A well-behaved LTS for the Pi-calculus (Abstract)

Paweł Sobociński[1]

ECS, *University of Southampton, UK*

The Pi-calculus [2,10] is one of the most well-known and widely-studied process calculi – roughly, it extends the binary synchronisations along a channel/name, familiar from CCS, with the ability to pass names as arguments. Thus, the input prefix becomes a binder and the synchronisation itself results in a substitution of the communicated name for the bound variable.

The Pi-calculus inherits another binder from CCS – the restriction operator. The ability to pass names as part of a synchronous communication means that it behaves rather differently in this setting - in particular, the scope of a restriction, static in CCS, becomes dynamic essentially because restricted names can be communicated along public channels. Additionally, it behaves somewhat like a global generator of new-names - since α-conversion ensures that whichever concrete name is chosen for a restricted name, it is different from all other names in the term – in fact, the global nature of new names is also enforced in the definition of bisimulation as we shall recall below. See also [12].

The (reduction) semantics of the Pi-calculus is very similar to that of CSS, 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 – reduction is not allowed under prefix. The reduction semantics naturally leads to a contextually defined equivalence: the barbed congruence. While canonical, contextually defined equivalences are difficult to reason about directly and for this reason it is helpful to define another "semantics": the so-called early labelled transition system (LTS). The labels of the transition system aim at classifying the possible interactions with the environment. Early congruence coincides with barbed congruence, but the LTS

[1] Research partially supported by EPSRC grant EP/D066565/1.

is much easier to use because of the power of coinduction.

$$\frac{}{a?xP \xrightarrow{a?b} P[b/x]}\ (\textsc{In}) \qquad \frac{}{a!bP \xrightarrow{a!b} P}\ (\textsc{Out}) \qquad \frac{P \xrightarrow{a!b} P' \quad Q \xrightarrow{a?b} Q'}{P\|Q \xrightarrow{\tau} P'\|Q'}\ (\textsc{Comm})$$

$$\frac{P \xrightarrow{a!b} P' \quad a\neq b}{\nu bP \xrightarrow{a!(b)} P'}\ (\textsc{Open}) \qquad \frac{P \xrightarrow{a!(b)} P' \quad Q \xrightarrow{a?b} Q' \quad b\notin fn(Q)}{P\|Q \xrightarrow{\tau} \nu b(P'\|Q')}\ (\textsc{Close})$$

$$\frac{P \xrightarrow{\alpha} P' \quad bn(\alpha)\cap fn(Q)=\varnothing}{P\|Q \xrightarrow{\alpha} P'\|Q}\ (\textsc{Par}) \qquad \frac{P \xrightarrow{\alpha} P' \quad b\notin n(\alpha)}{\nu bP \xrightarrow{\alpha} \nu bP'}\ (\textsc{Res})$$

The so-called early LTS, presented above for the finite, sum-free fragment without match or mismatch, is widely known and can be presented and used by humans in a fairly simple way provided that a number of conventions is followed – we shall discuss these in more detail below. In order to save space we omitted the symmetric versions of (Comm), (Close) and (Par); one normally works with an abstract syntax in which $\|$ is associative and commutative – such an equivalence relation on syntax is usually referred to as structural congruence.

It is worth mentioning that rules (Open) and (Close) perform two roles: scope extrusion and generation of new name observable (the bound output label $a!(b)$). The side-conditions on the derivation rules deserve some further consideration. The requirements on (Res) and (Open) are fairly obvious and inherited from CCS – they ensure that ν behaves like CCS restriction in that one cannot communicate on a restricted channel from outside its scope. The conditions required of (Close) and (Par) merit more attention – in particular they make sure that the (Open)/(Close) mechanism for scope extrusion functions correctly.

Indeed, consider the process $P = (\nu b\, a!b) \parallel b?x$ where the scope of the restriction encompasses only the left component. If the side-condition on (LPar) was lifted one would be able to derive the label $P \xrightarrow{a!(b)} b?x$. But, using (Close), one would then be able to derive $P \parallel a?y \xrightarrow{\tau} \nu b\, b?x$ with the result that the freely occurring b in right component of P would have become incorrectly bound after the interaction. The side condition on (Close) exists for similar reasons.

Even with the side-conditions, the definition of bisimulation has to be altered. In particular, in the bisimulation game played on (P, Q), if P challenges with $P \xrightarrow{a!(b)} P'$ then Q must respond if and only if $b \notin fn(Q)$. Technically this is due to the side condition in the (Par) rule: consider the processes

$$\nu b\, a!b \quad \text{and} \quad (\nu b\, a!b) \parallel \nu c c!b$$

which are clearly equivalent; the second component of the second process has no observable behaviour – but clearly the first process can perform a $a!(b)$ action while the second cannot. Intuitively, this phenomenon demonstrates ν'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 [11, Convention 1.4.10] argue that all such problems can be solved 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 been tampered with. It has also been noticed by a number of authors that conventions which are fine for humans are non-trivial to implement when using proof-assistants or theorem-provers. There have been several works [1, 6, 4] 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 [5] for an overview and a comparison of the approaches.

As outlined above, the LTS can be considered as a tool for reasoning about a contextually defined equivalence on the reduction semantics; following this line of reasoning raises a natural question as to whether the LTS can be derived in some systematic way from the reduction semantics; research in this direction was commenced by Leifer and Milner [8] and continued by Klin, Sassone and the author [7]. 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 $c[t] \to t'$ 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 framework 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 [7] – 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

$$\langle a!bP, -_1 \rangle \xrightarrow{-_1 \| a?x -_2} P \| -_1$$

where the context forms an (unsubstantiated) part of the term and the labels are able to modify the context. Another feature is that the information in the labels is restricted by what can be observed in one reduction step – this rules out Pi labels which also contain the name being transmitted.

We can use some of the intuitions gained from the work described in the paragraph above to develop a new LTS for the Pi-calculus. To do this 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 interaction. The meta-syntax is a simply-typed lambda calculus, with β-reduction forming a part of the structural congruence.

We start with an untyped syntax on which we give type rules, starting with two base types; a name type $\mathscr{N}$ and a process type $\mathscr{P}$. Types in this setting are included only to make the meta-syntax formal, they certainly have less structure than usual Pi-calculus type systems based on channel types. Differently from the classical presentation, we treat ν as a standard binder which binds variables of name

type. We assume a countable supply of variables of each type. In all these choices, our presentation more similar to Engberg and Nielsen's ECCS [2,3], the "precursor" to the Pi-calculus. Actually, all the basic ingredients of the theory (scope extrusion, fresh name generation) were already present in the aforementioned work; the main novelty of the Pi-calculus was the collapse of the syntactic classes of variables and names. It can be argued that the resulting simplicity is deceptive.

We use the syntactic convention of a, b for name constants, k, l for terms of name type, x, y for variables (including name variables), P, Q for terms of process type, X, Y for variables of function type. Type contexts are finite maps from variables to types. We shall consider only typed terms.

Types

$$\sigma \ ::= \ \mathscr{N} \mid \mathscr{P} \mid \sigma \to \sigma$$

Syntax

$$M \ ::= \ x \mid a \mid 0 \mid M \parallel M \mid M!MM \mid M?xM \mid \nu xM \mid \lambda x:\sigma.M \mid M(M)$$

Type rules

$$\frac{}{\Gamma\vdash a{:}\mathscr{N}}\ (:\textsc{Name}) \qquad \frac{}{\Gamma\vdash 0{:}\mathscr{P}}\ (:\textsc{Zero})$$

$$\frac{\Gamma\vdash M{:}\mathscr{P} \quad \Gamma\vdash M'{:}\mathscr{P}}{\Gamma\vdash M\|M'{:}\mathscr{P}}\ (:\textsc{Par}) \qquad \frac{\Gamma\vdash M{:}\mathscr{P} \quad \Gamma\vdash k{:}\mathscr{N} \quad \Gamma\vdash l{:}\mathscr{N}}{\Gamma\vdash k!l.M{:}\mathscr{P}}\ (:\textsc{OutPref})$$

$$\frac{\Gamma,x{:}\mathscr{N}\vdash M{:}\mathscr{P}}{\Gamma\vdash \nu xM{:}\mathscr{P}}\ (:\textsc{Nu}) \qquad \frac{\Gamma,x{:}\mathscr{N}\vdash M{:}\mathscr{P} \quad \Gamma\vdash k{:}\mathscr{N}}{\Gamma\vdash k?x.M{:}\mathscr{P}}\ (:\textsc{InPref})$$

$$\frac{\Gamma(x){=}\sigma}{\Gamma\vdash x{:}\sigma}\ (:\textsc{Var}) \qquad \frac{\Gamma,x{:}\sigma\vdash M{:}\sigma'}{\Gamma\vdash \lambda x{:}\sigma.M{:}\sigma\to\sigma'}\ (:\lambda) \qquad \frac{\Gamma\vdash M_1{:}\sigma\to\sigma' \ \Gamma\vdash M_2{:}\sigma}{\Gamma\vdash M_1(M_2){:}\sigma'}\ (:\textsc{App})$$

We give a brief outline of the part of the LTS concerned with communication and scope extrusion below. The LTS is defined structurally. Because we operate in a typed setting, the states of the LTS are pairs of typed context and term. We denote such pairs syntactically as $\langle \Gamma \mid V \rangle$ where Γ is a type context and V is a term. The interplay between the type context and the term is explained by a type preservation result, described following an intuitive account of the transitions.

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 $\mathscr{I} = \mathscr{N} \to \mathscr{P}$ (cf (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 $\mathscr{N}$ (cf (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 (Tau)). There is a technical similarity to the presentation of the late semantics using abstractions and concretions [9]; see the (Comm) rule below. Notice, however, that we do not say anything about the actual nature of the name transmitted in the labels.

LTS - Communication fragment

$$\frac{}{\langle \Gamma \mid k?xP\rangle \xrightarrow{k?} \langle \Gamma \mid \lambda x{:}\mathscr{N}.P\rangle}\ (\textsc{In}) \qquad \frac{}{\langle \Gamma \mid k!lP\rangle \xrightarrow{k!} \langle \Gamma \mid \lambda X{:}\mathscr{I}.P \| X(l)\rangle}\ (\textsc{Out})$$

$$\frac{\langle \Gamma \mid P\rangle \xrightarrow{k?} \langle \Gamma \mid U\rangle}{\langle \Gamma \mid P\|Q\rangle \xrightarrow{k?} \langle \Gamma \mid \lambda x{:}\mathscr{N}.U(x)\|Q\rangle}\ (\|\textsc{In}) \qquad \frac{\langle \Gamma \mid P\rangle \xrightarrow{k!} \langle \Gamma \mid T\rangle}{\langle \Gamma \mid P\|Q\rangle \xrightarrow{k!} \langle \Gamma \mid \lambda X{:}\mathscr{I}.T(X)\|Q\rangle}\ (\|\textsc{Out})$$

$$\frac{\langle \Gamma,y{:}\mathscr{N} \mid P\rangle \xrightarrow{k?} \langle \Gamma,y{:}\mathscr{N} \mid U\rangle \quad (k\neq y)}{\langle \Gamma \mid \nu yP\rangle \xrightarrow{k?} \langle \Gamma \mid \lambda x{:}\mathscr{N}.\nu yU(x)\rangle}\ (\nu\textsc{In}) \qquad \frac{\langle \Gamma,y{:}\mathscr{N} \mid P\rangle \xrightarrow{k!} \langle \Gamma,y{:}\mathscr{N} \mid T\rangle \quad (k\neq y)}{\langle \Gamma \mid \nu yP\rangle \xrightarrow{k!} \langle \Gamma \mid \lambda X{:}\mathscr{I}.\nu yT(X)\rangle}\ (\nu\textsc{Out})$$

$$\frac{\langle \Gamma \mid P\rangle \xrightarrow{k!} \langle \Gamma \mid T\rangle \quad \langle \Gamma \mid Q\rangle \xrightarrow{k?} \langle \Gamma \mid U\rangle}{\langle \Gamma \mid P\|Q\rangle \xrightarrow{\tau} \langle \Gamma \mid T(U)\rangle}\ (\textsc{Tau})$$

$$\frac{\langle \Gamma \mid P\rangle \xrightarrow{\tau} \langle \Gamma \mid P'\rangle}{\langle \Gamma \mid P\|Q\rangle \xrightarrow{\tau} \langle \Gamma \mid P'\|Q\rangle}\ (\|\textsc{Tau}) \qquad \frac{\langle \Gamma,y{:}\mathscr{N} \mid P\rangle \xrightarrow{\tau} \langle \Gamma,y{:}\mathscr{N} \mid P'\rangle}{\langle \Gamma \mid \nu yP\rangle \xrightarrow{\tau} \langle \Gamma \mid \nu yP'\rangle}\ (\nu\textsc{Tau})$$

In the rules (‖In) and (νIn), the variable x in the right hand side of the result is chosen fresh for U. Similarly, in (‖Out) and (νOut) the variable X is chosen fresh for T. The LTS has a regular structure – there is one axiom for each kind of label - (In), (Out) and (Tau) for input, output and τ respectively. Further, there is precisely one rule for each syntactic constructor and each kind of label.

We 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:

$$\frac{\dfrac{}{\langle x{:}\mathscr{N} \mid a!x.P\rangle \xrightarrow{a!} \langle x{:}\mathscr{N} \mid \lambda X{:}\mathscr{I}.P\|X(x)\rangle}\ (\textsc{Out})}{\langle \varnothing \mid \nu xa!x.P\rangle \xrightarrow{a!} \langle \varnothing \mid \lambda X{:}\mathscr{I}.\nu x(\lambda X'{:}\mathscr{I}.P\|X'(x))X\rangle}\ (\nu\textsc{Out})$$

but $\lambda X : \mathscr{I}.\nu x(\lambda X' : \mathscr{I}.P \parallel X'(x))X \equiv \lambda X : \mathscr{I}.\nu x(P \parallel X(x))$, and so, using (Tau) we obtain the correct extrusion of scope:

$$\frac{\dfrac{}{\langle \varnothing \mid a?y.Q\rangle \xrightarrow{a?} \langle \varnothing \mid \lambda y.Q\rangle}\ (\textsc{In}) \qquad \langle \varnothing \mid \nu xa!x.P\rangle \xrightarrow{a!} \langle \varnothing \mid \lambda X{:}\mathscr{I}.\nu x(P\|X(x))\rangle}{\langle \varnothing \mid a?y.Q\|\nu x.a!x.P\rangle \xrightarrow{\tau} \langle \varnothing \mid \nu x.(P\|(\lambda y.Q)(x))\rangle}\ (\textsc{Tau})$$

since $\nu x.(P \parallel (\lambda y.Q)(x)) \equiv \nu x(P \parallel Q[x/y])$.

One can prove a type preservation result about the LTS – in particular demonstrating that the result of applying the (Tau) rule is typeable. Indeed, suppose that $\Gamma \vdash P : \mathscr{P}$. Then the following hold:

- if $\langle \Gamma \mid P \rangle \xrightarrow{k?} \langle \Gamma \mid U \rangle$ then $\Gamma \vdash U : \mathscr{N} \to \mathscr{P}$;
- if $\langle \Gamma \mid P \rangle \xrightarrow{k!} \langle \Gamma \mid T \rangle$ then $\Gamma \vdash T : \mathscr{I} \to \mathscr{P}$;
- if $\langle \Gamma \mid P \rangle \xrightarrow{\tau} \langle \Gamma \mid P' \rangle$ then $\Gamma \vdash P' : \mathscr{P}$.

The LTS presented above is only a fragment and bisimilarity does not give a satisfactory equivalence; it is presented here only to give a basic idea of the techniques involved. In particular, there is no observation of the identity of the name communicated (input, output, or fresh output). The full LTS contains extra labels and derivation rules which take care of such observables by suitably evaluating λ-abstracted terms, yielding the standard early equivalence. The two functions (scope extrusion and fresh name generation) of the ν operator are thus dealt with separately. Intuitively, transitions which arise out of process terms, as presented here, represent the part of the interaction which is controlled by the process; transitions out of lambda-abstracted terms, omitted in this presentation, represent the part controlled by the context. The full LTS does not require complicated side-conditions, variable conventions and the resulting equivalence is obtained using ordinary bisimulation indexed by type contexts.

The reader will notice that in fact the LTS resembles closely that for CCS – in particular one could give a complete LTS for that calculus in a similar style (without passing the name to the context in the result of (Out) and without awaiting a name in the result of (In)). This demonstrates one feature and a design principle associated with the LTS – modularity. The idea 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: the Pi-calculus extends CCS with extra features and thus it shouldn't be surprising that the synchronous communication feature of Pi is handled with a CCS-like subsystem. Contrast this situation with the standard LTS in which communication is more complicated - there are two different kinds of output labels (free and bound) and two different axioms for τ. While presentations such as [1] reduce this difference, it may prove that the technique advocated in this paper generalises well to other calculi – eg higher order Pi-calculus or the ambient calculus.

References

[1] G. L. Cattani and P. Sewell. Models for name-passing processes: interleaving and causal. *Information and Computation*, 190:136–178, 2004.

[2] U. Engberg and M. Nielsen. A calculus of communicating systems with label passing. Technical Report DAIMI PB-208, University of Aarhus, May 1986.

[3] U. Engberg and M. Nielsen. A calculus of communicating systems with label passing - ten years after. In *Proof, Language, and Interaction; Essays in Honour of Robin Milner*, pages 599–622. MIT Press, 2000.

[4] G. U. Ferrari, U. Montanari, and M. Pistore. Minimising transition systems for name passing calculi: a co-algebraic formulation. In *FoSSaCS '02*, pages 129–158, 2002.

[5] M. P. Fiore and S. Staton. Comparing operational models of name-passing process calculi. *Information and Computation*, 204(4):435–678, 2006.

[6] M. P. Fiore and D. Turi. Semantics of name and value passing. In *LiCS '01*, pages 93–104, 2001.

[7] B. Klin, V. Sassone, and P. Sobociński. Labels from reductions: towards a general theory. In *Algebra and Coalgebra in Computer Science, Calco '05*, volume 3629 of *LNCS*, pages 30–50. Springer, 2005.

[8] J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. In *International Conference on Concurrency Theory Concur '00*, volume 1877 of *LNCS*, pages 243–258. Springer, 2000.

[9] R. Milner. *Communicating and Mobile Systems: the Pi-calculus*. Cambridge University Press, 1999.

[10] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, ii. *Information and Computation*, 100(1):41–77, 1992.

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

[12] L. Wischik. Old names for Nu. In *Dagstuhl Seminar 04241*, 2004.