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, Sweden

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. 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 $M \mathrel{\dot{\leftrightarrow}} K$. Second, N must match the pattern X.

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

Assertions and conditions can, in general, form any logical theory. Also the data terms can be drawn from an arbitrary set. One of our major contributions has been to pinpoint the precise requirements on the data terms and logic for a calculus to be useful in the sense that the natural formulation of bisimulation satisfies the expected algebraic laws (see Section 2). It turns out that it is necessary to view the terms and logics as *nominal* [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 $\textsc{match}(N, \widetilde{x}, X)$ then N matches the pattern X by instantiating $\widetilde{x}$ to $\widetilde{L}$. The receiving agent $\underline{K}(\lambda\widetilde{x})X \,.\, Q$ then continues as $Q[\widetilde{x} := \widetilde{L}]$.

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

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

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

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

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

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, Scala, F#). 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.

Sorts for the pi-calculus were first described by Milner [6]. Hüttel's typed psi-calculi [7] 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 [8] 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. [9] 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 [10] and the join-calculus [11], 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 Section 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 Section 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, 12] 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 $\dot{\leftrightarrow} : \mathbf{T}\times\mathbf{T} \to \mathbf{C}$, assertion composition $\otimes : \mathbf{A}\times\mathbf{A} \to \mathbf{A}$, the unit assertion $\mathbf{1}$, and the entailment relation $\vdash \subseteq \mathbf{A} \times \mathbf{C}$.*

The binary functions $\dot{\leftrightarrow}, \otimes$ 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 pattern-matching function MATCH, which is used when the input takes place, and a function VARS which yields the possible combinations of binding names in the pattern. 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*

$$\begin{array}{rll} \textsc{match} : & \mathbf{T} \times \mathcal{N}^* \times \mathbf{X} \to \mathcal{P}(\mathbf{T}^*) & \textit{Pattern matching} \\ \textsc{vars} : & \mathbf{X} \to \mathcal{P}(\mathcal{P}(\mathcal{N})) & \textit{Pattern variables} \end{array}$$

Intuitively, if $\widetilde{L} \in \textsc{match}(N, \widetilde{x}, X)$ then an output of the term N matches an input with the pattern X, binding $\widetilde{x}$, and the receiving agent continues after substituting $\widetilde{L}$ for $\widetilde{x}$.

The VARS operator gives the possible sets of names in a pattern which are bound by an input prefix. For example, an input prefix with a pairing pattern $\langle x, y\rangle$ may bind both x and y, only one of them, or none, so $\textsc{vars}(\langle x, y\rangle) = \{\{x, y\}, \{x\}, \{y\}, \{\}\}$. This way, we can let the input prefix $\underline{c}(\lambda x)\langle x, y\rangle$ only match pairs where the second argument is the name y. To model a calculus where input patterns cannot be selective in this way, we may instead define $\textsc{vars}(\langle x, y\rangle) = \{\{x, y\}\}$. This ensures that input prefixes that use the pattern

$\langle x, y\rangle$ must be of the form $\underline{M}(\lambda x, y)\langle x, y\rangle$, where both x and y are bound. Another use for $\textsc{vars}$ is to exclude the binding of terms in certain positions, such as the keys of cryptographic messages (cf. Example 5).

Requirements on $\textsc{vars}$ and $\textsc{match}$ 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* $\textsc{sort} : \mathcal{N} \uplus \mathbf{T} \uplus \mathbf{X} \rightarrow \mathcal{S}$, *and the four compatibility predicates*

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

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

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

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

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

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* $\textsc{match}$ *we require that if* $\widetilde{x} \in \textsc{vars}(X)$ *are distinct and* $\widetilde{N} \in \textsc{match}(M, \widetilde{x}, X)$ *then it must hold that* $[\widetilde{x} := \widetilde{N}]$ *is a substitution, that* $\mathrm{n}(\widetilde{N}) \subseteq \mathrm{n}(M) \cup (\mathrm{n}(X) \setminus \widetilde{x})$*, and that for all name swappings* $(\widetilde{x}\ \widetilde{y})$ *we have* $\widetilde{N} \in \textsc{match}(M, \widetilde{y}, (\widetilde{x}\ \widetilde{y})X)$ *(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 $\widetilde{x} \subseteq \mathrm{n}(N)$), 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 [13], 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 $\textsc{vars}(a) = \{a\}$, meaning that the pattern variable must always occur bound, and $\textsc{match}(M, a, a) = \{M\}$ if $\textsc{sort}(a) \prec \textsc{sort}(M)$. On patterns we define substitution as $a\sigma = a$ when $a\#\sigma$.

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
$(\!\vert\Psi\vert\!)$	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 $\text{SORT}(M)\ \overline{\propto}\ \text{SORT}(N)$. 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 $\text{SORT}(M)\ \underline{\propto}\ \text{SORT}(X)$. 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 $(\!\vert\Psi\vert\!)$ contributes Ψ to its environment. We often write **if** φ **then** P for **case** $\varphi : P$, and nothing or $\mathbf{0}$ for the empty case statement **case**.

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] and in the appendix of this paper. 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

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

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 Section 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.*

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

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 [14] 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 [15], the unsorted and the sorted polyadic pi-calculus [6, 16], and the polyadic synchronization pi-calculus [17] 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. Details can be found in the appendix.

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 $M \mathrel{\dot{\leftrightarrow}} N = \top$ 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 $\textsc{match}(M, \widetilde{x}, X) = \emptyset$ 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 $\textsc{vars}(X)$; see Example 3 and Example 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 $n(\rho) = \mathrm{n}(\widetilde{M}, \widetilde{a})$. 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($\Downarrow$)
$\mathbf{T} = \mathbf{X} = \mathrm{range}(\Downarrow)$
$M[\widetilde{y} := \widetilde{L}] = M\{^{\widetilde{L}}/_{\widetilde{y}}\}\Downarrow$
$\textsc{vars}(X) = \bigcup\{\mathcal{P}(\mathrm{n}(Y) \setminus \mathrm{n}(\rho)) : Y \text{ stable} \wedge X = Y\rho\}$
$\textsc{match}(M, \widetilde{x}, X) = \{\widetilde{L} : M = X\{^{\widetilde{L}}/_{\widetilde{x}}\}\}$

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

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

The stable terms are those that do not contain any occurrence of $\texttt{plus}$. 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 $X = \texttt{succ}^n(a)$.

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

Example 4 (Symmetric encryption). We can also consider variants on the construction in Example 2, such as a simple Dolev-Yao style [18] 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 $\mathcal{S} = \mathcal{S}_\mathcal{N} = \{\texttt{message}, \texttt{key}\}$, and consider the term algebra over the signature with the two function symbols $\texttt{enc}, \texttt{dec}$ of sort $\texttt{message} \times \texttt{key} \to \texttt{message}$. The rewrite rule $\texttt{dec}(\texttt{enc}(M, K), K) \to M$ induces a convergent rewrite system $\Downarrow^{\text{enc}}$, where the terms not containing $\texttt{dec}$ are stable.

The construction of Example 2 yields that $\widetilde{x} \in \textsc{vars}(X)$ if $\widetilde{x} \subseteq \text{n}(X)$ are pair-wise different and no x_i occurs as a subterm of a $\texttt{dec}$ in X. This construction would permit to bind the keys of an encrypted message upon reception, e.g. $\underline{a}(\lambda m, k)\texttt{enc}(m, k) \,.\, P$ 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 $\mathbf{REWRITE}(\Downarrow^{\text{enc}})$, except
$\textsc{vars}(X) = \mathcal{P}(n(X) \setminus \{a : a \preceq \texttt{dec}(Y_1, Y_2) \preceq X \vee$ $(a \preceq Y_2 \wedge \texttt{enc}(Y_1, Y_2) \preceq X)\})$

As an example, the agent
$(\nu a, k)(\overline{a}\, \texttt{enc}(\texttt{enc}(M, l), k) \mid \underline{a}(\lambda y)\texttt{enc}(y, k) \,.\, \overline{c}\, \texttt{dec}(y, l))$ 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 $()$, $(\cdot, \cdot)$, $\texttt{eKey}(\cdot)$, $\texttt{dKey}(\cdot)$, $\texttt{enc}(\cdot, \cdot)$ and $\texttt{enc}^{-1}(\cdot, \cdot)$. 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 `impl` denotes implementable terms not containing $\mathtt{enc}^{-1}$, and `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 $\textsc{match}$ function collects all evaluations of the received term, which are non-deterministically selected from by the $\textsc{In}$ rule. This example also highlights the use of object languages with binders, a common application of nominal logic.

We let substitution on terms be the usual capture-avoiding syntactic replacement, and define reduction contexts $\mathcal{R} ::= [\,] \mid \mathcal{R}\ M \mid (\boldsymbol{\lambda} x.M)\ \mathcal{R}$. Reduction $\rightarrow$

is the smallest pre-congruence for reduction contexts that contain the rules for β-reduction ($\boldsymbol{\lambda}x.M\ N \to M[x := N]$) and $\cdot \,[\!]\, \cdot$ (namely $M_1 \,[\!]\, M_2 \to M_i$ if $i \in \{1,2\}$). We use the single-name patterns of Example 1, but include evaluation in matching.

NDLAM

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

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

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

$\textsc{match}(M, x, x) = \{N \ :\ M \to^* N \not\to\}$

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 Section 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. Information and Computation **186** (2003) 165–193
[3] Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: Proceedings of POPL '01, ACM (January 2001) 104–115
[4] Haack, C., Jeffrey, A.: Pattern-matching spi-calculus. Information and Computation **204**(8) (2006) 1195–1263
[5] Gelernter, D.: Generative communication in Linda. ACM TOPLAS **7**(1) (January 1985) 80–112
[6] Milner, R.: The polyadic π-calculus: A tutorial. In Bauer, F.L., Brauer, W., Schwichtenberg, H., eds.: Logic and Algebra of Specification. Volume 94 of Series F., NATO ASI, Springer (1993)
[7] Hüttel, H.: Typed psi-calculi. In Katoen, J.P., König, B., eds.: CONCUR 2011 – Concurrency Theory. Volume 6901 of LNCS., Springer (2011) 265–279
[8] Blanchet, B.: Using Horn clauses for analyzing security protocols. In Cortier, V., Kremer, S., eds.: Formal Models and Techniques for Analyzing Security Protocols. Volume 5 of Cryptology and Information Security Series. IOS Press (March 2011) 86–111
[9] Fournet, C., Gordon, A.D., Maffeis, S.: A type discipline for authorization policies. In Sagiv, M., ed.: Proc. of ESOP 2005. Volume 3444 of LNCS., Springer (2005) 141–156
[10] Given-Wilson, T., Gorla, D., Jay, B.: Concurrent pattern calculus. In Calude, C., Sassone, V., eds.: Theoretical Computer Science. Volume 323 of IFIP Advances in Information and Communication Technology. Springer (2010) 244–258
[11] Fournet, C., Gonthier, G.: The reflexive CHAM and the join-calculus. In: Proc. POPL. (1996) 372–385
[12] Gabbay, M.J., Pitts, A.M.: A new approach to abstract syntax with variable binding. Formal Aspects of Computing **13** (2001) 341–363
[13] Johansson, M., Victor, B., Parrow, J.: Computing strong and weak bisimulations for psi-calculi. Journal of Logic and Algebraic Programming **81**(3) (2012) 162–180
[14] Johansson, M., Bengtson, J., Parrow, J., Victor, B.: Weak equivalences in psi-calculi. In: Proc. of LICS 2010, IEEE (2010) 322–331
[15] Milner, R.: Communication and Concurrency. Prentice-Hall, Inc. (1989)
[16] Sangiorgi, D.: Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms. PhD thesis, University of Edinburgh (1993) CST-99-93 (also published as ECS-LFCS-93-266).
[17] Carbone, M., Maffeis, S.: On the expressive power of polyadic synchronisation in π-calculus. Nordic Journal of Computing **10**(2) (2003) 70–98
[18] Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE Transactions on Information Theory **29**(2) (1983) 198–208
[19] McCarthy, J.: A basis for a mathematical theory of computation. Computer Programming and Formal Systems (1963) 33–70
[20] Borgström, J., Gutkovas, R., Rodhe, I., Victor, B.: A parametric tool for applied process calculi. Accepted for publication in ACSD 2013. Available from `http://www.it.uu.se/research/group/mobility/applied/psiworkbench`.
[21] Parrow, J., Borgström, J., Raabjerg, P., Åman Pohjola, J.: Higher-order psi-calculi (2012) Accepted for publication in MSCS. Available from `http://www.it.uu.se/research/group/mobility`.
[22] Pohjola, J.Å., Borgström, J., Parrow, J., Raabjerg, P., Rodhe, I.: Negative premises in applied process calculi. submitted (2013)
[23] Åman Pohjola, J.: Isabelle proof scripts for sorted psi-calculi (2012)

A Complete Definitions

A.1 Frames and transitions

Each agent affects other agents that are in parallel with it via its frame, which may be thought of as the collection of all top-level assertions of the agent. A *frame* F is an assertion with local names, written $(\nu\widetilde{b})\Psi$ where $\widetilde{b}$ is a sequence of names that bind into the assertion Ψ. We use F, G to range over frames, and identify alpha-equivalent frames. We overload $\otimes$ to frame composition defined by $(\nu\widetilde{b_1})\Psi_1\otimes(\nu\widetilde{b_2})\Psi_2 = (\nu\widetilde{b_1}\widetilde{b_2})(\Psi_1\otimes\Psi_2)$ where $\widetilde{b_1}\#\widetilde{b_2}, \Psi_2$ and vice versa. We write $\Psi\otimes F$ to mean $(\nu\epsilon)\Psi\otimes F$, and $(\nu c)((\nu\widetilde{b})\Psi)$ for $(\nu c\widetilde{b})\Psi$.

Intuitively a condition is entailed by a frame if it is entailed by the assertion and does not contain any names bound by the frame, and two frames are equivalent if they entail the same conditions. Formally, we define $F \vdash \varphi$ to mean that there exists an alpha variant $(\nu\widetilde{b})\Psi$ of F such that $\widetilde{b}\#\varphi$ and $\Psi \vdash \varphi$. We also define $F \simeq G$ to mean that for all φ it holds that $F \vdash \varphi$ iff $G \vdash \varphi$.

Definition 8 (Frames and Transitions). *The* frame $\mathcal{F}(P)$ of an agent P *is defined inductively as follows:*

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

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

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

The operational semantics is the same as for the original psi-calculi, except for the use of MATCH in rule IN. We identify alpha-equivalent agents and transitions (see [1] for details). In a transition the names in $\mathrm{bn}(\alpha)$ bind into both the action object and the derivative, therefore $\mathrm{bn}(\alpha)$ is in the support of α but not in the support of the transition. This means that the bound names can be chosen fresh, substituting each occurrence in both the action and the derivative.

As shown in the introduction, well-formedness is not preserved by transitions in the original psi-calculi. However, in sorted psi-calculi the usual well-formedness preservation result holds.

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

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

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

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

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

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

Table 1. Operational semantics.

B Meta-theory

We begin by recollecting the definition of strong labelled bisimulation on well-formed agents from Bengtson et al. [1], to which we refer for examples, intuitions and the exact formulations of theorems.

Definition 9 (Strong bisimulation). *A* strong bisimulation $\mathcal{R}$ *is a ternary relation on assertions and pairs of agents such that* $\mathcal{R}(\Psi, P, Q)$ *implies the following four statements.*

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

We define bisimilarity $P \dot{\sim}_\Psi Q$ *to mean that there is a bisimulation* $\mathcal{R}$ *such that* $\mathcal{R}(\Psi, P, Q)$, *and write* $\dot{\sim}$ *for* $\dot{\sim}_{\mathbf{1}}$. *Strong* congruence *is defined by* $P \sim_\Psi Q$ *iff for* *all sequences $\widetilde{\sigma}$ of substitutions it holds that $P\widetilde{\sigma} \mathrel{\dot{\sim}_\Psi} Q\widetilde{\sigma}$. We write $P \sim Q$ for $P \sim_{\mathbf{1}} Q$.*

There is also a notion of weak bisimilarity ($\dot{\approx}$) where τ-transitions cannot be observed; see [14] for its precise definition.

We seek to establish the following properties of bisimulation.

Theorem 4 (Congruence properties of $\dot{\sim}$). *For all Ψ:*

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

Definition 10. *$P \sim_\Psi Q$ means that for all sequences σ of substitutions it holds that $P\sigma \mathrel{\dot{\sim}_\Psi} Q\sigma$, and we write $P \sim Q$ for $P \sim_{\mathbf{1}} Q$.*

We seek to establish the following properties of bisimulation congruence.

Theorem 5. *Strong congruence $\sim_\Psi$ is a congruence for all Ψ.*

Theorem 6 (Structural equivalence). *$a\#Q, \widetilde{x}, M, N, X, \widetilde{\varphi}$ implies*

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

B.1 Trivially sorted calculi

A *trivially* sorted psi calculus is one where $\prec = \underline{\propto} = \overline{\propto} = \mathcal{S} \times \mathcal{S}$ and $\mathcal{S}_\nu = \mathcal{S}$, i.e., the sorts do not affect how terms are used in communications and substitutions. For technical reasons we here first establish the expected algebraic properties of bisimilarity and its induced congruence in trivially sorted psi-calculi, and then investigate how these results are lifted to arbitrary sorted calculi.

Theorem 7. *Theorem 4, Theorem 5, and Theorem 6 hold for trivially sorted psi-calculi.*

Additionally, the corresponding results on the algebraic properties of weak bisimilarity and its induced congruence as defined and presented in [14] also hold for trivially sorted psi-calculi.

These results have all been machine-checked in Isabelle [23]. As indicated these proof scripts apply only to trivially sorted calculi, meaning that the only extension to our previous formulation is in the input rule which now uses MATCH. We have also machine-checked Theorem 1 (preservation of well-formedness) in this setting.

The restriction to trivially sorted calculi is a consequence of technicalities in Nominal Isabelle: it requires every name sort to be declared individually, and there are no facilities to reason parametrically over the set of name sorts. There is also a discrepancy in that our definitions in Section 2 considers only well-sorted alpha-renamings, while the mechanisation works with a single sort of names and thus allows for ill-sorted alpha-renamings. This is only a technicality, since every use of alpha-renaming in the formal proofs is to ensure that the bound names in patterns and substitutions avoid other bound names—thus, whenever we may work with an ill-sorted renaming, there would be a well-sorted renaming that suffices for the task.

B.2 Arbitrary sorted psi-calculi

We here extend the results of Theorem 7 to arbitrary sorted psi-calculi. The idea is to introduce an explicit error element $\bot$ corresponding to an ill-sorted term, pattern, condition and assertion. For technical reasons we must also include one extra condition `fail` (in order to ensure the compositionality of $\otimes$) and in the patterns we need different error elements with different support (in order to ensure the preservation of pattern variables under substitution).

Let $I = (\mathbf{T}, \mathbf{X}, \mathbf{C}, \mathbf{A})$ be a sorted psi-calculus. We construct a trivially sorted psi-calculus $U(I)$ with one extra sort, `error`. The parameters of $U(I)$ are defined by $U(I) = (\mathbf{T} \cup \{\bot\}, \mathbf{X} \cup \{(\bot, S) \;:\; S \subset_{\text{fin}} \mathcal{N}\}, \mathbf{C} \cup \{\bot, \mathtt{fail}\}, \mathbf{A} \cup \{\bot\})$. Here $\bot$ and `fail` are constant symbols with empty support of sort `error`. $\bot$ is not a channel, never entailed, matches nothing and entails nothing but `fail`. `fail` is entailed only by $\bot$. We define $\Psi \otimes \bot = \bot \otimes \Psi = \bot$ for all Ψ, and otherwise $\otimes$ is as in I. MATCH and $\dot{\leftrightarrow}$ in $U(I)$ are the same as in I, and for $\Psi \neq \bot$ we let $\Psi \vdash \varphi$ in $U(I)$ iff $\Psi \vdash \varphi$ in I. Substitution is then defined in $U(I)$ as follows:

$$T[\widetilde{a} := \widetilde{N}]_{U(I)} = \begin{cases} T[\widetilde{a} := \widetilde{N}]_I & \text{if } \textsc{sort}(a_i) \prec_I \textsc{sort}(N_i) \text{ and} \\ & \quad N_i \neq \bot \text{ for all } i, \text{ and } T \neq (\bot, S) \\ (\bot, S \setminus \widetilde{a}) & \text{if } T = (\bot, S) \text{ is a pattern} \\ (\bot, \bigcup\bigcup \textsc{vars}(T)) & \text{otherwise, if } T \text{ is a pattern} \\ \bot & \text{otherwise} \end{cases}$$

Lemma 1. *Assume that P and Q are well-formed processes in I, and $\Psi \neq \bot$.*

1. *$U(I)$ as defined above is a sorted psi-calculus.*
2. *P and Q are well-formed when considered as processes of $U(I)$.*
3. *$\Psi \rhd P \xrightarrow{\alpha} P'$ in $U(I)$ iff $\Psi \rhd P \xrightarrow{\alpha} P'$ in I.*
4. *$P \dot{\sim}_\Psi Q$ in I iff $P \dot{\sim}_\Psi Q$ in $U(I)$, and $P \dot{\approx}_\Psi Q$ in I iff $P \dot{\approx}_\Psi Q$ in $U(I)$.*

With Lemma 1, we can lift the congruence and the structural equivalence results for trivially sorted psi-calculi to arbitrary sorted calculi:

Theorem 8. *All clauses of Theorem 7 pertaining to strong bisimilarity, strong congruence and weak bisimilarity are valid in all sorted psi-calculi.*

Proof. We show only the proofs for strong congruence and commutativity of the parallel operator. The other proofs are analogous.

Fix a sorted psi-calculus I. Assume $P \mathrel{\dot{\sim}_\Psi} Q$ holds in I. By Lemma 1.4, $P \mathrel{\dot{\sim}_\Psi} Q$ holds in $U(I)$. Theorem 7 thus yields $P \mid R \mathrel{\dot{\sim}_\Psi} Q \mid R$ in $U(I)$, and Lemma 1.4 yields the same in I.

Let P and Q be well-formed in I and $\Psi \neq \bot$. By Theorem 7, $P \mid Q \sim_\Psi Q \mid P$ holds in $U(I)$. By Definition 9, $(P \mid Q)\widetilde{\sigma} \mathrel{\dot{\sim}_\Psi} (Q \mid P)\widetilde{\sigma}$ in $U(I)$ for all $\widetilde{\sigma}$. By Theorem 1, when $\widetilde{\sigma}$ is well-sorted then $(P \mid Q)\widetilde{\sigma}$ and $(Q \mid P)\widetilde{\sigma}$ are well-formed. By Lemma 1.4, $(P \mid Q)\widetilde{\sigma} \mathrel{\dot{\sim}_\Psi} (Q \mid P)\widetilde{\sigma}$ in I. $P \mid Q \sim_\Psi Q \mid P$ follows by definition.

This approach does not yield a similar result for strong congruence, since the closure of bisimilarity under well-sorted substitutions does not imply its closure under ill-sorted substitutions. Consider a well-sorted instance I such that $\mathbf{0} \sim_\Psi (\!|\mathbf{1}|\!)$. The corresponding property of $U(I)$ does not follow: if σ is ill-sorted then $\mathbf{1}\sigma = \bot$, but $\mathbf{0} \mathrel{\dot{\sim}_\Psi} (\!|\bot|\!)$ does not hold since only $\bot$ entails `fail`. Instead, we have performed a direct proof: it is identical, line by line, to the proof in the trivially sorted case.

A direct manual proof would also be necessary to obtain similar results for weak congruence for the same reasons as in the strong case. We have chosen not to pursue this result since the weak case is less straight-forward and a manual proof would be error-prone.

B.3 Pattern matching in original calculi

In many cases we can recover the pattern matching of the original psi-calculi.

Theorem 9. *Suppose* $(\mathbf{T}, \mathbf{C}, \mathbf{A})$ *is an original psi calculus [1] where pattern variables are preserved by substitutions* $(\mathrm{n}(N\sigma) \supseteq \mathrm{n}(N) \setminus \mathrm{n}(\sigma))$. *Let* $\mathbf{X} = \mathbf{T}$ *and* $\textsc{vars}(X) = \mathcal{P}(\mathrm{n}(X))$ *and* $\textsc{match}(M, \widetilde{x}, X) = \{\widetilde{L} : M = X[\widetilde{x} := \widetilde{L}]\}$ *and* $\mathcal{S} = \mathcal{S}_{\mathcal{N}} = \mathcal{S}_\nu = \{s\}$ *and* $\underline{\propto} = \overline{\propto} = \prec = \{(s, s)\}$ *and* $\textsc{sort} : \mathcal{N} \uplus \mathbf{T} \uplus \mathbf{X} \to \{s\}$*; then* $(\mathbf{T}, \mathbf{X}, \mathbf{C}, \mathbf{A})$ *is a sorted psi calculus.*

This result has been machined-checked in Isabelle for trivially-sorted calculi.

C Process Calculi Examples

We here consider some variants of popular process calculi. One main point of our work is that we can represent them directly as psi-calculi, without elaborate coding schemes. In our original psi-calculi we could in this way directly represent the monadic pi-calculus; for the other calculi presented below an unsorted psi-calculus would contain terms with no counterpart in the represented calculus, as explained in sections 1.2 and 1.3. We establish that our formulations enjoy a strong operational correspondence with the original calculus, under trivial mappings that merely specialise the original concrete syntax (e.g., the pi-calculus prefix $a(x)$ maps to $\underline{a}(\lambda x)x$ in psi). This correspondence is significantly stronger than standard correspondence results (cf. Gorla, I&C 208(9):1031-1053, 2010). Because of the simplicity of the mapping and the strength of the correspondence we use the phrasing that psi-calculi *represent* other process calculi, in contrast to *encoding* them.

C.1 Unsorted Polyadic pi-calculus

In the polyadic pi-calculus [6] the only values that can be transmitted between agents are tuples of names. Tuples cannot be nested. An input binds a tuple of distinct names and can only communicate with an output of equal length, resulting in a simultaneous substitution of all names. In the unsorted polyadic pi-calculus there are no further requirements on agents, in particular $a(x) \mid \overline{a}\langle y, z\rangle$ is a valid agent. This agent has no communication action since the lengths of the tuples mismatch.

PPI

$\mathbf{T} = \mathcal{N} \cup \{\langle\widetilde{a}\rangle : \widetilde{a} \subset_{\text{fin}} \mathcal{N}\}$
$\mathbf{C} = \{\top\} \cup \{a = b \mid a, b \in \mathcal{N}\}$
$\mathbf{X} = \{\langle\widetilde{a}\rangle : \widetilde{a} \subset_{\text{fin}} \mathcal{N} \wedge \widetilde{a} \text{ distinct}\}$
$\dot{\leftrightarrow} = \text{identity on names}$ $\qquad$ $\mathbf{1} \vdash a = a$
$\textsc{vars}(\langle\widetilde{a}\rangle) = \{\widetilde{a}\}$
$\textsc{match}(\langle\widetilde{a}\rangle, \widetilde{x}, \langle\widetilde{x}\rangle) = \{\widetilde{a}\}$ if $|\widetilde{a}| = |\widetilde{x}|$
$\mathcal{S}_{\mathcal{N}} = \{\texttt{chan}\}$ $\qquad$ $\mathcal{S} = \{\texttt{chan}, \texttt{tup}\}$
$\textsc{sort}(a) = \texttt{chan}$ $\qquad$ $\textsc{sort}(\langle\widetilde{a}\rangle) = \texttt{tup}$
$\mathcal{S}_{\nu} = \{\texttt{chan}\}$ $\qquad$ $\prec = \{(\texttt{chan}, \texttt{chan})\}$
$\overline{\propto} = \underline{\propto} = \{(\texttt{chan}, \texttt{tup})\}$

As an example the agent $\underline{a}(\lambda x, y)\langle x, y\rangle \,.\, \overline{a}\ \langle y\rangle \,.\, \mathbf{0}$ is well-formed, since $\texttt{chan} \mathrel{\underline{\propto}} \texttt{tup}$ and $\texttt{chan} \mathrel{\overline{\propto}} \texttt{tup}$, with $\textsc{vars}(\langle x, y\rangle) = \{\{x, y\}\}$. This demonstrates that **PPI** disallows anomalies such as nested tuples but does not enforce a sorting discipline to guarantee that names communicate tuples of the same length.

PPI is a direct representation of the polyadic pi-calculus as presented by Sangiorgi [16] (with replication instead of process constants). Let $[\![\cdot]\!]$ be the function that maps the polyadic pi-calculus to **PPI** processes, and analogously for labels: it maps summation to $\top$-guarded **case** statements, matching $[x = y]$ to $x = y$-guarded **case** statements, input prefix $a(\widetilde{x})$ to $\underline{a}(\lambda\widetilde{x})\widetilde{x}$, and is homomorphic over all other operators.

We obtain a strong operational correspondence between the calculi:

Theorem 10. *If P and Q are polyadic pi-calculus processes, then:*

1. *If $P \xrightarrow{\alpha} P'$ then $[\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$*

2. If $[\![P]\!] \xrightarrow{\alpha'} P''$ *then* $P \xrightarrow{\alpha} P'$ *where* $[\![\alpha]\!] = \alpha'$ *and* $[\![P']\!] = P''$

As it turns out, the representation of polyadic pi-calculus is surjective modulo strong bisimilarity. We show this by defining a translation $\overline{P}$ in the other direction: it is defined homomorphically except for **case** statements, which are mapped to a sum of possibly match-guarded processes, and $\overline{(\!|\mathbf{1}|\!)} = \mathbf{0}$.

Theorem 11. *If P is a* **PPI** *process, then* $P \mathrel{\dot\sim} [\![\overline{P}]\!]$.

A version of LINDA [5] can be obtained by adding a term $\bullet$ of a new sort `ts` denoting the tuple space, letting $\overline{\propto} = \underline{\propto} = \{(\mathtt{ts}, \mathtt{tup})\}$ and defining MATCH and VARS as in Theorem 9.

C.2 Sorted polyadic pi-calculus

Milner's classic sorting [6] regime for the polyadic pi-calculus ensures that pattern matching in inputs always succeeds, by enforcing that the length of the pattern is the same as the length of the received tuple. This is achieved as follows. Milner assumes a countable set of subject sorts S ascribed to names, and a partial function $\mathtt{ob} : \mathrm{S} \rightharpoonup \mathrm{S}^*$, assigning a sequence of object sorts to each sort. The intuition is that if a has sort s then any communication along a must be a tuple of sort $\mathtt{ob}(s)$. An agent is *well-sorted* if for any input prefix $a(b_1, \ldots b_n)$ it holds that a has some sort s where $\mathtt{ob}(s)$ is the sequence of sorts of $b_1, \ldots, b_n$ and similarly for output prefixes.

SORTEDPPI	
Everything as in **PPI** except:	
$\mathcal{S}_{\mathcal{N}} = \mathcal{S}_{\nu} = \mathrm{S}$	$\mathcal{S} = \mathrm{S}^*$
$\mathrm{SORT}(a_1, \ldots, a_n) = \mathrm{SORT}(a_1), \ldots, \mathrm{SORT}(a_n)$	
$\mathrm{MATCH}(\widetilde{a}, \widetilde{x}, \widetilde{x}) = \{\widetilde{a}\}$	if $\mathrm{SORT}(\widetilde{a}) = \mathrm{SORT}(\widetilde{x})$
$\prec = \{(s, s) : s \in \mathrm{S}\}$	$\overline{\propto} = \underline{\propto} = \{(s, \mathtt{ob}(s)) : s \in \mathrm{S}\}$

As an example, let $\mathrm{SORT}(a) = s$ with $\mathtt{ob}(s) = t_1, t_2$ and $\mathrm{SORT}(x) = t_1$ with $\mathtt{ob}(t_1) = t_2$ and $\mathrm{SORT}(y) = t_2$ then the agent $\underline{a}(\lambda x, y)(x, y) \,.\, \overline{x}\; y \,.\, \mathbf{0}$ is well-formed, since $s \mathrel{\underline{\propto}} t_1, t_2$ and $t_1 \mathrel{\overline{\propto}} t_2$, with $\mathrm{VARS}(x, y) = \{\{x, y\}\}$.

A formal comparison with the system in [6] is complicated by the fact that Milner uses so called concretions and abstractions as agents. Restricting attention to agents in the normal sense we have the following result, where $[\![\cdot]\!]$ is the function from the previous example.

Theorem 12. *P is well-sorted iff $[\![P]\!]$ is well-formed.*

Proof. A trivial induction over the structure of P, observing that the requirements are identical.

C.3 Polyadic synchronisation pi-calculus

Carbone and Maffeis [17] explore the so called pi-calculus with polyadic synchronisation, ${}^{e}\pi$, which can be thought of as a dual to the polyadic pi-calculus. Here action subjects are tuples of names, while the objects transmitted are just single names. It is demonstrated that this allows a gradual enabling of communication by opening the scope of names in a subject, results in simple representations of localities and cryptography, and gives a strictly greater expressiveness than standard pi-calculus.

In order to represent ${}^{e}\pi$, only minor modifications to the representation of the polyadic pi-calculus in Example C.1 are necessary. To allow tuples in subject position but not in object position, we invert the relations $\overline{\propto}$ and $\underline{\propto}$. Moreover, ${}^{e}\pi$ does not have name matching conditions $a = b$.

PSPI

Everything as in **PPI** except:
$\mathbf{C} = \{\top, \bot\}$ $\quad$ $\mathbf{X} = \mathcal{N}$ $\quad$ $\mathbf{1} \nvdash \bot$
$\widetilde{a} \mathrel{\dot\leftrightarrow} \widetilde{b}$ is $\top$ if $\widetilde{a} = \widetilde{b}$, and $\bot$ otherwise
$\text{MATCH}(a, \langle x \rangle, x) = \{a\}$
$\overline{\propto} = \underline{\propto} = \{(\mathtt{tup}, \mathtt{chan})\}$

In showing that this representation is fully abstract, for convenience we will consider a dialect of ${}^{e}\pi$ without the τ prefix. This has no cost in terms of expressiveness since the τ prefix can be encoded using a scoped communication. Let the mapping $[\![\cdot]\!]$ from ${}^{e}\pi$ processes into **PSPI** be homomorphic on all operators except sum, which is mapped to a $\top$-guarded **case** statement.

The proofs are similar to the polyadic pi-calculus case. The main differences are due to the fact that ${}^{e}\pi$ is formulated in terms of a late semantics with a structural congruence rule.

Lemma 2. *Let $\equiv$ be structural congruence on ${}^{e}\pi$ processes. If $P \equiv Q$ then $[\![P]\!] \sim [\![Q]\!]$.*

Proof. The relation $\mathcal{R} = \{(P, Q) : [\![P]\!] \sim [\![Q]\!]\}$ satisfies all the axioms defining $\equiv$ and is also a process congruence. Since $\equiv$ is the least such congruence, $\equiv \subseteq \mathcal{R}$.

Theorem 13.

1. *If $P \xrightarrow{\tilde{x}(y)} P'$ then for all z, $[\![P]\!] \xrightarrow{\langle \tilde{x} \rangle\, z} P''$ where $P'' \mathrel{\dot\sim} [\![P']\!][y := z]$.*
2. *If $P \xrightarrow{\alpha} P'$ and α is not an input, then $[\![P]\!] \xrightarrow{[\![\alpha]\!]} P''$ where $P'' \mathrel{\dot\sim} [\![P']\!]$.*
3. *If $[\![P]\!] \xrightarrow{\langle \tilde{x} \rangle\, z} P''$ then for all $y \# P$, $P \xrightarrow{\tilde{x}(y)} P'$ where $[\![P'\{z/y\}]\!] = P''$.*
4. *If $[\![P]\!] \xrightarrow{\alpha'} P''$ and α is not an input, then $P \xrightarrow{\alpha} P'$ where $[\![\alpha]\!] = \alpha'$ and $[\![P']\!] = P''$.*

Proof. By induction on the derivation of the transitions, using Lemma 2 in the Struct cases.

Polyadic synchronization pi has only guarded choice. We say that a **PSPI** process is case-guarded if in all its subterms of the form **case** $\varphi_1 : P_1 \; [] \cdots [] \; \varphi_n : P_n$, if $\varphi_i = \top$ then $P_i = \overline{M}\; N.Q$ or $P_i = \underline{M}(\lambda\widetilde{x})X.Q$. The case-guarded **PSPI** processes correspond directly to polyadic synchronization pi processes with guarded choice.

Theorem 14. *For all case-guarded* **PSPI** *processes R there exists an ${}^{e}\pi$ process $\overline{R}$ such that $R \mathrel{\dot\sim} [\![\overline{R}]\!]$.*

C.4 Value-passing CCS

We can also encode value-passing CCS [15], using substitution to perform evaluation of expressions.

Value-passing CCS [15] is an extension of pure CCS to admit arbitrary data from some set $\mathbf{V}$ to be sent along channels; there is no dynamic connectivity so channel names cannot be transmitted. When a value is received in a communication it replaces the input variable everywhere, and where this results in a closed expression it is evaluated, so for example $a(x)\,.\,\overline{c}(x+3)$ can receive 2 along a and become $\overline{c}\;5$. There are conditional **if** constructs that can test if a boolean expression evaluates to true, as in $a(x)\,.\,\textbf{if}\; x > 3 \;\textbf{then}\; P$.

To represent this as a psi-calculus we assume an arbitrary set of value expressions $e \in \mathbf{E}$ of sort `exp`, a subset of which is the boolean expressions $b \in \mathbf{E_B}$. The names are either used as channels (and then have the sort `chan`) or expression variables (of sort `exp`); only the latter can appear in expressions and be substituted by values. An expression is closed if it has no name of sort `exp` in its support, otherwise it is open. The values $v \in \mathbf{V}$ have sort `value`, the boolean values are $\{\text{true}, \text{false}\}$. We let E be an evaluation function on expressions, that takes each closed expression to a value and leaves open expressions unchanged. We write $e\{\widetilde{V}/\widetilde{x}\}$ for the result of syntactically replacing all $\widetilde{x}$ simultaneously by $\widetilde{V}$ in the (boolean) expression e, and assume that the result is a valid (boolean) expression. For example $(x+3)\{2/x\} = 2{+}3$, and $E(2{+}3) = 5$. Evaluation takes place in substitution: $(x+3)[x := 2] = E((x+3)\{2/x\}) = E(2+3) = 5$. Since patterns are single names we need to add a failure pattern $\perp$ as explained in Example 1.

VPCCS

$\mathcal{N} = \mathcal{N}_{\text{Ch}} \cup \mathcal{N}_{\text{Var}}$	$\mathcal{S}_{\mathcal{N}} = \{\texttt{chan}, \texttt{exp}\}$
$\mathbf{T} = \mathcal{N} \cup \mathbf{E} \cup \mathbf{V}$	$\mathcal{S} = \mathcal{S}_{\mathcal{N}} \cup \{\texttt{value}\}$
$\mathbf{C} = \mathbf{E_B}$	$e \in \mathbf{E} \Rightarrow \textsc{sort}(e) = \texttt{exp}$
$\mathbf{A} = \{\mathbf{1}\}$	$v \in \mathbf{V} \Rightarrow \textsc{sort}(v) = \texttt{value}$
$\mathbf{X} = \mathcal{N} \cup \{\perp\}$	$e \in \mathbf{E} \Rightarrow e[\widetilde{x} := \widetilde{M}] = E(e\{\widetilde{M}/\widetilde{x}\})$
$\mathbf{1} \vdash \text{true}, \quad \mathbf{1} \nvdash \text{false}$	$\prec = \{(\texttt{exp}, \texttt{value})\}$
$\dot\leftrightarrow$ = identity on names	$\mathcal{S}_\nu = \{\texttt{chan}\}$
$\textsc{match}(v, a, a) = \{v\}$ if $v \in \mathbf{V}$	$\overline{\propto} = \underline{\propto} = \{(\texttt{chan}, \texttt{exp}), (\texttt{chan}, \texttt{value})\}$
$\textsc{vars}(a) = \{a\}$	

Closed value-passing CCS processes correspond to **VPCCS** agents P where all free names are of sort `chan`, i.e., where $\textsc{sort}(\mathrm{n}(P)) \subseteq \{\texttt{chan}\}$.

We show full abstraction with regards to value-passing CCS as defined by Milner [15], with the following modifications: We use replication instead of process constants, with the standard semantics. We consider only finite sums. Milner allows for infinite sums without specifying exactly what infinite sets are allowed and how they are represented, making a fully formal comparison difficult. Introducing infinite sums naively in psi-calculi means that agents might exhibit infinite support and exhaust the set of names, rendering crucial operations such as α-converting all bound names to fresh names impossible. We do not consider relabellings at all. Relabelling has fallen out of fashion since the same effect can be obtained by abstracting over channels, and it is not included in the psi-calculi framework. Only a finite number of channels may be restricted by the set L in a restriction $P \setminus L$. With finite sums, this results in no loss of expressivity since agents have finite support.

Milner's restrictions are sets of names, which we represent as a sequence of ν-binders; for that reason we use a total ordering of any set of names (this is always available since the set of names is countable). Formally we assume an injective and support-preserving function $\sigma : \mathcal{P}_{\mathrm{fin}}(\mathcal{N}_{\texttt{chan}}) \to (\mathcal{N}_{\texttt{chan}})^*$. The mapping $[\![\cdot]\!]$ from value-passing CCS into **VPCCS** is defined homomorphically on all operators except the following:

$$\begin{aligned}
[\![\Sigma_i\ P_i]\!] &= \mathbf{case}\ \top : [\![P_1]\!]\ [\!]\ \cdots\ [\!]\ \top : P_i \\
[\![\mathbf{if}\ b\ \mathbf{then}\ P]\!] &= \mathbf{case}\ b : [\![P]\!] \\
[\![P \setminus L]\!] &= (\nu\sigma(L))[\![P]\!]
\end{aligned}$$

Lemma 3. *If P is a closed **VPCCS** process and $P \xrightarrow{\alpha} P'$, then P' is closed.*

Theorem 15. *If P and Q are closed value-passing CCS processes, then:*

1. *if $P \xrightarrow{\alpha} P'$ then $[\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$.*
2. *if $[\![P]\!] \xrightarrow{\alpha'} P''$ then $P \xrightarrow{\alpha} P'$ where $[\![\alpha]\!] = \alpha'$ and $[\![P']\!] = P''$.*

Proof. By induction on the derivations of P' and P'', respectively. The full proof is given in Appendix D.4.

Theorem 16. *For all closed value-passing CCS processes P and Q, $P \sim Q$ iff $[\![P]\!] \dot{\sim} [\![Q]\!]$*

Proof. $\{(P,Q) : [\![P]\!] \dot{\sim} [\![Q]\!] \wedge P, Q$ closed$\}$ is a bisimulation in value-passing CCS by coinduction, using Theorem 15.

Symmetrically, $\{(\mathbf{1}, [\![P]\!], [\![Q]\!]) : P \sim Q \wedge P, Q$ closed$\}$ is a bisimulation in **VPCCS**. The static equivalence and extension of arbitrary assertion cases are vacuous since there's only one frame. Symmetry follows from symmetry of $\sim$, and simulation follows by Theorem 15 and the fact that $\sim$ is a bisimulation.

Value-passing pi-calculus To demonstrate the modularity of psi-calculi, assume that we wish a variant of the pi-calculus enriched with values in the same way as value-passing CCS. This is achieved with only a minor change to **VPCCS**:

VPPI
Everything as in **VPCCS** except:
$\textsc{match}(z, a, a) = \{z\}$ if $z \in \mathbf{V} \cup \mathcal{N}_{ch}$
$\prec = \{(\mathtt{exp}, \mathtt{value}), (\mathtt{chan}, \mathtt{chan})\}$
$\overline{\propto} = \underline{\propto} = \{(\mathtt{chan}, \mathtt{exp}), (\mathtt{chan}, \mathtt{value}), (\mathtt{chan}, \mathtt{chan})\}$

Here also channel names can substitute channel names, and they can be sent and received along channel names.

D Full proofs for Appendix C

The following is full proofs of Appendix 3; we present them here, in a seperate section, due to their length.

We will assume that the reader is acquainted with the relevant psi-calculi presented in Section 3, as well as the definitions, notation and terminology of Sangiorgi [16], Carbone and Maffeis [17], and Milner [15], respectively. We will use their notation except as concerns the treatment of bound names, where we will adopt our notation, e.g. we will write $\text{bn}(\alpha)\#Q$ instead of $\text{bn}(\alpha) \cap \text{fn}(Q) = \emptyset$.

D.1 Auxiliary Lemma

The following lemma is used in the isomorphism proofs in the subsequent sections.

Lemma 4 (Flatten Case). *Let I be a Psi instance. Suppose $\mathbf{A}_I = \{\mathbf{1}\}$ and there is an equivariant condition $\top \in \mathbf{C}_I$ such that $\mathbf{1} \vdash \top$. Let $R = \mathbf{case}\ \top : (\mathbf{case}\ \widetilde{\varphi} : \widetilde{P})\ [\!]\ \widetilde{\phi} : \widetilde{Q}$ and $R' = \mathbf{case}\ \widetilde{\varphi} : \widetilde{P}\ [\!]\ \widetilde{\phi} : \widetilde{Q}$; Then $R \sim R'$.*

Proof. Let $\mathcal{R} = \bigcup_\sigma \{(\mathbf{1}, R\sigma, R'\sigma), (\mathbf{1}, R'\sigma, R\sigma)\} \cup \sim$. We first show that $\mathcal{R}$ is a bisimulation. We only consider the simulation cases as the other cases are trivial (there is only one assertion). Suppose a transition from R is derived as follows

$$\textsc{Case}\ \dfrac{\textsc{Case}\ \dfrac{P_i\sigma \xrightarrow{\alpha} P_i' \qquad \mathbf{1} \vdash \varphi_i\sigma}{\mathbf{case}\ \widetilde{\varphi}\sigma : \widetilde{P}\sigma \xrightarrow{\alpha} P_i'}}{\mathbf{case}\ \top : (\mathbf{case}\ \widetilde{\varphi}\sigma : \widetilde{P}\sigma)\ [\!]\ \widetilde{\phi}\sigma : \widetilde{Q}\sigma \xrightarrow{\alpha} P_i'}$$

then R' can simulate this with the following

$$\textsc{Case}\ \dfrac{P_i\sigma \xrightarrow{\alpha} P_i' \qquad \mathbf{1} \vdash \varphi_i\sigma}{\mathbf{case}\ \widetilde{\varphi}\sigma : \widetilde{P}\sigma\ [\!]\ \widetilde{\phi} : \widetilde{Q} \xrightarrow{\alpha} P_i'}$$

By reflexivity of $\sim$, we know $P'_i \sim P'_i$. By noting $\sim \subseteq \mathcal{R}$, we can conclude this case $\forall\sigma.(\mathbf{1}, P'_i\sigma, P'_i\sigma) \in \mathcal{R}$.

In case a transition of R is derived by

$$\textsc{Case}\ \frac{Q_i\sigma \ \xrightarrow{\alpha}\ Q'_i \qquad \mathbf{1} \vdash \phi_i\sigma}{\mathbf{case}\ \top : (\mathbf{case}\ \widetilde{\varphi}\sigma : \widetilde{P}\sigma)\ [\!]\ \widetilde{\phi}\sigma : \widetilde{Q}\sigma \ \xrightarrow{\alpha}\ Q'_i}$$

R' can simulate it with

$$\textsc{Case}\ \frac{Q_i\sigma \ \xrightarrow{\alpha}\ Q'_i \qquad \mathbf{1} \vdash \phi_i\sigma}{\mathbf{case}\ \widetilde{\varphi}\sigma : \widetilde{P}\sigma\ [\!]\ \widetilde{\phi}\sigma : \widetilde{Q}\sigma \ \xrightarrow{\alpha}\ Q'_i}$$

Again by reflexivity of $\sim$ we find $Q'_i \sim Q'_i$, and thus $\forall\sigma.(\mathbf{1}, Q'_i\sigma, Q'_i\sigma) \in \sim \subseteq \mathcal{R}$.

By a similar argument, we can show that R simulates R'. Since $\mathcal{R}$ is a bisimulation that is closed under all substitutions, $\mathcal{R} \subseteq \sim$.

D.2 Polyadic Pi-Calculus

We follow the exposition of Polyadic Pi-Calculus given by Sangiorgi in [16] with only departure being that we use replication in the labelled operational semantics instead of process constant invocation.

For convenience, we give an explicit definition of the encoding function given in Example C.1.

Definition 11 (Polyadic Pi-Calculus to PPi).
Agents:

$$\begin{aligned}
[\![P + Q]\!] &= \mathbf{case}\ \top : [\![P]\!]\ [\!]\ \top : [\![Q]\!] \\
[\![[x = y]P]\!] &= \mathbf{case}\ x = y : [\![P]\!] \\
[\![x(\tilde{y}).P]\!] &= \underline{x}(\lambda\tilde{y})\langle\tilde{y}\rangle.[\![P]\!] \\
[\![\overline{x}\langle\tilde{y}\rangle.P]\!] &= \overline{x}\langle\tilde{y}\rangle.[\![P]\!] \\
[\![0]\!] &= 0 \\
[\![P \mid Q]\!] &= [\![P]\!] \mid [\![Q]\!] \\
[\![\nu x P]\!] &= (\nu x)[\![P]\!] \\
[\![!P]\!] &= ![\![P]\!]
\end{aligned}$$

Actions:

$$\begin{aligned}
[\![(\nu\tilde{y'})\overline{z}\langle\tilde{y}\rangle]\!] &= \overline{z}\ (\nu\tilde{y'})\ \langle\tilde{y}\rangle \\
[\![x\langle\tilde{z}\rangle]\!] &= \underline{x}\ \langle\tilde{z}\rangle \\
[\![\tau]\!] &= \tau
\end{aligned}$$

In output action $\tilde{y'}$ do not bind into z.

Definition 12 (PPi to Polyadic Pi-Calculus).
Process:

$$\begin{aligned}
\overline{(\!|\mathbf{1}|\!)} &= \mathbf{0}\\
\overline{\mathbf{0}} = \overline{\mathbf{case}} &= \mathbf{0}\\
\overline{\mathbf{case}\ \varphi_1 : P_1 \ [\!]\ \dots\ [\!]\ \varphi_n : P_n} &= \overline{\varphi_1 : P_1} + \cdots + \overline{\varphi_n : P_n}\\
\overline{!P} &= !\overline{P}\\
\overline{(\nu x)P} &= \nu x \overline{P}\\
\overline{P \mid Q} &= \overline{P} \mid \overline{Q}\\
\overline{\underline{x}(\lambda \widetilde{y})\langle \widetilde{y}\rangle.P} &= x(\tilde{y}).\overline{P}\\
\overline{\overline{x}\langle \tilde{y}\rangle.P} &= \overline{x}\langle \tilde{y}\rangle.\overline{P}
\end{aligned}$$

Case clause:

$$\begin{aligned}
\overline{x = y : P} &= [x = y]\overline{P}\\
\overline{\top : P} &= \overline{P}
\end{aligned}$$

The following is proof of the strong operational correspondence.

Proof (of Theorem 10).

1. We show $P \xrightarrow{\alpha} P' \Rightarrow [\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$ by induction on the derivation of P'.

   Alp:
   Trivial in nominal logic.

   Out:
   We have that $\overline{x}\langle\tilde{y}\rangle.P \xrightarrow{\overline{x}\langle\tilde{y}\rangle} P$. Since $x \dot{\leftrightarrow} x$, we can derive $\overline{x}\ \langle\tilde{y}\rangle.[\![P]\!] \xrightarrow{\overline{x}\ \langle\tilde{y}\rangle} [\![P]\!]$.

   Inp:
   We have that $x(\tilde{y}).P \xrightarrow{x\langle\tilde{z}\rangle} P\{\tilde{z}/\tilde{y}\}$ with $|\tilde{z}| = |\tilde{x}|$, hence $\widetilde{z} \in \textsc{match}(\langle\widetilde{z}\rangle, \widetilde{y}, \langle\widetilde{y}\rangle)$. Using this and $x \dot{\leftrightarrow} x$ we can derive $\underline{x}(\lambda\widetilde{y})\langle\widetilde{y}\rangle.[\![P]\!] \xrightarrow{\underline{x}\ \langle\tilde{z}\rangle} [\![P]\!][\widetilde{y} := \widetilde{z}]$. By an easily checkable equality, $[\![P]\!][\widetilde{y} := \widetilde{z}] = [\![P\{\tilde{z}/\tilde{y}\}]\!]$, which completes the proof.

   Sum:
   Here $P \xrightarrow{\alpha} P'$ and by induction, $[\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$. Thus we can derive $\mathbf{case}\ \top : [\![P]\!]\ [\!]\ \top : [\![Q]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$.

   Par:
   Here $P \xrightarrow{\alpha} P'$ and $\mathrm{bn}(\alpha)\#Q$, and by induction, $[\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$. Then we can choose an alpha-variant of the frame of $[\![Q]\!]$ which is sufficiently fresh to allow the derivation $[\![P]\!] \mid [\![Q]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!] \mid [\![Q]\!]$.

   Com:
   Here $P \xrightarrow{(\nu\widetilde{y'})\overline{x}\langle\tilde{y}\rangle} P'$, $Q \xrightarrow{x\langle\tilde{y}\rangle} Q'$ with $\widetilde{y'} \subseteq \tilde{y}$ and $\widetilde{y'}\#Q$. By induction, $[\![P]\!] \xrightarrow{\overline{x}\ (\nu\widetilde{y'})\ \langle\widetilde{y}\rangle} [\![P']\!]$ and $[\![Q]\!] \xrightarrow{\underline{x}\ \langle\widetilde{y}\rangle} [\![Q']\!]$. Moreover, we note that $x \dot{\leftrightarrow} x$. Finally, we can choose alpha-variants of the frames of $[\![P]\!]$ and $[\![Q]\!]$ which are sufficiently fresh to allow the derivation $[\![P]\!] \mid [\![Q]\!] \xrightarrow{\tau} (\nu\widetilde{y'})([\![P']\!] \mid [\![Q']\!])$.

Match:
Here $P \xrightarrow{\alpha} P'$ and by induction, $[\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$. Thus we can derive **case** $x = x : [\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$.

Rep:
Here $P \mid !P \xrightarrow{\alpha} P'$ and by induction, $[\![P]\!] \mid ![\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$. Thus we can derive $![\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$.

Res:
Here $P \xrightarrow{\alpha} P'$ with $x\#\alpha$, and by induction, $[\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$. Hence we derive $(\nu x)[\![P]\!] \xrightarrow{[\![\alpha]\!]} (\nu x)[\![P']\!]$.

Open:
Here $P \xrightarrow{(\nu \widetilde{y'})\overline{z}\langle\tilde{y}\rangle} P'$ with $x \neq z$, $x\#\widetilde{y'}$ and $x \in \tilde{y}$. By induction, $[\![P]\!] \xrightarrow{\overline{z}\,(\nu\widetilde{y'})\,\langle\tilde{y}\rangle} [\![P']\!]$. Hence we derive $(\nu x)[\![P]\!] \xrightarrow{\overline{z}\,(\nu\widetilde{y'}\cup x)\,\langle\tilde{y}\rangle} [\![P']\!]$.

2. We now show that if $[\![P]\!] \xrightarrow{\alpha'} P''$ then $P \xrightarrow{\alpha} P'$ where $[\![\alpha]\!] = \alpha'$ and $[\![P']\!] = P''$. The proof is similar, by induction on the derivation of $[\![P']\!]$. We show only the "interesting" case:

Case:
Here $[\![P]\!] \xrightarrow{\alpha'} P''$ and by induction, $P \xrightarrow{\alpha} P'$ where $[\![\alpha]\!] = \alpha'$ and $[\![P']\!] = P''$. Since $P_C =$ **case** $\widetilde{\varphi} : \widetilde{P}$ is in the range of $[\![\cdot]\!]$, either $P_C = \top : [\![P]\!] \;[\!]\; \top : [\![Q]\!]$, $P_C = \top : [\![Q]\!] \;[\!]\; \top : [\![P]\!]$ or $P_C =$ **case** $x = y : [\![P]\!]$. We proceed by case analysis:
(a) When $P_C = \top : [\![P]\!] \;[\!]\; \top : [\![Q]\!]$, we note that $[\![P + Q]\!] = P_C$ and imitate the derivation of P'' from P_C with the derivation $P + Q \xrightarrow{\alpha} P'$, using the Sum rule.
(b) The case when $P_C = \top : [\![Q]\!] \;[\!]\; \top : [\![P]\!]$ is symmetric to the previous case.
(c) When $P_C =$ **case** $x = y : [\![P]\!]$, since $1 \vdash x = y$ by the induction hypothesis, $x = y$. we note that $[\![[x = x]P]\!] = P_C$ and imitate the derivation of P'' from P_C with the derivation $[x = x]P \xrightarrow{\alpha} P'$, using the Match rule.

Proof (of Theorem 11).
By structural induction on P. We only consider the case of **case** agent as other cases are trivial.

case case $\varphi_1 : P_1 \;[\!]\; \dots \;[\!]\; \varphi_n : P_n$**:**
We get an induction hypothesis for every $i \in \{1..n\}$, IH_i: $P_i \sim [\![\overline{P_i}]\!]$.
We proceed by induction on n.

base case $n = 0$**:**
$[\![\overline{\textbf{case}}]\!] = [\![\mathbf{0}]\!] = \mathbf{0}$. By reflexivity of $\sim$, $\mathbf{0} \sim \mathbf{0}$.

induction step $n+1$:

The IH for this case is

$$\llbracket \overline{\textbf{case}\ \varphi_1 : P_1 \;[\!]\; \dots \;[\!]\; \varphi_n : P_n} \rrbracket \sim \textbf{case}\ \varphi_1 : P_1 \;[\!]\; \dots \;[\!]\; \varphi_n : P_n = P'$$

We need to show that $Q \sim \llbracket \overline{Q} \rrbracket$ for $Q = \textbf{case}\ \varphi_1 : P_1 \;[\!]\; \dots \;[\!]\; \varphi_n : P_n \;[\!]\; \varphi_{n+1} : P_{n+1}$.

We compute

$$\begin{aligned}
\llbracket \overline{Q} \rrbracket &= \llbracket \overline{\varphi_1 : P_1} + \dots + \overline{\varphi_n : P_n} + \overline{\varphi_{n+1} : P_{n+1}} \rrbracket \\
&= \textbf{case}\ \top : \llbracket \overline{\varphi_1 : P_1} \rrbracket \;[\!]\; \dots \;[\!]\; \top : \llbracket \overline{\varphi_n : P_n} \rrbracket \;[\!]\; \top : \llbracket \overline{\varphi_{n+1} : P_{n+1}} \rrbracket \\
&\sim \text{(by Lemma 4)} \\
&\quad \textbf{case}\ \top : (\textbf{case}\ \top : \llbracket \overline{\varphi_1 : P_1} \rrbracket \;[\!]\; \dots \;[\!]\; \top : \llbracket \overline{\varphi_n : P_n} \rrbracket) \;[\!]\; \top : \llbracket \overline{\varphi_{n+1} : P_{n+1}} \rrbracket \\
&\sim \text{(by } IH) \\
&\quad \textbf{case}\ \top : (\textbf{case}\ \varphi_1 : P_1 \;[\!]\; \dots \;[\!]\; \varphi_n : P_n) \;[\!]\; \top : \llbracket \overline{\varphi_{n+1} : P_{n+1}} \rrbracket \\
&= \textbf{case}\ \top : P' \;[\!]\; \top : \llbracket \overline{\varphi_{n+1} : P_{n+1}} \rrbracket \\
&= Q'
\end{aligned}$$

We distinguishe the cases of φ_{n+1}:

case $\varphi_{n+1} = \top$:

$$\begin{aligned}
Q' &= \textbf{case}\ \top : P' \;[\!]\; \top : \llbracket \overline{\top : P_{n+1}} \rrbracket \\
&= \textbf{case}\ \top : P' \;[\!]\; \top : \llbracket \overline{P_{n+1}} \rrbracket \\
&\sim \text{(by } IH_{n+1}) \\
&\quad \textbf{case}\ \top : P' \;[\!]\; \top : P_{n+1} \\
&\sim \text{(by Lemma 4)} \\
&\quad \textbf{case}\ \varphi_1 : P_1 \;[\!]\; \dots \;[\!]\; \varphi_n : P_n \;[\!]\; \top : P_{n+1} = Q
\end{aligned}$$

We conclude this case.

case $\varphi_{n+1} = x = y$:

$$\begin{aligned}
Q' &= \textbf{case}\ \top : P' \;[\!]\; \top : \llbracket \overline{x = y : P_{n+1}} \rrbracket \\
&= \textbf{case}\ \top : P' \;[\!]\; \top : (\textbf{case}\ x = y : \llbracket \overline{P_{n+1}} \rrbracket) \\
&\sim \text{(by } IH_{n+1}) \\
&\quad \textbf{case}\ \top : P' \;[\!]\; \top : (\textbf{case}\ x = y : P_{n+1}) \\
&\sim \text{(by Lemma 4)} \\
&\quad \textbf{case}\ \varphi_1 : P_1 \;[\!]\; \dots \;[\!]\; \varphi_n : P_n \;[\!]\; \top : (\textbf{case}\ x = y : P_{n+1}) \\
&\sim \text{(by Lemma 4)} \\
&\quad \textbf{case}\ \varphi_1 : P_1 \;[\!]\; \dots \;[\!]\; \varphi_n : P_n \;[\!]\; x = y : P_{n+1} = Q
\end{aligned}$$

By concluding this case, we conclude the proof.

From the strong operational correspondence, we obtain full abstraction.

Theorem 17. *$P \sim_e Q$ iff $\llbracket P \rrbracket \dot{\sim} \llbracket Q \rrbracket$*

Proof. $\{(P,Q) : [\![P]\!] \stackrel{\bullet}{\sim} [\![Q]\!]\}$ is an early bisimulation in the polyadic pi-calculus by coinduction, using Theorem 10.

Symmetrically, $\{(\mathbf{1}, [\![P]\!], [\![Q]\!]) : P \sim_e Q\}$ is a bisimulation in **PPi**. The static equivalence and extension of arbitrary assertion cases are vacuous since there's only one frame. Symmetry follows from symmetry of $\sim_e$, and simulation follows by Theorem 10 and the fact that $\sim_e$ is an early bisimulation.

Lemma 5. *$[\![\cdot]\!]$ is injective, that is, for all P, Q, if $[\![P]\!] = [\![Q]\!]$ then $P = Q$.*

Proof. By induction on P and Q while inspecting all the possible cases.

Lemma 6. *$[\![\cdot]\!]$ is surjective up to $\sim$, that is, for every P there is a Q such that $[\![Q]\!] \sim P$.*

Proof. By structural induction on the well formed agent P.

case $\underline{x}(\lambda\widetilde{y})\langle\widetilde{y}\rangle.P'$**:**
IH tells us that, for some Q', $[\![Q']\!] \sim P'$. Let $Q = x(\tilde{y}).Q'$. Then, $[\![Q]\!] = [\![x(\tilde{y}).Q']\!] = \underline{x}(\lambda\widetilde{y})\langle\widetilde{y}\rangle.[\![Q']\!] \sim \underline{x}(\lambda\widetilde{y})\langle\widetilde{y}\rangle.P'$. This is what we needed to derive.

case $\overline{x}\langle\tilde{y}\rangle.P'$**:**
By IH, we have for some Q', $[\![Q']\!] \sim P'$. Let $Q = \overline{x}\langle\tilde{y}\rangle.Q'$. Now $[\![Q]\!] = \overline{x}\langle\tilde{y}\rangle.[\![Q']\!] \sim \overline{x}\langle\tilde{y}\rangle.P'$, which is what we wanted to derive.

case $P \mid P'$**:**
By IH, we have that for some Q', Q'', $[\![Q']\!] \sim P$ and $[\![Q'']\!] \sim P'$. Then let $Q = Q' \mid Q''$, thus $[\![Q]\!] = [\![Q']\!] \mid [\![Q'']\!] \sim P \mid P'$.

case $(\nu x)P$**:**
By IH, for some Q', $[\![Q']\!] \sim P$. Let $Q = \nu x Q'$. Then $[\![Q]\!] = (\nu x)[\![Q']\!] \sim (\nu x)P$.

case $!P$**:**
By IH, for some Q', $[\![Q']\!] \sim P$. Let $Q = !Q'$. Then $[\![Q]\!] = ![\![Q']\!] \sim !P$.

case $(\!|\mathbf{1}|\!)$**:**
Let $Q = \mathbf{0}$. Then $[\![Q]\!] = \mathbf{0} \sim (\!|\mathbf{1}|\!)$.

case case $\widetilde{\varphi} : \widetilde{P'}$**:**
For induction hypothesis $\text{IH}_{\textbf{case}}$, we have for every i there is Q'_i such that $[\![Q'_i]\!] \sim P'_i$. The proof goes by induction on the length of $\widetilde{\varphi}$.

base case:
Let $Q = \mathbf{0}$, then $[\![Q]\!] = \mathbf{0} \sim \textbf{case}$.

induction step:
At this step, we get the following IH

$$[\![Q'']\!] \sim \textbf{case}\ \varphi_1 : P_1 \ [\!]\ \ldots\ [\!]\ \varphi_n : P_n$$

We need to show that there is some $[\![Q]\!]$ such that

$$[\![Q]\!] \sim \textbf{case}\ \varphi_1 : P_1 \ [\!]\ \ldots\ [\!]\ \varphi_n : P_n \ [\!]\ \varphi_{n+1} : P_{n+1}$$

First, we note that $\mathrm{IH}_{\mathbf{case}}$ holds for every i and in particular $i = n+1$, thus we get $[\![Q'_{n+1}]\!] \sim P_{n+1}$. Second, we note that φ_{n+1} has two forms, thus we proceed by case analysis on φ_{n+1}.

case $\varphi_{n+1} = \top$**:**

Let $Q = Q'' + Q'_{n+1}$. Then

$$\begin{aligned}
[\![Q]\!] &= \mathbf{case}\ \top : [\![Q'']\!] \ [\!]\ \top : [\![Q'_{n+1}]\!] \\
&\sim \mathbf{case}\ \top : (\mathbf{case}\ \varphi_1 : P_1 \ [\!]\ \dots \ [\!]\ \varphi_n : P_n) \\
&\quad [\!]\ \top : [\![Q'_{n+1}]\!] \\
&\sim \mathbf{case}\ \top : (\mathbf{case}\ \varphi_1 : P_1 \ [\!]\ \dots \ [\!]\ \varphi_n : P_n) \\
&\quad [\!]\ \top : P_{n+1} \\
&\sim \text{(by Lemma 4)} \\
&\quad \mathbf{case}\ \varphi_1 : P_1 \ [\!]\ \dots \ [\!]\ \varphi_n : P_n \\
&\quad [\!]\ \top : P_{n+1}
\end{aligned}$$

This case is concluded.

case $\varphi_{n+1} = x = y$**:**

Let $Q = Q'' + [x = y]Q'_{n+1}$. Then

$$\begin{aligned}
[\![Q]\!] &= \mathbf{case}\ \top : [\![Q'']\!] \ [\!]\ \top : [\![[x = y]Q'_{n+1}]\!] \\
&\sim \mathbf{case}\ \top : (\mathbf{case}\ \varphi_1 : P_1 \ [\!]\ \dots \ [\!]\ \varphi_n : P_n) \\
&\quad [\!]\ \top : (\mathbf{case}\ x = y : [\![Q'_{n+1}]\!]) \\
&\sim \mathbf{case}\ \top : (\mathbf{case}\ \varphi_1 : P_1 \ [\!]\ \dots \ [\!]\ \varphi_n : P_n) \\
&\quad [\!]\ \top : (\mathbf{case}\ x = y : P_{n+1}) \\
&\sim \text{(by Lemma 4)} \\
&\quad \mathbf{case}\ \varphi_1 : P_1 \ [\!]\ \dots \ [\!]\ \varphi_n : P_n \\
&\quad [\!]\ \top : (\mathbf{case}\ x = y : P_{n+1}) \\
&\sim \text{(by permuting and applying Lemma 4)} \\
&\quad \mathbf{case}\ \varphi_1 : P_1 \ [\!]\ \dots \ [\!]\ \varphi_n : P_n \ [\!]\ x = y : P_{n+1}
\end{aligned}$$

This is the last part we needed to check, we conclude the proof.

Theorem 18. $[\![\cdot]\!]$ *is an isomorphism up to* $\sim$.

Proof. Directly follows from Lemma 5 and Lemma 6

D.3 Polyadic Synchronisation Pi-Calculus

We follow the exposition of Polyadic Synchronisation Pi-Calculus, ${}^{e}\pi$, of Carbone and Maffeis [17].

We give an explicit definition of encoding function defined in Example C.3.

Definition 13 (Polyadic synchronisation pi-calculus to PSPi).
Agents:

$$\begin{aligned}
[\![\widetilde{x}(y).P]\!] &= \underline{\langle\widetilde{x}\rangle}(\lambda y)y.[\![P]\!] \\
[\![\widetilde{x}\langle y\rangle.P]\!] &= \overline{\langle\widetilde{x}\rangle}\ y.[\![P]\!] \\
[\![P \mid Q]\!] &= [\![P]\!] \mid [\![Q]\!] \\
[\![(\nu x)P]\!] &= (\nu x)[\![P]\!] \\
[\![!P]\!] &= ![\![P]\!] \\
[\![0]\!] &= 0 \\
[\![\Sigma_i \alpha_i.P_i]\!] &= \textbf{case}\ \top_i : [\![\alpha_i.P_i]\!]
\end{aligned}$$

Actions:

$$\begin{aligned}
[\![\tilde{x}\langle \nu c\rangle]\!] &= \overline{\langle\tilde{x}\rangle}\ (\nu c)\ c \\
[\![\tilde{x}\langle c\rangle]\!] &= \overline{\langle\tilde{x}\rangle}\ c \\
[\![\tau]\!] &= \tau \\
[\![\tilde{x}(y)]\!] &= \textit{undefined}
\end{aligned}$$

Because in [17] Carbone and Maffeis defines late style laballed semantics for ${}^e\pi$ the input action has no translation.

Definition 14 (PSPi to Polyadic synchronisation pi-calculus).

$$\begin{aligned}
\overline{(\!|\mathbf{1}|\!)} &= \mathbf{0} \\
\overline{\mathbf{0}} &= \mathbf{0} \\
\overline{!P} &= !\overline{P} \\
\overline{(\nu x)P} &= (\nu x)\overline{P} \\
\overline{P \mid Q} &= \overline{P} \mid \overline{Q} \\
\overline{\langle\widetilde{a}\rangle y.P} &= \overline{a}\langle y\rangle.\overline{P} \\
\overline{\underline{\widetilde{x}}(\lambda y)y.P} &= \overline{x}(y).\overline{P} \\
\overline{\tau.P} &= \tau.\overline{P} \\
\overline{\textbf{case}\ \top : \alpha_i.P_i} &= \Sigma_i\overline{\alpha_i.P_i}
\end{aligned}$$

Lemma 7 (Lemma 2). *If $P \equiv Q$ then $[\![P]\!] \sim [\![Q]\!]$*

Proof. The relation $\mathcal{R} = \{(P, Q) : [\![P]\!] \sim [\![Q]\!]\}$ satisfies all the axioms defining $\equiv$ and is also a process congruence. Since $\equiv$ is the least such congruence, $\equiv \subseteq \mathcal{R}$.

We give proof for the strong operational correspondence.

Proof (of Theorem 13).

1. By induction on the derivation of P', avoiding z.

   Prefix:

   Here $\Sigma_i\tilde{x}_i(y_i).P_i \xrightarrow{\tilde{x}_i(y_i)} P_i$. We have that

$$\begin{aligned}
[\![\Sigma_i\tilde{x}_i(y_i).P_i]\!] = \textbf{case}\ \top &: \underline{\langle\tilde{x}\rangle}(\lambda y_1)y_1.[\![P_1]\!]\ [\!] \\
\cdots\ [\!]\ \top &: \underline{\langle\tilde{x}\rangle}(\lambda y_i)y_i.[\![P_i]\!]
\end{aligned}$$

Since $\textsc{match}(z, \langle y_i \rangle, y_i) = \{z\}$, we can use the Case and In rules to derive the transition

$$\begin{gathered}\mathbf{case}\ \top : \underline{\langle \tilde{x}_1 \rangle}(\lambda y_1)y_1.[\![P_1]\!]\ [\!]\ \cdots\ [\!]\ \top : \underline{\langle \tilde{x}_i \rangle}(\lambda y_i)y_i.[\![P_i]\!] \\ \xrightarrow{\langle \tilde{x} \rangle\ z} \\ [\![P_i]\!][y_i := z]\end{gathered}$$

Finally, we have $P'' = [\![P_i]\!][y_i := z]$ and use reflexivity of $\dot{\sim}$.

Bang:
Here $P \mid !P \xrightarrow{\tilde{x}(y)} P'$ and by induction, $[\![P]\!] \mid ![\![P]\!] \xrightarrow{\langle \tilde{x} \rangle\ z} P''$ with $P'' \mathrel{\dot{\sim}} [\![P']\!][y := z]$. By rule Rep, we also have that $![\![P]\!] \xrightarrow{\langle \tilde{x} \rangle\ z} P''$.

Par:
Here $P \xrightarrow{\tilde{x}(y)} P'$, $y\#Q$ and by induction, $[\![P]\!] \xrightarrow{\langle \tilde{x} \rangle\ z} P''$ with $P'' \mathrel{\dot{\sim}} [\![P']\!][y := z]$. Using the Par rule we derive $[\![P]\!] \mid [\![Q]\!] \xrightarrow{\langle \tilde{x} \rangle\ z} P' \mid [\![Q]\!]$. Since $\dot{\sim}$ is closed under $|$, $P'' \mid [\![Q]\!] \mathrel{\dot{\sim}} [\![P']\!][y := z] \mid [\![Q]\!]$. Finally, since $y\#Q$, $[\![P']\!][y := z] \mid [\![Q]\!] = [\![P' \mid Q]\!][y := z]$.

Struct:
Here $P \equiv Q$, $Q \xrightarrow{\tilde{x}(y)} Q'$ and $Q' \equiv P'$. By induction we obtain Q'' such that $[\![Q]\!] \xrightarrow{\langle \tilde{x} \rangle\ z} Q''$ where $Q'' \mathrel{\dot{\sim}} [\![Q']\!][y := z]$. By Lemma 2, $[\![P]\!] \sim [\![Q]\!]$ and $[\![Q']\!] \sim [\![P']\!]$, and by definition of $\sim$, $[\![Q']\!][y := z] \sim [\![P']\!][y := z]$. Since $[\![P]\!] \sim [\![Q]\!]$ and $[\![Q]\!] \xrightarrow{\langle \tilde{x} \rangle\ z} Q''$, there exists P'' such that $[\![P]\!] \xrightarrow{\langle \tilde{x} \rangle\ z} P''$ and $Q'' \mathrel{\dot{\sim}} P''$. By transitivity of $\dot{\sim}$ and the fact that $\sim \subseteq \dot{\sim}$, $P'' \mathrel{\dot{\sim}} [\![P']\!][y := z]$.

Res:
Here $P \xrightarrow{\tilde{x}(y)} P'$, $a \neq y$, $a \neq z$ $a\#\tilde{x}$, and by induction, $[\![P]\!] \xrightarrow{\langle \tilde{x} \rangle\ z} P''$ with $P'' \mathrel{\dot{\sim}} [\![P']\!][y := z]$. This gives us sufficient freshness conditions to derive $(\nu a)[\![P]\!] \xrightarrow{\langle \tilde{x} \rangle\ z} (\nu a)P''$. Since $\dot{\sim}$ is closed under restriction, $(\nu a)P'' \mathrel{\dot{\sim}} (\nu a)([\![P']\!][y := z])$. Finally, a is sufficiently fresh to so that $(\nu a)([\![P']\!][y := z]) = ((\nu a)[\![P']\!])[y := z]$

2. By induction on the derivation of P'. The cases not shown here are similar to the previous clause of this theorem, where P does an input.

Comm:
Here $P \xrightarrow{\overline{\tilde{x}}\langle y \rangle} P'$ and $Q \xrightarrow{\tilde{x}(z)} Q'$. By induction, $[\![P]\!] \xrightarrow{\overline{\langle \tilde{x} \rangle}\ y} P''$ where $P'' \mathrel{\dot{\sim}} [\![P']\!]$ and by the previous clause of this theorem, $[\![Q]\!] \xrightarrow{\langle \tilde{x} \rangle\ y} Q''$ such that $[\![Q']\!][z := y] \mathrel{\dot{\sim}} Q''$. The Com rule lets us derive the transition

$$[\![P]\!] \mid [\![Q]\!] \xrightarrow{\tau} P'' \mid Q''$$

To complete the induction case, we note that $(\nu y)(P'' \mid Q'') \mathrel{\dot{\sim}} [\![(\nu y)(P' \mid Q'\{y/z\})]\!]$

Close:
Here $P \xrightarrow{\overline{\tilde{x}}\langle \nu y \rangle} P'$ and $Q \xrightarrow{\tilde{x}(y)} Q'$. We assume $y\#Q$; if not, y can be

α-converted so that this holds. By induction, $[\![P]\!] \xrightarrow{\overline{\langle\tilde{x}\rangle}\ (\nu y)\ y} P''$ where $P'' \dot{\sim} [\![P']\!]$ and by the previous clause of this theorem, $[\![Q]\!] \xrightarrow{\langle\tilde{x}\rangle\ y} Q''$ such that $[\![Q']\!][y := y] = [\![Q']\!] \dot{\sim} Q''$. The COM rule lets us derive the transition

$$[\![P]\!] \mid [\![Q]\!] \xrightarrow{\tau} (\nu y)(P'' \mid Q'')$$

To complete the induction case, we note that $(\nu y)(P'' \mid Q'') \dot{\sim} [\![(\nu y)(P' \mid Q')]\!]$

Open:
Here $P \xrightarrow{\overline{\tilde{x}}\langle y\rangle} P'$ with $y \neq x$, and by induction, $[\![P]\!] \xrightarrow{\overline{\langle\tilde{x}\rangle}\ y} P''$ where $P'' \dot{\sim} [\![P']\!]$. By OPEN, we derive $(\nu y)[\![P]\!] \xrightarrow{\overline{\langle\tilde{x}\rangle}\ (\nu y)\ y} P''$.

3. By induction on the derivation of P", avoiding y.

Par:
Here $[\![P]\!] \xrightarrow{\underline{x}\ \langle\tilde{z}\rangle} P''$, $y\#P, Q$, and by induction $P \xrightarrow{\tilde{x}(y)} P'$ where $[\![P'\{z/y\}]\!] = P''$. By PAR using $y\#Q$, we derive $P \mid Q \xrightarrow{\tilde{x}(y)} P' \mid Q$. Finally, we note that since $y\#Q$, $[\![(P' \mid Q)\{z/y\}]\!] = P'' \mid [\![Q]\!]$.

Case:
Here $P_C \xrightarrow{\underline{\tilde{x}}\ z} P''$, where $P_C = \textbf{case}\ \widetilde{\varphi} : \widetilde{Q}$ is in the range of $[\![\cdot]\!]$ - hence P_C must be the encoding of some prefix-guarded sum, ie $P_C = [\![\Sigma_i \alpha_i.P_i]\!] = \textbf{case}\ \top : [\![\alpha_1]\!].[\![P_1]\!]\ [\!]\ \ldots\ [\!]\ \top : [\![\alpha_i]\!].[\![P_i]\!]$. By transition inversion we can deduce that for some j, $\alpha_j = \tilde{x}(y)$ and $[\![P_j]\!][y := z] = P''$. By the PREFIX rule, $\Sigma_i \alpha_i.P_i \xrightarrow{\tilde{x}(y)} P_j$.

Out:
A special case of CASE.

Rep:
Here $[\![P]\!] \mid ![\![P]\!] \xrightarrow{\underline{x}\ \langle\tilde{z}\rangle} P''$ and by induction $P \mid !P \xrightarrow{\tilde{x}(y)} P'$ where $[\![P'\{z/y\}]\!] = P''$. By BANG we derive $!P \xrightarrow{\tilde{x}(y)} P'$.

Scope:
Here $[\![P]\!] \xrightarrow{\underline{x}\ \langle\tilde{z}\rangle} P''$, $y\#P, Q$, $a\#\tilde{x}, y, z$ and by induction $P \xrightarrow{\tilde{x}(y)} P'$ where $[\![P'\{z/y\}]\!] = P''$. Since $a\#\tilde{x}, y, z$, the RES rule admits the derivation $(\nu a)P \xrightarrow{\tilde{x}(y)} (\nu a)P'$, and $[\![((\nu a)P')\{z/y\}]\!] = (\nu a)P''$

4. By induction on the derivation of P". The cases not shown are similar to the previous clause of this theorem.

Com:
Here $[\![P]\!] \xrightarrow{\overline{\langle\tilde{x}\rangle}\ (\nu\tilde{y'})\ y} P''$, $[\![Q]\!] \xrightarrow{\langle\tilde{x}\rangle\ y} Q''$ and $y'\#Q$. Either $\tilde{y'} = \epsilon$ or $\tilde{y'} = y$; we proceed by case analysis.

(a) If $\tilde{y'} = \epsilon$, we have $P \xrightarrow{\overline{\tilde{x}}\langle y\rangle} P'$ where $[\![P']\!] = P''$ by induction and, by the previous clause of this theorem, $Q \xrightarrow{\tilde{x}(z)} Q'$ where $[\![Q'\{y/z\}]\!] = Q''$. The COMM rule then lets us derive $P \mid Q \xrightarrow{\tau} P' \mid Q'\{y/z\}$.

(b) If $\tilde{y'} = y$, we have $P \xrightarrow{\overline{\tilde{x}}\langle\nu y\rangle} P'$ where $[\![P']\!] = P''$ by induction and, by the previous clause of this theorem, $Q \xrightarrow{\tilde{x}(y)} Q'$ where $[\![Q'\{y/y\}]\!] = [\![Q']\!] = Q''$. The Close rule then lets us derive $P \mid Q \xrightarrow{\tau} (\nu y)(P' \mid Q')$.

Open:

Here $[\![P]\!] \xrightarrow{\overline{\langle\tilde{x}\rangle}\ y} P''$ with $y \neq x$. By induction, $P \xrightarrow{\overline{\tilde{x}}\langle y\rangle} P'$ where $[\![P']\!] = P''$. By rule Open, $(\nu y)P \xrightarrow{\overline{\tilde{x}}\langle\nu y\rangle} P'$.

We give the full abstraction result for this calculus.

Theorem 19. *For all* ${}^{e}\pi$ *processes* P *and* Q*,* $P \stackrel{\cdot}{\sim} Q$ *iff* $[\![P]\!] \stackrel{\cdot}{\sim} [\![Q]\!]$

Proof. $\mathcal{R} = \{(P,Q) : [\![P]\!] \stackrel{\cdot}{\sim} [\![Q]\!]\}$ is an early bisimulation in the polyadic synchronisation pi-calculus; if $P \mathrel{\mathcal{R}} Q$ then

1. If $P \xrightarrow{\tilde{x}(y)} P'$ and $[\![P]\!] \stackrel{\cdot}{\sim} [\![Q]\!]$, since $\mathcal{R}$ is equivariant, we can assume that $y\#P, Q$ without loss of generality. Fix z. By Theorem 13.1, $[\![P]\!] \xrightarrow{\underline{\langle\tilde{x}\rangle}\ z} P''$ where $P'' \stackrel{\cdot}{\sim} [\![P']\!][y := z] = [\![P'\{z/y\}]\!]$. Hence, since $[\![P]\!] \stackrel{\cdot}{\sim} [\![Q]\!]$, $[\![Q]\!] \xrightarrow{\underline{\langle\tilde{x}\rangle}\ z} Q''$ where $P'' \stackrel{\cdot}{\sim} Q''$. Hence, by Theorem 13.3 using $y\#Q$, $Q \xrightarrow{\tilde{x}(y)} Q'$ where $[\![Q'\{z/y\}]\!] = Q''$. By transitivity, $[\![P'\{z/y\}]\!] \stackrel{\cdot}{\sim} [\![Q'\{z/y\}]\!]$.
2. If $P \xrightarrow{\alpha} P'$ and $[\![P]\!] \stackrel{\cdot}{\sim} [\![Q]\!]$, since $\mathcal{R}$ is equivariant, we can assume that $\mathrm{bn}(\alpha)\#P, Q$ without loss of generality. By Theorem 13.2, we have that $[\![P]\!] \xrightarrow{[\![\alpha]\!]} P''$ with $P'' \stackrel{\cdot}{\sim} [\![P']\!]$. Hence, since $[\![P]\!] \stackrel{\cdot}{\sim} [\![Q]\!]$ and $\mathrm{bn}(\alpha)\#Q$, there is a Q'' such that $[\![Q]\!] \xrightarrow{[\![\alpha]\!]} Q''$ and $Q'' \stackrel{\cdot}{\sim} P''$. By Theorem 13.4, there is Q' such that $Q \xrightarrow{\alpha} Q'$ and $[\![Q']\!] = Q''$. By transitivity, $[\![P']\!] \stackrel{\cdot}{\sim} [\![Q']\!]$.

Symmetrically, we show that $\mathcal{R} = \{(\mathbf{1}, [\![P]\!], [\![Q]\!]) : P \stackrel{\cdot}{\sim} Q\}$ is a bisimulation up to $\stackrel{\cdot}{\sim}$ in **PSPi**:

Static equivalence:

Vacuous since there's only one frame.

Symmetry:

By symmetry of $\stackrel{\cdot}{\sim}$

Simulation:

Here $[\![P]\!] \xrightarrow{\alpha'} P''$ and $P \stackrel{\cdot}{\sim} Q$. We proceed by case analysis on α':

1. If $\alpha' = \underline{\langle\tilde{x}\rangle}\ z$, then by Theorem 13.3 and a sufficiently fresh y, $P \xrightarrow{\tilde{x}(y)} P'$ where $[\![P'\{z/y\}]\!] = P''$. Since $P \stackrel{\cdot}{\sim} Q$, there exists Q' such that $Q \xrightarrow{\tilde{x}(y)} Q'$ and $P'\{z/y\} \stackrel{\cdot}{\sim} Q'\{z/y\}$. Hence, by Theorem 13.1, $[\![Q]\!] \xrightarrow{\underline{\langle\tilde{x}\rangle}\ z} Q''$ where $Q'' \stackrel{\cdot}{\sim} [\![Q']\!][y := z] = [\![Q'\{z/y\}]\!]$. We have that $P'' = [\![P'\{z/y\}]\!] \mathrel{\mathcal{R}} [\![Q'\{z/y\}]\!] \stackrel{\cdot}{\sim} Q''$, which suffices.

2. If α' is not an input, since $\mathcal{R}$ is equivariant, we can assume that $\mathrm{bn}(\alpha')\#P, Q$ without loss of generality. Since $[\![P]\!] \xrightarrow{\alpha'} P''$, by Theorem 13.4 we have that $P \xrightarrow{\alpha} P'$ where $[\![\alpha]\!] = \alpha'$ and $[\![P']\!] = P''$. Since $P \mathrel{\dot{\sim}} Q$, there is Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \mathrel{\dot{\sim}} Q'$. By Theorem 13.2, $[\![Q]\!] \xrightarrow{[\![\alpha]\!]} Q''$, where $Q'' \mathrel{\dot{\sim}} [\![Q']\!]$. Hence $P'' = [\![P']\!] \mathrel{\mathcal{R}} [\![Q']\!] \mathrel{\dot{\sim}} Q''$, which suffices.

Extension of arbitrary assertion:
Vacuous since there's only one frame.

Lemma 8. *$[\![\cdot]\!]$ is surjective up to $\sim$ on the set of case-guarded processes, that is, for every case-guarded P there is a Q such that $[\![Q]\!] \sim P$.*

Proof. By induction on a well formed agent P.

case $\underline{\langle\widetilde{x}\rangle}(\lambda y)y.P'$:
It is valid to consider only this form, since $\{y\} \in \textsc{vars}(y)$. The IH is for some Q', $[\![Q']\!] \sim P'$. Let $Q = \widetilde{x}(y).Q'$. Then $[\![Q]\!] = \underline{\langle\widetilde{x}\rangle}(\lambda y)y.[\![Q']\!] \sim \underline{\langle\widetilde{x}\rangle}(\lambda y)y.P'$.

case $\overline{\langle\widetilde{x}\rangle}\; y.P'$:
From IH, we get for some Q', $[\![Q']\!] \sim P'$. Let $Q = \widetilde{x}\langle y\rangle.Q'$. Then $[\![Q]\!] = \overline{\langle\widetilde{x}\rangle}\; y.[\![Q']\!] \sim \overline{\langle\widetilde{x}\rangle}\; y.P'$.

case $P' \mid P''$:
From IH, for some Q', Q'', we have $[\![Q']\!] \sim P'$ and $[\![Q'']\!] \sim P''$. Let $Q = Q' \mid Q''$. Then $[\![Q]\!] = [\![Q']\!] \mid [\![Q'']\!] \sim P' \mid P''$.

case $(\nu x)P'$:
Let $Q = \nu x Q'$, then by induction hypothesis $[\![Q]\!] = (\nu x)[\![Q']\!] \sim (\nu x)P'$.

case $!P'$:
Let $Q = !Q'$ (Q' from IH). $[\![Q]\!] = ![\![Q']\!] \sim !P'$.

case $\mathbf{0}$:
Then $[\![\mathbf{0}]\!] = \mathbf{0} \sim \mathbf{0}$.

case $(\!|\mathbf{1}|\!)$:
Then $[\![\mathbf{0}]\!] = \mathbf{0} \sim (\!|\mathbf{1}|\!)$.

case case $\widetilde{\varphi} : \widetilde{P'}$:
For induction hypothesis $\mathrm{IH}_{\mathbf{case}}$, we have for every i there is Q'_i such that $[\![Q'_i]\!] \sim P'_i$. The proof goes by induction on the length of $\widetilde{\varphi}$.

base case:
Let $Q = \mathbf{0}$, then $[\![Q]\!] = \mathbf{0} \sim \mathbf{case}$.

induction step:
At this step, we get the following IH

$$[\![Q'']\!] \sim \mathbf{case}\ \varphi_1 : P_1 \mathbin{[\,]} \ldots \mathbin{[\,]} \varphi_n : P_n$$

We need to show that there is some $[\![Q]\!]$ such that

$$[\![Q]\!] \sim \mathbf{case}\ \varphi_1 : P_1 \mathbin{[\,]} \ldots \mathbin{[\,]} \varphi_n : P_n \mathbin{[\,]} \varphi_{n+1} : P_{n+1} = P$$

First, we note that IH$_{\textbf{case}}$ holds for every i and in particular $i = n+1$, thus we get $[\![Q'_{n+1}]\!] \sim P_{n+1}$. Second, we note that φ_{n+1} has two forms, thus we proceed by case analysis on φ_{n+1}.

case $\varphi_{n+1} = \bot$:
Let $Q = Q''$. Then

$$\begin{aligned}
[\![Q]\!] &= [\![Q'']\!] \\
&\sim \textbf{case } \varphi_1 : P_1 \ [\!] \ldots [\!] \ \varphi_n : P_n \\
&\sim \textbf{case } \varphi_1 : P_1 \ [\!] \ldots [\!] \ \varphi_n : P_n \ [\!] \ \bot : P_{n+1}
\end{aligned}$$

This case is concluded.

case $\varphi_{n+1} = \top$:
From the assumption, we know that P_{n+1} is of form $\alpha.P'_{n+1}$ and that $[\![Q'_{n+1}]\!] \sim \alpha.P'_{n+1}$. By investigating the construction of Q'_{n+1} we can conclude that $Q'_{n+1} = \alpha.Q''_{n+1}$ where $[\![Q''_{n+1}]\!] \sim P'_{n+1}$. The agent from IH Q'' is either $\mathbf{0}$, or prefixed agent, or a mixed sum.
In case $Q'' = \mathbf{0}$, let $Q = Q'_{n+1}$, then $[\![Q]\!] = [\![Q'_{n+1}]\!] \sim P$.
In case Q'' is prefixed agent, let $Q = Q'' + Q'_{n+1}$. Since Q'' and Q'_{n+1} are prefixed, Q is well formed. Then $[\![Q]\!] = \textbf{case } \top : [\![Q'']\!] \ [\!] \ \top : [\![Q'_{n+1}]\!] \sim \textbf{case } \varphi_1 : P_1 \ [\!] \ldots [\!] \ \varphi_n : P_n \ [\!] \ \top : P_{n+1}$.
In case Q'' is a sum, let $Q = Q'' + Q'_{n+1}$. Since Q'_{n+1} is guarded, Q is well formed. Then

$$\begin{aligned}
[\![Q]\!] &= \textbf{case } \top : [\![Q'']\!] \ [\!] \ \top : [\![Q'_{n+1}]\!] \\
&\sim \textbf{case } \top : (\textbf{case } \varphi_1 : P_1 \ [\!] \ldots [\!] \ \varphi_n : P_n) \\
&\quad\ [\!] \ \top : [\![Q'_{n+1}]\!] \\
&\sim \text{(by Lemma 4)} \\
&\quad\ \textbf{case } \varphi_1 : P_1 \ [\!] \ldots [\!] \ \varphi_n : P_n \\
&\quad\ [\!] \ \top : [\![Q'_{n+1}]\!] \\
&\sim \textbf{case } \varphi_1 : P_1 \ [\!] \ldots [\!] \ \varphi_n : P_n \\
&\quad\ [\!] \ \top : P'_{n+1}
\end{aligned}$$

This concludes the proof.

Lemma 9. *$[\![\cdot]\!]$ is injective, that is, for all P, Q, if $[\![P]\!] = [\![Q]\!]$ then $P = Q$.*

Proof. By induction on P and Q while inspecting all the possible cases.

Theorem 20. *$[\![\cdot]\!]$ is an isomorphism up to $\sim$ between ${}^{e}\pi$ and the case-guarded processes in* **PSPI**.

Proof. Directly follows from Lemma 9 and Lemma 8

D.4 Value-passing CCS

Lemma 10. *If P is a* **VPCCS** *process such that $P \xrightarrow{\overline{M}\,(\nu\widetilde{x})\,N} P''$ then $\widetilde{x} = \epsilon$*

Proof. By induction on the derivation of P'. Obvious in all cases except Open, where we derive a contradiction since only values can be transmitted yet only channels can be restricted - hence the name a is both a name and a value.

Strong operational correspondence:

Proof (of Theorem 15).

1. By induction on the derivation of P'.

   Act:
   We have that $\alpha.P \xrightarrow{\alpha} P$. Since $\alpha.P$ is in the range of $\widehat{\cdot}$, there must be x and v such that either $\alpha = \overline{x}(v)$ (for if α was an input, $\alpha.P$ would be outside the range of $\widehat{\cdot})$. The Out rule then admits the derivation $\overline{x}\ v.[\![P]\!] \xrightarrow{\overline{x}\ v} [\![P]\!]$

   Sum:
   There are two cases to consider: either $\Sigma_i P_i$ is the encoding of an input, or a summation.
   (a) If $\Sigma_i P_i = \Sigma_v x(v).P\{v/y\} = \widehat{x(y).P}$ we have that $\alpha = x(v)$. Then for each v, we can derive $\underline{x}(\lambda y)y.[\![P]\!] \xrightarrow{\underline{x}\ v} [\![P\{v/y]\!]$ using the In rule.
   (b) Otherwise, we have that $P_j \xrightarrow{\alpha} P'$ and by induction,
   $$[\![P_j]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$$
   The Case rule lets us derive
   $$\mathbf{case}\ \top : [\![P_1]\!]\ [\!]\ \cdots\ [\!]\ \top : P_i \xrightarrow{[\![\alpha]\!]} [\![P']\!]$$
   This suffices since $[\![\Sigma_i P_i]\!] = \mathbf{case}\ \top : [\![P_1]\!]\ [\!]\ \cdots\ [\!]\ \top : P_i$.

   Com1:
   Here $P \xrightarrow{\alpha} P'$ and by induction, $[\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$. The Par rule admits derivation of the transition $[\![P]\!] \mid [\![Q]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!] \mid [\![Q]\!]$, using Lemma 10 to discharge the freshness side condition.

   Com2:
   Symmetric to Com1.

   Com3:
   Here $P \xrightarrow{\alpha} P'$, $Q \xrightarrow{\overline{\alpha}} Q'$. Since α is in the range of $\widehat{\cdot}$, there are x and v such that $\alpha = x(v)$ and $\overline{\alpha} = \overline{x}(v)$ (or vice versa, in which case read the next sentence symmetrically). By the induction hypotheses, $[\![P]\!] \xrightarrow{\underline{x}\ v} [\![P']\!]$ and $[\![Q]\!] \xrightarrow{\overline{x}\ v} [\![Q']\!]$ - hence $[\![P]\!] \mid [\![Q]\!] \xrightarrow{\tau} [\![P']\!] \mid [\![Q']\!]$ by the Com rule, using Lemma 10 to discharge the freshness side condition.

   Res:
   Here $P \xrightarrow{\alpha} P'$ with $L\#\alpha$ - hence $\sigma(L)\#\alpha$. By induction, $[\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$. Then we use the Res rule $|L|$ times to derive $(\nu\sigma(L))[\![P]\!] \xrightarrow{[\![\alpha]\!]} (\nu\sigma(L))[\![P']\!]$.

Rep:

Here $P \mid !P \xrightarrow{\alpha} P'$. By induction, $[\![P]\!] \mid ![\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$, and by the Rep rule, $![\![P]\!] \xrightarrow{[\![\alpha]\!]} [\![P']\!]$

2. By induction on the derivation of P'.

In:

Here $\underline{x}(\lambda y)y.[\![P]\!] \xrightarrow{\underline{x}\ v} [\![P\{v/y\}]\!]$. We match this by deriving $\widehat{x(y).P} \xrightarrow{x(v)} \widehat{P}\{v/y\}$ using the Act and Sum rules.

Out:

Here $\overline{x}\ v.[\![P]\!] \xrightarrow{\overline{x}\ v} [\![P]\!]$. We match this by deriving $\widehat{\overline{x}(v).P} \xrightarrow{\overline{x}(v)} \widehat{P}$ using the Act rule.

Com:

Here $[\![P]\!] \xrightarrow{\overline{x}\ (\nu\tilde{y})\ v} P''$, $[\![Q]\!] \xrightarrow{\underline{x}\ v} Q''$. By Lemma 10, $\tilde{y} = \epsilon$, and by induction, $P \xrightarrow{\overline{x}(v)} P'$ and $Q \xrightarrow{x(v)} Q'$where $[\![P']\!] = P''$ and $[\![Q']\!] = Q''$. Using the Com3 rule we derive $P \mid Q \xrightarrow{\tau} P' \mid Q'$

Par:

Easy.

Case:

Our case statement can either be the encoding of either a summation or an **if** statement. We proceed by case analysis:

(a) Here $[\![P_j]\!] \xrightarrow{\alpha'} P''$. By induction, $P_j \xrightarrow{\alpha} P'$ where $[\![\alpha]\!] = \alpha'$. By Sum, $\Sigma_i P_i \xrightarrow{\alpha} P'$.

(b) Here $[\![P]\!] \xrightarrow{\alpha'} P''$ and $\mathbf{1} \vdash b$. By induction, $P \xrightarrow{\alpha} P'$ where $[\![\alpha]\!] = \alpha'$ and $[\![P']\!] = P''$. Since b evaluates to true, $\widehat{\textbf{if } b \textbf{ then}}\ P = \widehat{P}$ - hence **if** b **then** $P \xrightarrow{\alpha} P'$.

Rep:

Easy.

Scope:

Here $[\![P]\!] \xrightarrow{\alpha'} P''$ with $x\sharp\alpha'$ and by induction, $P \xrightarrow{\alpha} P'$ where $\alpha' = [\![\alpha]\!]$ and $P'' = [\![P']\!]$. Hence we can derive $P \setminus \{x\} \xrightarrow{\alpha} P' \setminus \{x\}$ by the Res rule.

Open:

Opening is not possible.