A Sorted Semantic Framework for Applied Process Calculi (Extended Abstract)

Johannes Borgström(✉), Ramūnas Gutkovas, Joachim Parrow, Björn Victor, and Johannes Åman Pohjola

Department of Information Technology, Uppsala University, Uppsala, Sweden
johannes.borgstrom@it.uu.se

Abstract. Applied process calculi include advanced programming constructs such as type systems, communication with pattern matching, encryption primitives, concurrent constraints, nondeterminism, process creation, and dynamic connection topologies. Several such formalisms, e.g. the applied pi calculus, are extensions of the the pi-calculus; a growing number is geared towards particular applications or computational paradigms.

Our goal is a unified framework to represent different process calculi and notions of computation. To this end, we extend our previous work on psi-calculi with novel abstract patterns and pattern matching, and add sorts to the data term language, giving sufficient criteria for subject reduction to hold. Our framework can accommodate several existing process calculi; the resulting transition systems are isomorphic to the originals up to strong bisimulation. We also demonstrate different notions of computation on data terms, including cryptographic primitives and a lambda-calculus with erratic choice. Substantial parts of the meta-theory of sorted psi-calculi have been machine-checked using Nominal Isabelle.

1 Introduction

There is today a growing number of high-level constructs in the area of concurrency. Examples include type systems, communication with pattern matching, encryption primitives, concurrent constraints, nondeterminism, and dynamic connection topologies. Combinations of such constructs are included in a variety of application oriented process calculi. For each such calculus its internal consistency, in terms of congruence results and algebraic laws, must be established independently. Our aim is a framework where many such calculi fit and where such results are derived once and for all, eliminating the need for individual proofs about each calculus.

Our effort in this direction is the framework of psi-calculi [1], which provides machine-checked proofs that important meta-theoretical properties, such as compositionality of bisimulation, hold in all instances of the framework. In this paper we introduce a novel generalization of pattern matching, decoupled from the definition of substitution, and introduce sorts for data terms and names.

M. Abadi and A. Lluch Lafuente (Eds.): TGC 2013, LNCS 8358, pp. 103–118, 2014.
DOI: 10.1007/978-3-319-05119-2_7,

The generalized pattern matching is a new contribution that holds general interest; here it allows us to directly capture computation on data in advanced process calculi, without elaborate encodings. We evaluate our framework by providing instances that are isomorphic to standard calculi, and by representing several different notions of computation. This is an advance over our previous work, where we had to resort to nontrivial encodings with unclear formal correspondence to the standard calculi.

1.1 Background: Psi-calculi

A psi-calculus has a notion of data terms, ranged over by K, L, M, N, and we write $\overline{M}\ N\,.\,P$ to represent an agent sending the term N along the channel M (which is also a data term), continuing as the agent P. We write $\underline{K}(\lambda\widetilde{x})X\,.\,Q$ to represent an agent that can input along the channel K, receiving some object matching the pattern X, where $\widetilde{x}$ are the variables bound by the prefix. These two agents can interact under two conditions. First, the two channels must be *channel equivalent*, as defined by the channel equivalence predicate $M \leftrightarrow K$. Second, N must match the pattern X.

Formally, a *transition* is of kind $\Psi \rhd P \xrightarrow{\alpha} P'$, meaning that in an environment represented by the *assertion* Ψ the agent P can do an action α to become P'. An assertion embodies a collection of facts used to infer *conditions* such as the channel equivalence predicate $\leftrightarrow$. To continue the example, if $N = X[\widetilde{x} := \widetilde{L}]$ we will have $\Psi \rhd \overline{M}\ N\,.\,P \mid \underline{K}(\lambda\widetilde{x})X\,.\,Q \xrightarrow{\tau} P \mid Q[\widetilde{x} := \widetilde{L}]$ when additionally $\Psi \vdash M \leftrightarrow K$, i.e. when the assertion Ψ entails that M and K represent the same channel. In this way we may introduce a parametrised equational theory over a data structure for channels. Conditions, ranged over by φ, can be tested in the **if** construct: we have that $\Psi \rhd$ **if** φ **then** P $\xrightarrow{\alpha}$ P' when $\Psi \vdash \varphi$ and $\Psi \rhd P \xrightarrow{\alpha} P'$. In order to represent concurrent constraints and local knowledge, assertions can be used as agents: $(\!|\Psi|\!)$ stands for an agent that asserts Ψ to its environment. Assertions may contain names and these can be scoped; for example, in $P \mid (\nu a)((\!|\Psi|\!) \mid Q)$ the agent Q uses all entailments provided by Ψ, while P only uses those that do not contain the name a.

Assertions and conditions can, in general, form any logical theory. Also the data terms can be drawn from an arbitrary set. One of our major contributions has been to pinpoint the precise requirements on the data terms and logic for a calculus to be useful in the sense that the natural formulation of bisimulation satisfies the expected algebraic laws (see Sect. 2). It turns out that it is necessary to view the terms and logics as *nominal* [2]. This means that there is a distinguished set of names, and for each term a well defined notion of *support*, intuitively corresponding to the names occurring in the term.

1.2 Extension: Generalized Pattern Matching

In our original definition of psi-calculi [1] (called "the original psi-calculi" below), patterns are just terms and pattern matching is defined by substitution in the

usual way: the output object N matches the pattern X with binders $\widetilde{x}$ iff $N = X[\widetilde{x} := \widetilde{L}]$. In order to increase the generality we now introduce a function MATCH which takes a term N, a sequence of names $\widetilde{x}$ and a pattern X, returning a set of sequences of terms; the intuition is that if $\widetilde{L}$ is in $\text{MATCH}(N, \widetilde{x}, X)$ then N matches the pattern X by instantiating $\widetilde{x}$ to $\widetilde{L}$. The receiving agent $\underline{K}(\lambda\widetilde{x})X \,.\, Q$ then continues as $Q[\widetilde{x} := \widetilde{L}]$.

As an example we consider a term algebra with two function symbols: `enc` of arity three and `dec` of arity two. Here $\texttt{enc}(N, n, k)$ means encrypting N with the key k and a random nonce n and and $\texttt{dec}(N, k)$ represents symmetric key decryption, discarding the nonce. Suppose an agent sends an encryption, as in $\overline{M}\ \texttt{enc}(N, n, k) \,.\, P$. If we allow all terms to act as patterns, a receiving agent can use $\texttt{enc}(x, y, z)$ as a pattern, as in $\underline{c}(\lambda x, y, z)\texttt{enc}(x, y, z) \,.\, Q$, and in this way decompose the encryption and extract the message and key. Using the encryption function as a destructor in this way is clearly not the intention of a cryptographic model. With the new general form of pattern matching, we can simply limit the patterns to not bind names in terms at key position. Together with the separation between patterns and terms, this allows to directly represent dialects of the spi-calculus as in Examples 4 and 5 in Sect. 3.

Moreover, the generalization makes it possible to safely use rewrite rules such as $\texttt{dec}(\texttt{enc}(M, N, K), K) \to M$. In the psi-calculi framework such evaluation is not a primitive concept, but it can be part of the substitution function, with the idea that with each substitution all data terms are normalized according to rewrite rules. Such evaluating substitutions are dangerous for two reasons. First, in the original psi-calculi they can introduce ill-formed input prefixes. The input prefix $\underline{M}(\lambda\widetilde{x})N$ is well-formed when $\widetilde{x} \subseteq \text{n}(N)$, i.e. the names $\widetilde{x}$ must all occur in N; a rewrite of the well-formed $\underline{M}(\lambda y)\texttt{dec}(\texttt{enc}(N, y, k), k) \,.\, P$ to $\underline{M}(\lambda y)N \,.\, P$ yields an ill-formed agent when y does not appear in N. Such ill-formed agents could also arise from input transitions in some original psi-calculi; with the current generalization preservation of well-formedness is guaranteed.

Second, in the original psi-calculi there is a requirement that a substitution of $\widetilde{L}$ for $\widetilde{x}$ in M must yield a term containing all names in $\widetilde{L}$ whenever $\widetilde{x} \subseteq \text{n}(M)$. The reason is explained at length in [1]; briefly put, without this requirement the scope extension law is unsound. If rewrites such as $\texttt{dec}(\texttt{enc}(M, N, K), K) \to M$ are performed by substitutions this requirement is not fulfilled, since a substitution may then erase the names in N and K. However, a closer examination reveals that this requirement is only necessary for some uses of substitution. In the transition

$$\underline{M}(\lambda\widetilde{x})N.P \xrightarrow{\underline{K}\ N[\widetilde{x}:=\widetilde{L}]} P[\widetilde{x} := \widetilde{L}]$$

the non-erasing criterion is important for the substitution above the arrow ($N[\widetilde{x} := \widetilde{L}]$) but unimportant for the substitution after the arrow ($P[\widetilde{x} := \widetilde{L}]$). In the present paper, we replace the former of these uses by the MATCH function, where a similar non-erasing criterion applies. All other substitutions may safely use arbitrary rewrites, even erasing ones.

1.3 Extension: Sorting

Applied process calculi often make use of a sort system. The applied pi-calculus [3] has a name sort and a data sort; terms of name sort must not appear as subterms of terms of data sort. It also makes a distinction between input-bound variables (which may be substituted) and restriction-bound names (which may not). The pattern-matching spi-calculus [4] uses a sort of patterns and a sort of implementable terms; every implementable term can also be used as a pattern.

To represent such calculi, we admit a user-defined sort system on names, terms and patterns. Substitutions are only well-defined if they conform to the sorting discipline. To specify which terms can be used as channels, and which values can be received on them, we use compatibility predicates on the sorts of the subject and the object in input and output prefixes. The conditions for preservation of sorting by transitions (subject reduction) are very weak, allowing for great flexibility when defining instances.

The restriction to well-sorted substitution also allows to avoid "junk": terms that exist solely to make substitutions total. A prime example is representing the polyadic pi-calculus as a psi-calculus. The terms that can be transmitted between agents are tuples of names. Since a tuple is a term it can be substituted for a name, even if that name is already part of a tuple. The result is that the terms must admit nested tuples of names, which do not occur in the original calculus.

1.4 Related Work

Pattern-matching is in common use in programming languages (e.g. Lisp, ML). LINDA [5] uses pattern-matching when receiving from a tuple space. The pattern-matching spi-calculus limits which variables may be binding in a pattern in order to match encrypted messages without binding unknown keys (cf. Example 5). In all these cases, the pattern matching is defined by substitution in the usual way. In more recent languages, such as Scala and F#, pattern matching may involve computation, similarly to this paper.

The Kell calculus [6] also uses pattern languages equipped with a match function. However, in the Kell calculus the channels are single names and appear as part of patterns, patterns may match multiple communications simultaneously (à la join calculus), and pattern variables only match names (not composite messages) making forwarding and partial decomposition impossible.

Sorts for the pi-calculus were first described by Milner [7]. Hüttel's typed psi-calculi [8] admit a family of dependent type systems, capable of capturing a wide range of earlier type systems for pi-like calculi formulated as instances of psi-calculi. However, the term language of typed psi-calculi is required to be a free term algebra (and without name binders); it uses only the standard notions of substitution and matching, and does not admit any computation on terms. The sophisticated type system of typed psi-calculi is intended for fine-grained control of the behaviour of processes, while we focus on an earlier step: the creation of a calculus that is as close to the modeller's intent as possible. Indeed,

sorted psi-calculi gives a formal account of the separation between variables and names in typed psi-calculi, and Hüttel's claim that "the set of well-[sorted] terms is closed under well-[sorted] substitutions, which suffices". Furthermore, we prove meta-theoretical results including preservation of well-formedness under structural equivalence; no such results exist for typed psi-calculi.

In the applied pi-calculus [3] the data language is a term algebra modulo an equational logic, which is suitable for modelling deterministic computation only. ProVerif [9] is a specialised tool for security protocol verification in an extension of applied pi, including a pattern matching construct. Its implementation allows pattern matching of tagged tuples modulo a user-defined rewrite system; this is strictly less general than the psi-calculus pattern matching described in this paper (cf. Example 2).

Fournet et al. [10] add a general authentication logic to a process calculus with destructor matching; the authentication logic is only used to specify program correctness, and do not influence the operational semantics in any way. A comparison of expressiveness to calculi with communication primitives other than binary directed communication, such as the concurrent pattern calculus [11] and the join-calculus [12], would be interesting. We here inherit positive results from the pi calculus, such as the encoding of the join-calculus.

1.5 Results and Outline

In Sect. 2 we define psi-calculi with the above extensions and explain the necessary change to the semantics. A formal account of the whole operational semantics and bisimulations can be found in an appendix. Our results are the usual algebraic properties of bisimilarity, preservation of well-formedness, and subject reduction.

We demonstrate the expressiveness of our generalization in Sect. 3 by directly representing calculi with advanced data structures and computations on them, even nondeterministic reductions.

2 Definitions

Psi-calculi are based on nominal data types. A nominal data type is similar to a traditional data type, but can also contain binders and identify alpha-variants of terms. Formally, the only requirements are related to the treatment of the atomic symbols called names as explained below. In this paper, we consider sorted nominal datatypes, where names may have different sorts.

We assume a set of sorts $\mathcal{S}$. Given a countable set of sorts for names $\mathcal{S}_\mathcal{N} \subseteq \mathcal{S}$, we assume countably infinite pair-wise disjoint sets of atomic *names* $\mathcal{N}_s$, where $s \in \mathcal{S}_\mathcal{N}$. The set of all names, $\mathcal{N} = \cup_s \mathcal{N}_s$, is ranged over by $a, b, \ldots, x, y, z$. We write $\widetilde{x}$ for a tuple of names $x_1, \ldots, x_n$ and similarly for other tuples, and $\widetilde{x}$ stands for the set of names $\{x_1, \ldots, x_n\}$ if used where a set is expected.

A sorted *nominal set* [2,13] is a set equipped with *name swapping* functions written $(a\ b)$, for any sort s and names $a, b \in \mathcal{N}_s$, i.e. name swappings must

respect sorting. An intuition is that for any member T it holds that $(a\ b)\cdot T$ is T with a replaced by b and b replaced by a. The support of a term, written $\mathrm{n}(T)$, is intuitively the set of names affected by name swappings on T. This definition of support coincides with the usual definition of free names for abstract syntax trees that may contain binders. We write $a\#T$ for $a \notin \mathrm{n}(T)$, and extend this to finite sets and tuples by conjunction. A function f is equivariant if $(a\ b)(f(T)) = f((a\ b)T)$ always. A *nominal data type* is a nominal set together with some functions on it, for instance a substitution function.

2.1 Original Psi-calculi Parameters

Sorted psi-calculi is an extension of the original psi-calculi framework [1].

Definition 1 (Original psi-calculus parameters). *The psi-calculus parameters from the original psi-calculus include three nominal data types: (data) terms $M, N \in \mathbf{T}$, conditions $\varphi \in \mathbf{C}$, and assertions $\Psi \in \mathbf{A}$; and four equivariant operators: channel equivalence $\leftrightarrow : \mathbf{T}\times\mathbf{T} \to \mathbf{C}$, assertion composition $\otimes : \mathbf{A}\times\mathbf{A} \to \mathbf{A}$, the unit assertion $\mathbf{1}$, and the entailment relation $\vdash \subseteq \mathbf{A} \times \mathbf{C}$.*

The binary functions $\leftrightarrow, \otimes$ and the relation $\vdash$ above will be used in infix form.

Two assertions are equivalent, written $\Psi \simeq \Psi'$, if they entail the same conditions, i.e. for all φ we have that $\Psi \vdash \varphi \Leftrightarrow \Psi' \vdash \varphi$. We impose certain requisites on the sets and operators. In brief, channel equivalence must be symmetric and transitive, the assertions with $(\otimes, \mathbf{1})$ must form an abelian monoid modulo $\simeq$, and $\otimes$ must be compositional w.r.t. $\simeq$ (i.e. $\Psi_1 \simeq \Psi_2 \implies \Psi \otimes \Psi_1 \simeq \Psi \otimes \Psi_2$). For details see [1].

2.2 New Parameters for Generalized Pattern-Matching

To the parameters of the original psi-calculi we add patterns X, Y, that are used in input prefixes, a function VARS which yields the possible combinations of binding names in the pattern, and a pattern-matching function MATCH, which is used when the input takes place. Intuitively, an input pattern $(\lambda\widetilde{x})X$ matches a message N if there are $\widetilde{L} \in \mathrm{MATCH}(N, \widetilde{x}, X)$; the receiving agent then continues after substituting $\widetilde{L}$ for $\widetilde{x}$. If $\mathrm{MATCH}(N, \widetilde{x}, X) = \emptyset$ then $(\lambda\widetilde{x})X$ does not match N; if $|\mathrm{MATCH}(N, \widetilde{x}, X)| > 1$ then one of the matches will be non-deterministically chosen. Below, we use "variable" for names that can be bound in a pattern.

Definition 2 (Psi-calculus parameters for pattern-matching). *The psi-calculus parameters for pattern-matching include the nominal data type $\mathbf{X}$ of (input) patterns, ranged over by X, Y, and the two equivariant operators*

$$\begin{array}{rll} \mathrm{MATCH} : \mathbf{T} \times \mathcal{N}^* \times \mathbf{X} \to \mathcal{P}_{\mathrm{fin}}(\mathbf{T}^*) & & \textit{Pattern matching} \\ \mathrm{VARS} : \mathbf{X} \to \mathcal{P}_{\mathrm{fin}}(\mathcal{P}_{\mathrm{fin}}(\mathcal{N})) & & \textit{Pattern variables} \end{array}$$

The VARS operator gives the possible (finite) sets of names in a pattern which are bound by an input prefix. For example, an input prefix with a pairing pattern $\langle x, y\rangle$ may bind both x and y, only one of them, or none, so $\mathrm{VARS}(\langle x, y\rangle) =$

$\{\{x,y\},\{x\},\{y\},\{\}\}$. This way, we can let the input prefix $\underline{c}(\lambda x)\langle x,y\rangle$ only match pairs where the second argument is the name y. To model a calculus where input patterns cannot be selective in this way, we may instead define $\text{VARS}(\langle x,y\rangle) = \{\{x,y\}\}$. This ensures that input prefixes that use the pattern $\langle x,y\rangle$ must be of the form $\underline{M}(\lambda x,y)\langle x,y\rangle$, where both x and y are bound. Another use for VARS is to exclude the binding of terms in certain positions, such as the keys of cryptographic messages (cf. Example 5).

Requirements on VARS and MATCH are given below in Definition 5. Note that the four data types $\mathbf{T}$, $\mathbf{C}$, $\mathbf{A}$ and $\mathbf{X}$ are not required to be disjoint. In most of the examples in this paper, the patterns $\mathbf{X}$ is a subset of the terms $\mathbf{T}$.

2.3 New Parameters for Sorting

To the parameters defined above we add a sorting function and four sort compatibility predicates.

Definition 3 (Psi-calculus parameters for sorting). *The psi-calculus parameters for sorting include the sorting function* $\text{SORT} : \mathcal{N} \uplus \mathbf{T} \uplus \mathbf{X} \to \mathcal{S}$*, and the four compatibility predicates*

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

The SORT operator gives the sort of a name, term or pattern; on names we require that $\text{SORT}(a) = s$ iff $a \in \mathcal{N}_s$. The sort compatibility predicates are used to restrict where terms and names of certain sorts may appear in processes. Terms of sort s can be used to send values of sort t if $s \mathrel{\overline{\propto}} t$. Dually, a term of sort s can be used to receive with a pattern of sort t if $s \mathrel{\underline{\propto}} t$. A name a can be used in a restriction (νa) if $\text{SORT}(a) \in \mathcal{S}_\nu$. If $\text{SORT}(a) \prec \text{SORT}(M)$ we can substitute the term M for the name a. In most of our examples, $\prec$ is a subset of the equality relation. These predicates can be chosen freely, although the set of well-formed substitutions depends on $\prec$, as detailed in Definition 4 below.

2.4 Substitution and Matching

We require that each datatype is equipped with an equivariant substitution function, which intuitively substitutes terms for names. The requisites on substitution differ from the original psi-calculi as indicated in the Introduction. Substitutions must preserve or refine sorts, and bound pattern variables must not be removed by substitutions.

We define a subsorting preorder $\leq$ on $\mathcal{S}$ as $s_1 \leq s_2$ if s_1 can be used as a channel or message whenever s_2 can be: formally $s_1 \leq s_2$ iff $\forall t \in \mathcal{S}.(s_2 \mathrel{\underline{\propto}} t \Rightarrow s_1 \mathrel{\underline{\propto}} t) \wedge (s_2 \mathrel{\overline{\propto}} t \Rightarrow s_1 \mathrel{\overline{\propto}} t) \wedge (t \mathrel{\underline{\propto}} s_2 \Rightarrow t \mathrel{\underline{\propto}} s_1) \wedge (t \mathrel{\overline{\propto}} s_2 \Rightarrow t \mathrel{\overline{\propto}} s_1)$. This relation compares sorts of terms, and so does not have any formal relationship to $\prec$ (which relates the sort of a name to the sort of a term).

Definition 4 (Substitution). *If $\widetilde{a}$ is a sequence of distinct names and $\widetilde{N}$ is an equally long sequence of terms such that* $\text{SORT}(a_i) \prec \text{SORT}(N_i)$ *for all i, we say that $[\widetilde{a} := \widetilde{N}]$ is a* substitution. *Substitutions are ranged over by σ.*

For each data type among $\mathbf{T}, \mathbf{A}, \mathbf{C}$ we define substitution on elements T of that data type as follows: we require that $T\sigma$ is an element of the same data type, and that if $(\widetilde{a}\ \widetilde{b})$ is a (bijective) name swapping such that $\widetilde{b}\#T, \widetilde{a}$ then $T[\widetilde{a} := \widetilde{N}] = ((\widetilde{a}\ \widetilde{b}).T)[\widetilde{b} := \widetilde{N}]$ (alpha-renaming of substituted variables). For terms we additionally require that $\text{SORT}(M\sigma) \leq \text{SORT}(M)$.

For substitution on patterns $X \in \mathbf{X}$, we require that if $\widetilde{x} \in \text{VARS}(X)$ and $\widetilde{x}\#\sigma$ then $X\sigma \in \mathbf{X}$ and $\text{SORT}(X\sigma) \leq \text{SORT}(X)$ *and $\widetilde{x} \in \text{VARS}(X\sigma)$ and alpha-renaming of substituted variables (as above) holds.*

Intuitively, the requirements on substitutions on patterns ensure that a substitution on a pattern with binders $((\lambda\widetilde{x})X)\sigma$ with $\widetilde{x} \in \text{VARS}(X)$ and $\widetilde{x}\#\sigma$ yields a pattern $(\lambda\widetilde{x})Y$ with $\widetilde{x} \in \text{VARS}(Y)$. As an example, consider the pair patterns discussed above with $\mathbf{X} = \{\langle x, y\rangle : x \neq y\}$ and $\text{VARS}(\langle x, y\rangle) = \{\{x, y\}\}$. We can let $\langle x, y\rangle\sigma = \langle x, y\rangle$ when $x, y\#\sigma$. Since $\text{VARS}(\langle x, y\rangle) = \{\{x, y\}\}$ the pattern $\langle x, y\rangle$ in a well-formed agent will always occur directly under the binder $(\lambda x, y)$, i.e. in $(\lambda x, y)\langle x, y\rangle$, and here a substitution for x or y will have no effect. It therefore does not matter what e.g. $\langle x, y\rangle[x := M]$ is, since it will never occur in derivations of transitions of well-formed agents. We could think of substitutions as partial functions which are undefined in such cases; formally, since substitutions are total, the result of this substitution can be assigned an arbitrary value.

Matching must be invariant under renaming of pattern variables, and the substitution resulting from a match must not contain any names that are not from the matched term or the pattern:

Definition 5 (Generalized pattern matching). *For the function* MATCH *we require that if $\widetilde{x} \in \text{VARS}(X)$ are distinct and $\widetilde{N} \in \text{MATCH}(M, \widetilde{x}, X)$ then it must hold that $[\widetilde{x} := \widetilde{N}]$ is a substitution, that $\text{n}(\widetilde{N}) \subseteq \text{n}(M) \cup (\text{n}(X) \setminus \widetilde{x})$, and that for all name swappings $(\widetilde{x}\ \widetilde{y})$ we have $\widetilde{N} \in \text{MATCH}(M, \widetilde{y}, (\widetilde{x}\ \widetilde{y})X)$ (alpha-renaming of matching).*

In the original psi-calculi equivariance of matching is imposed as a requirement on substitution on terms, but there is no requirement that substitutions preserve pattern variables. For this reason, the original psi semantics does not preserve the well-formedness of agents (an input prefix $\underline{M}(\lambda\widetilde{x})N\,.\,P$ is well-formed when $\widetilde{x} \subseteq \text{n}(N)$), although this is assumed by the operational semantics [1]. In contrast, the semantics of pattern-matching psi-calculi does preserve well-formedness, as shown below in Theorem 1.

In many process calculi, and also in the symbolic semantics of psi [14], the input construct binds a single variable. This is a trivial instance of pattern matching where the pattern is a single bound variable, matching any term.

Example 1 Given values for the other requisites, we can take $\mathbf{X} = \mathcal{N}$ with $\text{VARS}(a) = \{a\}$, meaning that the pattern variable must always occur bound, and $\text{MATCH}(M, a, a) = \{M\}$ if $\text{SORT}(a) \prec \text{SORT}(M)$. On patterns we define substitution as $a\sigma = a$ when $a\#\sigma$.

2.5 Agents

Definition 6 (Agents). *The* agents*, ranged over by $P, Q, \ldots$, are of the following forms.*

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

In the Input any name in $\widetilde{x}$ binds its occurrences in both X and P, and in the Restriction a binds in P. An assertion is guarded *if it is a subterm of an Input or Output. An agent is* well-formed *if, for all its subterms, in a replication $!P$ there are no unguarded assertions in P, and in* **case** $\varphi_1 : P_1 \,[]\, \cdots \,[]\, \varphi_n : P_n$ *there are no unguarded assertion in any P_i. Substitution on agents is defined inductively on their structure, using the substitution function of each datatype based on syntactic position, avoiding name capture.*

In comparison to [1] we restrict the syntax of well-formed agents by imposing requirements on sorts: the subjects and objects of prefixes must have compatible sorts, and restrictions may only bind names of a sort in $\mathcal{S}_\nu$.

Definition 7. *In sorted psi-calculi, an agent is* well-formed *if additionally the following holds for all its subterms. In an Output $\overline{M}\ N.P$ we require that* $\text{SORT}(M) \mathrel{\overline{\propto}} \text{SORT}(N)$*. In an Input $\underline{M}(\lambda\widetilde{x})X.P$ we require that $\widetilde{x} \in \text{VARS}(X)$ is a tuple of distinct names and* $\text{SORT}(M) \mathrel{\underline{\propto}} \text{SORT}(X)$*. In a Restriction $(\nu a)P$ we require that* $\text{SORT}(a) \in \mathcal{S}_\nu$*.*

The output prefix $\overline{M}\ N.P$ sends N on a channel that is equivalent to M. Dually, $\underline{M}(\lambda\widetilde{x})X.P$ receives a message matching the pattern X from a channel equivalent to M. A non-deterministic case statement **case** $\varphi_1 : P_1 \,[]\, \cdots \,[]\, \varphi_n : P_n$ executes one of the branches P_i where the corresponding condition φ_i holds, discarding the other branches. Restriction $(\nu a)P$ scopes the name a in P; the scope of a may be extruded if P communicates a data term containing a. A parallel composition $P \mid Q$ denotes P and Q running in parallel; they may proceed independently or communicate. A replication $!P$ models an unbounded number of copies of the process P. The assertion $(\!|\Psi|\!)$ contributes Ψ to its environment. We often write **if** φ **then** P for **case** $\varphi : P$, and nothing or **0** for the empty case statement **case**.

2.6 Semantics and Bisimulation

The semantics of a psi-calculus is defined inductively as a structural operation semantics yielding a labelled transition relation. The full definition can be found

in our earlier work [1]. We here only comment on the one change necessary to accommodate the generalized pattern matching. The original input rule reads

$$\frac{\Psi \vdash M \leftrightarrow K}{\Psi \rhd \underline{M}(\lambda \widetilde{y})X.P \xrightarrow{\underline{K}\ X[\widetilde{y}:=\widetilde{L}]} P[\widetilde{y} := \widetilde{L}]}$$

and means that the instantiating substitution $[\widetilde{y} := \widetilde{L}]$ is applied both in the transition label and in the agent after the transition. Our new input rule is

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

Here the matching with the transition label and the substitution applied to the following agent may be different. The MATCH predicate determines both the former (by designating the term N) and the latter (by designating the substitution), but there is no requirement on how they relate. As explained in Sect. 1.2 this means we can introduce evaluation of terms in the substitution or in the matching.

Theorem 1 (Preservation of well-formedness). *If P is well-formed, then $P\sigma$ is well-formed, and if $\Psi \rhd P \xrightarrow{\alpha} P'$ then P' is well-formed.*

Proof The first part is by induction on P. The interesting case is input $\underline{M}(\lambda \widetilde{x})X.Q$: assume that Q is well-formed, that $\widetilde{x} \in \text{VARS}(X)$, that $\text{SORT}(M) \underline{\propto} \text{SORT}(X)$ and that $\widetilde{x}\#\sigma$. By induction $Q\sigma$ is well-formed. By sort preservation we get $\text{SORT}(M\sigma) \leq \text{SORT}(M)$, so $\text{SORT}(M\sigma) \underline{\propto} \text{SORT}(X)$. By preservation of patterns by non-capturing substitutions we have that $\widetilde{x} \in \text{VARS}(X\sigma)$ and $\text{SORT}(X\sigma) \leq \text{SORT}(X)$, so $\text{SORT}(M\sigma) \underline{\propto} \text{SORT}(X\sigma)$. The second part is by induction on the transition rules, using part 1 in the IN rule.

Note that well-formedness implies conformance to the sorting discipline; therefore this theorem shows a kind of subject reduction property.

The definition of strong and weak bisimulation and their algebraic properties are unchanged from our previous work [1]. The results can be summarized as follows:

Theorem 2 (Properties of bisimulation). *All results on bisimulation established in [1] and [15] still hold in sorted psi-calculi with generalized matching.*

Theorem 2 has been formally verified in Isabelle/Nominal by adapting our existing proof scripts. The main difference is in the input cases of inductive proofs. This represents no more than two days of work, with the bulk of the effort going towards proving a crucial technical lemma stating that transitions do not invent new names with the new pattern matching. We have also machine-checked the proof of Theorem 1. Unfortunately, in Isabelle/Nominal there are currently no facilities to reason parametrically over the set of name sorts. Therefore the mechanically checked proofs only apply to psi-calculi with a trivial sorting (a single sort that is admitted everywhere); we complement them with manual proofs to extend these to arbitrary sortings.

3 Examples

Several well-known process algebras can be directly represented as a sorted psi-calculus by instantiating the parameters in the right way. With this we mean that the syntax is isomorphic and that the operational semantics is exactly preserved in a strong operational correspondence modulo strong bisimulation. There is no need for elaborate coding schemes and the correspondence proofs are straightforward.

Theorem 3 (Process algebra representations). *CCS with value passing [16], the unsorted and the sorted polyadic pi-calculus [7,17], and the polyadic synchronization pi-calculus [18] can all be directly represented as sorted psi-calculi.*

The list can certainly be made longer, though each process algebra currently has a separate definition and therefore requires a separate formal proof. For example, a version of LINDA [5] can easily be obtained as a variant of the polyadic pi-calculus. To illustrate the technique, the only difference between polyadic pi-calculus and polyadic synchronization pi-calculus is about admitting tuples of names in prefix subjects.

More interestingly we demonstrate that we can accommodate a variety of structures for communication channels; in general these can be any kind of data, and substitution can include any kind of computation on these structures. This indicates that the word "substitution" may be a misnomer — a better word may be "effect" — though we keep it to conform with our earlier work. The examples below use default values for the parameters where $\mathbf{A} = \{\mathbf{1}\}$, $\mathbf{C} = \{\top, \bot\}$ and $M \leftrightarrow N = \top$ iff $M = N$, otherwise $\bot$. We let $\mathbf{1} \vdash \top$ and $\mathbf{1} \nvdash \bot$. We also let $\underline{\propto} = \overline{\propto} = \mathcal{S} \times \mathcal{S}$, $\mathcal{S}_\nu = \mathcal{S}_\mathcal{N}$, and let $\prec$ be the identity on $\mathcal{S}$, unless otherwise defined. Finally we let $\text{MATCH}(M, \widetilde{x}, X) = \emptyset$ where not otherwise defined, we write $\preceq$ for the subterm (non-strict) partial order, and we use the standard notion of simultaneous substitution unless otherwise stated.

Example 2 (Convergent rewrite system on terms). We here consider deterministic computations specified using a rewrite system on terms containing names. This example highlights how a notion of substitution restricts the possible choices for $\text{VARS}(X)$; see Examples 3 and 4 for two concrete instances.

Let Σ be a sorted signature, and $\cdot\Downarrow$ be normalization with respect to a convergent rewrite system on the nominal term algebra over $\mathcal{N}$ generated by the signature Σ. We write ρ for sort-preserving capture-avoiding simultaneous substitutions $\{^{\widetilde{M}}/_{\widetilde{a}}\}$ where every M_i is in normal form; here $n(\rho) = \text{n}(\widetilde{M}, \widetilde{a})$. A pattern (term) X is stable if for all ρ, $X\rho\Downarrow = X\rho$. The patterns include the stable patterns Y and all instances X thereof (i.e., where $X = Y\rho$); such an X can bind any names occurring in Y but not in ρ.

REWRITE($\Downarrow$)
$\mathbf{T} = \mathbf{X} = \text{range}(\Downarrow)$
$M[\widetilde{y} := \widetilde{L}] = M\{^{\widetilde{L}}/_{\widetilde{y}}\}\Downarrow$
$\text{VARS}(X) = \bigcup\{\mathcal{P}(\text{n}(Y) \setminus \text{n}(\rho)) : Y \text{ stable} \wedge X = Y\rho\}$
$\text{MATCH}(M, \widetilde{x}, X) = \{\widetilde{L} : M = X\{^{\widetilde{L}}/_{\widetilde{x}}\}\}$

As a simple instance of Example 2, we may consider Peano arithmetic.

Example 3 (Peano arithmetic). Let $\mathcal{S} = \mathcal{S}_{\mathcal{N}} = \{\mathtt{nat}, \mathtt{chan}\}$. We take the signature consisting of the function symbols $\mathtt{zero} : \mathtt{nat}$, $\mathtt{succ} : \mathtt{nat} \to \mathtt{nat}$ and $\mathtt{plus} : \mathtt{nat} \times \mathtt{nat} \to \mathtt{nat}$. The rewrite rules $\mathtt{plus}(K, \mathtt{succ}(M)) \to \mathtt{plus}(\mathtt{succ}(K), M)$ and $\mathtt{plus}(K, \mathtt{zero}) \to K$ induce a convergent rewrite system $\Downarrow^{\mathrm{Peano}}$.

The stable terms are those that do not contain any occurrence of $\mathtt{plus}$. The construction of Example 2 yields that $\widetilde{x} \in \textsc{vars}(X)$ if $\widetilde{x} = \varepsilon$ (which matches only the term X itself), or if $\widetilde{x} = a$ and $X = \mathtt{succ}^n(a)$.

Writing i for $\mathtt{succ}^i(\mathtt{zero})$, the agent $(\nu a)(\overline{a}\ 2 \mid \underline{a}(\lambda y)\mathtt{succ}(y)\,.\,\overline{c}\ \mathtt{plus}(3, y))$ of $\mathbf{REWRITE}(\Downarrow^{\mathrm{Peano}})$ has one visible transition, with the label $\overline{c}\ 4$. In particular, the object of the label is $\mathtt{plus}(3, y)[y := 1] = \mathtt{plus}(3, y)\{{}^{1}/_{y}\}\Downarrow^{\mathrm{Peano}} = 4$.

Example 4 (Symmetric encryption). We can also consider variants on the construction in Example 2, such as a simple Dolev-Yao style [19] cryptographic message algebra for symmetric cryptography, where we ensure that the encryption keys of received encryptions can not be bound in input patterns, in agreement with cryptographic intuition.

Let $\mathcal{S} = \mathcal{S}_{\mathcal{N}} = \{\mathtt{message}, \mathtt{key}\}$, and consider the term algebra over the signature with the two function symbols $\mathtt{enc}, \mathtt{dec}$ of sort $\mathtt{message} \times \mathtt{key} \to \mathtt{message}$. The rewrite rule $\mathtt{dec}(\mathtt{enc}(M, K), K) \to M$ induces a convergent rewrite system $\Downarrow^{\mathrm{enc}}$, where the terms not containing $\mathtt{dec}$ are stable.

The construction of Example 2 yields that $\widetilde{x} \in \textsc{vars}(X)$ if $\widetilde{x} \subseteq \mathrm{n}(X)$ are pair-wise different and no x_i occurs as a subterm of a $\mathtt{dec}$ in X. This construction would permit to bind the keys of an encrypted message upon reception, e.g. $\underline{a}(\lambda m, k)\mathtt{enc}(m, k)\,.\,P$ would be allowed although it does not make cryptographic sense. Therefore we further restrict $\textsc{vars}(X)$ to those sets not containing names that occur in key position in X, thus disallowing the binding of k above.

SYMSPI
As $\mathbf{REWRITE}(\Downarrow^{\mathrm{enc}})$, except $\textsc{vars}(X) = \mathcal{P}(n(X) \setminus \{a : a \preceq \mathtt{dec}(Y_1, Y_2) \preceq X \vee (a \preceq Y_2 \wedge \mathtt{enc}(Y_1, Y_2) \preceq X)\})$

As an example, the agent $(\nu a, k)(\overline{a}\ \mathtt{enc}(\mathtt{enc}(M, l), k) \mid \underline{a}(\lambda y)\mathtt{enc}(y, k)\,.\,\overline{c}\ \mathtt{dec}(y, l))$ has a visible transition with label $\overline{c}\ M$.

Example 5 (Pattern-matching spi-calculus). A more advanced version of Example 4 is the treatment of data in the pattern-matching spi-calculus [4], to which we refer for more examples and motivations of the definitions below. Features of the calculus includes a non-homomorphic definition of substitution that does not preserve sorts, and a sophisticated way of computing permitted pattern variables. This example highlights the flexibility of sorted psi-calculi in that such a specialized modelling language can be directly represented, in a form that is very close to the original.

We start from the term algebra T_Σ over the unsorted signature Σ consisting of the function symbols $()$, $(\cdot,\cdot)$, $\mathtt{eKey}(\cdot)$, $\mathtt{dKey}(\cdot)$, $\mathtt{enc}(\cdot,\cdot)$ and $\mathtt{enc}^{-1}(\cdot,\cdot)$. The operation $\mathtt{enc}^{-1}$ is "encryption with the inverse key", which is only permitted to occur in patterns. We add a sort system on T_Σ where $\mathtt{impl}$ denotes implementable terms not containing $\mathtt{enc}^{-1}$, and $\mathtt{pat}$ those that may only be used in patterns. The sort $\bot$ denotes ill-formed terms, which do not occur in well-formed processes. Substitution is defined homomorphically on the term algebra, except for $\mathtt{enc}^{-1}(M_1,M_2)\sigma$ which is $\mathtt{enc}(M_1\sigma,\mathtt{eKey}(N))$ when $M_2\sigma=\mathtt{dKey}(N)$, and $\mathtt{enc}^{-1}(M_1\sigma,M_2\sigma)$ otherwise. We let $\Vdash\subset\mathcal{P}(T_\Sigma)\times\mathcal{P}(T_\Sigma)$ be deducibility in the Dolev-Yao message algebra (for the precise definition, see [4]). The definition of $\textsc{vars}(X)$ below allows to bind only those names that can be deduced from X and the other names occurring in X. This excludes binding an unknown key, like in Example 4.

PMSPI

$\mathbf{T}=\mathbf{X}=T_\Sigma$
$\mathcal{S}_\mathcal{N}=\{\mathtt{impl}\}\qquad\mathcal{S}=\{\mathtt{impl},\mathtt{pat},\bot\}$
$\prec=\overline{\propto}=\{(\mathtt{impl},\mathtt{impl})\}$
$\underline{\propto}=\{(\mathtt{impl},\mathtt{impl}),(\mathtt{impl},\mathtt{pat})\}$
$\textsc{sort}(M)=\mathtt{impl}$ if $\forall N_1,N_2.\ \mathtt{enc}^{-1}(N_1,N_2)\not\preceq M$
$\textsc{sort}(M)=\bot$ if $\exists N_1,N_2.\ \mathtt{enc}^{-1}(N_1,\mathtt{dKey}(N_2))\preceq M$
$\textsc{sort}(M)=\mathtt{pat}$ otherwise
$\textsc{match}(M,\widetilde{x},X)=\{\widetilde{L}\ :\ M=X[\widetilde{x}:=\widetilde{L}]\}$
$\textsc{vars}(X)=\{S\subseteq\mathrm{n}(X)\ :\ ((\mathrm{n}(X)\setminus S)\cup\{X\})\Vdash S\}$

As an example, consider the following transitions in **PMSPI**:

$$\begin{aligned}(\nu a,k,l)(\overline{a}\ \mathtt{enc}(\mathtt{dKey}(l),\mathtt{eKey}(k)).\overline{a}\ \mathtt{enc}(M,\mathtt{eKey}(l))\\ \mid \underline{a}(\lambda y)\mathtt{enc}(y,\mathtt{eKey}(k))\,.\,\underline{a}(\lambda z)\mathtt{enc}^{-1}(z,y)\,.\,\overline{c}\ z)\\ \xrightarrow{\tau}(\nu a,k,l)(\overline{a}\ \mathtt{enc}(M,\mathtt{eKey}(l))\mid \underline{a}(\lambda z)\mathtt{enc}(z,\mathtt{eKey}(l))\,.\,\overline{c}\ z)\\ \xrightarrow{\tau}(\nu a,k,l)\overline{c}\ M.\end{aligned}$$

Note that $\sigma=[y:=\mathtt{dKey}(l)]$ resulting from the first input changed the sort of the second input pattern: $\textsc{sort}(\mathtt{enc}^{-1}(z,y))=\mathtt{pat}$, but $\textsc{sort}(\mathtt{enc}^{-1}(z,y)\sigma)=\textsc{sort}(\mathtt{enc}(z,\mathtt{eKey}(l)))=\mathtt{impl}$. However, this is permitted by Definition 4, since $\mathtt{impl}\leq\mathtt{pat}$.

Example 6 (Nondeterministic computation). The previous examples considered total deterministic notions of computation on the term language. Here we consider a data term language equipped with partial non-deterministic evaluation: a lambda calculus with the erratic choice operator $\cdot[]\cdot$. Due to the non-determinism and partiality, evaluation cannot be part of the substitution function. Instead, the MATCH function collects all evaluations of the received term, which are

non-deterministically selected from by the In rule. This example also highlights the use of object languages with binders, a common application of nominal logic.

We let substitution on terms be the usual capture-avoiding syntactic replacement, and define reduction contexts $\mathcal{R} ::= [\,] \mid \mathcal{R}\ M \mid (\boldsymbol{\lambda} x.M)\ \mathcal{R}$. Reduction $\rightarrow$ is the smallest pre-congruence for reduction contexts that contain the rules for β-reduction ($\boldsymbol{\lambda} x.M\ N \rightarrow M[x := N]$) and $\cdot \,[\!]\, \cdot$ (namely $M_1 \,[\!]\, M_2 \rightarrow M_i$ if $i \in \{1, 2\}$). We use the single-name patterns of Example 1, but include evaluation in matching.

NDLAM

$\mathcal{S}_{\mathcal{N}} = \mathcal{S} = \{s\}$ $\qquad$ $\mathbf{X} = \mathcal{N}$

$M ::= a \mid M\ M \mid \boldsymbol{\lambda} x.M \mid M \,[\!]\, M$

where x binds into M in $\boldsymbol{\lambda} x.M$

$\text{MATCH}(M, x, x) = \{N \ :\ M \rightarrow^* N \not\rightarrow\}$

As an example, the agent $(\nu a)(\underline{a}(y) \,.\, \overline{c}\ y \,.\, \mathbf{0} \mid \overline{a}\ ((\boldsymbol{\lambda} x.x\ x) \,[\!]\, (\boldsymbol{\lambda} x.x)) \,.\, \mathbf{0})$ has two visible transitions, with labels $\overline{c}\ \boldsymbol{\lambda} x.x\ x$ and $\overline{c}\ \boldsymbol{\lambda} x.x$.

4 Conclusions and Further Work

We have described two features that taken together significantly improve the precision of applied process calculi: generalised pattern matching and substitution, which allow us to model computations on an arbitrary data term language, and a sort system which allows us to remove spurious data terms from consideration and to ensure that channels carry data of the appropriate sort. The well-formedness of processes is thereby guaranteed to be preserved by transitions. We have given examples of these features, ranging from the simple polyadic pi-calculus to the highly specialized pattern-matching spi-calculus, in the psi-calculi framework.

The meta-theoretic results carry over from the original psi formulations, and many have been machine-checked in Isabelle. We have also developed a tool for sorted psi-calculi [20], the Psi-calculi Workbench (Pwb), which provides an interactive simulator and automatic bisimulation checker. Users of the tool need only implement the parameters of their psi-calculus instances, supported by a core library.

Future work includes developing a symbolic semantics with pattern matching. For this, a reformulation of the operational semantics in the late style, where input objects are not instantiated until communication takes place, is necessary. We also aim to extend the use of sorts and generalized pattern matching to other variants of psi-calculi, including higher-order psi calculi [21] and reliable broadcast psi-calculi [22]. As mentioned in Sect. 2.6, further developments in Nominal Isabelle are needed for mechanizing theories with arbitrary but fixed sortings.

References

1. Bengtson, J., Johansson, M., Parrow, J., Victor, B.: Psi-calculi: a framework for mobile processes with nominal data and logic. LMCS 7(1:11) (2011)
2. Pitts, A.M.: Nominal logic, a first order theory of names and binding. Inf. Comput. **186**, 165–193 (2003)
3. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: Proceedings of POPL, pp. 104–115. ACM, January 2001
4. Haack, C., Jeffrey, A.: Pattern-matching spi-calculus. Inf. Comput. **204**(8), 1195–1263 (2006)
5. Gelernter, D.: Generative communication in Linda. ACM TOPLAS **7**(1), 80–112 (1985)
6. Schmitt, A., Stefani, J.-B.: The KELL calculus: a family of higher-order distributed process calculi. In: Priami, C., Quaglia, P. (eds.) GC 2004. LNCS, vol. 3267, pp. 146–178. Springer, Heidelberg (2005)
7. Milner, R.: The polyadic π-calculus: a tutorial. In: Bauer, F.L., Brauer, W., Schwichtenberg, H. (eds.) Logic and Algebra of Specification. NATO ASI, vol. 94, pp. 203–246. Springer, Heidelberg (1993)
8. Hüttel, H.: Typed ψ-calculi. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 265–279. Springer, Heidelberg (2011)
9. Blanchet, B.: Using Horn clauses for analyzing security protocols. In Cortier, V., Kremer, S., eds.: Formal Models and Techniques for Analyzing Security Protocols. Cryptology and Information Security Series, vol. 5, pp. 86–111. IOS Press (2011)
10. Fournet, C., Gordon, A.D., Maffeis, S.: A type discipline for authorization policies. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 141–156. Springer, Heidelberg (2005)
11. Given-Wilson, T., Gorla, D., Jay, B.: Concurrent pattern calculus. In: Calude, C.S., Sassone, V. (eds.) TCS 2010. IFIP AICT, vol. 323, pp. 244–258. Springer, Heidelberg (2010)
12. Fournet, C., Gonthier, G.: The reflexive CHAM and the join-calculus. In: Proceedings of the POPL, pp. 372–385 (1996)
13. Gabbay, M.J., Pitts, A.M.: A new approach to abstract syntax with variable binding. Formal Aspects Comput. **13**, 341–363 (2001)
14. Johansson, M., Victor, B., Parrow, J.: Computing strong and weak bisimulations for psi-calculi. J. Logic Algebraic Program. **81**(3), 162–180 (2012)
15. Johansson, M., Bengtson, J., Parrow, J., Victor, B.: Weak equivalences in psi-calculi. In: Proceedings of LICS 2010, pp. 322–331. IEEE (2010)
16. Milner, R.: Communication and Concurrency. Prentice-Hall Inc., Upper Saddle River (1989)
17. Sangiorgi, D.: Expressing mobility in process algebras: first-order and higher-order paradigms. Ph.D thesis, University of Edinburgh, CST-99-93 (1993)
18. Carbone, M., Maffeis, S.: On the expressive power of polyadic synchronisation in π-calculus. Nord. J. Comput. **10**(2), 70–98 (2003)
19. Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE Trans. Inf. Theor. **29**(2), 198–208 (1983)
20. Borgström, J., Gutkovas, R., Rodhe, I., Victor, B.: A parametric tool for applied process calculi. In: Proceedings of the 13th International Conference on Application of Concurrency to System Design (ACSD'13). IEEE (2013)
21. Parrow, J., Borgström, J., Raabjerg, P., Åman Pohjola, J.: Higher-order psi-calculi. Mathematical Structures in Computer Science FirstView (June 2013)

22. Åman Pohjola, J., Borgström, J., Parrow, J., Raabjerg, P., Rodhe, I.: Negative premises in applied process calculi. Technical Report 2013-014, Department of Information Technology, Uppsala University (2013)