An Investigation on the Optimal Implementation of Processes

Cosimo Laneve

Technical Report UBLCS-96-1

January 1996

Department of Computer Science
University of Bologna
Piazza di Porta S. Donato, 5
40127 Bologna (Italy)
Recent Titles from the UBLCS Technical Report Series

94-7 Comparative Semantics of LO, S. Castellani, P. Ciancarini, April 1994.
94-20 Dynamic Allocation of Signature Files in Multiple-Disk Systems, P. Ciaccia, August 1994.
94-21 Parallel Independent Grid Files Based on a Dynamic Declustering Method Using Multiple Error Correcting Codes, P. Ciaccia, November 1994.
95-13 Optimal Multi-Block Read Schedules for Partitioned Signature Files, P. Ciaccia, August 1995.
An Investigation on the Optimal Implementation of Processes

Cosimo Laneve

Technical Report UBLCS-96-1

January 1996

Abstract

We undertake the analysis of Lévy's notion of family relation in a concurrent scenario by providing an optimal implementation of an higher-order, concurrent language. The language considered turns out to be a sub-language of (pure) Concurrent ML, where transmissible processes cannot show up any nondeterminism.
1 Introduction

In the last years a new class of languages is gaining prominence: the so-called higher order concurrent languages. Among them we recall Concurrent ML [13], Facile [4] and \( \pi \)-calculus [10]. These languages bring together two fundamental computational paradigms of computer science, functional programming and concurrency, with the aim of developing a support for applications requiring a combination of complex computations and concurrent/distributed computing. For instance, a manifest feature of higher order concurrent languages is the possibility to send complex data, such as processes and functions. This feature allows to describe in a direct way those facilities of distributed programming as code transmission.

One open problem in this area is how much the functional model of computation can be adapted for modeling (higher order) concurrent features. This problem is analyzed here through the looking glasses of the implementations. In particular we consider the algorithms already defined for implementing (pure) functional languages and study their adaptations in order to encompass concurrent facilities. The algorithms we study are the so-called optimal implementations [8, 5, 2], which are graph rewriting systems that have been provided for implementing the \( \lambda \)-calculus.

1.1 The notion of optimality in a concurrent scenario

The optimal implementation (in the sense of Lévy [9]) of a functional language relies on the principle that no work (i.e. no reduction) is ever duplicated. In particular two copies of the same expression are kept shared as much as possible in order to be evaluated at most once. Up to our knowledge, investigations about optimality have been never attempted in concurrency theory. This means that, foremost, we must clarify when an implementation of a concurrent language may be called “optimal”. To this aim, take the following prototype concurrent language:

\[
p ::= 0 \mid z \mid \gamma z.p \mid \overline{\gamma}p.p \mid \overline{p}.p
\]

where the parallel composition “\( \mid \)” is monoidal with 0 as identity; the receiver \( \gamma z.p \) binds the free occurrences of \( z \) in \( p \) and the sender \( \overline{\gamma}p.q \) transmits the process \( p \) along the channel \( \gamma \). The definitions of free variables and substitution are the standard ones (see [3]).

The computational engine is provided by the communication rule below:

\[
\gamma z.p[\overline{r}.q \rightarrow p^{r/z}].q
\]

where \( p^{r/z} \) denotes the substitution of \( r \) for the free occurrences of \( z \) in \( p \).

Observe that, in the above language, the communication rule is the unique mechanism for performing work. Therefore in this context “optimality” means “avoiding duplication of the same communication”. For instance, consider the process

\[
\begin{array}{c}
A \\
\gamma z.(x | z) | \overline{\gamma}(\lambda y.y | \overline{x}0).0 \\
B \\
\end{array}
\]

and perform the communication \( B \). We obtain the process

\[
\begin{array}{c}
A \\
\lambda y.y | \overline{x}0.0 \\
A \\
\lambda y.y | \overline{x}0.0 \\
\end{array}
\]

where the communication \( A \) has been duplicated. However the two communications \( A \) must be considered as two instances of the same internal computation and there is no sensible reason to perform it twice. An “optimal” implementation of the concurrent language should avoid such duplication. In functional languages this is usually achieved by means of suitable mechanisms keeping shared the expression \( \lambda y.y | \overline{x}0.0 \).
1.2 The nondeterminism is troublesome

Let’s go further on. Consider the process

\[
\gamma x. (x \mid x) \vdash (\lambda y. y \mid \lambda 0. 0) \mid \lambda p. q
\]

and perform again the communication \( B \). We obtain the process

\[
A \{ \lambda y. y \mid \lambda 0. 0 \} \mid \lambda p. q \]

where one of the two communications \( A \) may be “broken” by the attempt of establishing a communication involving the process \( \lambda p. q \). That is, the same subprocess, in this case \( \lambda y. y \), may be involved in different communications. Clearly, the evolution of a process depends, in general, on the choice performed at each step. This phenomenon, which misses in functional languages, is quite usual in concurrency theory and it is called (internal) nondeterminism.

The constraint that an optimal implementation must never duplicate work has subtle consequences when nondeterminism shows up. In particular, sharing communications at a given step of the computation means “centralizing the choices concerning them” (that is, performing all the choices at the same time). In the example above we have two possibilities: (a) recognizing that the communications \( A \) are copies and then firing them at once or (b) firing one of the copies of \( A \) and then performing \( C \) or \( D \). Choose the possibility (b). Once having performed one copy of \( A \), we are obliged to fire \( C \) or \( D \) otherwise we should invalidate the requirement of optimality. This means that the second copy of \( A \) must be “disallowed” in favour of \( C \) or \( D \). However the redexes \( C \) and \( D \) may be “virtual”, in the sense that they will become redexes in the future (because, for instance, the process \( \lambda p. q \) is inside the scope of a send or a receive). To this aim we must keep track of what communications have been performed in the past and forbid the contraction of copies. Pursuing this approach on, we have to admit processes which must be considered in normal form even if they have subterms which are redexes (because these redexes have been disallowed).

Keeping track of the communications which have been performed along a computation is surely expensive. This may be avoided by deciding that all the copies of a communication must be always kept shared, namely their evaluation must be performed at the same time (the computation is complete). This choice is surely debatable, since interesting computations may be left out. Take for instance the process

\[
\lambda y. 0 \mid \lambda 0. 0 \mid \lambda y. y \mid \lambda x. (x \mid x. 0) \mid (\lambda x. (x \mid x. 0)). 0
\]

This process never terminates if the two copies of \( A \) are kept shared. On the other hand, if the sharing is broken, the process evaluates to 0.

1.3 Avoiding troubles

At the present stage of our research we want to avoid any decision limiting the nondeterminism that a process may show up. Therefore we have decided to consider a syntactical restriction of the language described in Subsection 1.1, where nondeterminism cannot be shown by processes that are copied. This is obtained by constraining those processes that can be sent as messages to be synchronous (the possible parallelism forces the communications between sub-processes through an operation which resembles very much to function application). The more general operation of parallel composition is left for other processes.

Surely our decision is too drastic: the syntactical restrictions on the calculus make it scanty and may be weakened. This does not scare us too much, since the aims of this paper are:
1. to provide a direct extension of the standard optimal \( \lambda \)-calculus implementations to a (simple) concurrent language. In particular, we aim at enriching the BOHM evaluator [1] with concurrent primitives, in order to define an optimal implementation of a fragment of CML.

2. to start a deeper analysis of what means (and/or what implies) to not duplicate processes. In this respect, we have chosen a language that fit better with optimal implementations, leaving the problem of generalizing it to future research.

1.4 Outline of the paper

The outline of the paper is as follows. In Section 2 we present our higher order concurrent language and its communication mechanism. In the same section we also define a labeling which allows to formalize the notion of family, i.e. those communications which are copies, and of optimal implementation, i.e. implementations where families have unique representations. The implementation is described in Section 3 and its correctness and optimality is stated in Section 4 (the proofs are provided in the full paper). In Section 6 we discuss some future themes of research.

2 The calculus

Consider the set of terms generated by the following grammar:

\[
\begin{align*}
t & ::= 0 \mid x \mid \gamma x.t \mid \tau t.t \mid t^0.t \\
p & ::= t \mid \gamma x.p \mid \tau t.p \mid p^0.p \mid p \mid \cdots \mid p
\end{align*}
\]

where \( x \) stays for a generic variable, \( \gamma \) is a port into a set \( \Gamma \). The expressions generated by the non terminal \( t \) are called synchronous processes and are ranged over by \( t, s, \cdots \); the expressions generated by the non terminal \( p \) are called processes and ranged over by \( p, q, r, \cdots \). The process 0 is the terminated process; \( \gamma x.p \) is the process which waits for a message \( t \) on the channel \( \gamma \) and then behaves as \( p[t/x] \); \( \tau t.p \) is the process that sends a message \( t \) on the channel \( \gamma \) and then behaves as \( p \); \( p_1 \| \cdots \| p_k \) is the parallel composition of the processes \( p_1, \cdots, p_k \): their communication is not forced, in the sense that they may also interact with processes in the environment; \( p \| q \) is the synchronous parallel composition of the \( p \) and \( q \): they must always communicate. Notice that there is a family of parallel operators \( "\|" \), every one with a different ariety. For simplicity these operators are noticed in the same way.

Processes are taken up-to the congruence \( \equiv \) generated by the following axioms:

- \( p = q \), if \( p \) and \( q \) are \( \alpha \)-equivalent;
- \( p^0 = p, p^0 \| q = q \| p, (p_1 \| \cdots \| p_k) = p_1 \| \cdots \| p_k \);
- \( p \| 0 = p, p \| q = q \| p \).

Remark 2.1 Observe that the synchronous composition \( \| \) is a binary, non associative operation, which has 0 as identity. There are many analogies between \( \| \) and the usual function application. However, the main differences between these two operators are:

1. \( \| \) may deadlock. This is the case in the processes \( \gamma x.p \| \bar{\mu}.q (\gamma \neq \mu), \gamma x.p \| \mu_y.q \) or \( \bar{\tau}s.p \| \bar{\mu}.q \).
2. there is no ordering on the arguments of \( \| \).

In the following we shall use the definitions of substitution and free variables of a term \( M \) (notation \( \text{fv}(M) \)). These are the standard ones (see [3]).

The evaluation mechanism is provided by the following two rewriting rules:

\[
\begin{align*}
\gamma x.t \| \tau t'.s & \rightarrow t[t'/x] \| s \\
\gamma x.p \mid \tau t.q & \rightarrow p[t/x] \mid q
\end{align*}
\]

where the notation \( p[t/x] \) denotes the substitution of \( t \) for the free occurrences of \( x \) in \( p \).
Example 2.2 We may easily encode \( \lambda \)-calculus in the above language. It is enough to use synchronous \textit{par} instead of application. The encoding \( [ \_] \) is defined by:

\[
\begin{align*}
[z] &= z \\
[\lambda z. M] &= \lambda z. [M] \\
[MN] &= [M][N].0
\end{align*}
\]

Observe that a non commutative operator as function application is modeled through a commutative one (the \( [\_] \)). This is not problematic, since one of the two argument of the \( [\_] \) is a sender and the other is a (possible) receiver.

Remark 2.3 By the above example, it is clear that we may implement the following sub-language of Concurrent ML [13]:

\[
e ::= z \mid \text{fun } x \Rightarrow e \mid e e \mid \text{let } x = e \text{ in } e \mid \text{let rec } x = e \text{ in } e
\]

\[
p ::= e \mid ( ) \mid \text{fun } x \Rightarrow p \mid \text{let } x = p \text{ in } p \mid \text{channel() \mid send}(c,e);p \mid \text{accept}(c);p \mid \text{spawn}(p);\ldots;\text{spawn}(p)
\]

Of course we may encode larger parts of Concurrent ML, since our transmissible processes may deadlock. However the above language should give a strong flavour of the expressive power of our calculus.

Definition 2.4 Let \( p \) be a process. A redex in \( p \) is a subterm of shape \( \gamma x. p \mid \gamma l. q \) or \( \gamma x. p \parallel \gamma l. q \). We assume that redexes are uniquely identified by names \( u, v, w \).

2.1 The labeling

Let \( A \) be a countable set of atomic labels. The labels we shall consider are terms generated by the following grammar:

\[
\ell ::= A \mid \ell \cdot \ell' \mid \ell \mid [\ell]
\]

The labeled processes are the terms \( \ell \) generated by the grammar:

\[
c ::= \ell(0) \mid \ell(z) \mid \ell(\gamma z.c) \mid \ell(\gamma l.c.c) \mid \ell(\ell(c))\mid\ell(c)
\]

\[
e ::= c \mid \ell(\gamma z.e) \mid \ell(\gamma l.c.e) \mid \ell(e)\mid\ell(e)\ldots\mid\ell(e)
\]

Labeled processes are quotiented by the same axioms of the structural equivalence as for processes plus the axioms

\[
\ell \parallel \ell(0) = e \quad e \parallel \ell(0) = e \quad \ell(e \ldots e') = \ell(e) \ldots \ell(e') \quad \ell(\ell(e)) = \ell \cdot \ell(e)
\]

The rewriting rules of the labeled processes are:

\[
\ell(\gamma z.e) \parallel \ell'(\gamma z'.e'') \Rightarrow \ell \cdot \ell'(\ell(e)[\ell'(e')/z])\parallel[\ell \cdot \ell'(e'')]
\]

\[
\ell(\gamma z.e) \parallel \ell'(\gamma z'.e'') \Rightarrow \ell \cdot \ell'(\ell(e)[\ell'(e')/z])\parallel[\ell \cdot \ell'(e'')]
\]

where \( \ell \) and \( \ell' \) have no common prefix, namely there is no \( \ell' \) such that \( \ell = \ell' \cdot \ell_1 \) and \( \ell' = \ell \cdot \ell_1 \). The set \( \{ \ell, \ell' \} \) will be called label of interaction.

The rewriting rule involving par-composition deserves some comments. Due to the axiom

\[
\ell(\ell(e) = \ell(e)[\ell'(e')],\text{we have, for instance,}
\]

\[
\ell(\ell'(\gamma z.p) | \ell''(\gamma l.q)) = \ell \cdot \ell'(\gamma z.p) | \ell \cdot \ell''(\gamma l,q)
\]

where we assume that \( \ell' \) and \( \ell'' \) have no common prefix. But

\[
\ell(\ell'(\gamma z.p) | \ell''(\gamma l.q)) \Rightarrow \ell(\ell'(\ell(p)[\ell''(l)/z]) | \ell''(q))
\]

UBLCS-96-1
and, in order to be consistent with the axiomatization, we must also guarantee that

\[
\ell \cdot \ell' (\gamma z. p) | \ell \cdot \ell' (\overline{\gamma} t . q) \rightarrow (\ell | \ell' (p | \ell' (t) | x) | (\ell' | \ell') (q)).
\]

This is the reason for keeping a suffix of the labels of the sender and of the receiver as label of interaction of a par-composition. Remark that this operation must be safe w.r.t. the problem of optimal reductions, because labeling is used in order to formalize the notion of optimal sharing (see Definition 2.6 below). Actually this is the case, since there is never sharing on redexes composed with the par operator (see Proposition 2.8).

Clearly there is a strong relationship between a labeled process and the one obtained by erasing labels. In particular, they perform the same reductions. We will not formalize this correspondence, since it is quite obvious, and in the following we will always go from labeled processes to the (corresponding) unlabeled ones (and vice versa).

**Definition 2.5** Let \( e \) be a labeled process where every label is atomic and different from any other (this property is described by the proposition \( \text{init} \)). Let \( e \rightarrow^* e' \). Two redexes in \( e' \) are in the same family if and only if they have the same label of interaction.

Redexes in the same family must be considered as copies, that is they are obtained by duplications performed into the past. Remark that families in the initial term have always cardinality 1 (because there is no past).

**Definition 2.6** An implementation is optimal if the representation of redexes in the same family is kept unique.

Said with other words, an optimal implementation performs no useless duplication.

**Remark 2.7** The syntactical counterpart of a reduction of an optimal implementation is the contraction of a complete family of redexes, namely all the redexes that have the same label of interaction. These contractions may reduce the possible nondeterminism because the choices concerning the redexes in a family must be done once, for every redex (see also the Introduction). For instance, take the process

\[
\tau = \gamma x. \gamma y. p \mid \overline{\gamma} t . q \mid \gamma x. \gamma y. p \mid \overline{\gamma} t . q
\]

where processes with the same underlining must be considered as copies. Then we may reduce \( \tau \) as follows (assuming \( x \neq y \) and \( y \notin \text{fv}(t) \)):

\[
\tau \rightarrow \gamma y. p[t/x] \mid \gamma x. \gamma y. p \mid \overline{\gamma} t . q \mid q = \tau'
\]

There are two redexes in \( \tau' \): one between the process \( \gamma y. p[t/x] \) and \( \overline{\gamma} t . q \) and the other, which is in the same family of the first reduction, between \( \gamma x. \gamma y. p \) and \( \overline{\gamma} t . q \). If we choose the complete family of the first redex to be fired as first reduction then we loose at the second step the redex between \( \gamma y. p[t/x] \) and \( \overline{\gamma} t . q \).

**Proposition 2.8** Let \( p \rightarrow^* q \) and \( \gamma x. q' \mid \overline{\gamma} r . \tau' \) be a redex occurring in \( q \). The cardinality of the family of this redex is always 1.

That is, redexes built with the parallel composition have never copies. This has a strong consequence on the implementation, since only synchronous processes may appear in the scope of a duplicator.

3 The implementation of processes

Let us come to the optimal implementation of our processes. As stated in Definition 2.6, the aim is to avoid duplications of redexes which are in the same family. This is performed by means of
a graphical representation of expressions, where sharing is performed by ad-hoc nodes (for this reason the graphs will be called sharing graphs). That is, besides the nodes needed for describing the operators of our calculus, called proper nodes, we need a set of nodes for implementing sharing, (better, optimal sharing, in the sense of Definition 2.6) as for usual \(\lambda\)-calculus (see [5]). These latter nodes are called control nodes. The set of nodes of the evaluator is illustrated in Figure 1. Control

![Figure 1. The set of nodes of the evaluator](image)

nodes (erasing, croissant, bracket and fan) interact with other nodes through the port in the bottom, which is called principal port. The proper nodes are channel nodes, nil, par and sync-par. Channel nodes and nil interact with the other nodes through the port in the top. Par and sync-par may interact with every port. The port in the top of the proper nodes is called output port. The other ports are called argument ports, except for the port at the left, in the bottom of a channel node \(\gamma\), which is called bound port.

Remark that the synchronous par has exactly two argument ports, whilst the par may have several arguments. This because it represents an unordered list of processes (due to the commutativity and associativity axioms).

There are several ways for explaining the purposes and the meaning of control nodes. For example, in [5] a linear logic metaphor is used. Alternatively, control nodes may be understood from the semantic point of view (i.e. the terms represented by sharing graphs) [8, 5, 2].

3.1 The encoding of processes

For each integer \(n\) and process \(p\), we define a graph \(P_n(p)\) by induction on \(p\) together with its set of free nodes which represent the free variables of \(p\). The translation of a process \(p\) is defined to be \(P(t) = P_0(t)\).

Nil \(0\). The encoding of \(0\) is:

\[ P_n(0) = \]

Variable \(x\).

\[ P_n(x) = \]

The \(n\)-indexed croissant represents the (free) variable \(x\).

Input \(\gamma x. p\). There are two cases, according to the variable \(x\) does or does not appears in \(p\). When \(x\) is in \(p\) we connect the the free node of \(p\) representing \(x\) with the bound port of \(\gamma\) (in this case the free nodes of \(\gamma x. p\) decrease of one). Otherwise the bound port of \(\gamma\) is connected with an erasing node. The two cases are drawn below:

\[ P_n(\gamma x. p) = \]

\(P_n(p)\)
We warn the reader that erasing nodes produce garbage. As far as we know there is no set of rules able to eliminate all the garbage which may appear in sharing graphs. For this reason in our basic system we avoid to deal with garbage collection rules.

**Output** $\overline{t}.p$. The output of a term $t$ over a channel $\gamma$ and continuation $p$ is encoded as follows:

$$P_n(\overline{t}.q) = P_n(t) P_n(q)$$

Remark that the encoding of $t$ is at a deeper level then the other part of the graph (in the linear logic terminology, $t$ is inside a box because it may be duplicated and/or erased [5]). The free nodes corresponding to the same variable in $t$ and in $p$ are linked together in $P_n(\overline{t}.p)$ by means of a fan.

**Parallel composition** $p_1 \mid \cdots \mid p_k$. For simplicity, let us draw the case $p \mid q$. The general case $p_1 \mid \cdots \mid p_k$ is similar, putting a tree of fans in the bottom in order to implement the sharing of common variables.

$$P_n(p \mid q) = P_n(p) P_n(q)$$

As for the output, the free nodes corresponding to the same variable in $p$ and in $q$ are linked together in $P_n(p \mid q)$ by means of a fan.

**Synchronous parallel composition** $s \parallel t$. The encoding of the synchronous parallel composition is quite similar to the one for the parallel composition:

$$P_n(s \parallel t) = P_n(s) P_n(t)$$

The root of the term is represented by the node in the top of $P(p)$. The free variables of $p$ are represented by the free nodes of $P(p)$.

### 3.2 The rewriting rules

Rewriting rules may be classified in four groups:

1. **control rules**, managing the interaction between control nodes;
2. **axiom rules**, implementing the axiomatization $\equiv$ of processes;
3. **communication rules**, performing the communication between two processes;
4. **interfacing rules**, describing the interaction between a control node and a proper node.

The discussion of these rules is in order.

1. **Control Rules.** These are the same 12 rules used for implementing optimal sharing in the $\lambda$-calculus (see [5], for instance). Control rules are illustrated in Figure 2.
2. Axiom Rules. These rules implement the axiomatization of processes. For instance we must take processes composed by the par node up-to commutativity, associativity and consider 0 as the identity. This is performed by the evaluator through a set of par-nodes taking a number \( n \) of arguments, for every \( n > 0 \). The rewriting rules are depicted in Figure 3. In the case of sync-par we have just one rule stating that \( p || 0 = p \).

3. Communication Rules. A communication between two processes, a sender and a receiver on the same channel, is performed when they are in a par context or in a sync-par context. In these cases we have a sort of generalization of \( \beta \)-reduction (where the second argument of the application has a continuation). The rules are illustrated in Figure 4.

It is important to observe that the communication of processes connected by the par node may cause nondeterminism. A typical configuration is depicted in Figure 5, where a sender on the channel \( \gamma \) is in parallel with two receivers on the same channel. Then a choice must be done, favouring exactly one of the receivers.

4. Interfacing Rules. The interaction between control nodes and parallel facilities has been the main difficulty of this paper, since being too liberal with the transmissible processes is conflicting with optimality (see Section 1). For this reason we have constrained messages of our calculus to be synchronous processes. As a consequence, the evaluator holds the property that no control node is in front of one of the port of some par node. However, control nodes may be well in front of other proper nodes, different form the par. An example of process whose evaluation

![Figure 2. The control rules (0 ≤ i < j)](image)

![Figure 3. The axiom rules](image)
Figure 4. The communication rules

Figure 5. A case of nondeterminism

shows up an interaction between a fan and a sync-par is \( \gamma x.((x||N)||x||M)) \mid (\overline{x}y.((x||P)).Q). \)

The interactions between control nodes and proper nodes, called interfacing rules, are illustrated

in Figure 6.

**Remark 3.1** The optimal evaluator implements the computation process in terms of a set of local and microscopic moves. Observe that these elementary moves may run in parallel, which put forward a further issue: the possibility of implementing the parallelism of a process at a granularity which is different, better, finer than the one expressed by the process itself.

Besides the critical pairs which cause nondeterminism (see Figure 5) there are other critical pairs, which concern par nodes, but are harmless.
Proposition 3.2 Let \( \mathcal{P}(p) \rightarrow^* N, N \xrightarrow{u} N' \) and \( N \xrightarrow{v} N'' \), such that \( u \) is an axiom rule and \( v \) is a communication rule. Then there exist \( u', v' \) such that \( N' \xrightarrow{u'} N' + \) and \( N'' \xrightarrow{v'} N'' + \).

4 Correctness

The correctness of the implementation is based on a read-back procedure which gives the process represented by a sharing graph. The implementation is correct if, whenever \( N \) is a sharing graph obtained by evaluating \( \mathcal{P}(p) \) and \( N \rightarrow N' \), then the process read-back from \( N \) eventually reduces to the process read-back from \( N' \).

The definition of the read-back procedure is provided by labeling edges of the sharing graphs through contexts and interpreting the nodes as contexts transformers. Expressions matching the sharing graphs are thus “unfoldings” of the graphs where only particular paths are considered (consistent paths), that is paths that behave well w.r.t. contexts.

4.1 Consistent paths

Definition 4.1 The set of contexts over a set \( X \) of variables is inductively generated by the following rules:

- \( \square \) is a context (the empty context);
- if \( a \) is a context then so are \( \circ \cdot a \) and \( \bullet \cdot a \);
- if \( a \) and \( b \) are contexts then also \( (a, b) \) is a context;
- every variable \( x \in X \) is a context.

Contexts will have the shape \( A = \langle \cdots (a_n, a_{n-1}) \cdots, a_0 \rangle \) and we will say that \( a_n \) is the subcontext of \( A \) at width \( n \) (notation \( A^n[i] \)).

The definition below describes how the nodes of sharing graphs modify the contexts when traversed. Notice that, the meaningful modifications are performed by control nodes. More precisely the traversal of a control node \( n \) can be forbidden if the external context does not allow the transformation performed by \( n \). As a consequence, there are illegal (better, non consistent) paths in the sharing graphs.

Definition 4.2 (consistent path) A consistent path in a sharing graph \( G \) is a directed path starting and ending respectively at a negative and positive port and such that

1. every edge of the path is labeled with a context;
2. consecutive pairs of edges traversing a control node satisfy one of the following constraints:
   
   \[
   \begin{align*}
   A^i[\langle b, \cdot \rangle] & \quad A^i[\langle b, a \rangle] & \quad A^i[\langle b, a \rangle] & \quad A^i[\langle b, a \rangle] \\
   A^i[\langle b, \square \rangle] & \quad A^i[\langle b, a \rangle] & \quad A^i[\langle b, a \rangle] & \quad A^i[\langle b, a \rangle]
   \end{align*}
   
3. consecutive pairs of edges traverse a proper node from the output port to an argument port, without changing the context.

Consistent paths will be taken equivalent up to contexts. That is, two consistent paths having pairwise equal edges are considered equal, even if the contexts differ.

Note that, it is not possible to traverse a channel node \( \gamma \) through its bound port: when a path arrives in front of a bound port, it stops there.

Example 4.3 As an example, consider the sharing graph in Figure 7.a. In Figure 7.b we have labeled the \( \bullet \)-branch of the 1-indexed fan, the edge connecting the two fans and the \( \circ \)-branch of the 0-indexed fan such that the corresponding path is a consistent path. The sharing graph in Figure 7.c has an input channel node \( \gamma \) in front of a 0-indexed fan. By definition, there are two consistent paths, starting respectively at the
Definition 4.4 A loop is a consistent path that starts at the body port of a channel node $\gamma$ and terminates at the bound port of the same node. Let $n$ be the index of the $\gamma$ node. A loop $\varphi$ of $\gamma$ is proper if the initial context of $\varphi$ is $\langle \dots \langle a, a_{n-1} \rangle, \cdots, a_0 \rangle$ and the final context is $\langle \cdots \langle b, a_{n-1} \rangle, \cdots, a_0 \rangle$.

4.2 The read-back

Let $P(p) \rightarrow^* N$. The read-back procedure gives the process associated to $N$. In particular, the read-back procedure outputs a variant of the syntax tree of the term, called syntax graph, where bound variables are represented by a port of the binder $\gamma$ and their occurrence is described by an edge entering into such port.

Remark 4.5 Since syntax graphs are graphical representations of processes, they are assumed to be quotiented by the same axiomatization defined in Section 2. Remark that two $\alpha$-convertible processes have the same syntax graph, because the representation of a bound variables is a port of the binder rather than a name.

The read-back procedure will use the following functions:

- **access-arg**($n$, $C$) gives the set of maximal consistent paths never traversing proper nodes and starting at the argument ports of the node $n$ with (an instantiation of the) context $C$;
- **end-node**($\varphi$), **end-port**($\varphi$), **end-context**($\varphi$) give the final node, port (better, number of the node end-node($\varphi$)) and final context of the consistent path $p$, respectively;
- **bound-endport**($\varphi$) is true if the consistent path $p$ ends at a bound port, false otherwise;
- **input-channel**($n$) is true if $n$ is a channel node $\gamma$, false otherwise;
- **level**($n$) (only defined when input-channel($n$) = true) gives the index of the node $n$;
- **connect**($n$, $i$, $m$, $j$) connects with an edge the $i$-th argument port of the node $n$ with the $j$-th port of $m$;
- **clean-up**($\langle \cdots \langle c_k, c_{k-1} \rangle, \cdots, c_0 \rangle$, $k$) = $\langle \cdots \langle z, c_{k-1} \rangle, \cdots, c_0 \rangle$, where $z$ is a fresh variable;
- **new-node**($n$) creates a new node of the same type as $n$.

Remark 4.6 Observe that every path in **access-arg**($n$, $C$) always starts at an argument port of $n$ (which is negative) and terminates at an output port of a proper node or at a bound port. This because these ports are the unique ports of proper nodes having positive polarities and, by definition of consistent path, their ending edge must be plugged into positive ports.

The definition of the read-back is in Figure 8. We assume that the number 0 is the index for the output port of nodes.

The meaning of the arguments of **read-back** is the following:

- **n** is the node in the sharing graph we are reading-back;

---

Figure 7. Context transformations
procedure read-back(n, C, m, S)
let access-arg(n, C) = \{\varphi_1, \ldots, \varphi_k\};
for i = 1 to k do
  if \neg bound-endport(\varphi_i) then
    \{ m′ := new-node(end-node(\varphi_i));
       connect(m, i, m′, 0);
    if input-channel(n)
      then S′ := S ∪ \{ (m, n, clean-up(\text{end-context}(\varphi_i), level(n)))\};
      else S′ := S;
    read-back(end-node(\varphi_i), end-context(\varphi_i), m′, S′);
  } else { let \langle m′, end-node(\varphi_i) \rangle, clean-up(\text{end-context}(\varphi_i), level(end-node(\varphi_i))) \in S;
              connect(m, i, m′, end-port(\varphi_i));
  }
}

Figure 8. The read-back procedure

\(C\) is the context of the consistent path accessing the output port of the node \(n\);
\(m\) is the node in the syntax graph where \(n\) is read-back into;
\(S\) is a set of triples whose first component is an input channel in the syntax graph, the second component is an input channel in the sharing graph and the third one is a context. The set of first projections of \(S\) determines exactly the input channels which may occur in the arguments of the node \(m\). The node binding a variable occurring in the arguments of \(m\) will be determined by means of the contexts.

Informally, \(\text{read-back}(n, C, m, S)\) behaves as follows. Initially, it is called with \(n\) as the root node of the sharing graph (the outermost proper node), \(C = \langle x, y \rangle\), \(m\) as the root node of the syntax graph (of the same type as \(n\)) and \(S = \emptyset\). The first call to \(\text{access-arg}\) gives the set of consistent paths accessing to the arguments of \(n\). Let us read-back the \(i\)-th argument of \(n\). To this aim, let \(p_i\) be the consistent path starting at the \(i\)-th argument port of \(n\). By definition of \(\text{access-path}\), \(p_i\) never traverses proper nodes and ends in front of a proper node \(n′ = \text{end-node}(p_i)\).

There are two cases:

1. The proper node \(n′\), possibly indexed by \(k\), is accessed from the output port. This case is easy: we create a new node \(m′\) corresponding to \(n′\), connect the \(i\)-th argument port of \(m\) and the output port of \(m′\). Then we reiterate the read-back with \(n′, C′, m′, S′\), where \(C′\) is the “cleaning-up” of the ending-context of \(p_i\) and \(S′\) is the updating of \(S\) with the triple \(\langle m′, n′, C′ \rangle\), provided that \(m\) is an input channel node (the reason for this operation is explained below).

2. The node \(n′\) is accessed from a bound port. Then the binder is one of the nodes got by reading-back \(n′\). The problem is that there could be a lot of them! By the last operation performed in the previous case, this node will eventually appear in the first component of some pair in \(S\). The right binder is determined by the associated context \(C\). Namely, let \(k\) be the index of \(n′\). There exists exactly one \(m′\) such that \(\langle m′, n′, \text{clean-up}(C′, k) \rangle\) belongs to \(S\). This fact is far from evident, and it is based on an essential invariant that we shall prove in the following section. This invariant states that, if \(k\) is the index of \(n′\), the context at level \(j < k\) at the output port of any binder is the same as the context at the same level at the bound port. Moreover if two binders, obtained by reading back \(n′\) are one inside the other (the outer binder performs bindings in its body, where the inner binder appears), they will be accessed with contexts which differ for at least a level \(h < k\).

4.3 The correctness of the read-back
The first property we state guarantees that the read-back “behaves well” w.r.t. the translation \(P\).
Proposition 4.7 Let \( n \) be the proper node at the root of \( \mathcal{P}(p) \), where \( p \) is a generic process. Then \( \text{read-back}(n, \{x, y\}, m, \emptyset) = p \), where \( m \) is a new node of the same type as \( n \).

Proposition 4.7 gives the static correctness of the read-back. The first dynamic result ensures the invariance of consistent paths (and, therefore, of the read-back) w.r.t. control rules, axiom rules and interfacing rules.

Proposition 4.8 The control rules, the axiom rules and the interfacing rules preserve the consistency of paths. Therefore they do not change the read-back.

In particular, axiom rules do not change the read-back because they implements the axiomatization \( \equiv \) of Section 2, and syntax graphs are taken up-to such axiomatization. Remark also that Proposition 4.8 is false when a communication rule is fired. For instance, in the process \( \gamma z. 0 \parallel \tau t. 0 \) there is a consistent path from the root to the process \( t \). However, after the communication, there is no consistent path between the root and \( t \), since \( t \) is disconnected.

The correctness of the read-back procedure relies on the following four properties.

Proposition 4.9 Let \( \mathcal{P}(p) \rightarrow^* N \), for some \( p \).

1. The independence property: The consistency of a path \( \varphi \) starting at the argument port of a channel node at level \( n \) is independent from the initial context at level \( n \). Namely, let \( \langle \cdots, a_n, \cdots \rangle \) be the the initial context of \( \varphi \). Then \( \varphi \) may be consistently labeled by starting with a context \( A' = \langle \cdots, a'_n, \cdots, a_0 \rangle \), where \( a'_n = \langle \cdots, (x, b_{k_2}), \cdots, b_0 \rangle \), for every \( k_2 \) large enough and every \( b_i \). Moreover, if \( \varphi \) terminates at a port of a \( m \)-indexed channel node, for every initial context like \( A' \), the final contexts of \( \varphi \) have always the same contexts at the first \( m \) levels.

2. The transparency property: Let \( \psi \cdot \varphi \) be a consistent path starting at the root node such that \( \varphi \) is a loop of a node \( \gamma \) at level \( n \). Then \( \psi \cdot \varphi \) can be decomposed into \( \psi' \cdot \varphi' \) such that \( \varphi \) is a proper loop.

3. The separation property: There exists no consistent path in \( N \) starting at the argument port of a \( k \)-indexed input channel \( n \) and terminating at the output port of \( n \) such that, for every \( j < k \), the initial and final contexts at level \( j \) are equal.

4. The termination property: Every consistent path in \( N \) is finite and can be lengthened consistently in order to start at the root node or at an erasing node and terminate at a bound port or at a free variable. Hence the procedure read-back always terminates.

PROOF: The independence, transparency, separation and termination properties will be proved all at the same time. Let \( \mathcal{P}(p) \rightarrow^* N \). We reason by induction on the length of this derivation.

(basic case) The independence, transparency and termination property may be easily proved by induction on the structure of \( p \) and by definition of \( \mathcal{P} \). In particular, as far as the transparency property is concerned, it is enough to observe that every loop in \( \mathcal{P}(p) \) is proper because there is no consistent path traversing a node twice. Due to the lack of such paths, the separation property holds, too.

(inductive case) Let \( \mathcal{P}(p) \rightarrow^* N' \simto N \). Assume that the four properties holds in \( N' \). By cases on the reduction \( u \). If \( u \) is a control rule or an axiom rule or an interfacing rule then, by Proposition 4.8, the consistency of paths does not change. Therefore the four properties follow easily by inductive hypothesis.

Let us consider the case when \( u \) is a communication rule. Assume that \( u \) traverses a par node: the case of sync-par node may be proved in a similar way and thus omitted. Foremost remark that consistent paths of \( N' \) which do not traverse the redex \( u \) do not change along \( u \). Hence the four properties hold for their residuals in \( N \) by inductive hypothesis. Therefore, consider paths which traverse \( u \). Take the following consistent paths in \( N' \):

\[ 3. \text{We have omitted the definition of residual of a path, since it is almost standard. Indeed it may be defined edgewise, by cases on the rewriting rule.} \]
namely a consistent path $\varphi$ accessing the par node from the root, a consistent path $\varphi \cdot e \cdot \psi$ accessing the body of $\gamma$ from the root, a path $\varphi \cdot e \cdot \phi$, such that $\phi$ is a proper loop of $\gamma$, two paths $\varphi \cdot e' \cdot \xi$ and $\varphi \cdot e' \cdot \zeta$ accessing the first and second argument of $\overline{\gamma}$, respectively. The four properties are proved in order:

**independence property** Consider the paths $\varphi \cdot e \cdot \psi$ and $\varphi \cdot e' \cdot \zeta$. Their residuals after $u$ are almost the same, except for edges $e$ and $e'$ which disappear. Remark that such edges do not modify the context. Hence, as far as these paths are concerned, the independence property holds by inductive hypothesis. Now take the path $\varphi \cdot e \cdot \phi \cdot e' \cdot e' \cdot \xi$, which is not consistent since it should terminate at the bound port of $\gamma$. However its residual after $u$ is consistent because $\gamma, \overline{\gamma}, e$ and $e'$ disappear in $N$. The consistency of the residual of $\varphi \cdot e \cdot \phi \cdot e' \cdot e' \cdot \xi$ follows by inductive hypothesis on the transparency property for the path $\xi$ and the transparency property for $\varphi \cdot e \cdot \phi$. The independence property holds for the residual of $\varphi \cdot e \cdot \phi \cdot e' \cdot e' \cdot \xi$ because the edges $e$ and $e'$ which disappear do not modify the context.

**transparency property** As for the previous item, the transparency property is an immediate consequence of the inductive hypothesis for the residuals of $\varphi \cdot e \cdot \psi$ and $\varphi \cdot e' \cdot \zeta$. Now assume that $\xi$ terminates at the bound port of an input channel node $\lambda$. Then, since $\varphi \cdot e' \cdot \xi$ holds the transparency property, $\varphi \cdot e' \cdot \xi$ may be decomposed into $\varphi' \cdot \xi'$ such that $\xi'$ is a proper loop. Due to the communication rule $u$, the residual of the path $\phi$ is “grafted” somewhere into $\varphi' \cdot \xi'$. This is not problematic if the grafting happens inside $\varphi'$. If the grafting happens inside $\xi'$ then use the inductive hypothesis of the independence property for $\xi'$.

**separation property** There is just one case which does not follow easily by inductive hypothesis: the residual of the path $\varphi \cdot e \cdot \phi \cdot e' \cdot e' \cdot \xi$. Let this residual be $\varphi' \cdot \phi' \cdot \xi'$ and assume that a $k$-indexed channel node $\lambda$ is traversed by $\varphi, \phi$ and $\xi$. The inductive hypothesis ensures that, if we take two occurrences of $\lambda$ such that (a) they are both in $\varphi$ (resp. $\phi, \xi$) or (b) one of them is in $\varphi$ and the other is in $\phi$ (resp. $\xi$) then, also in $\varphi', \phi' \cdot \xi'$, they are accessed with two contexts differing for some level $k'$, $k' < k$. But what about an occurrence of $\lambda$ in $\phi$ and another in $\xi$? Let $\phi = \phi^1 \cdot \phi^2$ and $\xi = \xi^1 \cdot \xi^2$ such that $\phi^1$ and $\xi^1$ both terminate at the output port of $\lambda$. By contradiction, assume that they have the same context at every level $k'$ with $k' < k$. Then, by the independence property, $\phi^1$ and $\xi^1$ must be the same path. Since $\xi$ terminates at the bound port of $\gamma$, the path $\varphi \cdot e' \cdot \xi$ must access $\gamma$ somewhere and must verify the transparency property. There are two possibilities:

1. $\varphi \cdot e' \cdot \xi$ may be decomposed into $\varphi^+ \cdot \phi^+ \cdot e'' \cdot \xi$ such that $\phi^+ \cdot e' \cdot \xi$ is a proper loop. In this case the consistent path $\varphi^+ \cdot \phi^+ \cdot e \cdot \phi$ has two occurrences of $\gamma$ which are accessed with contexts whose levels are the same till the level of $\gamma$. This invalidates the inductive hypothesis on $\varphi \cdot e \cdot \phi$.

2. $\varphi \cdot e' \cdot \xi$ may be decomposed into $\varphi \cdot e' \cdot \xi^+ \cdot \xi^1$ such that $\xi^+ \cdot \xi^1$ terminates in front of $\lambda$ and $\xi^1$ accesses the node $\gamma$ and terminates at its bound port. In this case the consistent path $\varphi \cdot e \cdot \phi^+ \cdot \xi^1$ accesses twice the node $\gamma$ with contexts whose levels are the same till the level of $\gamma$. Again the inductive hypothesis on the transparency property is invalidated.

**termination property** This property holds in $N$ because the consistent paths created by $u$ are finite compositions of residuals of consistent paths in $N'$.

The next step is the proof that, when we are reading back a sharing graph, we are always able to find consistent paths connecting the ports of two proper nodes, i.e. the function access-arg
never fails. A sufficient condition for this lengthening being always possible is provided by the absence of deadlock configurations.

**Definition 4.10** A deadlock is one of the following configurations:
1. two different control nodes with the same index are connected through their principal ports;
2. a control node with index \( n \) and a channel node with index \( m \) are connected through their principal ports and \( n \geq m \);
3. two channel nodes at different level are connected at the same par node or sync-par node;
4. a control node connected through its principal port to a par node.

**Remark 4.11** It is not possible that two channel nodes are connected through their principal ports for polarity reasons, since such ports are positive.

**Lemma 4.12** There is no deadlock in sharing graphs yielded by contracting \( P(p) \), for every \( p \).

**Proof:** There are four deadlock configurations whose absence must be proved. The configurations of type 1 of Definition 4.10 are excluded by the termination property. Indeed the edge connecting the two control nodes is consistent but cannot be consistently lengthened (the reader is invited to check this sentence).

In order to prove the absence of the remaining configurations, we need few properties, which are defined in terms of the following functions:
- \( \text{index} \) gives the index of a control node or channel node.
- let \( \mathcal{S} \) be the function defined on paths traversing exactly one control node or a par node or a sync-par node and such that \( \mathcal{S}(\text{ene}^c) = 1 \), if \( n \) is a croissant traversed from the auxiliary port to the principal port or \( n \) is a bracket traversed from the principal port to the auxiliary port; \( \mathcal{S}(\text{ene}^c) = -1 \), if \( n \) is a bracket traversed from the auxiliary port to the principal port or \( n \) is a croissant traversed from the principal port to the auxiliary port; \( \mathcal{S}(\text{ene}^c) = 0 \) otherwise.
- let \( \mathcal{R} \) be the function defined on consistent paths \( e_k m_k e_{k-1} \cdots e_1 m_1 e_0 \) never traversing channel nodes, ending at the principal port of a \( h \)-indexed channel node \( n \) or at a bound port of a \((h-1)\)-indexed input channel node and defined as follows:
  \[
  \mathcal{R}(e_0) = h \\
  \mathcal{R}(e_k m_k e_{k-1} \cdots e_1 m_1 e_0) = \mathcal{R}(e_{k-1} \cdots e_1 m_1 e_0) + \mathcal{S}(e_k m_k e_{k-1})
  \]
- let \( \mathcal{U} \) be the function defined on consistent paths \( e_0 m_0 e_1 \cdots e_k m_k e_{k+1} \) never traversing channel nodes and starting at the second argument port (or at the body port) of a \( h \)-indexed channel node or at the first argument port of a \((h-1)\)-indexed channel node or at the argument port of a par node (in this case \( h = 0 \)) and defined as follows:
  \[
  \mathcal{U}(e_0) = h \\
  \mathcal{U}(e_0 m_0 e_1 \cdots e_k m_k e_{k+1}) = \mathcal{U}(e_0 \cdots e_i) + \mathcal{S}(e_i m_i + e_{i+1})
  \]

The properties we must prove are in order:

**a. 1.** For every consistent path \( \varphi = e_k m_k e_{k-1} \cdots e_1 m_0 e_0 \) never traversing channel nodes and ending at a port of a channel node we have
- \( \text{index}(m_i) < \mathcal{R}(e_{i-1} \cdots e_0) \), if \( m_i \) is traversed from the auxiliary port to the principal port;
- \( \text{index}(m_i) \leq \mathcal{R}(e_{i-1} \cdots e_0) \), otherwise.

Moreover, if \( \varphi \) also starts at a channel node:
- \( \mathcal{R}(\varphi) = j + 1 \), if \( \varphi \) starts at the first argument of a \( j \)-indexed output channel;
- \( \mathcal{R}(\varphi) = j \), otherwise.
If \( \varphi \) starts at a par node:

- \( \mathcal{R}(\varphi) = 0 \).

2. For every consistent path \( \varphi = e_0m_1e_1 \cdots e_km_{k+1} \) never traversing channel nodes and starting at an argument port of a channel node or at a par node:
   - \( \text{index}(m_i) > U(e_0 \cdots e_{i-1}) \), if \( m_i \) is traversed from the principal port to the auxiliary port;
   - \( \text{index}(m_i) \geq U(e_0 \cdots e_{i-1}) \), otherwise.

b. Let \( \varphi \) and \( \psi \) be two consistent paths starting at the same sync-par node, terminating at the principal port of a channel node or at a bound port and never traversing channel nodes. Then

\[
\mathcal{R}(\varphi) = \mathcal{R}(\psi)
\]  

Let us show that \( a \) and \( b \) imply the absence of configurations of type 2, 3 and 4 of Definition 4.10. The property \( a \) forbids deadlocks of type 2 and configurations where an argument port of a par node is connected with the output port of a \( k \)-indexed channel node and \( h \neq 0 \) (deadlock of type 3 concerning par nodes). The property \( b \) forbids deadlocks of type 3 concerning sync-par nodes.

As far as deadlocks of type 4, there are two cases: a control node is (1) in front of the output port of a par node \( n \) or (2) in front of an argument port. Concerning (1), take a consistent path \( \psi_e \mathcal{N} \varphi \) traversing in order the control node \( n \), the par node \( n \) and terminating at a channel node, such that no channel node is traversed. Then, by \( a \), \( \mathcal{R}(\varphi) = 0 = \mathcal{R}(\mathcal{N} \varphi) \). Therefore \( \text{index}(m_i) > \mathcal{R}(\mathcal{N} \varphi) \), thus invalidating \( a \). Concerning (2), let \( \mathcal{N} \varphi \) be a consistent path starting at an argument port of \( n \), traversing a control node from the principal port to the auxiliary port and terminating at a channel node, such that no channel node is traversed. Then, by \( b \), \( \text{index}(m_i) \leq \mathcal{R}(\mathcal{N} \varphi) = \mathcal{R}(\mathcal{N} \varphi) = 0 \). This means that \( \text{index}(m_i) = 0 \). But this contradicts the fact that, by \( a \), \( \text{index}(m_i) > U(e) = 0 \).

Therefore, we may reduce to prove \( a \) and \( b \).

Let \( P(p) \rightarrow^{*} N \). We reason by induction on the length of this derivation. The basic case follows by definition of \( P \). Let \( P(p) \rightarrow^{*} N' \) and assume that \( a \) and \( b \) hold for \( N' \). By cases on the last reduction \( u \).

- \( u \) is an axiom rule or an interfacing rule: The reader is invited to check that axiom rules or interfacing rules do not modify the definitions of \( \mathcal{R} \) and \( U \). Hence \( a \) and \( b \) hold for \( N \), too.

- \( u \) is a control rule: The properties \( a \) and \( b \) almost follow by inductive hypothesis, since \( \mathcal{R} \) and \( U \) are also invariant w.r.t. control rules. There is only one problematic case. Take a par node \( n \) and a consistent path \( \varphi = e_0m_1e_1 \cdots e_{k+1} \) in \( N' \) starting at an argument port of \( n \) and terminating at a channel node, without traversing channel nodes, and assume that \( u \) involves the nodes \( m_k \) and \( m_{k-1} \). Since \( \mathcal{R}(\varphi) = 0 \) and \( U(e_k) = 0 \) it must be that \( \text{index}(m_k) = 0 \). Moreover, \( m_k \) cannot be a bracket, since in this case \( \mathcal{R}(\varphi) > 0 \). Let us discuss the case when \( m_k \) is a croissant (the other is even simpler). There are two possibilities for a control node with the principal port towards \( n \) to appear in \( N \):
   1. \( m_k \) and \( m_{k-1} \) are 0-indexed croissants, one in front of the other;
   2. \( m_k \) is a 0-indexed croissant and \( m_{k-1} \) is a \( i \)-indexed control node and \( i > 0 \).

In case 1, \( 0 = U(e_0e_1m_{k-1}e_{k-2} \cdots e_0) > \text{index}(m_{k-2}) \), which is impossible. In case 2, \( 1 = U(e_0e_1m_{k-1}) > \text{index}(m_{k-1}) \), which is again impossible since \( \text{index}(m_{k-1}) \geq 1 \).

- \( u \) is a communication rule: There are two subcases: par composition and the sync-par composition. The discussion of the latter one is almost the same as the first one and therefore we omit it. So, let \( \gamma \) and \( \overline{\gamma} \) be the two channel nodes connected to a par node in the redex \( u \). Remark that in this case \( \gamma \) and \( \overline{\gamma} \) have index 0, by inductive hypothesis on \( a \). According to the rule, the body port of \( \gamma \) and the second argument of \( \overline{\gamma} \) are connected directly to the par node and the bound port of \( \gamma \) is connected to the first argument of \( \overline{\gamma} \). Let us check \( a \) and \( b \).

Take, for instance, paths connecting the bound port of \( \gamma \) and the first argument of \( \overline{\gamma} \) (the checks for the other paths are easy). Let \( \varphi = \varphi' \cdot e \) be a consistent path which never traverses channel nodes and terminating at the bound port of \( \gamma \) and let \( \psi \) be a consistent path which never traverses channel nodes, starting at the first argument of \( \overline{\gamma} \) and terminating at a proper node \( n \). Remark that \( \mathcal{R}(e) = \mathcal{R}(\psi) = 1 \). Therefore the properties about \( \mathcal{R} \) on the residual
of \( \varphi \cdot u \cdot \psi \) follow by inductive hypothesis. As far as \( U \) is concerned, let \( \psi = e' \cdot \psi' \). Then
\[ U(\varphi) = 1 = U(e) \]
and, again, the properties about \( U \) on the residual of \( \varphi \cdot u \cdot \psi \) follow by inductive hypothesis. 

\[ \square \]

\textbf{Theorem 4.13} Let \( \mathcal{P}(\varphi) \rightarrow^* N \) and let \( \varphi' = \text{read-back}(n, (x, y), m, \emptyset) \), where \( n \) is the root node of \( N \) and \( m \) is a new root node. Then

1. \( p \rightarrow^* \varphi' \);
2. if \( N \) is in canonical form then also \( \varphi' \) is in canonical form.

\textbf{Proof (sketch):} 1. By induction on the length of the derivation yielding \( N \). The basic case is Proposition 4.7. Let \( \mathcal{P}(p) \rightarrow^* N' \overset{u}{\rightarrow} N \) and assume by induction that \( p \rightarrow^* \text{read-back}(n', (x, y), m', \emptyset) = q \), where \( n' \) is the root node of \( N' \) and \( m' \) is a new root node. We reason by cases on the sharing graph reduction \( u \). If \( u \) is due to a control rule or to an axiom rule or to an interfacing rule then \( q = \varphi' \) by Proposition 4.8. Therefore \( p \rightarrow^* \varphi' \) follows by inductive hypothesis.

The remaining case is when \( u \) is due to a communication rule. This case is depicted below, where a consistent path \( \varphi \) (the dashed arrow) starting at the root and accessing the par-node \( n \) is illustrated. We discuss only the case of par-composition, since the sync-par is similar.

Let \( e \) and \( e' \) be the edges connecting the nodes \( \gamma \) and \( n \) to \( n \). It is easy to prove (by induction on the length of \( \varphi \)) that the \( \text{read-back} \) associates to \( \varphi \) a unique node \( n' \) in the syntax graph \( q \). Similarly for \( \varphi \cdot e \) and \( \varphi \cdot e' \). Now, let \( \psi \) be a consistent path of \( N' \) starting at the body port of \( \gamma \) or at the second argument of \( \overline{n} \). Then the residual of \( \varphi \cdot e \cdot \psi \) is a consistent path of \( N \) by definition. Therefore the correctness of the body of \( \gamma \) and the second argument of \( \overline{n} \) follow by induction.

Moreover, by definition of \( \text{read-back} \), a bound variable of \( \gamma \) will be connected in \( N \) with that instance of the body of \( \overline{n} \) accessed from the root with \( \varphi \cdot e' \). This holds provided there exists a proper loop \( \phi \) such that \( \varphi \cdot e \cdot \phi \) is consistent. The existence of \( \phi \) is guaranteed by the transparency property. Furthermore \( \phi \) is unique by the separation property, since every instance of \( \gamma \) in \( \varphi \cdot e \cdot \phi \) is accessed by a context which differs from that of any other instance for some level less than the index of \( \gamma \). Now, let \( \xi = \varphi \cdot e' \cdot \xi' \) be a consistent path traversing the first argument of \( \overline{n} \). Then the residual of \( \varphi \cdot e \cdot \phi \cdot e' \cdot e \cdot \xi \) is a consistent path in \( N \) by the independence property. Consequently, the correctness of the read-back, as far as the access path \( \varphi \) is concerned, follows by inductive hypothesis.

Finally, remark that the above reasoning is parametric w.r.t. \( \varphi \). This means that, if \( U \) is the set of the consistent paths accessing \( n \) and \( W \) is the set of the corresponding nodes in the syntax graph \( q \), we obtain \( q \rightarrow^* \varphi' \) by firing all the copies of the redex \( \gamma \cdot \overline{n} \) corresponding to the par nodes in \( W \).

2. This property is easy, by 1 and Lemma 4.12. Indeed, by 1 and the definition of \( \text{read-back} \), every redex in \( \varphi' \) have a counterimage in \( N \) which consists of two opposite channel nodes connected to the same par (or sync-par) node through two consistent path traversing control nodes only. This configuration cannot be a deadlock, by Lemma 4.12. Therefore no control node may appear along the two consistent paths, otherwise some interaction could be possible, invalidating the hypothesis that \( N \) is in normal form. On the other hand, the foregoing configuration should be a communication redex, thus contradicting again the hypothesis that \( N \) is in canonical form. \( \square \)
5 Optimality

In this section we prove that our implementation of the sharing is optimal, in the sense of Lévy and of Definition 2.6. That is, if two redexes yielded by a labeled derivation have the same label, then they have the same representation in the sharing graph. As in $\lambda$-calculus (see [7]), it is possible that duplication of labels goes ahead w.r.t. the reduction of proper redexes. The problem is due to the fact that fan-nodes may duplicate labeled edges (e.g. when a fan is along an edge connecting a sync-par with a channel node). In order to cope with such situations, we must determine, for every redex, a part of it that is never duplicated by propagation of fans. To this aim, Lamping [8] introduces the notion of prerequisite chain of a node in the syntax graph (not in the sharing graph!). This notion must be adapted to our case.

Remark 5.1 By property c of Lemma 4.12, a fan node never appears in front of a port of a par node. Hence redexes due to par composition can never be duplicated (or shared) by the evaluator. This is the sharing graph counterpart of Proposition 2.8.

Definition 5.2 A syntax graph is in canonical form if it has no axiom redex.

It is possible to prove that, given a syntax graph $G$ of a process, there exists the syntax graph $G'$ in canonical form such that $G$ and $G'$ are equivalent w.r.t. $\equiv$ of Section 2 (in particular $G'$ may be obtained by a derivation where only axiom rules are fired. This derivation is finite because, at each step, the number of par/sync-par/nil nodes strictly decreases). The notion of normal graph allows to provide a simpler definition of prerequisite chain.

Definition 5.3 A prerequisite chain is a path
1. starting at the output port of a proper node (including the root) and terminating at the output port of channel nodes or at the root;
2. traversing channel nodes from the auxiliary ports to the principal ports and par/sync-par nodes from one argument port to another argument port.
A prerequisite chain is proper if it starts at the output port of a channel node.

The following property is an immediate consequence of the definition.

Property 5.4 For every syntax graph $E$ in canonical form and every node $n$ in $E$, there exists at least a prerequisite chain starting at $n$.

The representation of a proper prerequisite chain in a sharing graph can never be totally duplicated by propagation of fans because fans (the control nodes performing duplication) can
- neither enter the chain from the output port of the par/sync-par node (there is no rule for this)
- nor enter the chain from the ending channel nodes since the output edges of the ending nodes are links of the prerequisite chain;
- nor enter the par nodes from arguments ports different from the two ports traversed by the prerequisite chain because, by property c of Lemma 4.12, never a fan may appear in front of a port of a par node.

Notice that, when a fan is in between a (communication) redex (for instance, in front of an argument port of a sync-par node), we have two different (proper) prerequisite chains in the read-back subgraph, each corresponding to the two branches of the fan.

For technical reasons in the proof of Lemma 5.7 below, we need a smooth generalization of the notion of prerequisite chain: the so-called prerequisite graph.

Definition 5.5 A prerequisite graph in the syntax graph in canonical form of a process is inductively defined by:
1. every proper prerequisite chain is a prerequisite graph;

2. let $G$ be a prerequisite graph, let $n$ be a par/sync-par node in $G$ and let $\varphi$ be a prerequisite chain starting at the output port of $n$. Let $G'$ be the graph consisting of $G$ and of the edges and nodes along $\varphi$. Assume that $G'$ has no sub-graph depicted in Figure 9, where $r$ is a channel node, $n$ and

Figure 9. Constraints over prerequisite graphs

are proper nodes and $e$ is the edge outgoing the port in the bottom, at the left of $r$. Then $G'$ is a prerequisite graph.

Observe that prerequisite graphs may be composed only with “closed” paths, which are prerequisite chains. In this way no fan is allowed to enter into a prerequisite graph from the outside (this fact could duplicate the representation of a prerequisite graph).

As we have already remarked, the representation of prerequisite graphs cannot be totally duplicated by propagation of fans. Below we prove the existence of a strong relation between prerequisite graphs which are copies and their representations. To this aim we need labels in order to formalize the notion of copy. It is folklore that the labeled calculus may be represented by the usual syntax graph where edges are marked by labels. Hence, we will not be fussy on this point (see, for instance, [9, 2]).

**Definition 5.6** Two graphs are in the same family if they are isomorphic as graphs and their corresponding edges have the same label. Let $\varphi \cdot e$ be a prerequisite chain and $\ell$ be the concatenations of labels along $\varphi$ and $e$ is the edge outgoing the port in the bottom, at the left of $r$. Then $G'$ is a prerequisite graph.

The reader has surely noticed several analogies between the above definition of label of a prerequisite chain and the notion of label of interaction introduced in Section 2.1. Indeed redexes $\gamma x. p \| \gamma k. q$ are particular prerequisite chains and in this case the two notions match.

**Lemma 5.7** Let $P \rightarrow N$ and $q = \text{read-back}(n, (x, y), m, \emptyset)$, where $n$ is the root of $N$ and $m$ is the root of $q$. If two prerequisite graphs of $q$, the canonical form of $q$, are in the same family then they have the same representation in $N$. 4

In the sketched proof of this lemma we use the notions of ancestor and residual of a graph without providing a formal definition. We prefer to rely on reader’s intuition, in order to simplify the presentation. The formalization of these notions may be provided edgewise, by cases on the rewriting rule. This is not particularly difficult. The pedantic reader may try to define ancestor and residual by himself. Remark that the residual of a prerequisite graph w.r.t. a communication rule is always connected. This follows by the definition of communication rule and the constraints in Figure 9.

**Proof (sketch):** The lemma is proved together with the property:

---

4. The representation of a prerequisite graph $G$ in a sharing graph may traverse same par/sync-par nodes which miss in $G$. This because prerequisite chains are defined on graphs in canonical form. Remark that such par/sync-par nodes may be erased by firing axiom/control/interfacing rules only.
(\*) Let \{\ell, \ell'\} be the label of a prerequisite chain and \xi be \ell \cdot \ell' or \ell' \cdot \ell. Then neither \(\xi\) nor \(\ell\) occur as sub-labels of prerequisite chains in different families or as sub-labels of edges.

The proof is by induction on the length of \(\mathcal{P}(p) \rightarrow^* \mathcal{N}\). The basic case is obvious since the property \(\text{init}\) is assumed. Let \(\mathcal{P}(p) \rightarrow^* \mathcal{N'} \rightarrow^* \mathcal{N}\) and let \(q' = \text{read-back}(\mathcal{N}', \langle x, y \rangle, \mathcal{N'}, \emptyset)\), where \(\mathcal{N'}\) is the root node of \(\mathcal{N'}\). Let \(q^+\) be the canonical form of \(q\). By cases on the last reduction \(u\).

Let \(u\) be a control rule or an axiom rule or an interfacing rule not involving a fan and a channel node. By Proposition 4.8, the read-back is invariant w.r.t. such rules. Therefore also prerequisite graphs do not change, since they are defined on (the canonical form of) the syntax graph of the read-back term. In particular, observe that if the interaction is due to a fan and a sync-par or a channel node, there is no duplication of prerequisite chains/graphs in the read-back term.

Let \(u\) be an interfacing rule between a fan and a channel node. We reason on proper prerequisite chains; the general reasoning being similar. The duplication of the channel node, let us say \(\tau\), also duplicates the edges \(e, e'\) of the auxiliary ports. This could duplicate representations of prerequisite chains created by firing the nodes connected to \(e\) or \(e'\). This case is illustrated below:

```
\begin{center}
\begin{tikzpicture}[scale=0.7]
  \node (f) at (0,0) [circle,draw] \{f\};
  \node (c) at (2,0) [circle,draw] \{c\};
  \node (l) at (4,0) [circle,draw] \{l\};
  \node (r) at (0,-2) [circle,draw] \{\tau\};

  \draw (f) -- (r);
  \draw (r) -- (c);
  \draw (r) -- (l);

\end{tikzpicture}
\end{center}
```

In the above picture, the firing of the redex between \(f\) and \(l\) may create a prerequisite chain passing through \(\tau\). And the interfacing rule could duplicate the representation of this prerequisite chain. However this should invalidate the inductive hypothesis for the prerequisite graph \(H\) consisting of the redex \(\tau\)-\(l\) and the prerequisite chain of the sync-par node (the existence of \(H\) is guaranteed by Property 5.4 applied to the sync-par node).

Let \(u\) be a communication rule. The property (\*) holds in \(\mathcal{N}\) by definition of labeled rewriting rule and the inductive hypothesis. So we care only of the property stated in the lemma.

By Theorem 4.13, \(q' \rightarrow^* q\) and, by Proposition 3.2, \(q'^+ \rightarrow^* q^+\). We must prove the existence of a bijection between families of prerequisite graphs in \(q^+\) and their representation in \(\mathcal{N}\). Let \(\{\ell, \ell'\}\) be the label of the set \(U\) of prerequisite chains represented by \(u\) and let \(\xi = \ell \cdot \ell'\) or \(\ell' \cdot \ell\).

It is not difficult to prove that the above bijection, which holds for \(\mathcal{N}'\), by inductive hypothesis, is preserved in \(\mathcal{N}\) as far as prerequisite graphs not containing a redex in \(U\) are concerned. Moreover these prerequisite graphs cannot have as sublabel \(\xi\) or \(\ell\) or \(\ell'\) which must occur in the residuals of graphs containing \(u\). Therefore it is enough to prove the existence of a bijection between the set of families created and/or modified by \(U\) and their representation in \(\mathcal{N}\).

Let \(F\) be a graph in \(\mathcal{N}'\), traversing \(u\) and representing a family of prerequisite graphs \(W\). The analysis of some basic cases of prerequisite graphs created by \(U\) is in order. The missing cases follow with reasonings which may be reduced to one of the following situations. Let \(q\) be the par/sync-par node involved in the reduction \(u\).

1. the par/sync-par node \(q\) cannot be erased in \(\mathcal{N}\) by axiom/control/interfacing rules. Assume that \(w\) is a prerequisite chain of \(F\) and \(w'\) be its residual after \(u\). Assume also that \(w\) traverses the body port of the \(\gamma\) node connected to \(q\) (the case when \(w\) traverses the second argument port of \(\tau\) is similar). There are two subcases: (a) when the second argument port of \(\tau\) is connected to a channel node \(m\) or (b) when such port is connected to a par/sync-par node \(p\). Let us discuss the two cases separately.
   (a) Let \(F'\) be the residual of \(F\). Let also \(w''\) be the prerequisite chain starting at \(m\) (see figure below). Let \(w'\) be the residual of \(w''\) after \(u\). The case is shown below.
The graph composed by \( F' \) and \( w' \) is a prerequisite graph, let it be \( F'' \). Let \( \{ \ell, \ell_1, \ell_2 \} \) be the label of \( w \) and \( \{ \ell', \ell' \cdot \ell_2 \} \) be the label of \( w'' \). The set of prerequisite graphs corresponding to \( F'' \) have some labels where at least one between \( \ell, \ell_1, \ell_2 \) occur as sub-labels. Due to these occurrences the prerequisite graphs of \( F'' \) differ from other families of prerequisite graphs. The unicity of the representations of two prerequisite graphs follows by inductive hypothesis.

(b) Let \( F' \) be the residual of \( F \) (also in this case \( F' \) is a connected graph) and let \( G \) be a prerequisite graph connected at \( p \) through an edge \( e \). The case is depicted below.

Remark that the graph \( G' \) consisting of \( G, e \) and \( u \) is a prerequisite graph in \( N' \). Notice also that, in \( N \), the graph \( F'' \) consisting of \( F' \), the residual of \( e \) and (the residual of) \( G \) is a prerequisite graph. The lemma holds for \( F'' \) because, as in the previous case, it is a composition of prerequisite graphs in \( N' \).

2. A prerequisite chain \( v \) outgoing the bound port of \( \gamma \) belongs to \( F \) and the par/sync-par node traversed by it – and different from \( q \) – cannot be erased in \( N \) by axiom/control/interfacing rules. This case is depicted below, where a prerequisite chain \( w \), not belonging to \( F \) is also illustrated.

Let \( p \) be the par node in the bottom, \( e \) be the edge outgoing the output port of \( p \) and \( n \) be the node at the other end of \( e \). Let \( G \) be the part of \( F \) connected to \( n \) and not traversing \( e \). Then the graph of \( N \) consisting of the residuals of \( G, e, w, p \) and \( v \) is a representation of prerequisite graphs and it verifies the lemma because composition of representations of prerequisite graphs of \( N \).

3. The node \( q \) may be erased in \( N \) by axiom/control/interfacing rules. Actually, the notion of prerequisite graph has been provided for dealing with this case. Consider the following configuration:
The dotted arrow $v$ is not a proper prerequisite chain, but it is part of a prerequisite graph (for instance take the graphs $u$ and $v$). As in the previous cases, the residual of $F$ after $u$ plus the residual of $w$ (if it does not belong to $F$) is a representation of a prerequisite graph which verifies the lemma because it is the composition of representations of prerequisite graphs in $N'$.

The theorem about optimality becomes an obvious consequence of the above lemma, as soon as one remarks that sync-par redexes of our calculus are particular prerequisite chains.

**Theorem 5.8** The sharing graph implementation is optimal. Namely families have always a unique representation in the sharing graphs.

An immediate consequence of the above theorem follows by a close inspection of the encoding in Example 2.2:

**Corollary 5.9** The encoding of $\lambda$-calculus in Example 2.2 is correct and optimal.

6 Conclusion and future work

In this paper we have provided an optimal implementation of an higher-order concurrent language. This language allows parallelism, nondeterminism and synchronous communication. The unique constraint is that transmissible processes must be deterministic. Optimality. Our programme of research aims at

1. developing an optimal implementation for calculi admitting every process to be transmitted.

   Besides the problems we discussed in the Introduction, we must also solve the problem of keeping shared copies of redexes when a request for breaking one of them appears. A possible way-out is to use a general form of fan, where sequences of control operators are absorbed into one. Such a technique has been already proposed by Guerrini [6] and proved to be advantageous in terms of efficiency and of the read-back procedure.

2. studying architectures for concurrent/distributed implementations of our evaluators. As we have remarked (see Remark 3.1), optimal algorithms describe processes at a finer level of granularity, where concurrent redexes have not to synchronize and, therefore, have a propensity towards parallelism.

Besides the above two items, there is another issue which intrigues us. Milner has recently introduced graphical representations of his action calculi (pi-nets in [11], action graphs in [12]). There are a lot of analogies between our sharing graphs and Milner’s graphs. One for all the notion of interaction, which underlies the two systems. And surely we will not be surprised if someone will present sharing graphs in terms of Milner’s action structures. What we consider interesting is the understanding of the deep relationships, if any, between the two systems. Such as, whether the mechanism of interaction is more “primitive” in one system or in the other. Or the existence of a “minimal set” of nodes, possibly got from the two systems, through which express concurrency.

**Acknowledgements** Many thanks to Andrea Asperti for the discussions we had together.
References


