THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

Functions as Processes

Citation for published version:
Milner, R 1992, 'Functions as Processes' Mathematical Structures in Computer Science, vol. 2, no. 2, pp. 119-141. DOI: 10.1017/S0960129500001407

Digital Object Identifier (DOI):
10.1017/S0960129500001407

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Publisher's PDF, also known as Version of record

Published In:
Mathematical Structures in Computer Science

Publisher Rights Statement:
Copyright © Cambridge University Press 1992. Reproduced with permission.

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 05. Apr. 2019

Math. Struct. in Comp. Science (1992), *vol.* 2, *pp.* 119–141

Functions as processes

ROBIN MILNER

University of Edinburgh

Received 15 May 1991; revised 20 September 1991

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 (Milner *et al.* 1989) 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 bisimulation* (Abramsky 1989) 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 and Boudol's Chemical Abstract Machine (Berry and Boudol 1990). The new formulation is shown to be consistent with the original.

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 (Milner *et al.* 1989) 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. Thus, in claiming to model 'functions as processes', we consider functions in the way that the founders of λ-calculus considered them: as computational descriptions rather than as abstract objects (see Barendregt (1981), the Introduction). It remains a challenge for the future to extend the correspondence to one which relates functions and concurrent processes as abstract objects. However, we make a step in that direction. For the encoding of lazy λ-calculus, we compare Abramsky's precongruence over λ-terms, which he called *applicative bisimulation* (Abramsky 1989), with that which is induced by the encoding; we also make a similar comparison for the call-by-value λ-calculus.

Recently, several generalizations of process calculus to embrace higher-order objects have been proposed; examples are by Boudol (1989), Nielson (1989) and the Calculus of Higher-order Communicating Systems (CHOCS) of Thomsen (1989, 1990). Explicitly or implicitly, they have enabled embedding of the λ-calculus. In particular, Thomsen shows that if M and N are λ-terms such that $\lambda \vdash M = N$ in the sense of Barendregt (1981),

then their translations into CHOCS are equivalent in a natural sense. These higher-order process calculi share with the λ-calculus the notion that a higher-order entity (function or process) can be transmitted as a value and bound to a variable; this common feature is naturally employed in embedding the λ-calculus in the process calculus.

The π-calculus, which builds upon the calculus ECCS of Engberg and Nielsen (1986), differs from higher-order calculi in the following respect: it replaces the notion that an agent can be transmitted as a value and bound to a variable, which we may call the *function* paradigm, by a significantly different notion, which we may call the *object* paradigm. There is a connection here with object-oriented programming; it lies in the idea of objects with independent state interacting with one another.† (Walker (1991) has recently shown that the semantics of (at least) simple object-oriented languages can be given by a quite direct translation into the π-calculus.) In the object 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.

The two paradigms seem equally basic and significant. Without arguing naïvely *in favour* of the object paradigm, our aim in the π-calculus is to present it in undiluted form. Since higher-order parameters are sacrificed, a succinct translation between the paradigms is by no means preordained; but we know 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 as assemblies of objects.

Both the π-calculus and higher-order process calculi can be seen as formalizations of Hewitt's Actor model (Hewitt *et al.* 1973, Clinger 1981) first put forward eighteen years ago, and arguably a forerunner of object-oriented programming. Though not a formal calculus, the Actor model pioneered the idea that functions and even humble data values can be modelled as processes. However, the Actor model leans towards the notion of passing higher-order objects in communication, rather than passing *access* to them; to quote from Hewitt (1977), "Actors make use of one universal communication mechanism called *actor transmission*, which consists of sending one actor (called the *messenger* of the transmission) to another actor (called the *target* of the transmission)".

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 and Boudol's Chemical Abstract Machine (Berry and Boudol 1990). 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;

† Other important ingredients of object-oriented programming, such as the notions of inheritance and subtyping, are not reflected in our object paradigm.

no knowledge of our original presentation of π-calculus Milner *et al.* (1989) is needed. However, we also give results which state that the new presentation is fully consistent with the original.

The valuable suggestions of an anonymous referee are acknowledged with gratitude.

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

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

Informally, $\bar{x}y.P$ 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 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

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

and we expect the result of interaction to be $P\{z/y\}|Q$. This differs from β-reduction $(\lambda xM)N \to M\{N/x\}$ in one essential: the 'sender' $\bar{x}z.Q$ 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 within

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

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

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

to hold for some congruence relation $\approx$.

To summarize: for this paper, the syntax of $\mathscr{P}$ is

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

There are a few differences from the π-calculus given in Milner *et al.* (1989). Only one is a novelty: the presence of $!P$. Replication 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). With a few more rules one can even handle full summation in the present formulation; see also the method adopted by Berry and Boudol in the Chemical Abstract Machine (Berry and Boudol 1990).

Some technical details and terminology:

— Lowest syntactic precedence is assigned to |, so for example $x(y).P \mid Q$ means $(x(y).P) \mid Q$.

— 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, $\mathrm{bn}(P)$ for the bound 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 $\bar{x}y$.

— 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 $\vec{x}$ to mean a sequence $x_1,\ldots,x_n$ of names; similarly $\vec{P}$ for a sequence of terms. Without risk of confusion, we treat $\vec{x}$ sometimes as a set. We also write $(\vec{x})$ for the multiple restriction $(x_1)\cdots(x_n)$.

— We shall use several convenient abbreviations:

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

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

3. The π-calculus: Equations and Reductions

Our operational intuition is simple: if term R contains two unguarded subterms $x(y).P$ and $\bar{x}z.Q$, 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 $x(y).P \mid \bar{x}z_1.Q_1 \mid \bar{x}z_2.Q_2$. There are two reductions

$$R \rightarrow P\{z_1/y\} \mid Q_1 \mid \bar{x}z_2.Q_2$$
$$R \rightarrow P\{z_2/y\} \mid \bar{x}z_1.Q_1 \mid Q_2$$

(2) Let R be $w(x).(x(y).P \mid \overline{x}z.Q)$. 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 Berry and Boudol (1990), 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.

Definition 3.1. The *structural congruence* relation, written $\equiv$, is the smallest congruence over $\mathscr{P}$ 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.

Definition 3.2. The *reduction* relation over $\mathscr{P}$, 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:

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

The present formulation of π-calculus differs strikingly from that in Milner *et al.* (1989). The essential difference is in the use of structural congruence. This is inspired by the Chemical Abstract Machine of Berry and Boudol (1990). Their insight is that the rules of structured operational semantics, as used in Milner (1989) or Milner *et al.* (1989) 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 Milner (1989) – 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 characterized in the behavioural congruence. (For example, observation congruence in Milner (1989) 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; 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 Milner *et al.* (1989) a labelled transition system is defined, which we shall here denote $\stackrel{\alpha}{\rightarrow}'$, where the *actions* α are of four kinds:

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

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; it follows from Propositions 5.2 and 5.3 below that $\rightarrow$ is identical with $\stackrel{\tau}{\rightarrow}' \equiv$.

For the next section we shall need the following:

Definition 3.4. 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* $\mathscr{X}$ be an infinite and co-infinite subset of $\mathscr{N}$. For this section, we shall let x, y, z range over $\mathscr{X}$, and u, v, w range over $\mathscr{N} - \mathscr{X}$. 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* $\mathscr{L}$ 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 use $\equiv$ upon $\mathscr{L}$ to mean syntactic equality up to α-conversion. We shall frequently need the sequential composition of several substitutions (note: *not* simultaneous substitution); so, understanding the members of $\vec{x}$ to be distinct, we introduce the abbreviation

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

We shall use the standard terms

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

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:

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

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

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.

Definition 4.2. 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 $x N_1 \cdots N_n$ $(n \geq 0)$. Writing $\mathscr{L}^0$ for the *closed* terms, if $M \in \mathscr{L}^0$ then $M' \in \mathscr{L}^0$ also, so M' must be an abstraction.

Abramsky (1989) defines an important preorder $\lesssim$, which we shall call *applicative simulation*, as follows:

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

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

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

(Abramsky called $\lesssim$ applicative *bi*-simulation; but it seems more consistent with previous usage to reserve the prefix *bi* for the induced equivalence relation.) 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 Abramsky (1989) that $(\lambda xM)N \eqsim M\{N/x\}$, and that $\lesssim$ is a precongruence. The latter follows from the result that, defining a *context* $\mathscr{C}[\_]$ to be a term with a single hole,

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

Allen Stoughton (Stoughton 1989) has pointed out that this "context lemma" has a simple direct proof, along the same lines as (but much simpler than) those proved by Berry and Levy (Berry 1979) or Milner (1977).

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). Several different means have been found to break this operation into smaller parts:

— In combinatory algebra (Curry and Feys 1958), Curry found combinators which progressively distribute the argument of an abstraction λxM to those parts of the body M which will use it (thus, in fact, eliminating variables altogether).

— 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. Our encoding below can be seen as a formalisation of this idea, whose first appearance seems to have been in Landin (1964).

— Much more recently Abadi *et al.* (1990) have introduced the $\lambda\sigma$-calculus, with explicit substitutions. Here, substitutions are syntactic objects distinct from terms; moreover if M is a term and s a substitution, then $M\{s\}$ *is* a term – it does not merely denote the result of applying s to M. (I have adapted the notation of Abadi *et al.* (1990) for comparative purposes.) This clearly has an affinity with the use of environments; but it differs in (for example) having explicit reduction rules which distribute substitutions over terms. It also has affinity with Curry's combinators – via this idea of distribution over terms; again it differs, because substitutions are a distinct syntax class with their own rules. (As the authors say, this remains closer to the use of variables in λ-calculus.)

We now turn to the encoding of $\mathscr{L}$ into $\mathscr{P}$. Each $M \in \mathscr{L}$ is encoded as $[\![M]\!]$, a map from names to $\mathscr{P}$. 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. Recall that, by our convention, u is not a variable of λ-calculus.

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

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

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 (Girard 1987) should be explored; his notation for it has been chosen here deliberately.

How does $[\![\lambda xM]\!]u$ 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:

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

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):

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

The following remarks will help in reading the above calculation:

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

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

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

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

— The last step, to (7), is the only one which goes beyond $\equiv$; it is a simple case of *strong bisimilarity* – see Milner *et al.* (1989) – 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).

Our example, up to step (5), also illustrates that in general

$$[\![(\lambda xM)N]\!]u \quad \to^* \quad (x)([\![M]\!]u \mid [\![x := N]\!])$$

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 $\mathscr{P}$ 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

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

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

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

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

(ii) $L \equiv M\{^{\vec{N}}/_{\vec{x}}\}$

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

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

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

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

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

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

Proof. Let $(L, P) \in \mathscr{S}$, 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

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

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

Case 2: $n > 0$ and M_0 is an abstraction $\lambda x_0 M_0'$; without loss of generality we can assume $x_0 \notin \vec{x}$. Then $L \to L'$, where

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

On the other hand, we have

$$[\![M]\!]u \equiv (\vec{v})\left(\begin{array}{ll} [\![M_0]\!]v_1 & \mid \ \bar{v}_1(y_1)v_2.[\![y_1 := M_1]\!] \\ & \mid \ \bar{v}_2(y_2)v_3.[\![y_2 := M_2]\!] \end{array} \right.$$

$$\cdots$$
$$\mid \ \overline{v}_n(y_n)u.[\![y_n := M_n]\!] \ \Big)$$

where $(\vec{v}) = (v_1)\cdots(v_n)$; 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, the effect of which is to replace $[\![M]\!]u$ in P by

$$(x_0)(v_2)\cdots(v_n)\Big(\ [\![M_0']\!]v_2 \ \mid \ [\![x_0 := M_1]\!]$$
$$\mid \ \overline{v}_2(y_2)v_3.[\![y_2 := M_2]\!]$$
$$\cdots$$
$$\mid \ \overline{v}_n(y_n)u.[\![y_n := M_n]\!] \ \Big)$$

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 $(L', P') \in \mathscr{S}$, satisfying condition B.

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

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

while within P we have

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

(which is just $\overline{x_i}u$ 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 $\overline{x_i}v_1$, thus replacing $[\![M]\!]u$ in P by the term

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

Now this is exactly $[\![M']\!]u$, where M' is $N_i M_1 \cdots M_n$. So we have shown that $P \to P''$ with $(L, P'') \in \mathscr{S}$. But N_i must be of the form $M_0' \cdots M_m'$, M_0', being either an abstraction or a variable x_j, $j > i$; so by Case 1 or 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 $(L', P') \in \mathscr{S}$ for some L'; so P is determinate. □

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

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

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

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

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

5. The π-calculus: Actions

The reduction relation $\rightarrow$ 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

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

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

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

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

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

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

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

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

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

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

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

$$\text{IN}: \qquad x(y).P \xrightarrow{x(y)} P$$
$$\text{OUT}: \qquad \bar{x}y.P \xrightarrow{\bar{x}y} P$$
$$\text{COM}: \qquad x(y).P \mid \bar{x}z.Q \xrightarrow{\tau} P\{z/y\} \mid Q$$

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

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

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

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

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, we first record the fact that we have faithfully recaptured our reduction relation in the form of $\xrightarrow{\tau}$:

Proposition 5.2. $\xrightarrow{\tau}$ is identical with $\rightarrow$.

Proof. A routine induction. □

Having defined a labelled transition system $\xrightarrow{\alpha}$, we should compare it with that which was used in Milner *et al.* (1989) to define the meaning of the π-calculus. For this purpose we suppose that we add to the rules of Milner *et al.* (1989) (Table 2, Part II) the following rule for replication:

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

Then indeed the two systems agree up to $\equiv$, as follows:

Proposition 5.3. If $\xrightarrow{\alpha}{}'$ are the relations defined in Milner *et al.* (1989) with the addition of REP, and alpha-convertible terms are identified, then $\xrightarrow{\alpha}$ is identical with $\xrightarrow{\alpha}{}' \equiv$.

Proof. A routine induction. □

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

Definition 5.4. P is *determinate* if it is r-determinate and also, whenever $P \rightarrow^* Q \xrightarrow{\alpha}$, for $\alpha \neq \tau$, then $Q \not\rightarrow$.

Lemma 5.5. $[\![M]\!]u$ 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 $\mathscr{P}$ 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:

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

Then the weakest reasonable preorder and precongruence are as follows:

Definition 6.2. (1) $P \sqsubseteq Q$ if, for all $\alpha \neq \tau$, $P \overset{\alpha}{\Rightarrow}$ implies $Q \overset{\alpha}{\Rightarrow}$.

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

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 $\mathscr{P}$ is strictly stronger than $\lesssim$ as far as λ-terms are concerned:

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

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

Now

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

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

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

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

$$[\![L_2 M_1 \cdots M_n]\!]u \equiv \mathscr{C}[[\![L_2]\!]v_1] \rightarrow^* P \not\rightarrow$$

by Lemma 5.5, since $P \overset{u(y)}{\rightarrow}$. Hence, by Theorem 4.6 again, $L_2 M_1 \cdots M_n \downarrow$ as required.

For (2), we adapt a counter-example of Ong (1988), which strengthens Abramsky's result that his canonical model of the lazy λ-calculus is not fully abstract (Abramsky

1989). Ong defines

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

where $\Xi\vec{N}\downarrow$ for all $\vec{N}$; for example, take $\Xi \stackrel{\text{def}}{=} Y\mathbf{K}$. (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

$$\mathbf{c}M\downarrow\mathbf{I} \quad \text{if} \quad M\downarrow$$
$$\mathbf{c}M\uparrow \quad \text{if} \quad M\uparrow$$

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

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

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

There are even simpler examples which show that preorder is not preserved by translation. For example, the terms xx and $x(\lambda y\, xy)$ are indistinguishable by $\lesssim$, whereas their translations can be distinguished by $\sqsubseteq$. The counter-example used in the proof has special interest since it shows how convergence-testing can be defined in π-calculus.

7. The call-by-value λ-calculus

We shall now, more briefly, repeat our programme for a version of the call-by-value λ-calculus of Plotkin (1975), where reduction in $\mathscr{L}^0$ may only occur when the argument is an abstraction. The terms $\mathscr{L}$ are as before, and it is also convenient to define the *values* $\mathscr{V}$ by

$$V \quad ::= \quad x \quad | \quad \lambda xM$$

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

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

$$\beta_{\mathrm{v}}: \qquad (\lambda xM)V \rightarrow_{\mathrm{v}} M\{V/x\}$$

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

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

We call $\rightarrow_{\mathrm{v}}$ *weak* because unlike Plotkin (1975) we preclude reduction inside an abstraction (i.e. we omit the ξ-rule).

Reduction $\rightarrow_v$ is no longer determinate since we allow parallel reduction of both components of a combination, but convergence $\downarrow_v$ is determinate; if $M \downarrow_v M'$ then M' is unique. Moreover, convergence is *strong*: if $M \downarrow_v M'$, all reduction sequences are finite. (The definitions of $\downarrow_v$ and $\uparrow_v$ are analogous with those for the lazy calculus.)

As for $\lesssim$ in the lazy calculus, it turns out that the corresponding applicative simulation relation $\lesssim_v$ is a precongruence. This has been proved by Ong (1990), again by means of a 'context lemma'; but in contrast to the lazy case the proof is unexpectedly hard. (I am grateful to Luke Ong for pointing out the mistake in my own attempt at a simple proof.) The expected form of β-conversion also holds, i.e $(\lambda xM)V \cong_v M\{^V/_x\}$.

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

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

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 $\mathscr{X}$ and now let p, q, r, u, v, w range over $\mathscr{N}-\mathscr{X}$. 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 $[\![y := V]\!]$:

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

Now the first action of a (translated) value, $[\![V]\!]p$, must be to announce its valuehood, thus providing access to an 'environment entry'. 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:

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

As an example, let us look at the reduction in π-calculus which corresponds to the β_v rule, $(\lambda zM)V \rightarrow_v M\{^V/_z\}$. Let us assume that z is not free in V, and that y is a new variable.

$$[\![(\lambda zM)V]\!]p \equiv (q)(r)\Big(\mathbf{ap}(p, q, r) \mid \overline{q}y.[\![y := \lambda zM]\!] \mid \overline{r}(z).[\![z := V]\!]\Big)$$
$$\rightarrow (r)(y)\Big(\overline{y}(v).r(z).\overline{v}zp \mid [\![y := \lambda zM]\!] \mid \overline{r}(z).[\![z := V]\!]\Big)$$

$$\begin{aligned}
&\to\ (r)(y)(v)\Big(r(z).\bar{v}zp \mid [\![y := \lambda z M]\!] \mid v(z)(p).[\![M]\!]p \mid \bar{r}(z).[\![z := V]\!]\Big)\\
&\to\ (y)(v)(z)\Big(\bar{v}zp \mid [\![y := \lambda z M]\!] \mid v(z)(p).[\![M]\!]p \mid [\![z := V]\!]\Big)\\
&\to\ (y)(z)\Big([\![y := \lambda z M]\!] \mid [\![M]\!]p \mid [\![z := V]\!]\Big)\\
&\sim\ (z)\Big([\![M]\!]p \mid [\![z := V]\!]\Big)
\end{aligned}$$

In the last step we have garbage-collected an 'environment-entry' which is no longer accessible, just as in the example we gave for lazy reduction in Section 4.3. The reader should note that the first two steps occur just because the operator $\lambda z M$ of the combination is a value, and do not depend upon the operand being a value; the last two steps will occur only when the operand has become a value (V in this case).

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

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

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

(ii) If $Q \overset{\alpha}{\to}$ for $\alpha \neq \tau$, then $Q \not\to$.

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:

Definition 7.4. 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 $M \in \mathcal{L}$, some $U_1, \ldots, U_k \in \mathcal{V}$ and distinct $x_1, \ldots, x_k \in \mathcal{X}$:

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

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

(iii) $P \equiv (\vec{x})\Big([\![M]\!]u \mid [\![\vec{x} := \vec{U}]\!]\Big)$

Lemma 7.5. 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 $[\![\vec{x} := \vec{U}]\!]$ with such an abstraction through a chain of variables $V \equiv x_{i_1}$, $U_{i_1} \equiv x_{i_2}$,

$\ldots$, $U_{i_j} \equiv \lambda z M_0'$. So, picking $y, z \notin \vec{x}$, there is a reduction of P (of some length m) which records the interaction between $[\![M_0]\!]p_0$ and the environment $[\![\vec{x} := \vec{U}]\!]$ in traversing this chain. We shall write it as a reduction of $[\![M_0]\!]p_0$ alone, since accessing an environment does not change it:

$$[\![M_0]\!]p_0 \rightarrow^m (y)\Big((v)(r)\Big(r(z).\bar{v}z p_0 \mid v(z)(p).[\![M_0']\!]p \mid [\![N_0]\!]r\Big) \mid [\![y := V]\!]\Big) \qquad (\#)$$

(The new environment entry here could be garbage-collected, as in the last remark after the example of reduction in Section 4, but we wish to respect $\equiv$ so we retain it.) This reduction of $[\![M_0]\!]p_0$ is determinate, except for inner reductions of $[\![N_0]\!]r$ of the same nature. It can be seen that all reductions of P are of this nature; moreover, they are independent of one another since accessing the environment does not change 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 in $\mathscr{L}$

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

where M' results from M by writing M_0' in place of M_0. (The substitution of V for y is irrelevant, since y is not free in M'; we insert the substitution to exhibit our simulation.)

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

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

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

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

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 pre-empts another; also it is clear that $\stackrel{\alpha}{\rightarrow}$ 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:

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

A. $L\downarrow_v V$ and $[\![L]\!]_v p \downarrow P$, where

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

B. $L\uparrow_v$ and $[\![L]\!]_v p \uparrow$.

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

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

(1) $[\![L_1]\!]_v p \sqsubseteq [\![L_2]\!]_v p$ implies $L_1 \lesssim_v L_2$

(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:

$$L_1 \stackrel{\text{def}}{=} \lambda x\Big((x\mathbf{I})(x\mathbf{K})\Big)$$
$$L_2 \stackrel{\text{def}}{=} \lambda x\Big((\lambda y\, y(x\mathbf{K}))(x\mathbf{I})\Big)$$
$$L_3 \stackrel{\text{def}}{=} \lambda x\Big((\lambda y\,(x\mathbf{I})y)(x\mathbf{K})\Big)$$

They are all equivalent under $\cong_{\mathrm{v}}$. 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 $\mathscr{P}$ 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)

$$(\mathbf{KI}\,\mathbf{I})(\mathbf{IK}) \quad \cong_{\mathrm{v}} \quad \mathbf{K}$$
$$(\mathbf{I}\,\mathbf{I})(\mathbf{KIK}) \quad \cong_{\mathrm{v}} \quad \mathbf{I}$$

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

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

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

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

We have already seen in Theorem 6.3(2) that it is finer than applicative simulation. There is a good reason for this, intuitively as follows: both $\sqsubseteq$ on π-terms and $\lesssim$ on λ-terms reflect reduction behaviour *in all appropriate contexts*, and we have seen that $[\![M]\!]u$ can be placed in stranger contexts in the π-calculus than can be inflicted upon M in the λ-calculus. But one can show that β-reduction respects $\sqsubseteq$ – i.e. $(\lambda xM)N \sqsupseteq M\{^N/_x\}$ where $\sqsupseteq = \sqsubseteq \cap \sqsupseteq$. Analogously, in the call-by-value calculus $(\lambda xM)V \sqsupseteq_{\mathrm{v}} M\{^V/_x\}$. But we should like to know much more about these preorders. ‡

Another question is whether we can encode other reduction strategies in the same direct

‡ At the time of final revision of this paper for publication, strong progress on these problems has been made by Sangiorgi; the results will be reported in his forthcoming PhD thesis (Sangiorgi 1992).

way. The simplest is probably an r-determinate variant of weak call-by-value reduction (Section 7), in which the rule APPR is replaced by

$$\text{APPR}_{\text{v}} \ : \ \frac{N \rightarrow_{\text{v}} N'}{VN \rightarrow_{\text{v}} VN'}$$

– constraining the operator to be a value before the operand is reduced. The encoding of this strategy looks like a simple exercise, and moreover the proof of correctness corresponding to Lemma 7.5 should be easier because of r-determinacy. More challenging is to consider call-by-value – and other – strategies which admit the ξ rule:

$$\xi \ : \ \frac{M \rightarrow M'}{\lambda xM \rightarrow \lambda xM'}$$

This can probably be achieved with little difficulty if we are prepared to modify the π-calculus by allowing reduction under input guards, by adding the rule

$$\frac{P \rightarrow P'}{x(y).P \rightarrow x(y)P'}$$

to those listed in Section 3.2. But it is quite possible that this 'eager' reduction rule can be simulated within the π-calculus as it now stands – at least for those instances of the rule needed for encoding the λ-calculus. This would be a more satisfying outcome.

Close examination of λ-calculus reduction strategies reveals what may be called an oddity, seen in the light of the object 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

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

Then of course

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

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 is a familiar situation in the λ-calculus; but it begins to look odd if we think of N not as a term to be copied, but (as we do in this paper) as an autonomous agent in the π-calculus. For in the latter case the β-reduction is modelled by transmitting via x not the *term* N, but *access* to the *agent* N. Then there seems no good reason why transmitting access in this way should necessarily coincide with the cloning of two or more copies of N. (Whereas it *does* seem reasonable for N to clone whenever it is *applied* to an argument within M, and this may happen many times.)

Thus, strategies which appear natural in the presence of *textual substitution* may not seen so natural in a model involving autonomous agents. The former have clearly been most deeply studied in research on the 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.

In fact the work of this paper was first reported (Milner 1990) before I learned about $\lambda\sigma$-calculus of explicit substitutions (Abadi *et al.* 1990) (The $\lambda\sigma$-calculus was discussed

briefly in Section 4.) I am grateful to Pierre-Louis Curien for pointing out that the correspondence between $\lambda\sigma$ and π may be in some respects closer than between λ and π, and it appears fruitful to pursue this connection. In particular, the finer grain of $\lambda\sigma$ (compared with λ) suggests that it admits a greater variety of reduction strategies – some of which may be easier to mimic in π than (for example) the normal-order reduction of λ.

As mentioned in the introduction, Thomsen (1989, 1990) has translated λ-calculus into CHOCS, which has dynamic binding of port-names. He also gives translations between π-calculus and Plain CHOCS, a version of CHOCS with static binding of port-names. Thomsen prepares the ground for proving that a natural form of equivalence is preserved by these translations, though the correct formulation and proof of these results has yet to be settled. In general, much remains to be learned about inter-translation among a variety of λ-calculi and concurrent calculi.

A different 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 Milner *et al.* (1989) 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 Milner *et al.* (1989) we can encode truth-values thus:

$$\begin{aligned} \mathbf{true}(x) &\stackrel{\text{def}}{=} x(u)(v).u \\ \mathbf{false}(x) &\stackrel{\text{def}}{=} x(u)(v).v \end{aligned}$$

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 $\overline{u}(z)$, and we assume that u and v are not free in P_1 or P_2, for it turns out that up to strong bisimilarity

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

In fact, the use of $\mid$ in P, in place of $+$ as suggested in Milner *et al.* (1989), yields what is needed.

One use of summation has been to provide normal forms, and thence complete axiomatisations, for agents under various congruences (Hennessy and Milner 1985, Milner 1989). 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

Abadi, M., Cardelli, L., Curien, P–L. and Lévy, J–J. (1990), Explicit Substitutions, *Proceedings of POPL '90*, pp. 31–46.

Abramsky, S. (1989), The Lazy Lambda Calculus, in: *Research Topics in Functional Programming*, ed. D. Turner, Addison Wesley, pp. 65–116.

Barendregt, H.P. (1981), *The Lambda Calculus, Its Syntax and Semantics*, Studies in Logic and Foundations of Mathematics, Vol 103, North Holland.

Berry, G. (1979), *Modèles Complètement Adéquats et Stables des lambda-calcul typés*, Thèse de Doctorat d'Etat, Université Paris VII.

Berry, G. and Boudol, G. (1990), The Chemical Abstract Machine, *Proc 17th Annual Symposium on Principles of Programming Languages.*

Boudol, G. (1989), Towards a Lambda-Calculus for Concurrent and Communicating Systems, *Proc TAPSOFT 1989*, Lecture Notes in Computer Science 351, Springer-Verlag, pp. 149–161.

Curry, H.B. and Feys, R. (1958), *Combinatory Logic, Vol 1*, North Holland.

Clinger, W.D. (1981), *Foundations of Actor Semantics*, Ph.D. thesis, Massachusetts Institute of Technology, Artificial Intelligence Lab Technical Report #633, May 1981. Reprinted in: *Towards Open Information Systems Science*, Hewitt, C.E., Manning, C.R., Inman, J.T. and Agha, G. (eds.), Cambridge, MA: MIT Press, Spring 1991.

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

Girard, J.-Y. (1987), Linear Logic, *Journal of Theoretical Computer Science*, Vol 50, pp. 111–102.

Hennessy, M. and Milner, R. (1985), Algebraic Laws for Non-determinism and Concurrency, *Journal of ACM*, Vol 32, pp. 137–161.

Hewitt, C., Bishop, P. and Steiger, R. (1973), A Universal Modular Actor Formalism for Artificial Intelligence, *Proc IJCAI '73*, Stanford, California, pp. 235–245.

Hewitt, C. (1977), Viewing Control Structures as Patterns of Passing Messages, *Journal of Artificial Intelligence*, Vol 8, No 3, pp. 323–364, June 1977.

Landin, P.J. (1964), The Mechanical Evaluation of Expressions, *Computer Journal*, Vol 4, pp. 308–320.

Milner, R. (1977), Fully Abstract Models of Typed Lambda-calculi, *Journal of Theoretical Computer Science*, Vol 5, pp. 1–23.

Milner, R. (1990), *Functions as Processes*, Research Report No. 1154, INRIA, Sophia Antipolis, February 1990.

Milner, R. (1989), *Communication and Concurrency*, Prentice Hall.

Milner, R., Parrow, J.G. and Walker, D.J. (1989), *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.

Nielson, F. (1989), The Typed λ-calculus with First-class Processes, *Proc PARLE 89*, Lecture Notes in Computer Science 366, Springer-Verlag.

Ong, C-H.L. (1988), Fully Abstract Models of the Lazy Lambda Calculus, *Proc 29th Symposium on Foundations of Computer Science*, pp. 368–376.

Ong, C-H.L. (1990), *Operational extensionality of Plotkin's pure untyped call-by-value lambda-calculus* λv, Unpublished memorandum.

Plotkin, G.D. (1975), Call-by-name and Call-by-value and the λ-calculus, *Journal of Theoretical Computer Science*, Vol 1, pp. 125–159.

Sangiorgi, D. (1992), Forthcoming PhD thesis, University of Edinburgh.

Stoughton, A. (1989), Private communication.

Thomsen, B. (1989), A Calculus of Higher-order Communicating Systems, *Proc 16th Annual Symposium on Principles of Programming Languages*, pp. 143–154.

Thomsen, B. (1990), *A Calculi for Higher-order Communicating Systems*, PhD Thesis, Department of Computing, Imperial College, London.

Walker, D.J. (1991), π-Calculus Semantics of Object-oriented Programming Languages, *Proc conference Theoretical Aspects of Computer Software, Japan*, Lecture Notes in Computer Science 526, Springer-Verlag, pp. 532–547.