A modular approach to the behavioural theory of mobility

Julian Rathke
ECS, University of Southampton, UK
jr2@ecs.soton.ac.uk

Pawel Sobociński
ECS, University of Southampton, UK
ps@ecs.soton.ac.uk

Abstract
We introduce a novel technique for the development of coinductive characterisations of contextually-defined equivalence for the Pi-calculus. The technique relies on splitting the derivation rules for labelled transition systems (LTSs) into two separate modular subsystems: a process-view and a context-view of interactions. In order to do so we introduce a typed meta-syntax for representing terms with holes. The approach yields a new context bisimulation for first-order Pi which refines to a composite description of the standard early LTS. The technique generalises to higher-order and asynchronous variants to give a modular account of their features.

1. Introduction
The Pi-calculus [15, 30] is a foundational model for the study of mobile processes. It has become one of the most well-known and widely studied process calculi. The formal study of its semantics has been thoroughly investigated in the literature, but more recently it has also begun to emerge as a useful tool on which to base analyses in various practical areas. For example, it has been extended in different ways [13, 22, 41, 43] and used as a basis for modelling distributed programming languages; its theory has informed their syntax, semantics and implementations [21, 33, 39]. It has also found use in the field of security and cryptoprotocol analysis via natural extensions to this domain [1, 2]. An emerging topic is the use of Pi as a tool for modelling and reasoning about web-services [6, 8, 11]. In particular, a Pi-calculus based language features as an underlying model for W3C’s working group on Web Services Choreography for the Web Services Choreography Description Language (WSDL) [12]. Indeed, there are varied application areas in which the Pi-calculus is being put to the test: from business processes [40] to modelling of biological systems using stochastic simulations [32, 34].

Each of these applications of Pi-calculus is based on specialising the Pi-calculus to the particular application domain; usually by extending one or more features of the language. With each change the behavioural theory of the language must be reworked in order to accommodate the new language features. This can be a non-trivial task and often leads to ad hoc solutions based upon a tailor-made LTS. However, fundamentally, at the heart of each application is a core language of binary synchronisations along a channel with the ability to pass names as arguments. Our goal is the extraction of a core transition system representing the behaviour of this Pi-calculus heart which could be used modularly with respect to language extensions. This is an ambitious undertaking and in order to proceed we must first understand to what extent the established semantics [30] of Pi-calculus are modular. That is the remit of this paper.

To see why the modularity of the standard Pi-calculus semantics is an issue we consider the role of the new name binder (ν). Recall that the Pi-calculus is essentially an extension of Milner’s CCS [29] in which channel names themselves are passed during communication and moreover, these channels may be restricted in their dynamically changing (extruding) scope by ν. The binder ν also behaves somewhat like a global generator of new names [42] since α-conversion ensures that whichever concrete name is chosen to represent a restricted name, it is different from all other names in the term. This dual nature of the new name binder - as both a binder and a generator of names - has a profound impact upon the LTS and its associated bisimulation equivalence for the language. As we shall see, the two roles of the new binder can in fact be studied separately in orthogonal LTS submodules. One of these modules focusses on the communication aspects of Pi (including dynamically scoped restriction), whereas another focusses on the data-passing aspects (including name generation).

The basic (reduction) semantics of the Pi-calculus is very similar to that of CCS, in fact, in the sum-free fragment we can express it essentially as the axiom

$$a!b. P \parallel a?x.Q \rightarrow P \parallel Q[b/x] \tag{1}$$

closed under evaluation contexts: parallel composition and restriction. The rule states that a process waiting to send a name b on channel a can synchronise with a process waiting to receive on the same channel a. After synchronisation the name is passed and substituted in to the receiving process. The reduction semantics naturally leads to a contextually defined equivalence: the barbed congruence [24, 31]. Although they are suitably canonical, contextually defined equivalences are difficult to reason about directly and for this reason LTSs are very useful. The labels aim at classifying possible interactions with an unknown environment in a structured way. By taking the canonical notion of equivalence over LTSs, bisimulation equivalence, it is shown that, by respecting substitutions of names, early bisimulation equivalence coincides with barbed congruence [36]. This is beneficial as the LTS semantics is much easier to reason with because of the structured approach, the removal of the quantification over all, arbitrarily complex contexts, and the power of coinduction as part of the bisimulation proof technique.

The early LTS, presented in Fig 1 for the finite, sum-free fragment without (mis)matching, is widely known [36] and enjoys an attractively simple presentation provided that a number of conventions are followed — we shall discuss these in more detail below. In order to simplify the presentation, we omitted the symmetric versions of (COMM), (CLOSE) and (PAR).

* Research partially supported by EPSRC grant EP/D066565/1.
perform an $a!(b)$ action while the second cannot. Intuitively, this phenomenon demonstrates $\nu$’s global behaviour as a fresh name generator – whichever name is present in the system, the fresh name observed (witnessed by the bound output label) will be different.

Sangiorgi and Walker [36, Convention 1.4.10] argue these issues regarding the name binder are effectively resolved by making sure that “... the bound names of any process or actions under consideration are chosen to be different from the names free in any other entities under consideration ...”. This is certainly true, but the convention is arguably a strong one and, conventions aside, the definition of bisimulation has nonetheless been tampered with. It has also been noticed by a number of authors that conventions which are fine for humans are often non-trivial to implement when using proof-assistants or theorem-provers. There have been several works [14, 18, 20] which aim at formalising these conventions, roughly the idea is to index the processes with the set of names which can appear within and consider bisimulation as a name-indexed relation. See [19] for an overview and a comparison of the approaches. Our related approach, introduced in §2 is to deal in a typed setting and consider indexed relations; in that case the side conditions are no longer necessary.

In order to develop a more modular way of presenting the LTS, we analyse its close relationship with the underlying reduction semantics. Indeed, as mentioned previously, the LTS can be considered as a tool for reasoning about a contextually defined equivalence on the reductions; following this line of reasoning raises a natural question as to whether the LTS can be derived in a systematic way from the reduction system; research in this direction was commenced by Leifer and Milner [27], continued by Sassone and the second author in [37, 38] and by Klin, Sassone and the second author [26]. In Leifer and Milner’s work, the labels of the resulting transition system are certain contexts $c[-]$ of the language.

Roughly, for a closed term $t$, $t \xrightarrow{c[-]} t’$ when $c[-]$ allows $t$ to reduce to $t’$ in one step, ie $c[t] \rightarrow t’$ and $c[-]$ is the minimal such context, in other words it doesn’t contain any redundant information in order for the reduction to occur. The minimality condition is expressed categorically; if the underlying category of terms and contexts satisfies certain assumptions then bisimilarity (defined in the usual way) on the LTS is a congruence. One of the limitations of the original theory is that the underlying reduction system is required to consist of ground rules, which practically rules out calculi such as CCS and Pi because the reduction rules contain parameters — eg $P$ and $Q$ in (1). A generalisation of the theory which aims at overcoming this problem was studied in [26] – reduction rules are open terms, and given another open term, one can compute both a minimal context and a minimal parameter which allows reduction. While work in progress, in a CCS-like setting a possible label would be of the form

$$a!hP_{-1}\xrightarrow{\nu[b/x]} P$$

where the context forms an (unsubstantiated) part of the term and the labels are able to modify the context.

In this paper, we use some of the intuitions gained from the ongoing work described in the paragraph above to develop a new approach to defining LTSs and we exemplify this approach using variants of the Pi-calculus. Technically, we expand the calculus with a typed meta-syntax for holes and contexts; the idea is that such contexts can form a part of a term after (a partial view of an) interaction. The meta-syntax is a simply-typed $\lambda$-calculus, with $\beta$-reduction forming a part of the structural congruence. Pleasently, by limiting our attention to the ‘one-step’ contexts for observing interactions, we obtain a separation in our new LTS of the interaction between term and context due to communication (the process-view) and the interaction between term and context due to observing identities of communicated names. This separation of concerns actually

### Figure 1. Standard early LTS, untyped.

The rules (OPEN) and (CLOSE) deal with the the two roles of the new name binder: generation of new name observable and scope extrusion. This may not be immediately apparent though as the rules have carefully combined these two roles in to a single mechanism. In particular, any process using rule (OPEN) to perform a bound output action $a!(b)$ can be construed to be announcing both $a / b$ can be construed to be announcing both $a / b$. This phenomenon demonstrates $\nu$’s global behaviour as a fresh name generator – whichever name is present in the system, the fresh name observed (witnessed by the bound output label) will be different.

Sangiorgi and Walker [36, Convention 1.4.10] argue these issues regarding the name binder are effectively resolved by making sure that “... the bound names of any process or actions under consideration are chosen to be different from the names free in any other entities under consideration ...”. This is certainly true, but the convention is arguably a strong one and, conventions aside, the definition of bisimulation has nonetheless been tampered with. It has also been noticed by a number of authors that conventions which are fine for humans are often non-trivial to implement when using proof-assistants or theorem-provers. There have been several works [14, 18, 20] which aim at formalising these conventions, roughly the idea is to index the processes with the set of names which can appear within and consider bisimulation as a name-indexed relation. See [19] for an overview and a comparison of the approaches. Our related approach, introduced in §2 is to deal in a typed setting and consider indexed relations; in that case the side conditions are no longer necessary.
leads to a separation at the level of our LTS which enables us to identify its modular sub-systems. The technique extends smoothly to deal with asynchronous communication and higher-order features.

Related work As explained previously, our work was inspired by the “deriving bisimulation congruences” approach first proposed by Leifer and Milner [27] and in particular the work on luxes by Klin, Sassone and the second author [26]. However, the approach presented in this paper does not use categorical concepts and relies on a syntactic analysis of Pi reduction semantics.

The closest related approaches in the literature are Ziegler, Miller and Palamidessi’s [44] use of higher order abstract syntax in formalising the LTSs of Pi and Milner’s use abstractions and concretions for characterising the late semantics [29]. Ziegler et al aim at formalising the early, late and open labelled semantics of Pi, as well as developing a congruence format for SOS rules involving name binding. Similarly to our approach, they use $\lambda$-expressions within terms, taking advantage of the rich proof theory of \( FOA^\Delta,\nabla \). However, their intuitions and goals are different. They bind names under a $\lambda$ in order to emphasise the “undefinedness” of a name and use this to capture open bisimilarity. Our use of $\lambda$-bindings, explained in §3 is to represent the context part in the process-view of an interaction. Technically, our treatment of the $\lambda$-expressions is rather simple and we have no need for a rich proof theory – adding the $\beta$-rule and $\alpha$-equivalence into the structural congruence of the Pi-calculus suffices.

While Ziegler et al use the usual (OPEN)(CLOSE) mechanism for dealing with the $\nu$ binder, the use of our technique avoids this by effectively splitting the roles of $\nu$ and dealing with them separately; we also demonstrate the advantages of such a modular approach on higher-order and asynchronous variants. The difference in approaches comes from different underlying goals; Ziegler et al are interested in a global view of a system of SOS rules – whether or not a system falls into a congruence format; instead we advocate a modular approach where properties of a full system can be derived by understanding how the system decomposes. However, there are technical similarities even at the SOS level – their (PAR) and (RES) rules are closely related to the corresponding rules in our communication fragment $C$, presented in Fig 7.

Despite the Ziegler et al development being somewhat technically similar to our approach, Milner’s [29] approach to capturing the early semantics of Pi using abstractions and concretions is closer to our approach in spirit. Abstractions and concretions are syntactic entities which arise as a result of the complementary roles of inputs and outputs in Pi. Our approach does this by treating both sides as certain types of abstractions which can be combined using standard $\lambda$ application in order to obtain process terms. Differently, our combined LTS $CA$ is an early system because we combine the process-view and the context-view into a single observation; had we kept them separate we would have likewise obtained a late LTS.

One of the achievements of [44], as well as several previous works [14, 18, 20], is the presentation of SOS rules without side-conditions; in fact, we achieve this in §2 without the use of $\lambda$-expressions as a by-product of working in a typed setting and distinguishing names and variables of name type. In particular, we choose to treat $\nu$ as an ordinary binder which binds variables of name type. In this choice, our presentation is similar to Engberg and Nielsen’s ECSCS [15, 16], the “precursor” to the Pi-calculus.1 In light of the aforementioned works, the results of §2 are not surprising and can be considered to be folkloric in the process calculus community.

Structure of the remainder In §2 we introduce our notation and reinterpret the standard SOS rules for early bisimilarity in our typed setting. In §3 we take advantage of the meta-syntax for holes (technically dealt with by having $\lambda$-expressions within our terms) in order to approach the derivation of an LTS modularly by considering the process-view of an interaction separately from the context-view. We show that there is a canonical LTS for the context-view, closely resembling applicative approaches [3]. In §4 we compare the resulting context equivalence LTS with standard early bisimilarity. In §5 we show further that the canonical context-view LTS can be refined, obtaining a SOS specification which generates an LTS identical to the standard early LTS. Following this, we demonstrate the modularity aspects of our techniques by defining LTS module specifications of semantics for the higher-order and asynchronous variants of the pi-calculus in §6. We conclude the paper with closing remarks on future work.

2. A simply-typed Pi calculus

In this section we revisit the syntax, structural congruence and the reduction semantics of Pi in a typed setting and reintroduce the early LTS. The type system, presented in the upper section of Fig 2 is very simple: there are two base types; a name type $\mathcal{N}$ and a process type $\mathcal{P}$. Additionally, we will most often use two particular function types: $\mathcal{N} \to \mathcal{P}$ and $(\mathcal{N} \to \mathcal{P}) \to \mathcal{P}$. Type contexts are denoted by $\Gamma$ and are finite maps of variables to types. We use the notation $\Gamma, x : \sigma$ to mean the context $\Gamma$ extended with the mapping $x \mapsto \sigma$; implicitly it is always assumed that $x$ is not already in the domain of $\Gamma$.

We start with an untyped syntax and give the type rules in the middle and lower sections of Fig 2. We shall consider terms of function type to be a part of a formal meta-syntax for terms with holes; these will be used in §3. Types in this setting are included only to make the meta-syntax formal, they have a different structure than usual Pi-calculus type systems based on channel types [36, Part III]. We consider only typeable terms.

Differently from the classical presentation, we treat $\nu$ as a standard binder which binds variables of name type. We assume a countable supply of variables of each type in addition to a separate countable supply of name constants. We shall use the syntactic convention of $a, b$ for name constants, $k, l$ for terms of name type (either constants or variables), $x, y$ for variables of name type, $P, Q$ for terms of process type, $X, Y$ for variables of function type.

The typed framework introduced in this section may strike the reader as excessive for the theory within this section; indeed, here our type contexts will only consist of lists of free name variables which appear in the terms. The additional generality (use of the function types) will be crucial importance as part our continued development in the following sections and as such we take the liberty in introducing the full theoretical foundation already at this point.

In calculi such as Pi, the notion of structural congruence gives us a useful notion of abstract syntax or “chemical soup” [7]. The basic axioms of structural congruence are presented in Fig 3. Structural congruence $\equiv$ is the largest relation which satisfies the axioms and is closed under all the syntactic features of the calculus: the output prefix, the input binder, the $\nu$ binder, the $\lambda$-binder and the parallel composition. Note that we include the $\beta$-reduction rule as part of the structural congruence; this is because the $\lambda$-bindings are used as meta-syntax for terms with holes; the filling of the holes, done by application, is a syntactic notion which does not have a computational component. The substitution within the $\beta$-rule is the usual capture-avoiding notion. We do not give the formal definition here and only emphasise is that our language contains three binders – substitution is capture-avoiding with respect to all three.

\footnote{Actually, all the basic ingredients of the theory (scope extrusion, fresh name generation) were already present in the aforementioned work; in later works the syntactic categories of variables and names were unified.}
\[
\sigma \ ::= \ \mathcal{N} \mid \mathcal{D} \mid \sigma \rightarrow \sigma \\
\]

\[
M \ ::= \ x \mid a \mid 0 \mid M \parallel M \mid M \cdot M \mid \nu x M \mid \text{if } M = M \text{ else } M \mid \lambda x : \sigma . M \mid M(M)
\]

\[
\begin{align*}
\text{Γ} \vdash a : \mathcal{N} &\quad \text{(NAME)} \\
\text{Γ} \vdash x : \sigma &\quad \text{(VAR)} \\
\text{Γ} \vdash \text{} &\quad \text{(NULL)} \\
\text{Γ} \vdash k, \cdot, \cdot \vdash \text{} &\quad \text{(IF)} \\
\text{Γ} \vdash M \parallel M' &\quad \text{(PAR)} \\
\text{Γ} \vdash \cdot, \cdot, M : \mathcal{D} &\quad \text{(OUTPREF)} \\
\text{Γ} \vdash \cdot, x : \mathcal{D} &\quad \text{(NU)} \\
\text{Γ} \vdash \cdot, \lambda x : \sigma . M : \mathcal{D} &\quad \text{(INPREF)} \\
\text{Γ} \vdash \cdot, M_1 : \sigma \rightarrow \sigma' &\quad \text{(\lambda)} \\
\text{Γ} \vdash M_2 : \sigma' &\quad \text{(\APP)} \\
\end{align*}
\]

Figure 2. Types, syntax and typing rules of first-order Pi.

\[
(P \parallel Q) \parallel R = P \parallel (Q \parallel R) \\
(P \parallel Q) \parallel P = P \parallel (Q \parallel P) \\
ν x P \equiv ν y (P \parallel Q) \equiv ν y P \quad (x \neq P) \\
\text{rp}(P) = P \parallel \text{rp}(P) \\
\text{rp}(P \parallel Q) = \text{rp}(P) \parallel \text{rp}(Q) \\
\text{rp}(0) = 0 \\
(\lambda x : M \cdot N) = M[N/x] \\
ν x P = ν y P[y/x] \\
ν x (P \parallel Q) = ν y (P \parallel Q) \\
(λ x : P = λ y : P)[y/x] = (g f \circ P)
\]

Figure 3. Structural congruence.

\[
\begin{align*}
\text{Γ} \vdash \text{} &\quad \text{(IN)} \\
\text{Γ} \vdash k \parallel P &\quad \text{(OUT)} \\
\text{Γ} \parallel P &\quad \text{(COMM)} \\
\text{Γ} \parallel P &\quad \text{(OPEN)} \\
\text{Γ} \parallel ν x P &\quad \text{(CLOSE)} \\
\end{align*}
\]

Figure 5. Standard early LTS, typed (Σ).

Of course, since we are dealing with a simply-typed \(\lambda\)-calculus we are not introducing any infinite behaviour and structural congruence, decidable without the \(\beta\)-rule [17], remains decidable.

The following is easily shown by individually considering the axioms in Fig 3.

\textbf{Lemma 2.1.} Structural congruence preserves typing; ie if \(M \equiv N\) then \(\Gamma \vdash M : \sigma\) iff \(\Gamma \vdash N : \sigma\).

\textbf{Definition 2.2.} (Typed transition system). A typed transition system is a has states pairs of a type context and a term typeable with the context; ie the states are contained in the set:

\[
\{ \ (\Gamma, Σ) \mid \exists σ, \ Σ \vdash Σ : σ \}
\]

We shall use the notation \(\Gamma \vdash Σ\) to refer to the elements of the set defined above. Our transition systems are presented in the structural style. We make one non-standard assumption: our transition systems are defined on the abstract syntax in the sense that we assume the implicit presence of the rule:

\[
P' \equiv P &\quad (\Gamma \parallel P) \overset{\Sigma}{\longrightarrow} (\Gamma \parallel P') \\
\]

The choice of including the rule \(\text{STRCNG}\) in our transition system greatly reduces the number of necessary rules, allowing us to concentrate on the more interesting cases and not on the rather standard “structural” rules. The price is that proofs based on structural induction over terms become less trivial. The reader who prefers dealing with concrete syntax will be able to translate our LTSs accordingly by adding the symmetric rules for parallel composition, rules for equality and inequality testing, etc.
Our first typed transition system is the reduction semantics for the Pi-calculus, presented in Fig 4. Subject reduction is easily shown using a straightforward induction on transition derivation.

**Proposition 2.3** (Subject reduction). If \( \Gamma \vdash P : \sigma \) and \( (\Gamma \vdash P) \rightarrow (\Gamma' \vdash P') \) then \( \Gamma \vdash P' : \sigma \).

The natural notion of equivalence relation on states of a typed transition system is an indexed relation. In particular, we shall consider indexed bisimulation, as defined below.

**Definition 2.4** (Indexed relation). A (type-context) indexed relation \( R \) is a set of tuples of the form \( (\Gamma, P, Q) \) where \( \Gamma \) is a type context and \( P, Q \) are terms and there exists a type \( \sigma \) such that \( \Gamma \vdash P : \sigma \) and \( \Gamma \vdash Q : \sigma \). We will use the notation \( (\Gamma \vdash P) R (\Gamma \vdash Q) \) to mean that \((\Gamma, P, Q) \in R\).

Fixing any type context \( \Gamma \), let \( R_\emptyset \) be the sub-relation \( \{ (\Gamma', P, Q) \mid \Gamma' = \Gamma \wedge (\Gamma, P, Q) \in R \} \). Closed relations (relating closed terms — those typeable in the empty type context) \( R_\emptyset \) will be of particular interest.

The reduction semantics naturally leads to a notion of contextually defined-equivalence, the barb-congruence; defined here as an indexed relation in the dynamic style [24]. We assume the notion of strong barb: the ability to immediately input or output on a particular channel.

**Definition 2.5** (Barb-congruence). Barb-congruence, denoted \( \equiv \), is the largest symmetric relation such that if \( (\Gamma \vdash P) \equiv (\Gamma' \vdash P') \) then:

(i) \( (\Gamma \vdash P) \rightarrow (\Gamma' \vdash P') \) then there exists \( (\Gamma' \vdash P'') \) such that \( (\Gamma \vdash P'') \equiv (\Gamma' \vdash P'') \);
(ii) \( (\Gamma \vdash P) \parallel l \) then \( (\Gamma \vdash l) \);
(iii) for all \( \Gamma' \supseteq \Gamma \) and \( (\Gamma' \vdash C) \) where \( C \vdash \lambda X : \sigma \ C' \) and \( X \) appears uniquely in \( C' \) we have \( (\Gamma' \vdash C(P)) \equiv (\Gamma' \vdash C(P)) \).

While the reduction semantics gives the behaviour of Pi-terms, and barb-congruence, given a canonical notion of process equivalence, it is a difficult notion to reason about directly; an LTS which classifies the possible interactions with the context is normally given. The typed LTS presented in Fig 5 is the standard early LTS translated into the typed setting; it is representative of the LTSs we consider.

Each typed transition system we shall consider has a type preservation lemma, proved by induction on derivation of transitions.

**Proposition 2.6** (Type preservation, \( S \)). Suppose that \( \Gamma \vdash P : \sigma \) and consider the LTS generated from \( S \) (Fig 5). Then for all \( \alpha \) and \( \Gamma \vdash P' \) such that \( (\Gamma \vdash P) \overset{\alpha}{\rightarrow} (\Gamma' \vdash P') \) we have \( \Gamma' \vdash P' : \sigma \).

A particular case of an indexed relation is indexed bisimulation, which is the canonical equivalence on any LTS.

**Definition 2.7**. An indexed bisimulation \( \sim \) wrt an LTS \( L \) is an indexed relation \( R \) which satisfies the following: If \( (\Gamma, P, Q) \in R \) then

- if \( (\Gamma \vdash P) \overset{\alpha}{\rightarrow} (\Gamma' \vdash P') \) in \( L \) then there exists \( (\Gamma' \vdash Q') \in L \) such that \( (\Gamma' \vdash P', Q') \in R \);
- if \( (\Gamma \vdash P) \overset{\alpha}{\rightarrow} (\Gamma' \vdash Q') \) in \( L \) then there exists \( (\Gamma' \vdash P') \in L \) such that \( (\Gamma' \vdash P', Q') \in R \).

Bisimilarity \( \sim \) is defined to be the largest (indexed) bisimulation.

**Remark 2.8**. It is worthwhile to examine the reasons for the fact that the side conditions on \( \text{(CLOSE)} \) and \( \text{(PAR)} \), present in the untyped LTS (of Fig 1), are not necessary in the typed version of Fig 5. In \( \text{(CLOSE)} \), the side condition is unnecessary as \( x \) is explicitly assumed not to be in \( \Gamma \) and thus cannot appear freely in \( Q \). Similarly, in \( \text{(PAR)} \), \( x \) does not appear in \( \Gamma \) and so cannot appear freely in \( Q \).

---

**Figure 6.** Pictorial intuition for the process actions of \( C \).

We shall need a technical lemma about the structure of terms which offer a transition. Similar structural lemmas will be given for other LTS fragments presented henceforward. Structural lemmas are one place where modularity will be exploited; the statement and proof of a structural lemma of a combined system follow from the respective components of the structural lemmas of its subsystems.

**Lemma 2.9** (Structural, \( S \)).

(i) A transition \( (\Gamma \vdash P) \overset{k/x}{\rightarrow} (\Gamma' \vdash P') \) is derivable iff there exist \( \mathsf{w}, R \) and \( S \) such that \( P \equiv \nu \mathsf{w} \langle k/xR \parallel S \rangle \) and \( P' \equiv \nu \mathsf{w} \langle k/xR \parallel S \rangle \);
(ii) a transition \( (\Gamma \vdash P) \overset{k/x}{\rightarrow} (\Gamma' \vdash P') \) is derivable iff there exist \( \mathsf{w}, R \) and \( S \) such that \( P \equiv \nu \mathsf{w} \langle k!IR \parallel S \rangle \) and \( P' \equiv \nu \mathsf{w} \langle k!IR \parallel S \rangle \);
(iii) a transition \( (\Gamma \vdash P) \overset{k/x}{\rightarrow} (\Gamma' \vdash P') \) is derivable iff there exist \( \mathsf{w}, R, S \) and \( T \) such that \( P \equiv \nu \mathsf{w} \langle k!IR \parallel k/xS \parallel T \rangle \) and \( P' \equiv \nu \mathsf{w} \langle R \parallel S[l/x] \parallel T \rangle \).

**Proof.** If direction for all four cases is immediate, using the rules in Fig 5. The only complication comes in the case of a \( \tau \) transition, where either the (OUT) or (OPEN) rule is used to derive the output component, followed by, respectively, an application of the (COMM) or the (CLOSE) rule. The only if direction for all three cases is a straightforward induction on the derivation of each kind of labelled transition.

It is worth recalling some standard results about early bisimilarity in the standard Pi-calculus and translating them into our nomenclature. Early bisimilarity is a so-called non-input congruence, meaning that it is a congruence with respect to all operators except the input prefix which may bind a free variable in term. It amounts to the following properties of \( \sim_S \):

**Proposition 2.10.**

(i) if \( (\Gamma \vdash P) \sim_S (\Gamma \vdash Q) \) and \( (\Gamma \vdash kX.qX) \sim (\Gamma \vdash k!IR) \);
(ii) if \( (\Gamma \vdash P) \sim_S (\Gamma \vdash Q) \) then \( (\Gamma \vdash k.xP) \sim_S (\Gamma \vdash k.xQ) \);
(iii) if \( (\Gamma \vdash P) \sim_S (\Gamma \vdash Q) \) and \( (\Gamma \vdash R) \) then \( (\Gamma \vdash P \parallel R) \sim_S (\Gamma \vdash Q \parallel R) \).

**Proof.** See [36, Theorem 2.2.8(1)].

In particular, because closed terms do not contain free name variables and it is straightforward to show that bisimilarity is in fact a congruence with respect to a non-binding input prefix, we easily obtain the following.

**Proposition 2.11.** \( \sim_S, S \) is a congruence.
3. A structured LTS

In this section we shall describe a modular approach to endowing the Pi-calculus with an LTS. Roughly the main idea is to split a labelled transition which corresponds to an interaction with a context into a process-view of the interaction (the process-view LTS $\mathcal{C}$, given in Fig 7) and a context-view of the interaction (Fig 8). The complete LTS is obtained by combining the two views (Fig 9). In the first-order Pi-calculus, it turns out that the two semantic roles of the $\nu$ binder appear in separate components – scope extrusion is dealt with by the process view and the observability of fresh names is dealt with by the context view.

We begin with an outline of the part of the process LTS concerned with the process-view of interaction – the fragment which deals with communication and scope extrusion. Before presenting the rules in a structural style, we shall discuss the motivation. Consider a typical Pi reduction, illustrated thrice in Fig 6. Since Pi reductions are binary synchronisations, the process can either provide the input part (upper reduction), the output part (middle reduction) or both components (lower reduction). In each case, the redex is split into the solid white component which identifies the process part, and the shaded cloud component which identifies the context part. After the reduction occurs, we can consistently re-apply the classification of the resulting term into a process part and a context part. Replacing the parts provided by the context with informal boxes, we can identify the “entangling” of the process with the context, illustrated in the middle column with the aid of dashed boxes. The entangled part consists in each case of the continuation of the process ($P$) together with any parts of the context which have been “absorbed”. Thus: (1) if the process provides the input, after an interaction it is entangled with with a name supplied by the context; (2) if it provides an output it is entangled with an entire process which has accepted the sent name; and (3) if the reduction happens within the process then no external context is necessary.

The LTS is defined structurally according to the rules in Fig 7. Henceforward we shall use the syntactic convention of writing $T$ for terms of type $\Lambda \rightarrow \Psi$ and $U$ for terms of type $\Lambda$. The labels in the fragment are the usual CCS labels. The intuitive idea is that a process offering an output on a channel $k$ can perform the interaction with a context, and evolve into a process consisting of its continuation in parallel with the interacting context, which has been passed the communicated name. This context is explicitly described in the resulting state as a $\lambda$-abstraction which binds a variable of type $\Lambda \rightarrow \Psi$ (cf $\text{OUT}$). On the other hand, a process offering an input on a channel $k$ can perform the communication with the context and obtain some name – the result is a $\lambda$-abstraction which binds a variable of type $\Lambda$ (cf $\text{IN}$). A process with a capability to input on $k$ in parallel composition with a process which has a capability to output on the said channel can perform the synchronisation without the need for an external context – the abstractions are combined via an application (cf $\text{TAU}$).

There is a technical similarity to the presentation of the late semantics using abstractions and continuations [29]; see the Comm rule below. Notice that we do not say anything about the actual nature of the data transmitted as part of the interaction.

The LTS presented in Fig 7 is only a fragment and bisimilarity does not give a satisfactory equivalence. In particular, there is no observation of the identity of the name communicated (input, output, or fresh output). Intuitively, transitions which arise out of process terms, as presented in Fig 7, represent the part of the interaction which is controlled by the process; transitions out of $\lambda$-abstracted terms, presented in Fig 8, represent the part of the interaction controlled by the context.

In the rules $\text{IIN}$ and $\text{IN}$, the variable $x$ in the right hand side of the result is chosen fresh for $U$. Similarly, in $\text{OUT}$ and $\text{OUT}$ the variable $X$ is chosen fresh for $T$. The idea is that each syntactic construct can be moved within the $\lambda$-abstraction; the $\nu$ binder in such a way that it possibly captures a variable within the term.

The LTS has a regular structure – there is one axiom for each kind of label - $\text{(IN)}$, $\text{OUT}$ and $\text{Tau}$ for input, output and $\tau$ respectively. Further, there is precisely one rule for each syntactic constructor and each kind of label.

Below give a brief example in order to demonstrate how scope extrusion is handled by the above rules. First observe that we can derive the following transition:

$$\vdash \text{IF } k=k \text{ then } P \text{ else } Q \xrightarrow{\alpha} \text{IF } \nu \alpha \text{ then } P \text{ else } Q$$

but $AX, \nu x ((AX') | (AX'' | X)) \Leftrightarrow AX, \nu x (P | (X | x))$, and so, using $\text{Tau}$ we obtain the correct extrusion of scope:

$$\vdash \text{IF } k=k \text{ then } P \text{ else } Q \xrightarrow{\alpha} \text{IF } \nu \alpha \text{ then } P \text{ else } Q$$

since $\nu x (P | (\nu y, Q | x)) \Leftrightarrow \nu x (P | Q | x/y)$.
One can prove a type preservation result about the C-LTS – in particular demonstrating that the result of applying the (TAU) rule is typeable.

**Lemma 3.1 (Type preservation, C).** Suppose that \( \Gamma \vdash P : \mathcal{P} \) and consider the LTSs generated from \( \mathcal{C} \) (Fig 7). Then the following hold:

- If \( (\Gamma \vdash P) \triangleright k \Gamma \vdash (\Gamma \vdash P) \) then \( \Gamma \vdash U : \mathcal{N} \rightarrow \mathcal{P} \);
- If \( (\Gamma \vdash P) \triangleright k \Gamma \vdash T : (\mathcal{N} \rightarrow \mathcal{P}) \rightarrow \mathcal{P} \);
- If \( (\Gamma \vdash P) \triangleright k \Gamma \vdash P' : \mathcal{P} \).

**Proof.** We use induction on the derivation of the transition. Each transition has a single axiom which clearly satisfies the required condition. Moreover, for an input transition \( k! \), clearly each of the rules ([IN] and \( \triangleright \text{IN} \)) as expected assuming the inductive hypothesis. One argues similarly for \( k! \) and rules ([OUT] and \( \triangleright \text{OUT} \)), as well as \( \tau \) and rules ([OUT] and \( \triangleright \text{OUT} \)).

We shall also need a structural lemma (cf Lemma 2.9) for the \( \mathcal{C} \) fragment.

**Lemma 3.2 (Structural, C).**

(i) A transition \( (\Gamma \vdash P) \triangleright k \Gamma \vdash (\Gamma \vdash P) \) is derivable if there exist \( \overline{w} \), \( R \) and \( S \) with \( k \notin \overline{w} \) such that \( P \equiv \nu \overline{w}(k!xR \parallel S) \) for \( x \notin \Gamma \) and \( U \equiv A x. \nu \overline{w}(R \parallel S) \);

(ii) a transition \( (\Gamma \vdash P) \triangleright k \Gamma \vdash T : (\mathcal{N} \rightarrow \mathcal{P}) \rightarrow \mathcal{P} \) then \( \Gamma \vdash U : \mathcal{N} \rightarrow \mathcal{P} \); where \( T = A x. (\nu \overline{w}(R \parallel S)) \parallel S \);

(iii) a transition \( (\Gamma \vdash P) \triangleright k \Gamma \vdash P' : \mathcal{P} \) is derivable if there exist \( \overline{w}, R \) and \( S \) such that \( P \equiv \nu \overline{w}(k!R \parallel k!xS \parallel T) \) and \( P' \equiv \nu \overline{w}(R \parallel S[l/x] \parallel T) \).

**Proof.** The if direction is a straightforward application of the rules of Fig 7. The only if direction is an induction on the derivation of the two kinds of labelled transitions.

A similar, if trivial, type preservation result can be given for the \( \mathcal{A} \) fragment. Here the structural lemma is vacuous.

**Lemma 3.3 (Type preservation, A).**

- If \( \Gamma \vdash U : \mathcal{N} \rightarrow \mathcal{P} \) and \( (\Gamma \vdash P) \triangleright k \Gamma \vdash (\Gamma \vdash P) \) then \( \Gamma \vdash U : \mathcal{N} \rightarrow \mathcal{P} \); where \( T = A x. (\nu \overline{w}(R \parallel S)) \parallel S \);

- If \( \Gamma \vdash P : \mathcal{N} \rightarrow \mathcal{P} \) and \( (\Gamma \vdash P) \triangleright k \Gamma \vdash T : (\mathcal{N} \rightarrow \mathcal{P}) \rightarrow \mathcal{P} \) then \( \Gamma \vdash U : \mathcal{N} \rightarrow \mathcal{P} \).

Consider the LTSs generated by taking a combined set of rules from \( \mathcal{C} \) and \( \mathcal{A} \). Because of unique typing, the two LTSs do not interfere in the sense that the derivation of each label is entirely within \( \mathcal{C} \) or \( \mathcal{A} \); in particular, the conclusions of the type preservation Lemmas 3.1 and 3.3 hold for the combined LTSs.

Transitions labelled with the \( \tau \) labels \( \{ k! \} \) will be referred to as the **process** actions and those with the \( \mathcal{A} \) labels \( \{ \alpha \} \) as the **context** actions. Process actions and context actions combine together to capture a single interaction (corresponding to a single step in the reduction semantics). We shall consider such complete actions as single observations; working with the LTS generated by the system \( \mathcal{C} \mathcal{A} \) featured in Fig 9. Note that the \( \tau \) labelled transitions are exactly those derived in the \( \mathcal{C} \) fragment.

**Lemma 3.4 (Structural, \( \mathcal{C} \mathcal{A} \)).**

(i) A transition \( (\Gamma \vdash P) \triangleright k \Gamma \vdash (\Gamma \vdash P') \) is derivable iff there exist \( \overline{w} \), \( R \) and \( S \) with \( k \notin \overline{w} \) such that \( P \equiv \nu \overline{w}(k!xR \parallel S) \) and \( P' \equiv \nu \overline{w}(R[l/x] \parallel S) \);

(ii) a transition \( (\Gamma \vdash P) \triangleright k \Gamma \vdash P' : \mathcal{P} \) is derivable if there exist \( \overline{w}, R \) and \( S \) such that \( f_{\Gamma}(U) \cap \overline{w} = \emptyset \) and \( P \equiv \nu \overline{w}(k!R \parallel S) \) and \( P' \equiv \nu \overline{w}(R[l/x] \parallel S) \).

**Proof.** Straightforward from Lemma 3.2 and the fact that substitution is capture-avoiding.

A type preservation lemma follows immediately from Lemmas 3.1 and 3.3.

**Lemma 3.5 (Type preservation, \( \mathcal{C} \mathcal{A} \)).** If \( \Gamma \vdash P : \mathcal{P} \) and \( (\Gamma \vdash P) \triangleright k \Gamma \vdash (\Gamma \vdash P') \) then \( \Gamma \vdash U : \mathcal{P} \).

Analogously to the case of the standard LTS, closed bisimilarity is a congruence with respect to the operators of \( \Pi \).

**Proposition 3.6.** \( \sim_{\mathcal{C} \mathcal{A}, \mathcal{P}} \) is a congruence.

The following lemma lets us prove easily that \( \sim_{\mathcal{C} \mathcal{A}} \) defines the same equivalence as barbed congruence. By design, there is a close correspondence with the LTSs of \( \mathcal{C} \mathcal{A} \) and the reductions – for each kind of label there is a context which interacts with the process and produces the same result. The converse is only a little more difficult; essentially the provided context has to be forced to interact with the term; this is done with the aid of an auxiliary name constant to provide a barb.

**Definition 3.7 (Label-characterising contexts).** Let \( \chi(k!l) = k!l \), \( \chi(k!U) = k!x. (U(x)) \) and \( \chi(\tau) = 0 \). Given a name constant \( c \), let \( \chi_{\mathcal{A}}(k!l) = k!l. c \) and \( \chi_{\mathcal{C}}(k!U) = k!x. c. (U(x)) \) and \( \chi_{\mathcal{C}}(\tau) = c! \).

**Lemma 3.8 (Characterising contexts).** Let \( c \) be a name constant fresh for \( P \). If \( P \triangleright c \) then \( P \triangleright c!P' \) and \( P \triangleright c \chi_{\mathcal{A}}(\alpha) \rightarrow P' \) and \( P \triangleright c\chi_{\mathcal{C}}(\alpha) \rightarrow P' \).

The characterising contexts let us easily show that bisimilarity and barb-congruence agree on closed terms.

**Theorem 3.9.** \( \sim_{\mathcal{C} \mathcal{A}, \mathcal{P}} \subseteq \sim_{\mathcal{C} \mathcal{A}, \mathcal{P}} \).

**Proof.** It is straightforward to show that \( \sim_{\mathcal{C} \mathcal{A}} \subseteq \sim_{\mathcal{C} \mathcal{A}, \mathcal{P}} \) using the fact that \( \sim_{\mathcal{C} \mathcal{A}} \) is defined to be the largest relation which is reduction closed, closed under bars and is a congruence (cf Definition 2.5)
it suffices to show that $\sim_{C,A}$ satisfies these properties. The first two are straightforward, the last follows from Proposition 3.6.

To show that $\sim_{C,A} \subseteq \sim_{C,A}$ it suffices to show that $\sim$ is a bisimulation, thus: $P \xrightarrow{\sim} P'$ then $P \parallel \chi_{(\alpha)} \xrightarrow{\tau} P''$ such that $P'' \xrightarrow{\sim} 1$. Thus $Q \parallel \chi_{(\alpha)} \xrightarrow{\tau} Q''$ such that $P'' \sim Q''$; so $Q'' \xrightarrow{\sim} 1$. Moreover, since $P'' \parallel \chi \xrightarrow{\tau} P'$, there exists $Q'$ such that $Q' \xrightarrow{\tau} Q'$ and $P' \sim Q'$; thus $Q \xrightarrow{\tau} Q'$.

4. Relating $C,A$ with $S$

The labels of the standard early LTS $S$ give more refined observations than the labels of $C,A$ since the output transitions in $C,A$ involve the instantiation of arbitrary terms of type $\mathcal{N} \rightarrow \mathcal{R}$. Here we relate the standard LTS (Fig 5) with the combined LTS (Fig 9); the main result, Theorem 4.8, is that the bisimilarities coincide. In §5 we shall show that actually the $S$ LTS can also be presented in a modular way by refining the context-actions $A$. Our results in these two sections depend on the presence of the $i$ - then $- else$ - operator in the language (or, equivalently, the match and mismatch prefix in other presentations). In a calculus without such an operator bisimilarity on $S$ does not agree with barb-congruence [9]. However, the proof of Theorem 3.9 does not rely on the ability to compare names, hence $C,A$ gives a canonical coinductive proof method for contextually-defined equivalence also in such a setting.

The following lemma is an immediate consequence of the structural Lemmas 2.9, 3.2 and 3.4. Notice that to relate the $\tau$ labelled transitions of $C,A$ with those of $S$ it suffices to consider those of $C$.

Lemma 4.1.

(i) $(\Gamma \vdash \rho) \xrightarrow{k,l} (\Gamma \vdash \rho')$ iff $(\Gamma \vdash \rho) \xrightarrow{k,l} (\Gamma \vdash \rho')$.

(ii) $(\Gamma \vdash \rho) \xrightarrow{k,l} (\Gamma \vdash \rho')$ iff $(\Gamma \vdash \rho) \xrightarrow{k,l} (\Gamma \vdash \rho')$.

The relationship between the various output transitions (kl$U$ in $C,A$ and kl$!x$ in $S$) is a little more involved as spelt out by the following lemma, the proof of which relies on the aforementioned structural lemmas.

Lemma 4.2. $(\Gamma \vdash \rho) \xrightarrow{k,l} (\Gamma \vdash \rho')$ is derivable in $C,A$ iff there exist $\tau \in \mathcal{S}$, $R$ so that $\rho' \equiv \pi_\tau((\Gamma \vdash \rho))$ and exactly one of the following holds:

(i) $l \notin \tau$ and $(\Gamma \vdash \rho') \xrightarrow{k,l} (\Gamma \vdash \rho')$ with $\rho'' \equiv \pi_\tau(R)$;

(ii) $l \in \tau$ and $(\Gamma \vdash \rho') \xrightarrow{k,l} (\Gamma \vdash \rho')$ with $\rho'' \equiv \pi_\tau(R)$ and $\underline{w} := \underline{w}^m$.

The conclusion of Proposition 4.3 below is that the observations in $S$ are at least as discriminating as in $C,A$. The only complication in the proof is the fact that the output label of $C,A$ allows one to pass a name to an arbitrary term of type $\mathcal{N} \rightarrow \mathcal{R}$.

Proposition 4.3. $\sim_{S} \subseteq \sim_{C,A}$.

Proof. We will show directly that $\sim$ is a $C,A$-bisimulation. There are three cases, depending on the kind of the labeled transition.

Case 1: Input $(\Gamma \vdash \rho) \xrightarrow{k,l} (\rho \vdash \rho')$. Obvious by Lemma 4.1.

Case 2: Output labelled transition $(\Gamma \vdash \rho) \xrightarrow{k,l} (\rho \vdash \rho')$. Using Lemma 4.2 we have that $\rho'' \equiv \pi_\tau((\Gamma \vdash \rho))$ and either:

Case 2a: $l \notin \tau$ and $(\Gamma \vdash \rho') \xrightarrow{k,l} (\Gamma \vdash \rho')$. We have $\rho'' \equiv \pi_\tau((\Gamma \vdash \rho))$ so substitution is capture avoiding, and using the assumption we have $(\Gamma \vdash \rho') \xrightarrow{k,l} (\Gamma \vdash \rho')$.

By Proposition 2.9 we have $Q'' \equiv \pi_\tau(R'') \equiv \pi_\tau(R')$ for some $\mathcal{T}$, $R$, $S'$ such that $k,l \notin \tau$ and so $Q'' \equiv \pi_\tau(R')$. But then by Lemma 4.2 we have $(\Gamma \vdash \rho) \xrightarrow{k,l} (\rho \vdash \rho')$. Now since $\sim_{S}$ is a congruence w.r.t (cf Proposition 2.10) we have $(\Gamma \vdash \rho' \equiv \pi_\tau(R')) \sim \rho$.

Case 2b: $l \in \tau$ and $(\Gamma \vdash \rho') \xrightarrow{k,l} (\rho \vdash \rho')$. Then $(\Gamma \vdash \rho') \xrightarrow{k,l} (\rho \vdash \rho')$ such that $(\Gamma \vdash \rho' \equiv \pi_\tau(R')) \sim S$. Then $Q'' \equiv \pi_\tau(R'' \equiv \pi_\tau(R')$ for some $\mathcal{T}$, $R'$ and $S''$. Then $(\Gamma \vdash \rho') \xrightarrow{k,l} (\rho \vdash \rho')$ such that $(\Gamma \vdash \rho' \equiv \pi_\tau(R'' \equiv \pi_\tau(R'))$ and so $(\Gamma \vdash \rho' \equiv \pi_\tau(R'' \equiv \pi_\tau(R'))$.

Case 3: Tau $(\Gamma \vdash \rho) \xrightarrow{k,l} (\rho \vdash \rho')$. Immediate by Lemma 4.1.

To show that the converse inclusion holds we need to show that the observations of $C,A$ are enough in order to observe the identity of the name outputted and whether this name is fresh. The following results confirm this fact; while in the $S$-LTS such observations are basic, in $C,A$ we shall consider certain traces. For any terms of type $m$ and $n$ let:

$Lk(m,n) \equiv m!n \quad Eq(m,n) \equiv \lambda x. if \ x = m \ then \! n \ else \! 0$

given a finite set of terms of name type $M = \{m_0, \ldots , m_l\}$ and a name term $n$ let:

$Neq(M,n) \equiv \lambda x. N_i$

where $N_i$ is defined inductively as follows:

$N_0 \equiv if \ x = m_0 \ then \! 0 \ else \! n!$

$N_{k+1} \equiv if \ x = m_{k+1} \ then \! 0 \ else \! N_k$

The following Lemmas are easily shown to hold using the structural Lemmas 2.9 and 3.4.

Lemma 4.4 (Free output trace). Let $c$ be a name constant fresh for $P$. The following are equivalent:

(i) there exists a transition $(\Gamma \vdash \rho) \xrightarrow{k,l} (\Gamma \vdash \rho')$;

(ii) there exists a trace $(\Gamma \vdash \rho) \xrightarrow{k,l} (\Gamma \vdash \rho')$ in $C,A$.

Lemma 4.5 (Bound output trace). Let $c$ be a name constant fresh for $P$ and let $T$ be a set of names of type such that $\{k \mid k \in P\} \subseteq T$. Then the following are equivalent:

(i) there exists a transition $(\Gamma \vdash \rho \equiv \pi_\tau(P)) \xrightarrow{k,l} (\Gamma \vdash \rho')$;

(ii) there exists a trace $(\Gamma \vdash \rho \equiv \pi_\tau(P)) \xrightarrow{k,l} (\Gamma \vdash \rho')$.

In general $(\Gamma \vdash x \vdash P) \sim (\Gamma \vdash x \vdash Q)$ does not imply $(\Gamma \vdash x \vdash P) \sim (\Gamma \vdash x \vdash Q)$. However, the following lemma shows that it holds in the case when the bound name can be leaked to the context.

Lemma 4.6. Let $c$ be a name constant fresh for $P$, $Q$ and suppose $(\Gamma \vdash x \vdash (P \parallel Lk(c,x))) \sim (\Gamma \vdash x \vdash (P \parallel Lk(c,x)))$. Then $(\Gamma \vdash x \vdash P) \sim (\Gamma \vdash x \vdash Q)$.

Proof. It suffices to show that $R$, defined below, is a bisimulation.

$R = \{(\Gamma \vdash x \vdash P), (\Gamma \vdash x \vdash Q) \mid (\Gamma \vdash x \vdash (P \parallel Lk(c,x))) \sim (\Gamma \vdash x \vdash (Q \parallel Lk(c,x)))\}$

The only interesting cases are input and output transitions along $x$.

(In) If $(\Gamma \vdash x \vdash P) \xrightarrow{k,l} (\Gamma \vdash x \vdash Q)$ (note that $k,l$ or $m$ may be $x$) then:

$(\Gamma \vdash x \vdash (P \parallel Lk(c,x))) \xrightarrow{\alpha_{Lk(c,x)}(k,l)} (\Gamma \vdash x \vdash (P \parallel Lk(c,x)))$

and by the assumption we have a corresponding trace

$(\Gamma \vdash x \vdash (Q \parallel Lk(c,x))) \xrightarrow{\alpha_{Lk(c,x)}(k,l)} (\Gamma \vdash x \vdash (Q \parallel Lk(c,x)))$

such that $(\Gamma \vdash x \vdash (P \parallel Lk(c,x))) \sim (\Gamma \vdash x \vdash (Q \parallel Lk(c,x)))$.

The existence of the second trace implies $(\Gamma \vdash x \vdash Q) \xrightarrow{k,l} (\Gamma \vdash x \vdash Q')$. 

2007/7/17
(Out) If \((\Gamma \triangleright x : P) \xrightarrow{k[U]} (\Gamma' \triangleright x : P')\) then (with \(x\) possibly in \(k\) and \(U\), which may be captured by the \(\lambda\)-binding in the transitions below) there is a trace

\[
(\Gamma \triangleright \nu x(P)(\nu k(c,x))) \xrightarrow{e!\lambda x.k?z.\langle U(z)\rangle} (\Gamma' \triangleright \nu x(P'(\nu k(c,x)))
\]
and so

\[
(\Gamma \triangleright \nu x(Q)(\nu k(c,x))) \xrightarrow{(e!\lambda x.k?z.\langle U(z)\rangle)} (\Gamma' \triangleright \nu x(Q'(\nu k(c,x)))
\]
which implies the existence of \((\Gamma' \triangleright Q')\).

**Proposition 4.7.** \(\sim_{CA} \subseteq \sim_S\).

**Proof.** We shall show that \(\sim_{CA}\) is an \(S\)-bisimulation, which shows that \(\sim_{CA} \subseteq \sim_S\). The cases of input and \(\tau\)-labelled transitions are immediate by Lemma 4.1.

If \((\Gamma \triangleright P) \xrightarrow{k!S} (\Gamma' \triangleright P')\), then using Lemma 4.4, we obtain a trace \((\Gamma \triangleright P) \xrightarrow{k!E\langle l,c \rangle} (\Gamma' \triangleright P')\), thus a trace \((\Gamma \triangleright Q) \xrightarrow{\alpha_0} (\Gamma' \triangleright Q')\) such that \(P' \sim_{CA} Q'\), finally using Lemma 4.4 we have \((\Gamma \triangleright P) \xrightarrow{k!S} (\Gamma' \triangleright Q')\).

Suppose that \((\Gamma \triangleright P) \xrightarrow{k!\langle c,x \rangle} (\Gamma' \triangleright P')\). By Lemma 4.5, there exists a trace \((\Gamma \triangleright P) \xrightarrow{k!\nu\nu e\langle T,c \rangle} (\Gamma' \triangleright \nu e\langle P'(\nu k(c,x))\rangle)\) and so

\[
(\Gamma \triangleright P) \xrightarrow{k!\nu\nu e\langle T,c \rangle \nu\nu e\langle \nu k(c,x) \rangle} (\Gamma' \triangleright \nu e\langle P'(\nu k(c,x))\rangle)
\]
which is matched by a trace

\[
(\Gamma \triangleright Q) \xrightarrow{k!\nu\nu e\langle T,c \rangle} (\Gamma' \triangleright \nu e\langle Q'(\nu k(c,x))\rangle)
\]
for some \(Q'\) such that \((\Gamma \triangleright \nu e\langle P'(\nu k(c,x))\rangle) \sim_{CA} (\Gamma \triangleright \nu e\langle Q'(\nu k(c,x))\rangle)\). By Lemma 4.6 we have \((\Gamma \triangleright P') \sim_{CA} (\Gamma \triangleright Q')\) and the existence of trace (2) implies that there is a trace \((\Gamma \triangleright Q) \xrightarrow{k!\langle c,x \rangle} (\Gamma \triangleright \nu e\langle P'(\nu k(c,x))\rangle)\) which by Lemma 4.5 gives \((\Gamma \triangleright Q) \xrightarrow{k!S} (\Gamma \triangleright Q')\).

Immediate from Propositions 4.3 and 4.7 is that the two equivalences coincide.

**Theorem 4.8.** Standard bisimilarity coincides with bisimilarity on the \(CA\) LTS. That is, for all closed terms \(P, Q\) of type \(\mathcal{P}\), we have \(P \sim S Q\) iff \(P \sim_{CA} Q\).

\section{5. Refining context actions}

As hinted at by the conclusions of Lemmas 4.4 and 4.5, there is a certain amount of redundancy in allowing context to instantiate a \(N \rightarrow \mathcal{P}\) binding which arises as a result of an output transition. In this section we shall present a refined system of context actions which instantiates only one process – a meta-process which accepts a name and “announces” or “leaks” it; the label \(k\) is then an abbreviation for the observation of a successful interaction of a context which tests the identity of the name, while the label \(\nu\) is an abbreviation for the observation of a fresh name.

Technically we do this by introducing a new part of the meta-language – a term \(k\) of process type where \(k\) is a term of name type; the syntax and the addition type rule are presented in the upper section of Fig. 10. This meta-term interacts with the syntactic features of \(P\) as shown by rules \((\text{Lk})\), \((\text{Ll})\), \((\text{ELk})\), and \((\text{FRSH})\) in the lower part of Fig. 10. The rule \((\text{FRSH})\) is related to the \((\text{OPEN})\) rule in \(S\). Instead of allowing an instantiation by an arbitrary process, \((\text{INSTPRC})\) simply relays the observation of the meta-process \(\lambda x.x\); name bindings are dealt canonically, as shown by \((\text{INSTNAME})\). We give the structural lemma for \(R\) below, we leave the obvious type preservation lemma to the reader.

**Lemma 5.1** (Structural, \(R\)).

**Figure 10.** Refined context actions: syntax and rules (\(R\)).

(i) A transition \((\Gamma \triangleright P) \xrightarrow{k!} (\Gamma' \triangleright P')\) is derivable iff there exist \(\nu\) and \(R\) with \(k \notin \nu\) such that \(P \equiv \nu \nu\langle k!x.R \parallel S\rangle\) and \(P' \equiv \nu \nu\langle \nu \nu\langle k!x.R \parallel S\rangle\).

(ii) A transition \((\Gamma \triangleright T) \xrightarrow{k!} (\Gamma' \triangleright T')\) is derivable iff there exist \(\nu\) and \(R\) with \(k \notin \nu\) such that \(T \equiv \lambda X.\nu \nu\langle X(k) \parallel R\rangle\) and \(T' \equiv \nu \nu\langle \nu \nu\langle X(k) \parallel R\rangle\).

(iii) A transition \((\Gamma \triangleright P) \xrightarrow{(x)!} (\Gamma' \triangleright P')\) is derivable iff there exist \(\nu\) and \(R\) such that \(P \equiv \nu \nu\langle x.R \parallel \nu \nu\langle x.R \parallel S\rangle\) and \(P' \equiv \nu \nu\langle \nu \nu\langle x.R \parallel \nu \nu\langle x.R \parallel S\rangle\).

(iv) A transition \((\Gamma \triangleright T) \xrightarrow{(x)!} (\Gamma' \triangleright T')\) is derivable iff there exist \(k, l\), \(\nu\), \(R\), \(S\) and \(T\) such that \(P \equiv \nu \nu\langle k!x.R \parallel k?x.S \parallel T\rangle\) and \(P' \equiv \nu \nu\langle \nu \nu\langle k!x.R \parallel k?x.S \parallel T\rangle\).

**Proof.** Immediate from Lemmas 3.2 and 5.1.

We shall consider the system \(C\mathcal{R}\) which is obtained by combining the \(C\) system with the refined system \(R\), analogously to how the system \(CA\) was obtained via the rules presented in Fig. 9.

**Lemma 5.2** (Structural, \(C\mathcal{R}\)).

(i) A transition \((\Gamma \triangleright P) \xrightarrow{k!} (\Gamma' \triangleright P')\) is derivable iff there exist \(\nu\), \(R\) and \(S\) with \(k \notin \nu\) such that \(P \equiv \nu \nu\langle k!x.R \parallel S\rangle\) and \(P' \equiv \nu \nu\langle \nu \nu\langle k!x.R \parallel S\rangle\).

(ii) A transition \((\Gamma \triangleright T) \xrightarrow{k!} (\Gamma' \triangleright T')\) is derivable iff there exist \(\nu\) and \(R\) with \(k \notin \nu\) such that \(T \equiv \lambda X.\nu \nu\langle X(k) \parallel R\rangle\) and \(T' \equiv \nu \nu\langle \nu \nu\langle X(k) \parallel R\rangle\).

(iii) A transition \((\Gamma \triangleright P) \xrightarrow{(x)!} (\Gamma' \triangleright P')\) is derivable iff there exist \(\nu\) and \(R\) with \(k \notin \nu\) such that \(P \equiv \nu \nu\langle x.R \parallel \nu \nu\langle x.R \parallel S\rangle\) and \(P' \equiv \nu \nu\langle \nu \nu\langle x.R \parallel \nu \nu\langle x.R \parallel S\rangle\).

(iv) A transition \((\Gamma \triangleright T) \xrightarrow{(x)!} (\Gamma' \triangleright T')\) is derivable iff there exist \(P \equiv \nu \nu\langle k!x.R \parallel k?x.S \parallel T\rangle\) and \(P' \equiv \nu \nu\langle \nu \nu\langle k!x.R \parallel k?x.S \parallel T\rangle\).

**Proof.** Immediate from Lemmas 3.2 and 5.1.

\(\square\)
and coincide, we can conclude that the transition systems are equal.

Perhaps more remarkably, the notions of bisimulation as it relies on canonical context actions which are not susceptible to variations in the presence of language operators such as matching/mismatching.

As for the first-order case though, the definition of context bisimulation is unattractive due to its reliance upon context actions containing arbitrary (typed) process terms. It is known [25, 35] that context bisimulation can however be refined to so-called ‘normal’ bisimulation, much in the same way as the \( \mathcal{C}A \) system is refined to \( \mathcal{R} \) by using a limited form of context action. We give the rules for the second-order refined system \( \mathcal{R}^{\text{asy}} \), in Fig 11. In this case however, we need to adjust the completed actions system, \( \mathcal{C}A \), to include the rule

In essence, the notion of bisimilarity yielded by \( \mathcal{C}A \) for the second-order language is Sangiorgi’s context bisimilarity [35]. It is therefore interesting to note by analogy that \( \sim_{\mathcal{C}A} \) also provides a definition of context bisimulation for the first-order Pi-calculus.

Because the conclusions of the structural Lemmas 2.9 and 5.2 coincide, we can conclude that the transition systems are equal.

**Corollary 5.3.** \( \mathcal{C}R = \mathcal{S} \).

In particular, we have given a derivation system for the standard first-order LTS in a modular way without using the \( \text{OPEN}/\text{CLOSE} \) mechanism. The two semantic roles of \( \nu \) are clearly separated in the modular approach – scope extrusion is dealt with by the process view, while observation of fresh names in the context view. As an immediate corollary of Theorem 4.8 and Lemma 5.3 we obtain:

**Corollary 5.4.** \( \sim_{\mathcal{C}R} = \sim_{\mathcal{C}A} \).

### 6. Modular variants of the Pi-calculus

In order to demonstrate to some extent how the labelled transition system \( \mathcal{C}A \) presented above can be seen as a modular system, we now apply the same ideas to two variants of the Pi-calculus: the higher-order Pi-calculus and the asynchronous Pi-calculus. For the former, it should be of no surprise that this can be done as the original LTSs for the higher-order Pi-calculus are presented using continuations and abstractions and so avoid difficulties with scope extrusion [35]. For the asynchronous language only the communication fragment differs and thus we expect to isolate any changes to the LTS to that for the process-view \( \mathcal{C} \).

#### 6.1 The higher-order Pi-calculus

Following [35], to simplify the presentation we will actually present the LTS for the second-order Pi-calculus and indicate the further modifications required to treat the full higher-order language.

Firstly, to define the second-order Pi-calculus we simply need to modify the type system in Fig 2 to allow the communication of process terms rather than names. The new type rules for input and output are:

\[
\frac{\Gamma \vdash M : \mathcal{P}}{\Gamma \vdash \text{in} \mathcal{A} \cdot M : \mathcal{P}} \quad (\text{OUTPREF})
\]

and

\[
\frac{\Gamma \vdash k \cdot R \cdot M : \mathcal{P} \quad \Gamma \vdash k \cdot \mathcal{A} \cdot \mathcal{P} \quad \Gamma \vdash k \cdot x \cdot \mathcal{P} \quad \Gamma \vdash \text{in} \cdot M \cdot \mathcal{P}}{\Gamma \vdash \text{in} \cdot \mathcal{P}} \quad (\text{INSREF})
\]

Now, remarkably, modulo types, the LTS in Fig 7 needs no modifications whatsoever. That is, up to typing, the CCS style communication core is identical for both first and higher-order languages. Perhaps more remarkably, the LTS in Figs 8,9 need no modifications either. The differences between the first- and second-order languages are dealt with using types alone.

Figure 11. Refined context actions: second-order (\( \mathcal{R}^{\text{asy}} \)).

### 6.2 The asynchronous Pi-calculus

There is an established [10,23] presentation of the variant of the Pi-calculus in which all communication is done asynchronously. This involves simply restricting the syntax of the language such that the residual of any output prefix is the nil process:

\[
\text{M!M0}
\]

The obvious effect of this is such that no process can be blocked waiting on a send of data. A less obvious effect is that this language restriction actually impacts upon the behavioural theory of the language considerably and makes receiving an unobservable action. This is well-accounted for in the literature [4], but here we show that a simple modification to the communication module \( \mathcal{C} \) only can also account for the change in behavioural theory. This reinforces the viewpoint that the mechanism of naming and scoping in moving to the asynchronous language is orthogonal to the underlying communication mechanism.

To obtain an LTS appropriate for the asynchronous language we define the system \( \mathcal{C}_a \) by adding the following rule to \( \mathcal{C} \):

\[
\frac{}{\Gamma \vdash 0 \rightarrow \Gamma \vdash \text{in} \cdot x \cdot k \cdot x \cdot \mathcal{P}} \quad (\text{aIN})
\]
This $C^\pi$ system can be combined in a modular fashion using the $C^\pi A$ rules with both $A$ and $R$ to yield the corresponding systems $C^\pi R A$ and $C^\pi R$.

The techniques described above can easily be applied to the asynchronous variant of the language also to establish analogous results:

**Proposition 6.1.** For closed processes of the asynchronous $\pi$-calculus, $\sim_{C^\pi A}$ is a symmetric relation.

A similar result establishing correspondence with (early) asynchronous bisimilarity [4] would be possible but we leave the details of this aside for this presentation.

### 6.3 Asynchrony and higher-order combined

In the true spirit of modularity, we also note that we can combine the systems above to obtain suitable LTSs for an asynchronous higher-order $\pi$-calculus. For example, for the second-order language above with the restricted output prefix $M!M_0$, we take systems $C^\pi A$ combined using $C A$ to obtain a canonical LTS ($C^\pi A$).

We can also take $C^\pi+R_\alpha$ combined using $C A$ with (CT) to obtain a refined system ($C^\pi R_\alpha$) for the asynchronous second-order $\pi$-calculus.

### 7. Conclusion and future work

We have demonstrated a design principle associated with LTSs – modularity. The basic goal is that the semantics for different features of a calculus should be given by standard LTS modules which fit together in a largely orthogonal way. Such modules work at the process-level and at the context-level, the glue which ties them together is a typed meta-syntax. We plan to carry out an analysis of the synchronisation disciplines of other calculi analogously to our analysis of the $\Pi$ reductions (cf Fig 6); for example the ambient calculus [13], complementing the work by Merro and Zappa Nardelli [28] on LTS characterisations. We also believe that our approach will be well-suited to a mechanised formalisation akin to ongoing work on the standard early LTS [5].

### References


