INRIA

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 1154

Programme 1
Programmation, Calcul Symbolique
et Intelligence Artificielle

FUNCTIONS AS PROCESSES

Robin MILNER

Février 1990

RR-1154

Functions as Processes
Les Fonctions vues comme des Processus

Robin Milner
University of Edinburgh
June 1989

Abstract This paper exhibits accurate encodings of the λ-calculus in the π-calculus. The former is canonical for calculation with functions, while the latter is a recent step [12] towards a canonical treatment of concurrent processes. With quite simple encodings, two λ-calculus reduction strategies are simulated very closely; each reduction in λ-calculus is mimicked by a short sequence of reductions in π-calculus. Abramsky's precongruence of *applicative simulation* [1] over λ-calculus is compared with that induced by the encoding of the lazy λ-calculus into π-calculus; a similar comparison is made for call-by-value λ-calculus.

The part of π-calculus which is needed for the encoding is formulated in a new way, inspired by Berry's and Boudol's Chemical Abstract Machine [3]. The new formulation is shown to be consistent with the original.

Résumé Ce papier montre des codages précis du λ-calcul en π-calcul. L'un est canonique pour faire des calculs avec des fonctions; l'autre est une tentative récente [12] en direction d'un traitement canonique des processus concurrents. A l'aide de codages assez simples, on simule très exactement deux stratégies de réduction; on imite chaque réduction en λ-calcul par une série brève de réductions en π-calcul. On compare la précongruence sur le λ-calcul d'Abramsky [1], *simulation applicative*, avec celle qu'induit le codage du λ-calcul *paresseux* en π-calcul; on compare également les deux précongruences dans le cas du λ-calcul avec *appel par valeur*.

On présente une formulation nouvelle de la partie du π-calcul qui sert au codage, inspirée de la Machine Abstraite Chimique de Berry et Boudol [3]. On montre que cette formulation nouvelle est fidèle à la formulation originelle.

Remerciements Cet article a été écrit pendant un séjour de quatre mois à l'INRIA Sophia Antipolis, à la fin de 1989. Je voudrais remercier l'INRIA pour son hospitalité, le Ministère de la Recherche et de la Technologie pour son soutien, ainsi que l'Université d'Edimbourg qui m'a accordé ce congé sabbatique.

Functions as Processes

Robin Milner
University of Edinburgh
June 1989

1 Introduction

The main purpose of this paper is to exhibit accurate encodings of the λ-calculus in the π-calculus. The former is canonical for calculation with functions, while the latter is a recent step [12] towards a canonical treatment of concurrent processes. We show that, with quite simple encodings, at least two λ-calculus reduction strategies can be simulated very closely; each reduction in λ-calculus is mimicked by a short sequence of reductions in π-calculus. For the encoding of lazy λ-calculus, we compare Abramsky's precongruence over λ-terms known as *applicative simulation* [1] with that which is induced by the encoding; we also make a similar comparison for the call-by-value λ-calculus.

Recently, several generalisations of process calculus to embrace higher-order objects have been proposed; examples are by Boudol [4], F.Nielsen [13] and Thomsen [16]. Explicitly or implicitly, they have enabled embedding of the λ-calculus. The π-calculus, which builds upon work by Engberg and M.Nielsen [7], differs in the following respect: It replaces the paradigm in which an agent can be transmitted as a value and bound to a variable, which we may call the *function* paradigm, by a significantly different paradigm. In the latter, which we may call the *object*[1] paradigm, that which is transmitted and bound is never an agent, but rather *access* to an agent. It is only as a special case that one agent may have *sole* access to another. The object paradigm is hardly new, but there has never been a canonical encapsulation of it, in the way that λ-calculus encapsulates the function paradigm. But there have been signs of its power; for example, following

[1] The connotation here is with object-oriented programming – in particular, the idea of objects with independent state interacting with one another.

an early ideas of Hewitt [6], even a humble data value can be modelled as an independent agent – one which remains constant (i.e. invariant under access.)

Both paradigms seem equally basic and significant. Without arguing naïvely *in favour* of the object paradigm, our aim in the π-calculus has been to present it in undiluted form. Since higher-order parameters are sacrificed, a succinct translation between the paradigms is by no means preordained; all that we know is that *some* translation from the function to the object paradigm must exist, since functional languages are successfully implemented on machines which – with their arithmetic units, programs and registers – can be seen just as assemblies of objects.

Another theme of the present paper is a new way of formulating the π-calculus, or at least the fragment of π-calculus that we need for encoding λ-calculus, inspired by Berry's and Boudol's Chemical Abstract Machine [3]. Their analogy of an aggregate of processes moving and interacting within a *solution* has probably occurred vaguely to many people, but Berry and Boudol have made the analogy technically robust. We reflect their intuition here by means of axioms for a structural congruence relation over process terms; this yields a welcome simplicity in presenting the reduction rules.

The paper is self-contained as far as its main purpose, encoding λ-calculus, is concerned; no knowledge of our original presentation of π-calculus [12] is needed. However, we also give results which state that the new presentation is fully consistent with the original.

2 The π-calculus: Terms

We presuppose an infinite set $\mathcal{N}$ of *names*. We shall use $x, y, z, \ldots$ and sometimes other small letters to range over $\mathcal{N}$.

The π-calculus consists of a set $\mathcal{P}$ of *terms*, sometimes called *agents*, which intuitively stand for processes. We shall use P, Q, R to range over $\mathcal{P}$. We shall write $P\{y/x\}$ to mean the result of replacing free occurrences of x by y in P, with change of bound names where necessary, as usual. (There will be two ways of binding names.)

The first class of terms consists of *guarded terms* $g.P$, where P is a term

and g is a *guard*; guards g have the form

$$g \quad ::= \quad \overline{x}y \quad | \quad x(y)$$

Informally, $\overline{x}y.P$ means 'send the name y along the link named x, and then enact P'; on the other hand, $x(y)$ means 'receive any name z along the link named x, and then enact $P\{^z/_y\}$'. Thus the guard $x(y)$ is like the λ-prefix λy in that it binds y; it is unlike λy in that every name $x \in \mathcal{N}$ is a binder like λ, but that only names (not terms) may replace bound names.

The second class of terms express concurrent behaviour. The principal form is *composition* $P|Q$, which, informally speaking, enacts P and Q concurrently allowing them to interact via shared links (i.e. shared names). Interaction can occur in the case

$$x(y).P \mid \overline{x}z.Q \tag{1}$$

and we expect the result of interaction to be $P\{^z/_y\}|Q$. This differs from β-reduction $(\lambda xM)N \rightarrow M\{^N/_x\}$ in one essential: the 'sender' $\overline{x}z.Q$ pursues an independent future (as Q) after the interaction, while in β-reduction the future behaviour of the argument N is controlled by M via the variable x (in a way which varies from one reduction strategy to another). This exactly reflects the crucial difference, mentioned in the introduction, between the *function* and *object* paradigms.

Allied to composition is *replication*, $!P$; roughly, it stands for $P|P|\cdots$, as many concurrent instances of P as you like. Also allied is *inaction*, the degenerate composition of no processes, denoted by $\mathbf{0}$.

The third class of terms has only one form: the *restriction* $(x)P$. It confines the use of x as a link to within P. Thus, no *intraaction*, i.e. interaction between components, can occur in

$$x(y).P \mid (x)(\overline{x}z.Q) \tag{2}$$

On the other hand, if (x) is applied to (1) then no *inter*action at x can occur between this term and any terms which may surround it; one therefore expects the equation

$$(x)(x(y).P \mid \overline{x}z.Q)) \;\approx\; (x)(P\{^z/_y\} \mid Q) \tag{3}$$

for some congruence relation $\approx$.

To summarise: for this paper, the syntax of P is

$$P \quad ::= \quad \overline{x}y.P \quad | \quad x(y).P \quad | \quad \mathbf{0} \quad | \quad P|Q \quad | \quad !P \quad | \quad (y)P$$

There are a few differences from the π-calculus given in [12]. Only one is a novelty: the presence of $!P$. Replication[2] replaces recursion; for many purposes replication does the job of recursion (perhaps even for all useful purposes), and is simpler to handle theoretically. Otherwise, we are dealing with a sub-calculus. We have omitted $\tau.P$ (silent guard), $[x{=}y]P$ (matching) and $P+Q$ (summation). The first two present no difficulty for the present formulation, nor does summation in the limited form $\sum_i g_i.P_i$ (sum of guarded terms). It is open how best to handle full summation in the present formulation; but see the method adopted by Berry and Boudol in the Chemical Abstract Machine [3].

Some technical details and terminology:

- There are two forms of binding: $x(y)$ and (y). Note that x is free in $x(y).P$. We use $\mathrm{fn}(P)$ for the free names of P, and $\mathrm{n}(P)$ for all names occurring in P.
- We call x the *subject* and y the *object* of the guards $x(y)$ and $\overline{x}y$.
- We say that an occurrence of a term Q in P is *guarded* if it occurs within any guarded term in P, otherwise it is *unguarded.*
- We shall often use $\tilde{x}$ to mean a sequence $x_1, \ldots, x_n$ of names; similarly $\tilde{P}$ for a sequence of terms. Without risk of confusion, we also treat $\tilde{x}$ sometimes as a set. We also write $(\tilde{x})$ for the multiple restriction $(x_1)\cdots(x_n)$.
- We shall use several convenient abbreviations:

[2] Later in the paper we formally state results comparing the two presentations of π-calculus. For that purpose, we assume that the rule for replication

$$\frac{P \xrightarrow{a} P'}{!P \xrightarrow{a} P'|!P}$$

is added to those in Table 2, Part II of [12].

- We shall often omit '.**0**' in an agent, and write for example $\overline{x}y$ instead of $\overline{x}y.\mathbf{0}$.
- We shall elide several guards with the same subject, for example $x(y)(z)$ means $x(y).x(z)$ and $\overline{x}yz$ means $\overline{x}y.\overline{x}z$.
- The agent $(y)\overline{x}y.P$ can be thought of as simultaneously creating and sending a new private name, when $x \neq y$; we abbreviate it to $\overline{x}(y).P$.

So with all these abbreviations we shall be able to write agents like $x(y)(z).\overline{y}z(x)$, meaning $x(y).x(z).\overline{y}z.(x)\overline{y}x.\mathbf{0}$.

3 The π-calculus: Equations and Reductions

Our operational intuition is simple: if term R contains two unguarded subterms $x(y).P$ and $\overline{x}z.Q$, and each restriction (x) contains both or neither (so that x means the same for both subterms), then they can interact; this interaction yields a reduction $R \rightarrow R'$. A few examples:

(1) Let R be $x(y).P \mid \overline{x}z_1.Q_1 \mid \overline{x}z_2.Q_2$. There are two reductions

$$\begin{aligned} R &\rightarrow P\{z_1/y\} \mid Q_1 \mid \overline{x}z_2.Q_2 \\ R &\rightarrow P\{z_2/y\} \mid \overline{x}z_1.Q_1 \mid Q_2 \end{aligned}$$

(2) Let R be $w(x).(x(y).P \mid \overline{x}z.Q)$. There is no reduction; the subterms are guarded.

(3) Let R be $x(y).P \mid (x)(\overline{x}z.Q)$. There is no reduction.

(4) Let R be $x(y).P \mid (z)(\overline{x}z.Q)$. Assuming z not free in P there is a reduction

$$R \rightarrow (z)(P\{z/y\} \mid Q)$$

We call this phenomenon *extrusion* (of the scope of a restriction); the name z is private to $(z)\overline{x}z.Q$, but its transmission has enlarged its scope to embrace the recipient.

To simplify the form of the reduction rules, we first define a structural congruence relation over terms. This approach is inspired by Boudol and Berry [3], though our formulation differs from theirs. The idea is that one should separate the laws which govern the neighbourhood relation among processes from the rules which specify their interaction.

3.1 Definition *Structural congruence*, written $\equiv$, is the smallest congruence over $\mathcal{P}$ satisfying these equations:

(1) $P \equiv Q$ whenever P is alpha-convertible to Q

(2) $P \mid \mathbf{0} \equiv P$, $P \mid Q \equiv Q \mid P$, $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$

(3) $!P \equiv P \mid !P$

(4) $(x)\mathbf{0} \equiv \mathbf{0}$, $(x)(y)P \equiv (y)(x)P$

(5) $(x)(P \mid Q) \equiv P \mid (x)Q$ if x not free in P ■

A few facts are easily seen:

- Using the equations, all unguarded restrictions can be moved outermost. Note particularly: $!(x)P \equiv (x)P \mid !(x)P \equiv (x)(P \mid !(x)P)$.
- The interaction condition mentioned at the start of this section is invariant under $\equiv$.
- Using the equations, any two potential interactors $x(y).P$, $\overline{x}z.Q$ can be brought together as $x(y).P \mid \overline{x}z.Q$, but possibly with alpha-conversion; for example, if z is free in P then

$$x(y).P \mid (z)(\overline{x}z.Q) \;\equiv\; (z')(x(y).P \mid \overline{x}z'.Q\{z'/z\})$$

where z' is new.

3.2 Definition *Reduction*, written $\rightarrow$, is the smallest relation satisfying the following rules:

$$\text{COM}: \quad x(y).P \mid \overline{x}z.Q \rightarrow P\{^z/_y\} \mid Q$$

$$\text{PAR}: \quad \frac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q}$$

$$\text{RES}: \quad \frac{P \rightarrow P'}{(y)P \rightarrow (y)P'}$$

$$\text{STRUCT}: \quad \frac{Q \equiv P \quad P \rightarrow P' \quad P' \equiv Q'}{Q \rightarrow Q'}$$

■

It is worth noticing that, just because of equation (3) in 3.1 for replication, structural congruence may be hard to determine (perhaps even undecidable), and this may cause some alarm in seeing the rule STRUCT, since we certainly want $\rightarrow$ to be computable. But we are saved by the invariance of the interaction condition under $\equiv$, noted above; the interactions possible in a given term are quite manifest. In fact:

3.3 Proposition A finite set $\text{Red}(P)$ of agents can be recursively computed from P, such that $P \rightarrow P'$ if and only if $P' \equiv P''$ for some $P'' \in \text{Red}(P)$. ■

The present formulation of π-calculus differs strikingly from that in [12]. The essential difference is in the use of structural congruence. This is inspired by the Chemical Abstract Machine of Berry and Boudol [3]. Their insight is that the rules of structured operational semantics, as used in [11] or [12] for example, treat in the same way two concepts which it is worth separating: the *physical structure* of a family of concurrent agents and their *interaction*. Of course, one advantage of avoiding the imposition of structural congruence equations as axioms is that, once a *behavioural* congruence is defined – as in [11] – it turns out that the structural equations are obtained as *theorems*, and this gives added confidence. But then these equations are not kept clearly distinct from other theorems which only hold

because of the particular way in which observation is characterised in the behavioural congruence. (For example, observation congruence in [11] is based upon the idea of a *sequential observer*, and yields equations which do not hold in a model which respects causality.) By proposing the structural laws as axioms, one makes the distinction and achieves some simplicity at the same time (at least, when summation is not considered); one also offers the challenge to find an interesting behavioural congruence which fails to satisfy the axioms (strongly suspecting that there is none!)

But now we have an obligation to show that the formulations agree. In [12] there is a labelled transition system $\stackrel{\alpha}{\rightarrow}$, where the *actions* α are of four kinds:

$$\alpha \quad ::= \quad \tau \quad | \quad \overline{x}y \quad | \quad x(y) \quad | \quad \overline{x}(y)$$

Of these, the τ-actions – i.e. the *intra*actions – correspond to our reductions; we do not need the rest yet. (This is another reason for the greater simplicity of the presentation here, as far as we have taken it; we have so far told enough of the story to encode λ-calculus, but no more.) In fact, the correspondence is nearly exact:

3.4 Proposition Over $\mathcal{P}$, the relations $\stackrel{\tau}{\rightarrow}\equiv$ and $\rightarrow$ are identical. ■

For the next section we shall need the following:

3.5 Definition P is *r-determinate* if, whenever $P \rightarrow^* Q$ and also $Q \rightarrow Q_1$ and $Q \rightarrow Q_2$, then $Q_1 \equiv Q_2$. Also, P *converges* to Q, written $P \downarrow Q$, if $P \rightarrow^* Q \not\rightarrow$; we write $P \downarrow$ to mean P converges to some Q, and $P \uparrow$ otherwise. ■

4 The lazy λ-calculus

Let the set of *Variables*, $\mathcal{X}$, be an infinite and co-infinite subset of $\mathcal{N}$. For this section, we shall let x, y, z range over $\mathcal{X}$, and u, v, w range over $\mathcal{N} - \mathcal{X}$. Our encoding of λ-calculus into π-calculus will be all the simpler because we treat a variable x of λ-calculus also as a name of π-calculus.

We shall use L, M, N to range over the *terms* $\mathcal{L}$ of λ-calculus, which are

defined as usual by:

$$M \quad ::= \quad x \quad | \quad \lambda x M \quad | \quad MN$$

the last two forms being *abstraction* and *application*. The *free variables* $\mathrm{fv}(M)$ of a term M are defined in the usual way. $\{^N/_x\}$ means substitution as usual. We shall frequently need the the sequential composition of several substitutions (note: *not* simultaneous substitution); so, understanding the members of $\tilde{x}$ to be distinct, we introduce the abbreviation

$$\{^{\tilde{N}}/_{\tilde{x}}\} \quad \text{stands for} \quad \{^{N_1}/_{x_1}\} \cdots \{^{N_k}/_{x_k}\}$$

We shall use the standard terms

$$\begin{aligned} \mathbf{I} &\stackrel{\text{def}}{=} \lambda x x \\ \mathbf{K} &\stackrel{\text{def}}{=} \lambda x \lambda y x \\ \mathbf{Y} &\stackrel{\text{def}}{=} \lambda f (\lambda x\, f(xx))(\lambda x\, f(xx)) \\ \mathbf{\Omega} &\stackrel{\text{def}}{=} (\lambda x\, xx)(\lambda x\, xx) \end{aligned}$$

There are many reduction relations $\rightarrow$, many of which satisfy the rule

$$\beta: \quad (\lambda x M)N \rightarrow M\{^N/_x\}$$

The relations differ as to which contexts admit reduction. The simplest, in some sense, is that which admits reduction only at the extreme left end of a term. This is known as lazy reduction:

4.1 Definition The *lazy reduction relation* $\rightarrow$ over $\mathcal{L}$ is the smallest which satisfies β, together with the rule

$$\text{APPL}: \qquad \frac{M \rightarrow M'}{MN \rightarrow M'N}$$

■

With the usual convention that LMN means $(LM)N$, this implies that in any term M, writing it as

$$M \equiv M_0 M_1 M_2 \cdots M_n \qquad (n \geq 0)$$

where M_0 is not an application, the only reduction possible is when $n \geq 1$ and $M_0 \equiv \lambda x N$, and then the reductum is

$$N\{M_1/x\}M_2 \cdots M_n$$

Thus $\rightarrow$ is r-determinate – i.e every M is r-determinate; given M, there is at most one M' for which $M \rightarrow M'$. We write $\rightarrow^+$ for the transitive closure of $\rightarrow$, and $\rightarrow^*$ for the transitive reflexive closure.

4.2 Definition M *converges* to M', written $M \downarrow M'$, if $M \rightarrow^* M' \not\rightarrow$; also, $M\downarrow$ means that M converges to some M'. ∎

If $M \downarrow M'$, then M' can only be an abstraction $\lambda x N$ or else of the form $xN_1 \cdots N_n$ $(n \geq 0)$. Writing $\mathcal{L}^0$ for the *closed* terms, if $M \in \mathcal{L}^0$ then $M' \in \mathcal{L}^0$ also, so M' must be an abstraction.

Abramsky [1] defines an important preorder $\lesssim$, which we shall call *applicative simulation*, as follows:

4.3 Definition Let $L, M \in \mathcal{L}^0$. Then $L \lesssim M$ if, for all sequences $\widetilde{N}$ in $\mathcal{L}^0$:

$$L\widetilde{N}\downarrow \text{ implies } M\widetilde{N}\downarrow$$

Furthermore, if $L, M \in \mathcal{L}$ with free variables $\tilde{x}$, we define $L \lesssim M$ to mean that $L\{\widetilde{N}/\tilde{x}\} \lesssim M\{\widetilde{N}/\tilde{x}\}$ for all $\widetilde{N}$ in $\mathcal{L}^0$. ∎

Abramsky continues to study both the model theory and the proof theory of $\lesssim$ and of applicative *bisimulation*, $\eqsim$ (i.e. $\lesssim \cap \gtrsim$). We need very little of this here, but should remark that it firmly establishes the importance both of lazy reduction and of these relations, and hence provides a natural point at which process calculus may try to make contact with λ-calculus. We recall from [1] that $(\lambda x M)N \eqsim M\{N/x\}$, and that $\lesssim$ is a precongruence. The latter follows from the result that, defining a *context* $C[\_]$ to be a term with a single hole,

$$M \lesssim N \text{ iff, for every closed context } C[\_],\ C[M]\downarrow \text{ implies } C[N]\downarrow$$

We now turn to the encoding of $\mathcal{L}$ into $\mathcal{P}$. Perhaps one hardly expects to find a more basic calculus than the λ-calculus. All the same, it takes as

primitive the remarkably complex operation of substitution (of terms for variables). Two important means have been found to break this operation into smaller parts. In combinatory algebra [5], Curry found combinators which progressively distribute the argument of an abstraction $\lambda x M$ to those parts of the body M which will use it (thus, in fact, eliminating variables altogether). On the other hand, implementations of functions and procedures in programming languages have traditionally used the notion of *environment*, a map from variables to terms; thus, instead of executing $M\{^N/_x\}$ one executes M itself in an environment which binds N to the variable x. The encoding which follows can be seen as a formalisation of the latter idea.

Each $M \in \mathcal{L}$ is encoded as $[\![M]\!]$, a map from names to $\mathcal{P}$. Thus $[\![M]\!]u$ is a term of π-calculus, and will have free names given by

$$\mathrm{fn}([\![M]\!]u) \;=\; \mathrm{fv}(M) \cup \{u\}$$

The name u is the link along which $[\![M]\!]$ 'receives' its arguments.

Now, suppose that M will itself be used in place of an argument represented by the variable x. Each time M is 'called', via x, it must be told by the caller where to receive its own arguments. (In more familiar terminology, it must be given a *pointer* to its arguments.) The 'environment entry' binding x to M is therefore the π-term

$$[\![x := M]\!] \stackrel{\mathrm{def}}{=} !\, x(w).[\![M]\!]w$$

In passing, note particularly the replication. This is not needed if M will be called at most once; therefore the *linear* λ-calculus, in which each variable x must occur exactly once in its scope, may be encoded in the fragment of π-calculus without replication. The link with Girard's 'of course' connective '!' of linear logic [8] should be explored; his notation for it has been chosen here deliberately.

How does $[\![\lambda x M]\!]u$ receive its arguments? Along u it receives (as x) the name of its first argument, and also the name of a link where the rest will be transmitted. This explains the first line of our encoding, which we now give in full:

$$[\![\lambda x M]\!]u \;\stackrel{\mathrm{def}}{=}\; u(x)(v).[\![M]\!]v$$

$$
\begin{aligned}
[\![x]\!]u &\stackrel{\text{def}}{=} \overline{x}u \\
[\![MN]\!]u &\stackrel{\text{def}}{=} (v)([\![M]\!]v \mid \overline{v}(x)u.[\![x:=N]\!]) \\
&\qquad (x \text{ not free in } N)
\end{aligned}
$$

Let us look at the reduction of a simple example, in which we assume x not free in N (recall the abbreviations listed at the end of Section 2):

$$
\begin{aligned}
[\![(\lambda x x)N]\!]u &\equiv (v)(v(x)(w).\overline{x}w \mid \overline{v}(x)u.[\![x:=N]\!]) \\
&\rightarrow (v)(x)(v(w).\overline{x}w \mid \overline{v}u.[\![x:=N]\!]) && (4) \\
&\rightarrow (x)(\overline{x}u \mid [\![x:=N]\!]) && (5) \\
&\equiv (x)(\overline{x}u \mid !\,x(w).[\![N]\!]w) \\
&\rightarrow [\![N]\!]u \mid (x)[\![x:=N]\!] && (6) \\
&\sim [\![N]\!]u && (7)
\end{aligned}
$$

The following remarks will help in reading the above calculation:

(1) In obtaining (4), recalling that $\overline{v}(x).Q$ means $(x)\overline{v}x.Q$, equation (5) of Definition 3.1 must first be used to allow COM to be applied.

(2) The restriction (v) is dropped in (5) because v no longer occurs. Formally, if $v \notin \mathrm{fn}(R)$ then

$$(v)R \equiv (v)(R \mid \mathbf{0}) \equiv R \mid (v)\mathbf{0} \equiv R \mid \mathbf{0} \equiv R$$

(3) In (6), (x) has been moved inwards, since $x \notin \mathrm{fn}([\![N]\!]u)$.

(4) The last step, to (7), is the only one which goes beyond $\equiv$; it is a simple case of *strong bisimilarity* – see [12] – and represents the garbage-collection of an environment entry $[\![x:=N]\!]$ which cannot be used further (since the subject x of its first action is restricted).

We are now ready to embark upon a proof that the reduction of $[\![M]\!]$ in the π-calculus simulates that of M in the λ-calculus very closely. The essential difference, as was mentioned earlier, is that substitutions $\{^M/_x\}$, which are actually performed upon λ-terms, are represented in $\mathcal{P}$ by what we have called environment entries $[\![x:=M]\!]$ which are agents in their own right.

We shall frequently need the parallel composition in π-calculus of several environment entries. So we introduce the abbreviation

$$[\![\tilde{x}:=\widetilde{N}]\!] \quad \text{stands for} \quad [\![x_1:=N_1]\!] \mid \cdots \mid [\![x_k:=N_k]\!]$$

We now define the correspondence between *closed* λ-terms and π-terms which is the basis of our simulation:

4.4 Definition Let the relation $S \subseteq \mathcal{L}^0 \times \mathcal{P}$ contain all pairs (L,P) such that for some $k \geq 0$, some $M, N_1, \ldots, N_k \in \mathcal{L}$ and distinct $x_1, \ldots, x_k \in \mathcal{X}$:

(i) $\mathrm{fv}(M) \subseteq \tilde{x}$, and $\mathrm{fv}(N_i) \subseteq \{x_{i+1}, \ldots, x_k\}$ for all $1 \leq i \leq k$. (So, in particular, $\mathrm{fv}(N_k) = \emptyset$.)

(ii) $L \equiv M\{\tilde{N}/\tilde{x}\}$

(iii) $P \equiv (\tilde{x})([\![M]\!]u \mid [\![\tilde{x}:=\widetilde{N}]\!])$ ∎

4.5 Lemma For any $(L,P) \in S$, P is r-determinate, and one of the following conditions holds:

A. L is an abstraction, and for some $(L',P') \in S$

$$L \equiv L' \quad \text{and} \quad P \downarrow P'$$

B. For some $(L',P') \in S$

$$L \to L' \quad \text{and} \quad P \to^+ P'$$

Proof Let $(L,P) \in S$, and let us use the notation of the definition. The determinacy of P will emerge during the proof that A or B holds. Let M be $M_0 M_1 \cdots M_n$ where M_0 is not an application.
Case 1: $n = 0$ and M_0 is an abstraction $\lambda x_0 M_0'$. Then clearly L is an abstraction, and by inspection

$$P \equiv (\tilde{x})(u(x_0)(v).[\![M_0']\!]v \mid [\![\tilde{x}:=\widetilde{N}]\!])$$

has no reduction. Thus condition A holds with $(L', P') = (L, P)$.

Case 2: $n > 0$ and M_0 is an abstraction $\lambda x_0 M_0'$; w.l.o.g. we can assume $x_0 \notin \tilde{x}$. Then $L \to L'$ where

$$\begin{aligned} L' &\equiv (M_0'\{M_1/x_0\}M_2 \cdots M_n)\{\tilde{N}/\tilde{x}\} \\ &\equiv (M_0' M_2 \cdots M_n)\{M_1/x_0\}\{\tilde{N}/\tilde{x}\} \end{aligned}$$

On the other hand we have

$$\begin{aligned} [\![M]\!]u \equiv (\tilde{v}) \Big([\![M_0]\!]v_1 \;&|\; \overline{v}_1(y_1)v_2.[\![y_1 := M_1]\!] \\ &|\; \overline{v}_2(y_2)v_3.[\![y_2 := M_2]\!] \\ &\quad \cdots \\ &|\; \overline{v}_n(y_n)u.[\![y_n := M_n]\!] \Big) \end{aligned}$$

where $(\tilde{v}) = (v_1) \cdots (v_n)$; also

$$[\![M_0]\!]v_1 \equiv v_1(x_0)(v).[\![M_0']\!]v$$

Now it is clear that P has two successive interactions at v_1, with no alternatives, whose effect is to replace $[\![M]\!]u$ in P by

$$\begin{aligned} (x_0)(v_2) \cdots (v_n) \Big([\![M_0']\!]v_2 \;&|\; [\![x_0 := M_1]\!] \\ &|\; \overline{v}_2(y_2)v_3.[\![y_2 := M_2]\!] \\ &\quad \cdots \\ &|\; \overline{v}_n(y_n)u.[\![y_n := M_n]\!] \Big) \end{aligned}$$

and now, respecting $\equiv$, the restriction (x_0) can be moved outermost in P, and the component $[\![x_0 := M_1]\!]$ to a new position just before $[\![x_1 := N_1]\!]$. Calling this new term P', we have that $P \to^2 P'$ and $(L', P') \in \mathcal{S}$, satisfying condition B.

Case 3: M_0 is a variable x_i. Then we proceed by induction on $k-i$. First, we have

$$L \equiv (N_i M_1 \cdots M_n)\{\tilde{N}/\tilde{x}\}$$

while within P we have

$$\begin{aligned} [\![M]\!]u \equiv (\tilde{v}) \Big(\overline{x_i}v_1 \;&|\; \overline{v}_1(y_1)v_2.[\![y_1 := M_1]\!] \\ &\quad \cdots \\ &|\; \overline{v}_n(y_n)u.[\![y_n := M_n]\!] \Big) \end{aligned}$$

(which is just $\overline{x}u$ when $n=0$). Then the only action of P is an interaction with the replicator $[\![x_i := N_i]\!]$, generating a copy $[\![N_i]\!]v_1$ which (up to $\equiv$) can be moved inwards to replace $\overline{x_i}v_1$, yielding in place of $[\![M]\!]u$

$$[\![M]\!]u \equiv (\tilde{v})\Big([\![N_i]\!]v_1 \mid \overline{v}_1(y_1)v_2.[\![y_1 := M_1]\!] \\ \cdots \\ \mid \overline{v}_n(y_n)u.[\![y_n := M_n]\!] \Big)$$

Now this is exactly $[\![M']\!]u$, where M' is $N_iM_1\cdots M_n$. So we have shown that $P \to P''$ with $(L,P'') \in \mathcal{S}$. But N_i must be of form $M_0'\cdots M_m'$, M_0 being either an abstraction or a variable x_j, $j > i$; so by Case 2 or by induction either A or B holds for (L,P'') in place of (L,P). But $P \to P''$, so the corresponding condition holds for (L,P) too.

Finally, P's reductions have in every case been determined, leading to P' with $(L',P') \in \mathcal{S}$ for some L'; so P is determinate. ■

4.6 Theorem (Lazy encoding) For all $L \in \mathcal{L}^0$, $[\![L]\!]u$ is r-determinate, and one of the following conditions holds:

A. $L\downarrow L'$ and $[\![L]\!]u \downarrow P'$, where

$$L' \equiv \lambda y M\{\tilde{N}/\tilde{x}\} \qquad \text{and} \qquad P' \equiv (\tilde{x})\Big([\![\lambda y M]\!]u \mid [\![\tilde{x} := \widetilde{N}]\!]\Big)$$

B. $L\uparrow$ and $[\![L]\!]u\uparrow$.

Proof Immediate, by iterating the lemma starting from $(L,[\![L]\!]u) \in \mathcal{S}$.■

5 The π-calculus: Actions

The reduction relation $\to$ only tells part of the story of the behaviour of a π-term P; it describes how P's parts may interact with each other, but not how P (or its parts) may interact with the environment.

At first sight, the same remark applies to reduction in λ-calculus; reduction takes a term up to an abstraction, but what happens later depends upon what argument the environment supplies. But the analogy breaks down, at least for the lazy λ-calculus; no interaction between a λ-term and

the environment can take place until it becomes an abstraction, and at that point nothing *but* interaction with the environment can occur. On the other hand, the essence of concurrency is that *inter*action and *intra*action (= reduction) can freely intermingle. Take for example the π-term

$$R_1 \equiv (x)(\overline{y}z.P \mid x(u).\mathbf{0} \mid \overline{x}v.\mathbf{0})$$

As far as reduction is concerned, it is no different from

$$R_2 \equiv (x)(x(u).\mathbf{0} \mid \overline{x}v.\mathbf{0})$$

But in a context where $y(w).Q$ is present, R_1 may behave very differently from R_2, since its interaction with the context may remove the guard from P.

We shall use the term *action* to embrace such *potential* interactions, as well as reduction. We naturally expect two kinds of potential interaction, input and output, and they are represented by agents of the form

$$\begin{array}{ll} (\tilde{z})(x(y).P \mid \cdots) & (x \notin \tilde{z}) \\ (\tilde{z})(\overline{x}y.P \mid \cdots) & (x, y \notin \tilde{z}) \end{array}$$

But there is also a third kind, when the object of an output guard is restricted. This is represented by an agent of the form

$$(y)(\tilde{z})(\overline{x}y.P \mid \cdots) \qquad (x, y \notin \tilde{z})$$

Thus, including reduction, there are four kinds of action. We represent them by the *action relation* $\xrightarrow{\alpha}$, where

$$\alpha \quad ::= \quad \tau \quad | \quad x(y) \quad | \quad \overline{x}y \quad | \quad \overline{x}(y)$$

Thus $\xrightarrow{\tau}$ will mean the same as $\rightarrow$. To define $\xrightarrow{\alpha}$ we modify and extend the reduction rules of 3.2:

5.1 Definition The *action relations* $\xrightarrow{\alpha}$ are the smallest which satisfy the following rules:

$$\text{IN}: \qquad x(y).P \xrightarrow{x(y)} P$$

$$\text{OUT}: \quad \overline{x}y.P \xrightarrow{\overline{x}y} P$$

$$\text{COM}: \quad x(y).P \mid \overline{x}z.Q \xrightarrow{\tau} P\{z/y\} \mid Q$$

$$\text{PAR}: \quad \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad (\mathrm{n}(\alpha) \cap \mathrm{fn}(Q) = \emptyset)$$

$$\text{RES}: \quad \frac{P \xrightarrow{\alpha} P'}{(y)P \xrightarrow{\alpha} (y)P'} \qquad (y \notin \mathrm{n}(\alpha))$$

$$\text{OPEN}: \quad \frac{P \xrightarrow{\overline{x}y} P'}{(y)P \xrightarrow{\overline{x}(y)} P'} \qquad (x \neq y)$$

$$\text{STRUCT}: \quad \frac{Q \equiv P \quad P \xrightarrow{\alpha} P' \quad P' \equiv Q'}{Q \xrightarrow{\alpha} Q'}$$

■

Comparing with 3.2, note that IN, OUT and OPEN are new – being responsible for generating the three new kinds of action – while PAR, RES and STRUCT are extended. Now, up to $\equiv$, we have recovered the relations defined in [12], and have faithfully recaptured $\rightarrow$ as $\xrightarrow{\tau}$:

5.2 Proposition

(1) If $\xrightarrow{\alpha}{}'$ are the relations defined in [12], and alpha-convertible terms are identified, then $\xrightarrow{\alpha}$ is identical with $\xrightarrow{\alpha}{}' \equiv$.

(2) $\xrightarrow{\tau}$ is identical with $\rightarrow$.

Proof In each case, a routine induction on the inference of an action from the rules. ■

Returning to the translation of λ-terms, we need to strengthen slightly the property of determinacy which we need for $[\![M]\!]u$:

5.3 Definition P is *determinate* if it is r-determinate and also, whenever $P \rightarrow^* Q \xrightarrow{\alpha}$, for $\alpha \neq \tau$, then $Q \not\xrightarrow{\tau}$. ■

5.4 Lemma $[\![M]\!]u$ is determinate.

Proof By inspection of the proof of Lemma 4.5. ∎

6 Observation precongruences

As is often done for process terms, we wish to define a notion of atomic observation of a π-term's behaviour, and then compare two terms according to the pattern of their observed behaviour. This typically yields a preorder or precongruence over terms. We only wish to go far enough to see to what extent our translation preserves the distinguishability of λ-terms under observation. To be precise, for what precongruence $\sqsubseteq$ on π-terms do we have

$$[\![M]\!]u \sqsubseteq [\![N]\!]u \text{ implies } M \lesssim N$$

or indeed its converse?

Since applicative simulation takes no account of the number of reductions taken to converge, the relevant precongruences over $\mathcal{P}$ will be those in which reduction $\rightarrow$ is not directly observable. Thus we are interested in actions $\alpha \neq \tau$ as atomic observations. This motivates the following:

6.1 Definition $\stackrel{\alpha}{\Rightarrow} \stackrel{\text{def}}{=} \rightarrow^* \stackrel{\alpha}{\rightarrow}$. ∎

Then the weakest reasonable preorder and precongruence are as follows:

6.2 Definition

(1) $P \sqsubseteq Q$ if, for all $\alpha \neq \tau$, $P \stackrel{\alpha}{\Rightarrow}$ implies $Q \stackrel{\alpha}{\Rightarrow}$.

(2) $P \sqsubseteq Q$ if, for all contexts $\mathcal{C}[\_]$, $\mathcal{C}[P] \sqsubseteq \mathcal{C}[Q]$. ∎

It is worth noting that although the *preorder* $\sqsubseteq$ is obviously weakened by restricting the range of α, the *precongruence* $\sqsubseteq$ remains unchanged even if we restrict the range of α to a singleton, which could be called 'termination'. But we are not primarily concerned with this refinement here.

Now, we can show that even this weak congruence over $\mathcal{P}$ is strictly stronger than $\lesssim$ as far as λ-terms are concerned:

6.3 Theorem (Lazy precongruence) Let $L_1, L_2 \in \mathcal{L}^0$. Then

(1) $[\![L_1]\!]u \sqsubseteq [\![L_2]\!]u$ implies $L_1 \lesssim L_2$

(2) $L_1 \lesssim L_2$ does not imply $[\![L_1]\!]u \sqsubseteq [\![L_2]\!]u$

Proof For (1), assume the hypothesis and let $L_1 M_1 \cdots M_n \downarrow$, say

$$L_1 M_1 \cdots M_n \to^* \lambda y L_1'$$

Now

$$[\![L_1 M_1 \cdots M_n]\!]u \equiv (\tilde{v}) \Big([\![L_1]\!]v_1 \mid \overline{v}_1(y_1)v_2.[\![y_1 := M_1]\!] \mid \cdots \mid \overline{v}_n(y_n)u.[\![y_n := M_n]\!] \Big)$$

Let this be $C[[\![L_1]\!]v_1]$. Then by Theorem 4.6 we have

$$C[[\![L_1]\!]v_1] \to^* (\tilde{x})\Big([\![\lambda y L_1'']\!]u \mid [\![\tilde{x} := \widetilde{N}]\!]\Big) \not\to$$

(assuming $y \notin \tilde{x}$) where $\lambda y L_1' \equiv (\lambda y L_1'')\{\tilde{N}/\tilde{x}\}$. Hence $C[[\![L_1]\!]v_1] \overset{u(x)}{\Rightarrow}$, so by the hypothesis $C[[\![L_2]\!]v_1] \overset{u(x)}{\Rightarrow}$ also. So

$$[\![L_2 M_1 \cdots M_n]\!]u \equiv C[[\![L_2]\!]v_1] \to^* P \not\to$$

by Lemma 5.4, since $P \overset{u(x)}{\to}$. Hence by Theorem 4.6 again $L_2 M_1 \cdots M_n \downarrow$ as required.

For (2), we adapt a counterexample of Ong [14], which strengthens Abramsky's result that his canonical model of the lazy λ-calculus is not fully abstract [1]. Ong defines

$$L_1 \overset{\text{def}}{=} x(\lambda y\, x \Xi \Omega y)\Xi$$
$$L_2 \overset{\text{def}}{=} x(x \Xi \Omega)\Xi$$

where $\Xi \widetilde{N} \downarrow$ for all $\widetilde{N}$; for example, take $\Xi \overset{\text{def}}{=} \mathbf{YK}$. (Recall that $\Omega \uparrow$.)

He shows that $L_1 \lesssim L_2$. On the other hand, he shows that $L_1\{^{\mathbf{c}}/_x\}\downarrow$ and $L_2\{^{\mathbf{c}}/_x\}\uparrow$, where $\mathbf{c}$ is a *new* combinator – not definable in lazy λ-calculus – such that

$$\begin{aligned} \mathbf{c}M\downarrow\mathbf{I} \quad &\text{if} \quad M\downarrow \\ \mathbf{c}M\uparrow \quad &\text{if} \quad M\uparrow \end{aligned}$$

Now we can interpret $\mathbf{c}$ (*convergence-testing*) in π-calculus thus:

$$[\![\mathbf{c}]\!]u \stackrel{\text{def}}{=} u(x)(v).\overline{x}(w).\overline{w}(y).[\![\mathbf{I}]\!]v$$

and then show

$$C[[\![L_1]\!]u] \stackrel{u(x)}{\Rightarrow} , C[[\![L_2]\!]u] \stackrel{u(x)}{\not\Rightarrow}$$

where $C[\_\_]$ is the context $(x)(\_\_ \mid [\![x{:=}\mathbf{c}]\!])$. Hence $[\![L_1]\!]u \not\preceq [\![L_2]\!]u$. ■

7 The call-by-value λ-calculus

We shall now, more briefly, repeat our programme for the call-by-value λ-calculus [15], where reduction in $\mathcal{L}^0$ may only occur when the argument is an abstraction. The terms $\mathcal{L}$ are as before, and it is also convenient to define the *values* $\mathcal{V}$ by

$$V \quad ::= \quad x \quad \mid \quad \lambda x M$$

We shall let U, V, W range over $\mathcal{V}$.

7.1 Definition The *call-by-value reduction relation* $\rightarrow_{\mathrm{v}}$ is the smallest which satisfies the rules

$$\beta_{\mathrm{v}}: \quad (\lambda x M)V \rightarrow_{\mathrm{v}} M\{^V/_x\}$$

$$\text{APPL}: \quad \frac{M \rightarrow_{\mathrm{v}} M'}{MN \rightarrow_{\mathrm{v}} M'N}$$

$$\text{APPR}: \quad \frac{N \rightarrow_{\mathrm{v}} N'}{MN \rightarrow_{\mathrm{v}} MN'}$$

■

Reduction $\rightarrow_v$ is no longer determinate, but it is well-known that convergence $\downarrow_v$ is; if $M \downarrow_v M'$ then M' is unique. Moreover, convergence is *strong*: if $M \downarrow_v M'$ then all reduction sequences are finite. (The definitions of $\downarrow_v$ and $\uparrow_v$ are analogous with those for the lazy calculus.)

The corresponding applicative simulation relation $\lesssim_v$ is a precongruence[3], and $(\lambda x M)V \cong_v M\{^V/_x\}$.

7.2 Fact $\lesssim$ and $\lesssim_v$ are incomparable.

Proof $\mathbf{I} \lesssim \mathbf{KI}\Omega \not\lesssim \Omega$, and $\mathbf{I} \not\lesssim_v \mathbf{KI}\Omega \lesssim_v \Omega$. ∎

We now turn to encoding in π-calculus. For the rest of this section, for legibility, we shall omit the subscript 'v' from relational symbols, and also from our translation function $[\![\_\_]\!]_v$ (though it differs from $[\![\_\_]\!]$ for the lazy calculus). We shall continue to let x, y, z range over $\mathcal{X}$ and now let p, q, r, u, v, w range over $\mathcal{N} - \mathcal{X}$. In our new encoding $[\![M]\!]p$, the name p will have a different significance. The reason is that two 'events' which coincided for the lazy calculus must now be separated, namely

- the signal at p that M has reduced to a value (needed when M is the *argument* of an application);
- the receipt of arguments by an abstraction M (needed when M is *applied*).

Further, our 'environment entries' will now contain only values. So we begin by defining $[\![y := V]\!]$:

$$[\![y := \lambda x M]\!] \stackrel{\text{def}}{=} !\, y(w).w(x)(p).[\![M]\!]p$$
$$[\![y := x]\!] \stackrel{\text{def}}{=} !\, y(w).\overline{x}w$$

Now the first action of a (translated) value, $[\![V]\!]p$, must be to announce

[3] For this, we have to prove: If $M \lesssim_v N$ then $C[M]\downarrow_v$ implies $C[N]\downarrow_v$. Allen Stoughton has pointed out that there is a simple direct proof of the corresponding 'context lemma' for lazy λ-calculus, following Berry and Levy [2] or Milner [10]; the same holds (with a little more trouble) for the call-by-value case.

its valuehood, thus providing access to an 'environment entry'.[4] Note that $[\![y{:=}V]\!]$ is here a *subterm* of $[\![V]\!]p$; whereas the opposite was true in the lazy encoding. And, in contrast with the lazy calculus, the translation $[\![MN]\!]p$ of an application must allow M and N to 'run' in parallel:

$$\begin{aligned} [\![V]\!]p &\stackrel{\text{def}}{=} \overline{p}(y).[\![y{:=}V]\!] \qquad (y \text{ not free in } V) \\ [\![MN]\!]p &\stackrel{\text{def}}{=} (q)(r)\big(\mathbf{ap}(p,q,r) \mid [\![M]\!]q \mid [\![N]\!]r\big) \\ \mathbf{ap}(p,q,r) &\stackrel{\text{def}}{=} q(y).\overline{y}(v).r(z).\overline{v}zp \end{aligned}$$

We now define the property which we wish $[\![M]\!]p$ to possess, in place of determinacy.

7.3 Definition P is *weakly determinate* if, whenever $P \rightarrow^* Q$, then

(i) If $Q \rightarrow Q_1$ and $Q \rightarrow Q_2$, then either $Q_1 \equiv Q_2$, or $Q_1 \rightarrow Q'$ and $Q_2 \rightarrow Q'$ for some Q'.

(ii) If $Q \stackrel{\alpha}{\rightarrow}$ for $\alpha \neq \tau$, then $Q \not\rightarrow$. ∎

We now set up a relation very closely analogous to that for the lazy calculus in Definition 4.4, as the basis for our simulation:

7.4 Definition Let the relation $\mathcal{S} \subseteq \mathcal{L}^0 \times \mathcal{P}$ contain all pairs (L, P) such that for some $k \geq 0$, some $M \in \mathcal{L}$, distinct $U_1, \ldots, U_k \in \mathcal{V}$ and some $x_1, \ldots, x_k \in \mathcal{X}$:

(i) $\mathrm{fv}(M) \subseteq \tilde{x}$, and $\mathrm{fv}(U_i) \subseteq \{x_{i+1}, \ldots, x_k\}$ for all $1 \leq i \leq k$. (So, in particular, $\mathrm{fv}(U_k) = \emptyset$.)

[4] There is some doubt as to whether a variable x should be considered as a value. It turns out not to matter much; this is because our semantics is based upon *closed* terms. There is, in fact, an encoding which agrees with the idea that a variable is *not* a value; one defines $[\![x]\!]p \stackrel{\text{def}}{=} \overline{p}x$ instead of $[\![x]\!]p \stackrel{\text{def}}{=} \overline{p}(y).[\![y{:=}x]\!]$. This has an interesting effect, for those who like the fine detail of transition systems. In our present encoding, variables behave like buffers or indirect references, and long computations build up chains of indirection; in particular, the infinite reduction of Ω takes longer and longer over each cycle, while in the alternative encoding the cycle is of fixed length. We chose the less efficient version, because it is easier to outline the proof of simulation to follow.

(ii) $L \equiv M\{\tilde{U}/\tilde{x}\}$

(iii) $P \equiv (\tilde{x})\big([\![M]\!]u \mid [\![\tilde{x}:=\tilde{U}]\!]\big)$ ∎

7.5 Lemma For any $(L,P) \in \mathcal{S}$, P is weakly determinate, and one of the following conditions holds:

A. L is an abstraction, and for some $(L',P') \in \mathcal{S}$

$$L \equiv L' \text{ and } P \downarrow P'$$

B. For some $(L',P') \in \mathcal{S}$

$$L \to L' \text{ and } P \to^+ P'$$

Proof (outline) Let $(L,P) \in \mathcal{S}$, and let us use the notation of the definition. The weak determinacy of P will emerge during the proof that A or B holds.

Case 1: M is a value. Then clearly L is an abstraction, and $P \downarrow P$, so A holds with $(L',P') = (L,P)$.

Case 2: M has at least one subterm of the form $M_0 \equiv V N_0$ (V a value) which does not lie within an abstraction, and P has the corresponding unguarded subterm

$$[\![M_0]\!]p_0 \equiv (q)(r)\big(\mathbf{ap}(p_0,q,r) \mid [\![V]\!]q \mid [\![N_0]\!]r\big)$$

It is clear that all reductions of P arise from subterms of this kind, in which N_0 may or may not be a value. Now V must either be an abstraction $\lambda z M_0'$, or be associated via $[\![\tilde{x}:=\tilde{U}]\!]$ with such an abstraction through a chain of variables $V \equiv x_{i_1}$, $U_{i_1} \equiv x_{i_2}$, ..., $U_{i_j} \equiv \lambda z M_0'$. So, picking $y,z \notin \tilde{x}$, for some m there is a reduction

$$[\![M_0]\!]p_0 \to^m (y)\big((v)(r)\big(r(z).\overline{v}z p_0 \mid v(z)(p)[\![M_0']\!]p \mid [\![N_0]\!]r\big) \mid [\![y:=V]\!]\big) \quad (\#)$$

with no alternatives.[5] In fact these reductions – for all subterms like M_0 – can occur within P independently of each other (since accessing an environment entry does not change it).

[5] As in remark (4) after the example of reduction in Section 4, the new environment entry here could be garbage-collected, but we wish to respect $\equiv$ so we retain it.

Furthermore, for at least *one* such subterm M_0, N_0 must be a value W say (by an easy induction on the structure of terms). Now these cases $M_0 \equiv VW$ correspond precisely to the redexes of L, and for such a redex we have the reduction[6] in $\mathcal{L}$

$$L \;\to\; L' \stackrel{\text{def}}{=} M'\{V/y\}\{W/z\}\{\tilde{U}/\tilde{x}\} \qquad (*)$$

where M' results from M by writing M_0' in place of M_0.

Now recalling that $[\![N_0]\!]r \equiv [\![W]\!]r \equiv \overline{r}(z).[\![z{:=}W]\!]$, we continue the reduction from (#), yielding altogether

$$[\![M_0]\!]p_0 \;\to^{m+3}\; (y)(z)\Big([\![M_0']\!]p_0 \mid [\![y{:=}V]\!] \mid [\![z{:=}W]\!]\Big)$$

again with no alternatives. (And again, the reductions of this kind can occur independently within P.) This yields the following reduction for P:

$$P \;\to^{m+3}\; (y)(z)(\tilde{x})\Big([\![M']\!]p \mid [\![y{:=}V]\!] \mid [\![z{:=}W]\!] \mid [\![\tilde{x}{:=}\tilde{U}]\!]\Big) \qquad (**)$$

since $[\![M']\!]p$ results from $[\![M]\!]p$ by writing $[\![M_0']\!]p$ in place of $[\![M_0]\!]p_0$. Now, comparing $(*)$ with $(**)$, we have achieved condition B.

Finally, we have seen that no reduction within P ever preempts another; also it is clear that $\stackrel{\alpha}{\to}$ for $\alpha \neq \tau$ is only possible in Case 1 where P has converged. Thus P is weakly determinate. ∎

The encoding theorem follows just as for the lazy calculus. To avoid confusion we re-adopt our subscript 'v' in stating it:

7.6 Theorem (Call-by-value encoding) For all $L \in \mathcal{L}^0$, $[\![L]\!]_{\mathrm{v}}p$ is weakly determinate, and one of the following conditions holds:

A. $L{\downarrow_{\mathrm{v}}}V$ and $[\![L]\!]_{\mathrm{v}}p{\downarrow_{\mathrm{v}}}P$, where

$$V \equiv W\{\tilde{U}/\tilde{x}\} \qquad \text{and} \qquad P \equiv (\tilde{x})\Big([\![W]\!]_{\mathrm{v}}p \mid [\![\tilde{x}{:=}\tilde{U}]\!]_{\mathrm{v}}\Big)$$

B. $L{\uparrow_{\mathrm{v}}}$ and $[\![L]\!]_{\mathrm{v}}p{\uparrow_{\mathrm{v}}}$. ∎

We also find the same relationship of precongruences as in the lazy case:

[6] The substitution of V for y is irrelevant, since y is not free in M'; we insert the substitution to exhibit our simulation. (See also previous footnote.)

7.7 Theorem (Call-by-value precongruence) Let $L_1, L_2 \in \mathcal{L}^0$. Then

(1) $[\![L_1]\!]_{\mathrm{v}} p \sqsubseteq [\![L_2]\!]_{\mathrm{v}} p$ implies $L_1 \lesssim_{\mathrm{v}} L_2$

(2) $L_1 \lesssim_{\mathrm{v}} L_2$ does not imply $[\![L_1]\!]_{\mathrm{v}} p \sqsubseteq [\![L_2]\!]_{\mathrm{v}} p$

Proof For (1), the proof is just as in Theorem 6.3. For (2), consider the following:

$$\begin{aligned} L_1 &\stackrel{\text{def}}{=} \lambda x\big((x\mathbf{I})(x\mathbf{K})\big) \\ L_2 &\stackrel{\text{def}}{=} \lambda x\big((\lambda y\, y(x\mathbf{K}))(x\mathbf{I})\big) \\ L_3 &\stackrel{\text{def}}{=} \lambda x\big((\lambda y\, (x\mathbf{I})y)(x\mathbf{K})\big) \end{aligned}$$

They are all equivalent under $\eqsim_{\mathrm{v}}$. But in L_2, x will be applied to $\mathbf{I}$ first, while in L_3 it will be applied to $\mathbf{K}$ first. (In L_1 either may happen.) So in $\mathcal{P}$ we construct a fickle 'function' which behaves differently on successive calls; it will behave like $\mathbf{KI}$ the first time it is called, and like $\mathbf{I}$ the second time. When (the encodings of) L_2 and L_3 are 'applied' to the fickle function, the results will be respectively (the encodings of)

$$\begin{aligned} (\mathbf{KII})(\mathbf{IK}) &\eqsim_{\mathrm{v}} \mathbf{K} \\ (\mathbf{II})(\mathbf{KIK}) &\eqsim_{\mathrm{v}} \mathbf{I} \end{aligned}$$

In fact, we define

$$\mathbf{fickle}(r) \stackrel{\text{def}}{=} \overline{r}(y).y(u).\big(u(x)(p).[\![\mathbf{KI}]\!]_{\mathrm{v}} p \mid y(v).v(x)(p).[\![\mathbf{I}]\!]_{\mathrm{v}} p\big)$$

and place each $[\![L_i]\!]q$ in the context

$$(q)(r)\big(\mathbf{ap}(p,q,r) \mid \_\_ \mid \mathbf{fickle}(r)\big) \qquad \blacksquare$$

8 Conclusion

We have only begun here to explore the treatment of functions in the π-calculus; the reader will already have posed many questions. For example: Exactly what is the preorder $\sqsubseteq$ induced upon λ-terms by

$$L \sqsubseteq M \stackrel{\text{def}}{=} [\![M]\!]u \sqsubseteq [\![M]\!]u \qquad ?$$

And are other reduction strategies easy to encode?

On the latter point, close examination of strategies reveals what may be called an oddity, seen in the light of the process paradigm. Consider any strategy in which all the rules β, APPL and APPR hold. Suppose that $M[x,x]$ is a term in which x occurs twice not within an abstraction, and suppose

$$N \equiv N_1 \to N_2 \to \cdots \to N_k \to \cdots$$

Then of course

$$(\lambda x M)N \to \cdots \to (\lambda x M)N_k \to M[N_k, N_k] \to^* M[N_{k+i}, N_{k+j}] \to \cdots$$

For the first k steps, N's reduction is 'shared'; thereafter, two separate reductions of N_k can continue within M, at different speeds (and in different directions too). This familiar situation looks odd if N is modelled as an agent; why should it clone into two or more copies just because *access* to N is transmitted through x? (Of course, it is reasonable for N to clone when it is eventually *applied* to two or more different arguments within M.) Naturally, the strategies which have been most deeply studied are those most easily expressed using the *textual substitution* which is basic to λ-calculus. One effect of providing π-calculus as a substrate may be to intensify the study of other strategies, such as those with shared reductions.

A question which should be pursued is: How badly do we need more than the fragment of π-calculus used here? This fragment, mainly because summation is absent, has yielded pleasantly to our new formulation. If summation is not needed for these encodings, when *is* it needed? In [12], it was used to encode computations over structured data types. But closer inspection suggests that it is not essential for this purpose. Without digressing too far, we can show how we may do without it. As in [12], we can encode truth-values thus:

$$\begin{aligned} \mathrm{t}(x) &\stackrel{\text{def}}{=} x(u)(v).u \\ \mathrm{f}(x) &\stackrel{\text{def}}{=} x(u)(v).v \end{aligned}$$

where the final u is short for $u(z)$. (Note the similarity with a standard encoding in λ-calculus. A more useful encoding may also use replication.) Then an agent P which wishes to use the truth value at x to choose its

future path may take the form

$$P \equiv \overline{x}(u)(v).(\overline{u}.P_1 \mid \overline{v}.P_2)$$

where again $\overline{u}.$ is short for $\overline{u}(z)$. For it turns out that up to strong bisimilarity

$$(x)(P \mid \mathbf{t}(x)) \rightarrow^3 P_1 \qquad \text{and} \qquad (x)(P \mid \mathbf{f}(x)) \rightarrow^3 P_2$$

In fact, the use of $\mid$ in P, in place of $+$ as suggested in [12], yields what is needed.

One use of summation has been to provide normal forms, and thence complete axiomatisations, for agents under various congruences [9,11]. At the same time, a price has been paid in that the congruences themselves are harder to define in the presence of summation. We leave the importance of summation as an open question.

As far as application is concerned, we hope that the results of this paper will throw some light on the semantics of programming languages which contain both concurrency and non-trivial use of procedures or functions.

References

[1] Abramsky, S., *The Lazy Lambda Calculus*, to appear in **Declarative Programming**, ed. D. Turner, Addison Wesley, 1988.

[2] Berry, G., *Modèles Complètement Adéquats et Stables des lambda-calcul typés*, Thèse se Doctorat d'Etat, Université Paris VII, 1979.

[3] Berry, G. and Boudol, G., *The Chemical Abstract Machine*, to appear in Proc 17th Annual Symposium on Principles of Programming Languages, 1990.

[4] Boudol, G., *Towards a Lambda-Calculus for Concurrent and Communicating Systems*, Proc TAPSOFT 1989, Lecture Notes in Computer Science 351, Springer-Verlag, pp149–161, 1989.

[5] Curry, H.B. and Feys, R., **Combinatory Logic, Vol 1**, North Holland, 1958.

[6] Clinger, W.D., *Foundations of Actor Semantics*, AI-TR-633, MIT Artificial Intelligence Laboratory, 1981.

[7] Engberg, U. and Nielsen, M., *A Calculus of Communicating Systems with Label-passing*, Report DAIMI PB–208, Computer Science Department, University of Aarhus, 1986.

[8] Girard, J.-Y., *Linear Logic*, Journal of Theoretical Science, Vol 50, pp111–102, 1987.

[9] Hennessy, M. and Milner, R., *Algebraic Laws for Non-determinism and Concurrency*, Journal of ACM, Vol 32, pp137–161, 1985.

[10] Milner, R., *Fully Abstract Models of Typed Lambda-calculi*, Journal of Theoretical Science, Vol 5, pp1–23, 1977.

[11] Milner, R., **Communication and Concurrency**, Prentice Hall, 1989.

[12] Milner, R., Parrow, J.G. and Walker, D.J., *A Calculus of Mobile Processes, Parts I and II*, Report ECS-LFCS-89-85 and -86, Laboratory for Foundations of Computer Science, Computer Science Department, Edinburgh University, 1989.

[13] Nielsen, F., *The Typed λ-calculus with First-class Processes*, Report ID-TR:1988-43, Inst. for Datateknik, Tekniske Hojskole, Lyngby, Denmark, 1988.

[14] Ong, C-H.L., *Fully Abstract Models of the Lazy Lambda Calculus*, Proc 29th Symposium on Foundations of Computer Science, pp368–376, 1988.

[15] Plotkin, G.D., *Call-by-name and Call-by-value and the λ-calculus*, Journal of Theoretical Science, Vol 1, pp125–159, 1975.

[16] Thomsen, B., *A Calculus of Higher-order Communicating Systems*, Proc 16th Annual Symposium on Principles of Programming Languages, pp143–154, 1989.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique