Psi-calculi: Mobile processes, nominal data, and logic

Jesper Bengtson Magnus Johansson Joachim Parrow Björn Victor

Dept. of Information Technology, Uppsala University, Sweden

Abstract

A psi-calculus is an extension of the pi-calculus with nominal data types for data structures and for logical assertions representing facts about data. These can be transmitted between processes and their names can be statically scoped using the standard pi-calculus mechanism to allow for scope migrations.

Other proposed extensions of the pi-calculus can be formulated as psi-calculi; examples include the applied pi-calculus, the spi-calculus, the fusion calculus, the concurrent constraint pi-calculus, and calculi with polyadic communication channels or pattern matching. Psi-calculi can be even more general, for example by allowing structured channels, higher-order formalisms such as the lambda calculus for data structures, and a predicate logic for assertions.

Our labelled operational semantics and definition of bisimulation is straightforward, without a structural congruence. We establish minimal requirements on the nominal data and logic in order to prove general algebraic properties of psi-calculi. The proofs have been checked in the interactive proof checker Isabelle.

We are the first to formulate a truly compositional labelled operational semantics for calculi of this calibre. Expressiveness and therefore modelling convenience significantly exceeds that of other formalisms, while the purity of the semantics is on par with the original pi-calculus.

1 Introduction

The pi-calculus [15] has a multitude of extensions where higher-level data structures and operations on them are given as primitive. To mention only two there are the spi-calculus by Abadi and Gordon [2] focusing on cryptographic primitives, and the applied pi-calculus of Abadi and Fournet [1] where agents can introduce statically scoped aliases of names for data, used e.g. to express how knowledge of an encryption is restricted. It is also parametrised by an arbitrary signature for expressing data and an equation system for expressing data equalities. The impact of these enriched calculi is considerable with hundreds of papers applying or developing the formalisms. As Abadi and Fournet rightly observe there is a tradeoff between "purity", meaning the simplicity and elegance of the original pi-calculus, and modelling convenience. Expressing complicated schemes in the original pi-calculus can simply become too gruesome and error prone.

But the modelling convenience of many high-level primitives comes at a price. The theory of the formalism may instead become gruesome and error prone, and it can be difficult to assess the effects of modifications to it. For example, the semantics of the applied pi-calculus is defined using two levels of processes (pure and extended), an inductively defined reduction relation, an algebraically defined structural congruence, and (in order to achieve compositionality) a notion of a barb and an explicit quantification over contexts. The authors therefore propose a more tractable semantics based on inductively defined labelled transitions, but as we show in Section 3.1 below it turns out to be non-compositional and in fact does not agree with the reduction semantics. For another example, the labelled semantics of the concurrent constraint pi-calculus by Buscemi and Montanari [7] also turns out to be non-compositional, as we show in Section 3.2. The fact that such mistakes can go unnoticed for years indicates the complexity of the proofs.

Our contribution in this paper is to define psi-calculi: a framework where a range of calculi can be formulated with a lean and symmetric semantics, and where proofs can be conducted using straightforward induction without resorting to a structural congruence or explicit quantification of contexts. We claim to be the first to formulate such truly compositional labelled operational semantics for calculi of this calibre. Psi-calculi accommodate not only the examples mentioned above, but also extensions such as the pi-calculus with polyadic synchronisation [8], fusion [19], and concurrent constraints [6]. The main idea is that a psi-calculus is obtained by extending the basic untyped pi-calculus with the following parameters: (1) a set of data terms, which can function as both communication channels and communicated objects, (2) a set of conditions, for use in conditional constructs such as **if** statements, (3) a set of assertions, used to express e.g. constraints or aliases. One of our main results is to identify minimal requirements on these parameters. These turn out to be quite general and natural.

Psi-calculi go beyond our previous work on extended pi-calculi [13] since we admit arbitrary assertions (and not only declarations of aliases), and arbitrary conditions (and not only equality tests). Also, we base our exposition on nominal data types and these accommodate e.g. alpha-equivalence classes of terms with binders. For example, we can use a higher-order logic for assertions and conditions, and higher-order formalisms such as the lambda calculus for data terms and channels. Last but not least the formalisation is leaner and more symmetric. Thus we get the best of two worlds: expressiveness and therefore modelling convenience significantly exceeds that of the applied pi-calculus, while the "purity" of the semantics is on par with the original pi-calculus.

The straightforward definitions make our proofs suitable for checking in a theorem prover. We have implemented our framework in Isabelle [16] using its nominal data type package [18], and proved all results in Section 4 [3]. This gives us absolute certainty of general results for a large class of calculi — at least to the point of the current state of the art for machine checked proofs.

In the next section we give the basic definitions of the syntax and semantics of psi-calculi. In Section 3 we relate to other work and demonstrate the expressiveness by showing how a variety of calculi can be formulated. In Section 4 we introduce a notion of bisimilarity and establish the expected algebraic results about it. Finally Section 5 concludes with ideas for further work.

2 Definitions

2.1 Nominal data types

We base psi-calculi on nominal data types. A reader unfamiliar with these needs not fear: we shall recapitulate what little background is needed and be generous with examples. A traditional data type can be built from a signature of constant symbols, functions symbols, etc. A nominal data type is more general, for example it can also contain binders and identify alpha-variants of terms. Formally a nominal data type is not required to be built in any particular way; the only requirements are related to the treatment of the atomic symbols called names as explained below.

As usual we assume a countably infinite set of atomic *names* $\mathcal{N}$ ranged over by $a, b, \dots, x, y, z$. Intuitively, names will represent the symbols that can be statically scoped, and also represent symbols acting as variables in the sense that they can be subjected to substitution. A typed calculus would distinguish names of different kinds but our account will be untyped. A typing may certainly contribute to clarity of expressions but it is not necessary for our results.

A *nominal set* [17, 11] 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 $a \notin \mathrm{n}(X)$. In a traditional data type we will have $a \# X$ if a does not occur syntactically in X. In for example the lambda calculus where alpha-equivalent terms are identified (i.e. the elements are alpha-equivalence classes of terms) the support corresponds to the free names. If A is a set of names we write $A \# X$ to mean $\forall a \in A \,.\, a \# X$.

We require all elements to have finite support, i.e., $\mathrm{n}(X)$ is finite for all X. It follows that for any X there are infinitely many a such that $a \# X$. Some elements will have empty support, a prime example is the identity function in the lambda calculus, or a term of a traditional data type not containing any names. Such elements satisfy *equivariance*, $(a\ b) \cdot X = X$ for all a, b. 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.

A *nominal data type* is just a nominal set together with a set of functions on it. In particular we require a substitution function, which intuitively substitutes elements for names. If X is an element of a data type, $\tilde{a}$ is a sequence of names without duplicates and $\tilde{Y}$ is an equally long sequence of elements, the *substitution* $X[\tilde{a} := \tilde{Y}]$ is an element of the same data type as X. In a traditional data type substitution can be thought of as replacing all occurrences of names $\tilde{a}$ by $\tilde{Y}$. In a calculus with binders it can be thought of as replacing the free names, alpha converting any binders to avoid capture. Formally, we define substitution as any equivariant function satisfying the substitution laws of [3], e.g. $z \#(X, \tilde{N}) \Rightarrow z \# X[\tilde{a} := \tilde{N}]$.

2.2 Agents

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:*

$\mathbf{T}$	*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:

$\dot{\leftrightarrow}: \mathbf{T} \times \mathbf{T} \rightarrow \mathbf{C}$	*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*

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 Ψ and Ψ' are assertions then so is $\Psi \otimes \Psi'$. Also we write $\Psi \vdash \varphi$, "Ψ entails φ", for $(\Psi, \varphi) \in \vdash$.

The data terms are used to represent all kinds of data, including communication channels. Intuitively, two agents can communicate if one sends and the other receives along the same channel. This is why we require a condition $M \dot{\leftrightarrow} N$ to say that M and N represent the same communication channel. For example, in the pi-calculus $\dot{\leftrightarrow}$ would be just identity.

The assertions will be used to declare information necessary to resolve the conditions. Assertions can be contained in agents and represent constraints; they can contain names and thereby be syntactically scoped and represent information known only to the agents within that scope. The operator $\otimes$ on assertions will, intuitively, be used to represent conjunction of the information in the assertions. The assertion $\mathbf{1}$ can be used to represent the least possible information in an assertion.

The intuition of entailment is that $\Psi \vdash \varphi$ means that given the information in Ψ, it is possible to infer φ. We say that two assertions are equivalent if they entail the same conditions:

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

We can now formulate our requisites on valid psi-calculus parameters:

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

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

Our requisites on a psi-calculus is that the channel equivalence is symmetric and transitive, that $\otimes$ is compositional, and that the equivalence classes of assertions form an abelian monoid. These requisites turn out to be strictly minimal for our results in Section 4 to hold. Note that channel equivalence is not required to be reflexive. Thus it is possible to have data terms that are not channel equivalent to anything at all, meaning that they cannot be used as channels.

In the following $\tilde{a}$ means a finite (possibly empty) 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 terms, conditions, assertions etc. in the same way.

A *frame* can intuitively be thought of as an assertion with local names:

Definition 4 (Frame)**.** *A* frame *F is a pair $\langle B_F, \Psi_F \rangle$ where B_F is a sequence of names that bind into the assertion Ψ_F. We use F, G to range over frames.*

Name swapping on a frame just distributes to its two components. We identify alpha equivalent frames, so $\mathrm{n}(F) = \mathrm{n}(\Psi_F) - \mathrm{n}(B_F)$. We overload $\mathbf{1}$ to also mean the least informative frame $\langle \epsilon, \mathbf{1} \rangle$ and $\otimes$ to mean composition on frames defined by $\langle B_1, \Psi_1 \rangle \otimes \langle B_2, \Psi_2 \rangle = \langle B_1 B_2, \Psi_1 \otimes \Psi_2 \rangle$ where $B_1 \# B_2, \Psi_2$ and vice versa. We also write $\Psi \otimes F$ to mean $\langle \epsilon, \Psi \rangle \otimes F$, and $(\nu b)F$ to mean $\langle b B_F, \Psi_F \rangle$.

Definition 5 (Equivalence of frames)**.** *We define $F \vdash \varphi$ to mean that there exist B_F and Ψ_F such that $F = \langle B_F, \Psi_F \rangle$, $B_F \# \varphi$, and $\Psi_F \vdash \varphi$. We also define $F \simeq G$ to mean that for all φ it holds that $F \vdash \varphi$ iff $G \vdash \varphi$.*

Intuitively a condition is entailed by a frame if it is entailed by the assertion and does not contain any names bound by the frame. Two frames are equivalent if they entail the same conditions.

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

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

In the Input $\underline{M}(\lambda \tilde{x})N.P$ we require that $\tilde{x} \subseteq \mathrm{n}(N)$ is a sequence without duplicates, and any name in $\tilde{x}$ 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.*

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 there is a pattern matching where the pattern $(\lambda\widetilde{x})N$ can match any term obtained by instantiating $\widetilde{x}$. In the traditional pi-calculus terms are just names and its input construct $a(x).P$ can be represented as $\underline{a}(\lambda x)x.P$. The **case** construct as expected works by behaving as one of the P_i for which the corresponding φ_i is true. So it embodies both an **if** (if there is only one branch) and a nondeterministic choice (if the conditions are overlapping).

Some notational conventions: We define the agent $\mathbf{0}$ as $(\!|\mathbf{1}|\!)$. The construct **case** $\varphi_1 : P_1 [] \cdots [] \varphi_n : P_n$ is sometimes abbreviated as **case** $\widetilde{\varphi} : \widetilde{P}$, or if $n = 1$ as **if** φ_1 **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 **case** $\top : P [] \top : Q$. Input subjects are underlined to facilitate parsing of complicated expressions; in simple cases we often omit the underline.

Formally, we define name swapping on agents by distributing it over all constructors, and substitution on agents by distributing it and avoiding captures by binders through alpha-conversion in the usual way. We identify alpha-equivalent agents; in that way we get a nominal data type of agents where the support n(P) of P is the union of the supports of the components of P, removing the names bound by λ and ν. This corresponds to the names with a free occurrence in P.

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

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

In the following we often write B_P and Ψ_P for the two elements of $\mathcal{F}(P)$. The intuition is that a frame collects information from the top-level assertions in an agent. $\mathbf{1}$ is the empty frame (corresponding to an agent that has no top-level assertions), and $\mathcal{F}(P)\otimes\mathcal{F}(Q)$ is the combination of the frames from two parallel agents. An agent where all assertions are guarded thus has the frame $\mathbf{1}$.

2.3 Operational semantics

The actions α 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 τ. The operational semantics will consist 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*
$\underline{M}\ N$	*Input*
τ	*Silent*

For actions we refer to M as the *subject* and N as the *object*. We define bn$(\overline{M}\ (\nu\tilde{a})N) = \tilde{a}$, and bn$(\alpha) = \emptyset$ if α is an input or τ.

Definition 9 (Transitions)**.** *A* transition *is of the kind $\Psi \rhd P \xrightarrow{\alpha} P'$, meaning that when the environment contains the assertion Ψ the agent P can do an α to become P'. The transitions are defined inductively in Table 1.*

The environmental assertions $\Psi \rhd \cdots$ in Table 1 expresses the effect that the environment has on the agent, by enabling conditions in CASE, by giving rise to action subjects in IN and OUT and by enabling interactions in COM. Thus Ψ never changes between hypothesis and conclusion except for the parallel operator, where an agent is part of the environment for another agent. In a derivation tree for a transition, the assertion will therefore increase towards the leafs by application of 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.

In comparison to the applied pi-calculus and the concurrent constraint pi calculus one main novelty is the inclusion of these environmental assertions. They are necessary to make our semantics compositional, i.e., the effect of the environment on an agent is wholly captured by the semantics. In contrast, the labelled transitions of the applied and the concurrent constraint pi calculi must rely on an auxiliary structural congruence, containing axioms such as scope extension $(\nu a)(P|Q) \equiv (\nu a)P|Q$ if $a\#Q$. With our semantics such laws are derived rather than postulated. The advantage of our approach is that proofs of metatheoretical results such as compositionality are much simpler since there is only the one inductive definition of transitions.

$$\textsc{In}\ \frac{\Psi \vdash M \dot{\leftrightarrow} K}{\Psi \rhd \underline{M}(\lambda\widetilde{y})N.P \xrightarrow{\underline{K}\,N[\widetilde{y}:=\widetilde{L}]} P[\widetilde{y}:=\widetilde{L}]} \qquad \textsc{Out}\ \frac{\Psi \vdash M \dot{\leftrightarrow} K}{\Psi \rhd \overline{M}\,N.P \xrightarrow{\overline{K}\,N} P} \qquad \textsc{Case}\ \frac{\Psi \rhd P_i \xrightarrow{\alpha} P' \quad \Psi \vdash \varphi_i}{\Psi \rhd \textbf{case}\ \widetilde{\varphi} : \widetilde{P} \xrightarrow{\alpha} P'}$$

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

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

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

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

3 Expressiveness and related calculi

In this section we explore the expressiveness of psi-calculi, mainly in comparison to other process calculi. To begin with a simple example, the pi-calculus [15] can be represented as a psi-calculus where the only data terms are names, the only assertion is $\mathbf{1}$, and the conditions are equality tests on names. Formally:

$$\begin{array}{ll}
\mathbf{T} & = \mathcal{N} \\
\mathbf{C} & = \{a = b.\ a, b \in \mathbf{T}\} \cup \{a \dot{\leftrightarrow} b.\ a, b \in \mathbf{T}\} \\
\mathbf{A} & = \{\mathbf{1}\} \\
\otimes & = \lambda\Psi_1, \Psi_2.\ \mathbf{1} \\
\vdash & = \{(\mathbf{1}, a = a).\ a \in \mathcal{N}\} \cup \{(\mathbf{1}, a \dot{\leftrightarrow} a).\ a \in \mathcal{N}\}
\end{array}$$

Here we can let $\top$ be $a = a$ and can thus represent pi-calculus sum through **case**. We obtain the polyadic pi-calculus by adding the tupling symbols t_n for tuples of arity n to $\mathbf{T}$., i.e. $\mathbf{T} = \{\mathrm{t}_n(a_1, \ldots, a_n).\ a_1, \ldots, a_n \in \mathcal{N}\}$ (where we abbreviate $\mathrm{t}_1(a)$ as a). The polyadic output is to simply output the corresponding tuple of object names, and the polyadic input $a(b_1, \ldots, b_n).P$ is represented by a pattern matching $\underline{a}(\lambda b_1, \ldots, b_n)\mathrm{t}_n(b_1, \ldots, b_n).P$. Strictly speaking this allows tuples also in subject position in agents, but such prefixes will not give rise to any transition since $M \dot{\leftrightarrow} M$ is only entailed when M is a name. In examples below, we write polyadic objects assuming such a representation.

3.1 Calculi for cryptography

Psi-calculi can express a variety of cryptographic operations on data. The main idea is that assertions define relations between ciphertext and plaintext. For example, let the assertion "$C = \mathsf{enc}(M, k)$" mean that encrypting the message M with the key k results in the ciphertext C, and let "$M = \mathsf{dec}(C, k)$" mean that decrypting C with key k yields M. Entailment contains equations relating encryption and decryption such as $\forall M, k.\,\mathsf{dec}(\mathsf{enc}(M, k), k) = M$. The point is that a secure key can be represented by a bound name: it is unguessable outside its scope. An example agent $\overline{a}C\,.\,(\nu k)((\!|C = \mathsf{enc}(M, k)|\!) \mid P)$ outputs a term C and asserts that it is the encryption of M using the bound k as key, without opening the scope of k. Therefore an agent receiving C can resolve the condition $\mathsf{dec}(C, k) = M$ only after receiving this k in a communication. Technically this is because of the freshness conditions in the $\textsc{Par}$ rule in Table 1 where B_Q is assumed fresh for P: this means that to apply the rule, P cannot use any name bound in the frame of Q.

This closely resembles the situation in applied pi [1]. By contrast, in the spi-calculus [2] encrypted messages such as $\mathsf{enc}(M, k)$ are transmitted directly. Consider an example spi-calculus process

$$P = (\nu k, m)(\overline{a}\langle\mathsf{enc}(m, k)\rangle\,.\,P' \qquad (1)$$

where $P' = b(x)\,.\,\textbf{if}\ x = m\ \textbf{then}\ \overline{c}$. Here P sends a fresh name m encrypted with a fresh key k to the environment, and then receives a value x. Assuming perfect encryption, the environment cannot know m or k, so P' cannot receive m along b, and the output on c will never be possible. However, in the spi-calculus the transition $P \xrightarrow{(\nu k,m)\overline{a}\langle \mathsf{enc}(m,k)\rangle} P'$ opens the scopes of k and m, so here scoping does not correspond to restriction of knowledge. A reasonable equivalence must explicitly keep track of which names are known, leading to several complex bisimulation definitions (see [4] for an overview).

The applied pi-calculus is parametrised by a signature Σ for data terms and an equational theory $\vdash_\Sigma$ over Σ, and more importantly introduces *active substitutions* $\{^M/_x\}$ of data terms for variables. These can be introduced by the inferred structural rule $(\nu x)(\{^M/_x\} \mid A) \equiv A[x := M]$. Applied pi distinguishes between names a, b, c and variables x, y, z (collectively called *atoms*) where only variables can be substituted, and uses a simple type system to distinguish names and variables of channel type from other terms of base type. Only names of channel type can be used as communication channels. Structured data terms cannot be sent directly, instead an *alias* variable such as x must be used, and the term itself does not occur on the transition label. We have $P \equiv Q$ for P above in (1), where

$$Q = (\nu x, k, m)(\{^{\mathsf{enc}(m,k)}/_x\} \mid \overline{a}x\,.\,P') \qquad (2)$$

Here $Q \xrightarrow{\overline{a}\,(\nu x)x} (\nu k, m)(\{^{\mathsf{enc}(m,k)}/_x\} \mid P')$ and only the alias of the encryption (its "value") appears on the label; the scope of k and m is not opened and in this sense they are still confidential to the environment. However, the labelled semantics does not allow sending structured data terms where the scope *should* be opened, such as a tuple of names in the polyadic pi-calculus.

The labelled semantics for applied pi turns out to be non-compositional. Consider the closed (extended) applied pi agents

$$A = (\nu a)(\{^a/_x\} \mid x.b.\mathbf{0}) \qquad B = (\nu a)(\{^a/_x\} \mid \mathbf{0})$$

where we omit the objects of the prefixes. They have the same frame and no transitions, and are thus semantically equivalent. But a context can contain x and can therefore use the active substitution to communicate with A. Formally, let $R = \overline{x}.\mathbf{0}$; we have by scope extension that $A|R \approx (\nu a)(\{^a/_x\} \mid x.b.\mathbf{0} \mid \overline{x}.0) \Downarrow b$, but it is not the case that $B|R \Downarrow b$. Therefore, *no* observational equivalence that is preserved by all contexts and satisfies scope extension can be captured by the labelled semantics. In this, Theorem 1 of [1] is incorrect; the labelled and observational equivalences do in fact not coincide, nor is labelled equivalence a congruence. This is relevant for other papers that use or develop the labelled semantics, e.g. [12, 14, 10, 9].

Possible fixes are to disallow aliases for channel names, to be satisfied with compositionality for closed contexts, or to allow variables in action subjects. The consequences are difficult to assess, and our proposed solution is to instead define a psi-calculus representing the corresponding applied pi-calculus as follows. Assertions are finite sets of active substitutions, $\otimes$ is union, and entailment deduces equality under $\vdash_\Sigma$ and application of all relevant active substitutions.

Requirements on the assertions in applied pi are that they can only contain one active substitution per variable, that the active substitutions are non-circular, that they do not occur under a replication etc. To stay as close to the applied pi-calculus as possible we inherit these restrictions and only consider agents that satisfy them. This allows us to write $M(\Psi^*)$ for the term resulting from the fixpoint of the substitutions in Ψ. We write $v(M)$ for the free variables of M, and Chan for the set of names of channel type.

$$\begin{array}{l}
\mathbf{T} \;= \text{the set of terms defined by } \Sigma\\
\mathbf{A} \;= \mathcal{P}_{\text{fin}}(\{\{^M/_x\} : M \in \mathbf{T}, x \text{ variable}\})\\
\mathbf{C} \;= \{M = N, \neg(M = N), M \leftrightarrow N : M, N \in \mathbf{T}\}\\
\mathbf{1} \;= \emptyset\\
\otimes \;= \cup\\
\Psi \vdash M = N \text{ if } \vdash_\Sigma M(\Psi^*) = N(\Psi^*)\\
\Psi \vdash \neg(M = N) \text{ if } v(M(\Psi^*)) \cup v(N(\Psi^*)) = \emptyset\\
\qquad\qquad\qquad\quad \wedge\neg(\Psi \vdash M = N)\\
\Psi \vdash M \leftrightarrow N \text{ if } \Psi \vdash M = N \wedge \exists c \in \mathsf{Chan} : \Psi \vdash M = c
\end{array}$$

Terms, assertions and conditions are as for applied pi except for the condition $\neg(M = N)$ which is needed to represent the **if** $M = N$ **then** P **else** Q construct of applied pi as **case** $M = N : P \;[]\; \neg(M = N) : Q$ in psi. As in applied pi, the terms compared for inequality need to be ground. Channel equivalence $M \leftrightarrow N$ requires that there is a channel name equal to both M and N.

The resulting psi-calculus differs from the applied pi-calculus in some ways, the most important being that in psi, $\overline{a}M.P \xrightarrow{\overline{a}M} P$ where M is a structured term, corresponds to sending the cleartext of M directly. This is not possible in the applied pi-calculus. In order to transmit M in the applied pi-calculus the structural rule $(\nu x)(\{^M/_x\} \mid A) \equiv A[x := M]$ must be used and an alias x for M be sent. To send an alias in this way in psi it must be introduced explicitly, as in $(\!|\{^M/_x\}|\!) \mid \overline{a}x.P$, and this agent is *not* the same as $\overline{a}M.P$.

Therefore, although the agents P and Q above (in (1) and (2)) are the same in the applied pi-calculus, the psi counterparts of the agents are different. In psi, P in (1) represents an agent that emits the clear text "$\mathtt{enc}(m, k)$". Any agent that receives this will immediately learn both m and k, and any scope of k will be opened in the process. This kind of agent can only indirectly be represented in the applied pi-calculus, by sending the restricted names separately one at a time. In contrast, the psi counterpart of (2) is $Q = (\nu x, k, m)((\!|\{^{\mathsf{enc}(m,k)}/_x\}|\!) \mid \overline{a}x\,.\,P')$ and defines Q to emit an alias for $\mathtt{enc}(m, k)$. As in the applied pi-calculus since k is scoped a recipient will not learn m. If the same recipient later receives k, an alias u for the message m can be constructed as $(\!|\{^{\mathsf{dec}(x,k)}/_{u}\}|\!)$.

Thus in psi, communication objects can range from literal data terms to indirect references, giving the user of the calculus the possibility to choose the appropriate form.

Another difference between the calculi is illustrated by the agent A of the compositionality counterexample: Its counterpart P_A in psi is $(\nu a)((\!|\{^{a}/_{x}\}|\!) \mid x.b.\mathbf{0}) \xrightarrow{x} (\nu a)((\!|\{^{a}/_{x}\}|\!) \mid b.\mathbf{0})$ and is not equivalent to $(\nu a)((\!|\{^{a}/_{x}\}|\!) \mid \mathbf{0})$; indeed also $P_A \mid \overline{x}.\mathbf{0} \xrightarrow{\tau}\xrightarrow{b}$ in our labelled semantics. This example motivates the requisite that $\dot{\leftrightarrow}$ must be symmetric, otherwise psi would suffer the same problem of compositionality.

3.2 Fusion and concurrent constraints

In the *explicit fusion calculus* by Wischik and Gardner, pi-F [19], the names in object position are fused by a communication, as in $a\tilde{b}\,.\,P \mid \overline{a}\tilde{d}\,.\,Q \xrightarrow{\tau} \{\tilde{b} = \tilde{d}\} \mid P \mid Q$ where $\{\tilde{b} = \tilde{d}\}$ (for $\tilde{b}$ and $\tilde{d}$ of equal length) is a *fusion* which allows us to treat each $b_i \in \tilde{b}$ as equivalent to $d_i \in \tilde{d}$. The explicit fusion calculus has a simple formulation as a psi-calculus:

$$\begin{array}{ll}
\mathbf{T} & = \mathcal{N} \\
\mathbf{C} & = \{a = b.\ a, b \in \mathbf{T}\} \cup \{a \dot{\leftrightarrow} b.\ a, b \in \mathbf{T}\} \\
\mathbf{A} & = \{\ \{a_1 = b_1, \ldots, a_n = b_n\}\ .\ a_i \in \mathcal{N}, b_i \in \mathcal{N}\} \\
\otimes & = \cup \\
\mathbf{1} & = \emptyset \\
\vdash & = \{(\Psi, a = b).\ a = b \in \mathrm{EQ}(\Psi)\} \quad \cup \\
 & \quad\ \{(\Psi, a \dot{\leftrightarrow} b).\ \Psi \vdash a = b\}
\end{array}$$

where $\mathrm{EQ}(\Psi)$ is the equivalence closure of Ψ (i.e. transitive, symmetric and reflexive closure). Thus terms are names, assertions are name fusions, and the entailment relation deduces equality between names based on fusion assertions treated as equivalence relations. We can represent the pi-F input $a\tilde{b}\,.\,P$ as $a(\tilde{c})\,.\,((\!|\{\tilde{b} = \tilde{c}\}|\!) \mid P)$ where $\tilde{c}\#a\tilde{b}\,.\,P$. For example, the fusion calculus communication $a\tilde{b}\,.\,P \mid \overline{a}\tilde{d}\,.\,Q \xrightarrow{\tau} \{\tilde{b} = \tilde{d}\} \mid P \mid Q$ is expressed as:

$$\begin{array}{l}
a(\tilde{c})\,.\,((\!|\{\tilde{b} = \tilde{c}\}|\!) \mid P) \mid \overline{a}\tilde{d}\,.\,Q \xrightarrow{\tau} \\
\qquad ((\!|\{\tilde{b} = \tilde{c}\}|\!) \mid P)[\tilde{c} := \tilde{d}] \mid Q \quad = \quad (\!|\{\tilde{b} = \tilde{d}\}|\!) \mid P \mid Q
\end{array}$$

The *concurrent constraint pi-calculus* (CC-Pi) [7] can be seen as a (monadic) pi-F calculus with **ask** and **tell** statements, parametrised by a constraint system in the form of a *named c-semiring*. The semantics is given by a structural congruence and a reduction relation. There is also a labelled operational semantics, but it is in fact not compositional. Consider the CC-Pi agents

$$P = (\nu b, x)(x = b \mid \overline{a}x\,.\,b\,.\,c) \qquad Q = (\nu b, x)(x = b \mid \overline{a}x)$$

(where insignificant objects are omitted). They have the same constraint store and the same transitions in all constraint contexts. However, they do not have the same transitions in all process contexts: a parallel context $R = a(y).\overline{y}$ tells the difference:

$$P \mid R \xrightarrow{\tau}\xrightarrow{\tau} (\nu b)(x = b \mid x = y \mid c) \xrightarrow{c}$$

while $Q \mid R$ of course has no such c transition. Thus Theorem 1 of [7] is incorrect: open bisimilarity is not a congruence.

The labelled semantics of CC-Pi has a curious asymmetry between the rule for prefixes and the rule for communication: in the first case, the constraint store cannot affect the label induced by the prefix, while in the communication case, the constraint store judges whether the subjects should be considered the same, enabling the communication. The psi-calculi have no such asymmetry: the assertions (corresponding to the store) allow the subject to be rewritten in the prefix rules and the subjects in COM are compared using the assertions (see the end of Sec. 3.3 for a discussion). A possible fix for CC-Pi would involve allowing the store to rewrite terms, thus also subjects in prefixes [5].

A psi-calculus corresponding to CC-pi with semiring $\mathcal{C}$ extends the psi-calculus for pi-F as follows:

$$\begin{array}{rl}
\mathbf{T} = \mathbf{A} & = \mathcal{C} \\
\mathbf{C} & = \mathcal{C} \cup \{a \dot{\leftrightarrow} b. a, b \in \mathcal{N}\} \\
\otimes & = \text{The similarly notated operator } \otimes \text{ in CC-Pi} \\
\mathbf{1} & = 1 \\
\Psi \vdash \varphi & \text{if } \left\{ \begin{array}{ll} \Psi \preceq \varphi & \text{if } \Psi, \varphi \in \mathcal{C} \\ \Psi \vdash a = b \wedge a, b \in \mathcal{N} & \text{if } \varphi = a \dot{\leftrightarrow} b \end{array} \right.
\end{array}$$

Thus terms, conditions and assertions are defined by the carrier of the named c-semiring (which by definition includes names and name fusions); $\mathbf{1}$ is the unit element, $\otimes = \otimes$, entailment is the CC-Pi partial order $\preceq$ and in addition $\Psi \vdash a \dot{\leftrightarrow} b$ if $\Psi \vdash a = b$ and both a and b are names. We extend a monadic version of the pi-F formulation above by representing **ask** $\varphi\,.\,P$ as **if** φ **then** P and **tell** $\Psi\,.\,P$ as $(\!|\Psi|\!) \mid P$. We avoid recursion (it can be encoded using replication).

There are a couple of ways this psi-calculus differs from CC-Pi, apart from the source of the non-compositionality mentioned above. Most prominently, in the semantics of CC-Pi the fusions resulting from communication are required to be consistent with the store (as defined by the constraint system). In contrast our semantics will allow transitions that lead to an inconsistent store. In general both of these approaches have been used in concurrent constraint systems. It appears not possible to integrate a consistency check in a psi-calculus communication without changing our COM-rule.

An example of how a CC-Pi communication is represented in psi:

In CC-Pi:

$$\begin{array}{rcl} P & = & \Psi \mid \overline{a}b \,.\, Q \mid cd \,.\, R \xrightarrow{\tau} \Psi \otimes (b = d) \mid Q \mid R \\ & & \text{if } \Psi \vdash a = c \text{ and } \Psi \otimes (b = d) \text{ consistent} \end{array}$$

In psi:

$$\begin{array}{rcl} P & = & (\!|\Psi|\!) \mid \overline{a}\, b \,.\, Q \mid c(x) \,.\, ((\!|x = d|\!) \mid R) \\ & \xrightarrow{\tau} & (\!|\Psi|\!) \mid (\!|b = d|\!) \mid Q \mid R \qquad \text{if } \Psi \vdash a = c \end{array}$$

3.3 Further examples

Polyadic synchronisation In a psi-calculus the channels can be arbitrary terms. This means that it is possible to introduce functions on channels (e.g., if M is a channel then so is $f(M)$). It also means that a channel can contain more than one name. An extension of this kind is explored by Carbone and Maffeis [8] in the so called pi-calculus with polyadic synchronisation, ${}^{e}\pi$. Here action subjects are tuples of names, and it is demonstrated that this allows a gradual enabling of communication by opening the scope of names in a subject, results in simple representations of localities and cryptography, and gives a strictly greater expressiveness than standard pi-calculus. We can represent ${}^{e}\pi$ by using tuples of names in subject position. The only modification to the representation of the polyadic pi-calculus is to extend $\vdash$ to $\vdash = \{(\mathbf{1}, M \dot{\leftrightarrow} M).\ M \in \mathbf{T}\}$, and to remove the conditions of type $M = N$ (since they can be encoded in ${}^{e}\pi$).

Higher-order data The data terms can be drawn from any nominal data type, including higher-order formalisms. It is thus possible to transmit functions between agents. For example, let $\mathbf{T}$ be the lambda calculus, containing abstractions $\boldsymbol{\lambda}x.M$ and applications MN. In the parallel composition $\overline{a}\,(\boldsymbol{\lambda}x.M) \,.\, P \mid a(z) \,.\, \overline{b}\, zN \,.\, Q$ the left hand component transmits the function $\boldsymbol{\lambda}x.M$ to the right, where the application of it to N is transmitted along b. Reduction would be represented as a binary predicate over lambda terms and could be tested in psi-calculus conditions (the reduction rules would be part of the definition of entailment). In this sense psi can resemble a higher-order calculus. It is even possible to let the terms be the psi-calculus agents themselves. An agent transmitted as a term cannot directly communicate with the agent that sent or received it, but there is a possibility of indirect interaction through the entailment relation. This area we leave for further study.

Advanced constraint systems Psi-calculi go beyond most concurrent constraint systems in two ways. Firstly, we allow arbitrary logics, even higher-order ones. Secondly, we allow an agent to transmit a constraint to another process. For example, assume that $\mathtt{c}$ is constraint and that $\mathtt{f}$ is a function from assertions to assertions. Then we can write $\overline{a}\, \mathtt{c} \,.\, P \mid a(z) \,.\, ((\!|\mathtt{f}(z)|\!) \mid Q)$ where the left hand agent sends a the constraint $\mathtt{c}$ to the right, and $\mathtt{f}$ is applied to it.

Unnamed ether Finally, a simple instantiation of the psi-calculus parameters illustrates the generality of the framework. Assume we want to describe a situation where there is only one unnamed communication channel, and where any process may declare a local alias for it. We define

$$\begin{array}{ll} \mathbf{T} & = \mathcal{N} \\ \mathbf{C} & = \{a \dot{\leftrightarrow} b.\ a, b \in \mathbf{T}\} \\ \mathbf{A} & = \mathcal{P}_{\text{fin}}(\mathcal{N}) \\ \otimes & = \cup \\ \mathbf{1} & = \emptyset \\ \vdash & = \{(\Psi, a \dot{\leftrightarrow} b).\ a, b \in \Psi\} \end{array}$$

In other words, the only terms are names, assertions are sets of names, and two names are channel equivalent if they are in an assertion. Omitting the action and prefix objects we get $\{a, b\} \rhd \overline{a}.\mathbf{0} \xrightarrow{\overline{a}} \mathbf{0}$, and also $\{a, b\} \rhd \overline{a}.\mathbf{0} \xrightarrow{\overline{b}} \mathbf{0}$. By the PAR rule we have $\emptyset \rhd \overline{a}.\mathbf{0}|(\!|\{a, b\}|\!) \xrightarrow{\overline{a}} \mathbf{0}|(\!|\{a, b\}|\!)$ and $\emptyset \rhd \overline{a}.\mathbf{0}|(\!|\{a, b\}|\!) \xrightarrow{\overline{b}} \mathbf{0}|(\!|\{a, b\}|\!)$. Applying a restriction we get $\emptyset \rhd (\nu a)(\overline{a}.\mathbf{0}|(\!|\{a, b\}|\!)) \xrightarrow{\overline{b}} (\nu a)(\mathbf{0}|(\!|\{a, b\}|\!))$ but no corresponding action with subject a because of the side condition on SCOPE. Thus, a communication through COM can be inferred from $(\nu a)(\overline{a}.\mathbf{0}|(\!|\{a, b\}|\!)) \mid b.\mathbf{0}$, but not from $(\nu a)(\overline{a}.\mathbf{0}|(\!|\{a, b\}|\!)) \mid a.\mathbf{0}$.

A more complicated example indicates the subtlety involved in establishing scope extension, i.e., if $a \# Q$ then the agents $(\nu a)(P|Q)$ and $(\nu a)P|Q$ have the same transitions. We have that $\{a\} \cup \{b\} \vdash a \dot{\leftrightarrow} b$, and thus similarly to the inference above that $(\nu a, b)((\!|\{a\}|\!) \mid (\!|\{b\}|\!) \mid \overline{a}.\mathbf{0} \mid b.\mathbf{0})$ has an internal communication. By scope extension this agent should have the same transitions as $P \mid Q$ where

$$P = (\nu a)((\!|\{a\}|\!) \mid \overline{a}.\mathbf{0}) \qquad Q = (\nu b)((\!|\{b\}|\!) \mid b.\mathbf{0})$$

Here $\mathcal{F}(P) = \langle a, \{a\} \rangle$ and $\mathcal{F}(Q) = \langle b, \{b\} \rangle$. A communication from $P \mid Q$ is inferred by COM and the premises

1. $\{a\} \rhd P \xrightarrow{\overline{b}} (\nu a)((\!|\{b\}|\!) \mid \mathbf{0})$
   (derived using $\{a\} \otimes \{b\} = \{a, b\} \vdash a \dot{\leftrightarrow} b$ in OUT)
2. $\{b\} \rhd Q \xrightarrow{a} (\nu a)((\!|\{a\}|\!) \mid \mathbf{0})$
   (derived using $\{b\} \otimes \{a\} = \{a, b\} \vdash a \dot{\leftrightarrow} b$ in IN)
3. $\{a\} \otimes \{b\} = \{a, b\} \vdash a \dot{\leftrightarrow} b$

The action subjects are derived by the assertions in both cases to not clash with the binders.

This example illustrates the necessity of using channel equivalence in all of IN, OUT and COM. It also demonstrates why transitions in Table 1 are defined with assertions and not frames, for whereas $\{a, b\} \vdash a \dot{\leftrightarrow} b$ the corresponding result cannot be obtained from the frames. $\mathcal{F}(Q) \otimes \{a\} = \langle b, \{a,b\}\rangle \nvdash a \leftrightarrow b$, so that frame is not useful for deriving a transition from P. The condition $B_Q \# \alpha$ in PAR prevents $P \mid Q$ from additionally having an action with the name a.

4 Bisimilarity

4.1 Definition

For two agents to be bisimilar we require that the actions from one can be simulated by the other. We also require that the agents be statically equivalent, meaning that their frames are equivalent. Finally, we require the equivalence to hold for all possible assertions in an environment. This leads to the following definition.

Definition 10 (Bisimulation)**.** *A* 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* 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 $P \dot{\sim}_\Psi Q$ *to mean that there exists a bisimulation* $\mathcal{R}$ *such that* $\mathcal{R}(\Psi, P, Q)$*, and write* $\dot{\sim}$ *for* $\dot{\sim}_{\mathbf{1}}$.

Interestingly, there is an alternative way to define bisimulation as a *binary* relation $\mathcal{R}$ such that $P \mathcal{R} Q$ implies static equivalence $\mathcal{F}(P) \simeq \mathcal{F}(Q)$, extension of arbitrary parallel assertion $((\!|\Psi|\!) | P) \mathcal{R} ((\!|\Psi|\!) | Q)$, symmetry, and simulation (as in Definition 10.4 with $\mathbf{1}$ for Ψ). Such a definition is more in line with standard contextual bisimulations. The drawback is that it relies on an operator in the calculus (parallel) for its definition. For conducting proofs our experience is that Definition 10 is preferable. We have shown that these bisimilarities coincide, i.e., the definitions results in the same $\dot{\sim}$.

An example clarifies the role of point number 3. Let β be a prefix and let φ be any non-trivial condition.

$$P = \beta.\beta.\mathbf{0} + \beta.\mathbf{0} + \beta.\ \textbf{if}\ \varphi\ \textbf{then}\ \beta.\mathbf{0}$$
$$Q = \beta.\beta.\mathbf{0} + \beta.\mathbf{0}$$

P can nondeterministically choose between three branches and Q between the two first of them. Here P and Q are not bisimilar. If P performs an action corresponding to its third case, reaching the agent $P' = \textbf{if}\ \varphi\ \textbf{then}\ \beta.\mathbf{0}$, there is no way that Q can simulate since neither $Q' = \mathbf{0}$ nor $Q' = \beta.\mathbf{0}$ is equivalent to P' in all environments. In fact, any reasonable variant of bisimulation that equates P and Q will not be preserved by parallel. To see this, let T be $\gamma.(\!|\Psi|\!)$, where γ is any prefix and Ψ an assertion that entails φ. Then the transition $P \mid T \xrightarrow{\beta} P' \mid T$ cannot be simulated by $Q|T$, since $P'|T$ can only do an action γ followed by an action β, whereas $\beta.\mathbf{0}|T$ can do β immediately, and $\mathbf{0}|T$ can do no β at all. This demonstrates why clause 3, extension of arbitrary assertion, is necessary: it says that after each step all possible extensions of the assertion must be considered. If we would merely require this at top level, i.e. remove clause 3 and instead require $\forall \Psi. \mathcal{R}(\Psi, P, Q)$ in the definition of $P \dot{\sim} Q$, the extensions would not recur; as a consequence P and Q in the example would be equivalent, and the equivalence would not be preserved by parallel.

For another example, consider

$$R = \textbf{if}\ \varphi\ \textbf{then}\ \beta\,.\,\textbf{if}\ \varphi\ \textbf{then}\ \beta.\mathbf{0} \qquad S = \textbf{if}\ \varphi\ \textbf{then}\ \beta.\beta.\mathbf{0}$$

In R the condition φ is checked twice. In general R and S are not equivalent. To see this, let Ψ and Ψ' be such that $\Psi \vdash \varphi$ and $\Psi \otimes \Psi' \nvdash \varphi$. We then have that $\Psi \rhd R \xrightarrow{\beta} \textbf{if}\ \varphi\ \textbf{then}\ \beta.\mathbf{0}$ and it cannot be simulated by $\Psi \rhd S \xrightarrow{\beta} \beta.\mathbf{0}$ because of the recurring clause of extension of arbitrary assertion: $\textbf{if}\ \varphi\ \textbf{then}\ \beta.\mathbf{0}$ has no transition in the environment $\Psi \otimes \Psi'$. However, if the entailment relation satisfies weakening, i.e. $\Psi \vdash \varphi \Rightarrow \Psi \otimes \Psi' \vdash \varphi$, we get the intuitive result that R and S are bisimilar. Weakening is a quite natural requirement, intuitively it says that no assertion can "undo" any entailments. This also demonstrates why we rejected the smaller and simpler definition of $\dot{\sim}$ as the largest relation satisfying

$$\forall \Psi. \Psi \rhd P \xrightarrow{\beta} P' \Longrightarrow \Psi \rhd Q \xrightarrow{\beta} Q' \ \wedge\ P' \dot{\sim} Q'$$

The difference is that here bisimulation recurringly requires to hold for *all* assertions, not only for those that are extensions of the ones passed so far. This would have the unintuitive effect of making R and S in the example above non-bisimilar, even if weakening holds.

4.2 Algebraic properties

Our results are that bisimilarity is preserved by the operators in the expected way, and also satisfies the expected structural algebraic laws.

Theorem 11.

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

Theorem 12. *For all* Ψ*:*

1. $P \dot{\sim}_\Psi Q \Longrightarrow P \mid R \dot{\sim}_\Psi Q \mid R.$
2. $P \dot{\sim}_\Psi Q \Longrightarrow (\nu a)P \dot{\sim}_\Psi (\nu a)Q.$
3. $P \dot{\sim}_\Psi Q \Longrightarrow !P \dot{\sim}_\Psi !Q.$
4. $\forall i. P_i \dot{\sim}_\Psi Q_i \Longrightarrow \textbf{case}\ \widetilde{\varphi} : \widetilde{P} \dot{\sim}_\Psi \textbf{case}\ \widetilde{\varphi} : \widetilde{Q}.$
5. $P \dot{\sim}_\Psi Q \Longrightarrow \overline{M}\, N.P \dot{\sim}_\Psi \overline{M}\, N.Q.$
6. $(\forall \widetilde{L}.\ P[\widetilde{a} := \widetilde{L}] \dot{\sim}_\Psi Q[\widetilde{a} := \widetilde{L}]) \Longrightarrow \underline{M}(\lambda\widetilde{a})N.P \dot{\sim}_\Psi \underline{M}(\lambda\widetilde{a})N.Q$

The most awkward part of the proof is for Theorem 12.1, which requires some ingenuity in formulating the right lemmas.

Definition 13. $P \sim_\Psi Q$ *means that for all* $\widetilde{x}, \widetilde{M}$ *it holds* $P[\widetilde{x} := \widetilde{M}] \dot{\sim}_\Psi Q[\widetilde{x} := \widetilde{M}]$*, and we write* $P \sim Q$ *for* $P \sim_{\mathbf{1}} Q$.

The laws of Theorem 11 are valid also for $\sim$, moreover we have:

Theorem 14. $\sim_\Psi$ *is a congruence for all* Ψ.

We have completely formalised all proofs in this section using Isabelle [3].

5 Conclusion and Further Work

We have defined a framework for mobile process calculi, parametrised on nominal types for data terms and for a logic to express assertions and conditions. We have explored the expressiveness in comparison to related calculi and established basic properties of strong bisimilarity. This is a starting point for many interesting investigations. Several versions of applied pi-calculi focus on cryptographic properties of agents where type checking proves an agent secure. We intend to explore typed psi-calculi to include this kind of reasoning. One idea is to find out what properties the type system must have in order for the usual theorems (e.g. subject reduction) to hold.

We are currently developing a symbolic semantics and are investigating weak and barbed equivalences for our framework.

References

[1] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proceedings of POPL '01*, pages 104–115. ACM, Jan. 2001.

[2] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The Spi calculus. *Journal of Information and Computation*, 148(1):1–70, 1999.

[3] J. Bengtson and J. Parrow. Psi-calculi in Isabelle. In *Proc. of TPHOLs 2009*, volume 5674 of *LNCS*. Springer, 2009. To appear.

[4] J. Borgström and U. Nestmann. On bisimulations for the spi calculus. *Mathematical Structures in Computer Science*, 15(03):487–552, 2005.

[5] M. G. Buscemi. Personal communication, Jan. 2009.

[6] M. G. Buscemi and U. Montanari. CC-Pi: A constraint-based language for specifying service level agreements. In R. De Nicola, editor, *Proceedings of ESOP 2007*, volume 4421 of *LNCS*, pages 18–32. Springer, 2007.

[7] M. G. Buscemi and U. Montanari. Open bisimulation for the concurrent constraint pi-calculus. In S. Drossopoulou, editor, *Proceedings of ESOP 2008*, volume 4960 of *LNCS*, pages 254–268. Springer, 2008.

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

[9] V. Cortier, M. Rusinowitch, and E. Zalinescu. Relating two standard notions of secrecy. *Logical Methods in Computer Science*, 3(3), 2007.

[10] S. Delaune, S. Kremer, and M. D. Ryan. Symbolic bisimulation for the applied pi-calculus. In V. Arvind and S. Prasad, editors, *Proc. of FSTTCS'07*, volume 4855 of *LNCS*, pages 133–145. Springer, Dec. 2007.

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

[12] J. Goubault-Larrecq, C. Palamidessi, and A. Troina. A probabilistic applied pi-calculus. In *Proc. of APLAS'07*, volume 4807 of *LNCS*, pages 175–190. Springer, 2007.

[13] M. Johansson, J. Parrow, B. Victor, and J. Bengtson. Extended pi-calculi. In *Proceedings of ICALP 2008*, volume 5126 of *LNCS*, pages 87–98. Springer, July 2008.

[14] S. Kremer and M. D. Ryan. Analysis of an electronic voting protocol in the applied pi-calculus. In M. Sagiv, editor, *Proc. of ESOP'05*, volume 3444 of *LNCS*, pages 186–200. Springer, Apr. 2005.

[15] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part I/II. *Journal of Information and Computation*, 100:1–77, Sept. 1992.

[16] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: a Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

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

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

[19] L. Wischik and P. Gardner. Explicit fusions. *Theoretical Computer Science*, 304(3):606–630, 2005.