Extended pi-Calculi

Magnus Johansson, Joachim Parrow, Björn Victor, and Jesper Bengtson

Department of Information Technology, Uppsala University

Abstract. We demonstrate a general framework for extending the pi-calculus with data terms. In this we generalise and improve on several related efforts such as the spi calculus and the applied pi-calculus, also including pattern matching and polyadic channels. Our framework uses a single untyped notion of agent, name and scope, an operational semantics without structural equivalence and a simple definition of bisimilarity. We provide general criteria on the semantic equivalence of data terms; with these we prove algebraic laws and that bisimulation is preserved by the operators in the usual way. The definitions are simple enough that an implementation in an automated proof assistant is feasible.

1 Introduction

The pi-calculus [1] is a foundational calculus for describing communicating systems with dynamic connectivity. Allowing only names of communication channels to be transmitted between processes, it is still expressive enough to encode other types of data such as booleans, integers, lists etc. Such data structures are convenient when modelling protocols, programming languages, and other complicated applications. However, having to work with encodings muddles the models and complicates their analysis – and constructing correct encodings can be quite difficult. To overcome these difficulties, extensions of the pi-calculus have been introduced, where higher-level data structures (and operations on them) are given as primitive. Our contribution in this paper is to establish a unifying untyped framework where a range of such calculi can be formulated, using a lean and symmetric semantics that is well suited for an automated proof assistant.

Perhaps the simplest and oldest of data structures for the pi-calculus is to allow tuples of names to be transmitted, leading to the polyadic pi-calculus [2] and its typed variants. This is a quite mild extension where for example the agent (or process) $\overline{a}\langle b, c\rangle \,.\, P$ can send the two names b and c to a receiving $a(x, y) \,.\, Q$ in one go. This needs to be faithful to the pi-calculus principle of scope extrusion, which says that if a name is transmitted outside its scope then that scope is extended to include the receiver. If for example both b and c are scoped, as in $(\nu b, c)\overline{a}\langle b, c\rangle \,.\, P$, then the scope of both b and c are extended to encompass Q. In this way the effect is the same as sending b and c individually in subsequent transmissions.

With subsequent advances of calculi for cryptographic applications another view has emerged, where a fresh key is represented as a scoped name, and a data structure such as $\mathrm{enc}(m, k)$ represents the encryption of m by the key k. Two main examples are the spi calculus [3] and the applied pi-calculus [4].

The spi calculus is equipped with encryption/decryption primitives. An example process

$$P = (\nu k, m)\overline{a}\langle \mathrm{enc}(m, k)\rangle \,.\, P' \qquad \text{where} \quad P' = b(x) \,.\, \mathbf{if}\ x = m\ \mathbf{then}\ \overline{c}\langle m\rangle$$

sends out, over a, the fresh name m encrypted with the fresh key k, and then receives a value x over b. If the received value is m, an output is sent on c. Intuitively, since the environment does not know k (or m), P' can never receive m, and the output on c can never happen.

However, a labelled transition for the process is $P \xrightarrow{(\nu k, m)\overline{a}\langle \mathrm{enc}(m,k)\rangle} P'$. Although the label contains the names m and k, and therefore their scopes are opened by the transition, the names should not become known to the environment since it cannot decrypt the message and find them. Thus, any reasonable equivalence must explicitly keep track of which names are known. There are several bisimulation equivalences of varying complexity for the spi calculus – see [5] for an overview.

The applied pi-calculus [4] improves the situation in several ways. Firstly, it is more general since the calculus is parameterised by a signature and an equation system for data structures. Secondly, and more importantly, the transition labels expose only what should be revealed. This is achieved by introducing variables with so called *active substitutions* $\{^{M}/_{x}\}$ of data terms for variables, the structural rule $\{^{M}/_{x}\} \mid P \equiv \{^{M}/_{x}\} \mid P[x := M]$, and by disallowing the sending of complex data directly – it must be sent using an *alias* variable such as x. Let us return to the process P above; we have $P \equiv Q$ where

$$Q = (\nu k, m, z)(\{^{\mathrm{enc}(m,k)}/_{z}\} \mid \overline{a}\langle z\rangle \,.\, P') \xrightarrow{(\nu z)\overline{a}\langle z\rangle} (\nu k, m)(\{^{\mathrm{enc}(m,k)}/_{z}\} \mid P')$$

Here, the transition label only reveals that a (fresh) value is sent, but not how the value is constructed; the scope of k and m is not opened, and their confidentiality to the environment is clear; the bisimulation definition is correspondingly simple. However, sending out a tuple of channel names as in the polyadic pi-calculus is not possible in the labelled semantics of the applied pi-calculus – see Section 3.

In this paper we define a generalisation of the (applied) pi-calculus where we allow *both* possibilities: data terms can be sent using an alias, not revealing the construction of the value, or as the text or syntax tree for the term, exposing all details. The process needs to explicitly use one form or the other, unlike the case in the applied pi-calculus where a term in an object position is always implicitly rewritten using an alias. Which form to use is up to the intended purpose of the term: to keep secrets the alias form must be used, while to extrude the scope of e.g. communication channels, the explicit form should be used. P and Q are not at all equivalent, and the structural rule from the applied pi-calculus mentioned above is neither needed nor valid. It is even possible to mix the different modes

by including an alias in an explicit data structure; the parts represented by the alias remain hidden.

Our new framework allows pattern matching inputs: e.g., $a(f(x,y)).P'$ can only communicate with an output $\overline{a}\langle f(M,N)\rangle$ for some data terms M, N. This can be seen as a generalisation of the polyadic pi-calculus where inputs of a certain arity can only communicate with outputs of the same arity. Another significant extension is that we allow arbitrary data terms also as communication channels. Thus it is possible to include functions that create channels and also polyadic synchronisation, cf. [6], where a channel may be composed of several names.

A reader might fear that to achieve all this the framework must be a complicated union of all work cited above, with a plethora of primitives. This is not the case. We use a single basic framework for extended pi-calculi, with a single untyped notion of agent and a single-level structural operational semantics that does not rely on any structural equivalence to rewrite agents, a single notion of name and scope, and a simple definition of bisimilarity. We provide general criteria on the semantic equivalence of data terms; with these we prove algebraic laws and that bisimulation is preserved by the operators in the usual way. Our framework facilitates comparisons between different approaches, and proofs about the calculi can be conducted using straightforward inductions over transitions. In this way we improve on existing definitions of the applied pi-calculus. The framework is lean enough that it is profitable to use an automated proof assistant and our preliminary efforts in this direction (using Isabelle [7]) hold promise.

Related Work. The different bisimulation definitions for the spi calculus are presented in [8–10, 5] and an overview can be found in [5]. Another example of an environment sensitive equivalence can be found in [11]. In [12], a spi calculus parameterised by a data signature and evaluation function is defined. The relation between the applied pi-calculus and extended pi-calculi is explored in Section 3.

The applied pi-calculus has been used for analysis of several security protocols, e.g. [13–15]. Ad-hoc variants of the applied pi-calculus have also been used to prove non-security properties, e.g. in [16] correctness of network based storage is shown using a variant of an applied pi-calculus with polyadicity (but without labelled semantics).

Polyadic synchronisation in the pi-calculus [6] is a subcalculus of an extended pi-calculi as discussed in Section 3. In [17] a Spi calculus with pattern matching is presented.

Disposition. In Section 2 we present the syntax and operational semantics of extended pi-calculi. In Section 3, we illustrate by examples how these can be put to use, and relate informally to other variants of the pi-calculus. Section 4 presents the labelled bisimulation relation, and our main results about it. Section 5 concludes with suggestions for further work.

2 Definitions

Assume a signature with a set of *symbols* f, g, each with a nonnegative arity. Among the symbols there is a countably infinite set of *names* $a, b, \ldots$, all with arity 0. (There may also be other symbols with arity 0.)

Definition 1 (Terms). *The* data terms*, or for short just* terms*, ranged over by* $M, N, \ldots$*, are defined by* $M ::= f(M_1, \ldots, M_n)$ *where* f *has arity* n*.*

If f has arity 0 we write just f for $f()$; in particular a name is a term. The *substitution* of a term N for a name a in a term M, written $M[a := N]$ is defined by $a[a := N] = N$ and $f \neq a \Longrightarrow f[a := N] = f$, extending homomorphically to all terms.

In the following we shall define agents, actions, transitions etc. in the style of nominal datatypes [18] building on our previous experience of nominal types for the π-calculus [19]. It is not necessary to understand nominal datatypes to appreciate the work presented here; suffice it to say that all alpha-equivalent agents, actions etc. are considered equal.

The set of free names (i.e. the names with an non-bound occurrence) of an object X is written as $\mathrm{fn}(X)$, and we abbreviate $\mathrm{fn}(X) \cup \mathrm{fn}(Y)$ to $\mathrm{fn}(X, Y)$ etc. We write $\mathrm{n}(X)$ for the names (bound and free) in X. We also write $a\#X$ ("a is fresh in X") to mean $a \notin \mathrm{fn}(X)$. Name permutation of a and b (exchanging all a for b and vice versa) in P is written $(a\,b) \bullet P$. In the following $\tilde{a}$ means a finite (possibly empty) sequence of distinct names, $a_1, \ldots, a_n$. When occurring as an operand of a set operator, $\tilde{a}$ means the corresponding set of names $\{a_1, \ldots, a_n\}$. Concatenation of sequences $\tilde{a}$ and $\tilde{b}$ is written $\tilde{a} \cdot \tilde{b}$ and the empty sequence is written ϵ.

Definition 2 (Agents and aliased names). *The* agents*, ranged over by* P, Q*, are the following:*

$\mathbf{0}$	Nil	$(\nu a)P$	Restriction
$\overline{M}\,N.P$	Output	$P \mid Q$	Parallel
$M(N).P$	Input	$!P$	Replication
if $M = N$ **then** P **else** Q	Conditional	$\{^{M}/_{a}\}$	Alias

We say that a *is the* aliased name *of* $\{^{M}/_{a}\}$*, and define the aliased names* $\mathrm{an}(P)$ *of an agent* P *to be the set of free names aliased in any subagent in* P*. In the input* $M(N).P$ *it is required that* $\mathrm{n}(N) \cap \mathrm{an}(P) = \emptyset$*. Input and restriction are binding occurrences of* $\mathrm{n}(N)$ *and* a*.*

All operators except Alias are familiar from the pi-calculus (we omit the sum operator but inclusion of guarded sum would not greatly affect our results). The Alias is similar in intention to what in the applied pi-calculus is called an "active substitution", though semantically it is a bit different in that our Alias will not always enforce a substitution. An aliased name cannot be bound by input. The reason is that with an input bound alias we can express aliasing of terms for

terms: $M(a).\{^N/_a\}$ can receive a term K to replace a, becoming $\{^N/_K\}$, which is not a syntactically correct agent.

Note that an aliased name is not a binder. Suppose we would restrict to a sub-calculus with a construct like "**alias** $a = M$ **in** P" to represent $(\nu a)(\{^M/_a\} \mid P)$, i.e., the construct binds a. In this sub-calculus it is impossible to directly express an agent such as $(\nu a)(((\nu k)(\{^{\mathrm{enc}(M,k)}/_a\} \mid P)) \mid Q)$. This agent could represent that a term M encrypted with key k has been sent to P and Q, and that only P has the key. So Q cannot use M until it receives k in a communication (opening the scope of k). In order to express aliases that become usable only when their "keys" are received, it seems that a binding "**alias...**" construct is not enough.

The Input construct contains an implicit pattern matching. For example $M\,(f(a,b)).P$ can input objects only of shape $f(K,L)$, thereby substituting K for a and L for b in P. We shall use the silent prefix $\tau.P$, which can be thought of as a shorthand for $(\nu a)(\overline{a}\,a.\mathbf{0} \mid a\,(c).P)$ for some $a, c\#P$, and let γ range over prefixes. A generalised restriction $(\nu\tilde{a})P$ means $(\nu a_1)\cdots(\nu a_n)P$, or just P if $n = 0$.

The *substitution* of M for a in the agent P is written $P[a := M]$, and is defined if $a \notin \mathrm{an}(P)$; in other words it is not possible to substitute something for an aliased name. Substitution is defined in the usual way, homomorphic on all operators and renaming bound names to avoid captures. When Z is a term or agent, $\widetilde{a} = a_1, \ldots a_n$ are pairwise distinct and $\widetilde{L} = L_1, \ldots, L_n$ we write $Z[\widetilde{a} := \widetilde{L}]$ to mean the simultaneous substitution of each L_i for a_i in Z.

Definition 3 (Frames). *A* frame F *is of kind* $(\nu\widetilde{a}_F)R_F$*, where* $\widetilde{a}_F$ *is a sequence of names and* R_F *is a finite relation between names and terms. The names in* $\widetilde{a}_F$ *are binding occurrences in the frame. The* domain $\mathrm{dom}(F)$ *is defined to be* $\mathrm{dom}(R_F) - \widetilde{a}_F$*, i.e. the domain of the relation but not including the bound names.*

The intuition is that a frame contains information about what terms should be considered equal, by relating aliases to terms. The bound names in a frame represent local placeholders, and cannot be contained in the terms to be compared for equality. Frames may be nondeterministic in that two different terms have the same alias (in that case the alias can represent either term) and even circular — we have found no reason to forbid such aliases even though in some contexts they would not make sense.

We abbreviate the empty frame $(\nu\epsilon)\emptyset$ to $\emptyset$ when no confusion can arise. We also write $(\nu a)F$ for $\nu(a \cdot \widetilde{a_F})R_F$, and $\{^T/_a\}$ for $(\nu\epsilon)\{(a,T)\}$, and $F \cup G$ for $\nu(\widetilde{a_F} \cdot \widetilde{a_G})(R_F \cup R_G)$, as always alpha-converting to avoid clashes.

Definition 4 (Equal-in-Frame relation). *An* Equal-in-Frame relation *(*EF-relation*) is a ternary relation between a frame* F *and two terms* M *and* N*, written* $F \vdash M = N$*.*

The intuition is that $F \vdash M = N$ means that given the knowledge of the aliases in F, it is possible to infer $M = N$. We shall not now define exactly how such inferences are done; rather, we shall give a set of conditions (in Section 4 below) on the EF-relation for our results to hold.

$$\textsc{In}\ \frac{F \vdash M = K}{F \rhd M(N).P \xrightarrow{K\ N[\tilde{a}:=\tilde{L}]} P[\widetilde{a} := \widetilde{L}]}\ \widetilde{a} = \mathrm{n}(N) \qquad \textsc{Out}\ \frac{F \vdash M = K}{F \rhd \overline{M}\ N.P \xrightarrow{\overline{K}\ N} P}$$

$$\textsc{Then}\ \frac{F \vdash M = N}{F \rhd \textbf{if}\ M = N\ \textbf{then}\ P\ \textbf{else}\ Q \xrightarrow{\tau} P}$$

$$\textsc{Else}\ \frac{\neg(F \vdash M = N)}{F \rhd \textbf{if}\ M = N\ \textbf{then}\ P\ \textbf{else}\ Q \xrightarrow{\tau} Q}$$

$$\textsc{Com}\ \frac{F \cup \mathcal{F}(Q) \rhd P \xrightarrow{(\nu\tilde{a})\overline{M}\ N} P' \qquad F \cup \mathcal{F}(P) \rhd Q \xrightarrow{M\ N} Q'}{F \rhd P \mid Q \xrightarrow{\tau} (\nu\widetilde{a})(P' \mid Q')}\ \widetilde{a}\#Q$$

$$\textsc{Par}\ \frac{F \cup \mathcal{F}(Q) \rhd P \xrightarrow{\alpha} P'}{F \rhd P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset$$

$$\textsc{Scope}\ \frac{F \rhd P \xrightarrow{\alpha} P'}{F \rhd (\nu a)P \xrightarrow{\alpha} (\nu a)P'}\ a\#\alpha, F$$

$$\textsc{Open}\ \frac{F \rhd P \xrightarrow{(\nu\widetilde{b_1}\cdot\widetilde{b_2})\overline{M}\ N} P'}{F \rhd (\nu a)P \xrightarrow{(\nu\widetilde{b_1}\cdot a\cdot\widetilde{b_2})\overline{M}\ N} P'}\ \begin{array}{l} a \in \mathrm{n}(N), \\ a\#M, F, \widetilde{b_1}, \widetilde{b_2} \end{array} \qquad \textsc{Rep}\ \frac{F \rhd P \mid !P \xrightarrow{\alpha} P'}{F \rhd !P \xrightarrow{\alpha} P'}$$

Table 1. Structural operational semantics. Symmetric versions are elided.

Definition 5 (Static equivalence). *Two frames F and G are* statically equivalent*, written $F \backsimeq G$, when $\mathrm{dom}(F) = \mathrm{dom}(G)$ and when for all terms M and N we have $F \vdash M = N$ iff $G \vdash M = N$.*

Definition 6 (Actions). *The* actions *ranged over by α, β are of the following three kinds: An* output *of kind $(\nu\tilde{a})\overline{M}\ N$, where $\tilde{a}$ are binding occurrences, an input of kind $M\ N$, and the silent action τ.*

For an output action $(\nu\tilde{a})\overline{M}\ N$ it will hold that $\mathrm{n}(M) \cap \tilde{a} = \emptyset$ and $\tilde{a} \subseteq \mathrm{n}(N)$. If $\tilde{a}$ is empty we write the action as just $\overline{M}\ N$; this corresponds to a free output in standard pi. In the actions above we will refer to M as the *subject*, corresponding to the channel over which communication takes places, and N as the *object* transferred in the communication. Note that the subject is a term and not necessarily a name. This admits functions in the signature that construct channels. When an output object N is syntactically complicated we sometimes write $\overline{M}\langle N\rangle$ for $\overline{M}\ N$, to facilitate reading, in both actions and prefixes.

The frame of an agent is, intuitively, what the agent contributes to its environment for resolution of aliases. It contains all the unguarded aliases, preserving the scope of names.

Definition 7 (Frame of an agent). *The function $\mathcal{F}$ from agents to frames is defined inductively as follows.*
$\mathcal{F}(\mathbf{0}) = \mathcal{F}(\gamma.P) = \mathcal{F}(\textbf{if } M = N \textbf{ then } P \textbf{ else } Q) = \emptyset, \quad \mathcal{F}(\{^M/_a\}) = \{^M/_a\},$
$\mathcal{F}((\nu a)P) = (\nu a)\mathcal{F}(P), \quad \mathcal{F}(P \mid Q) = \mathcal{F}(P) \cup \mathcal{F}(Q), \quad \mathcal{F}(!P) = \mathcal{F}(P)$

Definition 8 (Transitions). *A* transition *is of kind $F \rhd P \xrightarrow{\alpha} Q$, meaning that when the environment contains the frame F the agent P can do an α to become Q. The transitions are defined inductively in Table 1.*

The frame is used to determine the action subjects. So if the EF-relation equates the different terms M and N in every frame then the agents $\overline{M}\,K.P$ and $\overline{N}\,K, P$ are bisimilar. In contrast, the frame does not affect objects, so the agents $\overline{K}\,M.P$ and $\overline{K}\,N, P$ are *not* bisimilar, and can indeed be distinguished by the agent $K\,(a)\,.\,(\bar{b}\,\langle \mathrm{t}_2(a, M)\rangle\,.\,\mathbf{0} \;\mid\; b\,(\mathrm{t}_2(c, c))\,.\,Q)$, where the non-linear pattern matching along b succeeds only if the corresponding output is a tuple of two identical terms. We have experimented with versions where the prefix rules, or the Com-rule, or the definition of bisimilarity (and combinations thereof) use the frame to generate or compare objects. In all cases we have encountered technical problems in that the scope extension law $(\nu a)(P|Q) \sim P|(\nu a)Q$ if $a\#P$ fails, or restriction fails to preserve bisimilarity.

3 Examples

We will now show how the extended pi-calculi relate to a few other calculi and give some examples.

The pi-calculus and the polyadic pi-calculus. Any monadic pi-calculus agent [1] is also an extended pi-calculus agent. Using only names as symbols, and with an EF-relation with only identity on names, the extended pi semantics directly corresponds to an early operational semantics for the pi-calculus (without sum). The frame of a pi agent will always be empty since there is no aliasing in the pi-calculus.

Adding tupling symbols t_n for tuples of arity n, we can also easily encode the polyadic pi-calculus [2] in the extended pi-calculus. The encoding $[\![\,\cdot\,]\!]_{\mathrm{P2E}}$ uses pattern matching in the input rule:

$$[\![a(b_1, \ldots, b_n).P]\!]_{\mathrm{P2E}} = a(\mathrm{t}_n(b_1, \ldots, b_n)).[\![P]\!]_{\mathrm{P2E}}$$
$$[\![\overline{a}\langle c_1, \ldots, c_n\rangle.P]\!]_{\mathrm{P2E}} = \overline{a}\langle \mathrm{t}_n(c_1, \ldots, c_n)\rangle.[\![P]\!]_{\mathrm{P2E}}$$

Pi-calculus with polyadic synchronisation. In [6] the subject of an action can be a vector of names. This is useful if we want a couple of processes to atomically communicate only if they share a set of parameters. Example applications of polyadic synchronisation given in [6] are modelling e-services [20] where a client and a server can only communicate if they agree on a set of service parameters, and simple representations of localities and cryptography. Using

polyadic synchronisation we can gradually enable a communication by opening the scope of names in a subject.

Using tuples of names as subjects in an extended pi-calculus we gain the expressiveness of polyadic synchronisation (strictly greater than standard pi-calculus). With the tupling symbols above, we can, e.g., encode **if** $a = b$ **then** P **else 0** as $(\nu c)(\overline{\mathrm{t}_2(c,a)}\langle c\rangle \mid \underline{\mathrm{t}_2(c,b)}(d)\,.\,P$ where $c, d \# P$, and the underlining is intended to clarify the input subject.

The applied pi-calculus. Our work is inspired by the applied pi-calculus [4]. For the most part the applied pi-calculus is a subset and can be translated directly into an extended pi-calculus, but there are a few noteworthy issues. Firstly, in the applied pi-calculus there is a distinction between variables and names (collectively called *atoms*) where only variables can be substituted. Secondly, as seen in Section 1, in the applied pi-calculus there are limitations on what can be output in an output action. For example the agent $\overline{a}\,M.P$, where M is not an atom, has no transitions since the only allowed output actions are of the form $\overline{a}\,u$ or $(\nu u)\overline{a}\,u$ where u is an atom. In order to derive a transition from $\overline{a}\,M.P$ it must first be rewritten using rules for structural equivalence: $\overline{a}\,M.P \equiv (\nu x)(\{^{M}/_{x}\} \mid \overline{a}\,x.P) \xrightarrow{(\nu x)\overline{a}\,x} \{^{M}/_{x}\} \mid P$. In a similar fashion names are the only subjects in the applied pi-calculus. Any other term in subject position in a prefix must be rewritten to a name using structural equivalence before a transition can be derived.

Assuming an EF-relation which, as in the applied pi-calculus, is based on a set of user supplied equations, given an agent A in the applied pi-calculus, let $[\![A]\!]_{\mathrm{A2E}}$ be a translation into an extended pi-calculus. $[\![\,\cdot\,]\!]_{\mathrm{A2E}}$ maps variables to names and is a homomorphism except for the output prefix which is translated as follows:

$$[\![\overline{u}\,M.P]\!]_{\mathrm{A2E}} = (\nu a)(\{^{M}/_{a}\} \mid \overline{u}\,a.[\![P]\!]_{\mathrm{A2E}}),\ a \notin \mathit{atoms}(\overline{u}\,M.P)\ (M \text{ not an atom})$$
$$[\![\overline{u}\,M.P]\!]_{\mathrm{A2E}} = \overline{u}\,M.[\![P]\!]_{\mathrm{A2E}} \quad (\text{when } M \text{ is an atom})$$

In the applied pi-calculus we have that $\overline{u}\,M.P \equiv (\nu x)(\{^{M}/_{x}\} \mid \overline{u}\,x.P)$ where $x \notin \mathit{atoms}(\overline{u}\,M.P)$, which exactly matches our translation. Assuming a well sorted original process and well sorted inputs, this translation gives rise to a process that has the same transitions as the original, except for outputs of names that are not channels, which in applied pi-calculus can be output both literally and as aliases. To handle this in an extended pi-calculus we would need a choice operator.

As indicated in the introduction, we can express cryptographic operations, given an appropriate signature and EF-relation. Using aliases for cryptographic terms, we can protect the secrecy of keys and encrypted messages. In contrast to the applied pi-calculus, we also have the possibility to send data terms *as is*, without using aliases. This is important to be able to directly represent the polyadic pi-calculus. Consider an example based on our encoding of polyadic pi above, where z is a variable in the applied pi-calculus represented as a name in

our framework:

$$P = [\![(\nu c, d)\overline{a}\langle c, d\rangle.c(z).P']\!]_{\mathrm{P2E}} = (\nu c, d)\overline{a}\langle \mathrm{t}_2(c, d)\rangle.c(z).P''$$

In an extended pi-calculus, $P \xrightarrow{(\nu c,d)\overline{a}\langle \mathrm{t}_2(c,d)\rangle} c(z).P''$ and $c(z).P'' \xrightarrow{c\,M} P''[z := M]$ for some M. In the applied pi-calculus, to have a transition, P must first be rewritten to $Q = (\nu c, d, x)(\{^{\mathrm{t}_2(c,d)}/_x\} \mid \overline{a}\,x\,.\,c(z)\,.\,P'')$. This rewritten process can, however, not open the scopes of c and d, but has the only transition $Q \xrightarrow{(\nu x)\overline{a}\,x} (\nu c, d)(\{^{\mathrm{t}_2(c,d)}/_x\} \mid c(z)\,.\,P'')$. This process is deadlocked.

The spi calculus. If we assume an EF-relation based on an equation system and a signature with primitives for encryption and decryption, similarly to what we did for the applied pi-calculus above, we can encode the spi calculus [3] using the translation used above for the applied pi-calculus. Note in particular that the encoding ensures that encryptions are always sent as aliases.

4 Theory

This section gives our main results: our labelled bisimulation is preserved by the operators in the usual way, and although we do not use structural congruence in our operational semantics, the usual structural rules are sound.

Definition 9 (Bisimulation). *A* bisimulation $\mathcal{R}$ *is a ternary relation between frames and pairs of agents such that* $\mathcal{R}(F, P, Q)$ *implies all of*

1. *Static equivalence:* $\mathcal{F}(P) \backsimeq \mathcal{F}(Q)$
2. *Symmetry:* $\mathcal{R}(F, Q, P)$
3. *Extension of arbitrary frame:* $\forall G.\ \mathcal{R}(F \cup G, P, Q)$
4. *Simulation: for all* α, P' *such that the bound names in* α *are disjoint from* $\mathrm{n}(Q, F)$ *there exists* Q' *such that*

$$F \rhd P \xrightarrow{\alpha} P' \implies F \rhd Q \xrightarrow{\alpha} Q' \quad \wedge \quad \mathcal{R}(F, P', Q')$$

We define $P \sim Q$ *to mean that there exists a bisimulation* $\mathcal{R}$ *such that* $\mathcal{R}(\emptyset, P, Q)$.

Because of the universal quantification of G in clause 3, bisimilarity corresponds to equal behaviour in all frames.

In order to establish properties of $\sim$ we will need to make assumptions about the EF-relation.

Equivariance:	$F \vdash M = N \implies (a\,b) \bullet F \vdash (a\,b) \bullet M = (a\,b) \bullet N.$
Equivalence:	$\{(M, N) : F \vdash M = N\}$ is an equivalence relation.
Strengthening:	$a\#(F, M, N) \wedge F \cup \{^T/_a\} \vdash M = N \implies F \vdash M = N.$
Weakening:	$F \vdash M = N \implies F \cup \{^T/_a\} \vdash M = N.$
Scope Introduction:	$F \vdash M = N \wedge a\#(M, N) \implies (\nu a)F \vdash M = N.$
Scope Elimination:	$(\nu a)F \vdash M = N \implies F \vdash M = N.$
Idempotence:	$F \backsimeq F \cup F$
Union:	$F \backsimeq G \implies F \cup H \backsimeq G \cup H.$

Weakening and strengthening say that no fewer equalities can be proved by adding to the frame or by subtracting an alias that is never used. Scope introduction and elimination similarly say that by removing a scope no less can be proved (here the bound name a is guaranteed to be distinct from anything in M or N), and likewise for adding a scope that does not capture anything in M and N. Note that a property for restriction, $F \preceq G \implies (\nu a)F \preceq (\nu a)G$, follows from scope introduction and elimination.

$$
\begin{array}{rr}
P \equiv P \mid \mathbf{0} & (\nu a)\mathbf{0} \equiv \mathbf{0} \\
P \mid (Q \mid R) \equiv (P \mid Q) \mid R & P \mid (\nu a)Q \equiv (\nu a)(P \mid Q) \quad \text{if } a\#P \\
P \mid Q \equiv Q \mid P & \overline{M}\,N.(\nu a)P \equiv (\nu a)\overline{M}\,N.P \quad \text{if } a\#M,N \\
!P \equiv P \mid !P & M(N).(\nu a)P \equiv (\nu a)M(N).P \quad \text{if } a\#M,N
\end{array}
$$

$$
\begin{array}{l}
\textbf{if } M = N \textbf{ then } P \textbf{ else } (\nu a)Q \equiv (\nu a)(\textbf{if } M = N \textbf{ then } P \textbf{ else } Q) \quad \text{if } a\#M,N,P \\
\textbf{if } M = N \textbf{ then } (\nu a)P \textbf{ else } Q \equiv (\nu a)(\textbf{if } M = N \textbf{ then } P \textbf{ else } Q) \quad \text{if } a\#M,N,Q
\end{array}
$$

$$
(\nu a)(\nu b)P \equiv (\nu b)(\nu a)P \qquad (\nu a)\{^{M}/_{a}\} \equiv \mathbf{0} \qquad \{^{M}/_{a}\} \equiv \{^{N}/_{a}\} \text{ if } \emptyset \vdash M = N
$$

Table 2. Structural rules.

With these conditions on EF we can prove the following:

Theorem 1. *If $P \equiv Q$ from the structural laws in Table 2 then $P \sim Q$.*

Bisimilarity is preserved by all operators except input. Interestingly, the reason that it is not preserved by input is not the same as in the ordinary pi-calculus. Let $a \neq b$ and consider the agent $P = \textbf{if } a = b \textbf{ then } P' \textbf{ else } \mathbf{0}$; in the ordinary pi-calculus (where this **if** is represented by a match) we have $P \sim \mathbf{0}$ and $c\,(a).P \not\sim c\,(a).\mathbf{0}$, since the input may instantiate a to b. But in our framework we do *not* have that $P \sim \mathbf{0}$. The reason is that a frame can identify a and b, as in $\{^{a}/_{b}\} \rhd P \xrightarrow{\tau} P'$. Instead, the counterexample is

$$P = (\nu a)(\overline{a}\,b\,.\,\mathbf{0} \mid a\,(f)\,.\,\mathbf{0})$$

where f is a symbol that is not a name. Here P has no transitions because the pattern matching will never succeed (remember that the frame is not used when comparing the objects in a communication). So $P \sim \mathbf{0}$. But $d(b)\,.\,P$ can instantiate b in P to f, so $d(b)\,.\,P \not\sim d(b)\,.\,\mathbf{0}$.

Theorem 2.

1. $P \sim Q \implies P \mid R \sim Q \mid R$.
2. $P \sim Q \implies (\nu a)P \sim (\nu a)Q$.
3. $P \sim Q \implies !P \sim !Q$.
4. $P \sim R \wedge Q \sim S \Rightarrow \textbf{if } M = N \textbf{ then } P \textbf{ else } Q \sim \textbf{if } M = N \textbf{ then } R \textbf{ else } S$.
5. $P \sim Q \implies \overline{M}\,N.P \sim \overline{M}\,N.Q$.
6. $(\forall \widetilde{L}.\, P[\widetilde{a} := \widetilde{L}] \sim Q[\widetilde{a} := \widetilde{L}]) \implies M\,(N).P \sim M\,(N).Q$, *where* $\widetilde{a} = \mathrm{n}(N)$.

5 Further Work

Our framework for extended pi-calculi has been designed to facilitate formalisation with a theorem prover. We intend to extend our previous work [19] where we formalised a significant part of the pi-calculus using the nominal datatype package [21] of the interactive theorem prover Isabelle [7].

A few generalisations of the current framework will be considered. In particular, we would like to generalise the notion of a frame so that it can contain more than just aliases. In a typed version of the applied pi-calculus [22], processes are extended with first order logical predicates which are used to prove correspondence assertions of cryptographic protocols. This mechanism could be modeled by allowing the frame to include such predicates.

There are several versions of applied pi-calculi focusing on cryptographic properties of agents, where type checking rather than bisimilarity proves a process secure [23, 24, 22]. By typing extended pi-calculi we will include this kind of reasoning.

We intend to add weak bisimulation to our framework. Moving from strong early to weak early bisimilarity is usually not difficult and we do not anticipate any problems here. We also intend to explore barbed bisimilarity.

We have given some preliminary encodings of other calculi into our framework but we have not formally proved any properties about them. It would be interesting to investigate in a formal fashion how they relate and see what equivalences are preserved by the encodings.

Finally, we aim to give a symbolic semantics for our framework. Non-symbolic semantics suffer from problems with exploding state spaces making it difficult for automatic tools such as the Mobility or Concurrency Workbenches [25, 26] to reason about them. A symbolic semantics could make use of the generalisations mentioned above to let the frame contain conditions on symbolic values.

References

1. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, part I/II. Journal of Information and Computation **100** (September 1992) 1–77
2. Milner, R.: The polyadic π-calculus: A tutorial. In Bauer, F.L., Brauer, W., Schwichtenberg, H., eds.: Logic and Algebra of Specification. Volume 94 of Series F., NATO ASI, Springer (1993)
3. Abadi, M., Gordon, A.D.: A calculus for cryptographic protocols: The Spi calculus. Journal of Information and Computation **148**(1) (1999) 1–70
4. Abadi, M., Fournet, C.: Mobile values, new names, and secure communication. In: Proceedings of POPL '01, ACM (January 2001) 104–115
5. Borgström, J., Nestmann, U.: On bisimulations for the spi calculus. In Kirchner, H., Ringeissen, C., eds.: Proceedings of AMAST 2002. Volume 2422 of LNCS., Springer (2002) 287–303
6. Carbone, M., Maffeis, S.: On the expressive power of polyadic synchronisation in π-calculus. Nordic Journal of Computing **10**(2) (2003) 70–98
7. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: a Proof Assistant for Higher-Order Logic. Volume 2283 of LNCS. Springer (2002)

8. Abadi, M., Gordon, A.D.: A bisimulation method for cryptographic protocols. Nordic Journal of Computing **5**(4) (Winter 1998) 267–303
9. Elkjær, A.S., Höhle, M., Hüttel, H., Overgård, K.: Towards automatic bisimilarity checking in the spi calculus. In Calude, C.S., Dinneen, M.J., eds.: Combinatorics, Computation & Logic. Volume 21(3) of Australian Computer Science Communications., Springer (January 1999) 175–189
10. Boreale, M., De Nicola, R., Pugliese, R.: Proof techniques for cryptographic processes. SIAM Journal on Computing **31**(3) (2002) 947–986
11. Durante, L., Sisto, R., Valenzano, A.: Automatic testing equivalence verification of spi calculus specifications. ACM Trans. Softw. Eng. Methodol. **12**(2) (2003) 222–284
12. Borgström, J.: Equivalences and Calculi for Formal Verifiation of Cryptographic Protocols. PhD thesis, EPFL, Lausanne (2008) To appear.
13. Fournet, C., Abadi, M.: Hiding names: Private authentication in the applied pi calculus. In: Software Security – Theories and Systems. Volume 2609 of LNCS., Springer (2002) 317–338
14. Abadi, M., Blanchet, B., Fournet, C.: Just fast keying in the pi calculus. ACM Trans. Inf. Syst. Secur. **10**(3) (2007)
15. Bhargavan, K., Fournet, C., Gordon, A.D.: A semantics for web services authentication. Theor. Comput. Sci. **340**(1) (2005) 102–153
16. Chaudhuri, A., Abadi, M.: Formal security analysis of basic network-attached storage. In: FMSE '05: Proceedings of the 2005 ACM workshop on Formal methods in security engineering, New York, NY, USA, ACM (2005) 43–52
17. Haack, C., Jeffrey, A.: Pattern-matching spi-calculus. Information and Computation **204**(8) (2006) 1195–1263
18. Pitts, A.M.: Nominal logic, a first order theory of names and binding. Information and Computation **186** (2003) 165–193
19. Bengtson, J., Parrow, J.: Formalising the pi-calculus using nominal logic. In: Proceedings of FoSSaCS 2007. Volume 4423 of LNCS., Springer (2007) 63–77
20. Carbone, M., Coccia, M., Ferrari, G., Maffeis, S.: Process algebra-guided design of java mobile network applications. In: Informal Proceedings of the FMTJP'01 Workshop, Budapest. (2001)
21. Urban, C.: Nominal techniques in Isabelle/HOL. To appear in Journal of Automatic Reasoning (2007)
22. Fournet, C., Gordon, A.D., Maffeis, S.: A type discipline for authorization in distributed systems. To appear in Proc. of CSF'07 (2007)
23. Gordon, A.D., Jeffrey, A.: Secrecy despite compromise: Types, cryptography, and the pi-calculus. In Abadi, M., de Alfaro, L., eds.: Proc. of CONCUR 2005. Volume 3653 of LNCS., Springer (2005) 186–201
24. Fournet, C., Gordon, A.D., Maffeis, S.: A type discipline for authorization policies. In Sagiv, M., ed.: Proc. of ESOP 2005. Volume 3444 of LNCS., Springer (2005) 141–156
25. Victor, B., Moller, F.: The Mobility Workbench — a tool for the π-calculus. In Dill, D., ed.: Proceedings of CAV '94. Volume 818 of LNCS., Springer (1994) 428–440
26. Cleaveland, R., Parrow, J., Steffen, B.: The Concurrency Workbench: a semantics-based tool for the verification of concurrent systems. ACM Trans. Program. Lang. Syst. **15**(1) (1993) 36–72