Deconstructing behavioural theories of mobility*

Julian Rathke and Paweł Sobociński

ECS, University of Southampton

Abstract. We re-examine the standard structural operational semantics of the π-calculus with the view that both process structure and contextual observational power should play roles in describing the behavioural theory. To that end we provide a decomposition of the operational semantics of π which allows for a systematic definition of labelled transitions. These are derived from the calculus' underlying reduction rules by following the contexts-as-labels philosophy while being presented using the structural approach. Our novel transition system refines to a composite description of the standard early LTS. We generalise our technique to higher-order and asynchronous variants.

Introduction

The π-calculus [6,13] is a foundational model for the study of mobile processes. It has become one of the most well-known and widely studied process calculi and extensions of it are now beginning to be used in a variety of application areas. Each of these applications is typically based on specialising the π-calculus to the particular domain; usually by extending one or more features of the language. However, a weak point of this approach is that 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. This is an undesirable situation and our goal is thus to develop methods by which suitable labelled transition systems for general process languages can be systematically defined based upon *both* structural rules [14] *and* contextual observable power [11]. This will be difficult to achieve in general but in this paper we take our first step by showing our approach to the problem as it applies to some π-calculus variants.

It may seem churlish of us to re-examine the well-established and finely crafted labelled transition semantics of the π-calculus [13] but we believe a deconstructive reading of these will deepen the understanding of LTSs more generally. In particular, we challenge the notion that structure is paramount in giving operational semantics (cf. Plotkin's SOS [14]) and argue that if observational power is also taken in to consideration when building labelled transition systems then the transition systems can be defined in a systematic manner such that the resulting bisimulation equivalences are better suited for characterising

* Research partially supported by EPSRC grant EP/D066565/1.

contextually defined equivalences. In this paper we support this argument by first considering the π-calculus without matching.

Traditional presentations of the LTS of the π-calculus emphasise the structural approach while neglecting observational power. To exemplify this, consider the standard early labelled transition rule for output actions:

$$a!b.P \xrightarrow{a!b} P$$

From a purely structural point of view this rule is perfectly sensible. But, if we consider the underlying reduction rule of π-calculus:

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

we see that the specific name b is parametric in this rule and does not genuinely play a structural role in the transition labelled $a!b$ – any context that provides an input on a enables the passing of any other name. Indeed, in a π-calculus without name-equality tests there actually is no context that justifies the observation of the specific name b being communicated and the standard 'structural' rule is inappropriate. This makes it untenable for us to accept the standard label above as a canonical labelled transition of the π-calculus. However, we can resolve this by focusing on the structure of the underlying reduction rule: we decompose the rule in to structure provided by the process and parameters to the rule provided by the context. In the case for output, the former of these provides a (partial) labelled transition of the form:

$$a!b.P \xrightarrow{a!} \lambda X.(P \parallel X(b))$$

in which the communication, but no further non-structural information, is represented. Here, the contribution from the context (process Q) is abstracted away. The complete labelled transition is then obtained by allowing the context to supply the missing parameters by applying the resulting abstraction above to an arbitrary process. Doing this may or may not subsequently ascertain the identity of b, depending on the power of the contexts of the language. Such output transitions do not rely on subtle observational powers regarding matching – they are generated from the reduction rule alone. A similar approach can be taken for input transitions and thus an entire LTS can be derived systematically thereby avoiding ad hoc solutions.

The focus on the relationship of an LTS with an underlying reduction system is clearly shared with previous work on systematically *deriving* transition systems from reductions [10, 11, 18, 19]. Indeed, the labels of our derived LTS have corresponding contexts which justify them from the point of view of the contexts-as-labels approach. This does not hold for the standard early LTS as for instance there is no context that accounts for the bound-output label.

In the above-mentioned approaches, labels are defined to be contexts which trigger reduction. However, while tied closely to observability, they suffer from a lack of a structural presentation. Our LTS enjoys the best of both approaches as it is presented structurally yet satisfies the contexts-as-labels criteria.

Related work We use a meta-syntax based on the simply-typed λ-calculus in order to denote terms that have a context-component as a result of an interaction. The technically related approaches in the literature include [4, 20] which use variants of the λ-calculus as a metasyntax. The difference in approaches arises from different underlying goals: the aforementioned works use the meta-syntax to study systems of SOS rules a posteriori, we are interested in defining new LTSs.

Milner's [12] approach to capturing the *late* semantics of π using abstractions and concretions is closer to our approach in spirit. Abstractions and concretions are syntactic entities that arise as a result of the complementary roles of inputs and outputs in π. In contrast to Milner, we do not need to define a special notion of application and substitution for abstractions and concretions because our 'concretions' retain structural information from the reduction rules which enables us to use the standard notion of capture-avoiding substitution.

Structure In §1 we introduce our base-calculus, a typed version of π without choice or matching, together with the meta-syntax for expressing partial interactions. In §2 we give the SOS rules which define our LTS and show that ordinary bisimilarity agrees with contextual equivalence. In §3 we show that the SOS methodology can be used also to define the standard early LTS. In §4 we show that the technique naturally generalises to higher-order and asynchronous settings.

1 A simply-typed π-calculus (without matching)

Here we revisit the syntax, structural congruence and the reduction semantics of π in a typed setting. We study a core language without choice and without name equality testing as this serves to demonstrate our point well.

The syntax, the types and the axioms of structural congruence are given in *Fig.* 1. A significant departure from the π-calculus is the inclusion of simply-typed λ-terms and applications in the language. These features are not to be considered as an extension of π-calculus, rather as a *meta-language*. λ-bindings are used as meta-syntax for manipulating terms.

The type system, presented in the upper section of *Fig.* 2 is very simple: there are three base types; a name type Nm and a process type Pr and a unit type $\mathbf{1}$. Our type system is used only in order to formalise the meta-syntax and is simpler than usual π-calculus type systems based on channel types [17, Part III]. We will consider only typeable terms.

A type context consist of a finite set of names Δ together with a finite map Γ from variables to types. We use the notation $\Gamma, x : \sigma$ to mean the context Γ extended with the mapping $x \mapsto \sigma$; implicitly it is always assumed that x is not already in the domain of Γ. Similarly, for names, $\Delta, a = \Delta + \{a\}$. 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

$$\sigma \quad ::= \quad \mathsf{Nm} \mid \mathsf{Pr} \mid \mathbf{1} \mid \sigma{\to}\sigma$$

$$M ::= x \mid a \mid 0 \mid M\|M \mid M!MM \mid M?xM \mid \nu aM \mid \mathrm{rp}(M) \mid 1 \mid \lambda x{:}\sigma.M \mid M(M)$$

$$(P\|Q)\|R \equiv P\|(Q\|R) \qquad P\|Q \equiv Q\|P \qquad P\|0 \equiv P$$

$$\nu a\nu bP \equiv \nu b\nu aP \qquad \nu a0 \equiv 0 \qquad \nu a(P\|Q) \equiv P\|\nu aQ \ \ (a\notin P) \qquad \nu aP \equiv \nu bP[b/a] \ \ (b\notin P)$$

$$\mathrm{rp}(P) \equiv P\|\mathrm{rp}(P) \qquad \mathrm{rp}(P\|Q) \equiv \mathrm{rp}(P)\|\mathrm{rp}(Q) \qquad \mathrm{rp}(0) \equiv 0$$

$$k?xP \equiv k?yP[y/x] \ \ (y\notin fr(P)) \qquad (\lambda x{:}\sigma.M)(N) \equiv M[N/x] \qquad \lambda x{:}\sigma.M \equiv \lambda y{:}\sigma.M[y/x] \ \ (y\notin fr(M))$$

Fig. 1. Types, syntax and structural congruence.

$$\frac{a\in\Delta}{\Delta;\Gamma\vdash a{:}\mathsf{Nm}}\ (:\textsc{Name}) \qquad \frac{\Gamma(x)=\sigma}{\Delta;\Gamma\vdash x{:}\sigma}\ (:\textsc{Var}) \qquad \frac{}{\Delta;\Gamma\vdash 0{:}\mathsf{Pr}}\ (:\textsc{Null}) \qquad \frac{}{\Delta;\Gamma\vdash 1{:}\mathbf{1}}\ (:\textsc{Unit})$$

$$\frac{\Delta;\Gamma\vdash k{:}\mathsf{Nm} \ \ \Delta;\Gamma\vdash l{:}\mathsf{Nm} \ \ \Delta;\Gamma\vdash P{:}\mathsf{Pr}}{\Delta;\Gamma\vdash k!lP{:}\mathsf{Pr}}\ (:\textsc{OutPref}) \qquad \frac{\Delta;\Gamma\vdash k{:}\mathsf{Nm} \ \ \Delta;\Gamma,x\vdash P{:}\mathsf{Pr}}{\Delta;\Gamma\vdash k?xP{:}\mathsf{Pr}}\ (:\textsc{InPref})$$

$$\frac{\Delta,a;\Gamma\vdash P{:}\mathsf{Pr}}{\Delta;\Gamma\vdash \nu aP{:}\mathsf{Pr}}\ (:\textsc{Nu}) \qquad \frac{\Delta;\Gamma\vdash P{:}\mathsf{Pr} \ \ \Delta;\Gamma\vdash Q{:}\mathsf{Pr}}{\Delta;\Gamma\vdash P\|Q{:}\mathsf{Pr}}\ (:\textsc{Par}) \qquad \frac{\Delta;\Gamma\vdash P{:}\mathsf{Pr}}{\Delta;\Gamma\vdash \mathrm{rp}(P){:}\mathsf{Pr}}\ (:\textsc{Rep})$$

$$\frac{\Delta;\Gamma,X{:}\sigma\vdash M{:}\sigma'}{\Delta;\Gamma\vdash \lambda X{:}\sigma.M{:}\sigma{\to}\sigma'}\ (:\lambda) \qquad \frac{\Delta;\Gamma\vdash M{:}\sigma{\to}\sigma' \ \ \Delta;\Gamma\vdash N{:}\sigma}{\Delta;\Gamma\vdash M(N){:}\sigma'}\ (:\textsc{App})$$

Fig. 2. Type rules of first-order π.

constants, k, l for terms of name type (either constants or variables), x, y for variables of name type, X, Y for variables generally, P, Q for terms of process type and M, N for general terms.

A *closed* term V is a typeable term that does not contain free variables – *i.e.* there exist Δ, σ such that $\Delta;\varnothing\vdash V{:}\sigma$ (we will often write just $\Delta\vdash V{:}\sigma$).

Structural congruence $\equiv$ is the smallest relation that satisfies the axioms and is closed under all the syntactic features of the calculus: the output prefix, the input binder, the ν binder, the λ-binder and the parallel composition. Exhibiting the non-computational role of the meta-language, β-reduction is part of structural congruence. Our language contains three binders. Substitution within the β-rule is the usual capture-avoiding notion *with respect to all three*. Structurally congruent terms have the same type.

Definition 1 (Indexed transition system) An indexed transition system $\mathcal{T}$ has states comprising pairs of a set of names Δ and a closed term V which has its free names in Δ; *i.e.* the states are contained in the set $\{\,(\Delta, V) \mid \exists\sigma.\ \Delta\vdash V{:}\sigma\,\}$. We shall use the notation $\langle\,\Delta\rhd V\,\rangle$ to refer to a state. Our transition systems are presented in the structural style. We make one non-standard assumption: we work with abstract syntax and thus assume the implicit presence of the rule:

$$\frac{P'\equiv P \quad \langle\,\Delta\rhd P\,\rangle\xrightarrow{\alpha}\langle\,\Delta'\rhd Q\,\rangle \quad Q\equiv Q'}{\langle\,\Delta\rhd P'\,\rangle\xrightarrow{\alpha}\langle\,\Delta'\rhd Q'\,\rangle}\ (\textsc{StrCng}).$$

The choice of including the rule (StrCng) in our transition system is a technical convenience rather than necessity and greatly reduces the number of rules required. This allows 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 straightforward.

Our first transition system is the reduction semantics for the π-calculus: rule

$$\frac{}{\langle\, \Delta \rhd a!bP \| a?xQ \,\rangle \rightarrow \langle\, \Delta \rhd P \| Q[b/x] \,\rangle}$$

and rules which close the relation under parallel composition and the ν binder. Subject reduction is easily shown using a straightforward induction on the derivation of the transition.

We can give an alternative definition of the reduction relation as the reduction relation of a *reactive system* [10, 11]; this style of definition makes the parametric nature of π-reductions more explicit.

We shall first need to define a general notion of *context*. Contexts are constructed in two stages. Firstly, for each type σ, we add *σ-typed holes* $-_\sigma$ and n-tuples (for any $n \in \mathbb{N}$) to the syntax, together with two additional type rules, given below. We refer to a term $(V_1, \ldots, V_n)$ as a *pre-context*.

$$M ::= \ldots \mid -_\sigma \mid (M,\ldots,M) \qquad \frac{}{\Delta;\Gamma \vdash -_\sigma : \sigma}\ (:\text{Hole}) \qquad \frac{\Delta \vdash V_1:\sigma_1 \ \ldots \ \Delta \vdash V_n:\sigma_n \ (n\in\mathbb{N})}{\Delta \vdash (V_1,\ldots,V_n):[\sigma_1 \ldots \sigma_n]}\ (:\text{Tup}).$$

Secondly, given a pre-context of type $[\sigma_1 \ldots \sigma_n]$ which contains m holes, we replace each hole symbol with a unique integer from 1 to m. Such a numbering uniquely determines a word over the set of types; the ith-letter being the type σ'_i of the ith-numbered hole. We say that the resulting tuple $(V'_1, \ldots, V'_n)$ is a $[\sigma'_1 \ldots \sigma'_m] \rightarrow [\sigma_1 \ldots \sigma_n]$ *context*. Each hole appears exactly once, thus the contexts are linear.

Note that ordinary closed terms of type σ are in 1-1 correspondence (and can be identified with) the $[] \rightarrow [\sigma]$ contexts. A *π-context* is a $[\mathsf{Pr}] \rightarrow [\mathsf{Pr}]$ context that does not contain elements of the meta-language – *i.e.* does not contain λ-terms or applications. An *evaluation context* is a π-context in which the process hole does not appear within the scope of a prefix. Substitution is syntactic: given a $[\overrightarrow{\sigma_1}] \rightarrow [\overrightarrow{\sigma_2}]$ context g and a $[\overrightarrow{\sigma_2}] \rightarrow [\overrightarrow{\sigma_3}]$ context f, $f \circ g$ is the $[\sigma_1] \rightarrow [\sigma_3]$ context obtained by substituting the ith component of g for the ith hole of f. Context substitution may involve free names of g being captured by name restrictions of f. The input-binder cannot capture because, by construction, contexts do not contain free variables. A λ-term of type $\sigma' \rightarrow \sigma$ is inherently different from a context $[\sigma'] \rightarrow [\sigma]$, since the former is possibly non-linear and in the latter the substitution is not capture-avoiding.

In order to consider π as a reactive system, we start with a set of name-indexed reduction rules: pairs l_a, r_a of $[\mathsf{Nm}, \mathsf{Pr}, \mathsf{Nm}\rightarrow\mathsf{Pr}]\rightarrow[\mathsf{Pr}]$ contexts defined

$$l_a \stackrel{\text{def}}{=} a!1_{\mathsf{Nm}}.2_{\mathsf{Pr}} \parallel a?y.3_{\mathsf{Nm}\rightarrow\mathsf{Pr}}(y) \quad \text{and} \quad r_a \stackrel{\text{def}}{=} 2_{\mathsf{Pr}} \parallel 3_{\mathsf{Nm}\rightarrow\mathsf{Pr}}(1_{\mathsf{Nm}}).$$

We construct the reduction relation as follows: $\langle\, \Delta \rhd P \,\rangle \rightarrow \langle\, \Delta \rhd P' \,\rangle$ iff there exist a name a, a $[] \rightarrow [\mathsf{Nm}, \mathsf{Pr}\,\mathsf{Nm} \rightarrow \mathsf{Pr}]$ context p (the parameters) and an evaluation context d such that $P \equiv d \circ l_a \circ p$ and $P' \equiv d \circ r_a \circ p$. The transition system defined is the same relation as given by the inductive, structural presentation.

The reduction semantics naturally leads to a notion of contextually-defined equivalence, the barb-congruence; defined here in the dynamic style [8]. We use the notion of strong barb: the ability to immediately input or output on a particular channel a, denoted $\downarrow_a$. The natural notion of equivalence relation on states of a typed transition system is an indexed relation.

Definition 2 (Indexed relation) A name-indexed relation R is a set of triples (Δ, P, Q) where Δ is a finite set of names and P and Q are closed terms typeable in Δ ($_{\Delta;\varnothing \vdash P,Q:\sigma}$). We write $PR_\Delta Q$ for $(\Delta, P, Q) \in R$.

Definition 3 (Reduction barbed congruence) *Barb-congruence, denoted* $\simeq$*, is the largest symmetric relation that, for any* $P \simeq_\Delta Q$ *is:*

(i) Reduction-preserving: *if* $\langle\, \Delta \rhd P \,\rangle \rightarrow \langle\, \Delta \rhd P' \,\rangle$ *then there exists a reduction* $\langle\, \Delta \rhd Q \,\rangle \rightarrow \langle\, \Delta \rhd Q' \,\rangle$ *such that* $P' \simeq_\Delta Q'$*;*
(ii) Barb-preserving: *if* $\langle\, \Delta \rhd P \,\rangle \downarrow_a$ *then* $\langle\, \Delta \rhd Q \,\rangle \downarrow_a$*;*
(iii) A congruence: *for all* π*-contexts* $\Delta' \vdash C{:}[\mathsf{Pr}]\rightarrow[\mathsf{Pr}]$ *we have* $C \circ P \simeq_{\Delta \cup \Delta'} C \circ Q$*.*

2 A structured LTS

In this section we shall describe our approach to endowing the π-calculus with an LTS. We split a labelled transition which corresponds to an interaction of a term with a context into a process-view of the interaction (*Fig.* 3) and a context-view (*Fig.* 4). The complete LTS is obtained by combining the two (*Fig.* 5). We emphasise that this LTS is obtained systematically from the underlying reduction rule of the π-calculus and, as such, may appear less elegant than an optimised or ad hoc system for the same language.

We begin with the process-view LTS $\mathcal{C}$, with its structural rules given in *Fig.* 3. Here and henceforward we shall use the syntactic convention of writing T for terms of type $(\mathsf{Nm}\rightarrow\mathsf{Pr})\rightarrow\mathsf{Pr}$ and U for terms of type $\mathsf{Nm}\rightarrow\mathsf{Pr}$. First, we focus on output transitions: a process $a!MP$ offering an output on a channel a, matches a subterm of the source of the single π-calculus reduction rule. Thus, in some context, it can engage in an interaction to evolve into (according to the target of the same reduction rule) a process consisting of its continuation P, in parallel with the continuation of the interacting context, Q, say. Furthermore, Q, has been passed the communicated name M. In the process-view, the interacting context is left unspecified and the target of the transition is a λ-abstraction that binds a variable of type $\mathsf{Nm} \rightarrow \mathsf{Pr}$ (cf (Out)).

On the other hand, a process offering an input on a channel a can interact and obtain some unspecified name – the result is a λ-abstraction that binds a variable of type Nm (cf (In)) . A process with both capabilities can perform

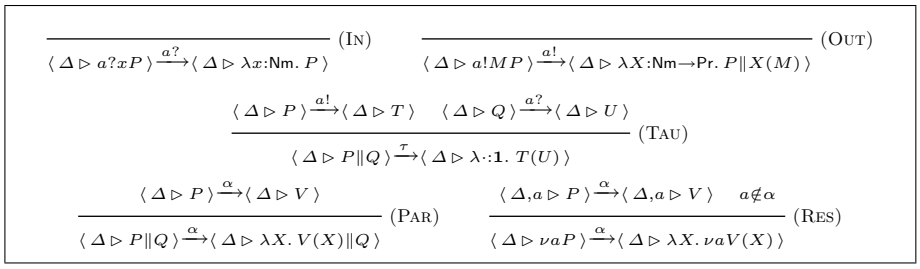

Fig. 3. Process-view fragment ($\mathcal{C}$).

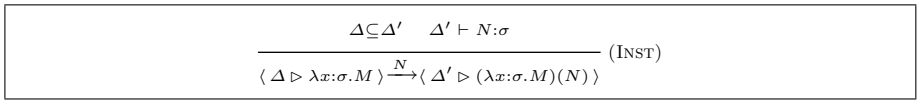

Fig. 4. Canonical context actions ($\mathcal{A}$).

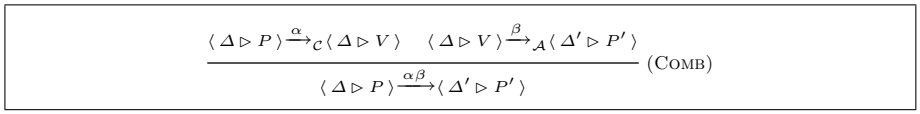

Fig. 5. Combined system of complete actions ($\mathcal{CA}$).

the synchronisation by itself – the abstractions are combined via an application (cf (Tau)). Note that the subtleties of scope extrusion are dealt with cleanly by leveraging the capture-free substitution of the λ-calculus metalanguage. The remaining rules ((Par) and (Res)) account for interaction within evaluation contexts and are purely structural.

Transitions which arise from process terms, as presented in *Fig.* 3, represent the part of the interaction which is controlled by the process. In fact, if the sole purpose of the LTS were to structurally define the reduction relation we could stop here as *Fig.* 3 fulfils this role. However, in order to characterise a contextually defined equivalence as a bisimilarity we need to account for the interactions with arbitrary contexts. The transitions which arise from λ-abstracted terms, presented in *Fig.* 4, represent these parts of interactions controlled by the context. Combining the process and context views allows us to completely describe behaviour in π. We do this in *Fig.* 5 by a simple conjunction of the two views of the interaction. The labels γ here are composite actions ($\alpha\beta$) consisting of both the structural process contribution α and contextual contribution β. The context-view transitions take a very simple form, that is, applicative labels carrying *any* well-typed process. This quantification may appear rather crude, and the LTS does not directly model such things as identities of names but it does at least provide a robust means of defining the context contribution of parameters to the completed labelled transition which is insensitive to subtleties in observational power of the underlying calculus. We return to this point in §3 where

we refine $\mathcal{CA}$ by limiting the contextual contributions β to a more tractable class of terms.

Owing to the construction of the LTS, there are contexts which witness the labels of $\mathcal{CA}$ in the sense that they induce a reduction with the same result; moreover they are parametric in the context component provided in $\mathcal{A}$. Let $\mu_{a?} \stackrel{\text{def}}{=} 1_{\mathsf{Pr}} \parallel a!2_{\mathsf{Nm}}$ and $\mu_{a!} \stackrel{\text{def}}{=} 1_{\mathsf{Pr}} \parallel a?y.2_{\mathsf{Nm}\to\mathsf{Pr}}(y)$.

Lemma 4 (Witness contexts)

(i) If $\langle\, \Delta \rhd P \,\rangle \xrightarrow{\alpha\beta}_{\mathcal{CA}} \langle\, \Delta' \rhd P' \,\rangle$ *then* $\langle\, \Delta' \rhd \mu_\alpha \circ (1_{\mathsf{Pr}}, \beta) \circ P \,\rangle \rightarrow \langle\, \Delta' \rhd P' \,\rangle$ *($\alpha \in \{a?, a!\}$);*
(ii) if $\langle\, \Delta \rhd P \,\rangle \xrightarrow{\tau 1}_{\mathcal{CA}} \langle\, \Delta \rhd P' \,\rangle$ *then* $\langle\, \Delta \rhd P \,\rangle \rightarrow \langle\, \Delta \rhd P' \,\rangle$. □

The converse of the the first part of above lemma does not hold; the context which triggers a reduction may provide redundant parts not vital to the actual interaction with the term. However, a converse relationship between reductions and labels from $\mathcal{CA}$ can be established provided the context can be forced to interact with a term; this is done with the aid of a fresh auxiliary name constant to provide a barb. This relationship is used in order to prove that contextual equivalence is contained in bisimilarity (completeness) and in general does not follow from our systematic derivation.

Lemma 5 (Characterising contexts) *Given a name constant u fresh for P, let $\chi_u(a?b) \stackrel{\text{def}}{=} a!b.u!$, $\chi_u(a!U) \stackrel{\text{def}}{=} a?x.u!.U(x)$ and $\chi_u(\tau 1) \stackrel{\text{def}}{=} u!$. If*

$$\langle\, \Delta \rhd P \| \chi_u(\gamma) \,\rangle \rightarrow \langle\, \Delta \rhd P'' \,\rangle \text{ with } \langle\, \Delta \rhd P'' \,\rangle \downarrow_u$$

and

$$\langle\, \Delta \rhd P'' \| u? \,\rangle \rightarrow \langle\, \Delta \rhd P' \,\rangle \text{ with } \langle\, \Delta \rhd P' \,\rangle \not\downarrow_u$$

then $\langle\, \Delta \rhd P \,\rangle \xrightarrow{\gamma} \langle\, \Delta \rhd P' \,\rangle$. *The converse of this also holds.* □

Given the LTS generated by $\mathcal{CA}$, we can make use of the standard notion of (strong) bisimilarity which we denote by $\sim_{\mathcal{CA}}$. The close relationship of $\mathcal{CA}$ with the underlying reduction system means that it is straightforward to prove that bisimilarity is a congruence.

Proposition 6 *$\sim_{\mathcal{CA}}$ is preserved by all π-contexts.* □

These propositions and the fact that bisimilarity is barb-preserving are sufficient to establish soundness ($\sim_{\mathcal{CA}}$ implies $\simeq$). The conclusion of Lemma 5 also lets us demonstrate completeness ($\simeq$ implies $\sim_{\mathcal{CA}}$) because every label of $\mathcal{CA}$ can be simulated as a barb-sensitive reduction.

Theorem 7 $\sim_{\mathcal{CA}} = \simeq$. □

The reader may like to compare this with a related result in [2] in which sophisticated mappings of extruded names are required in the definition of the LTS. Although the result there captures the equivalence using a more tractable LTS, our approach captures the same equivalence in a systematic way which extends immediately to π-calculus with matching and further variants. It is the robustness of our technique which is of value here.

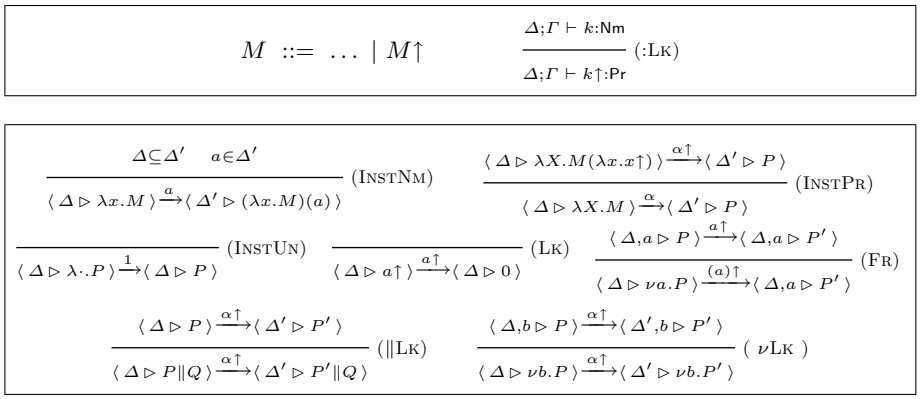

Fig. 6. Refined context actions: syntax and rules ($\mathcal{R}$).

3 Refining context actions

In the calculus above we did not include a choice operator and test for name equality. It can be shown that in this setting standard early bisimilarity does not agree with barbed congruence [2]. However, bisimilarity on our novel LTS does (*cf.* Theorem 7). More significantly though, the proof of this theorem does not rely on the ability to compare names and remains true in the presence of matching.

As is well known, when name equality *is* present in the π-calculus, standard early bisimilarity does happen to provide a labelled characterisation of barbed congruence. In this section we shall show that the early LTS for π can itself be decomposed into our process and context views analogously to the presentation of $\mathcal{CA}$. Although we do not derive this refinement in a systematic manner, it turns out that the standard LTS can still usefully be construed as combination of the base structural rules $\mathcal{C}$ and a refinement of the context-view transitions $\mathcal{A}$. We include the standard early LTS $\mathcal{S}$ in Appendix A for reference.

Technically the refinement is achieved by introducing a new part of the meta-language – a term $k\uparrow$ of process type where k is a term of name type; the syntax and the additional type rule are presented in the upper section of *Fig.* 6. This meta-term interacts with the syntactic features of π as shown by rules (Lk), (‖Lk), (νLk) and (Fr) in the lower part of *Fig.* 6. The label $k\uparrow$ is an abbreviation for the observation of a successful interaction of a context that tests the identity of the name k, while the label $(x)\uparrow$ is an abbreviation for the observation of a fresh name. Thus, the rule (Fr) is related to the (Open) rule in $\mathcal{S}$ but, unlike (Open), this rule is divorced from the underlying communication rules. Instead of allowing an instantiation by an arbitrary process as we find in $\mathcal{A}$, (InstPr) simply relays the observation of the meta-process $\lambda x.x\uparrow$; name bindings are dealt canonically, as shown by (InstNm).

We shall consider the system $\mathcal{CR}$ that is obtained by combining the $\mathcal{C}$ system with the refined system $\mathcal{R}$, analogously to how the system $\mathcal{CA}$ was obtained via

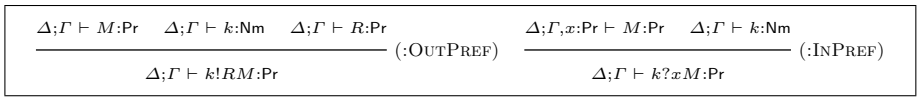

Fig. 7. Type rules for second-order.

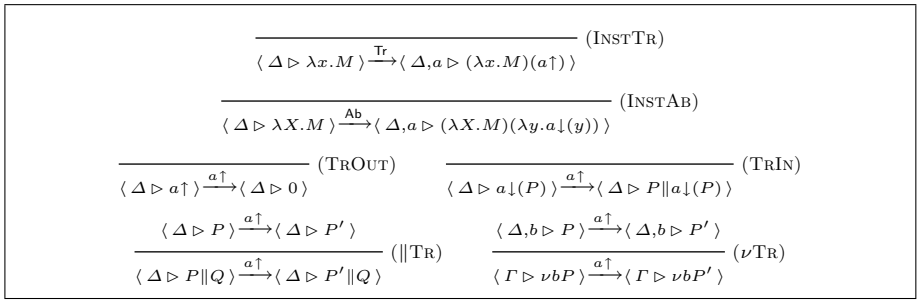

Fig. 8. Refined context actions: second-order ($\mathcal{R}^{so}$).

the rules presented in *Fig.* 5. Using structural analysis it is not difficult to prove that $\mathcal{CR}$ and $\mathcal{S}$ are equal in the sense that (up to minor relabeling on τ transitions) exactly the same transitions are derived.

Theorem 8 $\mathcal{CR} = \mathcal{S}$. □

As an immediate corollary of Theorem 7 (with matching), Theorem 8 and the known result that standard early bisimilarity $\sim_{\mathcal{S}}$ coincides with $\simeq$ we obtain:

Corollary 9 $\sim_{\mathcal{S}} = \sim_{\mathcal{CR}} = \sim_{\mathcal{CA}} = \simeq$. □

4 Modular variants of the π-calculus

We believe that splitting an LTS into a process-view and a context-view, based on the underlying reductions, naturally leads to more modular and robust theories. In order to justify this belief, we now apply these ideas to two variants of the π-calculus: the higher-order π-calculus and the asynchronous π-calculus. For the former, it should be of no surprise that this can be done as the original LTSs for the higher-order π-calculus are presented using concretions and abstractions and so avoid difficulties with scope extrusion [16]. 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 $\mathcal{C}$.

4.1 The higher-order π-calculus

Following [16], to simplify the presentation we will actually present the LTS for the second-order π-calculus In order to define the second-order π-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 given in *Fig.* 7.

Now, remarkably, modulo types, the rules of *Fig.* 3 need 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 rules in *Fig.* 4 and *Fig.* 5 need no modifications either. The differences between the first- and second-order languages are dealt with using types alone.

In essence, the notion of bisimilarity yielded by $\mathcal{CA}$ for the second-order language is Sangiorgi's context bisimilarity [16]. It is therefore interesting to note by analogy that $\sim_{\mathcal{CA}}$ provides a definition of context bisimulation for the first-order π-calculus. As for the first-order case though, context bisimulation is unattractive due to its reliance upon context actions containing arbitrary (typed) process terms. It is known [9, 16] that context bisimulation can be refined to so-called 'normal' bisimulation, much in the same way as the $\mathcal{CA}$ system is refined to $\mathcal{R}$ by using a limited form of context action. We give the rules for the second-order refined system $\mathcal{R}^{so}$, in *Fig.* 8. In this case however, we need to adjust the completed actions system, $\mathcal{CA}$, to include the rule

$$\frac{\langle\, \Delta \rhd P \,\rangle \xrightarrow{a\uparrow}_{\mathcal{R}^{so}} \langle\, \Delta \rhd P' \,\rangle}{\langle\, \Delta \rhd P \,\rangle \xrightarrow{a\uparrow} \langle\, \Delta \rhd P' \,\rangle}\ (\text{CTR})$$

and by combining $\mathcal{C}$ and $\mathcal{R}^{so}$ using this augmented $\mathcal{CA}$, we obtain the refined system $\mathcal{CR}^{so}$ for second-order π-calculus.

In order to present the refined system, again we introduce an augmented meta-language. This takes the form of processes $a\uparrow : \mathsf{Pr}$ and process abstractions $a\downarrow : \mathsf{Pr} \to \mathsf{Pr}$. These are entirely analogous to the triggers and abstractions of [16]. The behaviour of the 'trigger' process $a\uparrow$ is simply to announce itself whenever it is executed and the 'abstraction' $a \downarrow (P)$ will repeatedly allow copies of P to be accessed by calling on the name a. We know from [9, 16] that these syntactic gadgets actually have real syntactic counterparts in the second-order language and are in effect just macros. We also know from [9, 16] that bisimulation equivalence generated by this refined LTS ($\sim_{\mathcal{CR}^{so}}$) coincides with context bisimilarity over second-order processes ($\sim_{\mathcal{CA}}$).

4.2 The asynchronous π-calculus

There is an established [3, 7] presentation of the variant of the π-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. 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 inputs unobservable. This is well-accounted for in the literature [1], 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 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 $\mathcal{C}$:

$$\frac{\langle\, \Delta \rhd P \,\rangle \xrightarrow{\tau} \langle\, \Delta \rhd P' \,\rangle}{\langle\, \Delta \rhd P \,\rangle \xrightarrow{a?} \langle\, \Delta \rhd \lambda x.\ P' \| a!x \,\rangle} \text{(AIN)}$$

This $\mathcal{C}^a$ system can be combined using the $\mathcal{CA}$ rules with both $\mathcal{A}$ and $\mathcal{R}$ to yield the corresponding systems $\mathcal{C}^a\mathcal{A}$ and $\mathcal{C}^a\mathcal{R}$ and the techniques described above can easily be applied to the asynchronous variant of the language also to establish analogous results.

Conclusion and future work

We have provided a new way of understanding the well-trodden labelled transition semantics of the π-calculus. Our approach attempts to combine the structural approach to semantics with the contexts-as-labels approach in order to obtain systematically defined labelled transition systems for process calculi. In this paper we have shown that this approach works very well for the π-calculus and we believe that the technique is robust and widely applicable. We have applied our approach to the ambient calculus [5] for which we have obtained a new, systematically derived, labelled transition system along with a sound and complete context-bisimilarity for that language [15]. Furthermore we plan to develop a general setting for our approach to pursue the synthesis of labelled transition systems from reduction rules in the spirit of [10, 11, 18, 19].

References

1. R. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous pi-calculus. *Theor. Comput. Sci.*, 195(2):291–324, 1998.
2. M. Boreale and D. Sangiorgi. Bisimulation in name-passing calculi without matching. In *13th LiCS*, 1998.
3. G. Boudol. Asynchrony and the π-calculus. Technical Report 1702, INRIA, Sophia-Antipolis, 1991.
4. R. Bruni and U. Montanari. Cartesian closed double categories, their lambda-notation, and the Pi-calculus. In *14th LiCS*, 1999.
5. L. Cardelli and A. Gordon. Mobile ambients. In *Proc. Foundations of Software Science and Computation Structures, FoSSaCS*, LNCS. Springer-Verlag, 1998.
6. U. Engberg and M. Nielsen. A calculus of communicating systems with label passing. Technical Report DAIMI PB-208, University of Aarhus, May 1986.
7. K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *5th ECOOP*, 1991.

8. K. Honda and N. Yoshida. On reduction-based process semantics. *Theor. Comput. Sci.*, 152(2):437–486, 1995.
9. A.S.A. Jeffrey and J. Rathke. Contextual equivalence for higher-order pi-calculus revisited. *Logic. Meth. Comput. Sci.*, 1(1:4), 2005.
10. B. Klin, V. Sassone, and P. Sobociński. Labels from reductions: towards a general theory. In *1st Calco*, volume 3629 of *LNCS*, pages 30–50. Springer, 2005.
11. J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. In *11th Concur*, volume 1877 of *LNCS*, pages 243–258. Springer, 2000.
12. R. Milner. *Communicating and Mobile Systems: the π-calculus.* CUP, 1999.
13. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, II. *Inf. Comput.*, 100(1):41–77, 1992.
14. G. D. Plotkin. A structural approach to operational semantics. Technical Report FN-19, DAIMI, Computer Science Department, Aarhus University, 1981.
15. J. Rathke and P. Sobociński. Deriving structural labelled transitions for mobile ambients, 2008. Submitted for publication.
16. D. Sangiorgi. Bisimulation for higher-order process calculi. *Inf. Comput.*, 131(2):141–178, 1996.
17. D. Sangiorgi and D. Walker. *The π-calculus.* CUP, 2001.
18. V. Sassone and P. Sobociński. Locating reaction with 2-categories. *Theor. Comp. Sci.*, 333(1-2):297–327, 2005.
19. P. Sewell. From rewrite rules to bisimulation congruences. *LNCS*, 1466:269–284, 1998.
20. A. Ziegler, D. Miller, and C. Palamidessi. A congruence format for name-passing calculi. In *2nd SOS*, volume 156 of *ENTCS*, pages 169–189, 2006.

A Standard early labelled transitions, typed ($\mathcal{S}$)

For reference purposes, we list below the standard early labelled transition system semantics for the π-calculus. They have been adapted to our typed setting but remain substantially the same as can be found in say, [17].

$$\frac{\Delta \subseteq \Delta' \quad b \in \Delta'}{\langle\, \Delta \rhd a?xP \,\rangle \xrightarrow{a?b} \langle\, \Delta' \rhd P[b/x] \,\rangle}\ (\textsc{In}) \qquad \frac{}{\langle\, \Delta \rhd a!bP \,\rangle \xrightarrow{a!b} \langle\, \Delta \rhd P \,\rangle}\ (\textsc{Out})$$

$$\frac{\langle\, \Delta \rhd P \,\rangle \xrightarrow{a!b} \langle\, \Delta \rhd P' \,\rangle \quad \langle\, \Delta \rhd Q \,\rangle \xrightarrow{a?b} \langle\, \Delta \rhd Q' \,\rangle}{\langle\, \Delta \rhd P \| Q \,\rangle \xrightarrow{\tau} \langle\, \Delta \rhd P' \| Q' \,\rangle}\ (\textsc{Comm})$$

$$\frac{\langle\, \Delta, b \rhd P \,\rangle \xrightarrow{a!b} \langle\, \Delta, b \rhd P' \,\rangle \ (a \neq b)}{\langle\, \Delta \rhd \nu b P \,\rangle \xrightarrow{a!(b)} \langle\, \Delta, b \rhd P' \,\rangle}\ (\textsc{Open}) \qquad \frac{\langle\, \Delta \rhd P \,\rangle \xrightarrow{a!(b)} \langle\, \Delta, b \rhd P' \,\rangle \quad \langle\, \Delta \rhd Q \,\rangle \xrightarrow{a?b} \langle\, \Delta, b \rhd Q' \,\rangle}{\langle\, \Delta, b \rhd P \| Q \,\rangle \xrightarrow{\tau} \langle\, \Delta \rhd \nu b (P' \| Q') \,\rangle}\ (\textsc{Close})$$

$$\frac{\langle\, \Delta \rhd P \,\rangle \xrightarrow{\alpha} \langle\, \Delta' \rhd P' \,\rangle}{\langle\, \Delta \rhd P \| Q \,\rangle \xrightarrow{\alpha} \langle\, \Delta \rhd P' \| Q \,\rangle}\ (\textsc{Par}) \qquad \frac{\langle\, \Delta, a \rhd P \,\rangle \xrightarrow{\alpha} \langle\, \Delta', a \rhd P' \,\rangle \quad a \notin \alpha}{\langle\, \Delta \rhd \nu a P \,\rangle \xrightarrow{\alpha} \langle\, \Delta' \rhd \nu a P' \,\rangle}\ (\textsc{Res})$$