



**Dipartimento di Informatica
Università degli Studi di Verona**

**Rapporto di ricerca 15/2004
Research report**

March 2004

An interaction calculus for concurrent systems

Partially supported by MURST project PROTOCOLLO 2002
and the *Università degli Studi di Verona* ■

Ugo Solitro

Questo rapporto è disponibile su Web all'indirizzo:
This report is available on the web at the address:
<http://www.di.univr.it/report>

Abstract

In this work we describe a simple calculus (called interaction calculus or **Calculus*) for the representation of concurrent systems. In this a system is collection of expressions (processes) that share a working space; their computational behaviour is determined by the interaction of processes.

The calculus is an attempt to describe concurrent systems by means of a “non functional” calculus which is, in some sense, strictly related with the lambda-calculus: computations are carried out by substitutions, but in our calculus they are originated by a symmetric interaction between two expressions, instead of the functional application of an operator to its operand. In this way we lose some good features of lambda calculus (the confluence property for instance), but we gain the capability of representing concurrency and mobility; all the same, we will discover that functions can be nicely encoded in the calculus.

Keywords: concurrency, interaction, linear logic, functional calculi

Introduction.

In this work we call *concurrent system* (or just a *system*) a collection of expression that represent *processes* sharing a common environment; some of the processes can interact together and, as a result of the interaction, modify themselves and their environment. A process itself can be a concurrent system; consequently a system is organized hierarchically in environments that have local resources and can communicate together by means of common devices.

As in π -calculus ([MPW92, Mil93]) we would like to provide an simple and powerful *milieu* for the study of concurrency, but here the focus is on the interaction instead of message passing.

The basic engine of the calculus has been inspired by the computational behaviour of proof-nets in *linear logic* [Gir87] and appears for the first time in [Sol89] (later developed in [Sol90, SV93]). A similar calculus has been presented in a paper by Abramsky [Abr94] as an encoding of proofs of linear logic; on the other hand in [BS94] Bellin and Scott show that linear logic proof can be also encoded in a variant of the π -calculus. But in the recent developments of our work ([PS00, PSB00]) we are mainly interested in the interaction process instead of logics; in this respect we took some inspirations comes from the CHAM [BB92] and from a paper of Lafont [Laf97] where is presented a computational system based on interacting nets that is not strictly related with the logic.

In this paper we first introduce a simple version of the calculus; the *Basic *Calculus* is an untyped calculus without explicit functional features. Then the language is extended with a minimal functional apparatus in order to achieve a faithful and natural interpretation of λ -calculus.

ATOMS	
$\frac{c \in \text{Const}}{c \in \mathbb{S}}$	$\frac{x \in \text{Var}}{x \in \mathbb{S}}$
PAIRS AND COUPLES	
$\frac{t_1, t_2 \in \mathbb{S}}{t_1 \cdot t_2 \in \mathbb{S}}$	$\frac{t_1, t_2 \in \mathbb{S}}{t_1 * t_2 \in \mathbb{S}}$
ENVIRONMENTS and BINDING	
$\frac{t_1, \dots, t_m \in \mathbb{S} \quad (m > 0)}{\{t_1, \dots, t_m\} \in \mathbb{S}}$	$\frac{x \in \text{Var}, \quad t \in \mathbb{S}}{\nu x. t \in \mathbb{S}}$

Table 1: FORMATION RULES.

1 The *Basic* *Calculus.

A generic *system* is a collection (actually a multiset) of processes: elementary, structured and interacting processes. As a result of their interaction, two processes can produce other interactions or modify the environment in which they stay.

1.1 The syntax.

Definition 1.1 (The Abstract Syntax of *Basic* *Calculus) *Systems* are built in *Basic* *Calculus using

- a denumerable set Var of variables ranged over $x_0, x_1, \dots$,
- a finite set $\text{Const} = \{0, 1, \varepsilon\}$ of constants,
- two binary constructors $\cdot$ (for **pairs**) and $*$ (for **couples**),
- the **restriction** operator ν .,
- and the finitary **environment** constructor $\{ \}$.

Formally the set $\mathbb{S}$ of expressions that represent systems, usually called **terms**, is inductively defined by the rules of Table 1.

In the concrete syntax parenthesis are allowed in order to solve the ambiguities.

We call (general) *environment* an expression like this

$$\nu x_1. \nu x_2. \dots \nu x_k. \{t_1, \dots, t_n\} \quad (\text{env})$$

with $k \geq 0$; It actually represent a collection of *processes* $t_1, \dots$ together with some declarations $\nu x_1. \dots$ of the variables that are local (bound, restricted) for the environment. Since $\{t_1, \dots, t_n\}$ is a multi-set the order of elements is not relevant, but repetitions (if any) are important.

A system can be an general environment or a *process*, namely

a pair $t_1 \cdot t_2$ that is a rigidly structured construction made up on two *sub-processes*;

a couple $t_1 * t_2$, where the *sub-processes* t_1 and t_2 are ready to interact;

or an atom,

that is a variable (whose role will be explained by the computation rules) or one of the constants:

- **1** is a *no operation process*, that is the interaction with it does nothing;
- **0** is a *process killer*: if a process interacts with **0** it will be destroyed;
- ε is a dead process, the outcome of an interaction with the killer process.

Example 1.1 *The system described by the following term*

$$\nu x. \nu y. \{y, \mathbf{0}, x * z, y, x \cdot z, \mathbf{1} * y\}$$

*represent a collection of five processes; three of them are static at the moment, but $x * z$ and $\mathbf{1} * y$ are ready to interact; in this case the variable z is not restricted and it represents something external to the current environment.*

Convention 1 *A few agreement about notation could be helpful.*

- We will use p, q, r, s, t as metavariables ranging over the set of terms.
- We will use **boldface** to represent generic lists of expression: hence instead $e_1, \dots, e_n$ we can write $\vec{e}$; also $\nu \vec{x}. \{\vec{t}\}$ will be a shorthand for $\nu x_1. \nu x_2. \dots \nu x_k. \{t_1, \dots, t_n\}$
- Since the idea of environment is naturally connected with the restriction we will often use a more compact notation for a generic environment like (env) , that is

$$\{x_1, \dots, x_k \mid t_1, \dots, t_n\};$$

moreover, adopting the convention on lists, we can write $\{\vec{x} \mid \vec{t}\}$.

- We agree that pairing and coupling operators associate on the left and the coupling operator has a lower priority.
- Finally, if Γ is a generic expression (not necessarily a term of the **Calculus*), $\Gamma(x)$ means that the variable x may occur in Γ .

The concept of free and bound occurrence of variables is formalized as usual.

Definition 1.2 Let $\text{var}(t)$ the set of all variables that appear in $t \in \mathbb{S}$.

An occurrence of a variable x is **bound** in t if and only if it appears in $\nu x. t'$ which is a subterm of t ; it is said to be **free** otherwise.

A term is **closed** when all its variables are bound.

The Definition 1.2 can be refrased in an inductive way as shown here below.

Corollary 1.1

1. The set of **bound variables** can be defined by induction:

$$\begin{aligned} \text{bv}(x) &= \text{bv}(\mathbf{0}) = \text{bv}(\mathbf{1}) = \text{bv}(\varepsilon) = \emptyset \\ \text{bv}(t \cdot t') &= \text{bv}(t * t') = \text{bv}(t) \cup \text{bv}(t') \\ \text{bv}(\nu x. t) &= \{x\} \cup \text{bv}(t) \\ \text{bv}(\{ t_1, \dots, t_n \}) &= \bigcup_{i=1}^n \text{bv}(t_i) \end{aligned}$$

2. If no bound variable occur also free in a term t , the set of free variables can be defined by

$$\text{fv}(t) = \text{var}(t) \setminus \text{bv}(t).$$

The **substitution** is defined in the common way: so if $t, t' \in \mathbb{S}$,

$$t[t'/x]$$

is the term obtained by the textual substitution of t' for every free occurrence of the variable x in t with the obvious restrictions:

- no free variable become bound,
- no bound variable become free.

1.2 The computations.

We illustrate now the operational semantics which is defined as a rewriting system.

As we have already stated, a system performs a computation by means of the interaction between processes; there are essentially three form of interaction:

- a process interacting with a variable is spread out in the environment where that variable is defined;
- two structured processes (pairs) interact and produce more elementary interactions.
- finally the interaction with a constant process;

The different forms of interaction are described by specific computational rules, but we also need a few rules that help us to get rid of the syntax limitations.

1. <i>renaming</i> :	if $y \notin fv(t)$,	$\nu x. t$	$\rightleftharpoons$	$\nu y. [y/x]t$
2. <i>*-comm</i> :		$p * q$	$\rightleftharpoons$	$q * p$
3. ν - <i>comm</i> :		$\nu x. \nu y. t$	$\rightleftharpoons$	$\nu y. \nu x. t$

Table 2: STRUCTURAL RULES.

1.2.1 Structural rules.

Pure structural rules are described in Table 2 and talk us about different expression that actually represent the same system. The rules can be applied in a direction way or in the other according to the direction of the displayed arrows: all pure structural rules can be applied in both directions.

Remark 1.1 What do precisely do the pure structural rules?

1. *The first rule is the common renaming.*
2. *The second one say that interaction is a commutative, namely the order of interacting processes is not important.*
3. *The last one says that the order of successive restriction is not relevant¹.*

Example 1.2

- $\nu x. \{x, y\} \rightarrow \nu z. \{z, y\}$
- $\nu x. \nu y. \{x, y\} \rightleftharpoons \nu y. \nu x. \{x, y\}$

Hereafter we forget the differences between terms that can be converted using pure structurally rules.

Definition 1.3

The **structural equality** is the equivalence relation on terms generated by the rules of Table 2.

More precisely, $t, t' \in \mathbb{S}$ are **structurally equal**, and write

$$t \equiv t',$$

if there is a finite sequence of applications of pure structural rule that transform t into t' .

¹We will see that restriction is not λ -abstraction! A restricted variable is just a local variable.

(basic)	$\nu x. \Gamma[t' * x]$	$\mapsto$	$\nu x. \Gamma[t'/x]$
(split)	$(p \cdot q) * (r \cdot s)$	$\mapsto$	$p * r, q * s$
(zero, one)	$\mathbf{0} * t \mapsto \varepsilon$		$t * \mathbf{1} \mapsto t$

Table 3: INTERACTION RULES.

Example 1.3

$$\begin{aligned}
\nu x. \{x, y\} &\equiv \nu z. \{z, y\}, \\
\nu x. \nu y. \{x, y\} &\equiv \nu y. \nu x. \{x, y\} \\
\nu x. \nu y. \{x, y, x * y\} &\equiv \nu y. \nu x. \{x, y, y * x\}
\end{aligned}$$

Remark 1.2 *Structural equality can also be used to extend the language. Assume that we want to introduce a new symbol M as a shorthand of t ; writing $M \equiv t$ we mean that the symbol M has been added to the language and the structural equality is extended by means of the previous equation.*

1.2.2 Computational rules.

The dynamic evolution of a system is described in the **Calculus* by means of the **interaction rules** that are shown in Table 3.

Note that all the interaction rules have a direction: this means that they are not reversible.

basic: this rule recall the β -conversion of λ -calculus, but the restriction operator does not disappear after the application of the rule!

split: an interaction between structured terms produces two simpler interactions.

constants: these rule explain in formal terms the meaning of the constant processes; there is no rule about ε .

Definition 1.4 (Computation) *We say that t **reduces** to t' , and we write $t \rightarrow t'$ if there is a finite sequence of application of interaction rules that transform t into t' (modulo structural equality and its extensions²).*

Remark 1.3 *In general the calculus is not confluent, as show in the examples below.*

²Namely the essential equality that we will define later.

Example 1.4 (Non deterministic choice with constants)

The expression

$$\{[x, y] \mathbf{1} \cdot \mathbf{0} * x, \mathbf{0} \cdot \mathbf{1} * x, x * p \cdot q\}$$

can reduce either to $\{p\}$, or to $\{q\}$ or to $\{p, q\}$, depending on how we (base) rule. The sub-expressions p and q are generic and, in particular, they could be unreducible.

The expression above represent a process able to make a non deterministic choice.

Remark 1.4 In the previous example constants plays an essential role. But are they really necessary? We could think about a calculus without constants, but they turns out to be useful in many examples.

Remark 1.5 If we consider interactions like $\mathbf{0} * \mathbf{1}$ which rule should be applied? (zero) or (one)? We get quite different results! We think that it should be reasonable associating different priorities to the rules³, but we do not intend to discuss now the question.

Example 1.5 Assume that $p \equiv p(x, y)$ and $q \equiv q(x)$.

The system represented by $t = \{[x, y] p(x, y), x * y, x * q(x)\}$ can reduce as follows

$$t \mapsto \{[x, y] p(y, y), y * q(y)\} \mapsto \{[x, y] p(q(y), q(y))\};$$

but also:

$$t \mapsto \{[x, y] p(q(y), y), q(y) * y\} \mapsto \{[x, y] p(q(q(y)), q(y))\}$$

There is no rule that allows an explicit interaction between environments: a term like $t * \{\vec{p}\}$ cannot reduce unless t is a variable or a constant.

In principle an environment is a sort of closed world that communicate with outer space by means of interaction. But up to now we lack of an effective control on communication among environments: the following examples show how a “dangerous” behaviour can arise.

Example 1.6 (Capture!) A well known situation is when a free variable becomes bound after a reduction:

$$\{[x] t(y) * x, [y] x\} \rightarrow \{[x, y] t(y)\}$$

which can be avoided with a renaming so that the condition about substitution are respected.

³In this work we assume that (basic) rule has the highest priority.

1. (ghosts)	$\{\vec{t}, \varepsilon\}$	$\rightarrow$	$\{\vec{t}\}$
2. (boundaries):	$\{\{\vec{t}\}\}$	$\rightarrow \{\vec{t}\},$	$\{t\} \rightleftharpoons t$
3. (useless names):	if $x \notin fv(t),$	$\nu x. t$	$\rightarrow t$
4. (scope):	if $x \notin fv(t),$	$\{\vec{t}, \nu x. \{\vec{t}'\}\}$	$\rightleftharpoons \nu x. \{\vec{t}, \vec{t}'\}$

Table 4: SEMI-STRUCTURAL RULES.

Example 1.7 (Escape?) *If we reduce the expression $\{x\} x, \{y\} t(y) * x \}$ we get $\{x\} t(y), \{y\} \}$ where y is no more bound! This is hardly acceptable.*

How to get rid of these problems?

In the first case a simple renaming can solve the problem, but for the second one we need something more.

1.2.3 Semi-structural rules

The *semi-structural rules*, defined in Table 4, allow a better control of the environments⁴ and prevent disagreeable situations like that described in Example 1.7.

Remark 1.6 *The intuitive role of the semi-structural rules can be easily justified.*

1. *A ghost process is just a placeholder of a dead process and can harmlessly be removed.*
2. *An environment with only one process is worthless if there is no binding associated to it.*
3. *It is possible to remove any useless restriction.*
4. *The boundary of an environment can be moved provided that the sets of free and bound variables are “preserved”.*

In particular, the (scoping) rule can be used to solve the problem of Example 1.7:

$$\{x\} x, \{y\} t(y) * x \} \rightarrow \{x, y\} x, t(y) * x \} \mapsto \{x, y\} t(y) \}$$

Structural and semi-structural rules identify system that are different only because of the concrete syntax or non essential differences.

⁴This group of rules are in some sense halfway between structural rules and interaction rules.

Definition 1.5 (Essential equivalence) *The **essential equivalence** is the smallest equivalence relation that extend structural equality and the semi-structural rules (interpreted as a relation on terms).*

*We say that two terms t, t' are **essentially equivalent** (or simply equivalent) and write $t = t'$, if there is a finite sequence of terms*

$$t \equiv t_0, t_1, \dots, t_n \equiv t'$$

such that for any $i < n$, t_{i+1} is obtain from t_i using a semi-structural rule or viceversa.

Remark 1.7 *The direction of arrows that appear in some of the rules in Table 4 has no practical effect in the essential equality, but it should be respected in any reasonable reduction process.*

$\frac{x \in \text{Var}}{x \in \Lambda}$	$\frac{x \in \text{Var} \quad M \in \Lambda}{(\lambda x.M) \in \Lambda}$
$\frac{M \in \Lambda \quad N \in \Lambda}{(MN) \in \Lambda}$	

Table 5: THE λ -TERMS

2 Functions in **Calculus*.

In this section we show how it is possible to describe functions in **Calculus*: we will be able to define a natural translation of λ -calculus with a few *addenda* to the basic version of the calculus. The new features of the calculus add capabilities in controlling the communication and interaction between environments.

2.1 The naïf translation of λ -calculus.

The set Λ of all λ -**terms** (well formed expression of λ -calculus) is defined by the formation rules of Table 5. Free and bound variables are defined in the usual way: so any occurrence of variable x in $M \in \Lambda$ become bound in $\lambda x.M$ because of the λ -abstraction. The structural equality on Λ is defined only by means α -conversion (*renaming* of bound variables): λ -terms that differ only by the name of bound variables are considered (structurally) equal. The unique computational rule that we consider here is the well known β -conversion:

$$(\lambda x.M)N \mapsto [N/x]M \quad (\beta)$$

A β -reduction can be represented in **Calculus* as an interaction between the formal parameter and its corresponding actual parameter. In a term like $\lambda x.M$ the λ abstraction is similar to restriction because it binds the name of a variable, but occurrence of the variable x in the prefix λx is also the formal parameter of the function that have to interact with its argument. We propose to encode the λ -term (actually a function) $\lambda x.M$ as a system like

$$\nu x. \{x, \mathbb{M}\}$$

(where $\mathbb{M}$ is the translation of M); and the application of the function to an actual parameter N (*redex* of β -conversion) can be represented by

$$\nu x. \{x * N, \mathbb{M}\}.$$

It is possible now to apply the interaction rules of **Calculus* so that

$$\nu x. \{ x * N, M \} \mapsto \nu x. \{ [N/x]M \} \multimap \{ [N/x]M \}$$

and the result seems exactly what we expect.

But unfortunately things do not work so easily!

Let us now formally define the understood (and simple-minded) translation. If $T \in \Lambda$ its encoding $\varphi(T)$ as a term of **Calculus* can be inductively defined as follows:

- if $T \equiv x$ then $\varphi(M) = x$;
- if $T \equiv \lambda x.M$ then $\varphi(T) = \{ x \} x, \varphi(M) \}$;
- if $T \equiv (\lambda x.M)N$ then $\varphi(T) = \{ x \} x * \varphi(N), \varphi(M) \}$.

The examples that follow will exhibit a few problems.

Example 2.1 The λ -term $\lambda x.x$ is translated by φ in the following system $\nu x. \{ x, x \}$; but if we do know how the expression is obtained, there is nothing that tell us which occurrence of the variable x represent the formal parameter and which is the body of the function.

Example 2.2 Consider now the λ -term $(\lambda x.M)y$; its translation in **Calculus* is

$$\nu x. \{ x * y, M \}$$

where the process $x * y$ is ready for interaction. There are at least two possible reduction sequences

$$\begin{aligned} \nu x. \{ x * y, M \} &\mapsto \nu x. \{ M[y/x] \} \multimap \{ M[y/x] \} \multimap M[y/x] \\ \nu x. \{ x * y, M \} &\mapsto \nu x. \{ M[x/y] \} \end{aligned}$$

The first one achieve the desired result; but the second one is clearly “wrong” in the sense that it do not correspond to β -reduction.

Example 2.3

The translation φ is not well defined, since it cannot be applied to a general λ -term like MN !

It is clear now that φ is not adequate: even if one could think to overcome the difficulties shown in the Examples 2.1 and 2.2 by means a smarter translation and a weak notion of representability, the problem exhibited in Example 2.3 seemingly do not have a natural solution in the basic version of **Calculus*.

An acceptable solution can be achieved extentending the calculus as illustrate below.

2.2 Polarization.

First of all we allow the **polarization** of variables. An occurrence of a variable can be receive a positive or negative polarization: **positive** occurrences represent an *output* and **negative** ones stand *input*. The same variable can have positive and negative occurrences at the same time, and also neutral occurrences (that is occurrences without polarization).

Definition 2.1 *Formally, the language of λ -Calculus is extended with the following two formation rules:*

$$\frac{x \in \text{Var}}{x^- \in \mathbb{S}} \qquad \frac{y \in \text{Var}}{y^+ \in \mathbb{S}} \qquad (\text{polarity})$$

As regard to computations, beside to the basic interaction rule, we have also

$$\begin{aligned} \nu x. \{ \vec{t}, t' * x^- \} &\mapsto \nu x. \{ \vec{t}[t'/x] \} & (\text{base}^-) \\ \nu x. \{ \vec{t}, t' * x^+ \} &\not\mapsto \nu x. \{ \vec{t}[t'/x] \} & (\text{base}^+) \end{aligned}$$

where

- the first rule is almost the original one,
- but the second one is a sort of negative rule since it says that a particular interaction is forbidden!

Example 2.4 *We have now a nice representation of the identity function $\lambda x.x$:*

$$\{ [x] x^-, x^+ \}$$

where it is clear which role every occurrence of variable x plays.

But polarization is useful in a more general way as the next example shows.

Example 2.5 (Non deterministic choice revisited)

Using the polarities we can describe a nice variant of the term defined in the Example 1.4 that give us a better control on the behaviour of the computations:

$$\{ [x, y] \mathbf{1} \cdot \mathbf{0} * x, \mathbf{0} \cdot \mathbf{1} * x, x^+ * p \cdot q \}.$$

The new term can reduce either to $\{ p \}$, or to $\{ q \}$, but not to $\{ p, q \}$.

Example 2.6 *A better translation of λ -term $(\lambda x.M)y$ could be*

$$\nu x. \{ x^- * y^+, \mathbb{M} \}$$

because the only reduction allowed is now

$$\nu x. \{ x^- * y^+, \mathbb{M} \} \mapsto \nu x. \{ \mathbb{M}[y/x] \} \rightarrow \{ \mathbb{M}[y/x] \} \rightarrow \mathbb{M}[y/x].$$

2.3 Observables.

We are not able yet to get rid of the translation of general application (see Example 2.3). We have already observed that no device that allows a general interaction between environment; it is reasonable to think that some of the components can be visible output of an environment so that we can interact with them. The second new feature we propose is related to the visibility of processes outside side their environment: we will prove that we can manage the most general case of functional if reveal certain occurrences of a variable.

We have already pointed out that negative occurrences of a variable in certain circumstances can be considered as formal parameters. Given an environment we decide that one negative occurrence of a bound variable can be accessible from outside the environment; we call these special occurrences *public* or **observable**.

The language of **Calculus* could be simply extended with the possibility of marking as observable some terms of an environment: so a generic environment will be partitioned in *public* and *private* processes; but in this work we opt for less general choice

Definition 2.2 *The language of *Calculus is enriched with the new formation rule*

$$\frac{x \in \text{Var} \quad p \in \mathbb{S}}{[x]p \in \mathbb{S}};$$

all the conventions already stated are extend in a natural way .

An environment can now have the following general aspect:

$$[x_1, \dots, x_r] \{ [z_1, \dots, z_s] t_1, \dots, t_n \} \equiv [x_1] \dots [x_r] \{ [z_1, \dots, z_s] t_1, \dots, t_n \}$$

where the x_i represent the formal parameters the z_j the local variables and the multiset of t_h the body of the environment⁵. The relation with λ -terms is quite clear: if t is represented by the system p , the function $\lambda x. t$ will be encoded by $[x]p$.

Remark 2.1 *The order of formal parameter (observable variables) is important!*

Remark 2.2 *A less “innovative” (and more general) notation for the new kind of environment could be*

$$\{ [x_1, \dots, x_r, z_1, \dots, z_s] x_1^-, \dots, x_r^-; t_1, \dots, t_n \}$$

The environment is divided in two part separated by a semicolon:

- *in the first part we have the ordered list of formal parameters,*

⁵Since we are dealing with functions, one can think that we have also to mark the output result; actually it turns out that this is not necessary for the representation of λ -terms.

- and in the second part we find the (private) processes.

This more general setting will be discussed in the future work.

We have still to face the problem the encoding of a general application. The idea is that if a process (a function) can reduce to an expression like $[x]t$, it should be possible to apply it (using the interaction) to another process t' .

Definition 2.3 If $t, t' \in \mathbb{S}$ we can **apply** t to t' according to the following new rule:

$$\frac{t, t' \in \mathbb{S}}{\langle t, t' \rangle \in \mathbb{S}} \quad (\text{weak interaction})$$

If t is (or reduces to) a functional process $[x]t''$ the interaction is caused by the new semi-structural rule

$$\langle [x]t'', t' \rangle \rightarrow \{x\} t' * x^-, t'' \} \quad (\text{apply})$$

Weak interaction is a sort of “waiting interaction” that can be activated the left process reaches the necessary functional “state”. Using the extended language we can define useful processes.

Example 2.7 (Non deterministic choice, again!) The following functional term

$$\text{Choose} \equiv [z]\{x\} \mathbf{1} \cdot \mathbf{0} * x, \mathbf{0} \cdot \mathbf{1} * x, x^+ * z \}$$

is a sort of general operator that makes non deterministic choices: it is easy to see that $\langle \text{Choose}, p \cdot q \rangle$ reduce either p or q , but not $p \cdot q$.

With these new devices we can easily represent functions and in particular some well known λ -terms.

Example 2.8 Consider the translation of the constant combinator $\mathbf{K} = \lambda xy.x$

$$\llbracket \mathbf{K} \rrbracket = [x]\{[y]\{x^+\} \} \}.$$

If $\llbracket M \rrbracket = p$ and $\llbracket N \rrbracket = q$ we have

$$\begin{aligned} \llbracket \mathbf{KMN} \rrbracket &= \langle \langle [x]\{[y]\{x^+\} \} \}, p \rangle, q \rangle \\ (\text{apply}) &\rightarrow \nu x. \{x * q, \langle [y]\{x^+\} \}, p \} \\ (\text{base}) &\mapsto \nu x. \{ \langle [y]\{q\} \}, p \} \\ (\text{apply}) &\rightarrow \nu x. \{ \nu y. \{y * p, q\} \} \\ (\text{base}) &\mapsto \nu x. \{ \nu y. \{q\} \} \\ (\text{structurals}) &\rightarrow^* q = \llbracket M \rrbracket \end{aligned}$$

We have now the tools necessary to an acceptable encoding of λ -terms.

Definition 2.4 (λ -Interpretation)

We define the **interpretation** $\llbracket - \rrbracket : \Lambda \longrightarrow \mathbb{S}$ of a λ -term in the * Calculus by induction on the construction of a λ -term:

$$\begin{aligned}\llbracket x \rrbracket &= x^+ \\ \llbracket \lambda x.M \rrbracket &= [x] \llbracket M \rrbracket \\ \llbracket MN \rrbracket &= \langle \llbracket M \rrbracket, \llbracket N \rrbracket \rangle\end{aligned}$$

Lemma 2.1

β -conversion is representable in * Calculus, namely

$$\llbracket (\lambda x.t)t' \rrbracket \rightarrow \llbracket t[t'/x] \rrbracket$$

PROOF.

$$\begin{aligned}\llbracket (\lambda x.t)t' \rrbracket &= \langle [x] \{ \llbracket t \rrbracket \}, \llbracket t' \rrbracket \rangle \\ &\rightarrow \{ [x] \llbracket t' \rrbracket * x^-, \llbracket t \rrbracket \} \mapsto \{ [x] \llbracket t \rrbracket \llbracket t' \rrbracket / x \} \rightarrow \{ \llbracket t \rrbracket \llbracket t' \rrbracket / x \} \\ &\rightarrow \llbracket t \rrbracket \llbracket t' \rrbracket / x = \llbracket t[t'/x] \rrbracket\end{aligned}$$

□

Definition 2.5 p **reduces elementarily** to p' , and we will write

$$p \rightarrow_1 p',$$

when just one instance of an interaction rule is used.

An **elementary step** of reduction is any finite sequence of application of rules with the restriction that only one interaction rule has been used.

Lemma 2.2

Let $p = \llbracket t \rrbracket$ for a term $t \in \Lambda$ and assume that $p \rightarrow_1 q$
There is a $t' \in \Lambda$ such that

$$q = \llbracket t' \rrbracket \quad \text{and} \quad t \rightarrow t'$$

PROOF. The proof is quite easy.

Even so we observe is that if $p \rightarrow_1 q$ it is possible the semi-structural rule (apply) could have been used more then once. Assume that the following semi-structural reduction of a sub-process of p

$$\langle [x]r, s \rangle \rightarrow \{ [x] s * x^-, r \}$$

and it is not involved in any interaction in the elementary step in consideration.

A problem could arise if the unique interaction of the elementary step require the substitution of displayed negative occurrence of the variable x ; but this is not the case since we have assumed that p is a correct translation of a λ -term: hence a conflict of this kind can be always solved by a renaming.

□

The next theorem is an immediate consequence of the previous lemmata.

Theorem 2.1 (λ -Representation)

1. If t is a λ -term and $t \rightarrow t'$, then $\llbracket t \rrbracket \rightarrow \llbracket t' \rrbracket$
2. Viceversa, assume $p = \llbracket t \rrbracket$ and $p \rightarrow p'$.
Then there is λ -term t' such that $t' = \llbracket p' \rrbracket$ and $t \rightarrow t'$.

Using the capability of representing function **Calculus* inherits effortlessly many useful tools from the λ -calculus: combinators, functional fixed-point operators, representations of numbers, ... In particular the next example shows a term that is an adaptation of a well know fixed point operator of λ -calculus.

Example 2.9 Let $h \equiv [x][y]\{y, \langle x, \langle x, y \rangle \rangle\}$ and $H \equiv \langle h, h \rangle$. Then for any process p

$$\{ \langle H, p \rangle \} \rightarrow \{ p, \langle H, p \rangle \}$$

In fact we have that

$$\begin{aligned} \{ \langle H, p \rangle \} &\equiv \{ \langle \langle h, h \rangle, p \rangle \} \equiv \{ \langle \langle [x][y]\{y, \langle x, \langle x, y \rangle \rangle\}, h \rangle, p \rangle \} \\ &\rightarrow_1 \{ \langle [y]\{y, \langle h, \langle h, y \rangle \rangle\}, p \rangle \} \rightarrow_1 \{ \{ p, \langle h, \langle h, p \rangle \rangle \} \} \\ &\rightarrow \{ p, \langle h, \langle h, p \rangle \rangle \} = \{ p, \langle H, p \rangle \} \end{aligned}$$

Conclusions.

The basic version of the **Calculus* presented in this paper is extremely simple and expressive enough. It provides an elementary framework in which it is possible to deal with concurrency and mobility, but, with some very reasonable additions, allows a natural treatment of functions; more the extended version of the **Calculus* enable also a good control of the interaction between environment.

Anyway a lot of work have still to be done. We are planning to show that the calculus can be tailored in order to provide a meaningful representation of logical proofs (of linear logic proofs and others) in a non functional setting; the problem of a general type system based on **Calculus* has to be investigated and developed; in particular the relation between this type system, the classical theory of types and logical systems must be extensively studied.

We plan also to study a type system for the **Calculus* in which types play a more relevant and active role in controlling computations. It is conceivable that in a future version of the calculus interactions are not necessarily defined inside a system in an explicit way.

The role of substitution could be also discussed: in the present definition of the calculus the effect of a basic interaction is a sort of “broadcasting” substitution which sometimes is not desirable. Again an adequate types could be useful in controlling also this aspect.

References

- [Abr94] S. Abramsky. Proofs as processes. *Theoretical Computer Science*, 135, 1994.
- [BB92] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.
- [BS94] G. Bellin and P. Scott. Remarks on the π -calculus and linear logics. *Theoretical Computer Science*, 135:11–65, 1994.
- [Gir87] J. Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.
- [Laf97] Yves Lafont. Interaction combinators. *Information and Computation*, 137:69–101, 1997.
- [Mil93] R. Milner. The polyadic π -calculus: a tutorial. In F.L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification, Vol. 94 Nato ASI Series F*. Springer-Verlag, 1993.
- [MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (I and II). *Information and Computation*, 100(1):1–77, 1992.
- [PS00] Corrado Priami and Ugo Solitro. Functional features of a calculus for logic and concurrency. In Maria Alpuente, editor, *Functional and Logic Programming*, pages 135–143. Universidad Politcnica de Valencia, 2000.
- [PSB00] Corrado Priami, Ugo Solitro, and Cristina Borboni. A non functional calculus: linear logic and concurrency. Technical report, Università degli Studi di Verona, 2000.
- [Sol89] U. Solitro. A typed calculus based on a fragment of linear logic. *Theoretical Computer Science*, 68(3):333–342, 1989.
- [Sol90] U. Solitro. *La Logica Lineare come Sistema di Tipi per un Calcolo non Funzionale*. PhD thesis, Dipartimento di Scienze dell’Informazione, University of Milano, 1990.
- [SV93] U. Solitro and S. Valentini. Local computation in linear logic. *Mathematical Logic Quarterly*, 39:201–212, 1993.



University of Verona
Department of Computer Science
Strada Le Grazie, 15
I-37134 Verona
Italy

<http://www.di.univr.it>

