widetilde{a})N} P'$, $\Psi \otimes \Psi_P \rhd R \xrightarrow{\underline{M}\,N} R_P$, $\Psi \otimes \Psi_P \otimes \Psi_R \vdash K \leftrightarrow M$ and $\widetilde{b}_P \# K$ and $\widetilde{b}_R \# M$. We may assume that $\widetilde{a} \# \Psi_R$.

By bisimilarity $\Psi \otimes \Psi_R \rhd Q \xrightarrow{\overline{K}\,(\nu \widetilde{a})N} Q'$ with $\Psi \otimes \Psi_R \rhd P' \mathrel{\dot{\sim}^{\text{HO}}} Q'$. By Lemma 44 there are M', R_Q with $\widetilde{b}_R \# M'$ such that $\Psi \otimes \Psi_Q \otimes \Psi_R \vdash K \leftrightarrow M'$ and $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\underline{M'}\,N} R_Q$ and $\Psi \rhd_\emptyset (P' \mid R_P) \; \overline{\mathcal{S}} \; (Q' \mid R_Q)$.

Finally, by Lemma 39 we get $\Psi \rhd_\emptyset (\nu \widetilde{a})(P' \mid R_P) \; \overline{\mathcal{S}} \; (\nu \widetilde{a})(Q' \mid R_Q)$.

Com-R As Com-L. □

It now follows that Theorem 31(1) holds.

Corollary 46. $P \mathrel{\dot{\sim}^{\text{HO}}_\Psi} Q \Longrightarrow P \mid R \mathrel{\dot{\sim}^{\text{HO}}_\Psi} Q \mid R$.

Proof. Assume that $\mathcal{F}(R) = (\nu \widetilde{b}_R)\Psi_R$ with $\widetilde{b}_R \# \Psi, P, Q$. By extension of arbitrary assertion we get $P \mathrel{\dot{\sim}^{\text{HO}}_{\Psi \otimes \Psi_P}} Q$, so $\Psi \rhd (P|R) \; \mathcal{S} \; (Q|R)$ by the definition of $\mathcal{S}$. By Theorem 45 and Theorem 37 we get $\mathcal{S} \subseteq \dot{\sim}^{\text{HO}}$, so $P \mid R \mathrel{\dot{\sim}^{\text{HO}}_\Psi} Q \mid R$. □

4.5 Formal proofs

All theorems in this paper have been machine-checked with the interactive theorem prover Isabelle. The proof scripts [ÅPR] are adapted and extended from Bengtson's formalisation of psi-calculi [Ben10]. They constitute 63334 lines of Isabelle code; Bengtson's code is 37417 lines. The bulk of the new code pertains to Theorems 26 and 31, which have quite involved proofs that depart significantly from Bengtson's. It is interesting to observe how wildly the effort involved in conducting the proofs varies. We briefly recount our experiences here.

With only minor modifications to Bengtson's proofs, we were able to re-prove all of the meta-theoretical results for psi-calculi (Theorems 17, 19, 20 and 21)

in a matter of days. We believe that situations like these, where results need to be reestablished under slightly different definitions, are among those where theorem provers truly shine.

By contrast, HO-bisimulation (Theorems 29, 31 and 28) is an example of a small change to the definitions which gave rise to man-months of work rather than days. This is because certain technical lemmas on which the old proofs depend are no longer valid in the context of HO-bisimulation. Hence, completely new proofs and proof ideas had to be developed. However, with HO-bisimulation in place, HO-congruence (Theorem 32) was mechanised in a matter of minutes.

The proofs pertaining to canonical instances and the encoding of operators (Theorems 15, 24, 25 and 26) also represent man-months of work, but for different reasons. Here simple and intuitive proof ideas turned out to be cumbersome to mechanise. In the case of Theorem 15, the encoding of canonical instances is complicated and unintuitive, because of the necessity to sidestep certain technical restrictions in the framework; for an example, nominal datatype definitions cannot depend on locale parameters. Theorem 26 gives rise to almost 9000 lines of proof script, even though the proof is conceptually simple. The main problem is the unwieldy candidate relation used for the proof, which includes many assumptions about the underlying psi-calculus. Moreover, it is closed under parallel composition and restriction, which significantly increases the size of the transition derivation trees we must follow and the amount of manual alpha-conversion we must perform, respectively. We believe that a much shorter proof can be obtained if a bisimulation up-to context technique is used instead, but we do not currently have a proof that such a technique is sound.

4.6 Comparing higher-order equivalences

Our definition of HO-bisimilarity is technically nontrivial and we here motivate it. Our primary concern is to not depart too much from the original bisimilarity since we have invested a substantial effort in an Isabelle proof repository and strive to re-use as much as possible. Therefore our approach is to amend the original definition as little as possible in order to validate Theorem 28. Even so, there are a number of alternatives in the precise formulation of Clause 1(b). The current definition requires in the conclusion that $\mathcal{R}(\mathbf{1}, P', Q')$, i.e. that P' and Q' are again bisimilar in the assertion $\mathbf{1}$, which by Clause 3 is the same as requiring $\forall\Psi.\mathcal{R}(\Psi, P', Q')$. As a consequence, the following strengthening of Theorem 28 (note the assertions Ψ) is not true in general:

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

We have failed to define a version of higher-order bisimilarity where this holds. An obvious attempt is to adjust Clause 1(b) to use $(\Psi, P', Q') \in \mathcal{R}$, i.e. with Ψ in place of $\mathbf{1}$, but with this we fail to prove Theorem 31.1, i.e. that bisimilarity is preserved by parallel composition. The reason is that our proof strategy using the relation $\mathcal{S}$ in Section 4.4 relies on the fact that

$$\Psi \otimes \Psi' \rhd P \mathrel{\dot{\sim}} Q \quad \Rightarrow \quad \Psi \rhd (\!|\Psi'|\!) \mid P \mathrel{\dot{\sim}} (\!|\Psi'|\!) \mid Q$$

This holds for ordinary bisimulation and for higher-order bisimulation, but fails if Clause 1(b) uses $(\Psi, P', Q') \in \mathcal{R}$. The counterexample is somewhat artificial and it remains to be seen if we can formulate a subset of higher-order calculi where this property holds, or if there is a different proof strategy for Theorem 31.1 that does not require the property, involving another candidate bisimulation relation.

Another possibility would be to include even more information in the assertion, as in $(\Psi \otimes \mathcal{F}(P), P', Q') \in \mathcal{R}$. In this case we instead fail to establish that HO-bisimilarity is transitive; again we do not know if there is a counterexample. The problems are highly technical and mainly involves how freshness conditions are propagated in the proofs.

The definition does not aspire to full abstraction with respect to observational criteria, and in this way it is very different from most existing work on higher-order calculi. It can immediately be seen that it is not complete in any sensible respect: the agents $\mathbf{0}$ and $(\!|\{M \Leftarrow \mathbf{0}\}|\!)$ should be indistinguishable from an observation viewpoint since neither has a transition and $M \Leftarrow \mathbf{0}$ does not give M any invocation possibilities, yet they fail bisimilarity on Clause 1(b). On the other hand it is straightforward to establish soundness for reasonable criteria. For example, say that a process P has the barb M if P has a transition with subject M, and that a congruence relation is barbed if related agents have the same barbs. Clause 4 in Definition 27 directly gives that HO-bisimilarity is barbed.

5 Conclusion

We have defined higher-order psi-calculi in a smooth extension from ordinary psi-calculi, meaning that we can re-use much of the mechanised proofs. Ordinary psi-calculi can be lifted in a systematic way to higher-order counterparts, yielding higher-order versions of the applied pi-calculus and the concurrent constraint pi-calculus.

We have integrated the proofs with our existing proof repositories based on Isabelle/Nominal. In some cases this process is surprisingly easy. In other places there are roadblocks related to the exact working of nominal datatypes with complicated constructors and locales. Yet we regard this effort as worthwhile. For the main results like Theorem 31 it is not efficient to embark on manual proofs; in psi-calculi these are notoriously error-prone because of the length, the number of cases to check, and the numerous side conditions related to freshness of names.

There are several interesting avenues to explore. One obvious is higher-order weak bisimulation and congruence. Here an immediate problem is that we can encode Sum, and therefore the usual example that weak bisimulation is not preserved by Sum may imply that it is not preserved by Parallel. For example, let us define weak higher order bisimulation by adapting the weak bisimulation from [BJPV10] in the same way as we here do for strong bisimulation. In other words, we require that a clause needs a weakly bisimilar clause. Then consider

the agents

$$\begin{array}{lcl} P & = & (\!|\{M \Leftarrow \tau \,.\, a \,.\, \mathbf{0}\}|\!) \\ Q & = & (\!|\{M \Leftarrow a \,.\, \mathbf{0}\}|\!) \\ R & = & \mathbf{run}\ M \mid (\!|\{M \Leftarrow b \,.\, \mathbf{0}\}|\!) \end{array}$$

Here we have that P and Q are weakly higher-order bisimilar but $P|R$ and $Q|R$ are not. This indicates that a less straightforward definition will be necessary. An obvious attempt is to require of clauses that they are weakly congruent (rather than weakly bisimilar), and this requires that both weak congruence and weak bisimilarity are defined in one simultaneous co-inductive definition, since each depends on the other.

The relationship between a calculus and its canonical higher-order counterpart should also be investigated. For example, bisimilarity on first-order processes is hopefully the same, and perhaps there is an interesting class of calculi where the canonical higher-order calculus can be encoded. Finally, higher-order calculi should be combined with other extensions of the psi-calculi framework. We have successfully integrated higher-order calculi and ordinary bisimulation with the broadcast extension presented in [BHJ$^+$11]. Here the total effort in the formalisation was roughly half a day, mainly to textually combine the proof files. This is a striking advantage of using formal proof repositories. We could also extend our recent work on sort systems [BGP$^+$12] to a higher-order setting.

In the invocation rule, the handle M must be exactly the same in the premise (where it occurs in $M \Leftarrow P$) and conclusion (where it occurs in $\mathbf{run}\ M$). This means that it is not possible to directly describe extraction of handles from complicated data structures. For example, consider one process defining two clauses $M_i \Leftarrow P_i$, and then sending the pair of the handles $\langle M_1, M_2\rangle$. A receiving process might want to receive the pair and invoke its first element. Expressing this as $a(x) \,.\, \mathbf{run}\ \pi_1(x)$ will not work. After the communication of $\langle M_1, M_2\rangle$ this becomes $\mathbf{run}\ \pi_1(\langle M_1, M_2\rangle)$ but the environment contains $M_1 \Leftarrow P_1$ and not $\pi_1(\langle M_1, M_2\rangle) \Leftarrow P_1$. What would be necessary here is a rewriting theory of projections, with axioms such as $\pi_1(\langle M_1, M_2\rangle) \to M_1$, to be used in the entailment relation.

Most cases of simple extractions such as projections can be handled by pattern matching, as in this case $a(\lambda x, y)\langle x, y\rangle \,.\, \mathbf{run}\ x$. In more complicated structures, for example to represent encryption and decryption of handles, pattern matching will not be sufficient and we must include information about the evaluation of handles in the assertions, where scoping can be used to make them local. This device is is already present for communication subjects as the channel equivalence predicate. It remains to be seen if it is feasible to introduce a similar relation for handles.

Acknowledgements We are very grateful to Magnus Johansson and Björn Victor for constructive and inspiring discussions.

References

[ÅPR] Johannes Åman Pohjola and Palle Raabjerg. Isabelle proofs for higher-order psi-calculi. Proof scripts for higher-order psi-calculi. Available at `http://www.it.uu.se/research/group/mobility/theorem/hopsi.tar.gz`.

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

[BGP+12] Johannes Borgström, Ramūnas Gutkovas, Joachim Parrow, Björn Victor, and Johannes Åman Pohjola. Sorted psi-calculi with generalised pattern matching. Unpublished manuscript, 2012.

[BHJ+11] Johannes Borgström, Shuqin Huang, Magnus Johansson, Palle Raabjerg, Björn Victor, Johannes Åman Pohjola, and Joachim Parrow. Broadcast psi-calculi with an application to wireless protocols. In Gilles Barthe, Alberto Pardo, and Gerardo Schneider, editors, *SEFM*, volume 7041 of *Lecture Notes in Computer Science*, pages 74–89. Springer, 2011.

[BJPV09] Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Psi-calculi: Mobile processes, nominal data, and logic. In *Proceedings of LICS 2009*, pages 39–48. IEEE, 2009. Full version at `http://user.it.uu.se/~joachim/psi-long.pdf`.

[BJPV10] Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Weak equivalences in psi-calculi. In *Proceedings of LICS 2010*, pages 322–331. IEEE, 2010.

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

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

[DHS09] Romain Demangeon, Daniel Hirschkoff, and Davide Sangiorgi. Termination in higher-order concurrent calculi. In Farhad Arbab and Marjan Sirjani, editors, *FSEN*, volume 5961 of *Lecture Notes in Computer Science*, pages 81–96. Springer, 2009.

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

[Joh10] Magnus Johansson. *Psi-calculi: a framework for mobile process calculi*. PhD thesis, Uppsala University, May 2010.

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

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

[LPSS08] Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi, and Alan Schmitt. On the expressiveness and decidability of higher-order process calculi. In *LICS*, pages 145–155. IEEE Computer Society, 2008.

[LPSS10] Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi, and Alan Schmitt. On the expressiveness of polyadic and synchronous communication in higher-order process calculi. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *ICALP (2)*, volume 6199 of *Lecture Notes in Computer Science*, pages 442–453. Springer, 2010.

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

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

[San96] Davide Sangiorgi. Bisimulation for higher-order process calculi. *Inf. Comput.*, 131(2):141–178, 1996.

[San98] Davide Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8(5):447–479, 1998. An extended abstract appeared in the *Proceedings of MFCS '95*, LNCS 969: 479–488.

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

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

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

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

Chapter 4

Extending psi-calculi

As is mentioned in Section 1.4.3, the two papers preceding this chapter makes light work of the fact that the theories presented have been formalised in Isabelle. In this Chapter, we will go deeper into some of the particulars of the formalisation, why it is as big as it is, how the extensions affect the theory files, and what may be done to ease work on possible future extensions.

Convincing a proof assistant that theories are correct is not generally a trivial task. The basic theories of the psi-calculi framework itself took a single person (Jesper Bengtson, Formalising Process Calculi [Ben10]) many months to verify through Isabelle.

Extending psi-calculi with new semantic rules can involve a significant amount of work, as those rules will have an impact on the overall theory. This of course depends on the number of rules and their complexity. It may be surprising, however, to discover where this effort is actually expended.

A great part of the extension work goes into extending basic definitions and proofs of induction and inversion rules, which is often very repetitive work. In the case of the broadcast extension, the size of that part nearly doubled. Much of this involved copying and pasting definitions and proofs with often relatively minor modifications: similar semantic rules breed similar induction and inversion rules and proofs. Repeating patterns in the definitions and proofs become more apparent through the copying and modification of existing proofs and definitions. Such patterns are useful for automation, either in building definitions and lemmas, or in building heuristics and proof strategies for automating the corresponding proofs.

4.1 Theory Files

To get started, we need to understand the basic structure of the formalisation. These are the most significant parts of the file hierarchy of the formal proofs, dealing directly with the calculus definitions themselves and the standard re-

sults. They are listed in order of dependence.

- `agent.thy`: The syntax of psi-calculi is set up. Lemmas are established for comparison and freshness on sequences of binders, making it easier to handle the binding sequences in the calculi. A substitution function is also defined for the known parts of the framework, while a locale (an environment of abstraction in Isabelle) is set up to handle substitution on the parameters (terms, assertions and conditions).

- `frame.thy`: Is similar to `agent.thy`, in that it primarily sets up the syntax of frames and establishes lemmas for the binding sequences. It also sets up locales for defining the custom logic.

- `semantics.thy`: The semantics of the calculi is defined, with induction and inversion lemmas (explained in Section 4.1.1). While that may sound simple and innocuous, `semantics.thy` ends up being the largest file by far in the mechanical theories, and the one where most effort is usually spent in working out extensions. We will take a closer look at how extensions impact the individual files in Section 4.2, and in Section 4.3, we will explain some of the reasons for the size of `semantics.thy`.

- `simulation.thy`: Definition of and lemmas pertaining to process simulation. Simulation pertains to one process doing a one-way simulation of another process. Simulation results are combined to show bisimulation results.

- `bisimulation.thy`: Definition of bisimulation with accompanying induction and coinduction lemmas.

- `simStructCong.thy`: A core part of the results is the derived structural congruences. `simStructCong.thy` proves the relevant simulations needed for those results.

- `bisimStructCong.thy`: Further, `bisimStructCong.thy` uses those simulations to prove the bisimulation results that make up the derived structural congruences.

4.1.1 Induction and Inversion

Induction rules are inferences containing a base case and one or more inductive steps. A simple classic example is list induction: "If the proposition holds for the empty list, and we can take any list for which the proposition holds, and add any element to that list without breaking the proposition, then the proposition holds for all lists.", where the base case is the empty list and the inductive step any list to which we add an element. A central point of inductive proofs is that when working on an inductive step, we get to assume that the proposition holds prior to the step (the induction hypothesis). So for every

inductive step we just have to prove that taking that step does not break the proposition.

The induction rules set up in `semantics.thy` regulate structural induction on the psi-calculi semantics. There are different rules to prove propositions specific to each kind of action. This means that if a proof is only needed for output actions, the inductive cases can be limited to only those that are derivable from an output action.

Inversion rules simply facilitate proofs by case distinction. A simple example would be that sometimes a proof needs to be significantly different based on whether or not two values are equal. The proposition should hold in either case, but splitting it into the cases of "equal" and "not equal" can simplify the proof significantly.

The inversion rules of `semantics.thy` are based on specific situations in the semantics, such as "any action under parallel", which in standard psi-calculi has four cases (internal communication in either direction, and external communication from either left or right process), or "an input under parallel", which has just two cases (transition from the left or the right process). Since inversion rules are simply for case distinction, they contain no induction hypotheses.

4.2 Impacts of Extensions

To understand how an extension impacts the basic theories, let us first take a statistical look at how much the sizes of the above files increase in the broadcast and higher order extensions, compared to the original. Increase in size can be used to roughly measure the effort expended on a certain file.

Table 4.1 shows the file sizes in bytes before and after extension with a percentage denoting the increase relative to the original size. This indicates roughly the amount of effort expended on each file in comparison to the originals, and thus about the comparative impact of the two extensions. Not surprisingly, broadcast psi with its 7 new semantic rules was a heavier task in its basic extension than higher-order psi. It should be noted though, that higher-order psi also gives rise to a notion of higher order bisimulation (definition 27 in Higher-order psi-calculi), which ended up requiring much more effort than the basic extension we summarise here. I did not participate in that part of the formalisation work.

Table 4.2 shows what percentage of total new bytes that go into each of these files for both broadcast psi and higher-order psi. This indicates roughly the distribution of the total amount of effort for each extension.

These tables also reflect the nature of the two extensions. Broadcast psi has no new process syntax, and thus leaves `agent.thy` entirely alone, whereas higher-order psi adds invocation. The work going into `frame.thy` is nearly the same for both extensions. This is because both extensions add to the predefined conditions, broadcast psi with input and output connectivity ($M \dot{\succ} K$ and $M \dot{\prec} K$), and higher-order psi with clauses ($M \Leftarrow P$). `simStructCong.thy`

	Psi	Broadcast psi		Higher-order psi	
	bytes	bytes	% increase	bytes	% increase
`agent.thy`	47815	47815	0	49354	3.22
`frame.thy`	100512	105297	4.76	105945	5.41
`semantics.thy`	771478	1414013	83.29	841568	9.09
`simulation.thy`	25819	35836	38.80	25819	0
`bisimulation.thy`	27479	42080	53.14	29880	8.74
`simStructCong.thy`	135755	313626	131.02	136031	0.20
`bisimStructCong.thy`	86075	127524	48.15	86075	0
Total	1194933	2086191	74.59	1274672	6.67

Table 4.1: File size increase in percentage of original size

	Broadcast psi	Higher-order psi
`agent.thy`	0	1.93
`frame.thy`	0.54	6.81
`semantics.thy`	72.09	87.90
`simulation.thy`	1.12	0
`bisimulation.thy`	1.64	3.01
`simStructCong.thy`	19.96	0.35
`bisimStructCong.thy`	4.65	0

Table 4.2: Percentage of total new bytes going into each file

is also interesting to consider, as the difference between the two extensions seems disproportionate. This is because the one new semantic rule of higher-order psi is so simple. It just says that an invocation may act like a process from a matching clause. The new rules of broadcast psi deal with additional communication actions and new ways of opening and closing scopes, which affects the congruences dramatically.

Even though the structural congruences are one of the more interesting parts of the formalisation, it still only makes up for about 20% of the effort of broadcast psi. From Table 4.2, we discover that most of the work by far goes into `semantics.thy`. In Table 4.1, we see that `semantics.thy` is already much bigger than any of the other files, and in fact nearly doubled in size for the broadcast extension.

4.3 The Inner Workings of semantics.thy

In spite of the size of `semantics.thy`, there is very little in the file that is explicitly mentioned in the papers in Chapter 2 and 3. Most of the significant results of the basic theories are derived later in `bisimulation.thy`, `simStructCong.thy` and `bisimStructCong.thy`. So in this section, we will take a short look at the contents of `semantics.thy` to explain why it is so big.

4.3.1 Semantics and Freshness

Freshness conditions are requirements of freshness attached to a semantic rule. For example, the Com rule (Table 1.1 in Section 1.5.1) comes attached with the condition $\widetilde{a}\#Q$, meaning that every name in the list $\widetilde{a}$ must be fresh in Q.

The most important part of `semantics.thy` is the inductive definition of the semantics, found early in the file. As mentioned in [BJPV09], the Par and Comm rules each have a symmetric version, explicitly defined in the mechanical theories. There is another apparent difference that is not really touched upon in the papers in Chapter 2 and 3: More freshness conditions are explicitly defined in the semantics of the formal theories than what is actually shown in those papers [Ben10]. This is because in the mechanical theories, the semantic rules of the papers are derived as lemmas from the initial inductive definition. Most of the freshness conditions can be discharged because the binding structures imply that α-conversion can always be applied to achieve those conditions. The remaining conditions are the ones presented in the papers as conditions on the semantics.

The ostensibly superfluous freshness conditions are included for the benefit of the induction and inversion rules. Doing it this way allows us to derive more useful induction rules, with included freshness conditions. If we did not include the freshness conditions as part of the induction rules, it would be necessary to derive them by α-converting terms every time an induction proof is made. In essence, this allows us to easily use the Barendregt Variable Convention [Bar84] in subsequent proofs.

The Barendregt Variable Convention is a useful assumption when constructing proofs involving name binders. Taken from Barendregt's book on the λ-calculus, it states:

1. Convention. Terms that are α-congruent are identified. So now we write $\lambda x.x \equiv \lambda y.y$, etcetera.

2. Variable Convention. If $M_1, \ldots, M_n$ occur in a certain mathematical context (e.g. definition, proof), then in these terms all bound variables are chosen to be different from the free variables.

Unfortunately, as Urban and Norrish shows [UN05], this can lead to inconsistencies if used as a general rule. It does not work for every kind of term construction, and so using the Variable Convention in a mechanical setting like Isabelle requires some extra work in proving that term constructions are compatible with α-conversion. That is what the freshness conditions are there for, and they are the reason for many of the complications in `semantics.thy`.

4.3.2 Automatic Generation

The Nominal package is able to generate suitable induction and inversion rules automatically in many contexts. As is noted in [Ben10], the automatic generation provided by the Nominal package is not quite sufficient for the psi-calculi

framework. Generation of inversion rules falls short when dealing with sequences of binders. Bengtson suggests a technique to handle generation of these in his thesis. The technique has not been implemented however, and thus it is still necessary to manually construct and prove the validity of the rules in `semantics.thy`. The situation is similar for induction rules. There are standard methods for deriving induction rules in the Nominal framework, but for induction on frames every step introduces new binders, and thus new α-conversions to handle. To avoid having to deal with this in later proofs, we must also build specialised induction rules.

4.4 Common Features of Extensions

There are a number of evident ways one can extend the framework, some less involved than others.

4.4.1 Semantic Rule Modifications

Modifying semantic rules during development is a frequent occurrence, and it is often surprisingly nonintrusive. Unless a lemma is actually rendered false, pushing through a relatively minor modification will often be achievable in a few days or less. The fundamental structures and case statements for the proof already exists.

As an example, this was the next-to-last instance of the invocation rule in higher-order psi:

$$\textsc{Invocation}\ \frac{\Psi \vdash M(\lambda\widetilde{\mathrm{x}})N \Leftarrow P \qquad \Psi \rhd P[\widetilde{\mathrm{x}} := \widetilde{T}] \xrightarrow{\alpha} P'}{\Psi \rhd \mathbf{run}\, M\langle N[\widetilde{\mathrm{x}} := \widetilde{T}]\rangle \xrightarrow{\alpha} P'}$$

This version of the rule implements arguments to higher order processes through name substitution. It was discarded in the last revision of the rule because it caused bisimilarity to not be closed under parallel, and was replaced by:

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

The change for this last version was pushed through `semantics.thy` in under a day. This means that one should not be too afraid to experiment with variations on the same rule while working on more significant extensions. Proof cases tend to be easy to port to similar rules.

4.4.2 New Semantic Rules

The broadcast psi and higher-order psi extensions in this thesis involve mainly the addition of semantic rules. Depending on their complexity, adding new

semantic rules to `semantics.thy` can be a somewhat tedious affair. A new semantic rule in itself does not usually require any new lemmas. It is merely a problem of extending the already existing induction and inversion rules with new cases, and treating those cases wherever the rules are used.

As mentioned earlier in Section 4.1.1, most of the induction and inversion rules are built for a number of fairly specific situations. Thus, for most semantic rule additions, we will only have to update some induction and inversion rules.

4.4.3 New Syntax

Psi-calculi contains a total of 6 syntactic categories. In the context of process description, we have processes, message terms and assertions, and in the context of semantics we also have actions, conditions and frames. Of those, message terms, assertions and conditions are either fully or partially defined as parameters in an instantiation, while processes, actions and frames are explicit parts of the theory.

Additions to any of the syntactic categories will usually be prompted by a need for new functionality, and thus new semantics in the calculi. The nature of the semantic changes and their impact depend heavily on which syntactic category we are dealing with. The addition of new process syntax will usually be accompanied by the addition of one or more semantic rules to deal with the new syntax, as was the case with higher-order psi.

For message terms, there might be cases where an extension would warrant adding explicit syntax to the theory, to do something special with certain kinds of messages in the semantics. So far though, no concrete example of such an extension exists. Such an extension may add capability to the framework, but will also make it more specialised.

Assertions and conditions are perhaps the only categories where explicit extensions would not make sense at all. If we want either to affect something in the semantics, the way to do it is through the conditions, as it happens with channel equivalence, broadcast connectivity and higher order clauses.

Extension of action syntax, the addition of more possible actions, is somewhat intrusive, causing both a need for more cases in many proofs, and additional induction and inversion rules. Many of those rules work explicitly on different kinds of actions, and thus counterparts for any additional actions should be created. This is necessary in broadcast psi, as it extends actions with broadcast in and broadcast out.

4.5 Related Work

As mentioned in Section 1.4.1, in the formalisation of psi-calculi we use nominal theory [Pit03, UT05] to represent terms with binders. Another common way of representing such terms is to use De Bruijn indices [dB72]. Using De Bruijn indices would make alpha-equivalent terms syntactically equivalent, thus making

such term comparisons trivial. Instead, binding depths must be recalculated every time the term structure is changed. As demonstrated by Hirschkoff in [Hir97] with his Coq formalisation of parts of the pi-calculus metatheory, this causes a significant amount of effort to be spent on such recalculations.

Syntax springing from nominal theory presents a more human-readable format than De Bruijn indices. The nominal package does also implement a decent amount of automation when it comes to α-conversion and alpha-equivalences. But as we may note later from Chapter 4, effort-wise, the choice appears to be a trade-off. The effort expended on handling α-conversions seems comparable to the effort expended by Hirschkoff on handling recalculations of binders in [Hir97].

We mentioned in the beginning of this chapter that in many cases we end up copying and pasting definitions and proofs for similar situations. Recent work by Whiteside, Aspinall, Dixon and Grov [WADG11] introduce proof refactoring for Isabelle, which may provide ways to alleviate such practices.

On the subject of extending large proofs, it is also worth mentioning the seL4 microkernel project, where Matichuk and Murray make use of a technique they call Extensible Specifications [MM12] for writing specifications with multiple levels of abstractions. To compare, the psi-calculi framework can be said to implement one level of such abstraction through its parameters. Another level of abstraction may present itself in the now emerging family of psi-calculi frameworks.

4.6 Future Work

4.6.1 Automatic Inversion Rule Generation

As mentioned in Section 4.3.2, there already exists a possible method for generating the inversion rules, though it has not been implemented. If we compare the size of `semantics.thy` with and without the inversion rules, we can estimate the benefit of automating the generation of them. As it turns out in Table 4.3, it would result in a cut of about 20% from `semantics.thy`, both in standard psi and the broadcast psi extension. This is quite significant, especially considering that the method itself is conceptually simple.

The main reason it has not been implemented yet is simply the lack of a high level language in Isabelle for generating lemmas and proofs. To effectively develop code on the ML level of Isabelle requires a significant amount of training and knowledge about the codebase. There is room for development in this respect.

4.6.2 Freshness Tactics

A significant part of the formal proofs concern the freshness of names in a variety of constructions. Most of them are very trivial one-liners, and for the most part they tend to be short, on the form "from *preconditions* have $\widetilde{a}\#B$

	decrease in bytes	decrease in %
Psi	146337	18.97
Broadcast psi	323610	22.89

Table 4.3: Effect on psi and broadcast psi of automating inversion rules

by *rule*", for some list of names $\widetilde{a}$ and some construction B. Here, "from" indicates the premise of the lemma, "have" indicates the conclusion, and "by" the proof itself (or the tactic applied to generate it). There are a number of common methods for proving freshness, used throughout `semantics.thy` and other theory files. The following sentence is repeated quite frequently in a number of proofs, for example:

from `$(p \bullet A_P)\#A_Q$` `$(p \bullet A_P)\#(q \bullet A_Q)$` `$(p \bullet A_P)\#\Psi_Q$`
have "$(p \bullet A_P)\#(p \bullet \Psi_Q)$"
by(simp add: freshChainSimps)

often simply with different name sequences and constructions:

from `$(q \bullet A_Q)\#xvec$` `$(q \bullet A_Q)\#(r \bullet xvec)$` `$(q \bullet A_Q)\#N$`
have "$(q \bullet A_Q)\#(r \bullet N)$"
by(simp add: freshChainSimps)

Or even just simpler permutation inferences:

from `$A_P\#P$` have "$(p \bullet A_P)\#(p \bullet P)$"

Thus, some improvement may be entailed by writing a tactic that would simply

1. Take in any number of facts about permutations and freshnesses.

2. Use those facts to prove as many additional freshness properties as is feasible using common inferences like the ones shown above.

Quantifying the benefits of such an approach would be difficult without implementing it. There are also potential downsides. In Isabelle, there are a variety of different ways of applying rules in proofs. Very often, rules are applied indirectly through automated tactics, such as `auto` and `simp`. Depending on the effectiveness of the heuristics of such tactics, this can sometimes be time-intensive. Direct rule applications are much more efficient, though they require more effort to handle manually. To ensure efficiency, the tactic we build should only make use of well-defined direct rule applications. Applications of a tactic like "by(simp add: freshChainSimps)" tend indeed to be time-consuming. This is generally fine when we are deriving some specific fact, but for automatic derivation of multiple facts, it would probably be too costly.

4.7 Conclusion

The Isabelle formalisations of the psi-calculi frameworks are large but manageable. A great part of the bulk lies with the definitions and proofs of specialised inversion and induction rules necessary to prevent α-conversions from becoming a problem in the more interesting proofs of the standard results. It may be possible to automate generation of the inversion rules and many of the freshness proofs, which could significantly diminish the amount of effort required in future extensions of the theories. Apart from the additional certainty the formalisations afford, minor modifications of the frameworks often require little effort in the formalisation. This is because most of the proofs will still be applicable with little to no changes, and in difference to manual proofs, rechecking formalised proofs is a mechanised task.

Bibliography

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

[AG97] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. In Richard Graveman, Philippe A. Janson, Clifford Neumann, and Li Gong, editors, *CCS '97: Proceedings of the 4th ACM conference on Computer and communications security*, pages 36–47. ACM, 1997.

[AILS07] Luca Aceto, Anna Ingólfsdóttir, Kim Guldstrand Larsen, and Jiří Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, 2007.

[Arm07] Joe Armstrong. A history of Erlang. In Barbara G. Ryder and Brent Hailpern, editors, *HOPL III: Proceedings of the third ACM SIGPLAN conference on History of programming languages*, pages 1–26. ACM, 2007.

[Bal04] Clemens Ballarin. Locales and locale expressions in Isabelle/Isar. In Stefano Berardi, Mario Coppo, and Ferruccio Damiani, editors, *Types for Proofs and Programs*, volume 3085 of *Lecture Notes in Computer Science*, pages 34–50. Springer, 2004.

[Bar84] Hendrik Pieter Barendregt. *The Lambda Calculus – Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.

[Ben10] Jesper Bengtson. *Formalising Process Calculi*. PhD thesis, Uppsala University, June 2010.

[BHJ+11] Johannes Borgström, Shuqin Huang, Magnus Johansson, Palle Raabjerg, Björn Victor, Johannes Åman Pohjola, and Joachim Parrow. Broadcast psi-calculi with an application to wireless protocols. In Gilles Barthe, Alberto Pardo, and Gerardo Schneider,

editors, *Software Engineering and Formal Methods*, volume 7041 of *Lecture Notes in Computer Science*, pages 74–89. Springer, 2011.

[BJPV09] Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Psi-calculi: Mobile processes, nominal data, and logic. In *2009 24th Annual IEEE Symposium on Logic In Computer Science*, pages 39–48. IEEE Computer Society, 2009.

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

[dB72] Nicolaas Govert de Bruijn. Lambda-calculus notation with nameless dummies: a tool for automatic formula manipulation with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 34(5):381–392, 1972.

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

[EM01] Cristian Ene and Traian Muntean. A broadcast-based calculus for communicating systems. In *15th International Parallel and Distributed Processing Symposium (IPDPS'01)*, page 149. IEEE Computer Society, 2001.

[FM00] Gian Luigi Ferrari and Ugo Montanari. Tile formats for located and mobile systems. *Information and Computation*, 156(1-2):173–235, 2000.

[GFM08] Fatemeh Ghassemi, Wan Fokkink, and Ali Movaghar. Restricted broadcast process theory. In Antonio Cerone and Stefan Gruner, editors, *2008 Sixth IEEE International Conference on Software Engineering and Formal Methods*, pages 345–354. IEEE Computer Society, 2008.

[God07] Jens Chr. Godskesen. A calculus for mobile ad hoc networks. In Amy L. Murphy and Jan Vitek, editors, *Coordination Models and Languages*, volume 4467 of *Lecture Notes in Computer Science*, pages 132–150. Springer, 2007.

[God10] Jens Chr. Godskesen. Observables for mobile and wireless broadcasting systems. In Dave Clarke and Gul A. Agha, editors, *Coordination Models and Languages*, volume 6116 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2010.

[Gut11] Ramūnas Gutkovas. Exercising psi-calculi : A psi-calculi workbench. Master's thesis, Uppsala University, June 2011.

[Haw07] Jeff Hawkins. Why can't a computer be more like a brain? *Spectrum, IEEE*, 44(4):21–26, April 2007.

[Hir97] Daniel Hirschkoff. A full formalisation of pi-calculus theory in the calculus of constructions. In Elsa L. Gunter and Amy P. Felty, editors, *Theorem Proving in Higher Order Logics*, volume 1275 of *Lecture Notes in Computer Science*, pages 153–169. Springer, 1997.

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

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

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

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

[MM12] Daniel Matichuk and Toby C. Murray. Extensible specifications for automatic re-use of specifications and proofs. In George Eleftherakis, Mike Hinchey, and Mike Holcombe, editors, *Software Engineering and Formal Methods*, volume 7504 of *Lecture Notes in Computer Science*, pages 333–341. Springer, 2012.

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

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

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

[Par95] Joachim Parrow. Interaction diagrams. *Nordic Journal of Computing*, 2(4):407–443, 1995.

[PBRP] Joachim Parrow, Johannes Borgström, Palle Raabjerg, and Johannes Åman Pohjola. Higher-order psi-calculi. Submitted to MSCS.

[PCC06] Andrew Phillips, Luca Cardelli, and Giuseppe Castagna. A graphical representation for biological processes in the stochastic pi-calculus. In Corrado Priami, Anna Ingólfsdóttir, Bud Mishra, and Hanne Riis Nielson, editors, *Transactions on Computational Systems Biology VII*, volume 4230 of *Lecture Notes in Computer Science*, pages 123–152. Springer, 2006.

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

[PPQ05] Davide Prandi, Corrado Priami, and Paola Quaglia. Process calculi in a biological context. *Bulletin of the EATCS*, 85:53–69, 2005.

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

[Pri95] Corrado Priami. Stochastic pi-calculus. *The Computer Journal*, 38(7):578–589, 1995.

[PV98] Joachim Parrow and Björn Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *13th Annual IEEE Symposium on Logic in Computer Science (LICS'98)*, pages 176–185. IEEE Computer Society, 1998.

[San93] Davide Sangiorgi. From pi-calculus to higher-order pi-calculus - and back. In Marie-Claude Gaudel and Jean-Pierre Jouannaud, editors, *TAPSOFT '93: Theory and Practice of Software Development*, volume 668 of *Lecture Notes in Computer Science*, pages 151–166. Springer, 1993.

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

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

[Tho89] Bent Thomsen. A calculus of higher order communicating systems. In *POPL '89: Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 143–154. ACM Press, 1989.

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

[UN05] Christian Urban and Michael Norrish. A formal treatment of the Barendregt variable convention in rule inductions. In Randy Pollack, editor, *MERLIN '05: Proceedings of the 3rd ACM SIGPLAN workshop on Mechanized reasoning about languages with variable binding*, pages 25–32. ACM, 2005.

[UT05] Christian Urban and Christine Tasson. Nominal techniques in Isabelle/HOL. In Robert Nieuwenhuis, editor, *Automated Deduction – CADE-20*, volume 3632 of *Lecture Notes in Computer Science*, pages 38–53. Springer, 2005.

[WADG11] Iain Whiteside, David Aspinall, Lucas Dixon, and Gudmund Grov. Towards formal proof script refactoring. In James H. Davenport, William M. Farmer, Josef Urban, and Florian Rabe, editors, *Intelligent Computer Mathematics*, volume 6824 of *Lecture Notes in Computer Science*, pages 260–275. Springer, 2011.

Recent licentiate theses from the Department of Information Technology

2012-007 Margarida Martins da Silva: *System Identification and Control for General Anesthesia based on Parsimonious Wiener Models*

2012-006 Martin Tillenius: *Leveraging Multicore Processors for Scientific Computing*

2012-005 Egi Hidayat: *On Identification of Endocrine Systems*

2012-004 Soma Tayamon: *Nonlinear System Identification with Applications to Selective Catalytic Reduction Systems*

2012-003 Magnus Gustafsson: *Towards an Adaptive Solver for High-Dimensional PDE Problems on Clusters of Multicore Processors*

2012-002 Fredrik Bjurefors: *Measurements in Opportunistic Networks*

2012-001 Gunnika Isaksson-Lutteman: *Future Train Traffic Control – Development and deployment of new principles and systems in train traffic control*

2011-006 Anette Löfström: *Intranet Use as a Leadership Strategy*

2011-005 Elena Sundkvist: *A High-Order Accurate, Collocated Boundary Element Method for Wave Propagation in Layered Media*

2011-004 Niclas Finne: *Towards Adaptive Sensor Networks*

2011-003 Rebecka Janols: *Tailor the System or Tailor the User? How to Make Better Use of Electronic Patient Record Systems*

2011-002 Xin He: *Robust Preconditioning Methods for Algebraic Problems, Arising in Multi-Phase Flow Models*

UPPSALA
UNIVERSITET

Department of Information Technology, Uppsala University, Sweden