A SORTED SEMANTIC FRAMEWORK FOR APPLIED PROCESS CALCULI

JOHANNES BORGSTRÖM, RAMŪNAS GUTKOVAS, JOACHIM PARROW, BJÖRN VICTOR, AND JOHANNES ÅMAN POHJOLA

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. Finally, we prove standard congruence and structural properties of bisimulation; substantial parts of the proof 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 [BJPV11], which provides machine-checked proofs that important meta-theoretical properties, such as compositionality of bisimulation, hold in all instances of the framework. We claim that the theoretical development is more robust than that of other calculi of comparable complexity, since we use a structural operational semantics given by a single inductive definition, and since we have checked most results in the theorem prover Nominal Isabelle [Urb08].

LOGICAL METHODS
IN COMPUTER SCIENCE
DOI:10.2168/LMCS-???

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 correspond to standard calculi, and use several different notions of computation. We define strong criteria for a psi-calculus to *represent* another process calculus, meaning that they are for all practical purposes one and the same. Representation is stronger than the standard *encoding* correspondences e.g. by Gorla[Gor10], which define criteria for one language to encode the behaviour of another. The representations that we provide of other calculi advance our previous work, where we had to resort to nontrivial encodings with unclear formal correspondence to the standard calculi.

1.1. **Background: Psi-calculi.** In the following we assume the reader to be acquainted with the basic ideas of process algebras based on the pi-calculus, and explain psi-calculi by a few simple examples. Full definitions can be found in the references above, and for a reader not acquainted with our work we recommend the first few sections of [BJPV11] for an introduction.

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 $M \dot{\leftrightarrow} K$. 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 $\Psi \vdash M \dot{\leftrightarrow} K$, 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* [Pit03]. This means that there is a distinguished set of names, and for each term a well defined notion of *support*, intuitively corresponding to the names occurring in the term. Functions and relations must be *equivariant*, meaning that they treat all names equally. In addition, we impose straight-forward requirements on the combination of assertions, on channel equivalence, and on substitution. Our requirements are quite general, and therefore our framework accommodates a wide variety of applied process calculi.

1.2. Extension: Generalized pattern matching.

In our original definition of psi-calculi ([BJPV11], 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 **??** and **??** in Section 5.

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 [BJPV11]; 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.

In this paper, we address these three issues by introducing explicit notions of patterns, pattern variables and matching. This allows us to control precisely which parts of messages can be bound by pattern-matching and how messages can be deconstructed. It also ensures that well-formedness is preserved by transitions and admits computations such as $\mathtt{dec}(\mathtt{enc}(M, N, K), K) \to M$ in substitutions.

1.3. **Extension: Sorting.** Applied process calculi often make use of a sort system. The applied pi-calculus [AF01] 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 [HJ06] 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. Such anomalies disappear when introducing an appropriate sort system; cf. Section 4.1.

1.4. **Related work.** Pattern-matching is in common use in functional programming languages. Scala admits pattern-matching of objects [EOW07] using a method `unapply` that turns the receiving object into a matchable value (e.g. a tuple). F# admits the definition of pattern cases independently of the type that they should match [SNM07], facilitating interaction with third-party and foreign-language code. Turning to message-passing systems, LINDA [Gel85] uses pattern-matching when receiving from a tuple space. Similarly, in Erlang, message reception from a mailbox is guarded by a pattern.

These notions of patterns, with or without computation, are easily supported by the MATCH construct. However, the standard first-match policy needs to be accomodated by extending the pattern language, as is usual for core calculi [Kri09].

Pattern matching in process calculi. The pattern-matching spi-calculus [HJ06] limits which variables may be binding in a pattern in order to match encrypted messages without binding unknown keys (cf. Example **??**). The Kell calculus [SS05] also uses pattern languages equipped with a match function. However, in the Kell calculus the channels are single names and appear as part of the pattern in the input prefix, patterns may match multiple communications simultaneously (à la join calculus), and first-order pattern variables only match names (not composite messages) making forwarding and partial decomposition impossible.

The applied pi-calculus [AF01] models deterministic computation by using for data language a term algebra modulo an equational logic. ProVerif [Bla11] 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. Section 5.1).

Other tools for process calculi extended with datatypes include mCRL2 [CGK$^+$13] for ACP, which allows higher order sorted term algebras and equational logic, and PAT3 [LSD11] which includes a CSP♯ [SLDC09] module where actions built over types like booleans and integers are extended with C♯-like programs. In all these cases, the pattern matching is defined by substitution in the usual way.

A comparison of expressiveness to calculi with non-binary (e.g., join-calculus [FG96]) or bidirectional (e.g., dyadic interaction terms [Hon93] or the concurrent pattern calculus [GWGJ10]) communication primitives would be interesting. We here inherit positive results from the pi calculus, such as the encoding of the join-calculus.

Sort systems for mobile processes. Sorts for the pi-calculus were first described by Milner [Mil93], and were developed in order to remove nonsensical processes using polyadic communication, similar to the motivation for the present work.

In contrast, Hüttel's dependently typed psi-calculi [Hüt11] is intended for a more fine-grained control of the behaviour of processes. Typed psi-calculi are capable of capturing a wide range of earlier sort systems for pi-like calculi formulated as instances of psi-calculi. However, 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 can be seen as a foundation for typed psi-calculi: we give a formal account of the separation between variables and names used in typed psi-calculi, and substantiate that Hüttel's claim that "the set of well-[sorted] terms is closed under well-[sorted] substitutions, which suffices" does not cause problems for the meta-theory of the language. Typed psi-caluli are also less general than sorted psi-calculi in some ways: the term language of typed psi-calculi is required to be a free term algebra (without name binders); it uses only the standard notions of substitution and matching, and does not admit any computation on terms. Furthermore, we prove meta-theoretical results including congruence results and structural equivalence laws for well-sorted bisimulation, and the preservation of well-sortedness under structural equivalence; no such results exist for typed psi-calculi.

The state-of-the art report [HV13] of WG1 of the BETTY project (EU COST Action IC1201) is a comprehensive guide to behavioural types for process calculi.

Fournet et al. [FGM05] add type-checking for a general authentication logic to a process calculus with destructor matching; there the authentication logic is only used to specify program correctness, and does not influence the operational semantics in any way.

1.5. **Results and outline.** In Section 2 we define psi-calculi with the above extensions and prove preservation of well-formedness. In Section 3 we prove the usual algebraic properties of bisimilarity. The proof is in two steps: a machine-checked proof for single-sorted calculi, followed by a manual proof based on the translation of a multi-sorted psi calculus instance to a corresponding single-sorted instance. We demonstrate the expressiveness of our generalization in Section 4 where we directly represent standard calculi, and in Section 5 where we give examples of 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 and elements of the data type 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 $\widetilde{x}$ also stands for the set of names $\{x_1, \ldots, x_n\}$ if used where a set is expected. We let π range over permutations of tuples of names: $\pi \cdot \widetilde{x}$ is a tuple of names of the same length as $\widetilde{x}$, containing the same names with the same multiplicities.

A sorted *nominal set* [Pit03, GP01] 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 $(a\ b) \cdot (f(T)) = f((a\ b) \cdot T)$ always holds; a relation $\mathcal{R}$ is equivariant if $x\ \mathcal{R}\ y$ implies that $(a\ b) \cdot x\ \mathcal{R}\ (a\ b) \cdot y$ holds; and a constant symbol C is equivariant if $(a\ b) \cdot C = C$. A *nominal data type* is a nominal set together with some equivariant 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 [BJPV11], which are given by three nominal datatypes (data terms, conditions and assertions) as discussed in the introduction.

Definition 2.1 (Original psi-calculus parameters)**.** The psi-calculus parameters from the original psi-calculus are the following nominal data types: (data) terms $M, N \in \mathbf{T}$, conditions $\varphi \in \mathbf{C}$, and assertions $\Psi \in \mathbf{A}$; equipped with the following 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 $\mathbf{1} \in \mathbf{A}$, and the entailment relation $\vdash \subseteq \mathbf{A} \times \mathbf{C}$.

The binary functions $\dot{\leftrightarrow}$ and $\otimes$ and the relation $\vdash$ above will be used in infix form. Two assertions are said to be 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 modulo entailment, 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 [BJPV11].

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

Definition 2.2 (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

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

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

Requisites on $\textsc{vars}$ and $\textsc{match}$ are given below in Definition 2.5. 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 2.3 (Psi-calculus parameters for sorting)**.** The psi-calculus parameters for sorting include the sorting function $\textsc{sort} : \mathcal{N} \uplus \mathbf{T} \uplus \mathbf{X} \to \mathcal{S}$, and the four compatibility predicates

$$\begin{array}{rcll} \underline{\propto} & \subseteq & \mathcal{S} \times \mathcal{S} & \text{can be used to receive,} \\ \overline{\propto} & \subseteq & \mathcal{S} \times \mathcal{S} & \text{can be used to send,} \\ \prec & \subseteq & \mathcal{S} \times \mathcal{S} & \text{can be substituted by,} \\ \mathcal{S}_\nu & \subseteq & \mathcal{S} & \text{can be bound by name restriction.} \end{array}$$

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 2.4 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 $\forall t \in \mathcal{S}.(s_2 \,\underline{\propto}\, t \Rightarrow s_1 \,\underline{\propto}\, t) \wedge (s_2 \,\overline{\propto}\, t \Rightarrow s_1 \,\overline{\propto}\, t) \wedge (t \,\underline{\propto}\, s_2 \Rightarrow t \,\underline{\propto}\, s_1) \wedge (t \,\overline{\propto}\, s_2 \Rightarrow t \,\overline{\propto}\, s_1)$. This relation compares the 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 2.4 (Requisites on 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 $T[\widetilde{a} := \widetilde{N}] = ((\widetilde{a}\ \widetilde{b}) \cdot T)[\widetilde{b} := \widetilde{N}]$ (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 $X\sigma \in \mathbf{X}$, and if $\widetilde{x} \in \textsc{vars}(X)$ and $\widetilde{x} \# \sigma$ then $\textsc{sort}(X\sigma) \leq \textsc{sort}(X)$ and $\widetilde{x} \in \textsc{vars}(X\sigma)$ and alpha-renaming of substituted variables (as above) holds for σ and X.

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.

In the original psi-calculi there is no requirement that substitutions on terms preserve names used as pattern variables (i.e., $\mathrm{n}(N\sigma) \supseteq \mathrm{n}(N) \setminus \mathrm{n}(\sigma)$). For this reason, the original psi semantics does not always 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 [BJPV11]. In pattern-matching psi-calculi, the operational semantics does preserve well-formedness, as shown below in Theorem 2.11.)

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 2.5 (Requisites on 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 $(\widetilde{x}\ \widetilde{y})$ with $\widetilde{y} \# X$ we have $\widetilde{N} \in \textsc{match}(M, \widetilde{y}, (\widetilde{x}\ \widetilde{y}) \cdot X)$ (alpha-renaming of matching).

In many process calculi, and also in the symbolic semantics of psi [JVP12], 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 2.6. 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 $a\#\sigma$.

When all substitutions on terms preserve names, we can recover the pattern matching of the original psi-calculi. Such psi-calculi also enjoy well-formedness preservation (Theorem 2.11).

Theorem 2.7. *Suppose* $(\mathbf{T}, \mathbf{C}, \mathbf{A})$ *is an original psi-calculus* [BJPV11] *where* $\mathrm{n}(N\sigma) \supseteq \mathrm{n}(N) \setminus \mathrm{n}(\sigma)$ *for all* N, σ*. 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.*

Proof. Straightforward; this result has been checked in Isabelle. □

2.5. **Agents.**

Definition 2.8 (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
$\mathbf{case}\ \varphi_1 : P_1 \mathrel{[\!]} \cdots \mathrel{[\!]} \varphi_n : P_n$	Case
$(\nu a)P$	Restriction
$P \mid Q$	Parallel
$!P$	Replication
$(\!\vert \Psi \vert\!)$	Assertion

In the Input all names in $\widetilde{x}$ bind their occurrences in both X and P, and in the Restriction a binds in P. Substitution on agents is defined inductively on their structure, using the substitution function of each datatype based on syntactic position, avoiding name capture.

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 $\mathbf{case}\ \varphi_1 : P_1 \mathrel{[\!]} \cdots \mathrel{[\!]} \varphi_n : P_n$ executes one of the branches P_i where the corresponding condition φ_i holds, discarding the other branches. Restriction $(\nu a)P$ scopes the name a in P; the scope of a may be extruded if P communicates a data term containing a. A parallel composition $P \mid Q$ denotes P and Q running in parallel; they may proceed independently or communicate. A replication $!P$ models an unbounded number of copies of the process P. The assertion $(\!|\Psi|\!)$ contributes Ψ to its environment. We often write **if** φ **then** P for **case** $\varphi : P$, and nothing or $\mathbf{0}$ for the empty case statement **case**.

In comparison to [BJPV11] we additionally 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 2.9. An assertion is *guarded* if it is a subterm of an Input or Output. An agent is *well-formed* if, for all its subterms,

1. in a replication $!P$ there are no unguarded assertions in P; and
2. in **case** $\varphi_1 : P_1 \mathbin{[\!]} \cdots \mathbin{[\!]} \varphi_n : P_n$ there is no unguarded assertion in any P_i; and
3. in an Output $\overline{M}\ N.P$ we require that $\textsc{sort}(M) \mathrel{\overline{\propto}} \textsc{sort}(N)$; and
4. in an Input $\underline{M}(\lambda\widetilde{x})X.P$ we require that
   - (a) $\widetilde{x} \in \textsc{vars}(X)$ is a tuple of distinct names and
   - (b) $\textsc{sort}(M) \mathrel{\underline{\propto}} \textsc{sort}(X)$; and
5. in a Restriction $(\nu a)P$ we require that $\textsc{sort}(a) \in \mathcal{S}_\nu$.

Requirements 3, 4b and 5 are new for sorted psi-calculi.

2.6. **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 2.10 (Frames and Transitions)**.** The *frame* $\mathcal{F}(P)$ *of an agent* P is defined inductively as follows:

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

The *actions* ranged over by α, β are of the following three kinds: Output $\overline{M}\ (\nu\tilde{a})\ N$ 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, defined in Table 1, is the same as for the original psi-calculi, except for the use of $\textsc{match}$ in rule $\textsc{In}$. We identify alpha-equivalent agents and transitions (see [BJPV11] 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.

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

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

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

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

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

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.

Theorem 2.11 (Preservation of well-formedness)**.** *If P is well-formed, then*

(1) *$P\sigma$ is well-formed; and*

(2) *if $\Psi \rhd P \xrightarrow{\alpha} P'$ then P' is well-formed.*

Proof. The first part is by induction on P. The output prefix case uses the sort preservation property of substitution on terms (Definition 2.4). The interesting case is input prefix $\underline{M}(\lambda\widetilde{x})X.Q$: assume that Q is well-formed, that $\widetilde{x} \in \textsc{vars}(X)$, that $\textsc{sort}(M) \underline{\propto} \textsc{sort}(X)$ and that $\widetilde{x}\#\sigma$. By induction $Q\sigma$ is well-formed. By sort preservation we get $\textsc{sort}(M\sigma) \leq \textsc{sort}(M)$, so $\textsc{sort}(M\sigma) \underline{\propto} \textsc{sort}(X)$. By preservation of patterns by non-capturing substitutions we have that $\widetilde{x} \in \textsc{vars}(X\sigma)$ and $\textsc{sort}(X\sigma) \leq \textsc{sort}(X)$, so $\textsc{sort}(M\sigma) \underline{\propto} \textsc{sort}(X\sigma)$.

The second part is by induction on the transition rules, using part 1 in the In rule. □

3. Meta-theory

As usual, the labelled operational semantics gives rise to notions of labelled bisimilarity. Similarly to the applied pi-calculus [AF01], the standard definition of bisimilarity needs to be adapted to take assertions into account. In this section, we show that both strong and weak bisimilarity satisfy the expected structural congruence laws and the standard congruence properties of name-passing process calculi. We first prove these results for calculi with a single sort (Theorem 3.12) supported by Nominal Isabelle, and then extend the result to all sorted psi-caluli (Theorem 3.16) by a manual proof. We start by recollecting the required definitions, beginning with the definition of strong labelled bisimulation on well-formed agents by Bengtson et al. [BJPV11], to which we refer for examples and more intuitions.

Definition 3.1 (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: for all Ψ' it holds that $\mathcal{R}(\Psi \otimes \Psi', P, Q)$.
(4) Simulation: for all α, P' such that $\mathrm{bn}(\alpha) \# \Psi, Q$ and $\Psi \vartriangleright P \xrightarrow{\alpha} P'$,
there exists Q' such that $\Psi \vartriangleright Q \xrightarrow{\alpha} Q'$ and $\mathcal{R}(\Psi, P', Q')$.

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

Above, (1) corresponds to the capability of a parallel observer to test the truth of a condition using **case**, while (3) models an observer taking a step and adding a new assertion Ψ' to the current environment.

We close strong bisimulation under substitutions to obtain a congruence in the usual way:

Definition 3.2 (Strong bisimulation congruence)**.** $P \sim_{\Psi} Q$ means that 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 $P \sim_{\mathbf{1}} Q$.

To illustrate the definitions of bisimulation and bisimulation congruence, we here prove a result about the **case** statement, to be used in Section 4.

Lemma 3.3 (Flatten Case)**.** *Suppose that there exists a condition* $\top \in \mathbf{C}$ *such that* $\Psi \vdash \top\widetilde{\sigma}$ *for all* Ψ *and substitution sequences* $\widetilde{\sigma}$. *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. We let $\mathcal{I} := \bigcup_{\Psi,P}\{(\Psi, P, P)\}$ be the identity relation, and

$$\mathcal{S} := \bigcup_{\Psi,\widetilde{P},\widetilde{Q},\widetilde{\phi},\widetilde{\varphi}} \{(\Psi, \mathbf{case}\ \varphi_{\top} : (\mathbf{case}\ \widetilde{\varphi} : \widetilde{P})\ []\ \widetilde{\phi} : \widetilde{Q}, \mathbf{case}\ \varphi_{\top} : \mathbf{case}\ \widetilde{\varphi} : \widetilde{P}\ []\ \widetilde{\phi} : \widetilde{Q}) : \varphi_{\top} \in \mathbf{C} \land \forall \Psi' \in \mathbf{A}.\ \Psi' \vdash \varphi_{\top}\}.$$

We prove that $\mathcal{T} := \mathcal{S} \cup \mathcal{S}^{-1} \cup \mathcal{I}$ is a bisimulation, where $\mathcal{S}^{-1} := \{(\Psi, Q, P) : (\Psi, P, Q) \in \mathcal{S}\}$. Then, $\mathcal{T}(\mathbf{1}, R\widetilde{\sigma}, R'\widetilde{\sigma})$ for all $\widetilde{\sigma}$, so $R \sim R'$ by the definition of $\sim$. The proof that $\mathcal{T}$ is a bisimulation is straightforward:

Static equivalence: The frame of a **case** agent is always $\mathbf{1}$, hence static equivalence follows by reflexivity of $\simeq$.

Symmetry: Follows by definition of $\mathcal{T}$.

Extension with arbitrary assertion: Trivial by the choice of candidate relation, since the Ψ in $\mathcal{S}$ and $\mathcal{I}$ are universally quantified.

Simulation: Trivially, any process P simulates itself. Fix $(\Psi, R, R') \in \mathcal{S}$, such that $R = \mathbf{case}\ \varphi_{\top} : (\mathbf{case}\ \widetilde{\varphi} : \widetilde{P})\ []\ \widetilde{\phi} : \widetilde{Q}$ and $R' = \mathbf{case}\ \widetilde{\varphi} : \widetilde{P}\ []\ \widetilde{\phi} : \widetilde{Q}$. Here $\Psi \vdash \varphi_{\top}$ follows by definition of $\mathcal{S}$. Since $\mathcal{T}$ includes both $\mathcal{S}$ and $\mathcal{S}^{-1}$, we must follow transitions from both R and R'.

- A transition from R via P_i can be derived as follows:
$$\textsc{Case}\ \dfrac{\textsc{Case}\ \dfrac{\Psi \vartriangleright P_i \xrightarrow{\alpha} P_i' \qquad \Psi \vdash \varphi_i}{\Psi \vartriangleright \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \xrightarrow{\alpha} P_i'} \qquad \Psi \vdash \varphi_\top}{\Psi \vartriangleright \mathbf{case}\ \varphi_\top : (\mathbf{case}\ \widetilde{\varphi} : \widetilde{P}) \mathbin{[]} \widetilde{\phi} : \widetilde{Q} \xrightarrow{\alpha} P_i'}$$
Then R' can simulate this with the following derivation:
$$\textsc{Case}\ \dfrac{\Psi \vartriangleright P_i \xrightarrow{\alpha} P_i' \qquad \Psi \vdash \varphi_i}{\Psi \vartriangleright \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \mathbin{[]} \widetilde{\phi} : \widetilde{Q} \xrightarrow{\alpha} P_i'}$$
By reflexivity of $\dot{\sim}_\Psi$, we get that $P_i' \mathrel{\dot{\sim}_\Psi} P_i'$.
- A transition from R' via Q_i can be derived as follows:
$$\textsc{Case}\ \dfrac{\Psi \vartriangleright Q_i \xrightarrow{\alpha} Q_i' \qquad \Psi \vdash \phi_i}{\Psi \vartriangleright \mathbf{case}\ \widetilde{\varphi} : \widetilde{P} \mathbin{[]} \widetilde{\phi} : \widetilde{Q} \xrightarrow{\alpha} Q_i'}$$
The process R can simulate this with the following derivation:
$$\textsc{Case}\ \dfrac{\Psi \vartriangleright Q_i \xrightarrow{\alpha} Q_i' \qquad \Psi \vdash \phi_i}{\Psi \vartriangleright \mathbf{case}\ \varphi_\top : (\mathbf{case}\ \widetilde{\varphi} : \widetilde{P}) \mathbin{[]} \widetilde{\phi} : \widetilde{Q} \xrightarrow{\alpha} Q_i'}$$
By reflexivity of $\dot{\sim}_\Psi$ we get $Q_i' \mathrel{\dot{\sim}_\Psi} Q_i'$.
- Symmetrically, R' can simulate transitions derived from R via Q_i, and R can simulate transitions derived from R' via P_i. □

Psi-calculi are also equipped with a notion of weak bisimilarity ($\dot{\approx}$) where τ-transitions cannot be observed, introduced by Bengtson et al. [JBPV10]. We here restate its definition, but refer to the original publication for examples and more motivation.

The definition of weak transitions is standard.

Definition 3.4 (Weak transitions)**.** $\Psi \vartriangleright P \Longrightarrow P'$ means that either $P = P'$ or there exists P'' such that $\Psi \vartriangleright P \xrightarrow{\tau} P''$ and $\Psi \vartriangleright P'' \Longrightarrow P'$.

For weak bisimulation we use static implication (rather than static equivalence) to compare the frames of the process pair under consideration.

Definition 3.5 (Static implication)**.** P *statically implies* Q in the environmental assertion Ψ, written $P \leq_\Psi Q$, if
$$\forall \varphi.\ \Psi \otimes \mathcal{F}(P) \vdash \varphi \ \Rightarrow\ \Psi \otimes \mathcal{F}(Q) \vdash \varphi$$

Definition 3.6 (Weak bisimulation)**.** A *weak bisimulation* $\mathcal{R}$ is a ternary relation between assertions and pairs of agents such that $\mathcal{R}(\Psi, P, Q)$ implies all of

(1) Weak static implication: for all Ψ' there exist Q', Q'' such that
$$\Psi \vartriangleright Q \Longrightarrow Q' \quad\wedge\quad \Psi \otimes \Psi' \vartriangleright Q' \Longrightarrow Q'' \quad\wedge\quad P \leq_\Psi Q' \quad\wedge\quad \mathcal{R}(\Psi \otimes \Psi', P, Q'')$$
(2) Symmetry: $\mathcal{R}(\Psi, Q, P)$
(3) Extension of arbitrary assertion: for all Ψ' it holds that $\mathcal{R}(\Psi \otimes \Psi', P, Q)$
(4) Weak simulation: for all P',
  (a) if $\Psi \vartriangleright P \xrightarrow{\tau} P'$ then $\exists Q'.\ \Psi \vartriangleright Q \Longrightarrow Q' \wedge \mathcal{R}(\Psi, P', Q')$; and

(b) for all $\Psi', \alpha \neq \tau$ such that $\mathrm{bn}(\alpha) \# \Psi, Q$, there exist Q', Q'', Q''' such that

$$\begin{array}{l} \Psi \rhd Q \Longrightarrow Q' \quad\wedge\quad \Psi \rhd Q' \xrightarrow{\alpha} Q'' \quad\wedge\quad \Psi \otimes \Psi' \rhd Q'' \Longrightarrow Q''' \\ \wedge\quad P \leq_{\Psi} Q' \quad\wedge\quad \mathcal{R}(\Psi \otimes \Psi', P', Q''') \end{array}$$

We define $P \overset{\bullet}{\approx} Q$ to mean that there exists a weak bisimulation $\mathcal{R}$ such that $\mathcal{R}(\mathbf{1}, P, Q)$ and we write $P \overset{\bullet}{\approx}_{\Psi} Q$ when there exists a weak bisimulation $\mathcal{R}$ such that $\mathcal{R}(\Psi, P, Q)$.

Above, (1) allows Q to take τ-transitions before and after enabling at least those conditions that hold in the frame of P, as per Definition 3.5. Moreover, when testing these conditions, the observer may also add an assertion Ψ' to the environment. In (4b), the observer may test the validity of conditions when matching a visible transition, and may also add an assertion as above.

To obtain a congruence from weak bisimulation, we must require that every τ-transition are simulated by a weak transition containing at least one τ-transition.

Definition 3.7. A *weak τ-bisimulation* $\mathcal{R}$ is a ternary relation between assertions and pairs of agents such that $\mathcal{R}(\Psi, P, Q)$ implies all conditions of a weak bisimulation (Definition 3.6) with 4a replaced by

(4a$'$) if $\Psi \rhd P \xrightarrow{\tau} P'$ then $\exists Q', Q''.\, \Psi \rhd Q \xrightarrow{\tau} Q' \wedge \Psi \rhd Q' \Longrightarrow Q'' \wedge \mathcal{R}(\Psi, P', Q'')$.

We then let $P \approx_{\Psi} Q$ mean that for all sequences $\widetilde{\sigma}$ of substitutions there is a weak τ-bisimulation $\mathcal{R}$ such that $\mathcal{R}(\Psi, P\widetilde{\sigma}, Q\widetilde{\sigma})$. We write $P \approx Q$ for $P \approx_{\mathbf{1}} Q$.

Lemma 3.8 (Comparing bisimulations)**.** *For all relations $\mathcal{R} \subseteq \mathbf{A} \times \mathbf{P} \times \mathbf{P}$,*

- *if $\mathcal{R}$ is a strong bisimulation then $\mathcal{R}$ is a weak τ-bisimulation.*
- *if $\mathcal{R}$ is a weak τ-bisimulation then $\mathcal{R}$ is a weak bisimulation.*

Corollary 3.9 (Comparing congruences)**.** *If $P \sim_{\Psi} Q$ then $P \approx_{\Psi} Q$.*

We seek to establish the following standard congruence and structural properties properties of strong and weak bisimulation:

Definition 3.10 (Congruence relation)**.** A relation $\mathcal{R} \subseteq \mathbf{A} \times \mathbf{P} \times \mathbf{P}$, where $(\Psi, P, Q) \in \mathcal{R}$ is written $P \mathrel{\mathcal{R}_{\Psi}} Q$, is a *congruence* iff for all Ψ, $\mathcal{R}_{\Psi}$ is an equivalence relation, and the following hold:

$$\begin{array}{lrcl}
\text{CP{\scriptsize AR}} & P \mathrel{\mathcal{R}_{\Psi}} Q & \Longrightarrow & (P \mid R) \mathrel{\mathcal{R}_{\Psi}} (Q \mid R) \\
\text{CR{\scriptsize ES}} & a \# \Psi \wedge P \mathrel{\mathcal{R}_{\Psi}} Q & \Longrightarrow & (\nu a)P \mathrel{\mathcal{R}_{\Psi}} (\nu a)Q \\
\text{CB{\scriptsize ANG}} & P \mathrel{\mathcal{R}_{\Psi}} Q & \Longrightarrow & !P \mathrel{\mathcal{R}_{\Psi}} !Q \\
\text{CC{\scriptsize ASE}} & \forall i. P_i \mathrel{\mathcal{R}_{\Psi}} Q_i & \Longrightarrow & \mathbf{case}\ [\!]\ \widetilde{\varphi} : \widetilde{P}\ \mathrel{\mathcal{R}_{\Psi}} \mathbf{case}\ [\!]\ \widetilde{\varphi} : \widetilde{Q} \\
\text{CO{\scriptsize UT}} & P \mathrel{\mathcal{R}_{\Psi}} Q & \Longrightarrow & \overline{M}\, N\,.\,P \mathrel{\mathcal{R}_{\Psi}} \overline{M}\, N\,.\,Q \\
\text{CI{\scriptsize N}} & P \mathrel{\mathcal{R}_{\Psi}} Q & \Longrightarrow & \underline{M}(\lambda\widetilde{x})X\,.\,P \mathrel{\mathcal{R}_{\Psi}} \underline{M}(\lambda\widetilde{x})X\,.\,Q
\end{array}$$

A relation that satisfies all of the above implications except CIn is called an *open congruence* if it also satisfies the following:

$$\text{CI{\scriptsize N}-2} \quad (\forall \widetilde{L}.\ P[\widetilde{x} := \widetilde{L}] \mathrel{\mathcal{R}_{\Psi}} Q[\widetilde{x} := \widetilde{L}]) \quad \Longrightarrow \quad \underline{M}(\lambda\widetilde{x})X\,.\,P \mathrel{\mathcal{R}_{\Psi}} \underline{M}(\lambda\widetilde{x})X\,.\,Q$$

A relation that does not satisfy rule CCase but is otherwise an open congruence is called a *weak open congruence*.

Definition 3.11 (Structural congruence)**.** *Structural congruence*, denoted $\equiv\, \in \mathbf{P} \times \mathbf{P}$, is the smallest relation such that $\{(\mathbf{1}, P, Q) \ : \ P \equiv Q\}$ is a congruence relation, and that satisfies the following clauses whenever $a \# Q, \widetilde{x}, M, N, X, \widetilde{\varphi}$:

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

A relation $\mathcal{R} \subseteq \mathbf{P} \times \mathbf{P}$ is *complete with respect to structual congruence* if $\equiv\, \subseteq \mathcal{R}$.

Our goal is to establish that for all Ψ the relations $\dot{\sim}_\Psi$, $\sim_\Psi$, $\dot{\approx}_\Psi$ and $\approx_\Psi$ are complete with respect to structural congruence; that $\dot{\sim}$ is an open congruence; that $\sim$ is a congruence; that $\dot{\approx}$ is a weak open congruence; and that $\approx$ is a congruence.

3.1. **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 3.12. *For trivially sorted psi-calculi, $\dot{\sim}_\Psi$, $\sim_\Psi$, $\dot{\approx}_\Psi$ and $\approx_\Psi$ are complete wrt. structural congruence for all Ψ, $\dot{\sim}$ is an open congruence, $\sim$ is a congruence, $\dot{\approx}$ is a weak open congruence, and $\approx$ is a congruence.*

These results have all been machine-checked in Isabelle [ÅP14]. The proof scripts are adapted from Bengtson's formalisation of psi calculi [Ben10]. They constitute 30579 lines of Isabelle code; Bengtson's code is 28414 lines. The same technical lemmas hold and the proof scripts are essentially identical, save for the input cases of inductive proofs and a more detailed treatment of structural congruence. This represents no more than three 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 matching construct. 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 2.11 (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.

3.2. **Arbitrary sorted psi-calculi.** We here extend the results of Theorem 3.12 to arbitrary sorted psi-calculi. The idea is to introduce an explicit error element $\bot$, resulting from application of ill-sorted substitutions. 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 $I = (\mathbf{T}_I, \mathbf{X}_I, \mathbf{C}_I, \mathbf{A}_I, \dots)$ be a sorted psi-calculus. We construct a trivially sorted psi-calculus $U(I)$ with one extra sort, `error`, and constant symbols $\bot$ and `fail`, with empty support of sort `error`, where $\bot$ is not a channel, never entailed, matches nothing and entails nothing but `fail`.

The parameters of $U(I)$ are defined by $U(I) = (\mathbf{T}_I \cup \{\bot\}, \mathbf{X}_I \cup \{(\bot, S) : S \subset_{\text{fin}} \mathcal{N}\}, \mathbf{C}_I \cup \{\bot, \mathtt{fail}\}, \mathbf{A}_I \cup \{\bot\})$. We define $\Psi \otimes \bot = \bot \otimes \Psi = \bot$ for all Ψ, and otherwise $\otimes$ is as in I. $\textsc{match}$ is the same in $U(I)$ as in I, plus $\textsc{match}(M, \widetilde{x}, (\bot, S)) = \emptyset$. Channel equivalence $\dot{\leftrightarrow}$ is the same in $U(I)$ as in I, plus $M \dot{\leftrightarrow} \bot = \bot \dot{\leftrightarrow} M = \bot \dot{\leftrightarrow} \bot = \bot$. For $\Psi \neq \bot$ we let $\Psi \vdash \varphi$ in $U(I)$ iff $\Psi \vdash \varphi$ in I, and we let $\bot \vdash \varphi$ iff $\varphi = \mathtt{fail}$. Substitution is then defined in $U(I)$ as follows:

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

Lemma 3.13. *$U(I)$ as defined above is a sorted psi-calculus, and any well-formed process P in I is well-formed in $U(I)$.*

Proof. A straight-forward application of the definitions. □

Processes in I have the same transitions in $U(I)$.

Lemma 3.14. *If P is well-formed in I and $\Psi \neq \bot$, then $\Psi \rhd P \xrightarrow{\alpha} P'$ in $U(I)$ iff $\Psi \rhd P \xrightarrow{\alpha} P'$ in I.*

Proof. By induction on the derivation of the transitions. The cases In, Out, Case and Com use the fact that $\textsc{match}$, $\vdash$ and $\dot{\leftrightarrow}$ are the same in I and $U(I)$, and that substitutions in I have the same effect when considered as substitutions in $U(I)$. □

Bisimulation in $U(I)$ coincides with bisimulation in I for processes in I.

Lemma 3.15. *Assume that P and Q are well-formed processes in I. Then $P \overset{\cdot}{\sim}_\Psi Q$ in I iff $P \overset{\cdot}{\sim}_\Psi Q$ in $U(I)$, and $P \overset{\cdot}{\approx}_\Psi Q$ in I iff $P \overset{\cdot}{\approx}_\Psi Q$ in $U(I)$.*

Proof. We show only the proof for the strong case; the weak case is similar. Let $\mathcal{R}$ be a bisimulation in $U(I)$. Then $\{(\Psi, P', Q') \in \mathcal{R} : \Psi \neq \bot \wedge P', Q' \text{ well-formed in } I\}$ is a bisimulation in I: the proof is by coinduction, using Lemma 3.14 and Theorem 2.11 in the simulation case.

Symmetrically, let $\mathcal{R}'$ be a bisimulation in I, and let $\mathcal{R}'_\bot = \{(\bot, P, Q) : \exists \Psi.(\Psi, P, Q) \in \mathcal{R}'\}$. Then $\mathcal{R}' \cup \mathcal{R}'_\bot$ is a bisimulation in $U(I)$: simulation steps from $\mathcal{R}'$ lead back to $\mathcal{R}'$ by Lemma 3.14. From $\mathcal{R}'_\bot$ there are no transitions, since $\bot$ entails no channel equivalence clauses. The other parts of Definition 3.1 are straightforward; when applying clause 3 with $\Psi' = \bot$ the resulting triple is in $\mathcal{R}'_\bot$. □

With Lemma 3.15, we can lift the congruence and the structural congruence results for trivially sorted psi-calculi to arbitrary sorted calculi:

Theorem 3.16. *All clauses of Theorem 3.12 are valid in all sorted psi-calculi.*

Proof. Fix a sorted psi-calculus I. For strong and weak bisimilarity, we show only the proofs for commutativity and congruence of the parallel operator. The other cases are analogous.

For commutativity of parallel composition, let P and Q be well-formed in I and $\Psi \neq \bot$. By Theorem 3.12, $P \mid Q \sim_\Psi Q \mid P$ holds in $U(I)$. By Definition 3.1, $(P \mid Q)\widetilde{\sigma} \mathrel{\dot{\sim}_\Psi} (Q \mid P)\widetilde{\sigma}$ in $U(I)$ for all $\widetilde{\sigma}$. By Theorem 2.11, when $\widetilde{\sigma}$ is well-sorted then $(P \mid Q)\widetilde{\sigma}$ and $(Q \mid P)\widetilde{\sigma}$ are well-formed. By Lemma 3.15, $(P|Q)\widetilde{\sigma} \mathrel{\dot{\sim}_\Psi} (Q|P)\widetilde{\sigma}$ in I for all well-formed $\widetilde{\sigma}$. $P|Q \sim_\Psi Q|P$ follows by definition. $P \mid Q \approx_\Psi Q \mid P$ follows by Corollary 3.9.

For congruence of parallel composition for bisimulation, assume $P \mathrel{\dot{\sim}_\Psi} Q$ holds in I. By Lemma 3.15, $P \mathrel{\dot{\sim}_\Psi} Q$ holds in $U(I)$. Theorem 3.12 thus yields $P \mid R \mathrel{\dot{\sim}_\Psi} Q \mid R$ in $U(I)$, and Lemma 3.15 yields the same in I. The same argument shows that $P \mathrel{\dot{\approx}_\Psi} Q$ implies $P \mid R \mathrel{\dot{\approx}_\Psi} Q \mid R$ in I.

This approach does not work for proving congruence properties for $\sim$ or $\approx$, since the closure of bisimilarity under well-sorted substitutions does not imply its closure under ill-sorted substitutions: consider a sorted psi-calculus I such that $\mathbf{0} \sim (\!|\mathbf{1}|\!)$. This equation does not hold in $U(I)$: if σ is ill-sorted then $\mathbf{1}\sigma = \bot$, but $\mathbf{0} \mathrel{\dot{\sim}} (\!|\bot|\!)$ does not hold since only $\bot$ entails `fail`. Instead, we have performed direct proofs: they are identical, line by line, to the proofs in the trivially sorted case (cf. [Ben10]).

□

4. Representing Standard Process Calculi

We here consider psi-calculi corresponding to some variants of popular process calculi. One main point of our work is that we can represent other calculi directly as psi-calculi, without elaborate coding schemes. In the original psi-calculi we could in this way directly represent the monadic pi-calculus, but for the other calculi presented below a corresponding unsorted psi-calculus would contain terms with no counterpart in the represented calculus, as explained in Section 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).

Because of the simplicity of the mapping and the strength of the correspondence we say that psi-calculi *represent* other process calculi, in contrast to *encoding* them. A representation is significantly stronger than standard correspondences, such as the approach to encodability proposed by Gorla [Gor10]. Gorla's criteria aim to capture the property that one language can encode the behaviour of another using some (possibly elaborate) protocol, while our criteria aim to capture the property that two languages are for all practical purposes one and the same.

Definition 4.1. A psi-calculus is a *representation* of a process calculus with processes $P \in \mathcal{P}$ and labelled transition system $\rightarrow \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$, if there exist an equivariant map $[\![\cdot]\!]$ from $\mathcal{P}$ to psi-calculus processes and an equivariant relation $\cong$ between $\mathcal{A}$ and psi-calculus actions that preserves the kind (input, output, tau) and subject of actions, such that

(1) $[\![\cdot]\!]$ is a simple homomorphism, i.e., for each process constructor f of $\mathcal{P}$ there is an equivariant psi-calculus context C such that $[\![f(P_1, \dots, P_n)]\!] = C[[\![P_1]\!], \dots, [\![P_n]\!]]$.

(2) $[\![\cdot]\!]$ is a strong operational correspondence (modulo structural equivalence), i.e.,
   (a) whenever $P \xrightarrow{\beta} Q$ then $[\![P]\!] \xrightarrow{\alpha} P'$ such that $[\![Q]\!] \equiv P'$ and $\beta \approxeq \alpha$; and
   (b) whenever $[\![P]\!] \xrightarrow{\alpha} P'$ then $P \xrightarrow{\beta} Q$ such that $[\![Q]\!] \equiv P'$ and $\beta \approxeq \alpha$.

A representation is *complete* if it additionally satisfies

(3) $[\![\cdot]\!]$ is surjective modulo strong bisimulation congruence, i.e., for each psi process P there is $Q \in \mathcal{P}$ such that $P \sim [\![Q]\!]$.

Briefly, the differences to Gorla's criteria are as follows:

- In Gorla's approach, the contexts that process constructors are translated to may fix certain names, or translate one name into several names, in accordance with a renaming policy. Our approach admits no such special treatment of names.
- Gorla requires the translation function to be name invariant up-to the renaming policy. We require equivariance, which corresponds to name invariance up-to the policy of renaming every name to itself.
- Gorla uses three criteria for semantic correspondence: weak operational correspondence modulo some equivalence for silent transitions, that the translation does not introduce divergence, and that reducibility to a success process in the source and target processes coincides. Clearly strong operational correspondence modulo structural equivalence implies all of these criteria.
- Our surjectivity requirement implies that the target language cannot express more behaviours than the source language, something that is not considered in Gorla's approach.

Our use of structural equivalence in the operational correspondence allows to admit representations of calculi that use a structural congruence rule to define a labelled semantics (cf. Section 4.4).

Below, for simplicity we let the assertions be the singleton $\{\mathbf{1}\}$ in all examples, with $\mathbf{1} \vdash \top$ and $\mathbf{1} \nvdash \bot$. We use the standard notion of simultaneous substitution, and let $\textsc{match}(M, \widetilde{x}, X) = \emptyset$ where not otherwise defined. Proofs of lemmas and theorems can be found in Appendix A.

4.1. **Unsorted Polyadic pi-calculus.** In the polyadic pi-calculus [Mil93] the only values that can be transmitted between agents are tuples of names. Tuples cannot be nested. The processes are defined as follows

$$\boxed{P, Q \quad ::= \quad \mathbf{0} \ \mid \ x(\tilde{y}).P \ \mid \ \overline{x}\langle\tilde{y}\rangle.P \ \mid \ [a = b]P \ \mid \ \nu x\, P \ \mid \ !P \ \mid \ P \,|\, Q \ \mid \ P + Q}$$

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 $a(x).P \ \mid \ \overline{a}\langle y, z\rangle.Q$ is a valid agent. This agent has no communication action since the lengths of the tuples mismatch.

We now present the psi-calculus **PPI**, which we will show represents the polyadic pi-calculus.

PPI

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

This being our first substantial example, we give a detailed explanation of the new instance parameters. Patterns $\mathbf{X}$ are finite vectors of distinct names. The sorts $\mathcal{S}$ are `chan` for channels and `tup` for tuples (of names); the only sort of names $\mathcal{S}_{\mathcal{N}}$ is channels, as is the sort of restricted names. The only sort of substitutions ($\prec$) are channels for channels; the only sort of sending ($\overline{\propto}$) and receiving ($\underline{\propto}$) is tuples over channels. In an input prefix all names in the tuple must be bound (VARS) and a vector of names $\widetilde{a}$ matches a pattern $\widetilde{y}$ if the lengths match and all names in the pattern are bound (in some arbitrary order).

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 $\texttt{chan} \ \underline{\propto}\ \texttt{tup}$ and $\texttt{chan}\ \overline{\propto}\ \texttt{tup}$, 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.

To prove that **PPI** is a psi-calculus, we need to check the requisites on the parameters (data types and operations) defined above. Clearly the parameters are all equivariant, since no names appear free in their definitions. For the original psi-calculus parameters (Definition 2.1), the requisites are symmetry and transitivity of channel equivalence, which hold because of the same properties of (entailment of) name equality, and abelian monoid laws and compositionality for assertion composition, which trivially hold since $\mathbf{A} = \{\mathbf{1}\}$. The standard notion of simultaneous substitution of names for names preserves sorts, and also satisfies the other requirements of Definition 2.4. To check the requisites on pattern matching (Definition 2.5), it is easy to see that MATCH generates only well-sorted substitutions (of names for names), and that $\mathrm{n}(\widetilde{b}) = \mathrm{n}(\langle\widetilde{a}\rangle)$ whenever $\widetilde{b} \in \textsc{match}(\langle\widetilde{a}\rangle, \widetilde{x}, \langle\widetilde{y}\rangle)$ Finally, for all name swappings $(\widetilde{x}\ \widetilde{y})$ we have $\textsc{match}(\langle\widetilde{a}\rangle, \widetilde{x}, \langle\widetilde{z}\rangle) = \textsc{match}(\langle\widetilde{a}\rangle, \widetilde{y}, (\widetilde{x}\ \widetilde{y}) \cdot \langle\widetilde{z}\rangle)$.

PPI is a direct representation of the polyadic pi-calculus as presented by Sangiorgi [San93] (with replication instead of process constants).

Definition 4.2 (Polyadic Pi-Calculus to **PPI**).
Let $[\![\cdot]\!]$ be the function that maps the polyadic pi-calculus to **PPI** processes as follows. The function $[\![\cdot]\!]$ is homomorphic for **0**, restriction, replication and parallel composition, and is otherwise defined as follows:

$$\begin{array}{rcl}
[\![P + Q]\!] & = & \textbf{case } \top : [\![P]\!] \ [\!]\ \top : [\![Q]\!] \\
[\![[x = y]P]\!] & = & \textbf{case } x = y : [\![P]\!] \\
[\![x(\tilde{y}).P]\!] & = & \underline{x}(\lambda\tilde{y})\langle\tilde{y}\rangle.[\![P]\!] \\
[\![\overline{x}\langle\tilde{y}\rangle.P]\!] & = & \overline{x}\langle\tilde{y}\rangle.[\![P]\!]
\end{array}$$

Similarly, we also translate the actions of polyadic pi-calculus. Here each action corresponds to a set of psi actions, since in a pi-calculus output label "the order of the bound names is immaterial" [SW01, p. 129], which is not the case in psi-calculi.

$$\begin{array}{rcl}
[\![(\nu\tilde{y})\overline{x}\langle\tilde{z}\rangle]\!] & = & \{\overline{x}\ (\nu\widetilde{y}')\ \langle\widetilde{z}\rangle\ :\ \widetilde{y}' = \pi\cdot\widetilde{y}\} \\
[\![x\langle\tilde{z}\rangle]\!] & = & \{\underline{x}\ \langle\tilde{z}\rangle\} \\
[\![\tau]\!] & = & \{\tau\}
\end{array}$$

Although the binders in bound output actions are ordered in psi-calculi, they can be arbitrarily reordered.

Lemma 4.3. *If $\Psi \rhd P \xrightarrow{\overline{M}\,(\nu\widetilde{a})\,N} Q$ then $\Psi \rhd P \xrightarrow{\overline{M}\,(\nu\pi\cdot\widetilde{a})\,N} Q$*

Proof. By induction on the derivation of the transition. The base case is trivial. In the Open rule, we use the induction hypothesis to reorder the bound names in the premise as desired; we can then add the opened name at any position in the action in the conclusion of the rule. The other induction cases are trivial. □

We can now show that $[\![\cdot]\!]$ is a strong operational correspondence.

Theorem 4.4. *If P and Q are polyadic pi-calculus processes, then:*

(1) *If $P \xrightarrow{\beta} P'$ then for all $\alpha \in [\![\beta]\!]$ we have $[\![P]\!] \xrightarrow{\alpha} [\![P']\!]$*
(2) *If $[\![P]\!] \xrightarrow{\alpha} P''$ then $P \xrightarrow{\beta} P'$ such that $\alpha \in [\![\beta]\!]$ and $[\![P']\!] = P''$*

Proof. By induction on the length of derivation of the transitions, using Lemma 4.3 in the **OPEN** case of (1). □

We have now shown that the polyadic pi-calculus can be embedded in **PPI**, with an embedding $[\![\cdot]\!]$ that is a strong operational correspondence.

In order to investigate surjectivity properties of the embedding $[\![\cdot]\!]$, we also define a translation $\overline{P}$ in the other direction.

Definition 4.5 (**PPi** to Polyadic Pi-Calculus)**.** The translation $\bar{\cdot}$ is homomorphic for $\mathbf{0}$, restriction, replication and parallel composition, and is otherwise defined as follows:

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

where condition-guarded processes are translated as

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

Above, note that the order of the binders in input prefixes is ignored. To show that the reverse translation is an inverse of $[\![\cdot]\!]$ modulo bisimilarity, we need to prove that their order does not matter.

Lemma 4.6. *In* **PPI**, *$\underline{x}(\lambda\widetilde{y})\langle\widetilde{z}\rangle.P \sim \underline{x}(\lambda\widetilde{z})\langle\widetilde{z}\rangle.P$.*

Proof. Straightforward from the definitions of MATCH and substitution on patterns. □

We now show that the embeddings $\bar{\cdot}$ and $[\![\cdot]\!]$ are inverses, modulo bisimilarity.

Theorem 4.7. *If P is a* **PPI** *process, then $P \sim [\![\overline{P}]\!]$.*

Proof. By structural induction on P. The input case uses Lemma 4.6. For **case** agents, we use an inner induction on the number of branches, with Lemma 3.3 applied in the induction case. □

Let the relation $\sim_e^c$ be an early congruence of polyadic pi-calculus agents as defined in [San93]. Then we have

Corollary 4.8. *If P is a polyadic pi-calculus process, then $P \sim_e^c \overline{[\![P]\!]}$.*

We also have

Corollary 4.9. *If P and Q are polyadic pi-calculus process, then $P \sim_e^c Q$ (i.e., P and Q are early labelled congruent) iff $[\![P]\!] \sim [\![Q]\!]$.*

Proof. Follows from the strong operational correspondence of Theorem 4.4, and $[\![\cdot]\!]$ commuting with substitutions. □

This shows that every **PPI** process corresponds to a polyadic pi-calculus process, modulo strong bisimulation congruence, since $\bar{\cdot}$ is surjective on the bisimulation classes of polyadic pi-calculus, and the inverse of $[\![\cdot]\!]$. In other words, **PPI** is a *representation.*

Theorem 4.10. **PPI** *is a complete representation of the polyadic pi-calculus.*

Proof. We let $\beta \approxeq \alpha$ iff $\alpha \in [\![\beta]\!]$.

(1) $[\![\cdot]\!]$ is a simple homomorphism by definition.
(2) $[\![\cdot]\!]$ is a strong operational correspondence by Theorem 4.4.
(3) $[\![\cdot]\!]$ is surjective modulo strong bisimulation congruence by Theorem 4.7. □

4.2. **LINDA** [Gel85]**.** A process calculus with LINDA-like pattern matching can easily be obtained from the **PPI** calculus, by modifying the possible binding names in patterns.

LINDA

Everything as in **PPI** except:
$\mathbf{X} = \{\langle\widetilde{a}\rangle : \widetilde{a} \subset_{\text{fin}} \mathcal{N}\}$
$\textsc{vars}(\langle\widetilde{a}\rangle) = \mathcal{P}(\widetilde{a})$
$\textsc{match}(\langle\widetilde{a}\rangle, \widetilde{x}, \langle\widetilde{y}\rangle) = \{\widetilde{c} : \langle\widetilde{a}\rangle = \langle\widetilde{y}\rangle[\widetilde{x} := \widetilde{c}]\}$

Here, any subset of the names occurring in a pattern may be bound in the input prefix; this allows to only receive messages with particular values at certain positions (sometimes called "structured names" [Gel85]) We also do not require patterns to be linear, i.e., the same variable may occur more than once in a pattern, and the pattern only matches a tuple if each occurrence of the variable corresponds to the same name in the tuple.

As an example, $\underline{a}(\lambda x)\langle x, x, z\rangle.P \mid \overline{a}\langle c, c, z\rangle.Q \xrightarrow{\tau} P[x := c] \mid Q$ while the agent $\underline{a}(\lambda x)\langle x, x, z\rangle.P \mid \overline{a}\langle c, d, z\rangle.Q$ has no τ transition.

To prove that **LINDA** is a psi-calculus, the interesting case is the preservation of variables of substitution on patterns in Definition 2.4, i.e., that $\widetilde{x} \in \textsc{vars}(\langle\widetilde{y}\rangle)$ and $\widetilde{x}\#\sigma$ implies $\widetilde{x} \in \textsc{vars}(\langle\widetilde{y}\rangle\sigma)$. This holds because standard substitution preserves names and structure: if $x \in \widetilde{y}$ and $x\#\sigma$, then there is $\widetilde{z}$ such that $\langle\widetilde{y}\rangle\sigma = \langle\widetilde{z}\rangle$ and $x \in \widetilde{z}$.

4.3. **Sorted polyadic pi-calculus.** Milner's classic sorting [Mil93] 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 in its domain. 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:
$\mathcal{S}_{\mathcal{N}} = \mathcal{S}_{\nu} = \mathrm{S}$ $\qquad \mathcal{S} = \mathrm{S}^*$
$\prec = \{(s,s) : s \in \mathrm{S}\}$ $\qquad \overline{\propto} = \underline{\propto} = \{(s, \mathtt{ob}(s)) : s \in \mathrm{S}\}$
$\textsc{sort}(\langle a_1, \ldots, a_n\rangle) = \textsc{sort}(a_1), \ldots, \textsc{sort}(a_n)$
$\textsc{match}(\langle \widetilde{a}\rangle, \widetilde{x}, \langle \widetilde{y}\rangle) = \{\pi \cdot \widetilde{a}\}$ $\quad$ if $\widetilde{x} = \pi \cdot \widetilde{y}$ and $\textsc{sort}(\langle \widetilde{a}\rangle) = \textsc{sort}(\langle \widetilde{y}\rangle)$

We need to show that MATCH always generates well-sorted substitutions: this holds since whenever $\widetilde{c} \in \textsc{match}(\langle \widetilde{a}\rangle, \widetilde{x}, \langle \widetilde{y}\rangle)$ we have that $[\widetilde{x} := \widetilde{c}] = [\pi \cdot \widetilde{y} := \pi \cdot \widetilde{a}]$ and $\textsc{sort}(y_i) = \textsc{sort}(a_i)$ for all i.

As an example, let $\textsc{sort}(a) = s$ with $\mathtt{ob}(s) = t_1, t_2$ and $\textsc{sort}(x) = t_1$ with $\mathtt{ob}(t_1) = t_2$ and $\textsc{sort}(y) = t_2$ 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 $\textsc{vars}(x, y) = \{\{x, y\}\}$.

A formal comparison with the system in [Mil93] 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 4.11. *P is well-sorted iff $[\![P]\!]$ is well-formed.*

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

Theorem 4.12. SORTEDPPI *is a complete representation of the sorted polyadic pi-calculus.*

Proof. The operational correspondence in Theorem 4.4 still holds when restricted to well-formed agents. The inverse translation $\bar{\cdot}$ maps well-formed agents to well-sorted processes, so the surjectivity result in Theorem 4.7 still applies. □

4.4. **Polyadic synchronisation pi-calculus.** Carbone and Maffeis [CM03] 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. The processes of ${}^{e}\pi$ is defined as follows.

$$\begin{array}{rcl} P, Q & ::= & \mathbf{0} \mid \Sigma_i \alpha_i . P_i \mid P \,|\, Q \mid (\nu a)P \mid\ !P \\ \alpha & ::= & \tilde{a}(x) \mid \tilde{a}\langle b\rangle \end{array}$$

In order to represent ${}^{e}\pi$, only minor modifications to the representation of the polyadic pi-calculus in Section 4.1 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 $a = b$, since they can be encoded (see [CM03]).

PSPI	
Everything as in **PPI** except:	
$\mathbf{C} = \{\top, \bot\}$	$\widetilde{a} \leftrightarrow \widetilde{b}$ is $\top$ if $\widetilde{a} = \widetilde{b}$, and $\bot$ otherwise
$\mathbf{X} = \mathcal{N}$	$\textsc{vars}(x) = \{\{x\}\}$
$\overline{\propto} = \underline{\propto} = \{(\mathtt{tup}, \mathtt{chan})\}$	$\textsc{match}(a, x, x) = \{a\}$

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 communication over a restricted fresh name. The ${}^{e}\pi$ calculus also uses an operational semantics with late input, unlike psi-calculi. In order to yield a representation, we consider an early version $\longrightarrow^{e}$ of the semantics, obtained by turning bound input actions into free input actions at top-level.

$$\textsc{eIn}\ \frac{P \xrightarrow{\tilde{x}(y)} P'}{P \xrightarrow{\tilde{x}\ z}{}^{e}\ P'\{z/y\}} \qquad \textsc{Out}\ \frac{P \xrightarrow{\tilde{x}\langle c\rangle} P'}{P \xrightarrow{\tilde{x}\langle c\rangle}{}^{e}\ P'} \qquad \text{B}\textsc{Out}\ \frac{P \xrightarrow{\tilde{x}\langle\nu c\rangle} P'}{P \xrightarrow{\tilde{x}\langle\nu c\rangle}{}^{e}\ P'} \qquad \textsc{Tau}\ \frac{P \xrightarrow{\tau} P'}{P \xrightarrow{\tau}{}^{e}\ P'}$$

Definition 4.13 (Polyadic synchronisation pi-calculus to **PSPI**). $\llbracket\cdot\rrbracket$ is homomorphic for $\mathbf{0}$, restriction, replication and parallel composition, and is otherwise defined as follows:

$$\begin{array}{rcl} \llbracket \Sigma_i \alpha_i . P_i \rrbracket & = & \mathbf{case}\ \top_i : \llbracket \alpha_i . P_i \rrbracket \\ \llbracket \widetilde{x}(y).P \rrbracket & = & \underline{\langle\widetilde{x}\rangle}(\lambda y)y.\llbracket P \rrbracket \\ \llbracket \widetilde{x}\langle y\rangle.P \rrbracket & = & \overline{\langle\widetilde{x}\rangle}\ y.\llbracket P \rrbracket \end{array}$$

We translate bound and free output, free input, and tau actions in the following way.

$$\begin{array}{rcl} \llbracket \tilde{x}\langle\nu c\rangle \rrbracket & = & \overline{\langle\tilde{x}\rangle}\ (\nu c)\ c \\ \llbracket \tilde{x}\langle c\rangle \rrbracket & = & \overline{\langle\tilde{x}\rangle}\ c \\ \llbracket \tilde{x}\ y \rrbracket & = & \langle\widetilde{x}\rangle\ y \\ \llbracket \tau \rrbracket & = & \tau \end{array}$$

The transition system in ${}^{e}\pi$ is given up to structural congruence, i.e., for all α we have $\xrightarrow{\alpha} = (\equiv\xrightarrow{\alpha}\equiv)$.

Definition 4.14. $\equiv$ is the least congruence satisfying alpha conversion, the commutative monoidal laws with respect to both ($|$,0) and (+,0) and the following axioms[1]:

$$(\nu x)P \mid Q \equiv (\nu x)(P \mid Q) \text{ if } x\#Q \qquad\qquad (\nu x)P \equiv P \text{ if } x\#P$$

The proofs of operational correspondence are similar to the polyadic pi-calculus case. We have the following initial results for late input actions.

Lemma 4.15.

(1) *If $P \xrightarrow{\tilde{x}(y)} P'$ then for all z, $\llbracket P\rrbracket \xrightarrow{\langle\tilde{x}\rangle\ z} P''$ where $P'' \equiv \llbracket P'\rrbracket[y := z]$.*

(2) *If $\llbracket P\rrbracket \xrightarrow{\langle\tilde{x}\rangle\ z} P''$ then for all $y\#P$, $P \xrightarrow{\tilde{x}(y)} P'$ where $\llbracket P'\{z/y\}\rrbracket = P''$.*

[1] The original definition of $\equiv$ [CM03] includes an additional axiom $[x = x]P \equiv P$ allowing to contract successful matches, but this axiom is omitted here since the ${}^{e}\pi$ calculus does not include the match construct. Unusually, the definition of $\equiv$ does not admit commuting restrictions, i.e., $(\nu x)(\nu y)P \not\equiv (\nu y)(\nu x)P$.

Proof. By induction on the derivation of the transitions. □

This in turn yields the desired operational correpondence.

Theorem 4.16.

(1) *If* $P \xrightarrow{\alpha}^{e} P'$ *and* $\alpha \neq \tilde{x}(y)$, *then* $[\![P]\!] \xrightarrow{[\![\alpha]\!]} P''$ *where* $P'' \equiv [\![P']\!]$.

(2) *If* $[\![P]\!] \xrightarrow{\alpha'} P''$, *then* $P \xrightarrow{\alpha}^{e} P'$ *where* $[\![\alpha]\!] = \alpha'$ *and* $[\![P']\!] = P''$.

Proof. By induction on the derivation of the transitions. □

Again, these results lead us to say that the polyadic synchronization pi-calculus can be *represented* as a psi-calculus.

Theorem 4.17. **PSPI** *is a representation of the polyadic synchronization pi-calculus.*

Proof. We let $\beta \approxeq \alpha$ iff $\alpha = [\![\beta]\!]$.

(1) $[\![\cdot]\!]$ is a simple homomorphism by definition.

(2) $[\![\cdot]\!]$ is a strong operational correspondence by Theorem 4.4. □

To investigate the surjectivity properties of $[\![\cdot]\!]$, we need to consider the fact that polyadic synchronization pi has only mixed (i.e., prefix-guarded) choice.

Definition 4.18 (Case-guarded)**.** A **PSPI** process is case-guarded if in all its subterms of the form **case** $\varphi_1 : P_1 \mathrel{[\!]} \cdots \mathrel{[\!]} \varphi_n : P_n$, for all $i \in \{1, \dots, n\}$, $\varphi_i = \top$ implies $P_i = \overline{M}\ N.Q$ or $P_i = \underline{M}(\lambda\widetilde{x})X.Q$.

We define the translation $\overline{R}$ from case-guarded **PSPI** processes to ${}^{e}\pi$ as the translation with the same name from **PPI**, except that $\bot$-guarded branches of **case** statements are discarded.

Theorem 4.19. *For all case-guarded* **PSPI** *processes* R *we have* $R \sim [\![\overline{R}]\!]$.

Proof. By structural induction on R. For **case** agents, we use an inner induction on the number of branches, with Lemma 3.3 applied in the induction case. □

Corollary 4.20. *If* P *is a polyadic synchronization pi-calculus process, then* $P \mathrel{\dot{\sim}} \overline{[\![P]\!]}$.

Corollary 4.21. *For all* ${}^{e}\pi$ *processes* P, Q, $P \mathrel{\dot{\sim}} Q$ *(i.e.,* P *and* Q *are early labelled congruent) iff* $[\![P]\!] \sim [\![Q]\!]$.

Proof. By strong operational correspondence 4.16, and $[\![\cdot]\!]$ commuting with substitutions. □

We thus have that the case-guarded **PSPI** processes correspond to polyadic synchronization pi, modulo flattening and structural congruence.

4.5. **Value-passing CCS.** Value-passing CCS [Mil89] is an extension of pure CCS to admit arbitrary data from some set **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 $a(x) \,.\, \mathbf{if}\ x > 3\ \mathbf{then}\ P$. Formally, the value-passing CCS processes are defined by the following grammar with x, y ranging over names, v over values, b over boolean expressions, and L over set of names.

$$\boxed{P, Q \ ::= \ x(y).P \ \mid \ \overline{x}(v).P \ \mid \ \Sigma_i\ P_i \ \mid \ \mathbf{if}\ b\ \mathbf{then}\ P \ \mid \ P \setminus L \ \mid \ P \,|\, Q \ \mid \ !P \ \mid \ \mathbf{0}}$$

To represent this as a psi-calculus we assume an arbitrary set of expressions $e \in \mathbf{E}$ including at least the values $\mathbf{V}$. A subset of $\mathbf{E}$ is the boolean expressions $b \in \mathbf{E_B}$. 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 $v \in \mathbf{V}$ are closed and have sort `value`; all other expressions have sort `exp`. The boolean values are $\mathbf{V_B} := \mathbf{V} \cap \mathbf{E_B} = \{\top, \bot\}$, and $\mathbf{1} \vdash \top$ but $\neg(\mathbf{1} \vdash \bot)$. 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 $E(2+3) = 5$. We define substitution on expressions to use evaluation, i.e. $e[\widetilde{x} := \widetilde{V}] = E(e\{\widetilde{V}/\widetilde{x}\})$. As an example, $(x+3)[x := 2] = E((x+3)\{2/x\}) = E(2+3) = 5$. We use the single-variable patterns of Example 2.6.

VPCCS

$\mathbf{T} = \mathcal{N} \cup \mathbf{E}$	$\mathcal{S}_{\mathcal{N}} = \{\mathtt{chan}, \mathtt{exp}\}$
$\mathbf{C} = \mathbf{E_B}$	$\mathcal{S} = \mathcal{S}_{\mathcal{N}} \cup \{\mathtt{value}\}$
$\mathbf{A} = \{\mathbf{1}\}$	$v \in \mathbf{V} \Rightarrow \mathrm{SORT}(v) = \mathtt{value}$
$\mathbf{X} = \mathcal{N}$	$e \in \mathbf{E} \setminus \mathbf{V} \Rightarrow \mathrm{SORT}(e) = \mathtt{exp}$
$a \mathrel{\dot{\leftrightarrow}} a = \top$	$e \in \mathbf{E} \Rightarrow e[\widetilde{x} := \widetilde{M}] = E(e\{\widetilde{M}/\widetilde{x}\})$
$e \mathrel{\dot{\leftrightarrow}} e' = \bot$ otherwise	$\prec = \{(\mathtt{exp}, \mathtt{value})\}$
$\mathrm{MATCH}(v, a, a) = \{v\}$ if $v \in \mathbf{V}$	$\mathcal{S}_\nu = \{\mathtt{chan}\}$
$\mathrm{VARS}(a) = \{a\}$	$\overline{\propto} = \underline{\propto} = \{(\mathtt{chan}, \mathtt{exp}), (\mathtt{chan}, \mathtt{value})\}$

Closed value-passing CCS processes correspond to **VPCCS** agents P where all free names are of sort `chan`. To prove that **VPCCS** is a psi-calculus, the interesting case is when the sort of a term is changed by substitution: let e be an open term, and σ a substitution such that $\mathrm{n}(e) \subseteq \mathrm{dom}(\sigma)$. Here $\mathrm{SORT}(e) = \mathtt{exp}$ and $\mathrm{SORT}(e\sigma) = \mathtt{value}$; this satisfies Definition 2.4 since $\mathtt{value} \leq \mathtt{exp}$ in the subsorting preorder (here $\mathtt{exp} \leq \mathtt{value}$ also holds, but is immaterial since there are no names of sort `value`).

We show that **VPCCS** represents value-passing CCS as defined by Milner [Mil89], with the following modifications:

- We use replication instead of process constants.
- 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 cofinite 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 the relabelling construct $P[f]$ of CCS 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.

- We only allow finite sets L in restrictions $P \setminus L$. With finite sums, this results in no loss of expressivity since agents have finite support.

Milner's restrictions are of sets of names, which we represent as a sequence of ν-binders. To create a unique such sequence from L, we assume an injective and support-preserving function $\overrightarrow{\cdot} : \mathcal{P}_{\mathrm{fin}}(\mathcal{N}_{\mathtt{chan}}) \to (\mathcal{N}_{\mathtt{chan}})^*$. For instance, $\overrightarrow{L}$ may be defined as sorting the names in L according to some total order on $\mathcal{N}_{\mathtt{chan}}$, which is always available since $\mathcal{N}_{\mathtt{chan}}$ is countable.

The mapping $\llbracket\cdot\rrbracket$ from value-passing CCS into **VPCCS** is defined homomorphically on parallel composition, output and $\mathbf{0}$, and otherwise as follows.

$$\begin{aligned}
\llbracket x(y).P \rrbracket &= \underline{x}(\lambda y)y.\llbracket P \rrbracket \\
\llbracket \Sigma_i\, P_i \rrbracket &= \textbf{case}\; \top : \llbracket P_1 \rrbracket \;[]\; \cdots \;[]\; \top : \llbracket P_i \rrbracket \\
\llbracket \textbf{if}\; b \;\textbf{then}\; P \rrbracket &= \textbf{case}\; b : \llbracket P \rrbracket \\
\llbracket P \setminus L \rrbracket &= (\nu \overrightarrow{L})\llbracket P \rrbracket
\end{aligned}$$

We translate the value-passing CCS actions as follows

$$\begin{aligned}
\llbracket x(v) \rrbracket &= \underline{x}\; v \\
\llbracket \overline{x}(v) \rrbracket &= \overline{x}\; v \\
\llbracket \tau \rrbracket &= \tau
\end{aligned}$$

As an example, in a version of **VPCCS** where the expressions $\mathbf{E}$ include natural numbers and operations on those,

$$\begin{aligned}
&\underline{a}(\lambda y)x\,.\,\textbf{case}\; x > 3 : \overline{c}(x+3) \\
&\xrightarrow{\underline{a}\,4} \quad (\textbf{case}\; x > 3 : \overline{c}(x+3))[x := 4] \\
&= \quad \textbf{case}\; E((x>3)\{{}^{4}\!/\!{}_{x}\}) : \overline{c}(E((x+3)\{{}^{4}\!/\!{}_{x}\})) \\
&= \quad \textbf{case}\; E(4>3) : \overline{c}(E(4+3)) \\
&= \quad \textbf{case}\; \top : \overline{c}7 \\
&\xrightarrow{\overline{c}\,7} \quad \mathbf{0}
\end{aligned}$$

In our psi semantics, expressions in processes are evaluated when they are closed by reception of variables (e.g. in the first transition above), while Milner simply identifies closed expressions with their values [Mil89, p55f].

Lemma 4.22. *If P is a closed* **VPCCS** *process and $P \xrightarrow{\alpha} P'$, then P' is closed.*

Theorem 4.23. *If P and Q are closed value-passing CCS processes, then*

(1) *if $P \xrightarrow{\alpha} P'$ then $\llbracket P \rrbracket \xrightarrow{\llbracket\alpha\rrbracket} \llbracket P' \rrbracket$; and*
(2) *if $\llbracket P \rrbracket \xrightarrow{\alpha'} P''$ then $P \xrightarrow{\alpha} P'$ where $\llbracket\alpha\rrbracket = \alpha'$ and $\llbracket P' \rrbracket = P''$.*

Proof. By induction on the derivations of P' and P'', respectively. The full proof is given in Appendix A.3. □

As before, this yields a representation theorem.

Theorem 4.24. **VPCCS** *is a representation of the closed agents of value-passing CCS (modulo the modifications described above).*

Proof. We let $\beta \approxeq \alpha$ iff $\alpha = \llbracket\beta\rrbracket$.

(1) $\llbracket\cdot\rrbracket$ is a simple homomorphism by definition.
(2) $\llbracket\cdot\rrbracket$ is a strong operational correspondence by Theorem 4.23. □

To investigate the surjectivity of the encoding, we let $\mathcal{P} = \{P \ : \ \textsc{sort}(\mathrm{n}(P)) \subseteq \{\mathtt{chan}\}\}$ be the **VPCCS** processes where all fre names are of channel sort.

Lemma 4.25. *If $P \in \mathcal{P}$, then there is a CCS process Q such that $P \sim [\![Q]\!]$.*

Proof. As before, we define an inverse translation $\bar{\cdot}$, that is homomorphic except for

$$\overline{\mathbf{case}\ b_1 : P_1\ [\!]\ \cdots\ [\!]\ b_i : P_i} = (\mathbf{if}\ b_1\ \mathbf{then}\ \overline{P_1}) + \cdots + (\mathbf{if}\ b_i\ \mathbf{then}\ \overline{P_i})$$

Using Lemma 3.3, we get $P \sim [\![\overline{P}]\!]$. □

Example 4.26 (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 be substituted for other channel names, and they can be sent and received along channel names.

5. Advanced Data Structures

We here demonstrate that we can accommodate a variety of term structures for data and 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. We focus on our new contribution in the patterns and sorts, and therefore make the following definitions that are common to all the examples (unless explicitly otherwise defined).

$\mathbf{A} = \{\mathbf{1}\}$	$\mathbf{1} \otimes \mathbf{1} = \mathbf{1}$
$\mathbf{C} = \{\top, \bot\}$	$\vdash = \{(\mathbf{1}, \top)\}$
$M \dot{\leftrightarrow} M = \top$	$M \dot{\leftrightarrow} N = \bot$ if $M \neq N$
$\textsc{match}(M, \widetilde{x}, X) = \emptyset$	$\prec = \{(s, s) \ : \ s \in \mathcal{S}\}$
$\underline{\propto} = \overline{\propto} = \mathcal{S} \times \mathcal{S}$	$\mathcal{S}_\nu = \mathcal{S}_\mathcal{N} = \mathcal{S}$

If t and u are from some term algebra, we write $t \preceq u$ when t is a (non-strict) subterm of u.

5.1. **Convergent rewrite systems on terms.**

In Example 4.26, the value language consisted of closed terms, with an opaque notion of evaluation. We can instead work with terms containing names and consider deterministic computations specified by a convergent rewrite system. The interesting difference is in which terms are admissible as patterns, and which choices of $\textsc{vars}(X)$ are valid. We first give a general definition and then give a concrete instance in Example 5.1.

Let Σ be a sorted signature with sorts $\mathcal{S}$, and $\cdot \Downarrow$ be normalization with respect to a convergent sort-preserving rewrite system on the nominal term algebra over $\mathcal{N}$ generated by the signature Σ. We let terms M range over the range of $\Downarrow$, i.e., the normal forms. 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 term M is stable if for all ρ, $M\rho\Downarrow = M\rho$. The patterns are all instances of stable terms, i.e., $X = M\rho$ where M is stable. Such a pattern X can bind any combination of names occurring in M but not in ρ. As an example, any term M is a pattern (since any name x is stable and $M = x\{M/x\}$) that can be used to match the term M itself (since $\emptyset \subseteq \mathrm{n}(x) \setminus \mathrm{n}(M, x) = \emptyset$).

REWRITE($\Downarrow$)

$\mathbf{T} = \mathbf{X} = \mathrm{range}(\Downarrow)$ $\qquad$ $\textsc{match}(M, \widetilde{x}, X) = \{\widetilde{L} : M = X\{\widetilde{L}/\widetilde{x}\}\}$

$M[\widetilde{y} := \widetilde{L}] = M\{\widetilde{L}/\widetilde{y}\}\Downarrow$ $\qquad$ $\textsc{vars}(X) = \bigcup\{\mathcal{P}(\mathrm{n}(M) \setminus \mathrm{n}(\rho)) : M \text{ stable} \wedge X = M\rho\}$

We need to show that the patterns are closed under substitution, including preservation of $\textsc{vars}$ (cf. Definition 2.4), and that matching satisfies the criteria of Definition 2.5. Since any term is a pattern, the patterns are closed under substitution. Since term substitution $\{\cdot/\cdot\}$ and normalization $\Downarrow$ are both sort-preserving, term and pattern substitution $[\cdot := \cdot]$ is also sort-preserving.

To show preservation of pattern variables, assume that $\widetilde{x} \in \textsc{vars}(X)$ is a tuple of distinct names. By definition there are M and ρ such that $X = M\rho$ with M stable and $\widetilde{x} \subseteq \mathrm{n}(M) \setminus \mathrm{n}(\rho)$. Assume that $\widetilde{x}\#\sigma$; then $X\sigma = (M\rho)\sigma = M(\sigma \circ \rho)$ with $\widetilde{x}\#\sigma \circ \rho$, so $\widetilde{x} \in \textsc{vars}(X\sigma)$.

For the criteria of Definition 2.5, additionally assume that $\widetilde{L} \in \textsc{match}(N, \widetilde{x}, X)$ and let $\sigma = [\widetilde{x} := \widetilde{L}]$. Since $\{\widetilde{L}/\widetilde{x}\}$ is well-sorted, so is $[\widetilde{x} := \widetilde{L}]$. We also immediately have $\mathrm{n}(\widetilde{L}) = \mathrm{n}(N) \cup (\mathrm{n}(X) \setminus \widetilde{x})$, and alpha-renaming of matching follows from the same property for term substitution.

Example 5.1 (Peano arithmetic)**.** As a simple instance of **REWRITE**($\Downarrow$), we may consider Peano arithmetic. The rewrite rules for addition (below) induce a convergent rewrite system $\Downarrow^{\mathrm{Peano}}$, where the stable terms are those that do not contain any occurrence of `plus`.

PEANO

Everything as in **REWRITE($\Downarrow$)** except:
$\mathcal{S} = \{\mathtt{nat}, \mathtt{chan}\}$
$\Sigma = \{\mathtt{zero} : \mathtt{nat}, \qquad \mathtt{succ} : \mathtt{nat} \to \mathtt{nat} \qquad \mathtt{plus} : \mathtt{nat} \times \mathtt{nat} \to \mathtt{nat}\}$
$\mathtt{plus}(K, \mathtt{zero}) \to K \qquad \mathtt{plus}(K, \mathtt{succ}(M)) \to \mathtt{plus}(\mathtt{succ}(K), M)$
$\textsc{vars}(\mathtt{succ}^n(a)) = \{\emptyset, \{a\}\} \qquad \textsc{vars}(M) = \{\emptyset\}$ otherwise

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

5.2. **Symmetric cryptography.** We can also consider variants of **REWRITE**($\Downarrow$), such as a simple Dolev-Yao style [DY83] 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.

The rewrite rule describing decryption $\mathtt{dec}(\mathtt{enc}(M,K),K) \to M$ induces a convergent rewrite system $\Downarrow^{\mathrm{enc}}$, where the terms not containing $\mathtt{dec}$ are stable. The construction of **REWRITE**($\Downarrow$) yields that $\widetilde{x} \in \textsc{vars}(X)$ if $\widetilde{x} \subseteq \mathrm{n}(X)$ are pair-wise different and no x_i occurs as a subterm of a $\mathtt{dec}$ in X. This construction would still permit to bind the keys of an encrypted message upon reception, e.g. $\underline{a}(\lambda m,k)\mathtt{enc}(m,k)\,.\,P$ would be allowed although it does not make cryptographic sense. Therefore we further restrict $\textsc{vars}(X)$ to those sets not containing names that occur in key position in X, thus disallowing the binding of k above. Below we give the formal definition (recall that $\preceq$ is the subterm preorder).

SYMSPI

Everything as in **REWRITE($\Downarrow^{\mathrm{enc}}$)** except:
$\mathcal{S} = \{\mathtt{message}, \mathtt{key}\}$
$\Sigma = \{\mathtt{enc} : \mathtt{message} \times \mathtt{key} \to \mathtt{message}, \quad \mathtt{dec} : \mathtt{message} \times \mathtt{key} \to \mathtt{message}\}$
$\mathtt{dec}(\mathtt{enc}(M,K),K) \to M$
$\textsc{vars}(X) = \mathcal{P}(n(X) \setminus \{a : a \preceq \mathtt{dec}(Y_1,Y_2) \preceq X \vee (a \preceq Y_2 \wedge \mathtt{enc}(Y_1,Y_2) \preceq X)\})$

The proof of the conditions of Definition 2.4 and Definition 2.5 for patterns is the same as for **REWRITE**($\cdot$) in Section 5.1 above.

As an example, the agent

$$(\nu a,k)(\overline{a}\ \mathtt{enc}(\mathtt{enc}(M,l),k) \mid \underline{a}(\lambda y)\mathtt{enc}(y,k)\,.\,\overline{c}\ \mathtt{dec}(y,l))$$

has a visible transition with label $\overline{c}\ M$: the subagent

$$\underline{a}(\lambda y)\mathtt{enc}(y,k)\,.\,\overline{c}\ \mathtt{dec}(y,l) \xrightarrow{\underline{a}\ \mathtt{enc}(\mathtt{enc}(M,l),k)} \overline{c}\ \mathtt{dec}(y,l)[y := \mathtt{enc}(M,l)]$$

since $\mathtt{enc}(M,l) \in \textsc{match}(\mathtt{enc}(\mathtt{enc}(M,l),k), y, \mathtt{enc}(y,k))$. The resulting process is

$$\overline{c}\ \mathtt{dec}(y,l)[y := \mathtt{enc}(M,l)] = \overline{c}\ \mathtt{dec}(y,l)\{{}^{\mathtt{enc}(M,l)}/_{y}\}\Downarrow\, = \overline{c}\ \mathtt{dec}(\mathtt{enc}(M,l),l)\Downarrow\, = \overline{c}\ M.$$

5.3. **Asymmetric cryptography.** A more advanced version of Section 5.2 is the treatment of data in the pattern-matching spi-calculus [HJ06], to which we refer for more examples and motivations of the definitions below. The calculus uses asymmetric encryption, and 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 specialized modelling features can be presented in a form that is very close to the original.

We start from the term algebra T_Σ over the unsorted signature

$$\Sigma = \{(),\ (\cdot,\cdot),\ \mathtt{eKey}(\cdot),\ \mathtt{dKey}(\cdot),\ \mathtt{enc}(\cdot,\cdot)\ \mathtt{enc}^{-1}(\cdot,\cdot)\}$$

The $\mathtt{eKey}(M)$ and $\mathtt{dKey}(M)$ constructions represent the encryption and decryption parts of the key pair M, respectively. The operation $\mathtt{enc}^{-1}(M,N)$ is encryption of M with the inverse of the decryption key N, which is not an implementable operation but only permitted to occur in patterns. We add a sort system on T_Σ with sorts $\mathcal{S} = \{\mathtt{impl}, \mathtt{pat}, \bot\}$, where $\mathtt{impl}$ denotes implementable terms not containing $\mathtt{enc}^{-1}$, and $\mathtt{pat}$ those that may only be used in patterns. The sort $\bot$ denotes ill-formed terms, which do not occur in well-formed processes. Names stand for implementable terms, so we let $\mathcal{S}_{\mathcal{N}} = \{\mathtt{impl}\}$. Substitution is defined homomorphically on the term algebra, except to avoid unimplementable subterms on the form $\mathtt{enc}^{-1}(M, \mathtt{dKey}(N))$.

In order to define $\textsc{vars}(X)$, we write $\widetilde{M} \Vdash \widetilde{N}$ if all $N_i \in \widetilde{N}$ can be deduced from $\widetilde{M}$ in the Dolev-Yao message algebra (i.e., using cryptographic operations such as encryption and decryption). For the precise definition, see [HJ06]. The definition of $\textsc{vars}(X)$ below allows to bind a set S of names only if all names in S can be deduced from the message term X using the other names occurring in X. This excludes binding an unknown key, like in Example **??**.

PMSPI

$\mathbf{T} = \mathbf{X} = T_\Sigma \qquad \mathcal{S} = \{\mathtt{impl}, \mathtt{pat}, \bot\} \qquad \mathcal{S}_\mathcal{N} = \{\mathtt{impl}\}$

$\prec\, = \overline{\propto} = \{(\mathtt{impl}, \mathtt{impl})\} \qquad \underline{\propto} = \{(\mathtt{impl}, \mathtt{impl}), (\mathtt{impl}, \mathtt{pat})\}$

$\textsc{sort}(M) = \mathtt{impl}$ 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

$\textsc{match}(M, \widetilde{x}, X) = \{\widetilde{L} \ :\ M = X[\widetilde{x} := \widetilde{L}]\}$

$\textsc{vars}(X) = \{S \subseteq \mathrm{n}(X) \ :\ ((\mathrm{n}(X) \setminus S) \cup \{X\}) \Vdash S\}$

$$\begin{array}{rcll}
x[\widetilde{y} := \widetilde{L}] &=& L_i & \text{if } y_i = x \\
x[\widetilde{y} := \widetilde{L}] &=& x & \text{otherwise.} \\
\mathtt{enc}^{-1}(M_1, M_2)[\widetilde{y} := \widetilde{L}] &=& \mathtt{enc}(M_1[\widetilde{y} := \widetilde{L}], \mathtt{eKey}(N)) & \text{when } M_2[\widetilde{y} := \widetilde{L}] = \mathtt{dKey}(N) \\
f(M_1, \ldots, M_n)[\widetilde{y} := \widetilde{L}] &=& f(M_1[\widetilde{y} := \widetilde{L}], \ldots, M_n[\widetilde{y} := \widetilde{L}]) & \text{otherwise.}
\end{array}$$

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

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

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 2.4 (Substitution), since $\mathtt{impl} \leq \mathtt{pat}$ (implementable terms can be used as channels or messages whenever patterns can be).

Terms (and patterns) are trivially closed under substitution. All terms in the domain of a well-sorted substitution have sort $\mathtt{impl}$, so well-sorted substitutions cannot introduce subterms of the forms $\mathtt{enc}^{-1}(N_1, N_2)$ or $\mathtt{enc}^{-1}(N_1, \mathtt{dKey}(N_2))$ where none existed; thus $\textsc{sort}(M\sigma) \leq \textsc{sort}(M)$ as required by Definition 2.4.

To show preservation of pattern variables, we have that $((\mathrm{n}(X) \setminus \widetilde{x}) \cup \{X\}) \Vdash \widetilde{x}$ implies that $((\mathrm{n}(X\sigma) \setminus \widetilde{x}) \cup \{X\sigma\}) \Vdash \widetilde{x}$ whenever $x \# \sigma$, by induction on $\Vdash$. Add definition, of $\Vdash$, give IH? The requisites on matching (Definition 2.5) follow from those on substitution.

5.4. **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 extended with the erratic choice operator $\cdot \parallel \cdot$ and the reduction rule $M_1 \parallel M_2 \to M_i$ if $i \in \{1, 2\}$. Due to non-determinism and partiality, evaluation cannot be part of the substitution function. Instead, we define the $\textsc{match}$ function to collect 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 $\mathcal{R} ::= [\,] \mid \mathcal{R}\ M \mid (\boldsymbol{\lambda} x.M)\ \mathcal{R}$ (we here use the boldface $\boldsymbol{\lambda}$ rather than the λ used in input prefixes). Reduction $\rightarrow$ is the smallest pre-congruence for reduction contexts that contain the rules for β-reduction ($\boldsymbol{\lambda} x.M\ N \rightarrow M[x := N]$) and $\cdot [\!] \cdot$ (see above). We use the single-name patterns of Example 2.6, but include evaluation in matching.

NDLAM

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

$M ::= a \mid M\ M \mid \boldsymbol{\lambda} x.M \mid M\ [\!]\ M$ $\quad\quad$ where x binds into M in $\boldsymbol{\lambda} x.M$

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

As an example, the agent $P \overset{def}{=} (\nu a)(\underline{a}(y) \,.\, \overline{c}\, y \,.\, \mathbf{0} \mid \overline{a}\ ((\boldsymbol{\lambda} x.x\ x)\ [\!]\ (\boldsymbol{\lambda} x.x)) \,.\, \mathbf{0})$ has the following transitions:

$$P \xrightarrow{\tau} (\nu a)(\overline{c}\ \boldsymbol{\lambda} x.xx \,.\, \mathbf{0} \mid \mathbf{0}) \xrightarrow{\overline{c}\ \boldsymbol{\lambda} x.xx} \mathbf{0}$$
$$P \xrightarrow{\tau} (\nu a)(\overline{c}\ \boldsymbol{\lambda} x.x \,.\, \mathbf{0} \mid \mathbf{0}) \xrightarrow{\overline{c}\ \boldsymbol{\lambda} x.x} \mathbf{0}.$$

6. 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. Using these features we have provided representations of other process calculi, ranging from the simple polyadic pi-calculus to the spi-calculus and non-deterministic computations, in the psi-calculi framework. The critera for representation (rather than encoding) are stronger than standard correspondences e.g. by Gorla, and mean that the psi-calculus and the calculus represented by it are for all practical purposes one and the same.

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 [BGRV13], 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 [PBRÅP13] and reliable broadcast psi-calculi [ÅPBP+13]. As mentioned in Section 3.1, further developments in Nominal Isabelle are needed for mechanizing theories with arbitrary but fixed sortings.

References

[AF01] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proceedings of POPL '01*, pages 104–115. ACM, January 2001.

[ÅP14] Johannes Åman Pohjola. Isabelle proof scripts for sorted psi-calculi. Available at `http://www.it.uu.se/research/group/mobility/theorem/sortedPsi.tar.gz`, 2014.

[ÅPBP^{+}13] Johannes Åman Pohjola, Johannes Borgström, Joachim Parrow, Palle Raabjerg, and Ioana Rodhe. Negative premises in applied process calculi. Technical Report 2013-014, Department of Information Tecnology, Uppsala University, 2013.

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

[BGRV13] Johannes Borgström, Ramūnas Gutkovas, Ioana Rodhe, and Björn Victor. A parametric tool for applied process calculi. In *Proc. 13th International Conference on Application of Concurrency to System Design (ACSD'13).* IEEE, 2013.

[BJPV11] Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Psi-calculi: a framework for mobile processes with nominal data and logic. *LMCS*, 7(1:11), 2011.

[Bla11] Bruno Blanchet. Using Horn clauses for analyzing security protocols. In Véronique Cortier and Steve Kremer, editors, *Formal Models and Techniques for Analyzing Security Protocols*, volume 5 of *Cryptology and Information Security Series*, pages 86–111. IOS Press, March 2011.

[CGK^{+}13] Sjoerd Cranen, Jan Friso Groote, Jeroen J. A. Keiren, Frank P. M. Stappers, Erik P. de Vink, Wieger Wesselink, and Tim A. C. Willemse. An overview of the mCRL2 toolset and its recent advances. In Nir Piterman and Scott A. Smolka, editors, *TACAS*, volume 7795 of *Lecture Notes in Computer Science*, pages 199–213. Springer, 2013.

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

[DY83] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[EOW07] Burak Emir, Martin Odersky, and John Williams. Matching objects with patterns. In *Proceedings of the 21st European Conference on Object-Oriented Programming*, ECOOP'07, pages 273–298, Berlin, Heidelberg, 2007. Springer-Verlag.

[FG96] Cédric Fournet and Georges Gonthier. The reflexive CHAM and the join-calculus. In *Proc. POPL*, pages 372–385, 1996.

[FGM05] Cédric Fournet, Andrew D. Gordon, and Sergio Maffeis. A type discipline for authorization policies. In Mooly Sagiv, editor, *Proc. of ESOP 2005*, volume 3444 of *LNCS*, pages 141–156. Springer, 2005.

[Gel85] David Gelernter. Generative communication in Linda. *ACM TOPLAS*, 7(1):80–112, January 1985.

[Gor10] Daniele Gorla. Towards a unified approach to encodability and separation results for process calculi. *Information and Computation*, 208(9):1031–1053, 2010.

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

[GWGJ10] Thomas Given-Wilson, Daniele Gorla, and Barry Jay. Concurrent pattern calculus. In Cristian Calude and Vladimiro Sassone, editors, *Theoretical Computer Science*, volume 323 of *IFIP Advances in Information and Communication Technology*, pages 244–258. Springer, 2010.

[HJ06] Christian Haack and Alan Jeffrey. Pattern-matching spi-calculus. *Information and Computation*, 204(8):1195–1263, 2006.

[Hon93] Kohei Honda. Types for dyadic interaction. In Eike Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993.

[Hüt11] Hans Hüttel. Typed psi-calculi. In Joost-Pieter Katoen and Barbara König, editors, *CONCUR 2011 – Concurrency Theory*, volume 6901 of *LNCS*, pages 265–279. Springer, 2011.

[HV13] Hans Hüttel and Vasco T Vasconcelos. The foundations of behavioural types. State-of-the art report of WG1 of the BETTY project (EU COST Action IC1201). To appear, 2013.

[JBPV10] Magnus Johansson, Jesper Bengtson, Joachim Parrow, and Björn Victor. Weak equivalences in psi-calculi. In *Proc. of LICS 2010*, pages 322–331. IEEE, 2010.

[JVP12] Magnus Johansson, Björn Victor, and Joachim Parrow. Computing strong and weak bisimulations for psi-calculi. *Journal of Logic and Algebraic Programming*, 81(3):162–180, 2012.

[Kri09] Neelakantan R. Krishnaswami. Focusing on pattern matching. In *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '09, pages 366–378, New York, NY, USA, 2009. ACM.

[LSD11] Yang Liu, Jun Sun, and Jin Song Dong. PAT 3: An extensible architecture for building multi-domain model checkers. In Tadashi Dohi and Bojan Cukic, editors, *ISSRE '11*, pages 190–199. IEEE, 2011.

[Mil89] Robin Milner. *Communication and Concurrency.* Prentice-Hall, Inc., 1989.

[Mil93] Robin Milner. The polyadic π-calculus: A tutorial. In Friedrich L. Bauer, Wilfried Brauer, and Helmut Schwichtenberg, editors, *Logic and Algebra of Specification*, volume 94 of *Series F.* NATO ASI, Springer, 1993.

[PBRÅP13] Joachim Parrow, Johannes Borgström, Palle Raabjerg, and Johannes Åman Pohjola. Higher-order psi-calculi. *Mathematical Structures in Computer Science*, FirstView, June 2013.

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

[San93] Davide Sangiorgi. *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).

[SLDC09] Jun Sun, Yang Liu, Jin Song Dong, and Chunqing Chen. Integrating specification and programs for system modeling and verification. In *TASE '09*, pages 127–135. IEEE Computer Society, 2009.

[SNM07] Don Syme, Gregory Neverov, and James Margetson. Extensible pattern matching via a lightweight language extension. In *Proceedings of the 12th ACM SIGPLAN International Conference on Functional Programming*, ICFP '07, pages 29–40, New York, NY, USA, 2007. ACM.

[SS05] Alan Schmitt and Jean-Bernard Stefani. The Kell calculus: A family of higher-order distributed process calculi. In Corrado Priami and Paola Quaglia, editors, *Global Computing*, volume 3267 of *LNCS*, pages 146–178. Springer Berlin Heidelberg, 2005.

[SW01] Davide Sangiorgi and David Walker. *The π-calculus: a Theory of Mobile Processes.* Cambridge University Press, 2001.

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

Appendix A. Full proofs for Section 4

The following is full proofs of Section 4; 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 4, as well as the definitions, notation and terminology of Sangiorgi [San93], Carbone and Maffeis [CM03], and Milner [Mil89], 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 $\mathrm{bn}(\alpha)\#Q$ instead of $\mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset$.

A.1. Polyadic Pi-Calculus.

We follow the exposition of Polyadic Pi-Calculus given by Sangiorgi in [San93] 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 4.1.

Definition A.1 (Polyadic Pi-Calculus to **PPi**).
Agents:

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

Actions:

$$\begin{array}{rcl}
\llbracket (\nu\widetilde{y'})\overline{z}\langle\tilde{y}\rangle \rrbracket & = & \overline{z}\,(\nu\widetilde{y'})\,\langle\widetilde{y}\rangle \\
\llbracket x\langle\tilde{z}\rangle \rrbracket & = & \underline{x}\,\langle\tilde{z}\rangle \\
\llbracket \tau \rrbracket & = & \tau
\end{array}$$

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

Definition A.2 (**PPi** to Polyadic Pi-Calculus).
Process:

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

Case clause:

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

We prove that substitution function distributes over the encoding function. We use this auxiliary result in some of the following theorems.

Lemma A.3. $\llbracket P \rrbracket[\tilde{y} := \tilde{z}] = \llbracket P\{\tilde{z}/\tilde{y}\} \rrbracket$

Proof. By induction on P. We consider only the agents where $bn(P) \cap fn(P)\{\tilde{z}/\tilde{y}\} = \emptyset$ as in Definition 2.1.1 in [San93] on page 21. We show the interesting cases of the substitution application as others are just homomorphic.

- case $P = P' + Q$.

$$\begin{array}{rcll}
\llbracket P' + Q \rrbracket[\tilde{y} := \tilde{z}] & = & \mathbf{case}\ \top[\tilde{y} := \tilde{z}] : \llbracket P' \rrbracket[\tilde{y} := \tilde{z}] \ [] \ \top[\tilde{y} := \tilde{z}] : \llbracket Q \rrbracket[\tilde{y} := \tilde{z}] & \\
& = & \mathbf{case}\ \top : \llbracket P' \rrbracket[\tilde{y} := \tilde{z}] \ [] \ \top : \llbracket Q \rrbracket[\tilde{y} := \tilde{z}] & \\
& = & \mathbf{case}\ \top : \llbracket P'\{\tilde{z}/\tilde{y}\} \rrbracket \ [] \ \top : \llbracket Q\{\tilde{z}/\tilde{y}\} \rrbracket & \text{(IH)} \\
& = & \llbracket P'\{\tilde{z}/\tilde{y}\} + Q\{\tilde{z}/\tilde{y}\} \rrbracket & \\
& = & \llbracket (P' + Q)\{\tilde{z}/\tilde{y}\} \rrbracket &
\end{array}$$

- case $P = [x = y]Q$.

$$\begin{array}{rcll}
\llbracket [x = y]Q \rrbracket[\tilde{y} := \tilde{z}] & = & \mathbf{case}\ x[\tilde{y} := \tilde{z}] = y[\tilde{y} := \tilde{z}] : \llbracket Q \rrbracket[\tilde{y} := \tilde{z}] & \\
& = & \mathbf{case}\ x[\tilde{y} := \tilde{z}] = y[\tilde{y} := \tilde{z}] : \llbracket Q\{\tilde{z}/\tilde{y}\} \rrbracket & \text{(IH)} \\
& = & [x\{\tilde{z}/\tilde{y}\} = y\{\tilde{z}/\tilde{y}\}]\llbracket Q\{\tilde{z}/\tilde{y}\} \rrbracket & \\
& = & \llbracket ([x = y]Q)\{\tilde{z}/\tilde{y}\} \rrbracket &
\end{array}$$

- case $P = a(\tilde{x}).Q$

$$\begin{array}{rcll}
\llbracket a(\tilde{x}).Q \rrbracket[\tilde{y} := \tilde{z}] & = & \underline{a[\tilde{y} := \tilde{z}]}(\lambda\widetilde{x})\langle\widetilde{x}\rangle.\llbracket Q \rrbracket[\tilde{y} := \tilde{z}] & \text{(From assumption } \tilde{x}\#[\tilde{y} := \tilde{z}]) \\
& = & \underline{a[\tilde{y} := \tilde{z}]}(\lambda\widetilde{x})\langle\widetilde{x}\rangle.\llbracket Q\{\tilde{z}/\tilde{y}\} \rrbracket & \text{(IH)} \\
& = & a\{\tilde{z}/\tilde{y}\}(\tilde{x}).\llbracket Q\{\tilde{z}/\tilde{y}\} \rrbracket & \\
& = & \llbracket (a(\tilde{x}).Q)\{\tilde{z}/\tilde{y}\} \rrbracket &
\end{array}$$

□

The following is proof of the strong operational correspondence. The labeled semantics of polyadic pi-calculus can be found on page 30 of [San93].

Proof of Theorem 4.4.

(1) By induction on the length of the derivation of P'. We have the following cases to check by considering the last rule applied to derive P'.

ALP:
Trivial since in Psi-calculi agents are identified up to alpha equivalence.

OUT:
Assume $\overline{x}\langle\tilde{y}\rangle.P \xrightarrow{\overline{x}\langle\tilde{y}\rangle} P$ and $\alpha \in \{\overline{x}\ \langle\tilde{y}\rangle\} = \llbracket \overline{x}\langle\tilde{y}\rangle \rrbracket$. Since $\mathbf{1} \vdash x \mathbin{\dot{\leftrightarrow}} x$ and $\llbracket \overline{x}\ \langle\tilde{y}\rangle.P \rrbracket = \overline{x}\ \langle\tilde{y}\rangle.\llbracket P \rrbracket$ and $\alpha = \overline{x}\ \langle\tilde{y}\rangle$, we can derive $\overline{x}\ \langle\tilde{y}\rangle.\llbracket P \rrbracket \xrightarrow{\overline{x}\ \langle\tilde{y}\rangle} \llbracket P \rrbracket$.

INP:
Assume $x(\tilde{y}).P \xrightarrow{x\langle\tilde{z}\rangle} P\{\tilde{z}/\tilde{y}\}$ with $\tilde{z} : \tilde{y}$ and $\alpha \in \llbracket \beta \rrbracket = \{\underline{x}\ \langle\tilde{z}\rangle\}$. We compute that $\llbracket x(\tilde{y}).P \rrbracket = \underline{x}(\lambda\widetilde{y})\langle\widetilde{y}\rangle.\llbracket P \rrbracket$ and $\widetilde{z} \in \text{MATCH}(\langle\widetilde{z}\rangle, \widetilde{y}, \langle\widetilde{y}\rangle)$. Using this and $\mathbf{1} \vdash x \mathbin{\dot{\leftrightarrow}} x$ we can derive $\underline{x}(\lambda\widetilde{y})\langle\widetilde{y}\rangle.\llbracket P \rrbracket \xrightarrow{\underline{x}\ \langle\tilde{z}\rangle} \llbracket P \rrbracket[\widetilde{y} := \widetilde{z}]$ with the In rule. By applying Lemma A.3 completes the proof.

SUM:
Assume $P + Q \xrightarrow{\beta} P'$ and $\alpha \in \llbracket \beta \rrbracket$, and also $P \xrightarrow{\beta} P'$. From induction hypothesis we have that for every $\alpha \in \llbracket \beta \rrbracket$, $\llbracket P \rrbracket \xrightarrow{\alpha} \llbracket P' \rrbracket$. Thus we can derive $\mathbf{case}\ \top : \llbracket P \rrbracket \ [] \ \top : \llbracket Q \rrbracket \xrightarrow{\alpha} \llbracket P' \rrbracket$ with the Case rule for every $\alpha \in \llbracket \beta \rrbracket$.

PAR:

Assume $P \mid Q \xrightarrow{\beta} P' \mid Q$ and $\alpha \in \llbracket\beta\rrbracket$. We also assume $P \xrightarrow{\beta} P'$ with $bn(\beta) \cap fn(Q) = \emptyset$. From induction hypothesis, we get that for every $\alpha \in \llbracket\beta\rrbracket$, $\llbracket P\rrbracket \xrightarrow{\alpha} \llbracket P'\rrbracket$. From assumption follows that $bn(\alpha)\#\llbracket Q\rrbracket$ for any $\alpha \in \llbracket\beta\rrbracket$. By applying the Par rule, we obtain the required transition $\llbracket P\rrbracket \mid \llbracket Q\rrbracket \xrightarrow{\alpha} \llbracket P'\rrbracket \mid \llbracket Q\rrbracket$.

COM:

Assume $P \mid Q \xrightarrow{\tau} \nu\tilde{y}'(P' \mid Q')$ with $\tilde{y}' \cap fn(Q) = \emptyset$. We also assume $P \xrightarrow{(\nu\tilde{y'})\overline{x}\langle\tilde{y}\rangle} P'$ and $Q \xrightarrow{x\langle\tilde{y}\rangle} Q'$. From induction hypothesis, we have that for every $\alpha' \in \llbracket(\nu\tilde{y'})\overline{x}\langle\tilde{y}\rangle\rrbracket$ and $\alpha'' \in \llbracket x\langle\tilde{y}\rangle\rrbracket$, $\llbracket P\rrbracket \xrightarrow{\alpha'} \llbracket P'\rrbracket$ and $\llbracket Q\rrbracket \xrightarrow{\alpha''} \llbracket Q'\rrbracket$ Moreover, we note that $\mathbf{1} \vdash x \dot{\leftrightarrow} x$ and $\tilde{y}'\#\llbracket Q\rrbracket$. Finally, we choose α' and α'' and choose alpha-variants of the frames of $\llbracket P\rrbracket$ and $\llbracket Q\rrbracket$ which are sufficiently fresh to allow the derivation $\llbracket P\rrbracket \mid \llbracket Q\rrbracket \xrightarrow{\tau} (\nu\widetilde{y'})(\llbracket P'\rrbracket \mid \llbracket Q'\rrbracket)$ with the Com rule.

MATCH:

Assume $[x = x]P \xrightarrow{\beta} P'$ and $\alpha \in \llbracket\beta\rrbracket$. We also assume $P \xrightarrow{\beta} P'$. From induction hypothesis we acquire that $\llbracket P\rrbracket \xrightarrow{\alpha} \llbracket P'\rrbracket$. Since $\mathbf{1} \vdash x = x$ and $\mathbf{case}\ x = x : \llbracket P\rrbracket = \llbracket[x = x]P\rrbracket$, we derive $\mathbf{case}\ x = x : \llbracket P\rrbracket \xrightarrow{\alpha} \llbracket P'\rrbracket$ with the Case rule.

REP:

Assume $!P \xrightarrow{\beta} P'$ and $\alpha \in \llbracket\beta\rrbracket$. Moreover, assume $P \mid !P \xrightarrow{\beta} P'$ and hence from induction hypothesis $\llbracket P \mid !P\rrbracket \xrightarrow{\alpha} \llbracket P'\rrbracket$. We compute $\llbracket P\rrbracket \mid !\llbracket P\rrbracket = \llbracket P \mid !P\rrbracket$ and apply the Rep rule to obtain $!\llbracket P\rrbracket \xrightarrow{\alpha} \llbracket P'\rrbracket$.

RES:

Assume $\nu xP \xrightarrow{\beta} \nu xP'$ where $x \notin n(\beta)$ and $\alpha \in \llbracket\beta\rrbracket$. We also assume $P \xrightarrow{\beta} P'$, to acquire from induction hypothesis $\llbracket P\rrbracket \xrightarrow{\alpha} \llbracket P'\rrbracket$. Now by obtaining $x\#\alpha$ from assumption and computing $\llbracket\nu xP\rrbracket = (\nu x)\llbracket P\rrbracket$, we derive $(\nu x)\llbracket P\rrbracket \xrightarrow{\alpha} (\nu x)\llbracket P'\rrbracket$ with the Scope rule.

OPEN:

Let $\beta = (\nu x, \tilde{y'})\overline{z}\langle\tilde{y}\rangle$. Assume $\nu xP \xrightarrow{\beta} P'$ and $x \neq z, x \in \tilde{y} - \tilde{y'}$ and $\alpha \in \llbracket\beta\rrbracket = \{\overline{z}\,(\nu\widetilde{y''})\,\langle\tilde{y}\rangle \ :\ \tilde{y}'' = \pi \cdot x, \tilde{y}'\}$. From induction hypothesis, we get that for every $\alpha' \in \llbracket(\nu\tilde{y}')\overline{z}\langle\tilde{y}\rangle\rrbracket = \{\overline{z}\,(\nu\widetilde{y''})\,\langle\tilde{y}\rangle \ :\ \tilde{y}'' = \pi \cdot \tilde{y'}\}$ we can derive $\llbracket P\rrbracket \xrightarrow{\alpha'} \llbracket P'\rrbracket$. We choose $\alpha' = \overline{z}\,(\nu\widetilde{y'})\,\widetilde{y}$ and by having $\llbracket\nu xP\rrbracket = (\nu x)\llbracket P\rrbracket$ derive, $(\nu x)\llbracket P\rrbracket \xrightarrow{\overline{z}\,(\nu x,\tilde{y'})\,\langle\tilde{y}\rangle} \llbracket P'\rrbracket$ with the Open rule. The side conditions of Open, $x\#\tilde{y}', z$ and $x \in n(\tilde{y})$, follow from assumptions.

From the assumption $\alpha \in \llbracket\beta\rrbracket$, it follows that, for any permutation π, α is of the form $\overline{z}\,(\nu\pi \cdot x, \widetilde{y'})\,\langle\tilde{y}\rangle$. By applying Lemma 4.3, we get the required α and transition $(\nu x)\llbracket P\rrbracket \xrightarrow{\alpha} \llbracket P'\rrbracket$. And this concludes this proof case.

(2) We now show that if $[\![P]\!] \xrightarrow{\alpha} P''$ then $P \xrightarrow{\beta} P'$ where $\alpha \in [\![\beta]\!]$ and $[\![P']\!] = P''$. We proceed by by induction on the length of the derivation of P''. We only write down the interesting cases:

Case:

Assume $[\![P]\!] \xrightarrow{\alpha} P''$. Because P'' is derived with the Case rule, $[\![P]\!]$ is of the form **case** $\tilde{\varphi} : \tilde{P}$. 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 $P + Q \xrightarrow{\beta} P'$, using the **SUM** rule and the fact obtained from induction hypothesis $\alpha \in [\![\beta]\!]$.

(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 $\mathbf{1} \vdash x = y$ 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 $[x = x]P \xrightarrow{\beta} P'$, using the **MATCH** rule and the fact obtained from induction hypothesis $\alpha \in [\![\beta]\!]$.

Open:

Assume $[\![P]\!] \xrightarrow{\overline{z}\,(\nu\tilde{y} \cup \{x\})\,\langle\tilde{y}'\rangle} P''$. Because P'' is derived with the Open rule, $[\![P]\!]$ is of the form $(\nu x)R$. Since $(\nu x)R$ is in the range of $[\![\cdot]\!]$, $P = \nu x R'$ where $R = [\![R']\!]$. From induction hypothesis, we have that $R \xrightarrow{\overline{z}\,(\nu\tilde{y})\,\langle\tilde{y}'\rangle} P''$ and $\overline{z}\,(\nu\tilde{y})\,\langle\tilde{y}'\rangle \in [\![\beta']\!]$ and $R' \xrightarrow{\beta'} P'$ and lastly $[\![P']\!] = P''$. Thus we use $\beta' = (\nu\tilde{y})\overline{z}\langle\tilde{y}'\rangle$ as it gives us $\overline{z}\,(\nu\tilde{y})\,\langle\tilde{y}'\rangle \in [\![\beta']\!]$ to derive using the rule **OPEN**, $\nu x R' \xrightarrow{(\nu x, \tilde{y})\overline{z}\langle\tilde{y}'\rangle} P'$. Clearly $\overline{z}\,(\nu\tilde{y} \cup \{x\})\,\langle\tilde{y}'\rangle \in [\![(\nu x, \tilde{y})\overline{z}\langle\tilde{y}'\rangle]\!]$ for every insertion of x.

From the strong operational correspondence, we obtain full abstraction. We use Sangiorgi's the definition of bisimulation and congruence of polyadic pi-calculus which can be found in [San93] on page 42.

Theorem A.4. *For polyadic-pi calculus agents P and Q we have $P \sim_e^c Q$ iff $[\![P]\!] \sim [\![Q]\!]$*

Proof. Direction $\Leftarrow$. Assume $[\![P]\!] \sim [\![Q]\!]$. We claim that the relation $\mathcal{R} = \{(P, Q) : [\![P]\!] \sim [\![Q]\!]\}$ is an *early congruence* in the polyadic pi-calculus.

For simulation, assume $P \xrightarrow{\beta} P'$. We need to show that for some Q', s.t. $Q \xrightarrow{\beta} Q'$ and $(P', Q') \in \mathcal{R}$. By Theorem 4.4 (1), we get $[\![P]\!] \xrightarrow{\alpha} [\![P']\!]$ for any $\alpha \in [\![\beta]\!]$. By Theorem 4.4 (2) and using the assumption $\alpha \in [\![\beta]\!]$ as well as the fact $[\![P]\!] \sim [\![Q]\!]$, we derive $[\![Q]\!] \xrightarrow{\alpha} [\![Q']\!]$. From simulation clause and that $[\![P]\!]$ and $[\![Q]\!]$ are congruent follows that $[\![P']\!] \sim [\![Q']\!]$ and hence $(P', Q') \in \mathcal{R}$. Symmetry case follows from the symmetry of $\sim$. Hence $\mathcal{R}$ is an early bisimulation. Since $\mathcal{R}$ is closed under all substitutions by Lemma A.3, it is an early congruence.

We prove direction $\Rightarrow$, assume $P \sim_e^c Q$. We claim the the relation $\{(\mathbf{1}, [\![P]\!], [\![Q]\!]) : P \sim_e^c Q\}$ is a congruence in **PPI**. The static equivalence and extension of arbitrary assertion cases are trivial since there is unit assertion only. Symmetry follows from symmetry of $\sim_e^c$, and simulation follows by Theorem 4.4 and the fact that $\sim_e^c$ is an early congruence.

□

Proof of Theorem 4.7. By structural induction on P. We only consider the case of **case** agent as other cases are trivial.

case $\textbf{case } \varphi_1 : P_1 \mathrel{[]} \dots \mathrel{[]} \varphi_n : P_n$**:**

We get an induction hypothesis for every $i \in \{1..n\}$, IH_i: $P_i \sim \llbracket \overline{P_i} \rrbracket$.

We proceed by induction on n.

base case $n = 0$**:**

$\llbracket \overline{\textbf{case}} \rrbracket = \llbracket \mathbf{0} \rrbracket = \mathbf{0}$. By reflexivity of $\sim$, $\mathbf{0} \sim \mathbf{0}$.

induction step $n + 1$**:**

The IH for this case is

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

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

We compute

$$\begin{aligned}
\llbracket \overline{Q} \rrbracket &= \llbracket \overline{\varphi_1 : P_1} + \dots + \overline{\varphi_n : P_n} + \overline{\varphi_{n+1} : P_{n+1}} \rrbracket \\
&= \textbf{case } \top : \llbracket \overline{\varphi_1 : P_1} \rrbracket \mathrel{[]} \dots \mathrel{[]} \top : \llbracket \overline{\varphi_n : P_n} \rrbracket \mathrel{[]} \top : \llbracket \overline{\varphi_{n+1} : P_{n+1}} \rrbracket \\
&\sim \text{(by Lemma 3.3)} \\
&\quad \textbf{case } \top : (\textbf{case } \top : \llbracket \overline{\varphi_1 : P_1} \rrbracket \mathrel{[]} \dots \mathrel{[]} \top : \llbracket \overline{\varphi_n : P_n} \rrbracket) \mathrel{[]} \top : \llbracket \overline{\varphi_{n+1} : P_{n+1}} \rrbracket \\
&\sim \text{(by } IH\text{)} \\
&\quad \textbf{case } \top : (\textbf{case } \varphi_1 : P_1 \mathrel{[]} \dots \mathrel{[]} \varphi_n : P_n) \mathrel{[]} \top : \llbracket \overline{\varphi_{n+1} : P_{n+1}} \rrbracket \\
&= \textbf{case } \top : P' \mathrel{[]} \top : \llbracket \overline{\varphi_{n+1} : P_{n+1}} \rrbracket \\
&= Q'
\end{aligned}$$

We distinguish the cases of φ_{n+1}:

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

$$\begin{aligned}
Q' &= \textbf{case } \top : P' \mathrel{[]} \top : \llbracket \overline{\top : P_{n+1}} \rrbracket \\
&= \textbf{case } \top : P' \mathrel{[]} \top : \llbracket \overline{P_{n+1}} \rrbracket \\
&\sim \text{(by } IH_{n+1}\text{)} \\
&\quad \textbf{case } \top : P' \mathrel{[]} \top : P_{n+1} \\
&\sim \text{(by Lemma 3.3)} \\
&\quad \textbf{case } \varphi_1 : P_1 \mathrel{[]} \dots \mathrel{[]} \varphi_n : P_n \mathrel{[]} \top : P_{n+1} = Q
\end{aligned}$$

We conclude this case.

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

$$\begin{aligned}
Q' &= \textbf{case } \top : P' \mathrel{[]} \top : \llbracket \overline{x = y : P_{n+1}} \rrbracket \\
&= \textbf{case } \top : P' \mathrel{[]} \top : (\textbf{case } x = y : \llbracket \overline{P_{n+1}} \rrbracket) \\
&\sim \text{(by } IH_{n+1}\text{)} \\
&\quad \textbf{case } \top : P' \mathrel{[]} \top : (\textbf{case } x = y : P_{n+1}) \\
&\sim \text{(by Lemma 3.3)} \\
&\quad \textbf{case } \varphi_1 : P_1 \mathrel{[]} \dots \mathrel{[]} \varphi_n : P_n \mathrel{[]} \top : (\textbf{case } x = y : P_{n+1}) \\
&\sim \text{(by Lemma 3.3)} \\
&\quad \textbf{case } \varphi_1 : P_1 \mathrel{[]} \dots \mathrel{[]} \varphi_n : P_n \mathrel{[]} x = y : P_{n+1} = Q
\end{aligned}$$

By concluding this case, we conclude the proof. □

Lemma A.5. *$\llbracket\cdot\rrbracket$ is injective, that is, for all P, Q, if $\llbracket P\rrbracket = \llbracket Q\rrbracket$ then $P = Q$.*

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

Lemma A.6. *$\llbracket\cdot\rrbracket$ is surjective up to $\sim$, that is, for every P there is a Q such that $\llbracket Q\rrbracket \sim P$.*

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', $\llbracket Q'\rrbracket \sim P'$. Let $Q = x(\tilde{y}).Q'$. Then, $\llbracket Q\rrbracket = \llbracket x(\tilde{y}).Q'\rrbracket = \underline{x}(\lambda\widetilde{y})\langle\widetilde{y}\rangle.\llbracket Q'\rrbracket \sim \underline{x}(\lambda\widetilde{y})\langle\widetilde{y}\rangle.P'$. This is what we needed to derive.

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

case $P \mid P'$:
By IH, we have that for some Q', Q'', $\llbracket Q'\rrbracket \sim P$ and $\llbracket Q''\rrbracket \sim P'$. Then let $Q = Q' | Q''$, thus $\llbracket Q\rrbracket = \llbracket Q'\rrbracket \mid \llbracket Q''\rrbracket \sim P \mid P'$.

case $(\nu x)P$:
By IH, for some Q', $\llbracket Q'\rrbracket \sim P$. Let $Q = \nu x Q'$. Then $\llbracket Q\rrbracket = (\nu x)\llbracket Q'\rrbracket \sim (\nu x)P$.

case $!P$:
By IH, for some Q', $\llbracket Q'\rrbracket \sim P$. Let $Q = !Q'$. Then $\llbracket Q\rrbracket = !\llbracket Q'\rrbracket \sim !P$.

case $(\!|\mathbf{1}|\!)$:
Let $Q = \mathbf{0}$. Then $\llbracket Q\rrbracket = \mathbf{0} \sim (\!|\mathbf{1}|\!)$.

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

base case:
Let $Q = \mathbf{0}$, then $\llbracket Q\rrbracket = \mathbf{0} \sim \mathbf{case}$.

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

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

We need to show that there is some $\llbracket Q\rrbracket$ such that

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

First, we note that IH$_{\mathbf{case}}$ holds for every i and in particular $i = n + 1$, thus we get $\llbracket Q'_{n+1}\rrbracket \sim P_{n+1}$. 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

$$\begin{aligned}
\llbracket Q \rrbracket &= \mathbf{case}\ \top : \llbracket Q'' \rrbracket \ [\!]\ \top : \llbracket Q'_{n+1} \rrbracket \\
&\sim \mathbf{case}\ \top : (\mathbf{case}\ \varphi_1 : P_1 \ [\!]\ \dots\ [\!]\ \varphi_n : P_n) \\
&\ [\!]\ \top : \llbracket Q'_{n+1} \rrbracket \\
&\sim \mathbf{case}\ \top : (\mathbf{case}\ \varphi_1 : P_1 \ [\!]\ \dots\ [\!]\ \varphi_n : P_n) \\
&\ [\!]\ \top : P_{n+1} \\
&\sim \text{(by Lemma 3.3)} \\
&\ \mathbf{case}\ \varphi_1 : P_1 \ [\!]\ \dots\ [\!]\ \varphi_n : P_n \\
&\ [\!]\ \top : P_{n+1}
\end{aligned}$$

This case is concluded.

case $\varphi_{n+1} = x = y$**:**
Let $Q = Q'' + [x = y]Q'_{n+1}$. Then

$$\begin{aligned}
\llbracket Q \rrbracket &= \mathbf{case}\ \top : \llbracket Q'' \rrbracket \ [\!]\ \top : \llbracket [x = y]Q'_{n+1} \rrbracket \\
&\sim \mathbf{case}\ \top : (\mathbf{case}\ \varphi_1 : P_1 \ [\!]\ \dots\ [\!]\ \varphi_n : P_n) \\
&\ [\!]\ \top : (\mathbf{case}\ x = y : \llbracket Q'_{n+1} \rrbracket) \\
&\sim \mathbf{case}\ \top : (\mathbf{case}\ \varphi_1 : P_1 \ [\!]\ \dots\ [\!]\ \varphi_n : P_n) \\
&\ [\!]\ \top : (\mathbf{case}\ x = y : P_{n+1}) \\
&\sim \text{(by Lemma 3.3)} \\
&\ \mathbf{case}\ \varphi_1 : P_1 \ [\!]\ \dots\ [\!]\ \varphi_n : P_n \\
&\ [\!]\ \top : (\mathbf{case}\ x = y : P_{n+1}) \\
&\sim \text{(by permuting and applying Lemma 3.3)} \\
&\ \mathbf{case}\ \varphi_1 : P_1 \ [\!]\ \dots\ [\!]\ \varphi_n : P_n \ [\!]\ x = y : P_{n+1}
\end{aligned}$$

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

Theorem A.7. $\llbracket \cdot \rrbracket$ *is an isomorphism up to* $\sim$.

Proof. Directly follows from Lemma A.5 and Lemma A.6. □

A.2. **Polyadic Synchronisation Pi-Calculus.** We follow the exposition of Polyadic Synchronisation Pi-Calculus, ${}^{e}\pi$, of Carbone and Maffeis [CM03].

We give an explicit definition of encoding function defined in Example 4.4.

Definition A.8 (Polyadic synchronisation pi-calculus to **PSPi**).
Agents:

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

Actions:

$$\begin{array}{rcl}
\llbracket \tilde{x}\langle \nu c\rangle \rrbracket & = & \overline{\langle\tilde{x}\rangle}\,(\nu c)\,c \\
\llbracket \tilde{x}\langle c\rangle \rrbracket & = & \overline{\langle\tilde{x}\rangle}\,c \\
\llbracket \tau \rrbracket & = & \tau \\
\llbracket \tilde{x}(y) \rrbracket & = & \text{undefined}
\end{array}$$

Because in [CM03] Carbone and Maffeis defines late style laballed semantics for ${}^{e}\pi$ the input action has no translation.

Definition A.9 (**PSPi** to Polyadic synchronisation pi-calculus)**.**

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

Lemma A.10. *If $P \equiv Q$ then $\llbracket P \rrbracket \sim \llbracket Q \rrbracket$*

Proof. The relation $\mathcal{R} = \{(P, Q) : \llbracket P \rrbracket \sim \llbracket Q \rrbracket\}$ 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 4.16.

(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

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

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

$$\textbf{case}\ \top : \underline{\langle\tilde{x}_1\rangle}(\lambda y_1)y_1.\llbracket P_1 \rrbracket\ []\ \cdots\ []\ \top : \underline{\langle\tilde{x}_i\rangle}(\lambda y_i)y_i.\llbracket P_i \rrbracket \xrightarrow{\langle\tilde{x}\rangle\ z} \llbracket P_i \rrbracket[y_i := z]$$

Finally, we have $P'' = \llbracket P_i \rrbracket[y_i := z]$ and use reflexivity of $\sim$.

Bang:

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

Par:

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

Struct:
Here $P \equiv Q$, $Q \xrightarrow{\tilde{x}(y)} Q'$ and $Q' \equiv P'$. By induction we obtain Q'' such that $\llbracket Q \rrbracket \xrightarrow{\langle\tilde{x}\rangle\ z} Q''$ where $Q'' \sim \llbracket Q' \rrbracket[y := z]$. By Lemma A.10, $\llbracket P \rrbracket \sim \llbracket Q \rrbracket$ and $\llbracket Q' \rrbracket \sim \llbracket P' \rrbracket$, and by definition of $\sim$, $\llbracket Q' \rrbracket[y := z] \sim \llbracket P' \rrbracket[y := z]$. Since $\llbracket P \rrbracket \sim \llbracket Q \rrbracket$ and $\llbracket Q \rrbracket \xrightarrow{\langle\tilde{x}\rangle\ z} Q''$, there exists P'' such that $\llbracket P \rrbracket \xrightarrow{\langle\tilde{x}\rangle\ z} P''$ and $Q'' \sim P''$. By transitivity of $\sim$, $P'' \sim \llbracket P' \rrbracket[y := z]$.

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

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

$$\llbracket P \rrbracket \mid \llbracket Q \rrbracket \xrightarrow{\tau} P'' \mid Q''$$

To complete the induction case, we note that $(\nu y)(P'' \mid Q'') \sim \llbracket (\nu y)(P' \mid Q'\{y/z\}) \rrbracket$

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, $\llbracket P \rrbracket \xrightarrow{\overline{\langle\tilde{x}\rangle}\ (\nu y)\ y} P''$ where $P'' \sim \llbracket P' \rrbracket$ and by the previous clause of this theorem, $\llbracket Q \rrbracket \xrightarrow{\langle\tilde{x}\rangle\ y} Q''$ such that $\llbracket Q' \rrbracket[y := y] = \llbracket Q' \rrbracket \sim Q''$. The Com rule lets us derive the transition

$$\llbracket P \rrbracket \mid \llbracket Q \rrbracket \xrightarrow{\tau} (\nu y)(P'' \mid Q'')$$

To complete the induction case, we note that $(\nu y)(P'' \mid Q'') \sim \llbracket (\nu y)(P' \mid Q') \rrbracket$

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

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

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

Case:
Here $P_C \xrightarrow{\underline{\tilde{x}}\ z} P''$, where $P_C = \mathbf{case}\ \widetilde{\varphi} : \widetilde{Q}$ is in the range of $\llbracket\cdot\rrbracket$ - hence P_C must be the encoding of some prefix-guarded sum, ie $P_C = \llbracket \Sigma_i \alpha_i.P_i \rrbracket = \mathbf{case}\ \top : \llbracket \alpha_1 \rrbracket.\llbracket P_1 \rrbracket\ [\!]\ \ldots\ [\!]\ \top : \llbracket \alpha_i \rrbracket.\llbracket P_i \rrbracket$. 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 $[\![P]\!] \mid ![\![P]\!] \xrightarrow{x\,\langle\tilde{z}\rangle} P''$ 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 $[\![P]\!] \xrightarrow{x\,\langle\tilde{z}\rangle} P''$, $y\#P, Q$, $a\#\tilde{x}, y, z$ 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 $[\![P]\!] \xrightarrow{\overline{\langle\tilde{x}\rangle}\,(\nu\widetilde{y'})\,y} P''$, $[\![Q]\!] \xrightarrow{\langle\tilde{x}\rangle\,y} Q''$ and $y'\#Q$. Either $\widetilde{y'} = \epsilon$ or $\widetilde{y'} = y$; we proceed by case analysis.

(a) If $\widetilde{y'} = \epsilon$, 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 $\widetilde{y'} = y$, 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. The definition of congruence for polyadic synchronisation pi-calculus can be found in [CM03] on page 6.

Theorem A.11. *For all ${}^e\pi$ processes P and Q, $P \sim Q$ iff $[\![P]\!] \sim [\![Q]\!]$*

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

(1) If $P \xrightarrow{\tilde{x}(y)} P'$ and $[\![P]\!] \sim [\![Q]\!]$, since $\mathcal{R}$ is equivariant, we can assume that $y\#P, Q$ without loss of generality. Fix z. By Theorem 4.16 (1), $[\![P]\!] \xrightarrow{\langle\tilde{x}\rangle\,z} P''$ where $P'' \sim [\![P']\!][y := z] = [\![P'\{z/y\}]\!]$. Hence, since $[\![P]\!] \sim [\![Q]\!]$, $[\![Q]\!] \xrightarrow{\langle\tilde{x}\rangle\,z} Q''$ where $P'' \sim Q''$. Hence, by Theorem 4.16.3 using $y\#Q$, $Q \xrightarrow{\tilde{x}(y)} Q'$ where $[\![Q'\{z/y\}]\!] = Q''$. By transitivity, $[\![P'\{z/y\}]\!] \sim [\![Q'\{z/y\}]\!]$.

(2) If $P \xrightarrow{\alpha} P'$ and $[\![P]\!] \sim [\![Q]\!]$, since $\mathcal{R}$ is equivariant, we can assume that $\mathrm{bn}(\alpha)\#P, Q$ without loss of generality. By Theorem 4.16.2, we have that $[\![P]\!] \xrightarrow{[\![\alpha]\!]} P''$ with $P'' \sim [\![P']\!]$. Hence, since $[\![P]\!] \sim [\![Q]\!]$ and $\mathrm{bn}(\alpha)\#Q$, there is a Q'' such that $[\![Q]\!] \xrightarrow{[\![\alpha]\!]} Q''$

and $Q'' \sim P''$. By Theorem 4.16.4, there is Q' such that $Q \xrightarrow{\alpha} Q'$ and $\llbracket Q' \rrbracket = Q''$. By transitivity, $\llbracket P' \rrbracket \sim \llbracket Q' \rrbracket$.

Symmetrically, we show that $\mathcal{R} = \{(\mathbf{1}, \llbracket P \rrbracket, \llbracket Q \rrbracket) : P \sim Q\}$ is a congruence in **PSPI**:

Static equivalence:
Trivial since there is only a unit assertion.

Symmetry:
By symmetry of $\sim$

Simulation:
Here $\llbracket P \rrbracket \xrightarrow{\alpha'} P''$ and $P \sim Q$. We proceed by case analysis on α':

(1) If $\alpha' = \underline{\langle \tilde{x} \rangle}\ z$, then by Theorem 4.16 (3) and a sufficiently fresh y, $P \xrightarrow{\tilde{x}(y)} P'$ where $\llbracket P'\{z/y\} \rrbracket = P''$. Since $P \sim Q$, there exists Q' such that $Q \xrightarrow{\tilde{x}(y)} Q'$ and $P'\{z/y\} \sim Q'\{z/y\}$. Hence, by Theorem 4.16 (1), $\llbracket Q \rrbracket \xrightarrow{\langle \tilde{x} \rangle\ z} Q''$ where $Q'' \sim \llbracket Q' \rrbracket[y := z] = \llbracket Q'\{z/y\} \rrbracket$. We have that $P'' = \llbracket P'\{z/y\} \rrbracket \mathrel{\mathcal{R}} \llbracket Q'\{z/y\} \rrbracket \sim Q''$, 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 $\llbracket P \rrbracket \xrightarrow{\alpha'} P''$, by Theorem 4.16 (4) we have that $P \xrightarrow{\alpha} P'$ where $\llbracket \alpha \rrbracket = \alpha'$ and $\llbracket P' \rrbracket = P''$. Since $P \sim Q$, there is Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \sim Q'$. By Theorem 4.16 (2), $\llbracket Q \rrbracket \xrightarrow{\llbracket \alpha \rrbracket} Q''$, where $Q'' \sim \llbracket Q' \rrbracket$. Hence $P'' = \llbracket P' \rrbracket \mathrel{\mathcal{R}} \llbracket Q' \rrbracket \sim Q''$, which suffices.

Extension of arbitrary assertion:
Trivial since there is only a unit assertion. □

Lemma A.12. $\llbracket \cdot \rrbracket$ *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* $\llbracket Q \rrbracket \sim P$.

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', $\llbracket Q' \rrbracket \sim P'$. Let $Q = \widetilde{x}(y).Q'$. Then $\llbracket Q \rrbracket = \underline{\langle \widetilde{x} \rangle}(\lambda y)y.\llbracket Q' \rrbracket \sim \underline{\langle \widetilde{x} \rangle}(\lambda y)y.P'$.

case $\overline{\langle \widetilde{x} \rangle}\ y.P'$**:**
From IH, we get for some Q', $\llbracket Q' \rrbracket \sim P'$. Let $Q = \widetilde{x}\langle y \rangle.Q'$. Then $\llbracket Q \rrbracket = \overline{\langle \widetilde{x} \rangle}\ y.\llbracket Q' \rrbracket \sim \overline{\langle \widetilde{x} \rangle}\ y.P'$.

case $P' \mid P''$**:**
From IH, for some Q', Q'', we have $\llbracket Q' \rrbracket \sim P'$ and $\llbracket Q'' \rrbracket \sim P''$. Let $Q = Q' \mid Q''$. Then $\llbracket Q \rrbracket = \llbracket Q' \rrbracket \mid \llbracket Q'' \rrbracket \sim P' \mid P''$.

case $(\nu x)P'$**:**
Let $Q = \nu x Q'$, then by induction hypothesis $\llbracket Q \rrbracket = (\nu x)\llbracket Q' \rrbracket \sim (\nu x)P'$.

case $!P'$**:**
Let $Q =!Q'$ (Q' from IH). $\llbracket Q \rrbracket = !\llbracket Q' \rrbracket \sim !P'$.

case $\mathbf{0}$**:**
Then $\llbracket \mathbf{0} \rrbracket = \mathbf{0} \sim \mathbf{0}$.

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

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

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

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

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

$$\begin{array}{lll} [\![Q]\!] & = & [\![Q'']\!] \\ & \sim & \mathbf{case}\ \varphi_1 : P_1 \ [\!]\ \dots\ [\!]\ \varphi_n : P_n \\ & \sim & \mathbf{case}\ \varphi_1 : P_1 \ [\!]\ \dots\ [\!]\ \varphi_n : P_n \ [\!]\ \bot : P_{n+1} \end{array}$$

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 $[\![Q]\!] = \mathbf{case}\ \top : [\![Q'']\!] \ [\!]\ \top : [\![Q'_{n+1}]\!] \sim \mathbf{case}\ \varphi_1 : P_1 \ [\!]\ \dots\ [\!]\ \varphi_n : P_n \ [\!]\ \top : P_{n+1}$.
In case Q'' is a sum, let $Q = Q'' + Q'_{n+1}$. Since Q'_{n+1} is guarded, Q is well formed. Then

$$\begin{array}{lll} [\![Q]\!] & = & \mathbf{case}\ \top : [\![Q'']\!] \ [\!]\ \top : [\![Q'_{n+1}]\!] \\ & \sim & \mathbf{case}\ \top : (\mathbf{case}\ \varphi_1 : P_1 \ [\!]\ \dots\ [\!]\ \varphi_n : P_n) \\ & & [\!]\ \top : [\![Q'_{n+1}]\!] \\ & \sim & \text{(by Lemma 3.3)} \\ & & \mathbf{case}\ \varphi_1 : P_1 \ [\!]\ \dots\ [\!]\ \varphi_n : P_n \\ & & [\!]\ \top : [\![Q'_{n+1}]\!] \\ & \sim & \mathbf{case}\ \varphi_1 : P_1 \ [\!]\ \dots\ [\!]\ \varphi_n : P_n \\ & & [\!]\ \top : P'_{n+1} \end{array}$$

This concludes the proof. □

Lemma A.13. *$\llbracket \cdot \rrbracket$ is injective, that is, for all P, Q, if $\llbracket P \rrbracket = \llbracket Q \rrbracket$ then $P = Q$.*

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

Theorem A.14. *$\llbracket \cdot \rrbracket$ is an isomorphism up to $\sim$ between ${}^{e}\pi$ and the case-guarded processes in* **PSPI**.

Proof. Directly follows from Lemma A.13 and Lemma A.12. □

A.3. **Value-passing CCS.**

Lemma A.15. *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. □

We assume a reverse translation $\widehat{\cdot}$ from **VPCCS** to value-passing CCS. We prove strong operational correspondence.

Proof of Theorem 4.23.

(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 $\overline{x}\ v.\llbracket P \rrbracket \xrightarrow{\overline{x}\ v} \llbracket P \rrbracket$

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 $\underline{x}(\lambda y)y.\llbracket P \rrbracket \xrightarrow{x\ v} \llbracket P\{v/y\rrbracket$ using the In rule.

(b) Otherwise, we have that $P_j \xrightarrow{\alpha} P'$ and by induction,

$$\llbracket P_j \rrbracket \xrightarrow{\llbracket \alpha \rrbracket} \llbracket P' \rrbracket$$

The Case rule lets us derive

$$\mathbf{case}\ \top : \llbracket P_1 \rrbracket\ []\ \cdots\ []\ \top : P_i \xrightarrow{\llbracket \alpha \rrbracket} \llbracket P' \rrbracket$$

This suffices since $\llbracket \Sigma_i P_i \rrbracket = \mathbf{case}\ \top : \llbracket P_1 \rrbracket\ []\ \cdots\ []\ \top : P_i$.

Com1:

Here $P \xrightarrow{\alpha} P'$ and by induction, $\llbracket P \rrbracket \xrightarrow{\llbracket \alpha \rrbracket} \llbracket P' \rrbracket$. The Par rule admits derivation of the transition $\llbracket P \rrbracket \mid \llbracket Q \rrbracket \xrightarrow{\llbracket \alpha \rrbracket} \llbracket P' \rrbracket \mid \llbracket Q \rrbracket$, using Lemma A.15 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 A.15 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 A.15, $\tilde{y} = \epsilon$, 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, **if** $\widehat{b\ \textbf{then}}\ P = \widehat{P}$ - 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. □