

# Higher-order psi-calculi

Joachim Parrow      Johannes Borgström      Palle Raabjerg

Johannes Åman Pohjola

January 26, 2011

## 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.

## 1 Introduction

Psi-calculi are 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  $\bar{a}P.Q$  sends the process  $P$  along  $a$  and then continues as  $Q$ . A recipient looks like  $a(X).(R|X)$ , receiving a process  $P$  and continuing as  $R|P$ , thus  $\bar{a}P.Q|a(X).(R|X)$  has a transition leading to  $Q|(R|P)$ . 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 for well-formed processes, 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  $\bar{a}P.Q$ . The normal definition would require the same action  $\bar{a}P$  leading to an agent that bisimulates  $Q$ . In some circumstances this is a too strong requirement. Assume  $P \sim P'$ , then it is reasonable to let  $\bar{a}P.Q \sim \bar{a}P'.Q$  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  $M, N, \dots$  and we write  $\overline{MN}.P$  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  $\underline{K}(\lambda\tilde{x})L.Q$  to represent an agent that can input along the channel  $K$ , receiving some object matching the pattern  $\lambda\tilde{x}L$ . These two agents can interact under two conditions: first, the two channels must be *channel equivalent*, as defined by the channel equivalence predicate  $M \dot{\leftrightarrow} K$ , and second,  $N$  must match the pattern, i.e.,  $N = L[\tilde{x} := \tilde{T}]$  for some sequence of terms  $\tilde{T}$ . The receiving agent then continues as  $Q[\tilde{x} := \tilde{T}]$ .

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

$$\Psi \triangleright \overline{MN}.P \mid \underline{K}(\lambda\tilde{x})L.Q \xrightarrow{\tau} P \mid Q[\tilde{x} := \tilde{T}]$$

exactly when  $N = L[\tilde{x} := \tilde{T}]$  and  $\Psi \vdash M \dot{\leftrightarrow} K$ . The latter says that the assertion  $\Psi$  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*  $\varphi$  in the **if** construct: we have that

$$\Psi \triangleright \mathbf{if} \varphi \mathbf{then} P \xrightarrow{\alpha} P'$$

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

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

the agent  $Q$  uses all entailments provided by  $\Psi$ , 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 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 to introduce the notion of a *clause*  $M \Leftarrow P$ , meaning that the data term  $M$  can be used as a handle to invoke the behaviour of  $P$  in the agent **run**  $M$ . A sender can transmit the handle  $M$  in an ordinary output  $\bar{a}M$  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  $b$ .

The only formal extension to the psi framework is the new agent form *invocation* **run**  $M$ , meaning invoke an agent represented by  $M$ , with the corresponding rule of action

$$\text{INVOCATION} \frac{\Psi \vdash M \Leftarrow P \quad \Psi \triangleright P \xrightarrow{\alpha} P'}{\Psi \triangleright \mathbf{run} \ M \xrightarrow{\alpha} P'}$$

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

$$(\nu b)(\bar{a}M_b.Q \mid (\{M_b \Leftarrow P\})) \mid a(x).(\mathbf{run} \ x \mid R)$$

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  $\tau$ -transition, which can be followed by an invocation:

$$\begin{array}{l|l}
 (\nu b)(\bar{a}M_b . Q \mid (\{M_b \leftarrow \alpha . P\})) & a(x) . (\mathbf{run} \ x \mid R) \quad \xrightarrow{\tau} \\
 (\nu b)(Q \mid (\{M_b \leftarrow \alpha . P'\})) & \mathbf{run} \ M_b \mid R \quad \xrightarrow{\alpha} \\
 (\nu b)(Q \mid (\{M_b \leftarrow \alpha . P'\})) & P' \mid R
 \end{array}$$

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:

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

Here the trigger  $b$  is used in a normal communication to activate  $P$ . A 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.

### 1.3 Exposition

In the next section we recapitulate the definitions of psi-calculi from [BJPV09]. 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. Finally we explore a slightly amended bisimulation definition which corresponds more closely to the higher-order bisimulations usually found in the literature.

Our goal is to establish all formal proofs in the interactive theorem prover Isabelle. At the time of writing this we are not completely done; we comment on the status of each theorem. One main point of this work is that the existing proof repository for psi-calculi can be re-used with comparably little effort.

## 2 Psi-calculi

We assume the reader to be acquainted with the main ideas of psi-calculi as presented in [BJPV09], and here recapitulate the relevant definitions.

As usual we assume a countably infinite set of atomic *names*  $\mathcal{N}$  ranged over by  $a, \dots, z$ . 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  $n(X)$ . We write  $a\#X$ , pronounced “ $a$  is fresh for  $X$ ”, for  $a \notin n(X)$ . 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.,  $n(X)$  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,  $a_1, \dots, a_n$ . The empty sequence is written  $\epsilon$  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  $\{a_1, \dots, a_n\}$ . 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  $\tilde{a} \subseteq n(N) \wedge \tilde{b} \in n(\tilde{M}) \implies \tilde{b} \in n(N[\tilde{a} := \tilde{M}])$  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:*

- T** the (data) terms, ranged over by  $M, N$
- C** the conditions, ranged over by  $\varphi$
- A** the assertions, ranged over by  $\Psi$

and the four equivariant operators:

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

and substitution functions  $[\tilde{a} := \tilde{M}]$ , 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  $M \dot{\leftrightarrow} N$  is a condition, pronounced “ $M$  and  $N$  are channel equivalent” and if  $\Psi$  and  $\Psi'$  are assertions then so is  $\Psi \otimes \Psi'$ . Also we write  $\Psi \vdash \varphi$ , pronounced “ $\Psi$  entails  $\varphi$ ”, for  $(\Psi, \varphi) \in \vdash$ .

**Definition 2** (assertion equivalence). *Two assertions are equivalent, written  $\Psi \simeq \Psi'$ , if for all  $\varphi$  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).

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

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  $(\nu \tilde{b})\Psi$  where  $\tilde{b}$  is a sequence of names that bind into the assertion  $\Psi$ . 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  $\Psi$  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  $(\nu \tilde{b}_1)\Psi_1 \otimes (\nu \tilde{b}_2)\Psi_2 = (\nu \tilde{b}_1 \tilde{b}_2)\Psi_1 \otimes \Psi_2$  where  $\tilde{b}_1 \# \tilde{b}_2, \Psi_2$  and vice versa. We write  $(\nu c)((\nu \tilde{b})\Psi)$  to mean  $(\nu c \tilde{b})\Psi$ .

**Definition 5.** *We define  $F \vdash \varphi$  to mean that there exists an alpha variant  $(\nu \tilde{b})\Psi$  of  $F$  such that  $\tilde{b} \# \varphi$  and  $\Psi \vdash \varphi$ . We also define  $F \simeq G$  to mean that for all  $\varphi$  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  $P, Q, \dots$ , are of the following forms.*

$\mathbf{0}$	Nil
$\overline{MN}.P$	Output
$\underline{M}(\lambda \tilde{x})N.P$	Input
$\mathbf{case} \varphi_1 : P_1 \square \dots \square \varphi_n : P_n$	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  $\underline{M}(\lambda\tilde{x})N.P$  it holds that  $\tilde{x} \subseteq \mathfrak{n}(N)$  is a sequence without duplicates, that in a replication  $!P$  the agent  $P$  is assertion guarded, and that in **case**  $\varphi_1 : P_1 \parallel \dots \parallel \varphi_n : P_n$  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  $\underline{M}(\lambda\tilde{x})N.P$  the intuition is that the pattern  $(\lambda\tilde{x})N$  can match any term obtained by instantiating  $\tilde{x}$ , 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  $\varphi_i$  is true. **case**  $\varphi_1 : P_1 \parallel \dots \parallel \varphi_n : P_n$  is sometimes abbreviated as **case**  $\tilde{\varphi} : \tilde{P}$ , or if  $n = 1$  as **if**  $\varphi_1$  **then**  $P_1$ . In psi-calculi where a condition  $\top$  exists such that  $\Psi \vdash \top$  for all  $\Psi$  we write  $P + Q$  to mean **case**  $\top : P \parallel \top : Q$ .

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  $\overline{MN}$  for  $\overline{MN}.\mathbf{0}$ .

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:*

$$\begin{aligned} \mathcal{F}(\mathbf{0}) &= \mathcal{F}(\underline{M}(\lambda\tilde{x})N.P) = \mathcal{F}(\overline{MN}.P) = \mathcal{F}(\mathbf{case} \tilde{\varphi} : \tilde{P}) = \mathcal{F}(!P) = \mathbf{1} \\ \mathcal{F}(\langle \Psi \rangle) &= \Psi \\ \mathcal{F}(P \mid Q) &= \mathcal{F}(P) \otimes \mathcal{F}(Q) \\ \mathcal{F}((\nu b)P) &= (\nu b)\mathcal{F}(P) \end{aligned}$$

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

The actions  $\alpha$  that agents can perform are of three kinds: output actions, input actions of the early kind, meaning that the input action contains the received object, and the silent action  $\tau$ . The operational semantics consists of

transitions of the form  $\Psi \triangleright P \xrightarrow{\alpha} P'$ . This transition intuitively means that  $P$  can perform an action  $\alpha$  leading to  $P'$ , in an environment that asserts  $\Psi$ .

**Definition 8** (Actions). *The actions ranged over by  $\alpha, \beta$  are of the following three kinds:*

$$\begin{array}{ll} \overline{M}(\nu \tilde{a})N & \text{Output, where } \tilde{a} \subseteq \mathfrak{n}(N) \\ \underline{M}N & \text{Input} \\ \tau & \text{Silent} \end{array}$$

For actions we refer to  $M$  as the *subject* and  $N$  as the *object*. We define  $\text{bn}(\overline{M}(\nu \tilde{a})N) = \tilde{a}$ , and  $\text{bn}(\alpha) = \emptyset$  if  $\alpha$  is an input or  $\tau$ . We also define  $\mathfrak{n}(\tau) = \emptyset$  and  $\mathfrak{n}(\alpha) = \mathfrak{n}(N) \cup \mathfrak{n}(M)$  if  $\alpha$  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). *A transition is of the kind  $\Psi \triangleright P \xrightarrow{\alpha} P'$ , meaning that when the environment contains the assertion  $\Psi$  the well formed agent  $P$  can do an  $\alpha$  to become  $P'$ . The transitions are defined inductively in Table 1. We write  $P \xrightarrow{\alpha} P'$  to mean  $\mathbf{1} \triangleright P \xrightarrow{\alpha} P'$ . In IN the substitution is defined by induction on agents, using substitution on terms, assertions and conditions for the base cases and avoiding captures through alpha-conversion in the standard way.*

Both agents and frames are identified by alpha equivalence. This means that we can choose the bound names fresh in the premise of a rule. In a transition the names in  $\text{bn}(\alpha)$  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  $\text{bn}(\alpha)$  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  $\text{bn}(\alpha) \subseteq \mathfrak{n}(\alpha)$ . 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 formalize the extension to higher-order psi-calculi described in the introduction.

#### 3.1 Basic definitions

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

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

$$\begin{array}{c}
\text{IN} \frac{\Psi \vdash M \leftrightarrow K}{\Psi \triangleright \underline{M}(\lambda \tilde{y})N.P \xrightarrow{\underline{K}N[\tilde{y}:=\tilde{L}]} P[\tilde{y}:=\tilde{L}]} \quad \text{OUT} \frac{\Psi \vdash M \leftrightarrow K}{\Psi \triangleright \overline{M}N.P \xrightarrow{\overline{K}N} P} \\
\\
\text{CASE} \frac{\Psi \triangleright P_i \xrightarrow{\alpha} P' \quad \Psi \vdash \varphi_i}{\Psi \triangleright \mathbf{case} \tilde{\varphi} : \tilde{P} \xrightarrow{\alpha} P'} \\
\\
\text{COM} \frac{\Psi_Q \otimes \Psi \triangleright P \xrightarrow{\overline{M}(\nu \tilde{a})N} P' \quad \Psi_P \otimes \Psi \triangleright Q \xrightarrow{\underline{K}N} Q' \quad \Psi \otimes \Psi_P \otimes \Psi_Q \vdash M \leftrightarrow K}{\Psi \triangleright P \mid Q \xrightarrow{\tau} (\nu \tilde{a})(P' \mid Q')} \tilde{a} \# Q \\
\\
\text{PAR} \frac{\Psi_Q \otimes \Psi \triangleright P \xrightarrow{\alpha} P'}{\Psi \triangleright P \mid Q \xrightarrow{\alpha} P' \mid Q} \text{bn}(\alpha) \# Q \\
\\
\text{SCOPE} \frac{\Psi \triangleright P \xrightarrow{\alpha} P'}{\Psi \triangleright (\nu b)P \xrightarrow{\alpha} (\nu b)P'} b \# \alpha, \Psi \\
\\
\text{OPEN} \frac{\Psi \triangleright P \xrightarrow{\overline{M}(\nu \tilde{a})N} P' \quad b \# \tilde{a}, \Psi, M}{\Psi \triangleright (\nu b)P \xrightarrow{\overline{M}(\nu \tilde{a} \cup \{b\})N} P'} b \in \mathfrak{n}(N) \quad \text{REP} \frac{\Psi \triangleright P \mid !P \xrightarrow{\alpha} P'}{\Psi \triangleright !P \xrightarrow{\alpha} P'}
\end{array}$$

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

and the entailment relation is extended to  $\vdash \subseteq \mathbf{A} \times (\mathbf{C} \uplus \mathbf{CI})$ , 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 **CI** 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:*

**run**  $M$  *Invoke an agent for which  $M$  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*

$$\text{INVOCATION} \frac{\Psi \vdash M \Leftarrow P \quad \Psi \triangleright P \xrightarrow{\alpha} P'}{\Psi \triangleright \mathbf{run} M \xrightarrow{\alpha} P'}$$

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,

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

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**  $\dots$ , 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  $P$ :

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

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.

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

$$(\nu b)\bar{a}M . (Q \mid (\{M \Leftarrow P\})) \xrightarrow{\alpha} Q \mid (\{M \Leftarrow P\})$$

If  $b \in n(P)$  then also  $b \in n(M)$ , and hence  $b$  is extruded whenever  $M$  is sent, i.e.  $\alpha = \bar{a}(\nu b)M$ . 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:

$$\mathbf{1} \triangleright \begin{array}{c} (\nu b)(Q \mid (\{M_b \Leftarrow \alpha . P\})) \quad \mid \quad (\nu c)(\mathbf{run} M_b \mid R) \\ (\nu b)(Q \mid (\{M_b \Leftarrow \alpha . P\})) \quad \mid \quad (\nu c)(P \mid R) \end{array} \xrightarrow{\alpha}$$

A derivation of this transition uses the INVOCATION rule

$$\frac{\{M_b \Leftarrow \alpha . P\} \vdash M_b \Leftarrow \alpha . P \quad \{M_b \Leftarrow \alpha . P\} \triangleright \alpha . P \xrightarrow{\alpha} P}{\{M_b \Leftarrow \alpha . P\} \triangleright \mathbf{run} M_b \xrightarrow{\alpha} P}$$

Through PAR and SCOPE we get

$$\{M_b \leftarrow \alpha . P\} \triangleright (\nu c)(\mathbf{run} M_b \mid R) \xrightarrow{\alpha} (\nu c)(P \mid R)$$

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

$$\mathbf{1} \triangleright (\{M_b \leftarrow \alpha . P\}) \mid (\nu c)(\mathbf{run} M_b \mid R) \xrightarrow{\alpha} (\{M_b \leftarrow \alpha . P\}) \mid (\nu c)(P \mid R)$$

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

**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**  $\varphi_1 : P_1 \square \cdots \square \varphi_n : P_n$  we write (choosing  $a\#P_i, \varphi_i$ )

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

**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

$$\text{FIX} \frac{P[X := \mathbf{fix} X . P] \xrightarrow{\alpha} P'}{\mathbf{fix} X . P \xrightarrow{\alpha} P'}$$

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  $\mathfrak{n}(P) \cup \{a\}$  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  $\mathbf{fix} X . X \mid P$ , 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.

### 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. Let there be a binary operator on terms  $\bullet\langle\bullet\rangle$ , in other words, if  $M$  and  $N$  are terms then so is  $M\langle N\rangle$ . Let the assertions contain finite sets of *parametrized clauses* of the form  $M(\lambda\tilde{x})N \Leftarrow P$ , with  $\tilde{x} \subseteq \text{n}(N)$  binding in  $N$  and  $P$ . Let the corresponding definition of entailment satisfy

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

for all  $\tilde{L}$  of the same length as  $\tilde{x}$ .

With parametrized 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 parametrized clauses

$$\begin{aligned} \text{STACK}(\lambda x)x &\Leftarrow \underline{\text{Push}}(\lambda y)y . \mathbf{run} \text{STACK}\langle \text{cons}(y, x) \rangle \\ \text{STACK}(\lambda xy)\text{cons}(x, y) &\Leftarrow \overline{\text{Pop}}x . \mathbf{run} \text{STACK}\langle y \rangle \end{aligned}$$

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. `Push` and `Pop` must have empty support — otherwise, their support 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, `cons`( $M, N$ ) is a term representing the usual list constructor.

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

$$\Psi \vdash \text{STACK}\langle \text{nil} \rangle \Leftarrow \underline{\text{Push}}(\lambda y)y . \mathbf{run} \text{STACK}\langle \text{cons}(y, \text{nil}) \rangle$$

and thus

$$\Psi \triangleright \mathbf{run} \text{STACK}\langle \text{nil} \rangle \xrightarrow{\underline{\text{Push}}M} \mathbf{run} \text{STACK}\langle \text{cons}(M, \text{nil}) \rangle$$

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

$$\Psi \triangleright \mathbf{run} \text{STACK}\langle \text{cons}(M, \text{nil}) \rangle \xrightarrow{\underline{\text{Push}}M'} \mathbf{run} \text{STACK}\langle \text{cons}(M', \text{cons}(M, \text{nil})) \rangle$$

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

$$\Psi \triangleright \mathbf{run} \text{STACK}\langle \text{cons}(M, \text{nil}) \rangle \xrightarrow{\overline{\text{Pop}}M} \mathbf{run} \text{STACK}\langle \text{nil} \rangle$$

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  $\Psi$ . 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  $!\bar{a}\text{STACK}.\mathbf{0}$ . Each recipient will get its own stack, which will develop independently of other copies. As formulated here all stacks will use the same channels *In* and *Out*; private channels can be achieved by including their names in the formal parameters of the clauses:

$$\begin{aligned} &\text{STACK}(\lambda i, o, x)i, o, x \leftarrow \underline{i}(\lambda y)y . \mathbf{run} \text{STACK}\langle i, o, \text{cons}(y, x) \rangle \\ &\text{STACK}(\lambda i, o, x, y)i, o, \text{cons}(x, y) \leftarrow \bar{o}x . \mathbf{run} \text{STACK}\langle i, o, y \rangle \end{aligned}$$

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.

$$\text{STACKSTART} \leftarrow \bar{c}\langle \text{Push}, \text{Pop} \rangle . \mathbf{run} \text{STACK}\langle (\text{Push}, \text{Pop}, \text{nil}) \rangle$$

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

$$(\nu \tilde{b})!\bar{a}\text{STACKSTART} . \mathbf{0}$$

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

$$!(\nu \tilde{b})\bar{a}\text{STACKSTART} . \mathbf{0}$$

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 parametrized 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 parametrized 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  $\mathcal{C}$ .

Parametrized clauses use a binary function on terms  $\bullet\langle\bullet\rangle : \mathbf{T} \times \mathbf{T} \rightarrow \mathbf{T}$ . 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 12** (Canonical higher-order psi-calculi). *Let a psi-calculus  $\mathcal{C}$  be defined by the parameters  $\mathbf{T}, \mathbf{C}, \mathbf{A}, \leftrightarrow, \otimes, \mathbf{1}, \vdash$ . Let  $\mathbf{S}$  be the set of finite sets of parametrized clauses as defined above. The canonical higher-order psi-calculus  $\mathcal{H}(\mathcal{C})$  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*

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

For substitution, assuming  $\tilde{x} \# \tilde{y}, \tilde{L}$  we define

$$\begin{aligned}
(M(\lambda\tilde{x})N \Leftarrow P)[\tilde{y} := \tilde{L}] &\text{ to be } M[\tilde{y} := \tilde{L}](\lambda\tilde{x})N[\tilde{y} := \tilde{L}] \Leftarrow P[\tilde{y} := \tilde{L}] \\
\text{and } (\Psi, S)[\tilde{x} := \tilde{L}] &\text{ to be } (\Psi[\tilde{x} := \tilde{L}], \{X[\tilde{x} := \tilde{L}] \mid X \in S\}).
\end{aligned}$$

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 not yield a useable calculus, since in any clause  $a \Leftarrow P$  we require  $n(a) \supseteq n(P)$ . We therefore begin by extending the terms to (nested) tuples, yielding the psi-calculus **Tup**:

**Definition 13** (The psi-calculus **Tup**).

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

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  $(\mathbf{1}, S) \triangleright P[\tilde{x} := \tilde{L}] \xrightarrow{\alpha} P'$ . We can then use  $M$  to invoke  $P$  with parameters  $\tilde{L}$  as follows:

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

**Theorem 14.** *For all  $\mathcal{C}$  and  $\bullet\langle\bullet\rangle$ ,  $\mathcal{H}(\mathcal{C})$  is a higher-order psi-calculus.*

The proof is currently being implemented in Isabelle, where the challenge is 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. Finally we suggest an alternative definition of bisimulation for higher-order communication.

### 4.1 Bisimulation

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

**Definition 15** (Bisimulation). *A strong bisimulation  $\mathcal{R}$  is a ternary relation between 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  $\alpha, P'$  such that  $\text{bn}(\alpha) \# \Psi, Q$  there exists a  $Q'$  such that

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

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

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 expected compositionality properties of strong bisimilarity are:

**Theorem 16.** *For all  $\Psi$ :*

1.  $P \dot{\sim}_{\Psi} Q \implies P \mid R \dot{\sim}_{\Psi} Q \mid R.$
2.  $P \dot{\sim}_{\Psi} Q \implies (\nu a)P \dot{\sim}_{\Psi} (\nu a)Q \quad \text{if } a \# \Psi.$
3.  $P \dot{\sim}_{\Psi} Q \implies !P \dot{\sim}_{\Psi} !Q$
4.  $\forall i. P_i \dot{\sim}_{\Psi} Q_i \implies \text{case } \tilde{\varphi} : \tilde{P} \dot{\sim}_{\Psi} \text{case } \tilde{\varphi} : \tilde{Q}$
5.  $P \dot{\sim}_{\Psi} Q \implies \overline{MN}.P \dot{\sim}_{\Psi} \overline{MN}.Q.$
6.  $(\forall \tilde{L}. P[\tilde{a} := \tilde{L}] \dot{\sim}_{\Psi} Q[\tilde{a} := \tilde{L}]) \implies$   
 $\underline{M}(\lambda \tilde{a})N.P \dot{\sim}_{\Psi} \underline{M}(\lambda \tilde{a})N.Q \quad \text{if } \tilde{a} \# \Psi.$

We say that a relation on agents is a *congruence* if it is preserved by all operators, i.e. as in Theorem 16 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 17.**  $P \sim_{\Psi} Q$  means that for all sequences  $\sigma$  of substitutions it holds that  $P\sigma \sim_{\Psi} Q\sigma$ , and we write  $P \sim Q$  for  $P \sim_{\mathbf{1}} Q$ .

**Theorem 18.**  $\sim_{\Psi}$  is a congruence for all  $\Psi$ .

The usual structural laws hold for strong congruence:

**Theorem 19.**  $\sim$  satisfies the following structural laws:

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

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

**Theorem 20.** All results on the algebraic properties of weak bisimulation as defined and presented in [BJPV10] also hold in higher-order psi-calculi.

The theorems above have all been verified in Isabelle. Although the proofs are quite large (tens of thousands of lines) it turned out that they could be completed in a few days. We just added invocation to our existing proofs for psi-calculi and re-ran almost all without any hitches: most of them gained one new inductive step corresponding to invocation, but in the majority of cases it could be discharged easily. This is a strong argument in favour of proof mechanization: although the initial effort may be large, subsequent updates and modifications need not be very costly. With only manual proofs, the procedure to go through them would be more daunting and error-prone.

## 4.2 Representing operators

In canonical higher-order calculi  $\mathcal{H}(\mathcal{C})$  Sum and Replication can be derived operators, and the  $n$ -ary Case operator can be derived from a unary If. Formally we need to assume that  $\mathbf{T}$  contains a term  $()$  with  $\mathfrak{n}() = \emptyset$  such that  $M[\tilde{x} := \tilde{L}] = ()$  iff  $M = ()$ . In assertions we then write  $M \Leftarrow P$  for the parameterized clause  $M(\lambda \varepsilon)() \Leftarrow P$ , and **run**  $M$  for the invocation **run**  $M\langle () \rangle$ . We also assume that for all substitution sequences  $\sigma$ ,  $M\langle () \rangle\sigma = K\langle N \rangle$  iff  $K = M\sigma$  and  $N = ()$ .

**Proposition 21.** In a canonical calculus, where we in the following assume that  $a \# \tilde{P}, \tilde{\varphi}$ , and that  $\mathfrak{n}(M_a\sigma) \supseteq \mathfrak{n}(a, \tilde{P}\sigma)$  when  $a \# \sigma$ :

1.  $!P_1 \sim (\nu a)(\mathbf{run} M_a \mid (\{\{M_a \Leftarrow P_1 \mid \mathbf{run} M_a\}\}))$
2.  $P_1 + P_2 \sim (\nu a)(\mathbf{run} M_a \mid (\{\{M_a \Leftarrow P_1, M_a \Leftarrow P_2\}\}))$
3. **case**  $\varphi_1 : P_1 \square \cdots \square \varphi_n : P_n$   
 $\sim (\nu a)(\mathbf{run} M_a \mid (\{\{M_a \Leftarrow \mathbf{if} \varphi_1 \mathbf{then} P_1, \cdots M_a \Leftarrow \mathbf{if} \varphi_n \mathbf{then} P_n\}\}))$

As an example consider the representation of Replication. Consider a transition from  $(\nu a)(\mathbf{run} M_a \mid (\{\{M_a \Leftarrow P \mid \mathbf{run} M_a\}\}))$ . It can only be by invocation where  $P \mid \mathbf{run} M_a$  has a transition leading to  $P' \mid \mathbf{run} M_a$  and results in

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

Using Theorem 19 and  $a \# P'$  we rewrite this as

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

In other words, the transition precisely corresponds to the transition of  $!P$  derived from  $P \mid !P$ . A fully formal proof is to establish a bisimulation up to  $\sim$ . The proofs are currently being conducted in Isabelle.

Note that Proposition 21 holds for canonical calculi but not for psi-calculi in general. At the very least, we need a kind of interference freedom of the logic, for example for 21.1:

$$\forall \Psi : \Psi \otimes (\nu a)(\{M_a \Leftarrow P \mid \mathbf{run} M_a\}) \simeq \Psi, \quad \text{and}$$

$$\text{if } a \# \Psi \text{ then } \Psi \otimes \{M_a \Leftarrow P \mid \mathbf{run} M_a\} \vdash M_a \Leftarrow R \text{ iff } R = P \mid \mathbf{run} M_a,$$

otherwise the representations will not be statically equivalent. These requirements are completely natural since  $\Psi$  here does not contribute knowledge about terms containing the name  $a$ . Interference freedom holds in canonical calculi, but it does not follow from the very general requisites of psi-calculi. Without interference freedom it is clear that the representation is not correct since the agents are not statically equivalent. We are currently investigating the exact minimal requirements.

### 4.3 Higher-order bisimulation

Higher-order communication is treated by bisimulation just like any other communication: An action  $\bar{a}P$  must be simulated by an identical action. Therefore, if  $P \neq P'$  we will have  $\bar{a}P.\mathbf{0} \not\sim \bar{a}P'.\mathbf{0}$ , even if  $P \sim P'$ , which somewhat spoils the claim for  $\sim$  to be a congruence in the ordinary sense of the word. This is unavoidable, considering that the recipient may use  $P$  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  $a(x).\mathbf{if} x = Q \mathbf{then} \dots$ , or to subject it to pattern matching in order to find its outermost operator; this corresponds to inspecting the process code. Unfortunately, transmitting invocation possibilities also opens the code for inspection in a similar way. Consider the two processes

$$\begin{aligned} Q &= (\nu b)\bar{a}M_b.(\{\{M_b \Leftarrow P\}\}) \\ Q' &= (\nu b)\bar{a}M_b.(\{\{M_b \Leftarrow P'\}\}) \end{aligned}$$

Suppose  $P \sim P'$ . There is a strong argument for regarding  $Q$  and  $Q'$  as equivalent, since the only thing a recipient can do with the handle  $M_b$  is to invoke a process, and if  $P \sim P'$  it should not matter which of them is invoked. But with the current definitions,  $Q \not\sim Q'$ . 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:  $\{M_b \leftarrow P\} \not\approx \{M_b \leftarrow P'\}$ , since they do not entail exactly the same clauses.

This suggests that a slightly relaxed version of bisimilarity is more appropriate. In contrast to standard higher order bisimilarities, that require objects of higher-order output actions to be bisimilar, we instead weaken static equivalence to require bisimilar (rather than identical) entailed clauses.

**Definition 22** (HO-Bisimulation). *A strong HO-bisimulation  $\mathcal{R}$  is a ternary relation between assertions and pairs of agents such that  $\mathcal{R}(\Psi, P, Q)$  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)  $\forall (M \leftarrow P') \in \mathbf{Cl}. \quad \Psi \otimes \mathcal{F}(P) \vdash M \leftarrow P' \Rightarrow$   
 $\exists Q'. \Psi \otimes \mathcal{F}(Q) \vdash M \leftarrow Q' \wedge \mathcal{R}(\Psi, P', Q')$

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  $\alpha, P'$  such that  $\text{bn}(\alpha) \# \Psi, Q$  there exists a  $Q'$  such that

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

We define  $P \sim_{\Psi}^{\text{HO}} Q$  to mean that there exists a strong bisimulation  $\mathcal{R}$  such that  $\mathcal{R}(\Psi, P, Q)$ , and write  $\sim^{\text{HO}}$  for  $\sim_{\mathbf{1}}^{\text{HO}}$ .

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

**Theorem 23.**  $P \sim Q \Rightarrow P \sim^{\text{HO}} Q$

The proof is that any bisimulation is also a HO-bisimulation: take  $Q' = P'$  in Clause 1(b).

**Corollary 24.**  $\sim^{\text{HO}}$  satisfies all structural laws of Theorem 19.

**Theorem 25.** In a canonical higher-order psi-calculus,

$$P \sim^{\text{HO}} Q \Rightarrow (\{M \leftarrow P\}) \sim^{\text{HO}} (\{M \leftarrow Q\})$$

The proof boils down to showing that

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

is a HO-bisimulation. The only nontrivial part is static equivalence. This does not hold for higher-order psi-calculi in general. It would be interesting to determine the necessary requirements.

Finally, the compositionality of  $\dot{\sim}^{\text{ho}}$  is under investigation:

**Conjecture 26.** *For all  $\Psi$ :*

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

The proofs are currently being conducted in Isabelle.

## 5 Conclusion

We have defined higher-order psi-calculi in a smooth extension from ordinary psi-calculi. This means that we can re-use much of the mechanized 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 are currently integrating 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. For the main results like Conjecture 26 it is not worthwhile 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. We believe that when the right framework is in place these proofs can be mechanized with a reasonable effort.

There are several interesting avenues to explore. One obvious is higher-order weak bisimulation and congruence, where it seems that the new requirement on static equivalence integrates nicely with our existing definitions of weak bisimulation. Another is to explore the exact requirements for results such as Proposition 21 to hold, perhaps through a type system on the assertions. 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.

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  $a(x) \cdot \mathbf{run} \pi_1(x)$  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  $\pi_1(\langle M_1, M_2 \rangle) \rightarrow M_1$ , 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  $a(\lambda xy)\langle x, y \rangle \cdot \mathbf{run} x$ . 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 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

- [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.
- [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 Hirschhoff, 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.
- [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.
- [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.