J Autom Reasoning (2016) 56:1–47
DOI 10.1007/s10817-015-9336-2

# Psi-Calculi in Isabelle

**Jesper Bengtson**[1] · **Joachim Parrow**[2] · **Tjark Weber**[2]

Received: 3 March 2013 / Accepted: 30 January 2014 / Published online: 19 August 2015


**Abstract** This paper presents a mechanisation of psi-calculi, a parametric framework for modelling various dialects of process calculi including (but not limited to) the pi-calculus, the applied pi-calculus, and the spi calculus. psi-calculi are significantly more expressive, yet their semantics is as simple in structure as the semantics of the original pi-calculus. Proofs of meta-theoretic properties for psi-calculi are more involved, however, not least because psi-calculi (unlike simpler calculi) utilise binders that bind multiple names at once. The mechanisation is carried out in the Nominal Isabelle framework, an interactive proof assistant designed to facilitate formal reasoning about calculi with binders. Our main contributions are twofold. First, we have developed techniques that allow efficient reasoning about calculi that bind multiple names in Nominal Isabelle. Second, we have adopted these techniques to mechanise substantial results from the meta-theory of psi-calculi, including congruence properties of bisimilarity and the laws of structural congruence. To our knowledge, this is the most extensive formalisation of process calculi mechanised in a proof assistant to date.

**Keywords** Psi-calculi · Process calculi · Proof assistants · Nominal logic · Mechanisation

In memory of Robin Milner

✉ Jesper Bengtson
jebe@itu.dk

Joachim Parrow
joachim.parrow@it.uu.se

Tjark Weber
tjark.weber@it.uu.se

[1] IT University of Copenhagen, Copenhagen, Denmark

[2] University of Uppsala, Uppsala, Sweden

## 1 Introduction

Process calculi are commonly used to describe the behaviour of concurrent systems. Seminal calculi that were developed in the early 1980s include Milner's CCS [35], Hoare's CSP [27], and Bergstra and Klop's ACP [14]. More recent examples are the pi-calculus [39] and its variants, for instance, the applied pi-calculus of Abadi and Fournet [1], and the concurrent constraint pi-calculus by Buscemi and Montanari [20]. These calculi provide high-level modelling primitives to describe interactions between independent agents. Algebraic laws allow the manipulation of process descriptions, and precise semantics permit formal reasoning about properties such as process equivalence.

For any such formalism to be practically useful, fundamental results must be established about it. One example is compositionality: that the semantics of a process can be deduced from the semantics of its components. This is crucial for dividing the construction of a system into parts that can be analysed separately.

Proving such properties of a process calculus often requires stamina and attention to detail. Intricate induction proofs and case analyses necessitate a fair amount of bookkeeping. When this is done using pen and paper, there is a temptation to take shortcuts by glossing over seemingly trivial parts. In some cases, this has led to the publication of erroneous proofs. For instance, both the applied pi-calculus and the concurrent constraint pi-calculus have been discovered to have flaws or incompletenesses in the sense that their claimed compositionality results do not hold [9]. Despite much research in the area, these errors went unnoticed for several years, indicating the complexity of these proofs.

The problem is aggravated by a trade-off between simplicity and elegance of a calculus on the one hand, and modelling convenience on the other hand. As an illustrative example, consider the lambda-calculus [23]: its minimal language makes it relatively easy to prove meta-theoretic properties (easier at least than for a full-fledged functional language, such as Standard ML [40]), but one would not want to write programs directly in the lambda-calculus. Similarly, there is no one-size-fits-all approach to process calculi. Some process calculi use a parsimonious language to explore fundamental principles of computing and facilitate proofs of meta-theoretic properties, while others are more tailored to application areas and include many constructions for modelling convenience. Such formalisms are now being developed en masse. There is a danger that for more applied calculi, proofs of meta-theoretic properties become gruesome and, therefore, will not be not carried out with the necessary care.

Proof assistants such as Coq [15], HOL4 [47] or Isabelle [52] can be of great benefit in this setting. These software tools provide excellent support to manage unwieldy case distinctions and large numbers of assumptions; they diligently keep track of every detail of a formalisation. Convincing a proof assistant that a statement is true may be a difficult and sometimes tedious task, not least because one cannot resort to hand-waving. The reward is that one obtains an unprecedented level of confidence in the statements so proven. Moreover, changes or additions to a calculus that would otherwise require weeks of careful proof revision can be checked mostly automatically, sometimes in minutes [7]. When Aydemir et al. posed the POPLmark challenge [3], they enunciated their vision of "a future in which the papers in conferences such as Principles of Programming Languages [...] are routinely accompanied by mechanically checkable proofs of the theorems they claim." We share this vision, and add that mechanised proofs are as valuable for process calculi as they are for programming languages.

Taking this future one step closer to becoming reality, this paper presents a formalisation of psi-calculi in Isabelle. Psi-calculi (Section 2) are a parametric family of process calculi that aim to resolve the conflict between elegance and modelling convenience sketched above. Obtained as extensions of the pi-calculus, psi-calculi have a truly compositional labelled operational semantics (without the complications of stratified process definitions, structural congruence or explicit quantification over contexts found in other calculi), as simple in structure as the semantics of the original pi-calculus. Yet their expressiveness significantly exceeds that of the applied pi-calculus. They accommodate pi-calculus extensions such as the spi-calculus [2], the fusion calculus [25], concurrent constraints, and polyadic synchronisation [21], to name but a few.

This paper details the first proof mechanisation of a family of process calculi of this calibre. The formalisation comprises approximately 32,000 lines of definitions and proofs and is available from the Archive of Formal Proofs [8]. One interesting aspect and a major difficulty when formalising any calculus with binders is the treatment of alpha-equivalence. For this, we base our formalisation on Urban's Nominal Isabelle framework [48]. More specifically, our contributions are the following.

- We extend Nominal Isabelle to reason atomically about sequences of binders, as opposed to single binders (Section 3).
- We define the basic notion of psi-calculi agents (Section 4) and their labelled operational semantics (Section 5). We achieve parametricity, i.e., the ability to instantiate our formal framework to concrete process calculi, through the use of *locales* [5], Isabelle's mechanism for local specifications.
- We use the induction rules provided by Nominal Isabelle to derive custom induction rules that remove the bulk of manual alpha-conversions, keeping the machine-checked proofs as close to their pen-and-paper counterparts as possible (Section 5).
- We describe heuristics for generating inversion principles (principles for case analysis) for calculi that use sequences of binders (Section 6). Nominal Isabelle has generated induction principles for such calculi (and also for calculi with single binders) for some time now, but adapting these techniques to cover inversion principles has been an open problem.
- In a similar way as for the pi-calculus, we define strong bisimilarity and prove that it is preserved by all operators except the input prefix (Section 7).
- We define strong equivalence by closing strong bisimilarity under parallel substitutions and prove that it is a congruence (Section 8).
- We define weak bisimilarity, which considers $\tau$-actions unobservable, and prove that it is preserved by all operators except nondetermistic choice and the input prefix.
- We define weak equivalence and prove that it is a congruence.
- We prove the tau-laws for weak bisimilarity.
- To facilitate reasoning about our operational semantics it does not include structural congruence. As a sanity check, we prove that all versions of bisimilarity preserve the laws of structural congruence.

In this paper, we focus on describing the first six of these items. Although the last four items represent a substantial amount of work we only treat them summarily; a thorough presentation can be found in the first author's PhD thesis [7]. This paper is partially based on a previous conference paper [11], but it covers the material in more detail and contains several new elements. Notably, the requirements on substitution (Section 4.3) have been simplified,

and Sections 6–8 (on inversion principles and the formalisation of strong bisimilarity and strong equivalence) are new.

## 2 Background

To achieve an elegant treatment of alpha-equivalence, we base our formalisation of psi-calculi on nominal logic [45]. In this section, we recapitulate the core concepts, as supported by the Nominal Isabelle [48] framework. We also provide a brief background on psi-calculi. For a more extensive treatment including motivations and examples see [9].

### 2.1 Nominal Logic

Nominal logic is a formalism designed to simplify the treatment of calculi involving binders. In informal reasoning about such calculi, the Barendregt variable convention [6] is often used. This convention states that all bound variables are distinct from the free variables that appear in a given mathematical context. The variable convention is difficult to justify formally; in fact, it is unsound in general when used in rule inductions [49]. Nominal logic allows reasoning about terms with binders up to alpha-equivalence. It thus places the informal, but convenient style of reasoning that is often practised in pen-and-paper proofs on a sound theoretical footing.

At the core of nominal logic is a countably infinite set of atomic *names* $\mathcal{N}$, ranged over by $a, \ldots, z$. In our formalisation, names will represent the symbols that can be statically scoped. They will also represent symbols acting as variables, in the sense that they can be subjected to substitution. A typed calculus would distinguish names of different kinds [16], but our account will be untyped.

A *nominal set* [45] is a set equipped with *name swapping* functions. The latter are written $(a\,b)$ for any names $a$ and $b$. Intuitively, for any member $X$ of the nominal set it holds that $(a\,b) \cdot X$ is $X$ with $a$ replaced by $b$ and $b$ replaced by $a$. Formally, a name swapping function is any function satisfying certain natural axioms, such as $(a\,b) \cdot (a\,b) \cdot X = X$. A *permutation* is a list of name swappings. We write $\varepsilon$ for the empty list, and $x\widetilde{x}$ for a list consisting of head $x$ and tail $\widetilde{x}$. Application is lifted from name swappings to permutations: $\varepsilon \cdot X = X$, and $(a\,b)\widetilde{x} \cdot X = (a\,b) \cdot \widetilde{x} \cdot X$. Application of permutations to tuples, lists, and other inductive data types is defined homomorphically: e.g., $p \cdot (u, v) = (p \cdot u, p \cdot v)$.

Even though we have not specified any particular syntax for elements of a nominal set, name swappings allow us to define what it means for a name to *occur* in an element: namely that it can be affected by swappings.

The names occurring in an element $X$ constitute the *support* of $X$, written $supp\ X$. We write $a \# X$, pronounced "a is fresh for X," for $a \notin supp\ X$. For instance, if the nominal set is an inductively defined data type, we have $a \# X$ if and only if $a$ does not occur syntactically in $X$. In the lambda-calculus, where alpha-equivalent terms are identified, the support of a term is the set of its free variables. If $A$ is a set of names, we write $A \# X$ to mean $\forall a \in A.\, a \# X$.

We require all elements to have finite support, i.e., $supp\ X$ is finite (possibly empty) for all $X$. Given a finite collection of elements $X_1, \ldots, X_n$, this requirement ensures the existence of infinitely many names $a$ such that $a \# X_1, \ldots, a \# X_n$.

A function $f$ is said to be *equivariant* if $(a\,b) \cdot f(X) = f((a\,b) \cdot X)$ for all $X$, and similarly for functions and relations of any arity. Intuitively, this means that $f$ treats all names equally. We write *eqvt* $f$ when $f$ is equivariant.

A *nominal data type* is a nominal set equipped with a collection of equivariant functions. In particular we shall consider *substitution functions*, which intuitively substitute elements for names. If $X$ is an element of a nominal data type, $\widetilde{x}$ is a sequence of names without duplicates, and $\widetilde{T}$ is an equally long sequence of elements, the substitution $X[\widetilde{x} := \widetilde{T}]$ is an element of the same data type as $X$. In a traditional (inductively defined) data type, substitution can be thought of as replacing all occurrences of names in $\widetilde{x}$ by corresponding elements in $\widetilde{T}$. In a calculus with binders, it can be thought of as replacing the free names, alpha-converting bound names as necessary to avoid capture. Formally, a substitution can be any equivariant function that satisfies certain substitution laws. The full list of these is given in Section 4.3.

By using nominal data types we obtain a general framework that allows many different instantiations. Our only requirements are on the notions of support, name swapping, and substitution. This corresponds precisely to the essential ingredients for data transmitted between agents (Section 2.2). Since names can be statically scoped and data sent into and out of scope boundaries, it must be possible to discern exactly which names are contained in a data item, and this is just the role of the support. In case a data element intrudes a scope, the scoped name needs to be alpha-converted to avoid clashes, and name swapping can achieve precisely this. When a term is received in a communication between agents, it must replace all occurrences of the placeholder in the input construct; this requires a substitution function that substitutes the received term for the placeholder.

Since these are our only assumptions on data terms, the psi-calculi framework can be instantiated also to data types that are not inductively defined, such as equivalence classes and sets defined by comprehension or co-induction. Examples include higher-order data types such as the lambda-calculus, or even agents of a psi-calculus.

Similarly, the notions of conditions (i.e., tests on data that agents can perform during their execution) and assertions (i.e., facts that can be used to resolve conditions) are formulated as nominal data types. This means that logics with binders, and even higher-order logics can be used. Moreover, alpha-variants of terms are formally equated, thereby facilitating the formalism and proofs.

## 2.2 Psi-Calculi

Psi-calculi provide a framework where a range of process calculi can be formulated with a lean and symmetric semantics, and where proofs can be conducted using straightforward induction, without resorting to a structural congruence or explicit quantification over contexts. This section gives a brief informal introduction to psi-calculi, in order to keep this paper self-contained. We refer to our formal development in Sections 4–7 for details, and to [9] for further explanations and examples.

Historically, psi-calculi evolved from the pi-calculus [39]. In the (basic, untyped) pi-calculus, a concurrent system is composed of agents that communicate names across channels. Names are scoped (i.e., known only to certain agents), but may be sent to agents outside their current scope. Names serve a dual role: they function as both channel names and communicated objects. This allows the pi-calculus to conveniently model changes in

the communication structure, such as those commonplace in mobile networks. A pi-calculus agent may send or receive a name on a channel, make a non-deterministic choice, execute in parallel with another agent, replicate (i.e., create a copy of itself), create a fresh (local) name, or test for equality of names. Based on this parsimonious language, the pi-calculus aims to be a universal model for concurrent computation. Similarly to the lambda calculus, it is Turing-complete, and does not contain primitives for numbers, lists, or other data structures. In contrast to the lambda calculus, it has no primitive notion of substituting a term for a name, and can thus be said to capture computation on a much lower level.

Psi-calculi enrich the minimal language of the pi-calculus for better modelling convenience, while preserving its relatively simple meta-theory. A psi-calculus is obtained by extending the pi-calculus with three parameters. The first is a set of data terms. These generalise names; like names in the pi-calculus, data terms function as both communication channels and communicated objects. The second is a set of conditions, for use in conditional constructs such as **if** statements. These generalise the test for name equality. The third is a set of assertions, used to express, e.g., constraints or aliases, which can resolve the conditions. These sets need not be disjoint. We assume that terms, conditions, and assertions are given by nominal data types. One of our main results is to identify minimal, general and natural requirements on these types (Section 4).

Psi-calculi are equipped with a labelled operational semantics (Section 5). A transition in this semantics is written $\Psi \triangleright P \xrightarrow{\alpha} P'$, and intuitively means that $\Psi$ is an assertion (representing an environment of $P$) under which the agent $P$ can take action $\alpha$ to become $P'$. Although psi-calculi are significantly more expressive than the applied pi-calculus, the simplicity of their semantics is on par with that of the original pi-calculus.

As a simple example that anticipates the syntax of psi-calculi agents, consider the agent $\underline{M}(\lambda\tilde{x})N.0$, which expects to receive an instance of the pattern $(\lambda\tilde{x})N$ on the channel $M$. The pattern can match any term $N'$ obtained by instantiating the names $\tilde{x}$ in $N$. This can be thought of as a generalisation of the polyadic pi-calculus [37], where the patterns are just tuples of names. When executed in parallel with the agent $\overline{K}N'.0$, which sends the term $N'$ on channel $K$, the two agents can communicate if (and only if) the environment asserts that $M$ and $K$ denote the same channel.

This example touches on three important features of psi-calculi: pattern matching, channel equivalence, and multiple binders (i.e., binders that bind multiple names at once). We formalise multiple binders in Section 3, before introducing psi-calculi agents and their syntax in more detail in Section 4.

## 3 Binding Sequences

A major difficulty when formalising any calculus with binders is to handle alpha-equivalence in a smooth and transparent fashion. Like other techniques that have been used in proof assistants to handle binders, Nominal Isabelle can only bind a single name at a time. Reasoning about single binders works well for many calculi, but psi-calculi require binding sequences of arbitrary length. As mentioned in Section 2.2, a binding sequence is needed for agents that expect an input: the agent $\underline{M}(\lambda\tilde{x})N.P$ has the sequence $\tilde{x}$ binding into $N$ and $P$. The second place where binding sequences are needed is in the definition of frames (Section 5.1). Frames are derived from agents, and as agents can have an arbitrary

number of binders, so can the frames. The third occurrence of binding sequences is in the operational semantics (Section 5.3). In the transition $\Psi \triangleright P \xrightarrow{\overline{M}(\nu \widetilde{a})N} P'$, where the agent $P$ takes an output action on channel $M$ to become $P'$, the sequence $\widetilde{a}$ represents the bound names in $P$ that occur in the object $N$.

Nominal2 [29] is a recent re-implementation of Nominal Isabelle, in large part motivated by our formalisation of psi-calculi. It supports multiple binders, i.e., structures that bind several names simultaneously, but it is currently not mature enough to be used for our formalisation. More specifically, it does not (yet) support the use of Isabelle's locales. We discuss Nominal2 in more detail in Section 9, together with other related work.

It is important to introduce binding sequences without complicating proofs unnecessarily. In this section we discuss how to adapt, in a clear and transparent manner, the core lemmas used for reasoning about single binders in Nominal Isabelle to handle sequences of binders. All definitions and theorems in the following sections have been formally checked in Isabelle. Where necessary, we will explain Isabelle-specific notation as we go along.

### 3.1 Definition

To obtain binding sequences, we define a nominal data type *bindSeq* by induction: any term of finite support is of type *bindSeq*, and binding a name yields another term of this type.

**Definition 1** (bindSeq)

$$\begin{aligned} \textbf{nominal\_datatype}\ \alpha\ \mathit{bindSeq} = {} & \mathit{Base}\ \alpha \\ \mid {} & \mathit{Bind}\ \ll \mathit{name} \gg\ (\alpha\ \mathit{bindSeq}) \end{aligned}$$

Thus, the nominal data type *bindSeq* is parametric in a type parameter $\alpha$. We generally use postfix notation, e.g., $\alpha$ *bindSeq*, for type constructors. This data type has two constructors: *Base* simply takes an argument of type $\alpha$, while *Bind* is recursive. It takes two arguments, a name and a term of type $\alpha$ *bindSeq*, and binds the name in the latter. Binding is made apparent by guillemets in the notation of Nominal Isabelle.

A binding sequence is obtained from a list of names by recursion, binding one name at a time.

**Definition 2** (bindSequence)

$$\begin{aligned} &\mathit{bindSequence} :: \mathit{name\ list} \Rightarrow \alpha \Rightarrow \alpha\ \mathit{bindSeq} \\ &\mathit{bindSequence}\ \varepsilon\ T = \mathit{Base}\ T \\ &\mathit{bindSequence}\ (x\widetilde{x})\ T = \mathit{Bind}\ x\ (\mathit{bindSequence}\ \widetilde{x}\ T) \end{aligned}$$

Thus, *bindSequence* is a function of two arguments, the first being a list of names and the second of type $\alpha$. We use the notation $[\widetilde{x}].T$ to mean *bindSequence* $\widetilde{x}\ T$. For the rest of this section, we compare the most common nominal mechanisms that are used for calculi with single binders to their counterparts that use binding sequences.

A binding sequence $\widetilde{x}$ is fresh for a term $X$ if all names $x$ in $\widetilde{x}$ are fresh for $X$.

| *Single binder* | *Binding sequence* |
|---|---|
| $a \sharp X \equiv a \notin \mathit{supp}\ X$ | $\widetilde{x} \sharp X \equiv \forall x \in \mathit{set}\ \widetilde{x}.\ x \sharp X$ |

Here, *set* $\widetilde{x}$ is the set of all elements that are contained in the list $\widetilde{x}$.

### 3.2 Alpha-Renaming

There are two steps involved in every alpha-conversion. First, a sufficiently fresh name is chosen. Second, a bound name (and all of its free occurrences under the binder) is replaced with this fresh name. Generating a name that is fresh for any term of finite support is natively supported in nominal logic. For binding sequences, we construct a permutation (given by a list of pairs of names) that, when applied to a sequence of names, ensures that the resulting sequence satisfies all desired freshness conditions.

| *Single binder* | *Binding sequence* |
|---|---|
| $\exists c.\ c \sharp \mathscr{C}$ | $\exists p.\ (p \cdot \widetilde{x}) \sharp \mathscr{C} \wedge \mathit{set}\ p \subseteq \mathit{set}\ \widetilde{x} \times \mathit{set}\ (p \cdot \widetilde{x})$ |

Alpha-renaming for binding sequences then mimics the single binder case very closely.

| *Single binder* | *Binding sequence* |
|---|---|
| $\dfrac{y \sharp T}{[x].T = [y].(x\ y) \cdot T}$ | $\dfrac{(p \cdot \widetilde{x}) \sharp T \qquad \mathit{set}\ p \subseteq \mathit{set}\ \widetilde{x} \times \mathit{set}\ (p \cdot \widetilde{x})}{[\widetilde{x}].T = [(p \cdot \widetilde{x})].(p \cdot T)}$ |

Long proofs tend to introduce many alpha-converting permutations, and it is important to have a means to remove these from all parts of a goal where they are not needed. For any term $p \cdot T$, the permutation $p$ can be removed if either no names in $p$ occur in $T$, or if the inverse permutation $p^-$ can be applied to the goal where $p \cdot T$ is found; $p^-$ will distribute through the goal by equivariance and cancel out when it reaches $p \cdot T$, since $p^- \cdot p \cdot T = T$. It is, however, not generally true that $p \cdot p \cdot T = T$. The case where no name in $p$ occurs in $T$ is unproblematic, and follows its single-binder version closely.

| *Single binder* | *Binding sequence* |
|---|---|
| $\dfrac{a \sharp T \qquad b \sharp T}{(a\ b) \cdot T = T}$ | $\dfrac{\mathit{set}\ p \subseteq \mathit{set}\ \widetilde{x} \times \mathit{set}\ \widetilde{y} \qquad \mathit{set}\ \widetilde{x} \sharp T \qquad \mathit{set}\ \widetilde{y} \sharp T}{p \cdot T = T}$ |

In fact, the single binder version is just a special case of the rule for binding sequences. However, the second of these techniques, i.e., applying the inverse permutation $p^-$ to the goal, is unsatisfactory. As desired, applying $p^-$ will cancel out any occurrence of $p$ in the goal, and moreover disappear from any other subterm that contains none of its names. But the freshness condition $(p \cdot \widetilde{x}) \# \mathscr{C}$ does not generally imply $(p^- \cdot \widetilde{x}) \# \mathscr{C}$. As an example, $(x\ y)(x\ z) \cdot x \# y$ holds, but $(x\ z)(x\ y) \cdot x \# y$ does not. Thus, by introducing inverse permutations we lose freshness properties of binding sequences. For this reason, it is simpler to work with permutations that are their own inverse. The following predicate accomplishes this.

**Definition 3** (distinctPerm) *distinctPerm p* ≡ *distinct ((map fst p) @ (map snd p))*

The *distinct* predicate takes a list as an argument and holds if there are no duplicates in that list. The infix operator @ concatenates two lists. Thus, the *distinctPerm* predicate ensures that all names in a permutation are distinct.

**Lemma 1** *If distinctPerm p then* $p \cdot p \cdot T = T$.

*Proof* By induction on $p$. □

By restricting ourselves to alpha-converting permutations that are their own inverse, we are not required to use explicit inverse permutations, and all freshness properties of binding sequences are preserved.

We ensure that whenever an alpha-converting permutation is generated, it is sufficiently fresh and its own inverse.

| *Single binder* | *Binding sequence* |
|---|---|
| $\exists c.\ c \sharp \mathscr{C}$ | $\exists p.\ (p \cdot \widetilde{x})\ \sharp\ \mathscr{C} \wedge \mathit{distinctPerm}\ p \wedge \mathit{set}\ p \subseteq \mathit{set}\ \widetilde{x} \times \mathit{set}\ (p \cdot \widetilde{x})$ |

### 3.3 Alpha-Equivalence

For single binders, the nominal approach to alpha-equivalence is quite straightforward. Two terms $[x].T$ and $[y].U$ are alpha-equivalent if and only if either $x = y$ and $T = U$, or $x \neq y$, $x\#U$ and $U = (x\ y) \cdot T$. Reasoning about binding sequences is more difficult. Exactly what does it mean for two terms $[\widetilde{x}].T$ and $[\widetilde{y}].U$ to be alpha-equivalent? As long as $T$ and $U$ do not themselves have binding sequences on the top level, we know that $[\widetilde{x}] = [\widetilde{y}]$ (where $[\widetilde{x}]$ denotes the length of the list $\widetilde{x}$). What happens when $\widetilde{x}$ and $\widetilde{y}$ partially share names?

Assumptions such as $[\widetilde{x}].T = [\widetilde{y}].U$ are obtained in proofs when we carry out induction over a term with binders. Typically, $[\widetilde{y}].U$ is the term in the original proof goal, and $[\widetilde{x}].\ T$ is the term that appears in the induction or inversion rule. These rules are designed in such a way that any bound names appearing in the rules can be assumed to be sufficiently fresh. More precisely, we can ensure that $\widetilde{x} \# \widetilde{y}$ and $\widetilde{x} \# U$.

If $[\widetilde{x}]$ is alpha-equivalent to $[\widetilde{y}].U$, there should be a permutation that equates these terms. We first prove the following auxiliary lemma.

**Lemma 2**

$(i)$ *If* $[a].T = [b].U$ *then* $a \in \mathit{supp}\ T \longleftrightarrow b \in \mathit{supp}\ U$.
$(ii)$ *If* $[a].T = [b].U$ *then* $a \sharp T \longleftrightarrow b \sharp U$.

*Proof* By the definition of alpha-equivalence on terms. □

We can now prove the following lemma about the existence of equating permutations for alpha-equivalent binding sequences.

**Lemma 3**

*If* $[\widetilde{x}].T = [\widetilde{y}].U$ *and* $\widetilde{x} \# \widetilde{y}$ *then* $\exists p.\ \mathit{set}\ p\ \subseteq\ \mathit{set}\ \widetilde{x}\ \times\ \mathit{set}\ \widetilde{y}\ \wedge\ \mathit{distinctPerm}\ p\ \wedge U = p \cdot T$.

*Proof* By induction on the length of $\widetilde{x}$ and $\widetilde{y}$. We construct $p$ by using Lemma 2 to filter out the pairs of names from $\widetilde{x}$ and $\widetilde{y}$ that do not occur in $T$ and $U$ respectively, and pairing the rest. Since $\widetilde{x} \# \widetilde{y}$, we obtain a permutation that contains no duplicates. □

We can equate $T$ and $U$ with this technique, but not $\widetilde{x}$ and $\widetilde{y}$. To do this, we must also know that $\widetilde{x}$ and $\widetilde{y}$ are distinct.

**Lemma 4**

$$\frac{[\widetilde{x}].T = [\widetilde{y}].U \qquad \widetilde{x} \mathrel{\sharp\!\!\sharp} \widetilde{y} \qquad \mathit{distinct}\,\widetilde{x} \qquad \mathit{distinct}\,\widetilde{y}}{\exists p.\ \mathit{set}\,p \subseteq \mathit{set}\,\widetilde{x} \times \mathit{set}\,(p \cdot \widetilde{x}) \wedge \mathit{distinctPerm}\,p \wedge \widetilde{y} = p \cdot \widetilde{x} \wedge U = p \cdot T}$$

*Proof* Similar to Lemma 3, but as we know that $\widetilde{x}$ and $\widetilde{y}$ are distinct, and that they share no names, the permutation $p$ is created by pairwise combining the names from both binding sequences. □

### 3.4 Distinct Binding Sequences

Because of Lemma 4, we prefer to work with binding sequences $[\widetilde{x}].T$ such that $\widetilde{x}$ is distinct. A problem is that this property is not necessarily preserved by alpha-conversion. Consider the term $[xy].\mathit{Base}\,y$, where the distinct sequence $xy$ binds into the name $y$. Since a name can only bind into a term once, any further occurrence of the same binder in the sequence will by definition be fresh for everything under its scope, and hence can be freely renamed to any other fresh name. For instance, $[xy].\mathit{Base}\,y = [yy].\mathit{Base}\,y$, so that we cannot a priori assume that distinctness is maintained when working up to alpha-equivalence.

This example motivates the following lemma, which states that any binding sequence can be replaced with a distinct one that is at least as fresh as the original.

**Lemma 5** *If* $\widetilde{x}\#\mathscr{C}$ *then* $\exists\widetilde{y}.[\widetilde{x}].T = [\widetilde{y}].T \wedge \mathit{distinct}\,\widetilde{y} \wedge \widetilde{y}\#\mathscr{C}$

*Proof* Since each name in $\widetilde{x}$ can only bind into $T$ once, we can construct $\widetilde{y}$ by replacing any duplicate name in $\widetilde{x}$ with a sufficiently fresh name. □

If we know that all members of a distinct binding sequence are in the support of the term the sequence is binding into, then alpha-converting the sequence maintains its distinctness.

**Lemma 6**

$$\frac{[\widetilde{x}].T = [\widetilde{y}].U \qquad \mathit{distinct}\,\widetilde{x} \qquad \mathit{set}\,\widetilde{x} \subseteq \mathit{supp}\,T}{\mathit{distinct}\,\widetilde{y}}$$

*Proof* By induction on the length of $\widetilde{x}$ and $\widetilde{y}$. □

Lemmas 5 and 6 allow us to ensure that binding sequences are kept distinct in proof contexts.

## 4 Psi-Calculi

In this section, we describe the formal definition of psi-calculi—that is, parameters of a psi-calculus, requirements on these parameters, and the notions of agents and static equivalence thus obtained. Our definitions have been implemented in Nominal Isabelle.

### 4.1 Terms, Assertions and Conditions

Psi-calculi are parametric in three (not necessarily disjoint) nominal data types:

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

Similarly to names in the pi-calculus, terms represent both data that is sent between agents, and the channels over which this data is communicated. Conditions act as guards for agents using non-deterministic choice, where any branch that has its guard satisfied may execute. Assertions represent the environment in which the agents act; they are used to determine whether conditions hold. They also occur inside agents, the intuition being that as an agent executes, more assertions are added to the evolving environment. The interplay between assertions and conditions is made more precise in Section 5, where the operational semantics of psi-calculi is defined.

### 4.2 Agents

Given the three nominal data types of terms, assertions and conditions, psi-calculi agents $P$, $Q$ are defined by the following grammar:

| | | |
|---|---|---|
| $P, Q ::=$ | $\mathbf{0}$ | Nil |
| | $\overline{M}N.P$ | Output |
| | $\underline{M}(\lambda\tilde{x})N.P$ | Input |
| | $\mathbf{case}\ \varphi_1 : P_1 \,[]\, \cdots \,[]\, \varphi_n : P_n$ | Case |
| | $P \mid Q$ | Parallel |
| | $(\nu a)P$ | Restriction |
| | $(\!\vert \Psi \vert\!)$ | Assertion |
| | $!P$ | Replication |

In the input form $\underline{M}(\lambda\tilde{x})N.P$, we require that $\tilde{x} \subseteq \mathit{supp}\, N$ is a sequence without duplicates. The names in $\tilde{x}$ bind occurrences in both $N$ and $P$. Restriction $(\nu a)P$ binds $a$ in $P$. In the input and output forms, $M$ is called the *subject* and $N$ the *object*. Input and output are similar to those in the pi-calculus, but arbitrary terms can function as both subjects and objects.

Intuitively, the input form $\underline{M}(\lambda\tilde{x})N.P$ expects to receive an instance of the pattern $(\lambda\tilde{x})N$ on the channel $M$. For instance, $\underline{M}(\lambda xy)\,f(x, y).P$ can only communicate with an output $\overline{M}\,f(N_1, N_2).Q$ for some data terms $N_1$, $N_2$. This can be thought of as a generalisation of the polyadic pi-calculus [37], where the patterns are just tuples of names. Another significant extension is that we allow arbitrary data terms also as communication channels. Thus it is possible to include functions that create channels.

The case construct behaves as any one of the agents $P_i$ for which the corresponding condition $\varphi_i$ is true. We sometimes abbreviate $\mathbf{case}\,\varphi_1 : P_1 \,[]\, \cdots \,[]\, \varphi_n : P_n$ as $\mathbf{case}\,\tilde{\varphi} : \tilde{P}$, and when $n = 1$ as $\mathbf{if}\,\varphi_1\,\mathbf{then}\,P_1$.

Defining a corresponding data type of agents in Isabelle is not entirely straightforward. Nominal data types in Isabelle are restricted in the sense that neither nested data types nor nested binders are permitted. When defining psi-calculi agents this is problematic in two respects. First, the input form requires that an arbitrary number of names can be bound. Second, the **case** operator takes an arbitary number of pairs that consist of one condition and one agent for each conditional branch.

To circumvent these problems, we create a mutually recursive nominal data type for agents. This data type is parametrised over three type variables $\alpha$, $\beta$, and $\gamma$, for terms, assertions and conditions respectively.

**Definition 4** (Agents)

$$
\begin{array}{l}
\textbf{nominal\_datatype}\ (\alpha, \beta, \gamma)\ psi = \\
\quad PsiNil \\
\quad \mid Output\ \alpha\ \alpha\ ((\alpha, \beta, \gamma)\ psi) \\
\quad \mid Input\ \alpha\ ((\alpha, \beta, \gamma)\ psiInput) \\
\quad \mid Case\ ((\alpha, \beta, \gamma)\ psiCase) \\
\quad \mid Par\ ((\alpha, \beta, \gamma)\ psi)\ ((\alpha, \beta, \gamma)\ psi) \\
\quad \mid Res\ \ll name \gg\ ((\alpha, \beta, \gamma)\ psi) \\
\quad \mid Assert\ \beta \\
\quad \mid Bang\ (\alpha, \beta, \gamma)\ psi \\
\textbf{and}\ (\alpha, \beta, \gamma)\ psiInput = \\
\quad Trm\ \alpha\ ((\alpha, \beta, \gamma)\ psi) \\
\quad \mid Bind\ \ll name \gg\ ((\alpha, \beta, \gamma)\ psiInput) \\
\textbf{and}\ (\alpha, \beta, \gamma)\ psiCase = \\
\quad EmptyCase \\
\quad \mid Cond\ \gamma\ ((\alpha, \beta, \gamma)\ psi)\ ((\alpha, \beta, \gamma)\ psiCase)
\end{array}
$$

We use the notation introduced in the informal grammar above as syntactic sugar for the corresponding constructors of this data type: e.g., $(\!|\Psi|\!)$ is short for $Assert\ \Psi$.

While this data type correctly formalises agents in psi-calculi, it is not convenient to work with. It is much more elegant to reason about input forms and case constructs in such a way that the mutually recursive structure of the *psi* data type becomes transparent. To accomplish this, we define the following wrapper functions.

**Definition 5** (inputChain)

$$
\begin{array}{lcl}
inputChain :: name\ list \Rightarrow \alpha \Rightarrow (\alpha,\ \beta,\ \gamma)\ psi \Rightarrow (\alpha,\ \beta,\ \gamma)\ psiInput \\
inputChain\ \varepsilon\ N\ P & = & Trm\ N\ P \\
inputChain\ (x\widetilde{x})\ N\ P & = & Bind\ x\ inputChain\ \widetilde{x}\ N\ P
\end{array}
$$

**Definition 6** (psiCases)

$$
\begin{array}{lcl}
psiCases :: (\gamma \times (\alpha,\ \beta,\ \gamma)\ psi)\ list \Rightarrow (\alpha,\ \beta,\ \gamma)\ psiCase \\
psiCases\ \varepsilon & = & EmptyCase \\
psiCases\ ((\varphi, P)\widetilde{C}) & = & Cond\ \varphi\ P\ psiCases\ \widetilde{C}
\end{array}
$$

We write $\underline{M}(\lambda\widetilde{x})N.P$ for $Input\ M\ (inputChain\ \widetilde{x}\ N\ P)$, and use $Cases\ \widetilde{C}$ as a shorthand for $Case\ (psiCases\ \widetilde{C})$.

### 4.3 Substitution

When the input form $\underline{M}(\lambda\widetilde{x})N.P$ receives an instance of the pattern $(\lambda\widetilde{x})N$, it continues as the agent $P$ with names in $\widetilde{x}$ instantiated accordingly to match the received instance (see the operational semantics in Section 5). To define this more precisely, we require a

substitution function on agents. Substitution in psi-calculi operates in much the same way as for other calculi with binders. It propagates through the structure of agents, avoiding capture by binders, until it reaches terms, assertions and conditions.

Terms, assertions and conditions are parameters of psi-calculi, and their exact structure is unknown. Hence we cannot define how substitution acts on them, but we must require that each of these types is equipped with an appropriate substitution function. We write $X[\widetilde{x} := \widetilde{T}]$ to denote a term, assertion, or condition $X$ whose free names in $\widetilde{x}$ have been substituted with the terms in $\widetilde{T}$. Intuitively, this substitution should be *parallel*: once a name has been replaced with a term, no further substitution is performed on this term. In practice, substitution can be any function that satisfies a minimal set of constraints. To model this, we introduce the notion of substitution types.

### *4.3.1 Substitution Types*

A *substitution type* is a type $\alpha$ that is equipped with a ternary function, written $\cdot[\cdot := \cdot]$, of type $\alpha \Rightarrow \mathit{name\ list} \Rightarrow \beta\ \mathit{list} \Rightarrow \alpha$. Intuitively, this function will substitute terms (of type $\beta$) for names in terms, assertions or conditions (of type $\alpha$). Hence the types $\alpha$ and $\beta$ may be equal, but this is not required. We impose three constraints on substitution functions.

**Definition 7** (Substitution function) A function $\cdot[\cdot := \cdot]$ of type $\alpha \Rightarrow \mathit{name\ list} \Rightarrow \beta\ \mathit{list} \Rightarrow \alpha$ is a *substitution function* if it satisfies

$$p \cdot X[\widetilde{x} := \widetilde{T}] = (p \cdot X)[p \cdot \widetilde{x} := p \cdot \widetilde{T}] \qquad \textsc{SubstEqvt}$$

$$\frac{|\widetilde{x}| = |\widetilde{T}| \qquad \mathit{distinct}\ \widetilde{x} \qquad \mathit{set}\ \widetilde{x} \subseteq \mathit{supp}\ X \qquad y \sharp X[\widetilde{x} := \widetilde{T}]}{y \sharp \widetilde{T}} \qquad \textsc{SubstFresh}$$

$$\frac{|\widetilde{x}| = |\widetilde{T}| \qquad \mathit{distinctPerm}\ p \qquad \mathit{set}\ p \subseteq \mathit{set}\ \widetilde{x} \times \mathit{set}\ (p \cdot \widetilde{x}) \qquad (p \cdot \widetilde{x}) \sharp X}{X[\widetilde{x} := \widetilde{T}] = (p \cdot X)[p \cdot \widetilde{x} := \widetilde{T}]} \qquad \textsc{SubstAlpha}$$

In Isabelle, we have defined a corresponding locale [5] for substitution types, i.e., types equipped with a substitution function.

For SubstFresh and SubstAlpha, we require the vectors that are being substituted and substituted for to be of equal length. Moreover, there must be no duplicates in the name vector. As the substitution function is intended to model parallel substitution, this does not impose any serious restriction. We now discuss the three requisites in turn.

The requisite RSubstEqvt ensures that the substitution function is equivariant, as we must be able to propagate permutations over substitutions.

The requisite SubstFresh states that the substitution function may not discard names in terms that are being substituted into a substitution type: if a term is to be substituted for a name, then the result of the substitution must not have smaller support than this term. The requisite only applies when all names being substituted are in the support of the substitution type's element. This requisite is necessary to ensure that the objects of transition labels (Section 5) record all received names; otherwise we lose the principle of scope extension [9, Section 2.5].

The final requisite SUBSTALPHA is required to mimic alpha-conversions. If the bound names of an input prefix are alpha-converted, then the corresponding names of the substitution must be similarly converted. This requisite achieves this.

With a locale for substitution types in place, we can proceed to define substitution on psi-calculi agents.

### *4.3.2 Agent Substitution*

In order to define substitution for psi-calculi agents, we create a locale that imports three separate instances of the locale for substitution types: one instance each for terms, assertions and conditions, respectively. Since psi-calculi agents are defined by a mutually inductive definition, the substitution function is defined by mutual recursion.

**Definition 8** (Capture-avoiding parallel substitution for agents)

$$
\begin{array}{rrcl}
 & \mathbf{0}[\widetilde{x} := \widetilde{T}] & = & \mathbf{0} \\
 & (\overline{M}N.P)[\widetilde{x} := \widetilde{T}] & = & \overline{M[\widetilde{x} := \widetilde{T}]}N[\widetilde{x} := \widetilde{T}].P[\widetilde{x} := \widetilde{T}] \\
 & (\mathit{Input}\ M\ I)[\widetilde{x} := \widetilde{T}] & = & \mathit{Input}\ M[\widetilde{x} := \widetilde{T}]\ I[\widetilde{x} := \widetilde{T}] \\
 & (\mathit{Case}\ C)[\widetilde{x} := \widetilde{T}] & = & \mathit{Case}\ C[\widetilde{x} := \widetilde{T}] \\
 & (P \mid Q)[\widetilde{x} := \widetilde{T}] & = & P[\widetilde{x} := \widetilde{T}] \mid Q[\widetilde{x} := \widetilde{T}] \\
\mathit{If}\ y \sharp \widetilde{x}\ \mathit{and}\ y \sharp \widetilde{T}\ \mathit{then} & ((\nu y)P)[\widetilde{x} := \widetilde{T}] & = & (\nu y)P[\widetilde{x} := \widetilde{T}] \\
 & (\!|\Psi|\!)[\widetilde{x} := \widetilde{T}] & = & (\!|\Psi[\widetilde{x} := \widetilde{T}]|\!) \\
 & (!P)[\widetilde{x} := \widetilde{T}] & = & !P[\widetilde{x} := \widetilde{T}] \\
 & & = & \\
 & (\mathit{Trm}\ M\ P)[\widetilde{x} := \widetilde{T}] & = & \mathit{Trm}\ (M[\widetilde{x} := \widetilde{T}])\ (P[\widetilde{x} := \widetilde{T}]) \\
\mathit{If}\ y \sharp \widetilde{x}\ \mathit{and}\ y \sharp \widetilde{T}\ \mathit{then} & (\mathit{Bind}\ y\ I)[\widetilde{x} := \widetilde{T}] & = & \mathit{Bind}\ y\ I[\widetilde{x} := \widetilde{T}] \\
 & & = & \\
 & \mathit{EmptyCase}[\widetilde{x} := \widetilde{T}] & = & \mathit{EmptyCase} \\
 & (\mathit{Cond}\ \Psi\ P\ C)[\widetilde{x} := \widetilde{T}] & = & \mathit{Cond}\ \Psi[\widetilde{x} := \widetilde{T}]\ P[\widetilde{x} := \widetilde{T}]\ C[\widetilde{x} := \widetilde{T}]
\end{array}
$$

As in the pi-calculus, substitution propagates through the structure of agents, avoiding capture by binders. When it reaches terms, assertions or conditions, the appropriate substitution function for these (which, in a slight abuse of notation, we write in the same way above) is applied.

Hence, the psi-calculi framework is parametric in substitution functions for terms, assertions and conditions; given these, substitution for agents is defined explicitly. We have shown that substitution for agents satisfies RSUBSTEQVT and SUBSTALPHA, provided the given substitution functions for terms, assertions and conditions do. The SUBSTFRESH property, on the other hand, is only needed for term substitution.

**Lemma 7**

$$p \cdot P[\widetilde{x} := \widetilde{T}] = (p \cdot P)[p \cdot \widetilde{x} := p \cdot \widetilde{T}]$$

$$\frac{|\widetilde{x}| = |\widetilde{T}| \qquad \mathit{set}\ p \subseteq \mathit{set}\ \widetilde{x} \times \mathit{set}(p \cdot \widetilde{x}) \qquad \mathit{distinctPerm}\ p \qquad (p \cdot \widetilde{x}) \# P}{P[\widetilde{x} := \widetilde{T}] = (p \cdot P)[p \cdot \widetilde{x} := \widetilde{T}]}$$

*Proof* By simultaneous induction over agents, input forms, and case constructs, using the corresponding properties for substitution on terms, assertions, and conditions. □

### 4.4 Nominal Operators

In addition to the type parameters and substitution functions for terms, assertions and conditions, psi-calculi are also parametric in the following equivariant operators.

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

The binary operators will be written in infix. Thus, if $M$ and $N$ are terms then $M \dot{\leftrightarrow} N$ is a condition, pronounced "$M$ and $N$ are channel equivalent," and if $\Psi$ and $\Psi'$ are assertions then so is $\Psi \otimes \Psi'$. Moreover, we write $\Psi \vdash \varphi$, pronounced "$\Psi$ entails $\varphi$," for $(\Psi, \varphi) \in \vdash$.

Similarly to the pi-calculus, data terms in psi-calculi represent all kinds of data, including communication channels. Intuitively, two agents can communicate if one sends and the other receives along the same channel. This is why we require a condition $M \dot{\leftrightarrow} N$ to say that $M$ and $N$ represent the same channel. Channel equivalence generalises the pi-calculus, where $\dot{\leftrightarrow}$ is just identity of names.

Assertions declare information that is used to resolve the conditions in case constructs. Assertions may be contained in agents and represent constraints; they may contain names and thereby be syntactically scoped, representing information known only to the agents within that scope. The operator $\otimes$ on assertions will, intuitively, represent conjunction of the information in two assertions. The assertion $\mathbf{1}$ is a unit for $\otimes$. Entailment $\Psi \vdash \varphi$ intuitively means that given the information in $\Psi$, it is possible to infer the condition $\varphi$.

We say that an assertion $\Psi$ *implies* an assertion $\Psi'$, written $\Psi \leq \Psi'$, if any condition $\varphi$ that is entailed by $\Psi$ is also entailed by $\Psi'$. We say that $\Psi$ and $\Psi'$ are *equivalent*, written $\Psi \simeq \Psi'$, if they imply each other.

**Definition 9** (Assertion implication and equivalence)

$$\begin{array}{lcl} \Psi \leq \Psi' & \equiv & \forall \varphi.\, \Psi \vdash \varphi \longrightarrow \Psi' \vdash \varphi \\ \Psi \simeq \Psi' & \equiv & \Psi \leq \Psi' \wedge \Psi' \leq \Psi \end{array}$$

We require these operators to satisfy a minimal set of properties. More precisely, equivalence must be a commutative compositional monoid, with assertions $\mathbf{A}$ as the carrier, $\mathbf{1}$ as its unit element, and $\otimes$ as the join operator. Channel equivalence must be symmetric and transitive. It need not be reflexive; this allows psi-calculi to have terms that cannot be used as communication channels.

**Definition 10** (Requirements on static equivalence)

| | |
|---|---|
| *If* $\Psi \vdash (M \dot{\leftrightarrow} N)$ *then* $\Psi \vdash (N \dot{\leftrightarrow} M)$. | CESYM |
| *If* $\Psi \vdash (M \dot{\leftrightarrow} N)$ *and* $\Psi \vdash (N \dot{\leftrightarrow} L)$ *then* $\Psi \vdash (N \dot{\leftrightarrow} L)$. | CETRANS |
| *If* $\Psi \simeq \Psi'$ *then* $\Psi \otimes \Psi'' \simeq \Psi' \otimes \Psi''$. | ACOMP |
| $\Psi \otimes \mathbf{1} \simeq \Psi$ | AID |
| $\Psi \otimes \Psi' \simeq \Psi' \otimes \Psi$ | ACOMM |
| $(\Psi \otimes \Psi') \otimes \Psi'' \simeq \Psi \otimes (\Psi' \otimes \Psi'')$ | AASSOC |

In Isabelle, we have defined a corresponding locale that imposes these requirements.

### 4.5 Summary

In the next section we will define the operational semantics of psi-calculi. For now, let us briefly recapitulate the parameters that are required for a valid psi-calculus instance, and the constraints that we impose on these parameters.

First, we require three nominal data types **T**, **A** and **C**, for terms, assertions and conditions respectively. These must be substitution types, i.e., types equipped with substitution functions. Second, we require a symmetric and transitive nominal operator for channel equivalence ($\dot{\leftrightarrow}$). Third, for assertions we require a composition operator ($\otimes$), a unit element (**1**) and an entailment relation ($\vdash$) such that assertion equivalence ($\simeq$) forms a commutative compositional monoid.

## 5 Operational Semantics

The complexity of the operational semantics of psi-calculi is on par with the standard pi-calculus semantics. Proofs of meta-theoretic properties, however, are more complicated. The main reason for this is the possible interplay of agents in a parallel composition $P|Q$. In the standard pi-calculus, the transitions from a parallel composition can be uniquely determined by the transitions from its components. In psi-calculi the situation is more complex. Here the assertions contained in $P$ can affect the conditions tested in $Q$ and vice versa. For this reason we introduce the notion of the *frame* of an agent, similar to the ones introduced by Abadi and Fournet [1], as the combination of an agent's top-level assertions, retaining all binders.

### 5.1 Frames

The frame of an agent represents the information that the agent exposes to the environment via its assertions. These can contain information about names, and names can be scoped using the $\nu$-binder. For instance, in a cryptographic application an assertion $\Psi$ could be that a datum represents the encoding of a message using a key $k$. This assertion can occur under the scope of $\nu k$, to signify that the key is known only locally. We write $(\nu k)\Psi$ do denote a frame consisting of the assertion $\Psi$ where the name $k$ is local.

In the general case, a frame is of the form $(\nu\tilde{b})\Psi$, where $\tilde{b}$ is a sequence of names that bind into the assertion $\Psi$. Frames are defined in Isabelle in the following manner.

**Definition 11** (Frames)

$$\begin{aligned}&\textbf{nominal\_datatype}\ \beta\ \mathit{frame} =\\ &\quad \mathit{FAssert}\ \beta\\ &\mid \mathit{FRes}\ \ll \mathit{name} \gg (\beta\ \mathit{frame})\end{aligned}$$

We use $F$, $G$ to range over frames. $(\nu\varepsilon)\Psi$ is short for *FAssert* $\Psi$, and $(\nu x)F$ means $FRes\ x\ F$. We write just $\Psi$ for $(\nu\varepsilon)\Psi$ when there is no risk of confusing a frame with an assertion.

As for input forms, we bind lists of names to a frame by recursing over the list.

**Definition 12** (frameResChain)

$$\begin{array}{lcl} \mathit{frameResChain} :: \mathit{name\ list} \Rightarrow \beta\ \mathit{frame} \Rightarrow \beta\ \mathit{frame} \\ \mathit{frameResChain}\ \varepsilon\ F & = & F \\ \mathit{frameResChain}\ (x\widetilde{x})F & = & (\nu x)\mathit{frameResChain}\widetilde{x}F \end{array}$$

We use $(\nu\widetilde{x})F$ as syntactic sugar for $\mathit{frameResChain}\ \widetilde{x}\ F$.

### *5.1.1 Frame Composition*

When two agents run in parallel, their frames are composed. We overload the $\otimes$-operator to also compose frames with assertions and frames with frames.

**Definition 13** (Composing frames with assertions) A frame $F$ composed with an assertion $\Psi$ is written $F \otimes \Psi$.

$$\begin{array}{rcl} ((\nu\varepsilon)\Psi) \otimes \Psi' & = & (\nu\varepsilon)(\Psi' \otimes \Psi) \\ \mathit{If}\ x \# \Psi'\ \mathit{then}\ ((\nu x)F) \otimes \Psi' & = & (\nu x)(F \otimes \Psi') \end{array}$$

**Definition 14** (Composing frames with frames) A frame $F$ composed with a frame $G$ is written $F \otimes G$.

$$\begin{array}{rcl} ((\nu\varepsilon)\Psi) \otimes G & = & G \otimes \Psi \\ \mathit{If}\ x \# G\ \mathit{then}\ ((\nu x)F) \otimes G & = & (\nu x)(F \otimes G) \end{array}$$

The following lemma is used to propagate binders in a composition to the outermost level.

**Lemma 8**

$$\frac{\widetilde{b}_F \# \Psi}{((\nu\widetilde{b}_F)\Psi_F) \otimes \Psi = (\nu\widetilde{b}_F)(\Psi \otimes \Psi_F)} \qquad \frac{\widetilde{b}_F \# \widetilde{b}_G \quad \widetilde{b}_F \# \Psi_G \quad \widetilde{b}_G \# \Psi_F}{((\nu\widetilde{b}_F)\Psi_F) \otimes ((\nu\widetilde{b}_G)\Psi_G) = (\nu\widetilde{b}_F\widetilde{b}_G)(\Psi_F \otimes \Psi_G)}$$

*Proof* By induction on $\widetilde{b}_F$ for the first case, and on $\widetilde{b}_G$ for the second. □

### *5.1.2 Frame of an agent*

The *frame of an agent*, written $\mathscr{F}\ P$, is the collection of assertions of $P$ that are not guarded by an input or output prefix, where all binders are retained. It is defined by recursion over the agent as follows.

**Definition 15** (Frame of an agent)

$$\begin{aligned}
\mathcal{F}\ \mathbf{0} &= \mathbf{1} \\
\mathcal{F}\ (\mathit{Input}\ M\ I) &= \mathbf{1} \\
\mathcal{F}\ (\overline{M}N.P) &= \mathbf{1} \\
\mathcal{F}\ (\mathit{Case}\ C) &= \mathbf{1} \\
\mathcal{F}\ (P \mid Q) &= \mathcal{F}\ P \otimes \mathcal{F}\ Q \\
\mathcal{F}\ (\!(\Psi)\!) &= (\nu\varepsilon)\Psi \\
\mathcal{F}\ ((\nu x)P) &= (\nu x)(\mathcal{F}\ P) \\
\mathcal{F}\ (!P) &= \mathbf{1}
\end{aligned}$$

For a simple example, if $a \# \Psi$, we have

$$\mathcal{F}\ (\!(\Psi_1)\!) \mid (\nu a)(\!(\Psi_2)\!) \mid \overline{M}N.(\!(\Psi_3)\!)) = (\nu a)(\Psi_1 \otimes \Psi_2)$$

Here, $\Psi_3$ occurs under a prefix and is therefore not included in the frame of the agent.

We often write $(\nu \tilde{b}_p)\Psi_p$ for $\mathcal{F}\ P$, but note that this is not a unique representation since frames are identified up to alpha-equivalence.

### *5.1.3 Frame Entailment and Equivalence*

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

**Definition 16** (Frame entailment and equivalence) For a frame $F$ and a condition $\varphi$, we define $F \vdash \varphi$ to mean that there exists an alpha-variant $F = (\nu \widetilde{b}_F)\Psi_F$ such that $\widetilde{b}_F \# \varphi$ and $\Psi_F \vdash \varphi$, i.e.,

$$F \vdash \varphi \quad \equiv \quad \exists \widetilde{b}_F \Psi_F . F = (\nu \widetilde{b}_F)\Psi_F \wedge \widetilde{b}_F \# \varphi \wedge \Psi_F \vdash \varphi$$

For two frames $F$ and $G$, we define

$$F \simeq G \quad \equiv \quad \forall \varphi . F \vdash \varphi \longleftrightarrow G \vdash \varphi$$

For instance, $(\nu ab)\Psi \simeq (\nu ba)\Psi$, and if $a \# \Psi$ then $(\nu a)\Psi \simeq \Psi$.

To take an example of first-order logic with equality, assume that the term $\mathsf{enc}(M, k)$ represents the encryption of message $M$ with key $k$. Let $\Psi$ be the assertion $C = \mathsf{enc}(M, k)$, stating that the ciphertext $C$ is the result of encrypting $M$ with $k$. If an agent contains this assertion, the environment of the agent will be able to use it to resolve tests on the data, in particular to infer that $C = \mathsf{enc}(M, k)$. In other words, if the environment receives $C$ it can test if this is the encryption of $M$. In order to restrict access to the key, $k$ can be enclosed in a scope $\nu k$. The environment of the agent will then have access to the frame $(\nu k)\Psi$ rather than $\Psi$ itself. This frame is much less informative, for example it does *not* hold that $(\nu k)\Psi \vdash C = \mathsf{enc}(M, k)$. Here great care has to be taken to formulate the class of allowed conditions. If these only contain equivalence tests of terms, $(\nu k)\Psi$ will entail nothing but tautologies and be equivalent to $\mathbf{1}$. But if quantifiers are allowed in the conditions, then by existential introduction $\Psi \vdash \exists k.(C = \mathsf{enc}(M, k))$, and since $k$ is not free in this condition we obtain $(\nu k)\Psi \vdash \exists k.(C = \mathsf{enc}(M, k))$. In other words, the environment will learn that $C$ is the encryption of $M$ for some key $k$.

Most of the properties of entailment carry over from assertions to frames. Channel equivalence is again symmetric and transitive. Frame composition is associative and commutative, the frame $\mathbf{1}$ being a unit. However, compositionality need not hold. In other words, there are psi-calculi with frames $F$, $G$, $H$ where $F \simeq G$ but not $F \otimes H \simeq G \otimes H$. An example is if there are assertions $\Psi$, $\Psi'$ and $\Psi_a$ for all names $a$, conditions $\varphi'$ and $\varphi_a$ for all names $a$, and where the entailment relation satisfies $\Psi_a \vdash \varphi_a$ and $\Psi' \vdash \varphi'$. Suppose composition is defined such that $\Psi \otimes \Psi = \Psi$ and all other compositions yield $\Psi'$. By adding a unit element this satisfies all requirements on a psi-calculus. In particular $\otimes$ is trivially compositional because no two different assertions are equivalent. Also $(\nu a)\Psi_a \simeq \Psi$, but $\Psi \otimes (\nu a)\Psi_a \not\simeq \Psi \otimes \Psi$ since $\Psi \otimes \Psi_a = \Psi' \vdash \varphi'$.

## 5.2 Actions

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

**Definition 17** (Actions)

$$\begin{array}{l}\textbf{nominal\_datatype}\ \alpha\ action = \\ \quad In\ \alpha\ \alpha \\ \mid Out\ \alpha\ (name\ list)\ \alpha \\ \mid Tau\end{array}$$

We write $\underline{M}N$ for $In\ M\ N$, $\overline{M}\ (\nu\widetilde{x})N$ for $Out\ M\ \widetilde{x}\ N$ and $\tau$ for $Tau$. We use $\alpha$, $\beta$ to range over actions.

For input and output actions, we refer to $M$ as the *subject* and $N$ as the *object*. As in the pi-calculus, the output $\overline{M}(\nu\widetilde{x})N$ represents an action sending $N$ along the channel $M$ and opening the scopes of the names $\widetilde{x}$. Note in particular that the support of this action includes $\widetilde{x}$, so that, for instance, $\overline{M}(\nu a)a$ and $\overline{M}(\nu b)b$ are different actions. Nonetheless, for reasons that will become apparent in the following section, we refer to the names $\widetilde{x}$ in an output action as its *bound names*.

**Definition 18** (subject, object, bn)

$$\begin{array}{llll} subject :: \alpha\ action \Rightarrow \alpha\ option & & object :: \alpha\ action \Rightarrow \alpha\ option & \\ subject\ (\underline{M}N) & = Some\ M & object\ (\underline{M}N) & = Some\ N \\ subject\ (\overline{M}\ (\nu\widetilde{x})N) & = Some\ M & object\ (\overline{M}(\nu\widetilde{x})N) & = Some\ N \\ subject\ (\tau) & = None & object\ (\tau) & = None \\ bn :: \alpha\ action \Rightarrow name\ list & & & \\ bn\ (\underline{M}N) & = \varepsilon & & \\ bn\ (\overline{M}(\nu\widetilde{x})N) & = \widetilde{x} & & \\ bn\ (\tau) & = \varepsilon & & \end{array}$$

## 5.3 Residuals

The operational semantics of psi-calculi consists of transitions of the form

$$\Psi \rhd P \xrightarrow{\alpha} P'$$

This transition intuitively means that in an environment that asserts $\Psi$, the agent $P$ can perform action $\alpha$, thereby becoming $P'$.

A first attempt to encode transitions, which works well for simpler calculi like CCS [10], is to define the operational semantics as an inductive predicate with four arguments: an environment $\Psi$, a process $P$, an action $\alpha$ and a derivative $P'$. However, previous mechanisations of both the pi-calculus and of psi-calculi have shown that this approach is impractical here. The key issue is that the action $\alpha$ may bind names, and these names bind not only in $\alpha$ but also into the derivative $P'$.

This observation was made already in the original presentation of the pi-calculus, where lemmas concerning variants of transitions are spelled out [38]. In his tutorial on the polyadic pi-calculus [37], Milner therefore uses "commitments" rather than labelled transitions.

In many presentations of the pi-calculus the issue is glossed over, and if $\alpha$-conversions are not defined rigorously the four-argument syntax for transitions works fine. But here it poses a problem: it would require us to explicitly state the rules for changing bound names, and we would not be able to rely on the otherwise smooth treatment of alpha-variants in the Nominal Isabelle framework.

Therefore, in our implementation we follow Milner [37], with a slight change of notation to avoid confusion of prefixes and commitments, and define a *residual* data type that contains both action and derivative. It binds the bound names of an action also in the derivative. We used a similar technique in our mechanisation of the pi-calculus [10], but for psi-calculi we must additionally take into account that an action can contain more than one bound name. We first define a nominal data type containing an object, a derivative and a sequence of names binding into both.

**Definition 19** (boundOutput)

$$\begin{array}{l}\textbf{nominal\_datatype}\ (\alpha,\ \beta,\ \gamma)\ \mathit{boundOutput} = \\ \quad \mathit{BOut}\ \alpha\ ((\alpha,\ \beta,\ \gamma)\ \mathit{psi}) \\ \mid \mathit{BStep}\ \ll \mathit{name} \gg ((\alpha,\ \beta,\ \gamma)\ \mathit{boundOutput})\end{array}$$

Binding sequences are obtained as usual, by recursively binding names in the nominal data type.

**Definition 20** (BOresChain)

$$\begin{array}{l}\mathit{BOresChain} :: \mathit{namelist} \Rightarrow (\alpha,\ \beta,\ \gamma)\ \mathit{boundOutput} \Rightarrow (\alpha,\ \beta,\ \gamma)\ \mathit{boundOutput} \\ \mathit{BOresChain}\ \varepsilon\ B = B \\ \mathit{BOresChain}\ x\widetilde{x}\ B = \mathit{BStep}\ x\ (\mathit{BOresChain}\ \widetilde{x}\ B)\end{array}$$

We can now define residuals. Just like psi-calculi agents, the data type of residuals is parametrised over three type variables $\alpha$, $\beta$, $\gamma$ for terms, assertion and conditions respectively.

**Definition 21** (Residuals)

$$\begin{array}{l}\textbf{nominal\_datatype}\ (\alpha,\ \beta,\ \gamma)\ \mathit{residual} = \\ \quad \mathit{RIn}\ \alpha\ \alpha\ ((\alpha,\ \beta,\ \gamma)\ \mathit{psi}) \\ \mid \mathit{ROut}\ \alpha\ ((\alpha,\ \beta,\ \gamma)\ \mathit{boundOutput}) \\ \mid \mathit{RTau}\ ((\alpha,\ \beta,\ \gamma)\ \mathit{psi})\end{array}$$

We use $V$ and $W$ to range over residuals. Having defined residuals, we face a discrepancy between the more traditional syntax for transitions (which suggests that they are a quarternary relation) and their intended semantics (where action and derivative are one construct, and names in the action bind into the derivative). One drawback is that we cannot define a function that extracts the bound names of a residual in nominal logic, since these are not invariant under alpha-conversion. To reconcile the two views, we define an infix function $\prec$ that creates a residual from an action and an agent.

**Definition 22** ($\prec$)

$$\begin{array}{rcl} \prec :: \alpha\ action \Rightarrow (\alpha,\ \beta,\ \gamma)\ psi \Rightarrow (\alpha,\ \beta,\ \gamma)\ residual & & \\ M\ N\ \prec\ P & = & RIn\ M\ N\ P \\ \overline{M}(\nu\widetilde{x})N\ \prec\ P & = & ROut\ M\ (BOresChain\ \widetilde{x}\ (BOut\ N\ P)) \\ \tau\ \prec\ P & = & RTau\ P \end{array}$$

We use the notation $\Psi \rhd P \xrightarrow{\alpha} P'$ to mean $\Psi \rhd P \longmapsto \alpha \prec P'$, where $\cdot \rhd \cdot \longmapsto \cdot$ is a ternary relation. By modeling transitions in this manner we get the best of two worlds. As required, the bound names of output actions bind into derivatives in the residual. But we can still determine which binders are used in an action, and what the objects and agents under their scope are. This allows us to impose conditions on the binders in labels; for instance, that they must be sufficiently fresh.

### *5.3.1 Alpha-Equivalence*

Dealing with alpha-equivalence of residuals is not entirely straightforward. What does it mean for two residuals $\alpha \prec P$ and $\beta \prec Q$ to be equivalent? The subjects are not under the scope of the binders, so clearly they must be equal. But the object and the derivatives are under the scope of the bound names of $\alpha$ and $\beta$. Hence they are not necessarily syntactically equal, but alpha-equivalent. Moreover, we do not want to resort to case analysis every time an equality between two residuals appears in a proof state: the very point of the residual construction is to avoid separate cases for actions with bound names and for those without. The following lemma, which is similar in spirit to Lemma 4, obtains an alpha-converting permutation that equates two residuals.

**Lemma 9**

$$\frac{\begin{array}{ccc} \alpha \prec P = \beta \prec Q & distinct\ (bn\ \alpha) & distinct\ (bn\ \beta) \\ bn\ \alpha\ \sharp\ bn\ \beta & bn\ \alpha\ \sharp\ \alpha \prec P & bn\ \beta\ \sharp\ \beta \prec Q \end{array}}{\begin{array}{c} \exists p.\ set\ p \subseteq set\ (bn\ \alpha) \times set\ (bn\ (p \cdot \alpha))\ \wedge\ \beta = p \cdot \alpha\ \wedge\ Q = p \cdot P \wedge \\ bn\ \alpha\ \sharp\ \beta\ \wedge\ bn\ \alpha\ \sharp\ Q\ \wedge\ bn\ (p \cdot \alpha)\ \sharp\ \alpha\ \wedge\ bn\ (p \cdot \alpha)\ \sharp\ P \end{array}}$$

*Proof* Given two residuals with disjoint bound names, an alpha-converting permutation $p$ is constructed by pairing corresponding bound names together. The requirements $bn\,\alpha\,\#\,\alpha \prec P$ and $bn\,\beta\,\#\,\beta \prec Q$ ensure that bound names do not occur outside their scope; thus, $p$ leaves the subjects of both residuals unaffected. □

We also prove an alpha-conversion lemma for residuals. As when alpha-converting binding sequences (Section 3.2), the permutation applied to the sequence must be fresh for

everything under the scope of the binder. Moreover, for the same reason as in the previous lemma, neither the original nor the new bound names may occur in the subject of the action.

**Lemma 10**

$$\frac{bn\ \alpha\ \sharp\ subject\ \alpha \qquad \begin{array}{c} bn\ (p \cdot \alpha)\ \sharp\ object\ \alpha \\ bn\ (p \cdot \alpha)\ \sharp\ subject\ \alpha \end{array} \qquad \begin{array}{c} bn\ (p \cdot \alpha)\ \sharp\ P \\ set\ p \subseteq set\ (bn\ \alpha) \times set\ (bn\ (p \cdot \alpha)) \end{array}}{\alpha \prec P = (p \cdot \alpha) \prec (p \cdot P)}$$

*Proof* By case analysis on $\alpha$. When $\alpha = \tau$ or $\alpha = \underline{M}N$ the permutation must be empty as neither action has bound names. In case $\alpha = \overline{M}(\nu \widetilde{x})N$ the permutation will cancel out from the subject $M$ as no names in $p$ occur in $M$. The residual is then alpha-converted to finish the proof. □

## 5.4 Operational Semantics

A standard way to model operational semantics is to use inductively defined predicates. For calculi without binders, this is relatively straightforward, and Isabelle generates both induction and inversion (elimination) rules automatically [53].

For calculi with binders, things are not as straightforward. One of the main achievements of the Nominal Isabelle framework is its treatment of rule induction. More precisely, how it makes formal the Barendregt variable convention, allowing us to pick an arbitrary context of names that all bound names will be fresh for when we carry out induction over transitions. As mentioned in Section 2.1, the Barendregt variable convention is unsound in the general case. Urban et al. [49] identified a *variable convention compatibility condition* that details exactly what is required of bound names for the variable convention to be sound.

One straightforward way to satisfy this property is to require that all bound names are *sufficiently fresh*, i.e., fresh for everything outside their scope. This has the additional benefit that when we carry out induction over the transition system, the bound names of each rule are by default fresh for everything outside their scope that is mentioned in the rule.

The labelled operational semantics of psi-calculi, describing the transitions that psi-calculi agents can take, is defined inductively, and shown in Fig. 1. Parallel agents affect each other through their frames [9].

For the remainder of this section, we demonstrate how we encode the operational semantics in Nominal Isabelle, and how we develop the standard induction rules as well as some custom ones. We discuss the PAR rule as an example. Our techniques apply equally to the other rules of the semantics. Shown in full, the PAR rule has the following form.

$$\frac{\mathcal{F}\ Q = (\nu \widetilde{b_Q})\Psi_Q \qquad \begin{array}{c} \Psi \otimes \Psi_Q \rhd P \xrightarrow{\alpha} P' \\ bn\ \alpha\ \sharp\ Q \qquad \widetilde{b_Q}\ \sharp\ \Psi \qquad \widetilde{b_Q}\ \sharp\ P \qquad \widetilde{b_Q}\ \sharp\ \alpha \end{array}}{\Psi \rhd P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \text{PAR}$$

For the operational semantics, we would like to obtain a stronger induction principle than the one afforded by this rule, namely one where the bound names $\widetilde{b_Q}$ and $bn\,\alpha$ are both distinct and fresh for everything outside their scope. Therefore, we use the following rule instead of PAR to define the operational semantics.

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

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

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

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

**Fig. 1** Operational semantics. Symmetric versions of COM and PAR are elided. In the rule COM we assume that $\mathscr{F}\ P = (\nu\tilde{b}_P)\Psi_P$ and $\mathscr{F}\ Q(\nu\tilde{b}_Q)\Psi_Q =$, where $\tilde{b}_P$ is fresh for all of $\Psi, \tilde{b}_Q, Q, M$ and $P$, and that $\tilde{b}_Q$ is correspondingly fresh. In the rule PAR we assume that $\mathscr{F}\ Q = (\nu\tilde{b}_Q)\Psi_Q$, where $\tilde{b}_Q$ is fresh for $\Psi$, $P$ and $\alpha$. In OPEN the expression $\tilde{a} \cup \{b\}$ means the sequence $\tilde{a}$ with $b$ inserted anywhere. This figure is taken from [9]

$$\frac{\begin{array}{c} \Psi \otimes \Psi_Q \rhd P \xrightarrow{\alpha} P' \qquad \mathscr{F}\ Q = (\nu\widetilde{b_Q})\Psi_Q \qquad \mathit{distinct}\ \widetilde{b}_Q \\ \widetilde{b}_Q \sharp P \qquad \widetilde{b}_Q \sharp Q \qquad \widetilde{b}_Q \sharp \Psi \qquad \widetilde{b}_Q \sharp \alpha \qquad \widetilde{b}_Q \sharp P' \\ \mathit{distinct}\ (\mathit{bn}\ \alpha) \qquad \mathit{bn}\ \alpha \sharp \mathit{subject}\ \alpha \\ \mathit{bn}\ \alpha \sharp \Psi \qquad \mathit{bn}\ \alpha \sharp \Psi_Q \qquad \mathit{bn}\ \alpha \sharp Q \qquad \mathit{bn}\ \alpha \sharp P \end{array}}{\Psi \rhd P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \textsc{ParS}$$

While the PARS rule provides convenient freshness conditions for proofs that perform induction on the transition system, the same freshness conditions make the PARS rule tedious to use on its own, i.e., to derive transitions for a parallel composition. To circumvent this problem, we derive the PAR rule from the PARS rule.

**Theorem 1** PAR *is a valid rule.*

*Proof* We first alpha-convert $\widetilde{b}_Q$ and $bn\ \alpha$ to be sufficiently fresh. Then Lemma 5 is used twice to ensure that both $\widetilde{b}_Q$ and $bn\ \alpha$ are distinct. □

The induction rule generated by Isabelle for the operational semantics is shown in Fig. 2. Its selling characteristic is that it allows all inductive proofs that use this rule to provide a freshness context $\mathscr{C}$ for which any bound names introduced by the rule are fresh. This greatly reduces the tedium of manual alpha-conversions that would have to be done otherwise.

### 5.5 Action Induction Rules

The induction rule from Fig. 2 works well only as long as the property to be proven does not depend on anything under the scope of a binder. Trying to prove the following statement illustrates the problem.

$$\mathit{If}\ \Psi \rhd P \xrightarrow{\overline{M}(\nu\widetilde{x})N} P'\ \mathit{and}\ z \# P\ \mathit{and}\ z \# \widetilde{x}\ \mathit{then}\ z \# N\ \mathit{and}\ z \# P'.$$

This statement is provable by induction on $\Psi \rhd P \xrightarrow{\overline{M}(\nu\widetilde{x})N} P'$. However, the induction rule in Fig. 2 will not prove it in a satisfactory way. Every applicable case in the induction rule will introduce its own residual $\overline{K}(\nu y)L \prec P''$ for which we know that $\overline{K}(\nu y)L \prec$

$$\left[\begin{array}{c} \Psi \rhd R \longmapsto W \\ \vdots \\ \dfrac{\left(\begin{array}{c} \Psi' \otimes \Psi_Q \rhd P \xrightarrow{\alpha} P' \\ \forall \mathscr{C}.\, Prop\ \mathscr{C}\ (\Psi' \otimes \Psi_Q)\ P\ (\alpha \prec P') \qquad \mathscr{F}\ Q = (\nu A_Q)\Psi_Q \qquad distinct\ A_Q \\ \widetilde{b}_Q \sharp \mathscr{C}' \qquad A_Q \sharp P \qquad A_Q \sharp Q \qquad A_Q \sharp \Psi' \qquad A_Q \sharp \alpha \qquad A_Q \sharp P' \\ distinct\ (bn\ \alpha) \qquad bn\ \alpha \sharp subject\ \alpha \\ bn\ \alpha \sharp \Psi' \qquad bn\ \alpha \sharp \Psi_Q \qquad bn\ \alpha \sharp Q \qquad bn\ \alpha \sharp P \end{array}\right)}{Prop\ \mathscr{C}'\ \Psi'\ (P \mid Q)\ (\alpha \prec P' \mid Q)}\ \textsc{Par} \\ \vdots \end{array}\right]$$

$$\overline{Prop\ \mathscr{C}\ \Psi\ R\ W}$$

**Fig. 2** Induction rule for the operational semantics of psi-calculi. The inductive cases share the name of the semantic rule from which they are derived. Only the PAR case is shown in detail. For space reasons, meta quantifiers have been suppressed—every term of every case is locally universally quantified

$P'' = \overline{M}(\nu \widetilde{x})N \prec P'$. What we need to prove relates to the term $P'$; what the inductive hypotheses will give us is related to the term $P''$, where all we know is that $P'$ and $P''$ are part of alpha-equivalent terms.

Proving the above statement is still possible with this induction rule, but in every step of every proof of this type, manual alpha-conversions and equivariance properties are needed. Figure 3 shows a derived induction rule that solves this problem once and for all.

$$\left[\begin{array}{c} \dfrac{\Psi \rhd R \xrightarrow{\alpha} R' \qquad bn\ \alpha \sharp subject\ \alpha \qquad distinct\ (bn\ \alpha) \qquad \left(\begin{array}{c} bn\ \alpha' \sharp \Psi' \qquad bn\ \alpha' \sharp P \qquad bn\ \alpha' \sharp subject\ \alpha' \qquad bn\ \alpha' \sharp \mathscr{C}' \\ bn\ \alpha' \sharp bn\ (p \cdot \alpha') \qquad set\ p \subseteq set\ (bn\ \alpha') \times set\ (bn\ (p \cdot \alpha')) \\ distinctPerm\ p \qquad bn\ (p \cdot \alpha') \sharp \alpha' \qquad bn\ (p \cdot \alpha') \sharp P' \qquad Prop\ \mathscr{C}'\ P\ \alpha'\ P' \end{array}\right)}{Prop\ \mathscr{C}'\ \Psi'\ P\ (p \cdot \alpha')\ (p \cdot P')}\ \textsc{Alpha} \\ \vdots \\ \dfrac{\left(\begin{array}{c} \Psi' \otimes \Psi_Q \rhd P \xrightarrow{\alpha'} P' \\ \forall \mathscr{C}.\, Prop\ \mathscr{C}\ (\Psi' \otimes \Psi_Q)\ P\ \alpha'\ P' \qquad \mathscr{F}\ Q = (\nu A_Q)\Psi_Q \qquad distinct\ A_Q \qquad \widetilde{b}_Q \sharp \mathscr{C}' \\ A_Q \sharp P \qquad A_Q \sharp Q \qquad A_Q \sharp \Psi' \qquad A_Q \sharp \alpha' \qquad A_Q \sharp P' \\ distinct\ (bn\ \alpha') \qquad bn\ \alpha' \sharp subject\ \alpha' \\ bn\ \alpha' \sharp \Psi' \qquad bn\ \alpha' \sharp \Psi_Q \qquad bn\ \alpha' \sharp Q \qquad bn\ \alpha' \sharp P \end{array}\right)}{Prop\ \mathscr{C}'\ \Psi'\ (P \mid Q)\ \alpha'\ (P' \mid Q)}\ \textsc{Par} \\ \vdots \end{array}\right]$$

$$\overline{Prop\ \mathscr{C}\ \Psi\ R\ \alpha\ R'}$$

**Fig. 3** Derived induction rule for transitions of the form $\Psi \rhd R \xrightarrow{\alpha} R'$. The extra ALPHA case ensures that the bound names of $\alpha$ can be freely alpha-converted in *Prop*. The inductive cases share the name of the semantic rule from which they are derived. For space reasons, meta quantifiers have been suppressed—every term of every case is locally universally quantified. To apply the rule, the requisites $bn\,\alpha \,\#\, subject\,\alpha$ and $distinct\ (bn\,\alpha)$ must be proved

In contrast to the induction rule from Fig. 2, where the property *Prop* takes a residual $W$ as its final argument, *Prop* in this rule takes an action $\alpha$ and an agent $R'$ as two separate arguments. By disassociating the action from the derivative in this manner we have lost the ability to alpha-convert the residual, but we have gained the ability to reason about terms under the scope of its binders. The extra ALPHA case in the induction rule is designed to allow *Prop* to mimic the alpha-conversion abilities that we have lost.

**Theorem 2** *The induction rule in Fig. 3 is valid.*

*Proof* We derive the induction rule in Fig. 3 from the original induction rule shown in Fig. 2. Lemma 9 is used in each step to generate the alpha-converting permutation. This lemma requires that the bound names of the transition are distinct, and do not occur free in the residual, hence in the subject of the action. These requirements can be found as two extra requisites $\mathit{distinct}\,(\mathit{bn}\,\alpha)$ and $\mathit{bn}\,\alpha\#\mathit{subject}\,\alpha$ in the derived induction rule. In every case, *Prop* is proven in the standard way, and then alpha-converted using the new inductive case ALPHA. □

With this induction rule, we must prove that the property that we are trying to establish respects alpha-conversions. The advantage is that this only has to be done once for each inductive proof. Moreover, the ALPHA case is generic; it does not require the agents or actions to be of a specific form.

We can now prove the freshness lemma that we stated earlier.

**Lemma 11** *If* $\Psi \rhd P \xrightarrow{\overline{M}(\nu\widetilde{x})N} P'$ *and* $z \# P$ *and* $z \# \widetilde{x}$ *and distinct* $\widetilde{x}$ *and* $\widetilde{x} \# M$ *then* $z \# N$ *and* $z \# P'$.

*Proof* By induction on $\Psi \rhd P \xrightarrow{\overline{M}(\nu\widetilde{x})N} P'$, using the induction rule from Fig. 3. □

The lemma illustrates a minor remaining problem with the method used to derive general induction rules: it requires the extra freshness and distinctness assumptions *distinct* $\widetilde{x}$ and $\widetilde{x} \# M$. In principle, a stronger version of the lemma without these assumptions could be proved, but they are needed to invoke the induction rule from Fig. 3.

If desired, these assumptions can later be removed by manual alpha-conversions. This is a tedious process in general, and the user has to decide for each proof whether it is worth the effort. In practice, these extra constraints are unproblematic: Nominal Isabelle is very good at ensuring that binding sequences are sufficiently fresh, and infrastructure to prove the distinctness properties is provided by our framework. In contrast, if the standard induction rule were used, manual alpha-conversions would have to be done for every inductive proof step.

### 5.6 Frame Induction Rules

A common type of proof for psi-calculi is induction over a transition where the agent has a specific frame. Trying to prove the following statement illustrates this.

$$\frac{\Psi \rhd P \xrightarrow{MN} P' \qquad \mathcal{F}\,P = (\nu\widetilde{b}_P)\Psi_P \qquad \mathit{distinct}\,\widetilde{b}_P \qquad \Psi \otimes \Psi_P \vdash M \leftrightarrow K \qquad \widetilde{b}_P \sharp \Psi \qquad \widetilde{b}_P \sharp P \qquad \widetilde{b}_P \sharp M \qquad \widetilde{b}_P \sharp K}{\Psi \rhd P \xrightarrow{KN} P'}$$

The statement asserts that an action subject $M$ can be replaced by a channel equivalent subject $K$ in an input action, where the frame of $P$ may be used to establish channel equivalence between $M$ and $K$.

The statement is provable by induction on $\Psi \rhd P \xrightarrow{MN} P'$. However, using the induction rule from Fig. 2, we suffer from a problem similar to the one discussed in the previous section: every inductive case will generate a frame alpha-equivalent to $(\nu \widetilde{b}_P)\Psi_P$, so that many tedious alpha-conversions are necessary in the proof.

To address this issue, we derive an induction rule for induction on transitions where the agent has a specific frame. This rule is shown in Fig. 4.

According to the operational semantics, the subject of a transition may well contain names that do not occur in the originating agent. However, the subject must be channel equivalent to the subject of a prefix that occurs syntactically. Intuitively, the following lemma obtains this prefix subject, which does not contain any names that are fresh for the agent and the bound names of its frame. Here, *the* is a function that retrieves the value of elements of *option* types, specified as *the* $(Some\ x) = x$.

$$\left[\begin{array}{c}
\Psi \rhd R \longmapsto W \qquad \mathscr{F}\ R = (\nu \widetilde{b}_R)\Psi_R \qquad distinct\ \widetilde{b}_R \\
\\
\dfrac{\left(\begin{array}{c}\widetilde{b}_P \sharp \Psi' \qquad \widetilde{b}_P \sharp P \qquad \widetilde{b}_P \sharp p \cdot \widetilde{b}_P \qquad \widetilde{b}_P \sharp V \qquad \widetilde{b}_P \sharp \mathscr{C}' \\ set\ p \subseteq set\ \widetilde{b}_P \times set\ (p \cdot \widetilde{b}_P) \qquad distinctPerm\ p \\ Prop\ \mathscr{C}'\ \Psi'\ P\ V\ \widetilde{b}_P\ \Psi_P\end{array}\right)}{Prop\ \mathscr{C}'\ \Psi'\ P\ V\ (p \cdot \widetilde{b}_P)\ (p \cdot \Psi_P)}\ \textsc{Alpha} \\
\\
\vdots \\
\\
\dfrac{\left(\begin{array}{c}\Psi' \otimes \Psi_Q \rhd P \xrightarrow{\alpha} P' \qquad \forall \mathscr{C}.\ Prop\ \mathscr{C}\ (\Psi' \otimes \Psi_Q)\ P\ (\alpha \prec P')\ \widetilde{b}_P\ \Psi_P \\ \\ \mathscr{F}\ P = (\nu \widetilde{b}_P)\Psi_P \qquad distinct\ \widetilde{b}_P \qquad \widetilde{b}_P \sharp \widetilde{b}_Q \qquad \widetilde{b}_P \sharp \Psi_Q \\ \widetilde{b}_P \sharp P \qquad \widetilde{b}_P \sharp Q \qquad \widetilde{b}_P \sharp \Psi' \qquad \widetilde{b}_P \sharp \alpha \qquad \widetilde{b}_P \sharp P' \qquad \widetilde{b}_P \sharp \mathscr{C}' \\ \\ \mathscr{F}\ Q = (\nu \widetilde{b}_Q)\Psi_Q \qquad distinct\ \widetilde{b}_Q \qquad \widetilde{b}_Q \sharp P \qquad \widetilde{b}_Q \sharp Q \\ \widetilde{b}_Q \sharp \Psi' \qquad \widetilde{b}_Q \sharp \alpha \qquad \widetilde{b}_Q \sharp P' \qquad \widetilde{b}_Q \sharp \Psi_P \qquad \widetilde{b}_Q \sharp \mathscr{C}' \\ \\ distinct\ (bn\ \alpha) \qquad bn\ \alpha \sharp subject\ \alpha \qquad bn\ \alpha \sharp \Psi' \\ bn\ \alpha \sharp \Psi_P \qquad bn\ \alpha \sharp \Psi_Q \qquad bn\ \alpha \sharp P \qquad bn\ \alpha \sharp Q\end{array}\right)}{Prop\ \mathscr{C}'\ \Psi'\ (P \mid Q)\ (\alpha \prec P' \mid Q)\ (\widetilde{b}_P \widetilde{b}_Q)\ (\Psi_P \otimes \Psi_Q)}\ \textsc{Par1} \\
\\
\vdots
\end{array}\right]$$

$$Prop\ \mathscr{C}\ \Psi\ R\ W\ \widetilde{b}_R\ \Psi_R$$

**Fig. 4** Derived induction rule for transitions of the form $\Psi \rhd R \longmapsto W$, where $R$ has the frame $(\nu \widetilde{b}_R)\Psi_R$. The extra ALPHA case ensures that the frame of $R$ can be alpha-converted in the predicate *Prop*. The inductive cases share the name of the semantic rule from which they are derived. For space reasons, meta quantifiers have been suppressed—every term of every case is locally universally quantified

**Lemma 12**

$$\frac{\begin{array}{cccccc}\Psi \rhd P \xrightarrow{\alpha} P' & & & & & \\ \mathcal{F}\,P = (\nu \widetilde{b}_P)\Psi_P & \mathit{distinct}\,\widetilde{b}_P & \mathit{bn}\,\alpha \sharp \mathit{subject}\,\alpha & \mathit{distinct}\,(\mathit{bn}\,\alpha) & & \\ \alpha \neq \tau & \widetilde{x} \sharp P & \widetilde{b}_P \sharp \Psi & \widetilde{b}_P \sharp \widetilde{x} & \widetilde{b}_P \sharp P & \widetilde{b}_P \sharp \mathit{subject}\,\alpha\end{array}}{\exists M.\ \Psi \otimes \Psi_P \vdash \mathit{the}\,(\mathit{subject}\,\alpha) \leftrightarrow M \wedge \widetilde{x} \sharp M}$$

*Proof* By induction on $\Psi \rhd P \xrightarrow{\alpha} P'$, using the induction rule from Fig. 4. Since $\alpha \neq \tau$, *the* (*subject* $\alpha$) is well-defined. □

This lemma obtains the subject from the prefix of $P$. Moreover, it ensures that any sequence of names $\widetilde{x}$ that are fresh for $P$ and for the bound names $\widetilde{b}_P$ of the frame of $P$ are also fresh for the subject.

The following lemmas can then be used to replace the subject of an action.

**Lemma 13** (Replacing the subject of an input action)

$$\frac{\begin{array}{ccccc}\Psi \rhd P \xrightarrow{MN} P' & \mathcal{F}\,P = (\nu \widetilde{b}_P)\Psi_P & \mathit{distinct}\,\widetilde{b}_P & & \\ \Psi \otimes \Psi_P \vdash M \leftrightarrow K & \widetilde{b}_P \sharp \Psi & \widetilde{b}_P \sharp P & \widetilde{b}_P \sharp M & \widetilde{b}_P \sharp K\end{array}}{\Psi \rhd P \xrightarrow{KN} P'}$$

*Proof* By induction on $\Psi \rhd P \xrightarrow{MN} P'$, using the induction rule from Fig. 4. □

**Lemma 14** (Replacing the subject of an output action)

$$\frac{\begin{array}{ccccc}\Psi \rhd P \xrightarrow{\overline{M}(\nu \widetilde{x})N} P' & \mathcal{F}\,P = (\nu \widetilde{b}_P)\Psi_P & \mathit{distinct}\,\widetilde{b}_P & & \\ \Psi \otimes \Psi_P \vdash M \leftrightarrow K & \widetilde{b}_P \sharp \Psi & \widetilde{b}_P \sharp P & \widetilde{b}_P \sharp M & \widetilde{b}_P \sharp K\end{array}}{\Psi \rhd P \xrightarrow{\overline{K}(\nu \widetilde{x})N} P'}$$

*Proof* By induction on $\Psi \rhd P \xrightarrow{\overline{M}N} P'$, using the induction rule from Fig. 4. □

## 6 Inversion Rules

Theorem provers use inversion rules (also known as case rules or elimination rules) to perform case analysis over inductively defined data types and predicates. These rules are used when reasoning about terms of a specific shape. For instance, a transition of the form $\Psi \rhd P \,|\, Q \xrightarrow{\alpha} R$ must be derived by one of the PAR rules or the COMM rule; a transition of the form $\Psi \rhd (\nu x)P \xrightarrow{\alpha} P'$ must be derived by either the OPEN or the RES rule. An inversion rule, when given a transition, splits the proof into one subgoal for each possible case from which the transition can be derived.

Isabelle automatically generates inversion rules for terms and predicates that are equated using standard equality of higher-order logic. The rule that is generated for the operational semantics of psi-calculi is presented in Fig. 5. For space reasons, only the PAR case is shown in detail. The rule allows inversion for terms of the form $\Psi_R \rhd R \longmapsto W$, and each case

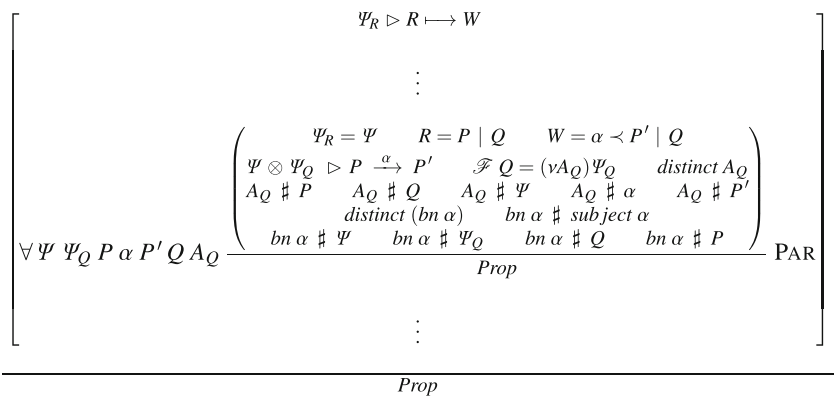


**Fig. 5** Generated inversion rule for the operational semantics of psi-calculi. The terms $\Psi_R$, $R$, $W$ and *Prop* are global for the entire rule. Each case has a set of equality constraints for these terms

of the rule imposes equality constraints on $\Psi_R$, $R$ and $W$ that are used to ascertain whether that particular case fires. For instance, a transition of the form $\Psi' \rhd S \mid T \xrightarrow{\beta} U$ can be derived by the PAR rule, and the inversion rule then provides $\Psi \otimes \Psi_Q \rhd P \xrightarrow{\alpha} P'$ along with three equality constraints $\Psi' = \Psi$, $S|T = P|Q$, and $\beta \prec U = \alpha \prec P'|Q$.

The first equality can immediately be used to substitute $\Psi$ for $\Psi'$ in proofs. By injectivity of the parallel composition operator $|$, we obtain $S = P$ and $T = Q$ from the second equality. Also these equalities can immediately be used in proofs. It is the third equality that causes difficulties, as both $\beta$ and $\alpha$ may contain bound names that bind into $U$ and $P'|Q$ respectively. Using the third equality in proofs therefore requires explicit alpha-conversions via Lemma 9 to reason about all possible alpha-variants of $\beta \prec U$.

This is unsatisfactory. When we prove a statement by inversion over the transition $\Psi \rhd P \xrightarrow{\alpha} P'$, we want to reason about the bound names that actually occur in $\alpha$, as that action has already been fixed in the proof context. We should not be forced into reasoning about alpha-equivalent variants.

This problem is not new. Berghofer et al. [13] already added inversion support for formalisations that use single binders to Nominal Isabelle. Their solution is to strengthen the standard inversion rule by quantifying all bound names globally, rather than locally as in Fig. 5. This allows the user to choose the bound names of each case to match those in the term that is being inverted. All equality constraints can then be used in proofs with the help of the standard injectivity rules for each constructor. For instance, if $\beta \prec T = \alpha \prec P'|Q$ and the bound names of $\alpha$ and $\beta$ are the same, which the user can guarantee as a result of the strengthened inversion rule, then we know that their subjects and objecs are also equal, and that $T = P'|Q$. However, fixing the bound names prior to inversion is sound only if these names are sufficiently fresh. The variable convention compatibility condition mentioned in Section 5.4 makes precise exactly what the freshness conditions are.

### 6.1 Inversion with Binding Sequences

Berghofer's extension to Nominal Isabelle only covers formalisations that use single binders, and it has remained an open question how to extend this approach to calculi that use binding sequences. We propose a technique to generate inversion rules for these types

of formalisations. Our technique has been used successfully to generate inversion rules for psi-calculi, and also by Berghofer in a formalisation of the simply typed lambda-calculus extended with let patterns for tuples [12].

Our approach is to derive a strengthened inversion rule from the standard rule generated by Isabelle. In this section we present, step-by-step, a heuristic for strengthening the standard rule. For the sake of exposition we start by presenting the PAR case of our strengthened inversion rule. This has the form

$$\forall \Psi\ \Psi_Q\ P\ \alpha\ P'\ Q\ A_Q\ \frac{\left(\begin{array}{cccccc} & & \widetilde{x} \sharp \Psi_R & \widetilde{x} \sharp R & \widetilde{x} \sharp W & \\ \hline \Psi_R = \Psi & R = P \mid Q & W = \alpha \prec P' \mid Q & \widetilde{x} = bn\,\alpha & & \\ \Psi \otimes \Psi_Q \rhd P \xrightarrow{\alpha} P' & & \mathscr{F}\ Q = (\nu A_Q)\Psi_Q & distinct\ A_Q & & \\ A_Q \sharp P & A_Q \sharp Q & A_Q \sharp \Psi & A_Q \sharp \alpha & A_Q \sharp P' & \\ & & A_Q \sharp \mathscr{C} & & & \end{array}\right)}{Prop}$$

where $\Psi_R$, $R$, $W$ and *Prop* are globally quantified just as in the standard inversion rule.

There are a few things to note here. First, the binding sequence $\widetilde{x}$ is globally quantified for the entire rule. Intuitively, it matches the bound names in the action of $W$. The action $\alpha$ is, however, locally quantified for the PAR case. The equality constraint $\widetilde{x} = bn\,\alpha$ ensures that $\widetilde{x}$ corresponds exactly to the bound names of $\alpha$, making the constraint $W = \alpha \prec P' \mid Q$ immediately usable in proofs by injectivity of its constructors. Second, $\widetilde{x}$ must be sufficiently fresh, i.e., fresh for $\Psi_R$, $R$ and $W$. The intuition is that as long as $\widetilde{x}$ is sufficiently fresh, $\alpha$ can be chosen such that $\widetilde{x} = bn\,\alpha$. Finally, the binding sequence $A_Q$ remains locally quantified. The reason for this is that $A_Q$ is not part of the transition being inverted, i.e., it is not syntactically present in $\Psi_R \rhd R \longmapsto W$. If we need an inversion rule that performs inversion over a transition where the agent has a specific frame, we can create such a rule in a similar way as in Section 5.6.

In the single binder case, deriving the strengthened inversion rule is straightforward: the user chooses the bound names for each case, and as long as those names are sufficiently fresh, the standard inversion rule can be alpha-converted to match. For rules using binding sequences things are not as simple. Here knowing that the sequences are sufficiently fresh is not enough. Consider a transition $\Psi \rhd P \xrightarrow{\alpha} P'$. In order to alpha-convert the bound names in $\alpha$ to a given sequence $\widetilde{x}$, we must know that $\widetilde{x}$ is distinct, is sufficiently fresh, and has the same length as the bound names of $\alpha$.

We use the following lemma to create alpha-converting permutations for distinct sequences of equal length.

**Lemma 15**

$$\frac{|\widetilde{x}| = |\widetilde{y}| \qquad \widetilde{x} \# \widetilde{y} \qquad distinct\ \widetilde{x} \qquad distinct\ \widetilde{y}}{\exists p.set,\ p \subseteq set\ \widetilde{x} \times set\ (p \cdot \widetilde{x}) \wedge distinctPerm\ p \wedge \widetilde{y} = p \cdot \widetilde{x}}$$

*Proof* By induction on $|\widetilde{x}|$. The permutation $p$ is constructed by pairwise combining corresponding elements from $\widetilde{x}$ and $\widetilde{y}$. □

The next step then is to calculate the length of binding sequences in actions. Given a transition of the form $\Psi \rhd P \xrightarrow{\alpha} P'$, this is straightforward: the desired number is simply $|bn\,\alpha|$. In the general case, transitions have the form $\Psi \rhd P \longmapsto W$, and we cannot define a function that extracts the bound names of a residual in nominal logic, for these are not invariant under alpha-conversion. However, the *length* of the binding sequence is invariant

under alpha-conversion, and it is possible to count the number of binders in a term. The following function *residualLength* returns the length of the binding sequence in the action of an arbitrary residual.

**Definition 23** (residualLength)

$$
\begin{aligned}
&\mathit{boundOutputLength} :: (\alpha,\ \beta,\ \gamma)\ \mathit{boundOutput} \Rightarrow \mathbb{N}\\
&\mathit{boundOutputLength}\ (\mathit{BOut}\ M\ P) = 0\\
&\mathit{boundOutputLength}\ (\mathit{BStep}\ x\ B) = \mathit{boundOutputLength}\ B + 1
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{residualLength} :: (\alpha,\ \beta,\ \gamma)\ \mathit{residual} \Rightarrow \mathbb{N}\\
&\mathit{residualLength}\ (\mathit{RIn}\ M\ N\ P) = 0\\
&\mathit{residualLength}\ (\mathit{ROut}\ M\ B) = \mathit{boundOutputLength}\ B\\
&\mathit{residualLength}\ (\mathit{RTau}\ P) = 0
\end{aligned}
$$

We define a similar function to calculate the length of the binding sequence in input forms. With these measures in place, we can state and prove the strengthened inversion rule, presented in Fig. 6.

This technique is general enough to be automated in Nominal Isabelle. It merely requires functions to calculate the length of binding sequences in nominal data types, and ways to retrieve the bound names in each specific case (such as the *bn* function). We believe the remaining automation work to be primarily an engineering challenge, rather than a theoretical one.

## 7 Strong Bisimilarity

Having mechanised the syntax and labelled operational semantics of agents, we now turn our attention to the meta-theory of psi-calculi. As mentioned in the introduction, any practically useful process calculus must satisfy certain fundamental properties, for instance compositionality: that the semantics of a process can be deduced from the semantics of

$$
\left[\begin{array}{c}
\Psi_R \rhd R \longmapsto W \qquad \mathit{distinct}\ \widetilde{x} \qquad |\widetilde{x}| = \mathit{residualLength}\ W\\
\vdots\\
\forall\, \Psi\ \Psi_Q\ P\ \alpha\ P'\ Q\ A_Q\ \dfrac{\left(\begin{array}{ccccc}
 & \widetilde{x} \sharp \Psi_R & \widetilde{x} \sharp R & \widetilde{x} \sharp W & \\
\Psi_R = \Psi & R = P \mid Q & W = \alpha \prec P' \mid Q & & \widetilde{x} = \mathit{bn}\ \alpha\\
\Psi \otimes \Psi_Q \rhd P \xrightarrow{\alpha} P' & & \mathscr{F}\ Q = (\nu A_Q)\Psi_Q & & \mathit{distinct}\ A_Q\\
A_Q \sharp P & A_Q \sharp Q & A_Q \sharp \Psi & A_Q \sharp \alpha & A_Q \sharp P'\\
 & & A_Q \sharp \mathscr{C} & &
\end{array}\right)}{\mathit{Prop}}\ \textsc{Par}\\
\vdots
\end{array}\right]
\over \mathit{Prop}
$$

**Fig. 6** Derived inversion rule for the operational semantics of psi-calculi. The terms $\Psi_R$, $R$, $W$ and *Prop* are global for the entire rule. Each case has a set of equality constraints for these terms

its components. In this section, we establish that strong bisimilarity for psi-calculi satisfies these properties. Thereby, we demonstrate that in our formalisation such meta-theoretic results can be proven with reasonable effort. Simultaneously, we validate our definitions of the psi-calculi framework, gaining additional confidence that these definitions are appropriate.

Bisimilarity, originally defined by Park [42] and popularised by Milner [36] among others, is a concept that appears in many areas of mathematical logic and computer science. Intuitively, two processes are bisimilar if they can mimic each other's actions. In this sense, bisimilar processes cannot be distinguished from each other by an observer. We consider *strong bisimilarity* in this section, meaning that all actions (including $\tau$-actions) must be matched exactly.

Strong bisimilarity for psi-calculi differs from that for the pi-calculus and CCS [10] in three main regards. First, bisimilarity is parametrised by an environment in which the agents operate. Second, as in the applied pi-calculus, the frames of bisimilar agents must be statically equivalent. Third, if two agents are bisimilar in an environment, they must be bisimilar for all possible extensions of that environment. These issues are explained at length in [9].

While these additions add to the complexity of the framework, the formalisation techniques used for the pi-calculus and CCS scale remarkably well. Bisimilarity is defined coinductively, through simulations that are defined in the standard way, with the exception that the simulation relation is ternary rather than binary.

The proof techniques deserve special mention. In the corresponding proofs for the pi-calculus, case analysis is performed on the actions that an agent can do, and the rules of the operational semantics are applied to the simulating agent to mimic these actions. In psi-calculi, however, an agent in a parallel composition may use the frame of the other agent to derive its transitions. For bisimilarity proofs, this requires that any transition derived using the frame of an agent must also be derivable using the frame of any bisimilar agent. One of our main contributions is a smooth and transparent formalisation of this requirement.

### 7.1 Definitions

Bisimilarity rests on a notion of *simulation*. Intuitively, an agent $P$ can simulate an agent $Q$ preserving a relation $\mathscr{R}$ if $P$ can take any action that $Q$ can take; moreover, the derivatives of $P$ and $Q$ must be related via $\mathscr{R}$. Simulation is parametrised by an environment in which the agents operate.

The definition of simulation for psi-calculi is straightforward. Since residuals are defined to contain both free and bound names, no explicit case distinction is necessary for different actions. Freshness conditions ensure that the bound names are fresh for the environment and the simulating agent.

**Definition 24** (Simulation) An agent $P$ simulating an agent $Q$ preserving the relation $\mathscr{R}$ in the environment $\Psi$ is denoted $\Psi \rhd P \hookrightarrow_{\mathscr{R}} Q$.

$$\Psi \rhd P \hookrightarrow_{\mathscr{R}} Q \quad \equiv \quad \begin{array}{l} \forall \alpha\ Q'.\Psi \rhd Q \xrightarrow{\alpha} Q' \wedge bn\,\alpha \# \Psi \wedge bn\,\alpha \# P \longrightarrow \\ \quad \exists P'.\Psi \rhd P \xrightarrow{\alpha} P' \wedge (\Psi, P', Q') \in \mathscr{R} \end{array}$$

Bisimilarity can now be defined coinductively in the standard way.

**Definition 25** (Strong bisimilarity) Bisimilarity, denoted $\dot{\sim}$, is defined coinductively as the greatest fixpoint satisfying

$$\begin{array}{lll} \Psi \rhd P \dot{\sim} Q \implies (\mathscr{F}\ P) \otimes \Psi \simeq (\mathscr{F}\ Q) \otimes \Psi & & \textsc{StatEq} \\ \quad \wedge\ \Psi \rhd P \hookrightarrow_{\dot{\sim}} Q & & \textsc{Simulation} \\ \quad \wedge\ \forall \Psi'. \Psi \otimes \Psi' \rhd P \dot{\sim} Q & & \textsc{Extension} \\ \quad \wedge\ \Psi \rhd Q \dot{\sim} P & & \textsc{Symmetry} \end{array}$$

## 7.2 Introduction and Elimination Rules

Simulation for psi-calculi (Definition 24) ensures freshness conditions for the bound names in the actions of transitions. For equivariant candidate relations, we derive an introduction rule where these bound names are additionally guaranteed to be distinct and sufficiently fresh for a context $\mathscr{C}$.

**Lemma 16** (Introduction rule for simulation)

$$\frac{\begin{array}{c} eqvt\ \mathscr{R} \\ \forall \alpha\ Q'.\ \dfrac{\begin{array}{c} \Psi \rhd Q \xrightarrow{\alpha} Q' \qquad bn\ \alpha \sharp P \qquad bn\ \alpha \sharp Q \qquad bn\ \alpha \sharp \Psi \\ distinct\ (bn\ \alpha) \qquad bn\ \alpha \sharp subject\ \alpha \qquad bn\ \alpha \sharp \mathscr{C} \end{array}}{\exists P'.\ \Psi \rhd P \xrightarrow{\alpha} P' \wedge (\Psi, P', Q') \in \mathscr{R}} \end{array}}{\Psi \rhd P \hookrightarrow_{\mathscr{R}} Q}\ \hookrightarrow\text{-I}$$

*Proof* Follows from the definition of $\hookrightarrow$. The bound names in $\alpha$ are alpha-converted to become distinct and avoid $\Psi$, $P$, $Q$, *subject* $\alpha$ and $\mathscr{C}$. The fact that $\mathscr{R}$ is equivariant allows the alpha-converting permutations to be applied to the derivatives in $\mathscr{R}$. □

Even though a name is fresh for an agent, it may appear in the subject of a transition taken by that agent, similarly to the situation for frame induction (Section 5.6). Therefore, we prove the following introduction rule for simulation, which additionally ensures that the action $\alpha$ is fresh for a sequence $\tilde{x}$ of names. Again, $\mathscr{R}$ must be equivariant, and $\tilde{x}$ must be fresh for the environment and the originating agents.

**Lemma 17** (Introduction rule for simulation ensuring fresh subjects)

$$\frac{\begin{array}{c} eqvt\ \mathscr{R} \qquad \tilde{x} \sharp \Psi \qquad \tilde{x} \sharp P \qquad \tilde{x} \sharp Q \\ \forall \alpha\ Q'.\ \dfrac{\begin{array}{c} \Psi \rhd Q \xrightarrow{\alpha} Q' \qquad bn\ \alpha \sharp P \qquad bn\ \alpha \sharp Q \qquad bn\ \alpha \sharp \Psi \\ bn\ \alpha \sharp subject\ \alpha \qquad distinct\ (bn\ \alpha) \qquad bn\ \alpha \sharp \mathscr{C} \qquad \tilde{x} \sharp \alpha \qquad \tilde{x} \sharp Q' \end{array}}{\exists P'.\ \Psi \rhd P \xrightarrow{\alpha} P' \wedge (\Psi, P', Q') \in \mathscr{R}} \end{array}}{\Psi \rhd P \hookrightarrow_{\mathscr{R}} Q}$$

*Proof* The introduction rule $\hookrightarrow$-I is used such that the bound names of the transition avoid both $\mathscr{C}$ and $\tilde{x}$. A fresh subject is obtained from Lemma 12, and exchanged in the transition by Lemma 13 in case $\alpha$ is an input action, or by Lemma 14 in case $\alpha$ is an output action. □

The elimination rule for simulation, as well as the introduction and elimination rules for bisimilarity, follow immediately from the respective definitions.

**Lemma 18** (Elimination rule for simulation)

$$\frac{\Psi \rhd P \hookrightarrow_{\mathscr{R}} Q \qquad \Psi \rhd Q \xrightarrow{\alpha} Q' \qquad bn\,\alpha \# \Psi \qquad bn\,\alpha \# P}{\exists P'.\Psi \rhd P \xrightarrow{\alpha} P' \wedge (\Psi, P', Q') \in \mathscr{R}} \hookrightarrow -E$$

*Proof* Follows from the definition of $\hookrightarrow$. □

**Lemma 19** (Introduction and elimination rules for bisimilarity)

$$\frac{(\mathscr{F}\ P) \otimes \Psi \simeq (\mathscr{F}\ Q) \otimes \Psi \qquad \Psi \rhd P \hookrightarrow_{\dot{\sim}} Q \qquad \forall \Psi'.\ \Psi \otimes \Psi' \rhd P \dot{\sim} Q \qquad \Psi \rhd Q \dot{\sim} P}{\Psi \rhd P \dot{\sim} Q} \dot{\sim} - \mathrm{I}$$

$$\frac{\Psi \rhd P \dot{\sim} Q}{(\mathscr{F}\ P) \otimes \Psi \simeq (\mathscr{F}\ Q) \otimes \Psi} \dot{\sim} - \mathrm{E1} \qquad \frac{\Psi \rhd P \dot{\sim} Q}{\Psi \rhd P \hookrightarrow_{\dot{\sim}} Q} \dot{\sim} - \mathrm{E2}$$

$$\frac{\Psi \rhd P \dot{\sim} Q}{\Psi \otimes \Psi' \rhd P \dot{\sim} Q} \dot{\sim} - \mathrm{E3} \qquad \frac{\Psi \rhd P \dot{\sim} Q}{\Psi \rhd Q \dot{\sim} P} \dot{\sim} - \mathrm{E4}$$

*Proof* Follows from the definition of $\dot{\sim}$. □

Since bisimilarity is defined coinductively, its introduction rule is hardly useful to prove that two agents are bisimilar. We use a standard technique to establish bisimilarity between agents. A symmetric candidate relation $\mathscr{X}$ is chosen such that all agents related by $\mathscr{X}$ simulate each other in a fixed environment, and their derivatives are either in $\mathscr{X}$ or bisimilar in that environment. Moreover, agents that are related by $\mathscr{X}$ must remain related—or become bisimilar—when the environment is extended with an arbitrary assertion. In effect this allows us to prove that two agents are bisimilar up to bisimulation. The following lemma codifies this proof technique.

**Lemma 20** (Coinduction rule for bisimilarity)

$$\frac{\begin{array}{ll} (\Psi, P, Q) \in \mathscr{X} & \\ \forall \Psi' R S.\ \dfrac{(\Psi', R, S) \in \mathscr{X}}{(\mathscr{F}\ R) \otimes \Psi' \simeq (\mathscr{F}\ S) \otimes \Psi'} & \textsc{StatEq} \\ \forall \Psi' R S.\ \dfrac{(\Psi', R, S) \in \mathscr{X}}{\Psi' \rhd R \hookrightarrow_{\mathscr{X} \cup \dot{\sim}} S} & \textsc{Simulation} \\ \forall \Psi' R S\ \Psi''.\ \dfrac{(\Psi', R, S) \in \mathscr{X}}{(\Psi' \otimes \Psi'', R, S) \in \mathscr{X} \vee \Psi' \otimes \Psi'' \rhd R \dot{\sim} S} & \textsc{Extension} \\ \forall \Psi' R S.\ \dfrac{(\Psi', R, S) \in \mathscr{X}}{(\Psi', S, R) \in \mathscr{X} \vee \Psi' \rhd S \dot{\sim} R} & \textsc{Symmetry} \end{array}}{\Psi \rhd P \dot{\sim} Q}$$

*Proof* Follows from the definition of $\dot{\sim}$. Isabelle automatically generates a coinduction rule for bisimilarity, from which this rule is derived. □

Hur et al. recently proposed parameterised coinduction [30], a proof technique that does not require the candidate relation to be specified up front, but allows it to be developed incrementally during the course of the proof. We hypothesise that this technique could have simplified our interactive proof development, but we caution that Isabelle does not offer specific automation for parameterised coinduction.

### 7.3 Preservation Properties

We prove that bisimilarity is preserved by all constructors of the *psi* data type (Definition 4) except *Input*. This replicates a similar result for the pi-calculus [10].

**Theorem 3** *Bisimilarity is preserved by all constructors except Input.*

The rest of this section outlines our proof of Theorem 3. For space reasons, we only discuss the *Res* and *Par* constructors, i.e., we show that $\Psi \rhd (\nu x)P \dot{\sim} (\nu x)Q$ whenever $\Psi \rhd P \dot{\sim} Q$, and that $\Psi \rhd P \mid Q \dot{\sim} P' \mid Q'$ whenever $\Psi \rhd P \dot{\sim} P'$ and $\Psi \rhd Q \dot{\sim} Q'$. The restriction case is of interest because it involves binding, and the parallel case is the most difficult of all, because an agent in a parallel composition may use the frame of the other agent to derive its transitions. The remaining cases, which present no special difficulties, are covered in our formal development [8].

The proofs will follow a standard pattern and be divided into simulation and bisimilarity lemmas. The simulation lemmas refer to arbitrary relations, and impose constraints only as required for the proof of the lemma. When proving bisimilarity, a candidate relation is chosen that satisfies the constraints of the simulation lemma. With this approach, we were able to add constraints while we were developing the formal theories. The candidate relations for bisimilarity closely resemble those for corresponding cases for the pi-calculus.

#### *7.3.1 Restriction*

We start by proving that simulation is preserved by restriction. The lemma requires that the bound name is fresh for the environment.

**Lemma 21** (Simulation is preserved by restriction)

$$\frac{\Psi \rhd P \hookrightarrow_{\mathscr{R}} Q \qquad eqvt\ \mathscr{R}' \qquad x \sharp \Psi \qquad \mathscr{R} \subseteq \mathscr{R}' \qquad \forall \Psi' R S y.\ \dfrac{(\Psi', R, S) \in \mathscr{R} \qquad y \sharp \Psi'}{(\Psi', (\nu y)R, (\nu y)S) \in \mathscr{R}'}}{\Psi \rhd (\nu x)P \hookrightarrow_{\mathscr{R}'} (\nu x)Q}$$

*Proof* The proof follows the usual structure: inversion on the restricted agent $(\nu x)Q$ demonstrates that its transitions are obtained through the SCOPE and OPEN rules of the operational semantics (Fig. 1). The same semantics rule can then be used to derive a corresponding transition for the simulating agent $(\nu x)P$.

Note that both SCOPE and OPEN require the bound name to be fresh for the subject of the transition. Since $x \sharp \Psi$, $x \sharp (\nu x)P$, and $x \sharp (\nu x)Q$, Lemma 17 allows us to consider only those actions and derivatives of $(\nu x)Q$ for which $x$ is fresh. □

We prove a corresponding lemma for binding sequences.

**Lemma 22**

$$\frac{\Psi \rhd P \hookrightarrow_{\mathscr{R}} Q \qquad eqvt\,\mathscr{R} \qquad \widetilde{x} \sharp \Psi \qquad \forall \Psi' R S y.\ \dfrac{(\Psi', R, S) \in \mathscr{R} \qquad y \sharp \Psi'}{(\Psi', (\nu y)R, (\nu y)S) \in \mathscr{R}}}{\Psi \rhd (\nu \widetilde{x})P \hookrightarrow_{\mathscr{R}} (\nu \widetilde{x})Q}$$

*Proof* By induction on $\widetilde{x}$, using Lemma 21. □

The static equivalence case of the coinduction rule for bisimilarity requires us to show that static equivalence of frames is preserved by restriction. We first prove the corresponding property for static implication.

**Lemma 23** *If $F \leq G$ then $((\nu x)G)$.*

*Proof* Let $\varphi$ be a condition. We must prove that if $((\nu x)F) \vdash \varphi$ then $((\nu x)G) \vdash \varphi$. The main complication lies in the fact that $x$ is not guaranteed to be fresh for $\varphi$.

We obtain a fresh name $y$ such that $y$ is fresh for everything in the proof context. Since $((\nu x)F) \vdash \varphi$, we have by alpha-conversion that $((\nu y)(xy)\cdot F) \vdash \varphi$. Because $y \# \varphi$, it follows that $(xy) \cdot F \vdash \varphi$. From $F \leq G$ we obtain $(xy) \cdot F \leq (xy) \cdot G$ by equivariance, and therefore $(xy) \cdot G \vdash \varphi$. Because $y \# \varphi$, it follows that $((\nu y)(xy) \cdot G) \vdash \varphi$. Hence, $((\nu x)G) \vdash \varphi$ by alpha-conversion. □

We can now prove that static equivalence of frames is preserved by restriction.

**Lemma 24** *If $F \simeq G$ then $((\nu x)F) \simeq ((\nu x)G)$.*

*Proof* Follows immediately from the definition of $\simeq$ and Lemma 23. □

We lift the previous lemma to sequences of restriction binders.

**Lemma 25** *If $F \simeq G$ then $((\nu \widetilde{x})F) \simeq ((\nu \widetilde{x})G)$.*

*Proof* By induction on $\widetilde{x}$, using Lemma 24. □

We can now prove that bisimilarity is preserved by restriction. The restricted name must not occur in the environment.

**Lemma 26** (Bisimilarity is preserved by restriction) *If $\Psi \rhd P \mathrel{\dot{\sim}} Q$ and $x \# \Psi$ then $\Psi \rhd (\nu x)P \mathrel{\dot{\sim}} (\nu x)Q$.*

*Proof* By coinduction (Lemma 20), with $\mathscr{X} = \{(\Psi, (\nu x)P, (\nu x)Q) : \Psi \rhd P \dot{\sim} Q \wedge x \# \Psi\}$.

StatEq: From $\Psi \rhd P \dot{\sim} Q$ we have $(\mathscr{F} P) \otimes \Psi \simeq (\mathscr{F} Q) \otimes \Psi$ by $\dot{\sim}$-E1. Hence, $(\mathscr{F}((\nu x)P)) \otimes \Psi \simeq (\mathscr{F}((\nu x)Q)) \otimes \Psi$ using $x \# \Psi$ and Lemma 24.

SIMULATION: From $\Psi \rhd P \mathrel{\dot{\sim}} Q$ we have $\Psi \rhd P \hookrightarrow_{\dot{\sim}} Q$ by $\dot{\sim}$-E2. Hence, using $x \# \Psi$ and the fact that bisimilarity and $\mathscr{X}$ are equivariant, $\Psi \rhd (\nu x)P \hookrightarrow_{\mathscr{X} \cup \dot{\sim}} (\nu x)Q$ by Lemma 21.

EXTENSION: Given $\Psi \rhd P \mathrel{\dot{\sim}} Q$, we must prove that $(\Psi \otimes \Psi', (\nu x)P, (\nu x)Q) \in \mathscr{X}$ for all assertions $\Psi'$, including those containing names that clash with $x$.

We obtain a fresh name $y$ such that $y \# \Psi$, $y \# \Psi'$, $y \# P$, and $y \# Q$. From $\Psi \rhd P \mathrel{\dot{\sim}} Q$ we have $\Psi \otimes (x\,y) \cdot \Psi' \rhd P \mathrel{\dot{\sim}} Q$ by $\dot{\sim}$-E3. Equivariance of bisimilarity and $\otimes$ then yields $(x\,y) \cdot \Psi \otimes \Psi' \rhd (x\,y) \cdot P \mathrel{\dot{\sim}} (x\,y) \cdot Q$. Since $x \# \Psi$ and $y \# \Psi$, the latter simplifies to $\Psi \otimes \Psi' \rhd (x\,y) \cdot P \mathrel{\dot{\sim}} (x\,y) \cdot Q$. Since $y \# \Psi$ and $y \# \Psi'$, we have $y \# \Psi \otimes \Psi'$. Hence, this demonstrates that $(\Psi \otimes \Psi', (\nu y)(x\,y) \cdot P, (\nu y)(x\,y) \cdot Q) \in \mathscr{X}$. Finally, since $y \# P$ and $y \# Q$, we obtain $(\Psi \otimes \Psi', (\nu x)P, (\nu x)\,Q) \in \mathscr{X}$ by alpha-conversion.

SYMMETRY: Symmetry of $\mathscr{X}$ follows from symmetry of bisimilarity.

□

We again prove a corresponding lemma for binding sequences.

**Lemma 27** *If* $\Psi \rhd P \mathrel{\dot{\sim}} Q$ *and* $\widetilde{x} \# \Psi$ *then* $\Psi \rhd (\nu \widetilde{x})P \mathrel{\dot{\sim}} (\nu \widetilde{x})Q$.

*Proof* By induction on $\widetilde{x}$, using Lemma 26. □

### *7.3.2 Parallel Composition*

The proof that bisimilarity is preserved by parallel composition is the most difficult of the preservation proofs. Historically, this is the property that most often fails in calculi of this complexity: the intricate correspondences between parallel processes and their assertions are hard to get completely right. Our main result is the following.

**Lemma 28** (Bisimilarity is preserved by parallel composition) *If* $\Psi \rhd P \mathrel{\dot{\sim}} Q$ *then* $\Psi \rhd P \,|\, R \mathrel{\dot{\sim}} Q \,|\, R$.

The more general result that $\Psi \rhd P \mathrel{\dot{\sim}} P'$ and $\Psi \rhd Q \mathrel{\dot{\sim}} Q'$ imply $\Psi \rhd P \,|\, Q \mathrel{\dot{\sim}} P' \,|\, Q'$ easily follows. For space reasons, we merely give an outline of the proof, pointing out the complicated cases. Freshness conditions and other details will be glossed over. We refer to [7] for a more complete discussion, and to our formal development as the ultimate, machine-checked reference [8]. An informal but detailed account of this proof is contained in [9].

It is sufficient to prove that $\Psi \otimes \Psi_R \rhd P \mathrel{\dot{\sim}} Q$ implies $\Psi \rhd P \,|\, R \mathrel{\dot{\sim}} Q \,|\, R$. The general case, where $P$ and $Q$ are bisimilar in the frame $\Psi$, then follows by choosing a sufficiently fresh frame of $R$.

The proof proceeds by coinduction (Lemma 20). We use the candidate relation

$$\mathscr{X} = \{(\Psi, (\nu \widetilde{x})(P|R), (\nu \widetilde{x})(Q|R)) : \Psi \otimes \Psi_R \rhd P \mathrel{\dot{\sim}} Q\}$$

We then need to prove the SYMMETRY, EXTENSION, STATEQ, and SIMULATION premises of the coinduction rule for this candidate relation.

Clearly, $\mathscr{X}$ is symmetric since bisimilarity is symmetric. This proves the SYMMETRY premise.

The EXTENSION premise follows from the definition of $\mathscr{X}$, using the extension property of bisimilarity ($\dot{\sim}$-E3).

To prove static equivalence (StatEq), we get to assume

$$(\mathcal{F}\ P) \otimes (\Psi \otimes \Psi_R) \simeq (\mathcal{F}\ Q) \otimes (\Psi \otimes \Psi_R)$$

and have to show that

$$(\mathcal{F}\ ((\nu\tilde{x})(P|R))) \otimes \Psi \simeq (\mathcal{F}\ ((\nu\tilde{x})(Q|R))) \otimes \Psi.$$

The equivalence follows by algebraic manipulations that exploit associativity and commutativity of $\otimes$, Lemma 25, and the fact that binding sequences in frames commute. The latter is proved by induction on the sequences involved.

The Simulation premise is the most difficult one. Given $\Psi \otimes \Psi_R \rhd P \dot{\sim} Q$ (hence by $\Psi \otimes \Psi_R \rhd P \hookrightarrow_{\dot{\sim}} Q$ -E2), we must prove $\Psi \rhd (\nu\tilde{x})(P|R) \hookrightarrow_{\mathscr{X} \cup \dot{\sim}} (\nu\tilde{x})(Q \mid R)$. It is sufficient to prove that simulation is preserved by parallel composition: we then have $\Psi \rhd P|R \hookrightarrow_{\mathscr{X} \cup \dot{\sim}} Q \mid R$, and the claim follows because simulation is preserved by restriction (Lemma 22).

We now sketch the proof that simulation is preserved by parallel composition. The Par inversion rule, applied to $Q|R$, gives us four cases: two where either $Q$ or $R$ performs an action, and two where the parallel processes communicate.

1. $\Psi \otimes \Psi_R \rhd Q \xrightarrow{\alpha} Q'$, where we need to find an agent $S$ such that $\Psi \rhd P|R \xrightarrow{\alpha} S$ and $(\Psi,\ S\ Q' \mid R) \in \mathscr{X}$.
2. $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\alpha} R'$, where we need to find an agent $s$ such that $\Psi \rhd P|R \xrightarrow{\alpha} S$ and $(\Psi,\ S\ Q \mid R') \in \mathscr{X}$.
3. $\Psi \otimes \Psi_R \rhd Q \xrightarrow{M\,N} Q', \Psi \otimes \Psi_Q \rhd R \xrightarrow{\overline{K}(\nu\tilde{x})N} R'$, and $\Psi \otimes (\Psi_Q \otimes \Psi_R) \vdash M \leftrightarrow K$, where we need to find an agent $S$ such that $\Psi \rhd P|R \xrightarrow{\tau} S$ and $(\Psi,\ S\ ,(\nu\tilde{x})(Q' \mid R')) \in \mathscr{X}$.
4. $\Psi \otimes \Psi_R \rhd Q \xrightarrow{\overline{M}(\nu\tilde{x})N} Q', \Psi \otimes \Psi_Q \rhd R \xrightarrow{K\,N} R'$, and $\Psi \otimes (\Psi_Q \otimes \Psi_R) \vdash M \leftrightarrow K$, where we need to find an agent $S$ such that $\Psi \rhd P|R \xrightarrow{\tau} S$ and $(\Psi,\ S\ ,(\nu\tilde{x})(Q' \mid R')) \in \mathscr{X}$.

Case 1 is straightforward, and can be solved in much the same way as the corresponding case for the pi-calculus and CCS [10]. Since $P$ and $Q$ are bisimilar, there is a derivative $P'$ such that $\Psi \otimes \Psi_R \rhd P \xrightarrow{\alpha} P'$ and $\Psi \otimes \Psi_R \rhd P' \dot{\sim} Q'$. Taking $S = P' \mid R$ concludes the case.

Case 2 is an easy case for both the pi-calculus and CCS. The witness agent is $S = P|R'$. However, for psi-calculi there are two complications. First, in order to derive the transition $\Psi \rhd P|R \xrightarrow{\alpha} P|R'$ we need to know that $\Psi \otimes \Psi_P \rhd R \xrightarrow{\alpha} R'$, but the inversion rule provides $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\alpha} R'$: the transition $\alpha$ is derived in the frame of $Q$, but it must be derived in the frame of $P$. Since $\Psi \otimes \Psi_R \rhd P \dot{\sim} Q$, we know that the frames $(\mathcal{F}\ P) \otimes (\Psi \otimes \Psi_R)$ and $(\mathcal{F}\ Q) \otimes (\Psi \otimes \Psi_R)$ are statically equivalent. Hence, we need a frame switching lemma for equivalent frames to enable the transition $\alpha$. Second, we need to prove that the derivatives are in the candidate relation, i.e., that $(\Psi,\ P \mid R',\ Q \mid R') \in \mathscr{X}$, This requires $\Psi \otimes \Psi'_R \rhd P \dot{\sim} Q$ but we only know that $\Psi \otimes \Psi_R\ P \dot{\sim} Q$. Hence, we need a derivative frame lemma to prove that whenever two processes are bisimilar in the frame of an agent, they are also bisimilar in the frame of any derivative.

The frame switching lemma is relatively straightforward.

**Lemma 29** (Frame switching)

$$\frac{\left(\begin{array}{c} \Psi_F \rhd P \xrightarrow{\alpha} P' \qquad \mathscr{F}\, P = (\nu A_P)\Psi_P \qquad \mathit{distinct}\, A_P \\ ((\nu A_F)\Psi_F \otimes \Psi_P) \leq ((\nu A_G)\Psi_G \otimes \Psi_P) \\ A_F \sharp P \qquad A_G \sharp P \qquad A_F \sharp \mathit{subject}\, \alpha \qquad A_G \sharp \mathit{subject}\, \alpha \\ A_P \sharp A_F \qquad A_P \sharp A_G \qquad A_P \sharp \Psi_G \end{array}\right)}{\Psi_G \rhd P \xrightarrow{\alpha} P'}$$

*Proof* By frame induction using the lemma from Fig. 4 on the transition $\Psi_F \rhd P \xrightarrow{\alpha} P'$ with the frame $\mathscr{F}\, P = (\nu A_P)\Psi_P$. The reason why the assumption $((\nu A_F)\Psi_F \otimes \Psi_P) \leq ((\nu A_G)\Psi_G \otimes \Psi_P)$ includes the frame of $P$ is that any assertion which comprises that frame will be available to the environment when the action $\alpha$ is derived from a prefix in $P$. □

The second lemma, which allows us to construct the frame of the derivative of a transition, is one of the most complex lemmas in our mechanisation. The complexity stems from the fact that the frame of a derivative may contain free names that are bound in the original agent and opened by the transition. Since these names are no longer bound, the alpha-converting permutation can no longer be hidden by the binders of the derivative but must be made explicit. The lemma is parametrised by two freshness contects, $\mathscr{C}$ and $\mathscr{C}'$, for the bound names of the derivative frame and the bound names of the transition respectively.

**Lemma 30** (Constructing the frame of a derivative)

$$\frac{\begin{array}{c} \Psi \rhd P \xrightarrow{\alpha} P' \qquad \mathscr{F}\, P = (\nu A_P)\Psi_P \qquad \mathit{distinct}\, A_P \\ bn\,\alpha \sharp \mathit{subject}\, \alpha \qquad \mathit{distinct}\,(bn\,\alpha) \qquad A_P \sharp \alpha \qquad A_P \sharp P \\ A_P \sharp \mathscr{C} \qquad A_P \sharp \mathscr{C}' \qquad bn\,\alpha \sharp P \qquad bn\,\alpha \sharp \mathscr{C}' \end{array}}{\begin{array}{l} \exists p\ \Psi'\ A_{P'}\ \Psi_{P'}.\ \mathit{set}\, p \subseteq \mathit{set}\,(bn\,\alpha) \times \mathit{set}\,(bn\,(p \cdot \alpha)) \ \wedge\ \mathit{distinctPerm}\, p \ \wedge \\ \qquad (p \cdot \Psi_P) \otimes \Psi' \simeq \Psi_{P'} \ \wedge\ \mathscr{F}\, P' = (\nu A_P{}')\Psi_{P'} \ \wedge\ \mathit{distinct}\, A_P{}' \wedge \\ \qquad A_P{}' \sharp P' \ \wedge\ A_P{}' \sharp \alpha \ \wedge\ A_P{}' \sharp p \cdot \alpha \wedge A_P{}' \sharp \mathscr{C} \ \wedge \\ \qquad bn\,(p \cdot \alpha) \sharp \mathscr{C}' \wedge bn\,(p \cdot \alpha) \sharp \alpha \ \wedge\ bn\,(p \cdot \alpha) \sharp P' \end{array}}$$

*Proof* By frame induction using the lemma from Fig. 4 on the transition $\Psi \rhd P \xrightarrow{\alpha} P'$ with the frame $\mathscr{F}\, P = (\nu A_P)\Psi_P$. □

This lemma is then used in conjuction with the EXTENSION property of bisimilarity to ensure that $\Psi \otimes \Psi'_R \rhd P \dot{\sim} Q$.

Cases 3 and 4 are symmetric, and we focus on Case 3. Since $P$ and $Q$ are bisimilar, there is a derivative $P'$ such that $\Psi \otimes \Psi_R \rhd P \xrightarrow{\overline{M}N} P'$ and $\Psi \otimes \Psi_R \rhd P' \dot{\sim} Q'$. We prove that $S = (\nu \widetilde{x})(P'|R')$ is the desired witness. In order to derive a communication $\Psi \rhd P \mid R \xrightarrow{\tau} (\nu \widetilde{x})(P'|R')$ we need to know that $\Psi \otimes \Psi_P \rhd R \xrightarrow{\overline{K}(\nu \widetilde{x})N} R'$ and $\Psi \otimes (\Psi_P \otimes \Psi_R) \vdash M \leftrightarrow K$, but we only know that $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\overline{K}(\nu \widetilde{x})N} R'$ and $\Psi \otimes (\Psi_Q \otimes \Psi_R) \vdash M \leftrightarrow K$. The frame switching lemma from Case 2 is not sufficient here, but we need a frame/channel switching lemma that obtains a new channel $K'$ that is equivalent to $M$, and simultaneously

replaces $\Psi_P$ for $\Psi_Q$ and $K$ for $K'$ in both $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\overline{K}(\nu \tilde{x})N} R'$ and $\Psi \otimes (\Psi_Q \otimes \Psi_R) \vdash M \leftrightarrows K$. Finding the correct formulation of this lemma was one of the hardest parts of the manual proofs, since there are many similar and simpler variants of it that turned out to be insufficient. Its role in the manual proof is discussed in [9, Lemma 5.11].

**Lemma 31** (Simultaneous frame/channel switching)

$$\frac{\left(\begin{array}{c} \Psi \otimes \Psi_Q \rhd R \xrightarrow{\overline{K}(\nu \tilde{x})N} R' \qquad \Psi \otimes \Psi_R \rhd P \xrightarrow{ML} P' \\ \\ \Psi \otimes (\Psi_Q \otimes \Psi_R) \vdash M \leftrightarrow K \\ \\ ((\nu A_Q)(\Psi \otimes \Psi_Q) \otimes \Psi_R) \leq ((\nu A_P)(\Psi \otimes \Psi_P) \otimes \Psi_R) \\ \mathcal{F}\, P = (\nu A_P)\Psi_P \qquad \mathcal{F}\, Q = (\nu A_Q)\Psi_Q \qquad \mathcal{F}\, R = (\nu A_R)\Psi_R \\ \\ \begin{array}{ccccc} \mathit{distinct}\, A_P & \mathit{distinct}\, A_R & A_R \sharp A_P & A_R \sharp A_Q & \\ A_R \sharp \Psi & A_R \sharp P & A_R \sharp Q & A_R \sharp R & A_R \sharp K \\ A_P \sharp \Psi & A_P \sharp R & A_P \sharp P & A_P \sharp M & A_Q \sharp R \qquad A_Q \sharp M \end{array} \end{array}\right)}{\exists K'.\, \Psi \otimes \Psi_P \rhd R \xrightarrow{\overline{K'}(\nu \tilde{x})N} R' \wedge \Psi \otimes (\Psi_P \otimes \Psi_R) \vdash M \leftrightarrows K' \wedge A_R \sharp K'}$$

*Proof* By frame induction using the lemma from Fig. 4 on the transition $\Psi \otimes \Psi_Q \rhd R \xrightarrow{\overline{K}(\nu \tilde{x})N} R'$ with the frame $R = (\nu A_R)\Psi_R$. □

Finally, in order to prove that the derivatives are in the candidate relation, the derivative frame lemma from Case 2 (Lemma 30) is employed again.

To summarise, the lemmas required are a frame switching lemma to replace equivalent frames in transitions (Lemma 29), a derivative frame lemma to replace the environment of a bisimilarity with any derivative environment (Lemma 30), and a frame/channel switching lemma to simultaneously replace equivalent frames in channel equivalence entailment when agents communicate (Lemma 31).

## 8 Strong Equivalence

In the previous section, we proved that bisimilarity is preserved by all constructors of the *psi* data type except *Input* (Theorem 3). Our formal development [8] also contains proofs that bisimilarity is reflexive, symmetric, and transitive, hence an equivalence relation. In a similar way as for the pi-calculus [10], we now obtain a congruence relation on agents by closing bisimilarity under substitutions. This congruence is called *strong equivalence*.

### 8.1 Sequential Substitution

For the pi-calculus, the standard way of defining strong equivalence is to close bisimilarity under single substitutions. This can be made to work for psi-calculi as well, but it requires extra axioms for substitution types (Section 4.3.1) that detail how empty substitutions behave, and when a substitution can be split into several smaller ones. In order to avoid these extra axioms, we define strong equivalence for psi-calculi by closing strong

bisimilarity under sequences of parallel substitutions. A parallel substitution (Definition 7) is specified by a pair consisting of a list of names and a list of terms. Sequential substitutions are modelled as lists of such pairs.

**Definition 26** (Sequential substitution) The sequential substitution $\sigma$ applied to a term $X$ of substitution type is denoted $X\sigma$.

$$X\sigma \equiv foldl\,(\lambda\, Q\, (\widetilde{x}\widetilde{T}).Q[\widetilde{x} := \widetilde{T}])X\,\sigma$$

Here, *foldl* is the usual left fold operation for lists. Thus, application of a sequential substitution iterates over the list $\sigma$ of (parallel) substitutions and applies each one in turn, starting from $X$. Sequential substitution is defined for terms, assertions, conditions, and agents.

## 8.2 Closure Under Substitution

The constraints on substitution types, defined in Section 4.3.1, require that the lists of names and terms have equal length, and that the names being substituted are distinct. The following predicate characterises well-formed substitutions.

**Definition 27** (wellFormedSubst)

$$wellFormedSubst\,\sigma \quad \equiv \quad filter\,(\lambda(\widetilde{x},\ \widetilde{T}).\,\neg(|\widetilde{x}| = |\widetilde{T}| \wedge distinct\,\widetilde{x}))\,\sigma\ = \varepsilon$$

Intuitively, the predicate filters out all elements $\widetilde{x}, \widetilde{T}$ of $\sigma$ such that either $\widetilde{x}$ is not distinct, or the lengths of $\widetilde{x}$ and $\widetilde{T}$ differ. If the list of such elements in $\sigma$ is empty, the sequential substitution is well-formed.

We now define closure under substitution in a similar way as for the pi-calculus.

**Definition 28** (Closure under substitution) The closure of a relation $\mathscr{R}$ under well-formed substitutions is denoted $\mathscr{R}^s$.

$$\mathscr{R}^s \quad \equiv \quad \{(\Psi, P, Q) : \forall\sigma.\, wellFormedSubst\,\sigma \longrightarrow (\Psi, P\sigma, Q\sigma) \in \mathscr{R}\}$$

## 8.3 Strong Equivalence

Strong equivalence, denoted $\sim$, is defined by closing bisimilarity under well-formed sequential substitutions.

**Definition 29** (Strong equivalence)

$$\Psi \rhd P\ \sim\ Q \quad \equiv \quad (\Psi, P, Q) \in \dot{\sim}^s$$

It follows from this definition that strong equivalence is subsumed by strong bisimilarity.

**Lemma 32** *If* $\Psi \rhd P \sim Q$ *then* $\Psi \rhd P \mathrel{\dot{\sim}} Q$.

*Proof* The empty sequence of substitutions is well-formed. □

To prove that strong equivalence is a congruence, we must show that it is preserved by the input prefix. As for the pi-calculus, we begin by proving that under certain circumstances, simulation and strong bisimilarity are preserved by the input prefix.

**Lemma 33**

$$\forall \widetilde{T}.\ \frac{|\widetilde{x}| = |\widetilde{T}|}{\dfrac{(\Psi, P[\widetilde{x} := \widetilde{T}], Q[\widetilde{x} := \widetilde{T}]) \in \mathscr{R}}{\Psi \rhd \underline{M}(\lambda \widetilde{x})N.P \hookrightarrow_{\mathscr{R}} \underline{M}(\lambda \widetilde{x})N.Q}}$$

*Proof* Follows immediately from the definition of $\hookrightarrow$ by inversion on the input transition. □

**Lemma 34**

$$\forall \widetilde{T}.\ \frac{|\widetilde{x}| = |\widetilde{T}|}{\dfrac{\Psi \rhd P[\widetilde{x} := \widetilde{T}] \mathrel{\dot{\sim}} Q[\widetilde{x} := \widetilde{T}]}{\Psi \rhd \underline{M}(\lambda \widetilde{x})N.P \mathrel{\dot{\sim}} \underline{M}(\lambda \widetilde{x})N.Q}}$$

*Proof* By coinduction with $\mathscr{X}$ set to

$$\{(\Psi, \underline{M}(\lambda \widetilde{x})N.P, \underline{M}(\lambda \widetilde{x})N.Q) : \forall \widetilde{T}.|\widetilde{x}| = |\widetilde{T}| \longrightarrow \Psi \rhd P[\widetilde{x} := \widetilde{T}] \mathrel{\dot{\sim}} Q[\widetilde{x} := \widetilde{T}]\}$$

The simulation case is discharged using Lemma 33, and all other cases follow immediately from the elimination rules of bisimilarity (Lemma 19). □

We can now prove that strong equivalence is preserved by the input prefix.

**Lemma 35**

$$\frac{\Psi \rhd P \sim Q \qquad \widetilde{x} \# \Psi \qquad \mathit{distinct}\, \widetilde{x}}{\Psi \rhd \underline{M}(\lambda \widetilde{x})N.P \sim \underline{M}(\lambda \widetilde{x})N.Q}$$

*Proof* We need to show that $\Psi \rhd (\underline{M}(\lambda \widetilde{x})N.P)\sigma \mathrel{\dot{\sim}} (\underline{M}(\lambda \widetilde{x})N.Q)\sigma$ for all well-formed substitutions $\sigma$.

We obtain a permutation $p$ such that $\mathit{set}\ p \subseteq \mathit{set}\, \widetilde{x} \times \mathit{set}\ (p \cdot \widetilde{x})$ and $p \cdot \widetilde{x}$ is fresh for everything in the proof context. After alpha-converting and pushing $\sigma$ over the binders, we have to prove that $\Psi \rhd \underline{M\sigma}(\lambda(p \cdot \widetilde{x}))(p \cdot N)\sigma(p \cdot P)\sigma \mathrel{\dot{\sim}} \underline{M\sigma}(\lambda(p \cdot \widetilde{x}))(p \cdot N)\sigma(p \cdot Q)\sigma$.

This follows from Lemma 34, provided we can show $\Psi \rhd (p \cdot P)\sigma[(p \cdot \widetilde{x}) := \widetilde{T}] \mathrel{\dot{\sim}} (p \cdot Q)\sigma[(p \cdot \widetilde{x}) := \widetilde{T}]$ for all $\widetilde{T}$ such that $|\widetilde{x}| = |\widetilde{T}|$.

From $\Psi \rhd P \sim Q$ we have $p \cdot \Psi \rhd p \cdot P \sim p \cdot Q$, and hence $\Psi \rhd p \cdot P \sim p \cdot Q$ since $\widetilde{x} \# \Psi$ and $(p \cdot \widetilde{x}) \# \Psi$. From $|\widetilde{x}| = |\widetilde{T}|$, $\mathit{distinct}\, \widetilde{x}$, and $\mathit{wellFormedSubst}\ \sigma$ we obtain $\mathit{wellFormedSubst}\ (\sigma[(p \cdot \widetilde{x}, \widetilde{T})])$. Therefore, $\Psi \rhd (p \cdot P)\sigma[(p \cdot \widetilde{x}, \widetilde{T})] \mathrel{\dot{\sim}} (p \cdot Q)\sigma[(p \cdot \widetilde{x}, \widetilde{T})]$ by the definition of $\sim$. □

Our main result about strong equivalence is the following theorem.

**Theorem 4** *Strong equivalence is a congruence relation.*

*Proof* That strong equivalence is preserved by *Input* follows from Lemma 35. That it is also preserved by the remaining constructors follows immediately from Theorem 3 and the definition of $\sim$, where any bound names are alpha-converted to avoid the substitutions □

## 9 Related Work

Taking the step from intuitive informal reasoning, in the style of Barendregt's variable convention [6], to a theory of alpha-equivalence that can be checked by computer has proven difficult. Aydemir et al. [4] give an excellent overview of the many techniques that have been devised to represent terms with binders. The four most prominent approaches in the literature are de Bruijn indices, higher-order abstract syntax, the locally nameless representation, and nominal logic. In the following discussion of related work, we focus on their application to process calculi. Solutions to the POPLmark challenge [44] provide a comparison of these techniques on a common set of benchmark problems from programming language theory.

Of these techniques, de Bruijn indices are the oldest. They were originally introduced by de Bruijn in [19], whose key idea was to represent all names by natural numbers that indicate the nesting level of the corresponding binder. With this representation, alpha-equivalent terms are syntactically equal. De Bruijn indices have proven useful for automated tools that reason about binders, but they are cumbersome when used in interactive proofs. The main problem appears when modifying the structure of terms, e.g., by applying a substitution, requires the recalculation of indices. Moreover, the representation is not intuitive for humans. Nevertheless, de Bruijn indices have been used successfully for large-scale formalisations in interactive proof assistants. In [26], Hirschkoff formalises a substantial part of the meta-theory of the pi-calculus in Coq. Of roughly 800 proved lemmas, 600 are concerned with manipulation of de Bruijn indices. Briais' formalisation of the spi-calculus in Coq [18] suffers from similar technical tedium.

Higher-order abstract syntax (HOAS) treats binders as functions from names to terms. This approach leaves all reasoning about bound names to the meta-theory of the logic. Alpha-equivalence is thereby obtained for free. On the other hand, it becomes impossible to reason specifically about bound names, as they are hidden by function abstractions. HOAS has been used to model the pi-calculus both in Coq, by Honsell et al. [28], and in Isabelle, by Röckl and Hirschkoff [46]. Earlier strenuous efforts to encode the pi-calculus in the HOL proof assistant [34, 41] used explicit names, and a manual definition of alpha-equivalence.

The locally nameless representation [22] employs de Bruijn indices to represent bound variables, but retains names for free variables. Again, alpha-equivalent terms are syntactically equal. Moreover, substitution and beta-reduction have much simpler definitions than with a pure de Bruijn representation. Aydemir et al. applied this approach to reason about core ML and other calculi in Coq [4]. They observe that the locally nameless representation, when combined with cofinite quantification over free names, leads to "developments that are faithful to informal practice, yet require no external tool support and little infrastructure within the proof assistant." However, in the absence of such infrastructure, key lemmas must be proved manually.

Recently, de Vries and Koutavas [51] suggested a combination of the locally nameless approach with permutation types (as used in Nominal Isabelle). This could simplify the

| Topic | Lines of code | Percentage |
|---|---|---|
| Basic nominal lemmas | 1,240 | 4% |
| Substitution, agents, frames | 3,379 | 10% |
| Operational semantics | 9,322 | 29% |
| Strong bisimilarity | 2,828 | 9% |
| Structural congruence | 2,805 | 9% |
| Weak bisimilarity | 7,889 | 24% |
| Structural congruence | 613 | 2% |
| Weak congruence | 1,633 | 5% |
| Structural congruence | 285 | 1% |
| Simplified weak bisimilarity | 774 | 2% |
| Tau-laws | 1090 | 3% |
| Other results and extensions | 419 | 1% |
| Total | 32,277 | 100% |

**Fig. 7** Size of different parts of the psi-calculi formalisation in Nominal Isabelle. The lion's share of the structural congruence proofs is done for strong equivalence; later congruence results follow as a corollary because strong equivalence is subsumed by all other versions of bisimilarity. Simplified weak bisimilarity is a simpler version of weak bisimilarity; the two are equivalent when the weakening axiom $\Psi \leq \Psi \otimes \Psi'$ holds

treatment of extruded names, for instance, in the OPEN rule of our operational semantics, which we currently address through residuals.

In the Isabelle proof assistant, infrastructure for reasoning about binders is available in the form of Urban's Nominal Isabelle [48] framework. The framework is based on nominal logic, originally devised by Gabbay and Pitts [24, 45]. Nominal logic, described in more detail in Section 2.1, is a first-order theory of names and binding that builds on name swapping as a primitive concept. We have previously used Nominal Isabelle to formalise Milner's CCS [7] and the pi-calculus [10]. Kahsai and Miculan [33] implemented the spi-calculus in Nominal Isabelle. The psi-calculi framework presented in this paper is more expressive than either of these calculi.

Work on Nominal Isabelle continues. A recent re-implementation [29], known as Nominal2, was in large part motivated by our formalisation. It simplifies the framework's theoretical foundations, and adds built-in support for multiple binders [50]. We intend to port our formalisation of psi-calculi to Nominal2, expecting that this will considerably simplify the current treatment of multiple binders via binding sequences (Section 3). However, because Nominal2 is fundamentally different from its predecessor, this is not a straightforward task. As a first step, we plan to enhance Nominal2 with support for Isabelle's *locales* [5], which our formalisation uses extensively to achieve parametricity. Until this has been achieved, the former version of Nominal Isabelle—which, despite being limited to single binders, remains the version that is bundled with the Isabelle proof assistant [31]—also remains the framework of choice for our formalisation.

## 10 Conclusion

The psi-calculi framework is the most advanced process calculus framework to date. It is expressive, it is general, and it has a simple semantics. In this paper, we presented the formalisation of its meta-theory in Isabelle.

A fully formalised framework has several benefits. The most obvious one is that we know with certainty (relative to the soundness of Isabelle) that our theorems are correct. This claim is not to be taken lightly, since there is a clear need for robust theories. As the complexity of process calculi increases, so does the complexity of proofs about them. During our formalisation efforts, we found errors in the published meta-theory of other popular

process calculi [9]. The Isabelle formalisation precludes the kind of human oversight that happens all too often in complex pen-and-paper proofs.

Another benefit is that formalised theories are extensible. The ramifications of changes are instantly apparent, making it safe to modify the calculus without risking inconsistencies. While pen-and-paper proofs would have to be carefully reexamined, formal proofs can be checked mostly automatically, sometimes in minutes [7]. The psi-calculi formalisation described here has already been used as a basis for the implementation of various extensions of psi-calculi in Isabelle, notably broadcast psi-calculi [17], higher-order psi-calculi [43], and sorted psi-calculi [16].

What sets our formalisation apart from formalisations of other process calculi is that the theory of psi-calculi was developed simultaneously with the formalisation. This had advantages and disadvantages. Theory development is an intricate process, where new insights invariably lead to changes that must then be mirrored also in the formalisation. Fortunately, the amount of backtracking required for psi-calculi was tolerable. One change was severe. We had finished the formalisation; all proofs were done; strong bisimilarity was proven to be a congruence. At the time, requisites on entailment were formulated in terms of frames, but it turned out to be too difficult to develop instances of the framework. This led to the current design, with requisites on the entailment relation in terms of assertions (Section 4.4). Accordingly, a nearly complete rewrite of the semantics and Isabelle theories was necessary. The lesson learned is that a proof assistant will only prove theorems correct, it will not determine their relevance.

On the other hand, the chances of finding bugs early in the theoretical development increase with the use of a proof assistant, as the proofs are constantly being verified. In this there is a similarity to rapid prototyping in software development, where design bugs are weeded out by experiments as early as possible. Formalising the proofs for psi-calculi in parallel with the theoretical development has turned out to be invaluable, and we would most likely not have finished the latter successfully without it. Uncountable times during formalisation we stumbled over slightly incorrect definitions and lemmas, prompting frequent (if minor) changes in the theoretical framework. Some errors escaped careful manual revision and were published [32], before we ultimately detected them with the help of Isabelle [7].

Machine-checked formalisations also encourage developers to keep theories simple, thereby serving as a version of Occam's razor. The simpler the theories, the easier they are to formalise, and the easier they are to use. A good example of this is how psi-calculi treat the entailment of conditions. Another example is that we prefer calculi without structural congruence in their semantics; the bugs that we found in other process calculi involved structural congruence in one way or another.

For our formalisation efforts, nominal logic has worked exceptionally well. One of its main benefits is that it provides reasoning about binders without referring to any particular structure of the nominal data type. The arbitrary binding schemes that we touched on in the previous sections also follow this notion, since alpha-equivalence and alpha-conversion lemmas can be established independently of the exact structure of the data type. Reasoning about alpha-equivalence with single binders is relatively well understood, but for psi- and other more complex calculi, the ability to reason about multiple binders is an essential requirement. We have shown that nominal logic, and in particular its implementation in Nominal Isabelle, are well suited for this task.

Our theory files, which comprise approximately 32,000 lines of definitions and proofs, are available from the Archive of Formal Proofs [8]. Figure 7 summarises the size of different parts of the formalisation in more detail. In particular, the formalisation of the operational semantics is significantly larger than for our previous formalisations of CCS and the

pi-calculus [10]. One reason is that much of the extra infrastructure for induction and inversion rules, which Nominal Isabelle derives automatically for the simpler calculi, must be set up manually for psi-calculi. The time spent on formalisation (including backtracking due to changes in the simultaneously developed theoretical framework) was roughly two years.

The congruence result for weak bisimilarity re-uses definitions and preservation lemmas that were originally developed for strong bisimilarity. Therefore, it appears smaller in size, despite being technically more challenging. For space reasons, we cannot present the Isabelle formalisation of our results for weak bisimilarity and weak congruence in detail. A discussion of the relevant proof techniques can be found in [10].

Our formalisation occasionally pushed Nominal Isabelle beyond its limits. We benefited from subsequent enhancements to the framework, which continues to be developed today. Improved automation allowed us to remove thousands of lines of proof script from our Isabelle theories.

Our next steps will be to further extend the meta-theory of psi-calculi, and to develop tools that support the verification of programs and protocols expressed as psi-calculi agents. Ideally, these tools would be verified with a proof assistant as well.

**Acknowledgments** We want to convey our sincere thanks to Stefan Berghofer for his hard work on enhancing Nominal Isabelle to include the features that we needed for this formalisation.

## References

1. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. ACM SIGPLAN Not. **36**(3), 104–115 (2001)
2. Abadi, M., Gordon, A.D.: A calculus for cryptographic protocols: The spi calculus. Inf. Comput. **148**, 36–47 (1999)
3. Aydemir, B.E., Bohannon, A., Fairbairn, M., Foster, N.J., Pierce, B.C., Sewell, P., Vytiniotis, D., Washburn, G., Weirich, S., Zdancewic, S.: Mechanized metatheory for the masses: The POPLmark challenge. In: Hurd, J., Melham, T. (eds.): Proceedings TPHOLs 2005, LNCS, vol. 3603, pp. 50–65. Springer (2005)
4. Aydemir, B.E., Charguéraud, A., Pierce, B.C., Pollack, R., Weirich, S.: Engineering formal metatheory. In: Necula, G.C., Wadler, P. (eds.): Proceedings POPL 2008, pp. 3–15. ACM (2008)
5. Ballarin, C.: Locales and locale expressions in Isabelle/Isar. In: Berardi, S., Coppo, M., Damiani, F. (eds.): Types for Proofs and Programs, International Workshop, TYPES 2003, Torino, Italy, April 30 – May 4, 2003, Revised Selected Papers, LNCS, vol. 3085, pp. 34–50. Springer (2003)
6. Barendregt, H.P.: The lambda calculus : its syntax and semantics. North-Holland Pub. Co (1981)
7. Bengtson, J.: Formalizing process calculi. Ph.D. thesis, Uppsala Universitet (2010)
8. Bengtson, J.: Psi-calculi in Isabelle. Archive of Formal Proofs. http://afp.sf.net/entries/Psi_Calculi.shtml, Formal proof development (2012)
9. Bengtson, J., Johansson, M., Parrow, J., Victor, B.: Psi-calculi: a framework for mobile processes with nominal data and logic. Logical Methods in Computer Science **7**(1) (2011)
10. Bengtson, J., Parrow, J.: Formalising the pi-calculus using nominal logic. Logical Methods in Computer Science **5**(2) (2008)
11. Bengtson, J., Parrow, J.: Psi-calculi in Isabelle. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.): Proceedings TPHOLs 2009, LNCS, vol. 5674, pp. 99–114. Springer (2009)
12. Berghofer, S.: Simply-typed lambda-calculus with let and tuple patterns. http://isabelle.in.tum.de/repos/isabelle/file/81e8fdfeb849/src/HOL/Nominal/Examples/Pattern.thy. Retrieved on October 1, 2013 (2010)
13. Berghofer, S., Urban, C.: Nominal inversion principles. In: Mohamed, O.A., Muñoz, C.A., Tahar, S. (eds.): Proceedings TPHOLs '08, LNCS, vol. 5170, pp. 71–85. Springer (2008)
14. Bergstra, J.A., Klop, J.W.: Process algebra for synchronous communication. Inf. Control. **60**(1–3), 109–137 (1984)
15. Bertot, Y.: A short presentation of Coq. In: Mohamed, O.A., Muñoz, C., Tahar, S. (eds.): Proceedings TPHOLs 2008, LNCS, vol. 5170, pp. 12–16. Springer (2008)

16. Borgström, J., Gutkovas, R., Parrow, J., Victor, B., Pohjola, J.Å.: Sorted psi-calculi with generalised pattern matching. To appear in LNCS 8358, Proceedings of TGC (2013)
17. Borgström, J., Huang, S., Johansson, M., Raabjerg, P., Victor, B., Pohjola, J.Å., Parrow, J.: Broadcast psi-calculi with an application to wireless protocols. In: Barthe, G., Pardo, A., Schneider, G. (eds.): Proceedings SEFM 2011, LNCS, vol. 7041, pp. 74–89. Springer (2011)
18. Briais, S.: A formalisation of the spi calculus in Coq (2007). Email to the Coq-club mailing list sent on Nov 2, 2007. Retrieved from, http://permalink.gmane.org/gmane.science.mathematics.logic.coq.club/1865 on October 1, 2013
19. de Bruijn, N.G.: Lambda calculus notation with nameless dummies. A tool for automatic formula manipulation with application to the Church-Rosser theorem. Indag. Math. **34**, 381–392 (1972)
20. Buscemi, M.G., Montanari, U.: CC-Pi: A constraint-based language for specifying service level agreements. In: De Nicola, R. (ed.): Proceedings ESOP 2007, LNCS, vol. 4421, pp. 18–32. Springer (2007)
21. Carbone, M., Maffeis, S.: On the expressive power of polyadic synchronisation in $\pi$-calculus. Nordic Journal of Computing **10**(2), 70–98 (2003)
22. Charguéraud, A.: The locally nameless representation. J. Autom. Reason., 1–46 (2011)
23. Church, A.: An unsolvable problem of elementary number theory. Am. J. Math. **58**(2), 345–363 (1936)
24. Gabbay, M.J., Pitts, A.M.: A new approach to abstract syntax with variable binding. Form. Asp. Comput. **13**, 341–363 (2001)
25. Gardner, P., Wischik, L.: Explicit fusions. In: Nielsen, M., Rovan, B. (eds.): Proceedings MFCS 2000, LNCS, vol. 1893, pp. 373–382. Springer (2000)
26. Hirschkoff, D.: A full formalisation of pi-calculus theory in the calculus of constructions. In: Gunter, E.L., Felty, A.P. (eds.): Proceedings TPHOLs '97, LNCS, vol. 1275, pp. 153–169. Springer (1997)
27. Hoare, C.A.R.: Communicating sequential processes. Commun ACM **21**(8), 666–677 (1978)
28. Honsell, F., Miculan, M., Scagnetto, I.: pi-calculus in (co)inductive-type theory. Theor. Comput. Sci. **253**(2), 239–285 (2001)
29. Huffman, B., Urban, C.: A new foundation for Nominal Isabelle. In: Kaufmann, M., Paulson, L.C. (eds.): Proceedings ITP 2010, LNCS, vol. 6172, pp. 35–50. Springer (2010)
30. Hur, C.K., Neis, G., Dreyer, D., Vafeiadis, V.: The power of parameterization in coinductive proof. In: Giacobazzi, R., Cousot, R. (eds.): The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013, pp. 193–206. ACM (2013)
31. Isabelle: Retrieved from, http://isabelle.in.tum.de/ on October 1, 2013 (2013)
32. Johansson, M., Parrow, J., Victor, B., Bengtson, J., Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I.: Extended pi-calculi: Proceedings ICALP 2008, LNCS, vol. 5126, pp. 87–98. Springer (2008)
33. Kahsai, T., Miculan, M.: Implementing spi calculus using nominal techniques. In: Beckmann, A., Dimitracopoulos, C., Löwe, B. (eds.): Proceedings CiE 2008, LNCS, vol. 5028, pp. 294–305. Springer (2008)
34. Melham, T.F.: A mechanized theory of the pi-calculus in HOL. Nordic Journal of Computing **1**(1), 50–76 (1994)
35. Milner, R.: A Calculus of Communicating Systems, LNCS, vol. 92. Springer (1980)
36. Milner, R.: Communication and Concurrency. Prentice-Hall, Inc (1989)
37. Milner, R.: The polyadic pi-calculus: a tutorial. In: Bauer, F.L., Brauer, W., Schwichtenberg, H. (eds.): Logic and Algebra of Specification, pp. 203–246. Springer (1993)
38. Milner, R.: Communicating and mobile systems - the Pi-calculus. Cambridge University Press (1999)
39. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, I/II. Inf. Comput. **100**(1), 1–77 (1992)
40. Milner, R., Tofte, M., Harper, R., MacQueen, D.: The Definition of Standard ML – Revised. MIT Press (1997)
41. Mohamed, O.A.: The theory of the pi-calcul in HOL. Ph.D. thesis, Henri Poincare University (1996)
42. Park, D.M.R.: Concurrency and automata on infinite sequences. In: Deussen, P. (ed.): Proceedings OF the LNCS Theoretical Computer Science, 5th GI-Conference, Karlsruhe, Germany, March 23–25, 1981, vol. 104, pp. 167–183. Springer (1981)
43. Parrow, J., Borgström, J., Raabjerg, P., Åman Pohjola, J.: Higher-order psi-calculi. Math. Struct. Comput. Sci. **FirstView**, 1–37 (2013). doi:10.1017/S0960129513000170
44. Pierce, B.C., Weirich, S.: Preface. J. Autom. Reason. **49**(3), 301–302 (2012)
45. Pitts, A.M.: Nominal logic, a first order theory of names and binding. Inf. Comput. **186**(2), 165–193 (2003)
46. Röckl, C., Hirschkoff, D.: A fully adequate shallow embedding of the $\pi$-calculus in Isabelle/HOL with mechanized syntax analysis. J. Funct. Program. **13**(2), 415–451 (2003)

47. Slind, K., Norrish, M.: A brief overview of HOL4. In: Mohamed, O.A., Muñoz, C., Tahar, S. (eds.): Proceedings TPHOLs 2008, LNCS, vol. 5170, pp. 28–32. Springer (2008)
48. Urban, C.: Nominal techniques in Isabelle/HOL. J. Autom. Reason. **40**(4), 327–356 (2008)
49. Urban, C., Berghofer, S., Norrish, M.: Barendregt's variable convention in rule inductions. In: Pfenning, F. (ed.): Proceedings CADE-21, LNCS, vol. 4603, pp. 35–50. Springer (2007)
50. Urban, C., Kaliszyk, C.: General bindings and alpha-equivalence in Nominal Isabelle. Logical Methods in Computer Science **8**(2) (2012)
51. de Vries, E., Koutavas, V.: Locally nameless permutation types. Submitted. Retrieved from, https://www.cs.tcd.ie/Edsko.de.Vries/pub/lnpt.pdf on October 1, 2013
52. Wenzel, M., Paulson, L.C., Nipkow, T.: The Isabelle framework. In: Mohamed, O.A., Muñoz, C., Tahar, S. (eds.): Proceedings TPHOLs 2008, LNCS, vol. 5170, pp. 33–38. Springer (2008)
53. Wenzel, M., et al.: The Isabelle/Isar Reference Manual. Retrieved from http://isabelle.in.tum.de/dist/Isabelle2013/doc/isar-ref.pdf on October 1, 2013 (2013)