Towards a lambda-calculus for concurrent and communicating systems

Gérard Boudol

UNITE 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° 885

TOWARDS A LAMBDA-CALCULUS FOR CONCURRENT AND COMMUNICATING SYSTEMS

Gérard BOUDOL

AOUT 1988

RR_0885

Towards a Lambda-Calculus for Concurrent and Communicating Systems

Un Lambda-Calcul pour le Parallélisme et la Communication

(Note)

Gérard Boudol
INRIA Sophia-Antipolis
06565-VALBONNE FRANCE

Abstract.

We introduce a calculus for concurrent and communicating processes, which is a direct and simple extension of the λ-calculus. The communication mechanism we use is that of Milner's calculus CCS: to communicate consists in synchronously sending and receiving a value through a shared port. Then the calculus is parameterized on a given set of port names, which are used in the two primitives for sending and receiving a value – as in the λ-calculus, a value can be any term. We use two parallel constructs: the first is interleaving, which does not allow communication between agents. The second, called cooperation, is a synchronizing construct which forces two agents to communicate on every port name. We show that the λ-calculus is a simple sub-calculus of ours: λ-abstraction is a particular case of reception (on a port named λ), and application is a particular case of cooperation.

Résumé.

Nous présentons un calcul de processus parallèles et communicants qui est une extension directe du λ-calcul. Le principe de la communication est celui du calcul CCS de Milner: c'est le rendez-vous entre une émission d'une valeur et sa réception à travers une porte commune. Notre calcul est donc paramétré par un ensemble de noms de portes, qui sont utilisés dans les primitives d'émission et de réception – ici, comme dans le λ-calcul non-typé, tout terme du calcul est une valeur possible. Nous introduisons deux autres construction: la première est l'entrelacement, qui n'autorise pas de communication entre les agents. La seconde, que nous appelons coopération, force au contraire deux agents à communiquer – jusqu'à la terminaison de l'un d'eux. Nous montrons que ce calcul contient le λ-calcul d'une manière très directe: l'abstraction est un cas particulier de réception (sur une porte nommée λ), et l'application est un cas particulier de coopération, où l'argument est explicitement émis (sur la porte λ).

Towards a Lambda-Calculus for Concurrent and Communicating Systems (Note)

Gérard Boudol
INRIA Sophia-Antipolis
06565-VALBONNE FRANCE

Abstract.
We introduce a calculus for concurrent and communicating processes, which is a direct and simple extension of the λ-calculus. The communication mechanism we use is that of Milner's calculus CCS: to communicate consists in synchronously sending and receiving a value through a shared port. Then the calculus is parameterized on a given set of port names, which are used in the two primitives for sending and receiving a value – as in the λ-calculus, a value can be any term. We use two parallel constructs: the first is interleaving, which does not allow communication between agents. The second, called cooperation, is a synchronizing construct which forces two agents to communicate on every port name. We show that the λ-calculus is a simple sub-calculus of ours: λ-abstraction is a particular case of reception (on a port named λ), and application is a particular case of cooperation.

1. Introduction.

The λ-calculus of Church formalizes in a very concise way the idea of functions being applied to arguments. Despite its simplicity, this calculus provides an astonishingly rich model for sequential evaluation, see [2]. A challenging problem that has emerged for some time is to devise a similar framework for concurrent and communicating processes, relying upon some "minimal" concepts for concurrency and communication. A natural claim is that such a formal model for processes should contain the λ-calculus as a simple sub-calculus – this would provide us with the full power of combinators. This note presents an attempt in this direction.

Regarding communication, our main source of inspiration is Milner's CCS [4]. Communication in CCS is a value passing act which two processes perform simultaneously: one of the two partners sends a value through a labelled port, while the other receives this value on a port labelled by the same name, say α. Correspondingly there are two communication primitives in CCS, an output construct and an input construct. The output construct is $\bar{\alpha}e.p$, representing a process sending

e on the port α, and then behaving as p. In this construct, e is an expression belonging to some outer language. The complementary input construct is $\alpha x.p$, representing a process receiving some value at the port α; here x is a bound variable, and receiving the value v yields a new process $p[x \mapsto v]$, that is p where v is substituted for x. Communication occurs when two concurrent processes perform matching send and receive actions. Therefore the *interaction law* may be stated, using $\|$ for parallel composition, as:

$$(\alpha x.p \parallel \bar{\alpha}e.q) \longrightarrow (p[x \mapsto v] \parallel q)$$

where v is the result of evaluating e. In CCS such a transition is labelled by the communication action τ.

Let us discuss briefly how one could use CCS's ideas to find a generalization of the λ-calculus. Milner remarked (*cf.* [4] p.128) that one may compare a function's argument places with input ports of a process. Indeed the terms $\lambda x.p$ of Λ and $\alpha x.p$ of CCS behave quite similarly: both of them wait for a value to be substituted for x in p. This suggests that one could regard these two constructs as the same one – λ is thus a port name, the only one for the λ-calculus (*cf.* [4] p.49). Another obvious idea is that application of a function to its argument should be a special kind of communication (see again [4] p.128), or more precisely that β-reduction should be the typical instance of an interaction law. Then application appears as a parallel composition, where the argument is explicitly sent to the function. Regarding the sending primitive, we shall keep to the philosophy of the λ-calculus, where any term is a possible value. Then the sending construct is $\bar{\alpha}p.q$, where p is any agent. In fact we are only interested in the case where q is a terminated process $\mathbb{1}$, which is like nil in CCS. Then $\bar{\alpha}p$ will be an abbreviation for $\bar{\alpha}p.\mathbb{1}$.

To work out the previous ideas, let us now introduce a first attempt – the calculus we actually propose will be a little bit more sophisticated. In order to build agents $\alpha x.p$ and $\bar{\alpha}p$ we need a denumerable set X of *variables* $x, y, z \ldots$, and a non-empty set N of *port names*. We shall use α, $\beta \ldots$ to range over port names. Then the syntax of the tentative calculus is given by the following grammar:

$$p \ ::= \ \mathbb{1} \mid x \mid \alpha x.p \mid \bar{\alpha}p \mid (p \parallel p)$$

where α is any port name. We shall use p, q, $r \ldots$ to range over terms. As usual, the variable x is bound if it is in the scope of an αx, and some care is needed in defining substitution. For simplicity, we shall adopt Barendregt's variable convention ([2]): in any mathematical context where they occur, the terms $p_1, \ldots, p_n$ are supposed to exhibit bound variables different from the free variables.

In this calculus, communication is given by an obvious adaptation of the interaction law of CCS, namely:

$$(\alpha x.p \parallel \bar{\alpha}q) \longrightarrow (p[x \mapsto q] \parallel \mathbb{1})$$

The term $\mathbb{1}$ represents a terminated process, and is a unit for parallel composition. Therefore the term we get after a communication, that is $(p[x \mapsto q] \parallel \mathbb{1})$, behaves like $p[x \mapsto q]$. Then the interaction law is similar to β-reduction, and, assuming that N contains a distinguished name λ, we can try to represent Λ as the subset of terms given by the grammar:

$$p \ ::= \ x \mid \lambda x.p \mid (p \parallel \bar{\lambda}p)$$

We could denote $(p \parallel \bar{\lambda}q)$ by (pq) (application), so that the previous rule is the β-rule, up to the simplification $(r \parallel \mathbb{1}) = r$. However this simple calculus fails to capture the λ-calculus. Let us see this point in more detail.

CCS also formalizes the natural idea that parallel composition is commutative and associative, so that processes need not be contiguous to communicate – unlike λ-terms where communication is sequential application. In other words, the following should hold in our tentative calculus:

$$(\cdots \parallel \alpha x.p \parallel \cdots \parallel \bar{\alpha}q \parallel \cdots) \longrightarrow (\cdots \parallel p[x \mapsto q] \parallel \cdots \parallel \mathbb{1} \parallel \cdots)$$

Let us assume for a while that we have two rules stating that parallel composition is commutative and associative:

$((p \parallel q) \parallel r) \longrightarrow s \;\vdash\; (p \parallel (q \parallel r)) \longrightarrow s$

$(p \parallel q) \longrightarrow s \;\vdash\; (q \parallel p) \longrightarrow s$

These two rules introduce *conflicts*, arising from communication (technically speaking, we should say that associativity introduces overlapping redexes). As in CCS, there is a possibility that inputs at the same port may have different sources, and outputs at the same port different destinations. Then two communications are conflicting if they share the same destination, or the same source, the typical example being $((\alpha x.p \parallel \bar{\alpha}q) \parallel \bar{\alpha}r)$, and solving the conflict introduces non-determinism. To our view, the non-determinism arising from conflicting communications is a rather pleasant feature. But there is a negative consequence to the associativity (and commutativity) of parallel composition, namely that we lose the correspondence with λ-calculus application. For instance the term $((\lambda xy.x)u)v$ cannot be accurately represented by $p = ((\lambda x.\lambda y.x \parallel \bar{\lambda}u) \parallel \bar{\lambda}v)$ since we have $p \xrightarrow{*} ((v \parallel \mathbb{1}) \parallel \mathbb{1})$.

We insist on obtaining the λ-calculus as a sub-calculus, without restricting the evaluation rules. More precisely, our goal is to find a direct generalization of the λ-calculus, that is a calculus where the operational semantics, once restricted to an appropriate subset of terms, gives an exact image of the β-reduction on Λ. Then we must abandon the parallel composition of CCS. Our proposal is to split it into two constructors. The first is the usual *interleaving* construct $(p \mid q)$, which consists in juxtaposing p and q, without any communication wire between them. This operator represents *concurrency*. The second construct, denoted $(p \odot q)$ and called *cooperation*, consists in plugging together p and q – up to termination of one of them. This operator provides for *communication*, which can only occur within a $(p \odot q)$. On the other hand, a compound process $(p \mid q)$ can propose communications to its environment, and the interleaving operator is commutative and associative (and satisfies $(p \mid \mathbb{1}) = p$). Therefore the interaction law becomes:

$$\gamma: \quad ((\cdots \mid \alpha x.p \mid \cdots) \odot (\cdots \mid \bar{\alpha}q \mid \cdots)) \longrightarrow ((\cdots \mid p[x \mapsto q] \mid \cdots) \odot (\cdots \mid \mathbb{1} \mid \cdots))$$

Like the operator considered by Milner in [4] (p.21), the operator $\odot$ is not associative. Its semantics will be such that $(p \odot \mathbb{1})$ behaves like p. In other words, $\odot$ is not a *static* operator: $(p \odot q)$ cannot communicate with another process if p and q are not terminated, but it will be free to do so once p or q terminates. Then the λ-calculus application (pq) may be represented by a combination of cooperation and output, namely $(p \odot \bar{\lambda}q)$. To ensure the correctness of this representation, we must introduce *structural rules*, which formalize the fact that reduction is compatible with the constructors. For instance there will be two rules allowing internal computations within a guarded process:

$p \longrightarrow p' \;\vdash\; \bar{\alpha}p \longrightarrow \bar{\alpha}p'$

$p \longrightarrow p' \;\vdash\; \alpha x.p \longrightarrow \alpha x.p' \qquad (\xi \text{ rule})$

These do not hold in CCS, where the transitions describe the behaviour of a reactive system, rather than an evaluation mechanism. Processes in our calculus could be qualified as *interactive* systems rather than reactive – in fact this is a matter of evaluation strategy.

One should observe that we still have conflicting communications, so that we can represent a *non-deterministic choice*, using the standard combinator $\mathbf{K} = \lambda x.\lambda y.x$, which chooses its first argument and deletes the second one:

$$(p \oplus q) = (\mathbf{K} \odot (\bar{\lambda}p \mid \bar{\lambda}q))$$

It is easy to see that we have:

$$(p \oplus q) \xrightarrow{*} (p \odot (\mathbb{1} \mid \mathbb{1})) \equiv p \qquad \text{and} \qquad (p \oplus q) \xrightarrow{*} (q \odot (\mathbb{1} \mid \mathbb{1})) \equiv q$$

where $\equiv$ is the syntactic equality, defined in the next section. Then we should say that $\oplus$ is an *internal* choice, meaning that $(p \oplus q)$ may evolve to one of p or q by internal communications. We also have, due to the structural rules:

$$p \longrightarrow p' \;\Rightarrow\; (p \oplus q) \longrightarrow (p' \oplus q) \qquad \text{and} \qquad q \longrightarrow q' \;\Rightarrow\; (p \oplus q) \longrightarrow (p \oplus q')$$

which shows that $\oplus$ is quite different from CCS sum. This example also shows that the Church-Rosser property no longer holds for reduction, and that it would be inconsistent to regard the associated conversion as establishing a notion of equality. We shall adopt an intensional notion of equality, namely that of *observational equivalence* of Milner [4], which relies on the communication capabilities of a process.

To conclude the informal presentation of our calculus, let us say a few words about the *binders*. In the λ-calculus, these are sequences $\lambda x_1 \ldots \lambda x_k$. corresponding to application to a stream of arguments. Since in our calculus we may have interleaved arguments, it seems natural to correspondingly generalize the binders, allowing not only sequences of αx's, but also interleavings. Then we will have terms of the form $\langle \alpha_1 x_1 \mid \cdots \mid \alpha_k x_k \rangle . p$, meaning that p waits for k unordered values. This allows us for instance to represent non-deterministic choice as a simple variant of the **K** combinator, namely $\oplus =_{\mathrm{def}} \langle \lambda x \mid \lambda y \rangle . x$.

2. The γ-calculus.

Given a denumerable set X of variables and a non-empty set N of port names, we first define the *binders*, which are the terms built according to the following grammar, where α is any port name, and x stands for any variable:

$$\rho \;::=\; \varepsilon \mid \alpha x \mid (\rho \cdot \rho) \mid (\rho \mid \rho)$$

Intuitively αx represents a reception on the port α, $(\rho \cdot \rho')$ represents sequencing of such receptions while $(\rho | \rho')$ represents their interleaving. The term ε is an *empty* binder, therefore we shall consider binders up to the congruence $\doteq$ generated by the equations:

$$(\rho \cdot \varepsilon) = \rho = (\varepsilon \cdot \rho)$$
$$(\rho \mid \varepsilon) = \rho = (\varepsilon \mid \rho)$$

The congruence $\doteq$ defines the syntactic equality over binders. Any binder ρ will bind the variables belonging to the set $\mathrm{var}(\rho)$ defined as follows:

(i) $\mathrm{var}(\varepsilon) = \emptyset$

(ii) $\mathrm{var}(\alpha x) = \{x\}$

(iii) $\mathrm{var}(\rho \cdot \rho') = \mathrm{var}(\rho) \cup \mathrm{var}(\rho')$

(iv) $\mathrm{var}(\rho \mid \rho') = \mathrm{var}(\rho) \cup \mathrm{var}(\rho')$

The syntax of the γ-calculus is given by the following grammar, where α is any port name of N, ρ is any binder, and x stands for any variable:

$$p \;::=\; \mathbb{1} \mid x \mid \bar{\alpha} p \mid \langle \rho \rangle . p \mid (p \odot p) \mid (p \mid p)$$

We denote by Γ the set of terms generated by this grammar, and we shall use $p, q, r \ldots$ to range over terms – which will be called *agents* or *processes*. For simplicity we shall denote $\langle \alpha x \rangle . p$ by $\alpha x . p$, and we shall omit some parentheses, using for instance $\langle \rho \mid \rho' \rangle . p$ instead of $\langle (\rho \mid \rho') \rangle . p$. A variable x is *bound* if it is in the scope of a binder. Then in substituting q for y in p, yielding $p[y \mapsto q]$, we might have to rename some bound variables of p. Although this is a standard matter (see [2], appendix C), it is worth to carefully define substitution. Here we adapt the definitions of [5]. The set $\mathrm{free}(p)$ of *free variables* of the term p is given by (*i*) $\mathsf{free}(\mathbb{1}) = \emptyset$

(*ii*) $\mathsf{free}(x) = \{x\}$

(*iii*) $\mathsf{free}(\bar{\alpha}p) = \mathsf{free}(p)$

(*iv*) $\mathsf{free}(\langle\rho\rangle.p) = \mathsf{free}(p) - \mathsf{var}(\rho)$

(*v*) $\mathsf{free}(p \odot q) = \mathsf{free}(p) \cup \mathsf{free}(q) = \mathsf{free}(p \mid q)$

A term p is *closed* if $\mathsf{free}(p) = \emptyset$. A *substitution* is any mapping $\sigma\colon X \to \Gamma$. We use $\sigma, \sigma' \ldots$ to range over the set S of substitutions. The identity substitution is denoted ι. The *updating* operation on substitutions is defined as follows: let $x \in X$, $p \in \Gamma$ and $\sigma \in S$; then the new substitution $\sigma' = (x \mapsto p/\sigma)$ is given by:

$$\sigma'(y) = \begin{cases} p & \text{if } y = x \\ \sigma(y) & \text{otherwise} \end{cases}$$

For a binder ρ and a renaming, that is a substitution $\xi\colon X \to X$, the result $\rho[\xi]$ of applying ξ to ρ is defined in an obvious way, that is by structural induction starting from $(\alpha x)[\xi] = \alpha\xi(x)$. To define substitution on terms of Γ, we assume given for each pair V, W of finite subsets of X a mapping $\mathsf{new}_{V,W}\colon V \to X - W$ satisfying:

(*i*) $\mathsf{new}_{V,W}$ is injective;

(*ii*) $\mathsf{new}_{V,W}(x) = x \quad \text{if } x \in V - W$.

This assumption makes sense since X is infinite. Then the result $p[\sigma]$ of applying the substitution σ to the term p is defined by structural induction, the only non obvious case being $p = \langle\rho\rangle.q$ with $\mathsf{var}(\rho) \neq \emptyset$. In that case, let:

$V = \mathsf{var}(\rho) = \{x_1, \ldots, x_n\}$

$W = \{v \mid \exists z \in \mathsf{free}(p)\ v \in \mathsf{free}(\sigma(z))\}$

$\mathsf{new}_{V,W}(x_i) = y_i \quad \text{for } 1 \leq i \leq n$

$\xi = [x_1 \mapsto y_1/\cdots/x_n \mapsto y_n/\iota]$

$\sigma' = [x_1 \mapsto y_1/\cdots/x_n \mapsto y_n/\sigma]$

Then we define $p[\sigma]$ to be $\langle\rho[\xi]\rangle.q[\sigma']$. We shall denote $p[x \mapsto q/\iota]$ by $p[x \mapsto q]$ (similarly $\rho[x \mapsto y]$ denotes $\rho[x \mapsto y/\iota]$), and define the *composition* $\rho \bullet \sigma$ of two substitutions by:

$$(\rho \bullet \sigma)(x) = \sigma(x)[\rho]$$

As usual, we regard terms differing only on the name of bound variables as syntactically identical. Moreover we also regard $\mathbb{1}$ as a terminated agent, which should be cancelled from parallel combinations. Then our *syntactical equality* is the congruence $\equiv$ generated by the following equations:

$(p \odot \mathbb{1}) = p = (\mathbb{1} \odot p)$

$(p \mid \mathbb{1}) = p = (\mathbb{1} \mid p)$

$\langle\varepsilon\rangle.p = p$

$\langle\rho\rangle.p = \langle\rho'\rangle.p \quad \text{if } \rho \doteq \rho'$

$\langle\rho\rangle.p = \langle\rho[x \mapsto y]\rangle.p[x \mapsto y] \quad \text{if } x \in \mathsf{var}(\rho) \text{ and } y \notin \mathsf{free}(p) \cup \mathsf{var}(\rho)$

One could prove that $\equiv$ is *substitutive*, that is $p \equiv q \Rightarrow p[\sigma] \equiv q[\sigma]$ for all substitution σ (*cf.* [5]). We shall say that an agent p is *terminated*, in notation $p\dagger$, if $p \equiv \mathbb{1}$.

To define the semantics, that is the laws of reduction, we shall use Milner's technique of labelled transitions. This is the best way to formalize the idea that processes need not be contiguous to communicate. Let us introduce some technical definitions. The semantics is given by means of labelled transitions $p \xrightarrow{a} p'$ where the action a may be $\bar{\alpha}_p$, which means sending p at port α, or α_p, which means receiving p at port α, or the communication action τ. This could be formalized by saying that the set of actions is $A = (N \times \Gamma) \cup (\Gamma \times N) \cup \{\tau\}$ (if we regard α_p and $\bar{\alpha}_p$ as notations for

(α, p) and (p, α) respectively). We shall say that a and b are *complementary* actions, in notation $a \frown b$, if $a = \alpha_p$ and $b = \bar{\alpha}_p$, or symmetrically $a = \bar{\alpha}_p$ and $b = \alpha_p$. To define the semantics of $\langle\rho\rangle.p$ we also need to specify the reception actions allowed by the binder ρ. To this end we introduce a transition relation $\rho \xrightarrow{a} \rho'$ between binders, where a has the form $\alpha_{x,p}$. This transition relation is the least one satisfying the following rules:

$$p \in \Gamma \ \vdash \ \alpha x \xrightarrow{\alpha_{x,p}} \varepsilon$$

$$\rho \xrightarrow{a} \rho' \ \vdash \ (\rho \cdot \rho'') \xrightarrow{a} (\rho' \cdot \rho'')$$

$$\rho \doteq \varepsilon \ \& \ \rho' \xrightarrow{a} \rho'' \ \vdash \ (\rho \cdot \rho') \xrightarrow{a} \rho''$$

$$\rho \xrightarrow{a} \rho' \ \vdash \ (\rho \mid \rho'') \xrightarrow{a} (\rho' \mid \rho'')$$

$$\rho \xrightarrow{a} \rho' \ \vdash \ (\rho'' \mid \rho) \xrightarrow{a} (\rho'' \mid \rho')$$

The transition relation $\longrightarrow$ on agents is the least subset of $\Gamma \times A \times \Gamma$ satisfying the rules given below (where, as usual, we denote $(p, a, p') \in \longrightarrow$ by $p \xrightarrow{a} p'$). The first two rules introduce the communication actions:

output R1: $\vdash \ \bar{\alpha}p \xrightarrow{\bar{\alpha}_p} \mathbb{1}$

input R2: $\rho \xrightarrow{\alpha_{x,q}} \rho' \ \vdash \ \langle\rho\rangle.p \xrightarrow{\alpha_q} \langle\rho'\rangle.p[x \mapsto q]$

One may observe that, due to R2, the *sort* of a process – that is the set of port names through which it may communicate – can evolve dynamically: if $p \xrightarrow{a} p'$ the sort of p' is not necessarily a subset of the sort of p. There is another rule concerning input, when the binder is empty:

input R3: $\rho \doteq \varepsilon \ \& \ p \xrightarrow{a} p' \ \vdash \ \langle\rho\rangle.p \xrightarrow{a} p'$

The interaction law is given by:

communication R4 (γ): $p \xrightarrow{a} p' \ , \ q \xrightarrow{b} q' \ \& \ a \frown b \ \vdash \ (p \odot q) \xrightarrow{\tau} (p' \odot q')$

The following rules state that the transition relation $\xrightarrow{\tau}$ is compatible with all the constructors, and that $\xrightarrow{a}$ is compatible with interleaving for any $a \in A$:

output R5: $p \xrightarrow{\tau} p' \ \vdash \ \bar{\alpha}p \xrightarrow{\tau} \bar{\alpha}p'$

input R6: $p \xrightarrow{\tau} p' \ \vdash \ \langle\rho\rangle.p \xrightarrow{\tau} \langle\rho\rangle.p'$

cooperation (left) R7: $p \xrightarrow{\tau} p' \ \vdash \ (p \odot q) \xrightarrow{\tau} (p' \odot q)$

cooperation (right) R8: $q \xrightarrow{\tau} q' \ \vdash \ (p \odot q) \xrightarrow{\tau} (p \odot q')$

interleaving (left) R9: $p \xrightarrow{a} p' \ \vdash \ (p \mid q) \xrightarrow{a} (p' \mid q)$

interleaving (right) R10: $q \xrightarrow{b} q' \ \vdash \ (p \mid q) \xrightarrow{b} (p \mid q')$

Our last two rules formalize the fact that cooperation only holds up to termination of one partners:

cooperation (right unit) R11: $p \xrightarrow{a} p' \ , \ q\dagger \ \vdash \ (p \odot q) \xrightarrow{a} p'$

cooperation (left unit) R12: $q \xrightarrow{b} q' \ , \ p\dagger \ \vdash \ (p \odot q) \xrightarrow{b} q'$

One can readily see from these rules that $(p \odot q)$ can only perform τ actions when $p \not\equiv \mathbb{1} \not\equiv q$, while communication between p and q is forbidden within the construct $(p \mid q)$. We shall mostly denote

$p \xrightarrow{\tau} p'$ by $p \longrightarrow p'$, and by definition this is the *γ-reduction* between terms of Γ.

Our first purpose is to show that the γ-calculus contains the λ-calculus, up to syntactical equality. Then we first have to check that syntactical equality is consistent with the operational semantics. Formally speaking, this amounts to show that $\equiv$ is a bisimulation. Our notion of bisimulation is a slight extension of Park and Milner's one, in three respects: first we wish to define bisimulation for non-closed terms. Then we shall say that p and q are similar if any instances $p[\sigma]$ and $q[\sigma]$ of p and q have similar behaviours. Second we must regard the actions – made out of agents – up to bisimulation, and third we must take into account the potential termination of agents. We shall use two notions of simulation: the first one, called strong, is relative to the transition relation $\longrightarrow$. We will see the second (weak) one latter. Let $R \subseteq \Gamma \times \Gamma$ be a relation on terms; we define its extension $\widehat{R} \subseteq A \times A$ on actions as follows:

$$a \, \widehat{R} \, b \;\Leftrightarrow_{\text{def}}\; a = b \text{ or } \exists \alpha \in N \, \exists p, q. \, p \, R \, q \;\&\; a = \bar{\alpha}_p \;\&\; b = \bar{\alpha}_q$$

A relation $R \subseteq \Gamma \times \Gamma$ is

(i) a *strong simulation* if it satisfies

S1: $p \, R \, q \;\&\; p[\sigma] \xrightarrow{a} p' \;\Rightarrow\; \exists b. \, a \, \widehat{R} \, b \; \exists q'. \, p' \, R \, q' \;\&\; q[\sigma] \xrightarrow{b} q'$

S2: $p \, R \, q \;\&\; p \dagger \;\Rightarrow\; q \dagger$

(ii) a *strong bisimulation* if it is a symmetric strong simulation.

The first property defining a simulation is a refinement of the usual one: (instances of) strongly similar agents must perform similar actions. Note that since $p[\iota] = p$ we have

$$p \, R \, q \;\&\; p \xrightarrow{a} p' \;\Rightarrow\; \exists b. \, a \, \widehat{R} \, b \; \exists q'. \, p' \, R \, q' \;\&\; q \xrightarrow{b} q'$$

The second property states that strong simulation preserves the termination property. This property, that is $p \equiv \mathbb{1}$, should not be confused with the property of being a normal form, that is γ-irreducibility (p is γ-irreducible if $\{ q \mid p \longrightarrow q \} = \emptyset$): to our view a term such as $\alpha x.x$ is not terminated since it can perform some actions – namely α_p. One should note that every strong simulation is substitutive:

FACT. *If R is a strong simulation then $p \, R \, q \;\Rightarrow\; p[\sigma] \, R \, q[\sigma]$ for all substitution σ.*

This holds because $(p[\sigma])[\rho] = p[\rho \bullet \sigma]$ (*cf.* [5]).

PROPOSITION. *The congruence $\equiv$ is a strong simulation on Γ.*

The proof is straightforward: one proceeds by induction on the proof of $p[\sigma] \equiv q[\sigma]$ (using the fact that $\equiv$ is substitutive), and then by induction on the proof of the transition $p[\sigma] \xrightarrow{a} p'$ to show that $\exists q' \equiv p' \; q[\sigma] \xrightarrow{a} q'$. In the case of R4, one must show that:

$$p \xrightarrow{\alpha_q} p' \;\&\; q \equiv q' \;\Rightarrow\; \exists p''. \, p' \equiv p'' \;\&\; p \xrightarrow{\alpha_{q'}} p''$$

One must also prove that $\doteq$ satisfies:

$$\rho_0 \xrightarrow{a} \rho_0' \;\&\; \rho_0 \doteq \rho_1 \;\Rightarrow\; \exists \rho_1'. \, \rho_0' \doteq \rho_1' \;\&\; \rho_1 \xrightarrow{a} \rho_1'$$

The details are omitted ∎

This result allows us to define the transition relation $\longrightarrow$ on $\Gamma/\!\equiv$. We shall abusively write transitions between simplified terms (obtained by cancelling $\mathbb{1}$ from parallel combinations), as for instance in a special case of the γ-rule: $(\alpha x.p \odot \bar{\alpha} q) \longrightarrow p[x \mapsto q]$.

Now we can show that the γ-calculus contains the λ-calculus – which we assume to be well-known. The syntax of Λ is given by the following grammar:

$$M \ ::= \ x \mid \lambda x.M \mid (MM)$$

The rules for β-reduction, denoted $M \longrightarrow N$, are:

R′1 (β): $(\lambda x.M)N \longrightarrow M[x \mapsto N]$

R′2: $M \longrightarrow M' \ \vdash \ \lambda x.M \longrightarrow \lambda x.M'$

R′3: $M \longrightarrow M' \ \vdash \ (MN) \longrightarrow (M'N)$

R′4: $N \longrightarrow N' \ \vdash \ (MN) \longrightarrow (MN')$

Assuming that $\lambda \in N$, we define the translation θ from Λ to Γ as follows:

$\theta(x) = x$
$\theta(\lambda x.M) = \langle \lambda x \rangle.\theta(M)$
$\theta(MN) = (\theta(M) \odot \bar{\lambda}\theta(N))$.

We assume that substitution is defined for λ-terms as it was defined for γ-terms, so that the translation preserves substitution, that is:

$$\forall M, N \in \Lambda \quad \theta(M[x \mapsto N]) = \theta(M)[x \mapsto \theta(N)]$$

PROPOSITION. *For all λ-terms M and N:*
(i) $M \longrightarrow N \ \Rightarrow \ \exists p \equiv \theta(N).\ \theta(M) \longrightarrow p$
(ii) $\theta(M) \longrightarrow p \ \Rightarrow \ \exists N \in \Lambda.\ M \longrightarrow N \ \&\ \theta(N) \equiv p$

PROOF: let $M, N \in \Lambda$ such that $M \longrightarrow N$. We proceed by induction on the proof of this transition to show that $\exists p \equiv \theta(N)\ \theta(M) \longrightarrow p$. If this transition is an instance of the β-rule , then we have $M = (\lambda x.P)Q$, $N = P[x \mapsto Q]$ and $\theta(M) = (\langle \lambda x \rangle.\theta(P) \odot \bar{\lambda}\theta(Q))$. Using R1 and R2 we have $\bar{\lambda}\theta(Q) \xrightarrow{a} \mathbb{1}$ with $a = \bar{\lambda}_{\theta(Q)}$ and $\langle \lambda x \rangle.\theta(P) \xrightarrow{b} \langle \varepsilon \rangle.\theta(P)[x \mapsto \theta(Q)]$ with $b = \lambda_{\theta(Q)}$. Therefore using the γ-rule (R4) we have $\theta(M) \longrightarrow (\langle \varepsilon \rangle.\theta(P)[x \mapsto \theta(Q)] \odot \mathbb{1}) \equiv \theta(N)$. All the other cases are trivial.

Conversely let us assume that $\theta(M) \longrightarrow p$. We proceed by induction on the structure of M to show that $\exists N \in \Lambda\ M \longrightarrow N \ \&\ \theta(N) \equiv p$. We cannot have $M \in X$, since a variable cannot perform any action. If $M = \lambda x.P$ then $\theta(M) = \langle \lambda x \rangle.\theta(P)$, and the transition $\theta(M) \longrightarrow p$ must be proved using R6 (since the action is τ). We easily conclude in this case using the induction hypothesis. If $M = (PQ)$ then $\theta(M) = (\theta(P) \odot \bar{\lambda}\theta(Q))$. The transition $\theta(M) \longrightarrow p$ cannot be proved using R11 or R12 (since $\theta(P) \not\equiv \mathbb{1}$). If it is proved using R7 or R8 (and then R5) the result follows from the induction hypothesis. If it is proved using R4, we have $\theta(P) \xrightarrow{a} p'$ and $\bar{\lambda}\theta(Q) \xrightarrow{b} q'$ with $a \frown b$; hence $a \neq \tau \neq b$, and the second transition can only be proved by means of R1. This implies that $b = \bar{\lambda}_{\theta(Q)}$ (and $q' = \mathbb{1}$), and therefore $a = \lambda_{\theta(Q)}$. It is easy to prove that for all $P \in \Lambda$ if $\theta(P) \xrightarrow{\lambda_s} p'$ then $P = \lambda x.P'$ & $p' = \langle \varepsilon \rangle.\theta(P')[x \mapsto s]$. Consequently we have $M = ((\lambda x.P')Q)$, and $\theta(N) = \theta(P')[x \mapsto \theta(Q)] \equiv (\langle \varepsilon \rangle.\theta(P')[x \mapsto \theta(Q)] \odot \mathbb{1})$ ∎

This result allows us to regard the λ-terms as a special kind of γ-terms. To simplify the notations, we shall use (pq) as an abbreviation for $(p \odot \bar{\lambda}q)$ – recall also that $\lambda x.p$ is a notation for $\langle \lambda x \rangle.p$. We shall keep the usual notation for the standard combinators, *cf.* [2], chapter 6. For instance **K** is the (γ-)term $\lambda x.\lambda y.x$, or more simply $\lambda xy.x$.

We already saw that the γ-calculus is strictly more powerful than the λ-calculus: the term $\langle \lambda x \mid \lambda y \rangle.x$ (non-deterministic choice) does not have any image in Λ. Let us see another example, showing that we find in Γ "parallel functions" (which we do not intend to precisely define) which are not definable in Λ. It is known that **K**, which could be denoted also **T**, and $\mathbf{F} = \lambda xy.y$ may be regarded as the *truth values*. One can define in the λ-calculus a combinator representing the *left sequential disjunction*, namely $\mathsf{V} = \lambda xy.(x\mathsf{T})y$. This combinator is left-sequential since one cannot reduce $(\mathsf{V}M)\mathsf{T}$ into T without evaluating M. From Berry's sequentiality theorem (*cf.* [2]), one can show that there is no λ-term representing a *parallel disjunction* O, such that $(\mathsf{O}M)\mathsf{T}$ and $(\mathsf{O}\mathsf{T})M$ can be reduced into T without evaluating M. On the other hand, this combinator is definable in the γ-calculus: this is just a parallel variant of V, namely

$$\mathsf{O} =_{\mathrm{def}} \langle \lambda x \mid \lambda y \rangle.(x\mathsf{T})y$$

Then it is easy to show that:

$$\forall p \in \Gamma \quad (\mathsf{O}\,p)\mathsf{T} \xrightarrow{*} \mathsf{T} \qquad \text{and} \qquad (\mathsf{O}\,\mathsf{T})p \xrightarrow{*} \mathsf{T}$$

since the combinator O can choose what argument eventually needs to be evaluated first:

$$(\mathsf{O}\,p)q \xrightarrow{*} (p\,\mathsf{T})q \qquad \text{and} \qquad (\mathsf{O}\,p)q \xrightarrow{*} (q\,\mathsf{T})p$$

Obviously we also have:

$$(\mathsf{O}\,\mathsf{F})\mathsf{F} \xrightarrow{*} \mathsf{F}$$

Let us see another example, showing that we can retrieve in the γ-calculus some of CCS's ideas about concurrent processes. One of Milner's intentions in designing CCS was to formalize the idea that a process performs possibly infinite sequences of communications with its environment. One may wonder whether it is possible to describe in the γ-calculus a system made out of processes continuously exchanging messages. The answer is positive, thanks to the existence of endlessly reducible terms. In the λ-calculus, the typical example of such terms is $\Omega = \Delta\Delta$, where $\Delta = \lambda x.xx$ is the usual duplicator, for we have $\Omega \longrightarrow \Omega$. Using this feature we can define a process which repeatedly accepts a message on a port α and then sends on port β a response elaborated using a "method" q. Let

$$\delta = \lambda y.\alpha x.(\bar{\beta} q \mid (y \odot \bar{\lambda} y)) \qquad \text{and} \qquad \omega = (\delta \odot \bar{\lambda}\delta)$$

These terms could be written more simply $\delta = \lambda y.\alpha x.(\bar{\beta} q \mid yy)$ and $\omega = (\delta\delta)$. Then we have:

$$\omega \longrightarrow \alpha x.(\bar{\beta} q \mid \omega) \xrightarrow{\alpha_p} (\bar{\beta} q[x \mapsto p] \mid \omega) \xrightarrow{*} (\bar{\beta} r \mid \omega) \xrightarrow{\bar{\beta}_r} (\mathbb{1} \mid \omega)$$

We should say that evaluating ω repeatedly creates the communication channels α and β. We could obviously have written a term receiving the "method" q on some port, and then applying it to several arguments.

Now let us return to the semantics of our calculus. We shall adopt Milner's observational equivalence [4] as our notion of equality. The observational equivalence is defined with respect to a transition relation where one abstracts from internal communications (i.e. τ actions). This transition relation $\Longrightarrow$ is the least subset of $\Gamma \times A \times \Gamma$ containing $\longrightarrow$ and satisfying the following rules:

O1: $\quad \vdash p \xRightarrow{\tau} p$

O2: $\quad p \xRightarrow{a} p''\,,\; p'' \xRightarrow{\tau} p' \;\vdash\; p \xRightarrow{a} p'$

O3: $\quad p \xRightarrow{\tau} p''\,,\; p'' \xRightarrow{a} p' \;\vdash\; p \xRightarrow{a} p'$

It should be clear that $p \xRightarrow{a} p'$ *iff* $a = \tau$ & $p' = p$ or

$$p(\xrightarrow{\tau})^* \xrightarrow{a} (\xrightarrow{\tau})^* p'$$

A relation $R \subseteq \Gamma \times \Gamma$ is

(i) a *weak*, or *observational simulation* if it satisfies

W1: $p\,R\,q \;\&\; p[\sigma] \stackrel{a}{\Longrightarrow} p' \;\Rightarrow\; \exists b.\, a\,\widehat{R}\,b\; \exists q'.\, p'\,R\,q' \;\&\; q[\sigma] \stackrel{b}{\Longrightarrow} q'$

W2: $p\,R\,q \;\&\; p\dagger \;\Rightarrow\; \exists q'.\, q \stackrel{\tau}{\Longrightarrow} q' \;\&\; q'\dagger$

(ii) a *weak*, or *observational bisimulation* if it is a symmetric weak simulation.

Note that the we have:

$$p\,R\,q \;\&\; p[\sigma]\dagger \;\Rightarrow\; \exists q'.\, q[\sigma] \stackrel{\tau}{\Longrightarrow} q' \;\&\; q'\dagger$$

since $p[\sigma] \stackrel{\tau}{\Longrightarrow} p[\sigma]$, hence there exist q'' such that $q[\sigma] \stackrel{\tau}{\Longrightarrow} q''$ and $p[\sigma]\,R\,q''$, therefore there exists q' such that $q'\dagger$ and $q'' \stackrel{\tau}{\Longrightarrow} q'$, hence $q[\sigma] \stackrel{\tau}{\Longrightarrow} q'$. Consequently any observational simulation is substitutive:

FACT. *If R is an observational simulation then $p\,R\,q \;\Rightarrow\; p[\sigma]\,R\,q[\sigma]$ for all substitution σ.*

The following characterization of observational simulations is useful:

LEMMA. *A relation $R \subseteq \Gamma \times \Gamma$ is an observational simulation if and only if*

(i) R is substitutive: $p\,R\,q \;\Rightarrow\; \forall\sigma \in S\;\, p[\sigma]\,R\,q[\sigma]$

(ii) $p\,R\,q \;\&\; p \stackrel{a}{\longrightarrow} p' \;\Rightarrow\; \exists b.\, a\,\widehat{R}\,b\; \exists q'.\, p'\,R\,q' \;\&\; q \stackrel{b}{\Longrightarrow} q'$

(iii) $p\,R\,q \;\&\; p\dagger \;\Rightarrow\; \exists q'.\, q \stackrel{\tau}{\Longrightarrow} q' \;\&\; q'\dagger$

PROOF: it is clear that any observational simulation satisfies these properties. Let us assume that R satisfies *(i)*-*(iii)*. We have to prove

$$p\,R\,q \;\&\; p[\sigma] \stackrel{a}{\Longrightarrow} p' \;\Rightarrow\; \exists b.\, a\,\widehat{R}\,b\; \exists q'.\, p'\,R\,q' \;\&\; q[\sigma] \stackrel{b}{\Longrightarrow} q'$$

Since R is substitutive, it is enough to prove this for $\sigma = \iota$. We proceed by induction on the proof of $p \stackrel{a}{\Longrightarrow} p'$: if this transition is $p \stackrel{a}{\longrightarrow} p'$, then we conclude using *(ii)*. The point is trivial if $p \stackrel{a}{\Longrightarrow} p'$ is an instance of O1, since $p' = p$, $a = \tau$ and $q \stackrel{\tau}{\Longrightarrow} q$ (by O1). The two other cases easily follow from the induction hypothesis ∎

The observational equivalence that we regard as our *semantic equality* is the coarsest weak bisimulation. Such a coarsest bisimulation exists, and is an equivalence, since we have:

DEFINITION and FACT. *Let us define:*

$$p \approx q \;\Leftrightarrow_{\text{def}}\; \exists R \subseteq \Gamma \times \Gamma \textit{ weak bisimulation such that } p\,R\,q$$

Then $\approx$ is a weak bisimulation. Moreover $\approx$ is an equivalence.

(the proof is omitted – the only point to check is that the composition of two weak simulations is a weak simulation).

A consequence of the previous lemma is that any strong simulation is also a weak one. Then for instance we have $\equiv\, \subseteq\, \approx$. This lemma also allows us to prove some algebraic properties of the operators with respect to $\approx$, as for example:

$(p \odot q) \approx (q \odot p)$
$(p \odot \mathbb{1}) \approx p \approx (\mathbb{1} \odot p)$
$(p \mid (q \mid r)) \approx ((p \mid q) \mid r)$
$(p \mid q) \approx (q \mid p)$
$(p \mid \mathbb{1}) \approx p \approx (\mathbb{1} \mid p)$

Note that the cooperation operator is not associative (up to $\approx$): for instance

$$((\lambda x.x \odot \bar{\lambda}p) \odot \bar{\lambda}q) \not\approx (\lambda x.x \odot (\bar{\lambda}p \odot \bar{\lambda}q))$$

since the first term – which is $(\mathsf{I}\,p)q$, where $\mathsf{I} = \lambda x.x$ is the usual identity combinator – can be reduced to (pq) (up to $\equiv$) while the second one, which could be written $\mathsf{I} \odot (\bar{\lambda} p \odot \bar{\lambda} q)$, cannot perform any computation.

We conjecture that the observational equivalence $\approx$ is a congruence. A consequence would be that we cannot define in the γ-calculus the CCS *sum* $(p+q)$, whose semantics is given by:

$$p \xrightarrow{a} p' \;\vdash\; (p+q) \xrightarrow{a} p'$$

$$q \xrightarrow{b} q' \;\vdash\; (p+q) \xrightarrow{b} q'$$

More precisely, if $\approx$ is a congruence then there is no γ-term r such that $(rp)q \approx (p+q)$ for all p and q, since $\approx$ is not a congruence with respect to CCS sum – for instance we have $(\mathsf{I}\,\mathbb{1}) \approx \mathbb{1}$ but $\mathsf{I} + (\mathsf{I}\,\mathbb{1}) \not\approx \mathsf{I} + \mathbb{1}$. Note that in general we have $(p \oplus \mathbb{1}) \not\approx p$ – take for example $p = \mathsf{I}$.

To conclude this note, let us briefly discuss the relationships between observational equivalence and some λ-theories (*cf.* [2]). First we should note that observational equivalence – which is consistent on Λ: $\mathsf{I} \not\approx \Omega$ – is not extensional. This means that the equation η (on λ-terms)

$$\lambda x.Mx = M \qquad x \text{ not free in } M$$

is not valid for $\approx$ – for instance $\lambda x.\Omega x \not\approx \Omega$, since the first term has a possible communication with its environment, while the second has no communication capability. For what regards β-convertibility $=_\beta$ of λ-terms, one could prove that β-convertible λ-terms are observationally equivalent, that is

$$\forall M, N \in \Lambda \;\; M =_\beta N \;\Rightarrow\; \theta(M) \approx \theta(N)$$

This implies a restricted kind of η-conversion: if M is really a function, that is if there exists N such that $M \xrightarrow{*} \lambda z.N$, then we have $\lambda x.Mx \approx M$ (for x not free in M). The converse of the previous implication is not true: observational equivalence is strictly weaker than β-conversion. To see this, let us say that a γ-term p is *locked* if it has no communication capability, and can never terminate, that is if $p \overset{a}{\Longrightarrow} p' \;\Rightarrow\; a = \tau \;\&\; p' \not\equiv \mathbb{1}$. It is easy to see that any two locked terms are equivalent – note that if p is locked and $p \overset{a}{\Longrightarrow} p'$ then p' is locked as well. Then for instance we have $\mathsf{YY} \approx \Omega$, where Y is the standard fixed point combinator $\lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$, while YY and Ω are not convertible. One should note that the two terms Ω and YY are *unsolvable* (*cf.* [2] for this notion). But an unsolvable λ-term is not necessarily locked, and observational equivalence does not equate all the unsolvables – for instance $\lambda x.\Omega$ is unsolvable, but not locked, and we saw that $\lambda x.\Omega \not\approx \Omega$. Then the observational equivalence of λ-terms is quite different from the usual semantics of the λ-calculus, and one may wonder whether it is too discriminating. On the other hand, it seems to be very close to the semantics given by Abramsky in [1] (and to the one given by Lévy in [3]), and the relationship between observational equivalence and Abramsky's applicative bisimulation should be established. This question is left for further investigation.

REFERENCES

[1] S. ABRAMSKY, *The Lazy Lambda-Calculus*, to appear in Declarative Programming, Ed. D. Turner, Addison Wesley (1988).

[2] H. P. BARENDREGT, *The Lambda Calculus*, Studies in Logic 103, North-Holland (1981).

[3] J.-J. LÉVY, *An Algebraic Interpretation of the $\lambda\beta K$-Calculus; and an Application of a Labelled λ-Calculus*, Theoretical Comput. Sci. 2 (1976) 97-114.

[4] R. MILNER, *A Calculus of Communicating Systems*, Lecture Notes in Comput. Sci. 92 (1980) reprinted in Report ECS-LFCS-86-7, Edinburgh University.

[5] A. STOUGHTON, *Substitution Revisited*, Comput. Sci. Rep. 1-87, University of Sussex (1987).