BRICS

Basic Research in Computer Science

BRICS LS-98-4 P. Quaglia: The π-Calculus: Notes on Labelled Semantics

The π-Calculus: Notes on Labelled Semantics

Paola Quaglia

BRICS Lecture Series LS-98-4

ISSN 1395-2048 December 1998

Copyright © 1998, BRICS, Department of Computer Science
University of Aarhus. All rights reserved.

Reproduction of all or part of this work
is permitted for educational or research use
on condition that this copyright notice is
included in any copy.

See back inner page for a list of recent BRICS Lecture Series publications. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark

Telephone:	+45 8942 3360
Telefax:	+45 8942 3255
Internet:	BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `LS/98/4/`

The π-Calculus: Notes on Labelled Semantics

Paola Quaglia

Paola Quaglia[1]

quaglia@brics.dk

BRICS[2]
Department of Computer Science
University of Aarhus
Ny Munkegade
DK-8000 Aarhus C, Denmark

December 1998

[1]Funded by the EU, under the Marie Curie TMR Programme.

[2]**B**asic **R**esearch **I**n **C**omputer **S**cience,
Centre of the Danish National Research Foundation.

Preface

The π-calculus [MPW92] is a name-passing calculus that allows the description of distributed systems with a dynamically changing interconnection topology. Name communication, together with the possibility of declaring and exporting local names, gives the calculus a great expressive power. For instance, it was shown that process-passing calculi, which express mobility at higher order, can be encoded naturally in π-calculus [San93a] .

Since its inception, the π-calculus has proliferated into a family of calculi differing slightly from one another either in the communication paradigm (polyadic vs monadic, asynchronous vs synchronous) or in the bisimulation semantics (labelled vs unlabelled, late vs early vs open vs barbed vs ...).

These short notes present a collection of the labelled strong semantics[3] of the (synchronous monadic) π-calculus. The notes could not possibly replace any of the standard references listed in the Bibliography. They are an attempt to group together, using a uniform notation and the terminology that got assessed over the last years, a few definitions and concepts otherwise scattered throughout the π-calculus literature.

I would like to thank James J. Leifer for his careful reading of the manuscript, and the helpful suggestions he provided.

[3]The definition of weak late semantics requires some ingenuity. But for this case, the weak corresponding of each of the semantics we present can be easily defined by mimicking the standard CCS-like pattern.

Contents

Preface **v**

1 Preliminaries **1**

2 Syntax **2**

3 Labelled semantics **3**

3.1 Late semantics 6

3.2 Early semantics 8

3.3 Open semantics 10

1 Preliminaries

The most primitive notion in the π-calculus is that of naming: data values communicated along channels (names) are themselves channels (names). As naming is distributed and involved in communication, the π-calculus builds on traditional process algebras (CCS [Mil89], MEIJE [AB84], ACP [BK84, BK85], CSP [BHR84, Hoa85]) in a precise sense. It allows one to specify the behaviour of distributed systems in which the interaction among independent and cooperating components may cause a dynamic change of the single partners acquaintances.

As an example, the π-calculus permits us to describe in an elegant and natural way common behaviours of, *e.g.*, operating systems. Think of the cooperation in the sharing of a common resource such as a printer. Whenever the printer manager is specified by a π-calculus agent, misbehavings in the communication protocol are prevented by the fact that the printer-process can settle down, by-need, private links with each client-process. In CCS an analogous scenario could at best be described by resorting to non-determinism and setting up in advance as many distinct port names as the number of potential requests to the printer.

The name-passing interaction paradigm is also responsible for the semantic enrichment of the π-calculus over synchronization process calculi. The parameter y of the input action $x(y)$ is a placeholder for something to be received. Then, depending on the operational intuition about input actions, the π-calculus semantics naturally proliferate (at least) in two distinct families: *early* and *late* [MPW93]. The early paradigm considers the act of committing on the input channel and the choice of the actual parameter as one single atomic event [MPW93]. The late view interprets the derivative of an inputting process as a function of the received name [MPW92]. Besides the early and the late semantics, other paradigms become natural as well, building on the intuition that name instantiation can be delayed more and more (see, for instance, *open* bisimulation [San96]).

The expressive power of the π-calculus is largely confirmed by the fact that it allows us to encode, usually in a fully abstract way:

- values and data structures [MPW92, Mil92b];
- the λ-calculus [Mil92a, San94a, San97];
- process-passing interaction paradigms [San93a, San93b, Ama93];

- concurrent object-oriented languages [Jon93, Wal95, LW95a];
- locality and causality dependencies [San94b, BS94] which are typical of true concurrent semantics (see, for example [DDNM90, BCHK93, NC95]).

Also, the polyadic version of the calculus [Mil92b] was shown to be suitable for reasoning about concurrent typing disciplines (see, *e.g.*, [Gay93, PS93, VH93, LW95b]). Eventually, the π-calculus is being used as theoretical foundation in the design of experimental concurrent programming languages [PT95, FG96].

2 Syntax

The π-calculus allows the description of process behaviours in terms of the actions they can perform, or equivalently, of the interactions they may be involved in. Processes are then given an operational interpretation as points of a labelled transition system. More abstract semantics are obtained by introducing equivalences which identify those processes that behave the same w.r.t. fixed notions of observation, *i.e.* of interaction with a possible external observer. These last semantics are defined either operationally as bisimulation games, or axiomatically by a set of appropriate algebraic laws.

In the following we will recall the π-calculus syntax. Operational and bisimulation semantics will be addressed in later sections.

Let $\mathcal{N}$ be a denumerably infinite set of names (ranged over by x, y, z, $\ldots$). The syntax of π-calculus processes (ranged over by P, Q, $\ldots$) is defined by the following grammar:

P	$::=$	$\mathtt{nil}$	inaction
	$\mid$	$\alpha.P$	prefix
	$\mid$	$[x = y]P$	match
	$\mid$	$P + P$	non-deterministic choice
	$\mid$	$P \mid P$	parallel composition
	$\mid$	$\nu y\, P$	restriction
	$\mid$	$!P$	replication or bang

A CCS-like precedence relation among operators is assumed, *e.g.* unary operators bind more than binary ones. Prefixes α are given by:

$$\begin{array}{rclll} \alpha & ::= & & \tau & \text{silent action} \\ & & \mid & x(y) & \text{input action} \\ & & \mid & \overline{x}y & \text{output action} \end{array}$$

The prefix $x(y)$ means 'input some name along the link named x and call it y'. It is called *bound input*, recording that brackets act as formal binder. Namely the prefix $x(y)$ in $x(y).P$ binds the free occurrences of y in P in the same way as in the λ-calculus $\lambda y.t$ binds the name y in t.

The prefix $\overline{x}y$ means 'output the name y along the link named x'. The prefix $\overline{x}y$ is called *free output*, as opposed to the *bound output* $\overline{x}(y)$. This last action is not available at the syntactic level and denotes the ability of communicating the private name y. Either in $x(y)$ or in $\overline{x}y$ or in $\overline{x}(y)$, the name x is said the *subject*, while y is called the *object* or parameter. Also, for the bound input $x(y)$, the name y is sometimes referred to as a *placeholder*.

Besides the prefix $x(y)$, another kind of formal binder is the restriction operator νy in $\nu y\, P$. If a name is not bound, it is called free. The set of the names which occur free in an action α (agent P) is written $\mathrm{fn}(\alpha)$ ($\mathrm{fn}(P)$). Dually, the set of bound names is written $\mathrm{bn}(\alpha)$ ($\mathrm{bn}(P)$). Sometimes $\mathrm{fn}(P, Q)$ is used as a shorthand for $\mathrm{fn}(P) \cup \mathrm{fn}(Q)$. The set of the names of an action α (agent P) is defined to be the union of its free and bound names and it is written $\mathrm{n}(\alpha)$ ($\mathrm{n}(P)$). Obviously, the unobservable action τ is such that $\mathrm{n}(\tau) = \emptyset$.

The relation of α-convertibility, denoted by $\equiv_\alpha$, is defined in the standard way. Syntactic identity of P and Q is written $P = Q$.

Sometimes we freely omit from process syntax all the unnecessary details (*e.g.* the trailing '`.nil`').

3 Labelled semantics

The π-calculus semantics often resorts to name substitutions (ranged over by σ, σ', ...). Name substitutions are functions from $\mathcal{N}$ to $\mathcal{N}$ defined almost everywhere as identities. Sometimes, when the substitution σ differs from the identity for the names in $\{ x_1, \ldots , x_n \}$, σ is simply written $\{ {}^{x_1\sigma}/_{x_1}, \ldots , {}^{x_n\sigma}/_{x_n} \}$. The term $P\sigma$ denotes the process obtained from P by

$$\text{Tau} \quad \tau.P \xrightarrow{\tau} P$$

$$\text{Inp} \quad x(y).P \xrightarrow{x(w)} P\,\{w/y\} \quad w \notin \mathrm{fn}(\nu y\, P) \qquad\qquad \text{Out} \quad \overline{x}y.P \xrightarrow{\overline{x}y} P$$

$$\text{Match} \quad \frac{P \xrightarrow{\alpha} P'}{[x = x]P \xrightarrow{\alpha} P'}$$

$$\text{Sum} \quad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

$$\text{Par} \quad \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \quad \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset$$

$$\text{Com} \quad \frac{P \xrightarrow{\overline{x}y} P' \quad Q \xrightarrow{x(z)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'\,\{y/z\}} \qquad\qquad \text{Close} \quad \frac{P \xrightarrow{\overline{x}(w)} P' \quad Q \xrightarrow{x(w)} Q'}{P \mid Q \xrightarrow{\tau} \nu w\,(P' \mid Q')}$$

$$\text{Open} \quad \frac{P \xrightarrow{\overline{x}y} P'}{\nu y\, P \xrightarrow{\overline{x}(w)} P'\,\{w/y\}} \quad y \neq x,\ w \notin \mathrm{fn}(\nu y\, P') \qquad \text{Res} \quad \frac{P \xrightarrow{\alpha} P'}{\nu y\, P \xrightarrow{\alpha} \nu y\, P'} \quad y \notin \mathrm{n}(\alpha)$$

$$\text{Bang} \quad \frac{P \mid !P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P'}$$

Table 1: The π-calculus operational semantics

simultaneously substituting, for each x, any free occurrence of x in P by $x\sigma$, with change of bound names to avoid name clashes. So, for instance, in order to prevent the capture of the name y, the application of the substitution $\{y/x\}$ to the process $\nu y\, \overline{x}y.\,\texttt{nil}$ results in $\nu w\, \overline{y}w.\,\texttt{nil}$ with $w \neq y$.

The π-calculus operational semantics is defined inductively, in the style of [Plo81], by the rules shown in Tab. 1 together with additional symmetric rules for the binary operators of non-deterministic choice and parallel composition. In Tab. 1 the transition label α stands for either the perfect event τ, or a bound input, or an output action, either free or bound.

The rules *Tau*, *Out*, *Sum*, *Par*, *Com*, and *Res* are similar to the corresponding rules of CCS.

The side condition of the rule *Par* merits some discussion. Briefly, it is intended to avoid the name captures which could arise from the joint application of the rules for asynchronous parallel composition and communication. For instance, by violating the side condition of *Par*, the following *incorrect* behaviour could be inferred:

$$\dfrac{\dfrac{Q = x(y).y \xrightarrow{x(y)} y}{Q \mid R = x(y).y \mid y \xrightarrow{x(y)} y \mid y} \qquad S = \overline{x}z.z \xrightarrow{\overline{x}z} z}{(Q \mid R) \mid S \xrightarrow{\tau} (\ (y \mid y)\,\{z/y\} \mid z\ ) = (z \mid z) \mid z}$$

The expected behaviour of $(Q \mid R) \mid S$ is instead $(Q \mid R) \mid S \xrightarrow{\tau} (z \mid y) \mid z$. This reflects the intuition that the transmission of z must not cause the substitution of the name y in R. In fact, as in the λ-calculus, the rightmost occurrence of y in $x(y).y$ is just a nameless pointer to the binder (y) of the prefix $x(y)$. So, it is in any respect distinct from the name y occurring in R.

The fact that the process $Q \mid R = x(y).y \mid y$ of the above example can only be allowed to perform input moves $x(w)$, with $w \neq y$, justifies the definition of the axiom *Inp*. A CCS-like axiom of the shape $x(y).P \xrightarrow{x(y)} P$ would deadlock Q in the parallel context $(\_ \mid R)$. Analogously, choosing any other suitable name z and defining the axiom as $x(y).P \xrightarrow{x(z)} P\,\{z/y\}$ would stop Q in the parallel context $(\_ \mid z)$. That is why *Inp* can be applied infinitely many times to the same process. More generally, that is why the π-calculus transition system is such that whenever a process P may perform a bound action α (*cf. Inp* and *Open*), it can also perform infinitely many other actions differing from α only for the identity of the bound name.

The matching operator is used to test names for equality. Process $[x = y]P$ behaves like P if x and y are the same name, it behaves like the inactive process otherwise.

The restriction operator inherits very little from CCS. Analogously to local variable declarations of block-structured languages, the restriction of y on top of P declares a new unique name for use in P. In view of its privacy, the name y cannot be used as communication subject. Anyway y is not necessarily destined to remain local to P. It can be exported outside by means of an output action. The rule *Open*, while removing the restriction operator, transforms the free output action $\overline{x}y$ into the bound output action $\overline{x}(w)$, where w is a new name. The information that w refreshes a name that was private is directly represented in the action, where w appears enclosed between brackets.

The joint use of the rules *Open* and *Close* causes a so-called *extrusion*. A bound output combines with an input action, and once the bound name has been received, a restriction is put on top of the synchronizing processes, meaning that the name is still private although its scope has grown.

The replication (or bang) operator '!' is used to express infinite behaviours. The rule for the replication operator suggests that process $!P$ can be thought of as the parallel composition of as many instances of P as desired. In fact, whatever is the action α that $P\ |!P$ can perform, the replication $!P$ can execute α as well.

As in the case of CCS-like languages, processes are quotiented by strong or weak equivalence relations defined as bisimulation games. In the following we recall the strong versions of *late* [MPW92], and *early* [MPW93], and *open* [San96] semantics.

3.1 Late semantics

The strong bisimulation game between the CCS-like processes P and Q requires any move of P to be matched by a move of Q, and vice-versa, with the derivatives P' and Q' able to play a similar game. This notion of behavioural equivalence does not fit with naming, the critical case being that of input actions.

The parameter of any input action is a placeholder for something to be received, and can become substituted by an arbitrary name. Then, mimicking an input action requires some care of the degree of non-determinism in the actual instantiation of the placeholder.

The *late* semantics [MPW92] gives input actions a functional operational intuition. When inputting, a process becomes a function of the actual transmitted name. So, the input clause of the definition of *strong late bisimulation* claims that the derivatives of the inputting processes continue to simulate for all instantiations of the formal parameter.

Definition 1 A binary symmetric relation $\mathcal{S}$ is a *late bisimulation* if $P\ \mathcal{S}\ Q$ implies that

- if $P \xrightarrow{\alpha} P'$ with $\alpha \neq x(y)$ and $\mathrm{bn}(\alpha) \notin \mathrm{fn}(P, Q)$, then for some Q', $Q \xrightarrow{\alpha} Q'$ and $P'\ \mathcal{S}\ Q'$

- if $P \xrightarrow{x(y)} P'$ with $y \notin \mathrm{fn}(P, Q)$, then for some Q', $Q \xrightarrow{x(y)} Q'$ and, for all w, $P'\{w/y\}\ \mathcal{S}\ Q'\{w/y\}$

P is *late bisimilar* to Q, written $P \dot{\sim}_L Q$, if $P\ \mathcal{S}\ Q$ for some late bisimulation $\mathcal{S}$. □

Example 2 Let $P = x(y).\tau.\texttt{nil} + x(y).\texttt{nil}$ and $Q = P + x(y).[y = z]\tau.\texttt{nil}$. Then $P \not\dot{\sim}_L Q$, the reason being that P cannot properly match the transition $Q \xrightarrow{x(y)} [y = z]\tau.\texttt{nil}$. A straightforward way to see this is rewriting the behaviours of the derivatives of Q and of P as functions of the name y. The function $\lambda y.\texttt{if}\ y = z\ \texttt{then}\ \tau.\texttt{nil}\ \texttt{else}\ \texttt{nil}$ (which interprets process $[y = z]\tau.\texttt{nil}$) is distinct from both the constant function $\lambda y.\tau.\texttt{nil}$ (given rise to by the move $P \xrightarrow{x(y)} \tau.\texttt{nil}$), and the constant function $\lambda y.\texttt{nil}$ (originated by $P \xrightarrow{x(y)} \texttt{nil}$). □

Equational theories for strong late bisimilarity, which are shown to be complete over finite processes, are presented both in the original paper on the π-calculus [MPW92], and in [PS95]. The two axiomatizations essentially differ for the use in [PS95] of a mismatching construct which allows to test for name inequality. The mismatching $[x \neq y]P$ behaves just the opposite of $[x = y]P$, *i.e.* as the conditional '$\texttt{if}\ x \neq y\ \texttt{then}\ P\ \texttt{else}\ \texttt{nil}$'. Mismatching does not preserve the following monotonicity property of process action capability w.r.t. name substitutions:

$$\text{if } P \xrightarrow{\alpha} P' \text{ then } P\sigma \xrightarrow{\beta} P'' \text{ with } \beta.P'' \equiv_\alpha (\alpha.P')\sigma$$

For instance, $[x \neq y]\tau \xrightarrow{\tau}$ but not $([x \neq y]\tau)\{x/y\} \xrightarrow{\tau}$. The above property is crucial to the π-calculus mathematical theory. So, the mismatching operator cannot be added with the π-calculus syntax in a completely harmless way. Nevertheless, the use of mismatching gives the axiom system of [PS95] great generality and flexibility, *e.g.* *early* semantics can be characterized by adding the late system with one single law.

Late bisimulation is an equivalence relation, but is not preserved by substitution of names, and then by input prefix. For this reason it is denoted by a dotted relational symbol, and sometimes referred to as a *ground* relation.

Example 3 The agent $P = [x = y]\overline{x}x.\texttt{nil}$, having no outgoing transition, is late bisimilar to the inactive process $Q = \texttt{nil}$. This is not the case after substituting x for y, *e.g.* after putting $x(y)$ on top of P and of Q. □

Late full congruence (also called *non-ground* late bisimilarity) is obtained by closing the ground equivalence over all name substitutions.

Definition 4 P and Q are *late congruent*, written $P \sim_L Q$, if $P\sigma \dot{\sim}_L Q\sigma$ for all substitutions σ. $\square$

A complete axiom system for late congruence of finite processes appears in [PS95]. Just like the equational theory for ground bisimilarity which was investigated in the same paper, the system makes use of the mismatching constructor.

3.2 Early semantics

The fact that names can be transmitted in interactions makes the π-calculus semantics naturally proliferate in two distinct families – *late* and *early* – depending on the operational intuition about input actions. We already commented on the late paradigm. It interprets the derivative of the inputting process as a function of the received name, and then insists for an input move to be matched by a single input step. The more liberal *early* view allows an input transition to be matched by distinct moves, depending on the actual transmitted parameter. Then, the input clause of *early bisimulation* [MPW93] only requires that for each received name there is a matching transition.

Definition 5 A binary symmetric relation $\mathcal{S}$ is an *early bisimulation* if $P \; \mathcal{S} \; Q$ implies that

- if $P \xrightarrow{\alpha} P'$ with $\alpha \neq x(y)$ and $\mathrm{bn}(\alpha) \notin \mathrm{fn}(P, Q)$, then for some Q', $Q \xrightarrow{\alpha} Q'$ and $P' \; \mathcal{S} \; Q'$
- if $P \xrightarrow{x(y)} P'$ with $y \notin \mathrm{fn}(P, Q)$, then for all w there exists Q' such that $Q \xrightarrow{x(y)} Q'$ and $P' \{w/y\} \; \mathcal{S} \; Q' \{w/y\}$

P is *early bisimilar* to Q, written $P \dot{\sim}_E Q$, if $P \; \mathcal{S} \; Q$ for some early bisimulation $\mathcal{S}$. $\square$

The following example shows that early semantics is strictly coarser than late.

$$x(y).P \xrightarrow{xw} P\{w/y\} \qquad\qquad \frac{P \xrightarrow{\overline{x}y} P' \qquad Q \xrightarrow{xy} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

Table 2: Rules for the early π-calculus transition system

Example 6 Let $P = x(y).\tau.\,\mathtt{nil} + x(y).\,\mathtt{nil}$ and $Q = P + x(y).[y = z]\tau.\,\mathtt{nil}$. In Ex. 2 it was shown that $P \not\dot{\sim}_L Q$. This is because P cannot match the transition $Q \xrightarrow{x(y)} [y = z]\tau.\,\mathtt{nil}$ in a late bisimulation game. When the game is assumed to follow an early strategy, depending on whether $w \neq z$ or not, process P may react to $Q \xrightarrow{x(y)} [y = z]\tau.\,\mathtt{nil}$ either by $P \xrightarrow{x(y)} \mathtt{nil}$ or by $P \xrightarrow{x(y)} \tau.\,\mathtt{nil}$, respectively. Then $P \dot{\sim}_E Q$. □

As for late semantics, early bisimulation is a ground equivalence relation, namely it is not preserved by input prefix. This can be shown by using the same processes considered in Ex. 3. Again, the corresponding congruence is defined by requiring bisimilarity over all substitutions.

Definition 7 P and Q are *early congruent*, written $P \sim_E Q$, if $P\sigma \dot{\sim}_E Q\sigma$ for all substitutions σ. □

Axiomatizations of early bisimilarity and congruence, which are proved to be complete over finite processes, are defined in [PS95].

As a final remark about early semantics, we want to point out that the early paradigm considers the act of committing on the input channel and the choice of the actual parameter as one single atomic event. Indeed, in [MPW93] the early semantics was given an alternative characterization in terms of strong bisimulation over the specialized *early transition system.* The early π-calculus transition system is obtained by replacing the *Com* rule in Tab. 1 with the two rules in Tab. 2. Contrary to the original π-calculus operational semantics (referred to as *late transition system*), the early transition system makes explicit use of *free input* actions. The free input xy informally means 'input the name y along the link named x'. In this respect free inputs naturally correspond to the kind of input actions obtained when translating CCS with value-passing into pure CCS with infinite summations.

Remarkably, coincidence results of the early semantics with the ordinary CCS-like bisimulation semantics can be stated. In order to show this, we first recall the usual definition of strong bisimulation over a CCS-like transition relation $\longrightarrow\!\!\triangleright$.

Definition 8 Assume $\longrightarrow\!\!\triangleright$ to be the operational transition relation between processes of a given calculus $\mathcal{P}$. A binary symmetric relation $\mathcal{S}$ over processes of $\mathcal{P}$ is a *strong bisimulation* if $P\ \mathcal{S}\ Q$ implies that

if $P \xrightarrow{\alpha}\!\!\triangleright P'$ then for some Q', $Q \xrightarrow{\alpha}\!\!\triangleright Q'$ and $P'\ \mathcal{S}\ Q'$

P is *strong bisimilar* to Q, written $P \sim Q$, if $P\ \mathcal{S}\ Q$ for some strong bisimulation $\mathcal{S}$. □

Lemma 9 [MPW93] *Assume that the transition relation and the actions considered in Definition 8 are, resp., the early transition relation, and actions α such that* $\mathrm{bn}(\alpha) \cap \mathrm{fn}(P, Q) = \emptyset$. *Then* $\dot{\sim}_E = \sim$.

PROOF: The coincidence of $\dot{\sim}_E$ and $\sim$ can be proved relying on the relationship between the late and the early transition relations. In the following, let us denote them by $\longrightarrow_L$ and by $\longrightarrow_E$, respectively. The result shown in [MPW93] establishes that, for all $\alpha \neq xy$, $P \xrightarrow{\alpha}_E P'$ iff $P \xrightarrow{\alpha}_L P'$. Also, it allows one to infer that the following two requirements on any relation $\mathcal{S}$ are equivalent:

$$\begin{array}{ll} \forall P, P', Q, x, w: & \text{if } P \xrightarrow{xw}_E P' \text{ then } \exists Q' : Q \xrightarrow{xw}_E Q' \text{ and } P'\ \mathcal{S}\ Q' \\ \forall P, P'', Q, x, y: & \text{if } P \xrightarrow{x(y)}_L P'' \text{ then } \forall w \exists Q'' : Q \xrightarrow{x(y)}_L Q'' \text{ and} \\ & P''\{^w\!/\!_y\}\ \mathcal{S}\ Q''\{^w\!/\!_y\} \end{array}$$ □

3.3 Open semantics

The *open bisimulation* [San96] builds on the intuition of moving name instantiation inside the definition of bisimulation, so to immediately capture the flavour of non-groundness.

Definition 10 A binary symmetric relation $\mathcal{S}$ is an *open bisimulation* if $P\ \mathcal{S}\ Q$ implies that for all name substitutions σ

if $P\sigma \xrightarrow{\alpha} P'$ with $\mathrm{bn}(\alpha) \notin \mathrm{fn}(P\sigma, Q\sigma)$, then for some Q', $Q\sigma \xrightarrow{\alpha} Q'$ and

$P'\ \mathcal{S}\ Q'$

$$\begin{tikzcd}
 & & \sim_O \arrow[dl] \\
 & \sim_L \arrow[dl] \arrow[dd] & \\
\sim_E \arrow[dd] & & \\
 & \dot\sim_L \arrow[dl] & \\
\dot\sim_E & &
\end{tikzcd}$$

Table 3: Relationship among late, early, and open semantics

P is *open bisimilar* to Q, written $P \sim_O Q$, if $P \; \mathcal{S} \; Q$ for some open bisimulation $\mathcal{S}$. □

The open paradigm delays the late view about input actions. Precisely, it delays the instantiation of the input formal parameter until it is really needed.

Example 11 Let the processes P and Q be defined as follows.

$$\begin{array}{rcl} P & = & x(y).(\tau.\tau + \tau) \\ Q & = & x(y).(\tau.\tau + \tau + \tau.[y = z]\tau) \end{array}$$

Although $P \sim_L Q$, it holds that $P \not\sim_O Q$. This depends on the fact that P cannot properly react to the game $Q \xrightarrow{x(y)}\xrightarrow{\tau} Q' = [y = z]\tau$. If P chooses to move by executing the transitions $P \xrightarrow{x(y)}\xrightarrow{\tau} P' = \mathtt{nil}$, then the substitution $\{z/y\}$ is such that $Q'\{z/y\} \not\sim_O P'\{z/y\} = \mathtt{nil}$. The only other possibility for P is moving by $P \xrightarrow{x(y)}\xrightarrow{\tau} P'' = \tau$. In this case the identity substitution is sufficient to discriminate between Q' and P''. □

The definition of open bisimulation, involving a universal quantification over substitutions, requires at each step an infinite number of checks. Nevertheless, a more efficient characterization of open bisimilarity was proposed. It is based on a specialized transition system. Labels are pairs (M, α), where M collects the conditions on names which are requested for action α to be performed. Intuitively, M represents the minimal requirement on substitutions to ensure the firing of action α. For instance $[x = y]\alpha.P \xrightarrow{([x=y],\alpha)} P$. The specialized notion of bisimulation involves (essentially) only checks on the minimal substitution induced by the first component of labels.

A complete axiomatization of open bisimilarity of finite processes is proposed in [San96].

As a final remark, notice that open congruence is strictly finer than late equivalence, which in turn is finer than early (*cf.* Ex. 6 and Ex. 11). The relationship among the equivalences considered so far is summarized in Tab. 3, where any arrow stands for strict inclusion.

References

[AB84] D. Austry and G. Boudol. Algèbre de processus et synchronisations. *Theoretical Computer Science*, 30(1):91–131, 1984.

[Ama93] R. Amadio. On the reduction of CHOCS bisimulation to π-calculus bisimulation. In E. Best, editor, *Proc. 4th International Conference on Concurrency Theory, CONCUR '93*, volume 715 of *LNCS*. Springer-Verlag, 1993.

[BCHK91] G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn. Observing Localities (Extended Abstract). In A. Tarlecki, editor, *Proc. 16th International Symp. on Mathematical Foundations of Computer Science, MFCS '91*, volume 520 of *LNCS*. Springer-Verlag, 1991.

[BCHK93] G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn. Observing localities. *Theoretical Computer Science*, 114(1):31–61, 1993. Full version of [BCHK91].

[BHR84] S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A Theory of Communicating Sequential Processes. *Journal of the ACM*, 31(3):560–599, 1984.

[BK84] J.A. Bergstra and J.W. Klop. Process Algebra for Synchronous Communication. *Information and Control*, 60:109–137, 1984.

[BK85] J.A. Bergstra and J.W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985.

[BS94] M. Boreale and D. Sangiorgi. A fully abstract semantics for causality in the pi-calculus. Report ECS-LFCS-94-297, Laboratory for Foundations of Computer Science, Computer Science Department, Edinburgh University, 1994. An extract appeared in the Proc. of STACS '95, LNCS 900.

[DDNM90] P. Degano, R. De Nicola, and U. Montanari. A partial ordering semantics for CCS. *Theoretical Computer Science*, 75:223–262, 1990.

[FG96] C. Fournet and G. Gonthier. The reflexive CHAM and the join-calculus. In *Proc. 23rd Annual ACM Symp. on Principles of Programming Languages*, pages 372–385, 1996.

[Gay93] S.J. Gay. A Sort Inference Algorithm for the Polyadic π-Calculus. In *Proc. 20th Annual ACM Symp. on Principles of Programming Languages*, pages 429–438, 1993.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[Jon93] C.B. Jones. A pi-calculus Semantics for an Object-Based Design Notation. In E. Best, editor, *Proc. 4th International Conference on Concurrency Theory, CONCUR '93*, volume 715 of *LNCS*. Springer-Verlag, 1993.

[LW95a] X. Liu and D. Walker. Confluence of Processes and Systems of Objects. In P.D. Mosses, M. Nielsen, and M.I. Schwartzbach, editors, *Proc. 6th International Joint Conference CAAP/FASE, TAPSOFT '95*, volume 915 of *LNCS*, pages 217–231. Springer-Verlag, 1995.

[LW95b] X. Liu and D. Walker. A Polymorphic Type System for the Polyadic π-calculus. In I. Lee and S.A. Smolka, editors, *Proc. 6th International Conference on Concurrency Theory, CONCUR '95*, volume 962 of *LNCS*, pages 103–116. Springer-Verlag, 1995.

[Mil89] R. Milner. *Communication and Concurrency.* International Series in Computer Science. Prentice Hall, 1989.

[Mil92a] R. Milner. Functions as processes. *Journal of Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[Mil92b] R. Milner. The Polyadic π-Calculus: a Tutorial. In F.L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*, pages 203–246. Springer-Verlag, 1992.

[MPW91] R. Milner, J. Parrow, and D. Walker. Modal Logics for Mobile Processes. In J.C.M. Baeten and J.F. Groote, editors, *Proc. 2nd International Conference on Concurrency Theory, CONCUR '91*, volume 527 of *LNCS*. Springer-Verlag, 1991.

[MPW92] R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, Part I and II. *Information and Computation*, 100(1):1–77, 1992.

[MPW93] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 114(1):149–171, 1993. Full version of [MPW91].

[NC94] M. Nielsen and C. Clausen. Bisimulation for Models in Concurrency. In B. Jonsson and J. Parrow, editors, *Proc. 5th International Conference on Concurrency Theory, CONCUR '94*, volume 836 of *LNCS*. Springer-Verlag, 1994.

[NC95] M. Nielsen and C. Clausen. Games and logics for a noninterleaving bisimulation. *Nordic Journal of Computing*, 2(2):221–249, 1995. Full version of [NC94].

[Plo81] G. D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI-FN-19, Computer Science Department, Aarhus University, 1981.

[PS93] B.C. Pierce and D. Sangiorgi. Typing and Subtyping for Mobile Processes. In *Proc. 8th IEEE Symp. on Logic in Computer Science*, pages 376–385, 1993.

[PS95] J. Parrow and D. Sangiorgi. Algebraic Theories for Name-Passing Calculi. *Information and Computation*, 120(2):174–197, 1995.

[PT95] B.C. Pierce and D.N. Turner. Pict: A Programming Language Based on the π-calculus. 1995.

[San92] D. Sangiorgi. The Lazy Lambda Calculus in a Concurrency Scenario. In *Proc. 7th IEEE Symp. on Logic in Computer Science*, 1992.

[San93a] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD thesis, Laboratory for Foundations of Computer Science, Computer Science Department, Edinburgh University, 1993. Report ECS-LFCS-93-266.

[San93b] D. Sangiorgi. From π-calculus to Higher-order π-calculus – and back. In M.-C. Gaudel and J.-P. Jouannaud, editors, *Proc. 4th International Joint Conference on Theory and Practice of Software Development, TAPSOFT '93*, volume 668 of *LNCS*. Springer-Verlag, 1993.

[San93c] D. Sangiorgi. A Theory of Bisimulation for the π-calculus. In E. Best, editor, *Proc. 4th International Conference on Concurrency Theory, CONCUR '93*, volume 715 of *LNCS*. Springer-Verlag, 1993.

[San94a] D. Sangiorgi. The Lazy Lambda Calculus in a Concurrency Scenario. *Information and Computation*, 111(1), 1994. Full version of [San92].

[San94b] D. Sangiorgi. Locality and Non-interleaving Semantics in calculi for mobile processes. Report ECS-LFCS-94-282, Laboratory for Foundations of Computer Science, Computer Science Department, Edinburgh University, 1994. To appear in *Theoretical Computer Science.*

[San96] D. Sangiorgi. A Theory of Bisimulation for the π-calculus. *Acta Informatica*, 33(1):69–97, 1996. Full version of [San93c].

[San97] D. Sangiorgi. The name discipline of uniform receptiveness. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Proc. 24th International Colloquium on Automata, Languages and Programming*, volume 1256 of *LNCS*, pages 303–313. Springer, 1997.

[VH93] V.T. Vasconcelos and K. Honda. Principal typing schemes in a polyadic π-calculus. In E. Best, editor, *Proc. 4th International Conference on Concurrency Theory, CONCUR '93*, volume 715 of *LNCS*. Springer-Verlag, 1993.

[Wal95] D. Walker. Objects in the π-Calculus. *Information and Computation*, 116(2):253–271, 1995.

Recent BRICS Lecture Series Publications

LS-98-4 Paola Quaglia. *The π-Calculus: Notes on Labelled Semantics*. December 1998. viii+16 pp.

LS-98-3 Olivier Danvy. *Type-Directed Partial Evaluation*. December 1998. Extended version of lecture notes to appear in Hatcliff, Mogensen and Thiemann, editors, *Partial Evaluation: Practice and Theory,* PEPT '98 Lecure Notes, LNCS, 1998.

LS-98-2 Carsten Butz. *Regular Categories and Regular Logic*. October 1998.

LS-98-1 Ulrich Kohlenbach. *Proof Interpretations*. June 1998.

LS-97-1 Jan Chomicki and David Toman. *Temporal Logic in Information Systems*. November 1997. viii+42 pp. Full version appears in Chomicki and Saake, editors, *Logics for Database and Information Systems,* 3:31–70, Kluwer Academic Publishers, 1998.

LS-96-6 Torben Braüner. *Introduction to Linear Logic*. December 1996. iiiv+55 pp.

LS-96-5 Devdatt P. Dubhashi. *What Can't You Do With LP?* December 1996. viii+23 pp.

LS-96-4 Sven Skyum. *A Non-Linear Lower Bound for Monotone Circuit Size*. December 1996. viii+14 pp.

LS-96-3 Kristoffer H. Rose. *Explicit Substitution – Tutorial & Survey*. September 1996. v+150 pp.

LS-96-2 Susanne Albers. *Competitive Online Algorithms*. September 1996. iix+57 pp.

LS-96-1 Lars Arge. *External-Memory Algorithms with Applications in Geographic Information Systems*. September 1996. iix+53 pp.

LS-95-5 Devdatt P. Dubhashi. *Complexity of Logical Theories*. September 1995. x+46 pp.

LS-95-4 Dany Breslauer and Devdatt P. Dubhashi. *Combinatorics for Computer Scientists*. August 1995. viii+184 pp.

LS-95-3 Michael I. Schwartzbach. *Polymorphic Type Inference*. June 1995. viii+24 pp.

LS-95-2 Sven Skyum. *Introduction to Parallel Algorithms*. June 1995. viii+17 pp. Second Edition.