Self-applicable partial evaluation for the π-calculus

Marc Gengler   Matthieu Martel

Ecole Normale Supérieure de Lyon
Laboratoire de l'Informatique du Parallélisme (LIP)
46, Allée d'Italie
69364 Lyon Cedex 07, France
E-mail: [Marc.Gengler, Matthieu.Martel]@lip.ens-lyon.fr

Abstract

In this paper, we are interested in self-applicable partial evaluation for the \(\pi\)-calculus, a language which models the concurrent behavior of communicating processes. We use the classic three-steps methodology. First, we write a meta-\(\bar{\text{I}}\)-interpreter for the language. Second, we introduce an abstract analysis that determines which operations (communications) can be executed at compile-time. The notion of well-annotatedness of terms is defined. Finally, we exhibit the self-applicable partial evaluator which is applied to well-annotated terms, and we prove its correctness with respect to the \(\bar{\text{I}}\)-interpreter. This approach is compatible with Futamura's projections. Proofs of correctness are based on the notion of weak reduction equivalence.

Keywords: Partial Evaluation, Parallelism, Pi-Calculus, Meta-\(\bar{\text{I}}\)-Interpretation, Binding-Time Analysis.

1 Introduction

Partial evaluation is an optimization technique which consists of specializing programs with respect to the known part of their inputs. Considerable work has been done in this area [1], however, most of the research has focused on sequential functional or imperative languages. In this paper, we are interested in partial evaluation for the \(\pi\)-calculus [3]. This language, à la CCS [8], describes the behavior of concurrent processes which interact using explicit communications over channels. \(\pi\)-calculus is well-suited to model the principles of concurrent as well as object-oriented languages [16].

Even though one can expect great improvements from partial evaluation techniques and related abstract interpretation methods in those areas, only few attempts have been made. First, concerning parallelism, applications could include load balancing, communication optimization [4], and run-time parallelization using techniques similar to those used in [5] for deferred compilation. Nevertheless, only some attempts have been done. For instance, [3] presents an online partial evaluator for a CCS-like language. Second, concerning object-oriented languages, reflection is an important topic and partial evaluation might address the lack of efficiency due to high-level abstractions [7].

Our goals are first to show that partial evaluation for concurrent languages is possible, and then to formalize it. Our approach uses the well established three-steps methodology developed in [2, 10, 11, 12, 17] for partial evaluation of the \(\lambda\)-calculus, in the context of concurrent languages.

(i) We define a meta-\(\bar{\text{I}}\)-interpreter for the \(\pi\)-calculus. We prove its correctness w.r.t. to the notion of weak reduction equivalence [9], denoted \(\approx\). \(\approx\) is defined in Section 2. A function \(f\) describes how programs can be encoded in the \(\pi\)-calculus. We also define the \(\bar{\text{I}}\)-interpreter, denoted \(\bar{E}\). The correctness criterion states that applying \(\bar{E}\) to the encoding \(\bar{f}\) of a \(\pi\)-term \(P\) yields to a program \(\approx\)-equivalent to \(P\). Reflection of \(\bar{E}\) is crucial to preserve properties of partial evaluation such as Futamura's projections [17]. However, as discussed in [1-4], it interests for its own, since reflection is a property one usually tries to establish when studying a new formal language such as \(\pi\)-calculus.

(ii) We propose a binding-time analysis (BTA for short) for the language, that determines which communications can be executed at compile-time. This BTA annotates terms, and outputs two-level terms [2] in which dynamic operations are underlined. As far as we are aware, this is the first binding-time analysis proposed for the \(\pi\)-calculus. We also introduce the notion of well-annotatedness, needed to define the correctness of partial evaluation.

(iii) Finally, we achieve our primary objective by introducing a self-applicable partial evaluator \(\text{Pev}\). \(\text{Pev}\) is applied to two-level terms, encoded using a function \(\text{Eval}\). Again, the proofs are based on the notion of weak reduction equivalence. We state that applying \(\text{Pev}\) to well-annotated terms preserves the \(\approx\)-equivalences.

In Section 2, we formalize the concepts of partial evaluation and introduce notations and definitions concerning the \(\pi\)-calculus. Section 3 presents the mechanisms used in the remainder of the paper, using a more familiar language.
the λ-calculus. Sections 4, 5 and 6 respectively deal with the interpreter, the binding-time analysis, and the partial evaluator. Results are discussed in Section 7.

2 Preliminary Definitions

Following the presentation of [17], a programming language $\mathcal{L}$ is specified by a set $\mathcal{P}$ of $\mathcal{L}$-programs, a set $\Delta$ of $\mathcal{L}$-data, and a family $\mathcal{S} = (S_n)_{n \in \mathbb{N}}$ of $(n+2)$-ary relations ($S$ is the semantics of $\mathcal{L}$). Let $\rho \in \mathcal{P}$, and $\delta_1, \ldots, \delta_n, \delta \in \Delta$, $(\rho, \delta_1, \ldots, \delta_n, \delta) \in S_n$ if and only if $\delta$ is the result of the application of the program $\rho$ to the data $\delta_1 \ldots \delta_n$.

An interpreter is a program which uses two kinds of data: the program $\rho$ to be executed, and the data $\delta_1 \ldots \delta_n$ to be applied to $\rho$. Since a self-interpreter for the language $\mathcal{L}$ is a program $\text{Eval} \in \mathcal{P}$, $\rho$ must be encoded under the form of $\mathcal{L}$-data, in order to be understood by $\text{Eval}$.

Definitions 1 (Encoding and meta-interpretation) A self-interpreter related to an encoding function $[\cdot] : \mathcal{P} \rightarrow \Delta$ is a program $\text{Eval} \in \mathcal{P}$ such that

$$ ([\text{Eval}, \rho], \delta_1, \ldots, \delta_n, \delta) \in S_{n+1} \iff (\rho, \delta_1, \ldots, \delta_n, \delta) \in S_n $$

(1)

Equation (1) states that the interpretation of the encoding of a program $\rho$ produces the result $\delta$ specified by the semantics of $\rho$.

A partial evaluator is a program $\text{Pev}$ which specializes a program $\rho$ with respect to the known part (the static part) of its data. Note that it is always possible to concatenate all the static (resp. dynamic) data used by $\rho$. Thus, we can assume that any program $\rho$ only uses data $\delta_1$ and $\delta_2$, which are respectively the concatenations of all the static and dynamic ones. In order to determine the static parts of a program $\rho$, a binding-time analysis $\Phi : \mathcal{P} \rightarrow \mathcal{P}_o$ is used. It yields an annotated version $\Phi(\rho) \in \mathcal{P}_o$ of $\rho$. These annotated terms must again be encoded as $\mathcal{L}$-data, in order to be understood by the partial evaluator.

Definitions 2 (Analysis and Partial evaluation) A self-applicable partial evaluator related to an encoding function $\Phi$ and to an encoding function $[\cdot] : \mathcal{P}_o \rightarrow \Delta$ is a program $\text{Pev} \in \mathcal{P}$, $\text{Pev} : \Delta \rightarrow \Delta$ such that

$$ ([\text{Pev}, \Phi(\rho)], \delta_1, \rho') \in S_2 \Rightarrow ([\text{Eval}, \rho', \delta_2, \delta) \in S_2 $$

(2)

Equation (2) states that partially evaluating a program $\rho$ w.r.t. the part $\delta_1$ of its data leads to a program $\rho'$ such that $\rho'(\delta_2) = \rho(\delta_1, \delta_2)$. Note that using (1), Equation (2) is equivalent to

$$ ([\text{Pev}, \Phi(\rho)], \delta_1, \rho') \in S_2 \Rightarrow (\rho, \delta_1, \delta_2, \delta) \in S_n $$

In this paper, we are interested in the case $\mathcal{L} = \pi$, namely, in self-interpretation and self-applicable partial evaluation for (asynchronous) $\pi$-calculus[9]. Programs, denoted by $\pi$-terms, are generated by the grammar:

$$
\begin{align*}
  P &::= \sum_{i=1}^n \pi_i. P_i \mid P \parallel Q \parallel P \parallel (\nu \alpha) P \\
  \pi &::= \nu \alpha | \alpha(x)
\end{align*}
$$

Channels (also called names) are the only values in the language. They are used to indicate the names of communication channels as well as transmitted values. Sending (resp. receiving) a message $\nu x$ over a channel $\alpha$ is done by the use of the instruction $\nu x$ (resp. $\alpha(x)$). The name $\alpha$ used in such a communication $\pi$ is called the subject of $\pi$. $P || Q$ represents the concurrent execution of $P$ and $Q$. $\pi P$ represents the replication of $P$, i.e. $\pi P = P || P || \ldots$ as many times as needed, $\nu x$ creates a new channel $x$ for communications, and $\sum_{i=1}^n \pi_i. P_i$ represents the non-deterministic choice among processes. Let $\rightarrow$ be the reduction relation over terms. The semantics of the sum operator is:

$$
\begin{align*}
  \text{eval}(\nu x.P) &\parallel \text{eval}(\alpha(x).Q) \\
  \rightarrow P \parallel Q | \nu x \leftarrow x
\end{align*}
$$

(3)

Remark that each term in a non-deterministic sum must start with a communication. The other reduction rules are:

$$
\begin{align*}
P &\rightarrow P' & Q &\rightarrow Q' & P &\rightarrow P' \\
\parallel Q &\rightarrow P || Q & \parallel Q &\rightarrow P || Q' & (\nu x) P &\rightarrow (\nu x) P'
\end{align*}
$$

The reflective transitive closure of $\rightarrow$ is denoted $\rightarrow^*$. The following abbreviations are also used. $\alpha(x_1 \ldots x_n)$ denotes $\alpha(x_1) \ldots \alpha(x_n)$, $\nu x_1 \ldots x_n$ denotes $\nu x_1 \ldots \nu x_n$ and $(\nu x_1 \ldots x_n)$ denotes $(\nu x_1) \ldots (\nu x_n)$. $\emptyset$ denotes the null process, i.e. the empty sum of non-deterministic processes. $\nu$ and $\alpha$ abbreviate $\nu()$ and $\alpha()$, i.e. the communications of empty messages, also called signals.

We introduce now two standard equivalence relations between processes. First, the structural congruence relation, denoted $\equiv$, identifies the terms whose syntax differs, but that express the same meaning. Structurally equivalent terms act identically in any context, and substituting one for the other does not modify the behavior of the system.

Definition 3 (Structural congruence) The structural congruence $\equiv$ is the smallest congruence relation over terms such that

$$
\begin{align*}
  (i) & \equiv \text{contains the alpha-equivalence relation} \\
  (ii) & | \text{and} \parallel \text{are associative, symmetric and} \emptyset \text{is the neutral element for these operators} \\
  (iii) & P \equiv Q \land (\nu x)[\eta y]. P \equiv (\nu y)[\eta x]. P \\
  (iv) & \text{If} P \text{ does not contain any free occurrence of} x, \text{then} \equiv (\nu x)[P || Q] \equiv (\nu x) Q
\end{align*}
$$

The reduction relation $\rightarrow$ is extended by the rule:

$$
\begin{align*}
P \equiv Q &\Rightarrow Q \rightarrow Q' & Q' \equiv P'
\end{align*}
$$

Also note that the following property used in the proofs in Sections 4 and 6 is easily verified:

$$
\begin{align*}
x \text{ not free in } P \Rightarrow (\nu x) P \equiv P
\end{align*}
$$

(4)

The second relation, denoted $\approx$, is the weak reduction equivalence relation. It identifies terms which look equal from an external point of view. If $P \approx Q$, then any process $R$ cannot distinguish between $P$ and $Q$ by observing its interactions. The notions of unsoundness and observability are fundamental to formalize $\approx$. 

Definitions 4 (Unguardedness and observability) Let \( P \) and \( Q \) be two processes. \( Q \) occurs unguarded in \( P \) if \( Q \) has some occurrence in \( P \) which is not under a prefix \( \pi \). \( P \) is observable at \( \alpha \), written \( P \downarrow_\alpha \), if \( \pi Q \) occurs unguarded in \( P \) for some \( Q \) and \( \pi \) such that \( \alpha \) is the subject of \( \pi \).

For instance, \( P \equiv (\alpha(x)Q_1 \parallel Q_2) \downarrow_\alpha \), since \( P \) can immediately communicate on \( \alpha \). Similarly, \((\alpha x)\alpha(y)P \downarrow_\alpha \) or \((\alpha x)\alpha(x)P \downarrow_\alpha \). However, \((\alpha x)\beta(y)P \downarrow_\alpha \) because no communication on \( \beta \) is possible before \( \alpha(x) \) is reduced (the communication on \( \beta \) is guarded by the one on \( \alpha \)). Note also that \( P \equiv ((\alpha x)\beta(y)P) \downarrow_\alpha \), even if \( x(y) \) is unguarded, because \( x \) is a new channel unknown outside of \( \beta \).

Using these notions, \( \downarrow_\alpha \) is the relation identifying terms observable on the same channels. This property must be preserved under reduction.

Definition 5 (Weak reduction equivalence) The weak reduction equivalence \( \downarrow_\alpha \) is the largest equivalence relation \( \equiv \) over processes such that \( R(P, Q) \) implies:

\[
\begin{align*}
(i) & \quad (P \to P') \Rightarrow (\exists Q' : (Q \to * Q' and R(P', Q'))) \\
(ii) & \quad (\forall \alpha, (P \downarrow_\alpha) \Rightarrow \exists Q' : (Q \to * Q' and Q' \downarrow_\alpha))
\end{align*}
\]

Let us remark that, since \( \downarrow_\alpha \) is symmetric, \( P \downarrow_\alpha Q \) also implies

\[
(Q \to Q') \Rightarrow (\exists P' : (P \to * P' and P' \equiv Q'))
\]

and

\[
(\forall \alpha, (Q \downarrow_\alpha) \Rightarrow \exists P' : (P \to * P' and P' \downarrow_\alpha))
\]

3 Related work

This section briefly describes partial evaluation for pure \( \lambda \)-calculus [10, 11, 12, 17]. It introduces the methodology of self-applicable partial evaluation of a formal language. This scheme, considered here for the well-known \( \lambda \)-calculus, will be used in Sections 4, 5 and 6 for self-applicable partial evaluation of the \( \pi \)-calculus.

As indicated in Section 2, three steps have to be considered. The first one concerns the meta-interpretator. An encoding of \( \lambda \)-terms is given, and the meta-interpretator which executes the encoded terms is introduced. The second step concerns the analysis of programs. It yields annotated terms which also have to be encoded in the \( \lambda \)-calculus. The last step concerns the partial evaluator which treats annotated terms.

Programs in the \( \lambda \)-calculus have to be encoded as data. The representation scheme \([.]_\Lambda \) is given in Figure 1. It combines the notions of signature representation, and of higher order abstract syntax [13]. Variables \( a, b \) and \( c \) act as switch operators. They indicate the syntactic category encoded by \([.]_\Lambda \). \( a \) indicates that \( t \) is a variable, \( b \) and \( c \) respectively indicate application and abstraction.

The evaluator \( \text{Eval}_\Lambda \), also given in Figure 1, executes \( \lambda \)-terms encoded with the function \([.]_\Lambda \). The operation corresponding to one of the indicators \( a, b \) or \( c \) is applied to the variable \( x \), the terms \([e_1]_\Lambda \) and \([e_2]_\Lambda \) or the term \((\lambda x[e]_\Lambda) \).

\[
\text{Eval}_\Lambda \equiv Y (\lambda x.\lambda m.m (\lambda x.x)) \equiv Y (\lambda m.\lambda p.p (m n)) \equiv Y (\lambda x.\lambda c.c (\lambda p.p (m n)))
\]

Figure 1: Encoding and evaluation of \( \lambda \)-terms.

The correctness property consists of proving that \( \text{Eval}_\Lambda \) applied to a program \([t]_\Lambda \) yields a term semantically equivalent to \( t \). In \( \lambda \)-calculus, this is done using the \( \beta \)-equivalence relation \( \equiv_\beta \).

\[
\text{Eval}_\Lambda \ [t]_\Lambda \equiv_\beta \ t \quad \text{(5)}
\]

The next step is to annotate programs. Annotations consist of underlying the dynamic operations. This is formalized by the notion of two-level \( \lambda \)-terms.

Definition 6 (Two-level \( \lambda \)-terms) A two-level \( \lambda \)-term is a term generated by the grammar

\[
t ::= x | (t_1 \cdot t_2) | \lambda x.t | (t_1, t_2) | \Delta x.t
\]

where \( x \), \( (t_1, t_2) \), and \( \Delta x.t \) represent static terms which can be reduced by the partial evaluator \( \text{PEv}_\Lambda \). \( (t_1, t_2) \) and \( \Delta x.t \) represent dynamic terms. In this case, \( \text{PEv}_\Lambda \) partially evaluates the sub-expressions and outputs the encoding \([t]_\Lambda \) of the residual program \( t \). Binding-time analyses producing two-level \( \lambda \)-terms can be found in [11, 12, 17].

\[
[x]_\Lambda \equiv \lambda x.\lambda c.c.x \\
[e_1 e_2]_\Lambda \equiv \lambda x.\lambda c.c.(e_1)\Lambda (e_2)\Lambda \\
[\lambda x.e]_\Lambda \equiv \lambda x.\lambda c.c.(\lambda x.\lambda c.c.e)\Lambda \\
[e_1 e_2]_\Lambda \equiv \lambda x.\lambda c.c.(e_1)\Lambda (e_2)\Lambda \\
[\Delta x.e]_\Lambda \equiv \lambda x.\lambda c.c.((\lambda p.p (m n)))
\]

Figure 2: Encoding and partial evaluation of two-level \( \lambda \)-terms.

Finally, \( \text{PEv}_\Lambda \) is introduced. Annotated terms are encoded using the function \([.]_\Lambda \), given in Figure 2. \([.]_\Lambda \) extends \([.]_\Lambda \) in order to encode the dynamic operations. Here,
five variables are used. d and e respectively represent dynamic application and abstraction. Figure 2 also describes $\text{Peval}_\lambda$, the partial evaluator for the $\lambda$-calculus which interprets two-level $\lambda$-terms. Static terms are reduced, and when a dynamic operation is encountered, its sub-expressions are partially evaluated and the resulting program is encoded as in $\text{Peval}_\lambda$.

**Proposition 7 (Correctness of Peval) Let $t$ be a $\lambda$-term with arity $n$ and $t_\alpha$ an annotated version of $t$. A partial evaluator $\text{Peval}_\lambda$ is correct w.r.t. $t_\alpha$'s annotation system if

$$\forall k \leq n, \forall s_1 \ldots s_k, \text{Peval}_\lambda \ [t_\alpha]_\lambda \ s_1 \ldots s_k = \beta [t \ s_1 \ldots s_k]_\lambda$$

This means that, $(\text{Peval}_\lambda \ [t_\alpha]_\lambda)$ applied to an acceptable freely chosen number of arguments will behave like the program which encodes the application of $t$ to the same arguments. More details about the correctness of the partial evaluator described in Figure 2 can be found in [12] and [17].

### 4 The interpreter

In this section, we introduce a self-interpreter, denoted $\text{Eval}$, for $\pi$-calculus. First, we describe the encoding function $[\cdot]$ of the terms. Next, we introduce $\text{Eval}$ and prove its correctness. Since the $\pi$-calculus is a communication based language, program sources are processes which send messages to the interpreter. $\text{Eval}$ receives and executes the encoded operations.

To keep the evaluator readable we assume without loss of generality the following fact. Every non-deterministic sum of processes $\sum_{i=1}^n \pi_i \cdot P_i$ has a constant size $n$ and can be rewritten

$$\sum_{i=1}^n \alpha_i(x_i) \cdot P_i + \sum_{i=n+1}^m \alpha_i[x_i] \cdot P_i$$

This hypothesis only permits to obtain a more compact representation of $\text{Eval}$. It can be removed either by using reserved channels indicating the length of the sums or by encoding integers, which can easily be achieved in $\pi$-calculus.

As indicated in Section 2, programs in the $\pi$-calculus have to be encoded as data. Since the only entity in the language is the process, all those data are represented under that form. Our encoding is close to the one described in Section 3 for pure $\lambda$-calculus. Letters from $a$ to $e$ denote reserved channels. A process $P$ is encoded onto a channel $\gamma_0$ by the function $[\cdot]_{\gamma_0}$ described in Figure 3.

Names $\pi_l$ to $\pi_r$ indicate which kind of process is encoded. A $\pi_l$ encodes the null process $\emptyset$. A $\pi_r$ is used to indicate a parallel composition. In this case, two new channels $\gamma_1$ and $\gamma_2$ are created and sent onto $\gamma_0$. Next, the processes $P_1$ and $P_2$ are encoded recursively, respectively onto $\gamma_1$ and $\gamma_2$. Name $\pi_i$ is sent to indicate the occurrence of the replication operator. In this case, the following action is done repeatedly. A new channel $\gamma$ is created, and sent onto $\gamma_0$. It is followed by the encoding of the $\gamma$ of the process to replicate. $([\alpha] \cdot P_\gamma)_{\gamma_0}$ produces $\gamma(x) \cdot [P_\gamma]_\lambda$. The message 2 indicates the beginning of such an encoding. Channels $\gamma_1$ and $\gamma_2$ are created and sent onto $\gamma_0$. The interpreter will send the new name on $\gamma_1$ and $[P_\gamma]_\lambda$ is recursively encoded. Finally, in order to encode the sum of $n$ processes, the channels $\gamma_1 \ldots \gamma_n$ and $\delta_1 \ldots \delta_n$ are created and sent onto $\gamma_0$. The prefixes of the $P_i$'s are also sent onto $\gamma_0$. The encoding of a reception reads on a channel $\delta$ the new value, and the process continuation encoding is recursively computed. Concerning the emission, $\delta$ is only used as a signal since no value is needed to encode the continuation.

The interpreter $\text{Eval}$ which evaluates programs encoded on the channel $\gamma_0$ is given Figure 3.

$\text{Eval}$ first reads the names $a \ldots e$ of the case indicators on $\gamma_0$. Next, it receives the description of the operation to execute, under the form of a name from $a$ to $e$. A $a$ indicates the end of the execution. When $b$ is encountered, the interpreter reads on $\gamma_0$ the locations $\gamma_1$ and $\gamma_2$ of the two continuations and executes $(\text{Eval } \gamma_1)$ concurrently with $(\text{Eval } \gamma_2)$. Note that $\text{Eval}$ is duplicated, and $(\text{Eval } \gamma_1) \ [\text{Eval } \gamma_2])$ evaluates $[P_1, \gamma_1]$ [resp. $[P_2, \gamma_2]]$, which indicates that the process has to be replicated. Locations $\gamma_1$ of the continuations are read on $\gamma_0$ and $(\text{Eval } \gamma_1)$ is done as many times as required. Names $\gamma_1$ and $\gamma_2$ are read on $\gamma_0$. The new name $x$ is created and sent to the encoding through $\gamma_1$. Next, $(\text{Eval } \gamma_2)$ is executed. An occurrence of $e$ is followed by reading on $\gamma_0$ the continuation channels $\gamma_1 \ldots \gamma_m$, the names $\delta_1 \ldots \delta_n$, and the prefixes of the $P_i$'s. When a value $v$ is received on $\alpha_i$, $1 \leq i \leq n$, it is sent on $\delta_i$ to the encoding. The substitution is done in $[P_i, \gamma_i]$, and the resulting term is

$$[P_i \ (x_i \leftarrow v), \gamma_i] \ (\text{Eval } \gamma_0)$$

Concerning emissions, no value has to be transmitted. Communications on $\delta_i, \frac{n+1}{2} + 1 \leq i \leq n$, are simple signals.

Finally, note that $\text{Eval}$ uses recursive calls. These calls are not allowed in pure $\pi$-calculus. However recursion may easily be simulated in the language as described in [9]. Let $\psi$ be a reserved channel used for recursion. The principle consists of substituting, for all $\gamma$, the term $\psi(\gamma) \cdot \text{Eval}$ to any occurrence of $\text{Eval } \gamma$ in $\text{Eval}$. Intuitively, communications are done instead of recursive calls. Additionally, $\psi(\gamma_0) \cdot \text{Eval}$ is substituted to $\text{Eval } \gamma_0$. This terms creates a new evaluator for every recursive call. The argument is read on $\psi$.

In order to prove the correctness of the interpreter, we introduce the following notation:

$$E(P) \equiv ([\gamma_0]) ([P_\gamma]_{\gamma_0} \ (\text{Eval } \gamma_0))$$

This means that $E(P)$ denotes the application of $(\text{Eval } \gamma_0)$ to $(P_\gamma)_{\gamma_0}$. The encoded term transmits the source program to the evaluator, via the restricted channel $\gamma_0$. Proofs are based on the notion of weak reduction equivalence. The choice of this relation is motivated by the fact that, as mentioned earlier, terms equivalent by weak reduction are not distinguishable from an external point of view.

**Lemma 8 (Compositionality of $E$) The following properties hold.**

(i) $E(P_1 \ (P_2)_{\gamma_0}) \equiv E(E(P_1)) \equiv E(P_2)$.

(ii) $E(P) \equiv E(P')$.

(iii) $E((\alpha \cdot P)_{\gamma_0}) \equiv (\alpha \cdot E(P))_{\gamma_0}$.

(iv) $E(\sum_{i=1}^n \pi_i P_i) \equiv \sum_{i=1}^n \pi_i E(P_i)_{\gamma_0}$.

**Proof**

For each case, for all $\alpha$, both terms of the equivalence are unobserved at $\alpha$ before reduction, since they communicate
\[
\begin{align*}
[0, \gamma_0] & = (\text{vabode})\overline{\gamma_0}([\text{abo}]) \overline{\alpha} \\
[P_1||P_2, \gamma_0] & = (\text{vabode})\overline{\gamma_0}(\text{abo})[\text{[(v\gamma_1 \gamma_2)/][\text{a}][\gamma_1][\gamma_2]][(P_1, \gamma_1)][(P_2, \gamma_2)]] \\
[P, \gamma_0] & = (\text{vabode})\overline{\gamma_0}(\text{abo})[\text{[(v\gamma_1 \gamma_2)/][\text{a}][\gamma_1][\gamma_2]]}[P, \gamma] \\
[(\nu x)P, \gamma_0] & = (\text{vabode})\overline{\gamma_0}(\text{abo})[\text{[(v\gamma_1 \gamma_2)/][\text{a}][\gamma_1][\gamma_2]]}[x, P, \gamma] \\
\left(\sum_{i=1}^{n} \pi_i P_i, \gamma_0\right) & = (\nu \gamma_1 \ldots \nu \gamma_n \ldots \gamma_1 \ldots \gamma_n \pi_1 \ldots \pi_n \pi_1 \ldots \pi_n [\gamma_1 \ldots \gamma_n] \ldots [\gamma_1 \ldots \gamma_n] \ldots [\gamma_1 \ldots \gamma_n]) \\
& + \left(\sum_{i=1}^{n} \delta_i (x_i)[P_i, \gamma_i] + \sum_{i=n+1}^{\delta} \delta_i [P_i, \gamma_i]\right) \\
\text{Eval}(\gamma_0) & \equiv \gamma_0(\text{abo}), \\
& \text{add} \left(\nu \gamma_0(\text{abo})(\text{Eval} \gamma_1)|| (\text{Eval} \gamma_2)\right) \\
& \text{add}(\nu \gamma_0(\gamma_i)(\text{Eval} \gamma_i)) \\
& \text{add}(\nu \gamma_0(\gamma_i)(\nu x)\overline{\pi}[x])(\text{Eval} \gamma_2) \\
& \text{add}(\nu \gamma_0(\gamma_i)(\nu \gamma_1 \ldots \nu \gamma_n \ldots \gamma_1 \ldots \gamma_n \alpha_0 \ldots \alpha_n \pi_1 \ldots \pi_n || (\text{Eval} \gamma_i) + \sum_{i=n+1}^{\delta} \overline{\pi}[x_i](\text{Eval} \gamma_i)) \\
\end{align*}
\]

Figure 3: Encoding and evaluation of Ψ-terms.

on the reserved channel γ₀. We have to prove that they are still equivalent under reduction. The proof uses the standard properties of ≡ given in Section 2.

Parallel Composition

\[
E(P_1||P_2) \equiv (\nu \gamma_0)((\text{vabode})\overline{\gamma_0}(\text{abo})[\text{[(v\gamma_1 \gamma_2)/][\text{a}][\gamma_1][\gamma_2]][(P_1, \gamma_1)][(P_2, \gamma_2)]])||(\text{Eval} \gamma_0) \\
\approx (\nu \gamma_1 \gamma_2)(([P_1, \gamma_1][[P_2, \gamma_2]])||(\text{Eval} \gamma_1)||(\text{Eval} \gamma_2)) \\
\equiv E(P_1)||E(P_2)
\]

Replication

\[
E(\nu x)P \equiv (\nu \gamma_0)((\text{vabode})\overline{\gamma_0}(\text{abo})[\text{[(v\gamma_1 \gamma_2)/][\text{a}][\gamma_1][\gamma_2]]}[P, \gamma])||(\text{Eval} \gamma_0)) \\
\approx \left(\nu x(P, \gamma)|| \gamma_0(\gamma)(\text{Eval})\right) \\
\approx \left([P, \gamma_1]|([\text{Eval} \gamma_1]||[P, \gamma_2]||([\text{Eval} \gamma_2])|| \ldots \right) \\
\equiv E(P)||E(P)|| \ldots \equiv !E(P)
\]

Restriction

\[
E((\nu x')P) \equiv (\nu \gamma_0)((\text{vabode})\overline{\gamma_0}(\text{abo})[\text{[(v\gamma_1 \gamma_2)/][\text{a}][\gamma_1][\gamma_2]]}[P, \gamma_2]|([\text{Eval} \gamma_0])) \\
\approx (\nu \gamma_1 \gamma_2)((\text{vabode})\overline{\gamma_0}(\text{abo})[\text{[(v\gamma_1 \gamma_2)/][\text{a}][\gamma_1][\gamma_2]]}[x, P, \gamma])||(\text{Eval} \gamma_0) \\
\approx (\nu \gamma_1 \gamma_2)(\text{Eval} \gamma_2)\left([\gamma_1(x),[P, \gamma_2]]||(\nu x')\overline{\gamma_1}[x'](\text{Eval} \gamma_2)\right)
\]

Note that in \( \gamma_1(x) \), x is bound \([9]\). While there is no free occurrence of x in the first term of the parallel composition, \((\nu x)\) may be factorized. It is easy to show that

\[
E((\nu \gamma_1 \gamma_2)(\gamma_1(x), [P, \gamma_2]|(\nu \gamma_1 \gamma_2)[x', \gamma_2]|[\text{Eval} \gamma_2])) \approx (\nu \gamma_1 \gamma_2)(\gamma_1(x), [P, \gamma_2]|(\nu x')\overline{\gamma_1}[x'](\text{Eval} \gamma_2)) \\
\approx (\nu x'((\nu \gamma_1 \gamma_2)(\gamma_1(x), [P, \gamma_2]|(\nu x')\overline{\gamma_1}[x'](\text{Eval} \gamma_2)))) \\
\approx (\nu x')E(P\{x' \leftarrow x\})
\]

Sum

\[
E(\sum_{i=1}^{n} \pi_i P_i) \\
\approx \left(\sum_{i=1}^{n} \delta_i (x_i)[P_i, \gamma_i] + \sum_{i=n+1}^{\delta} \delta_i [P_i, \gamma_i]\right)|| \\
\left(\sum_{i=1}^{n} \alpha_i (x_i)[P_i, \gamma_i](\text{Eval} \gamma_i) \\
+ \sum_{i=n+1}^{\delta} \overline{\pi}[x_i](\text{Eval} \gamma_i)\right) \\
\equiv S
\]

Thus, \( E(\sum_{i=1}^{n} \pi_i P_i) \rightarrow^* S \) such that \( S \downarrow \alpha_i, 1 \leq i \leq n \), and if \( S \) communicates on \( \alpha_i \) then:

\[
S \rightarrow [P_i, \gamma_i] || (\text{Eval} \gamma_i)
\]

Additionally, for all \( \alpha_i \), \( \sum_{i=1}^{n} \pi_i, E(P_i) \subseteq \gamma \), and may reduce to \( E(P_i) \). For all \( \alpha_i \), \( \sum_{i=1}^{n} \pi_i, E(P_i) \) \( \not\subseteq \gamma \) since \( \gamma \) is a local channel. This proves that \( E(\sum_{i=1}^{n} \pi_i, P_i) \subseteq \sum_{i=1}^{n} \pi_i, E(P_i) \).

\[ \square \]

Proposition 9 (Correctness of the interpreter)

\[ \forall P \in \pi, P \approx E(P) \tag{8} \]

**Proof**

Immediate from Lemma 8 and by induction on the structure of \( P \).

- If \( P \equiv \theta \) then for all \( \alpha \), \( P \not\subseteq \gamma \).

\[ E(P) = (\nu \gamma) \left( \text{vehicle}(\text{color}[\alpha | \text{blue}], \overline{\alpha}), \text{Eval} \gamma \right) \]

Clearly, \( E(P) \not\subseteq \gamma \) for all \( \alpha \) since \( \gamma \) is a local channel. Furthermore, \( E(P) \) reduces to 0 without being observable at \( \alpha \).

- If \( P \equiv P_1 || P_2 \) then \( P \not\subseteq \gamma \), and only if \( P_1 \not\subseteq \gamma \) or \( P_2 \not\subseteq \gamma \).

By Lemma 8, \( E(P_1) \not\subseteq E(P_2) \not\subseteq E(P_2) \). So, \( E(P) \not\subseteq \gamma \) iff \( E(P_1) \not\subseteq \gamma \) or \( E(P_2) \not\subseteq \gamma \). Induction completes the proof.

- If \( P \equiv P_1 \) then \( P \not\subseteq \gamma \) if \( P_1 \not\subseteq \gamma \) and by Lemma 8, \( E(P) \not\subseteq E(P_1) \).

- If \( P \equiv (\nu \varphi)P \) then \( P \not\subseteq \gamma \) if \( P \not\subseteq \gamma \) and by Lemma 8, \( E((\nu \varphi)P) \not\subseteq E(P) \).

- If \( P \equiv \sum_{i=1}^{n} \pi_i, P_i \) then \( P \not\subseteq \gamma \) and \( P \not\subseteq \gamma \).

By Lemma 8, \( E(P) \not\subseteq E(P) \). Thus, \( E(P) \not\subseteq \gamma \). Furthermore by induction, if \( E(P) \not\subseteq \gamma \), \( E(P) \not\subseteq \gamma \) if \( E(P_1) \not\subseteq \gamma \).

\[ \square \]

5 Binding-time analysis

In this Section, we introduce a binding-time analysis (BTA) that determines which sub-expressions in a π-term are static, and which are dynamic. Note that in the calculus the only reduction rule concerns the sum operator. Thus, the analysis must consist of annotating which sums can be reduced at compile-time. Following [2], we introduce the notion of two-level π-terms.

**Definition 10 (Two-level π-terms)** A two-level π-term is an expression generated by the following grammar:

\[
P ::= \sum_{i=1}^{n} \pi_i, P_i \mid \sum_{i=1}^{n} \pi_i, P_i \mid P || Q \mid \nu \varphi \mid (\nu \varphi)P
\]

\[ \pi ::= \pi(x) \mid \alpha(x) \]

An underlined sum indicates the dynamic behavior of the term. Such a dynamic term cannot be reduced at compile-time, because none of the prefixes \( \pi_i \) can communicate. Note that a sum \( \sum_{i=1}^{n} \pi_i, P_i \) is static as soon as it reduces to \( P_i \) for some \( i \), i.e., as soon as one of the communications \( \pi_i \) may be achieved statically.

The analysis of a process \( P \) requires some assumptions about which communications may be achieved between \( P \) and the processes \( Q_i \) running concurrently with \( P \). These hypotheses are formalized by the notion of BTA-context \( \Gamma \).

\( \Gamma \) is an abstract description of the communications that are assumed to occur between \( P \) and the \( Q_i \)'s during the analysis of \( P \).

**Definitions 11 (BTA-contexts and BTA-assertions)**

Let \( P \) be a term to analyze. A BTA-context is a π-term \( \Gamma \) which indicates the communications that are assumed to occur between \( P \) and other processes during the analysis of \( P \).

Assertions of the form

\[ \Gamma \vdash P : \omega \]

state that, under the assumption \( \Gamma \) on the processes running concurrently with \( P \), \( P \) may be annotated \( \omega \), where \( \omega \) is a two-level π-term.

For instance, let us consider the term \( P \equiv \alpha(x), 0 \uplus b(y), 0 \) and the BTA-assertion:

\[ 0 \vdash P : \alpha(x), 0 \uplus b(y), 0 \]

(10)

(10) states that the sum operator must be considered as dynamic without assumption. (11) states that assuming that \( \alpha(x) + b(y) \) is reduced by interference with another process, the sum operator in \( P \) may be static.

Intuitively, BTA-contexts are used to recursively analyze the sub-expressions of an initial term. Consider \( P \equiv P_1 || P_2 \) and assume that \( \Gamma_1, \Gamma_2 \vdash P_i : \omega_i, 1 \leq i \leq 2 \). Then \( \Gamma \vdash P : \omega \) represents the hypotheses done when analyzing \( P_1 \) [resp. \( P_2 \)]. Thus, \( \Gamma \vdash P \) in a two-level \( \pi \)-terms, \( \Gamma_1, \Gamma_2 \) are correct if they agrees each other, i.e., since \( \Gamma_1 \) and \( \Gamma_2 \) are \( \pi \)-terms, if \( \Gamma_1 \vdash \Gamma_2 \rightarrow \rightarrow \theta \). If this last property is satisfied, \( \omega \) may be partially evaluated since the static communications are those whose counterpart is present in \( P \). A formal definition of correctness is provided by Definition 12.

Before introducing the inference system for the BTA, we define the following abbreviation, \( \bar{\pi}(\bar{x}) \) represents the communication symmetric to \( \pi \), i.e., \( \bar{\pi}(\bar{x}) \equiv \pi(\bar{x}) \) and \( \bar{\pi}(\bar{x}) \equiv \pi(\bar{x}) \).

Rules for the analysis are described in Figure 4.

The first rule states that one may annotate the π-term \( \theta \) by the two-level π-term \( \theta \), under the empty hypothesis \( \theta \). Rules concerning the parallel, duplicate and new operators just follow the structure of the term. The most important rules concern the sum operator. Consider that \( \Gamma_1 \vdash P_i : \omega_i, 1 \leq i \leq n \). The first rule concerning \( \sum \), states that the sum may be annotated static under the assumption that the term \( \sum_{i=1}^{n} \pi_i, P_i \) can be reduced by interaction with the processes running concurrently with the process we analyze. Secondly, in order to underline (i.e. annotate dynamic) \( \sum_{i=1}^{n} \pi_i, P_i \), we use the context:

\[ \Gamma \equiv \sum_{i=1}^{n} \pi_i, P_i \]
\[
\begin{align*}
\begin{array}{c}
\begin{array}{c}
0 \vdash 0 : 0 \\
\Gamma_1 \vdash P_1 : \omega_1 \quad \Gamma_2 \vdash P_2 : \omega_2 \\
\Gamma_1 \mid \Gamma_2 \vdash P_1 \parallel P_2 : \omega_1 \parallel \omega_2 \\
\Gamma \vdash P : \omega \\
\Pi \vdash \Pi : \omega \\
\Gamma \vdash P : \omega \\
\Gamma \vdash (\text{tx} \, P) : (\text{tx} \, \omega) \\
\Gamma_i \vdash P_i : \omega_i, \ 1 \leq i \leq n \\
\sum_{i=1}^{n} \pi_i \Gamma_i \vdash \sum_{i=1}^{n} \pi_i P_i : \sum_{i=1}^{n} \pi_i \omega_i \\
\Gamma_i \vdash P_i : \omega_i, \ 1 \leq i \leq n \\
\sum_{i=1}^{n} \pi_i \Gamma_i \vdash \sum_{i=1}^{n} \pi_i P_i : \sum_{i=1}^{n} \pi_i \omega_i
\end{array}
\end{array}
\end{align*}
\]

Figure 4: Binding-time analysis of \(\pi\)-terms.

Since each communication \(\pi_i\) may occur in \(\Gamma\), this context states that for all \(i, 1 \leq i \leq n\), \(\Gamma_i\) may be a BTA-context for \(\sum_{i=1}^{n} \pi_i P_i\). Furthermore, \(\Gamma\) satisfies the property: \(\Gamma \rightarrow^{*} 0\) iff \(\forall i, \ 1 \leq i \leq n, \Gamma_i \rightarrow^{*} 0\). Thus, no assumption is done about the \(\pi\)'s and the behavior of \(\Gamma\) only depends on the behavior of the \(\Gamma_i\)'s.

For instance, let us consider the process

\[P \equiv a(x) + b(y) \parallel [u]\]

The analysis of \(P\) may yields the BTA-assertion

\[
\begin{align*}
\left( a(x) + b(y) \parallel [u] \right) \vdash P : a(x) + b(y) \parallel [u] \quad (12)
\end{align*}
\]

The BTA-assertion in equation (12) indicates that under the BTA-context \(a(x) + b(y) \parallel [u]\), both instructions \(a(x) + b(y)\) and \([u]\) are static since a communication on the channel \(a\) may be executed at compile-time reducing \(P\) to \(0\).

Finally, to deal with the correctness of the partial evaluator with respect to annotated terms, we introduce the notion of well-annotatedness, used in the propositions of Section 6.

Definition 12 (Well-annotatedness) A term \(P\) is well-annotated with annotation \(\omega\) if \(\Gamma \vdash P : \omega\), for some \(\Gamma\) such that \(\Gamma \rightarrow^{*} 0\). In this case, we write \(\Gamma \vdash P : \omega\) instead of \(\Gamma \vdash P : \omega\).

6 Partial Evaluation

In this section, we show how to evaluate annotated terms. First, we give an encoding \(\_\_\_)\) for the two-level \(\pi\)-terms. Second, we introduce the natural extension of Eval that yields a partial evaluator and prove its correctness with respect to the binding-time analysis and to the weak reduction equivalence relation. Similarly to Eval, Pev reads the input program on a reserved channel \(\gamma_0\). In addition, Pev has to produce a \(\_\_\_)\)-encoded residual program. This is done using a second reserved channel. So, Pev reads the input program on a first channel \(\mu_0\), and outputs the encoding of the reduced term onto a second channel \(\gamma_0\).

The encoding for annotated terms differs from \([\_\_\_]\) only in the treatment of sum processes. As remarked in \([3]\), the partial evaluator has to execute as many instructions in a given program as possible. Particularly, Pev may resolve non-determinism. Note that a sum process is annotated static as soon as at least one of the \(\pi\)'s is static. From Equation (3), it is obvious that the choice is made between static messages. This leads to the encoding function \([\_\_\_), given in Figure 5. Reserved channels \(r, s, t, u, v, w\) are used instead of \(a, \ldots, c\).

As indicated, two cases have to be distinguished when encoding communications.

Equations (13) and (14) only differ by the message \(\pi\) or \(\overline{\pi}\) they send. \(\pi\) indicates that \(\sum_{i=1}^{n} \pi_i P_i\) is static, since at least one of the \(\pi\)'s can communicate. \(\overline{\pi}\) is sent when the communications cannot be computed at compile-time, and must be encoded as residual code.

The partial evaluator Pev, given in Figure 6, has to deal with two kinds of instructions. When a communication \(\pi\) is annotated static, \(\pi\) is executed. Otherwise, Pev produces the encoding of the residual program \([\_\_\_, \pi]\), and recursively partially evaluates the continuation of the program.

We present now the most natural extension of Eval to a partial evaluator, denoted Pev, and prove its correctness. Let \([\_\_]\) be an annotated input \(\pi\)-term to specialize. Similarly to Section 4, recursion is left implicit. Pev has to execute communications annotated static in \(P\), and to produce the encoding \([R]\) of the residual program \(R\).

First of all, since \([\_\_]\) has to be encoded on a particular channel, Pev takes two arguments. The first one, \(\mu_0\), indicates the location of the encoding of the annotated input program \(P\). The second, \(\gamma_0\), locates the channel used to encode the output residual program \(R\). Equation (15), in Figure 6, describes the structure of Pev.

In the \(\pi\)-term \(A\), the residual code generated on \(\gamma_0\), when the message 0 is encountered, is the encoding of the null process.

In order to partially evaluate the parallel composition of \(P_1\) and \(P_2\), Pev creates two new channels \(\gamma_1\) and \(\gamma_2\) as well as channels \(a\) to \(e\) (\(\pi\)-term 0). Locations \(\mu_1\) and \(\mu_2\) of the continuations are received on \(\mu_0\), and \(P_1\) and \(P_2\) are partially evaluated.

Concerning the replication operator (\(\pi\)-term 0), an occurrence of message \(t\) yields Pev to produce a variant of the encoding function \([\_\_]\), in which continuations are partially evaluated.

When the encoding of \((\text{tx}\, P)\) is encountered on \(\mu_0\), the encoding of the restriction operator is transmitted to the evaluator. Eval sends the new name to Pev which transmits it to the encoding process in order to achieve the specialization.

Concerning the sum, we had two choices (13) and (14) previously. In (13), \(\pi\) indicates the occurrence of a static communication. The action is executed and the continuation is partially evaluated. \(\overline{\pi}\) is similar to the corresponding case of Eval, except that \((P\ v_1\,\gamma_0)\) is substituted to \((Eval\,\gamma_1)\). When, in (14), a message \(\pi\) is encountered, the dynamic communications are encoded on \(\gamma_0\) and Pev is applied to the continuations.

In order to deal with the correctness of Pev, we introduce the following notations where \(\omega\) represents a two-level \(\pi\)-term:

\[
P(\omega, \gamma_0) \equiv (\text{tx} \, P)([\omega, \mu_0] \parallel (P\ \mu_0\,\gamma_0)) \quad (16)
\]
Figure 5: Encoding of two-level π-terms.

\[
\begin{align*}
[0, \mu_0] &= (r \, stuvw \, \overline{[r \, stuvw]}).
\\
[P_1] [P_2, \mu_0] &= (r \, stuvw \, \overline{[r \, stuvw]} \, \overline{[(\nu_1 x_1) [\overline{[\nu_1 x_1]}]]} \, ([P_1, \mu_1] \, || \, [P_2, \mu_2]))
\\
[1] [P_2, \mu_0] &= (r \, stuvw \, \overline{[r \, stuvw]} \, \overline{[(\nu_1 x_1) [\overline{[\mu_1]}]]} \, [P_2, \mu_1])
\\
[1] [P_2, \mu_0] &= (r \, stuvw \, \overline{[r \, stuvw]} \, \overline{[(\nu_1 x_1) [\overline{[\mu_1]}]]} \, [\mu_1(x)] \, [P_2, \mu_2])
\\
\left[\sum_{i=1}^{n} \pi_i, P_2, \mu_0\right] &= (r \, stuvw \, \overline{[r \, stuvw]} \, \overline{\pi_i \, (x_i) \, [P_2, \mu_i]} \, + \, \sum_{r=\frac{n}{2}+1}^{n} \pi_i \, [P_2, \mu_i])
\\
\left[\sum_{i=1}^{n} \pi_i, P_2, \mu_0\right] &= (r \, stuvw \, \overline{[r \, stuvw]} \, \overline{\pi_i \, (x_i) \, [P_2, \mu_i]} \, + \, \sum_{r=\frac{n}{2}+1}^{n} \pi_i \, [P_2, \mu_i])
\end{align*}
\]

\[\text{(13)}\]

\[\text{(14)}\]

Figure 6: Partial evaluation of two-level π-terms.

\[
\text{Pev}(\mu_0, \gamma_0) \equiv \mu_0(r \, \overline{[\text{stuvw}]}) \, (A + B + C + D + E + F)
\]

\[\text{(15)}\]

A \equiv r \, (\nu_1 \, (\text{abcde}) \, [\overline{\gamma_0}] \, \overline{[\text{abcde}]}) \, \overline{\mu_0} \, [0, \gamma_0]

\text{Beginning of the encoding } [P_1 || P_2, \mu_0]

B \equiv s_{\mu_0}((\mu_1 \, \overline{[\text{abcde}]}) \, (\nu_1 \, \gamma_1 \, \gamma_2) \, (\nu_2 \, \gamma_2) \, [\overline{[\nu_1 \, \gamma_1 \, \gamma_2]}] \, \overline{[\nu_2 \, \gamma_2]})

\left(\text{Pev} \, [\mu_1 \, \gamma_1] \, || \, \text{Pev} \, [\mu_2 \, \gamma_2]\right)

\text{(Pev}_{\mu_i \, \gamma_i}) \text{ substituted to } [P_i, \gamma_i] \text{ in } [P_1, \gamma_1] \, || \, [P_2, \gamma_2], \, i = 1, 2

\text{Beginning of } [P_2, \gamma_0]

C \equiv t \, (\overline{[\text{abcde}]}) \, \overline{[\text{abcde}]}) \, \overline{[\overline{[\text{abcde}]})} \, (\overline{[\gamma_1 \, \gamma_2]} \, \overline{[\overline{[\gamma_1 \, \gamma_2]}]} \, \overline{[\gamma_1(x)]} \, \overline{[\gamma_2]} \, \text{Pev}_{\mu_2 \, \gamma_2})

\overline{[\text{abcde}]} \, \text{Pev}_{\mu_1 \, \gamma_1} \text{ substituted to } [P_2, \gamma_0]

D \equiv u \, \overline{[\mu_1 \, \overline{[\nu_1 \, \gamma_1 \, \gamma_2]}]} \, \overline{[\mu_1 \, \overline{[\text{abcde}]})} \, (\overline{[\gamma_1 \, \gamma_2]} \, \overline{[\overline{[\gamma_1 \, \gamma_2]}]} \, \overline{[\gamma_1(x)]} \, \overline{[\gamma_2]} \, \text{Pev}_{\mu_2 \, \gamma_2})

\overline{[\text{abcde}]} \, \text{Pev}_{\mu_1 \, \gamma_1} \text{ substituted to } [P_2, \gamma_0]

E \equiv v \, \overline{[\mu_0 \, (\mu_1 \, \nu_1 \, \overline{[\nu_1 \, \gamma_1 \, \gamma_2]} \, \overline{[\overline{[\nu_1 \, \gamma_1 \, \gamma_2]}]} \, \overline{[\gamma_1]} \, \overline{[\gamma_2]} \, \overline{[\gamma_1 \, \gamma_2]} \} \, (\overline{[\text{abcde}]}) \, (\nu_1 \, \gamma_1 \, \gamma_2) \, (\nu_2 \, \gamma_2) \, \text{Pev}_{\mu_2 \, \gamma_2})

\overline{[\text{abcde}]} \, \text{Pev}_{\mu_1 \, \gamma_1} \text{ substituted to } [P_2, \gamma_0]

F \equiv w \, \overline{[\mu_1 \, \nu_1 \, \overline{[\nu_1 \, \gamma_1 \, \gamma_2]} \} \, (\overline{[\text{abcde}]}) \, (\nu_1 \, \gamma_1 \, \gamma_2) \, (\nu_2 \, \gamma_2) \, \text{Pev}_{\mu_2 \, \gamma_2})

\overline{[\text{abcde}]} \, \text{Pev}_{\mu_1 \, \gamma_1} \text{ substituted to } [P_2, \gamma_0]

\text{Beginning of the encoding } [\sum_{i=1}^{n} \pi_i, P_2, \gamma_0]

\left(\sum_{r=\frac{n}{2}+1}^{n} \pi_i \, [\overline{[\text{abcde}]}) \, \overline{[\text{abcde}]}) \, (\text{Pev}_{\mu_1 \, \gamma_1}) \, + \, \sum_{r=\frac{n}{2}+1}^{n} \pi_i \, [\overline{[\text{abcde}]}) \, \overline{[\text{abcde}]}) \, (\text{Pev}_{\mu_1 \, \gamma_1})\right)
The term $P(\omega, \gamma_0)$ is the parallel composition of the encoding of the annotated term $\omega$ on $\mu_0$ and of $(\operatorname{Pev} \mu_0 \gamma_0)$. It represents the partial evaluation of a term $P$, annotated $\omega$. The residual code is sent onto $\gamma_0$. $P(\omega)$ is the parallel composition of $P(\omega, \gamma_0)$ and $(\operatorname{Eval} \gamma_0)$, i.e., the concurrent execution of three terms: the encoding, the partial evaluator, and the interpreter. We prove the correctness of the partial evaluation by comparing the behaviors of $P(\omega)$ and $P$. Note that we only consider well-annotated terms. No warranty is given for partial evaluation correctness of non-well-annotated terms.

**Lemma 13 (Compositionality of $P$)** Let $P$ be a $\pi$-term and $\omega$ a well-annotated two-level $\pi$-term associated to $P$. Then, the following properties are verified:

1. $\omega \equiv \omega_1 \parallel \omega_2 \Rightarrow P(\omega) \simeq P(\omega_1) \parallel P(\omega_2)$.
2. $\omega \equiv \omega_1 \Rightarrow P(\omega) \simeq P(\omega_1)$.
3. $\omega \equiv (\nu x)_{\omega_1} \Rightarrow P(\omega) \simeq (\nu x)P(\omega_1)$.
4. $\omega \equiv \sum_{i=1}^{n} \pi_i P_i \Rightarrow P(\omega) \simeq \sum_{i=1}^{n} \pi_i P(\omega_i)$, the sum being either static or dynamic.

**Proof**

The proof is by induction on the structure of $P$. Cases $\omega \equiv \omega_1 \parallel \omega_2$, $\omega \equiv \omega_1$, $\omega \equiv (\nu x) \omega_1$ are similar to those of Lemma 8. For sum processes two cases have to be distinguished. The difference appears in Equations (19) and (21). Note in both cases the location of the communication $\pi_i$ (recall that $\pi_i$ denotes $\alpha_i(x_i)$ or $\nu x_i$).

- if $\omega = \sum_{i=1}^{n} \pi_i P_i$, i.e., all communications are dynamic, we have

$$
[\omega, \mu_0] 
\equiv \left[ \nu x \text{stau}_{\omega} [\pi_i \text{stau}_{\mu_i}] \nu x_1 \ldots \nu x_n \right]_{\pi_i \mu_i} 
\mid P_1[x_1, \ldots, x_n, \nu x_1 \ldots x_n] 
\parallel \sum_{i=1}^{n} \eta_i(x_i, [P_i, \mu_i]) 
\parallel \sum_{i=1}^{n} \eta_i(x_i, [P_i, \mu_i]) 
\parallel \sum_{i=1}^{n} \alpha_i(x_i, \nu x_1 \ldots x_n), (\text{Pev} \mu_i \gamma_i) 
\parallel \sum_{i=1}^{n} \alpha_i(x_i, \nu x_1 \ldots x_n), (\text{Pev} \mu_i \gamma_i) 
\parallel \sum_{i=1}^{n} \alpha_i(x_i, \nu x_1 \ldots x_n), (\text{Pev} \mu_i \gamma_i)
$$

as the annotated term. Consequently

$$
P(\omega) 
\equiv \left[ \nu x \text{stau}_{\omega} [\pi_i \text{stau}_{\mu_i}] \nu x_1 \ldots \nu x_n \right]_{\pi_i \mu_i} 
\mid \sum_{i=1}^{n} \eta_i(x_i, [P_i, \mu_i]) 
\mid \sum_{i=1}^{n} \delta_i(x_i, [\nu x_1 \ldots x_n], (\text{Pev} \mu_i \gamma_i)) 
\mid \sum_{i=1}^{n} \delta_i(x_i, [\nu x_1 \ldots x_n], (\text{Pev} \mu_i \gamma_i)) 
\mid \sum_{i=1}^{n} \alpha_i(x_i, \nu x_1 \ldots x_n), (\text{Eval} \gamma_i)
$$

Call $P'$ the term defined by (19). We have $P' \downarrow_{\alpha_i}$, $1 \leq i \leq n$. Assume now that $P'$ communicates on $\alpha_i$.

$$
P' 
\simeq \sum_{i=1}^{n} [P_i, \mu_i] 
\parallel \sum_{i=1}^{n} (\text{Pev} \mu_i \gamma_i) 
\parallel (\text{Eval} \gamma_i) 
\simeq P(\omega_i)
$$

On the other hand, $\sum_{i=1}^{n} \pi_i P(\omega_i) \downarrow_{\alpha_i}$, and also reduces to $P(\omega_i)$ after communication on $\alpha_i$. Furthermore, the term $\omega_i$ occurring in $P(\omega_i)$ is well-annotated since the annotation $[\sum_{i=1}^{n} \pi_i \Gamma_i]_{\sum_{i=1}^{n} \pi_i}$ ensures that

$$
(\Gamma \rightarrow^* \emptyset) 
\iff \left( \forall i, 1 \leq i \leq n, \Gamma_i \rightarrow^* \emptyset \right)
$$

- if $\omega = \sum_{i=1}^{n} \pi_i P_i$, i.e., some communications are static, we have

$$
P(\omega) 
\equiv \left[ \nu x \text{stau}_{\omega} [\pi_i \text{stau}_{\mu_i}] \nu x_1 \ldots \nu x_n \right]_{\pi_i \mu_i} 
\mid \sum_{i=1}^{n} \eta_i(x_i, [P_i, \mu_i]) 
\mid \sum_{i=1}^{n} \delta_i(x_i, [\nu x_1 \ldots x_n], (\text{Pev} \mu_i \gamma_i)) 
\mid \sum_{i=1}^{n} \delta_i(x_i, [\nu x_1 \ldots x_n], (\text{Pev} \mu_i \gamma_i)) 
\mid \sum_{i=1}^{n} \alpha_i(x_i, \nu x_1 \ldots x_n), (\text{Eval} \gamma_i)
$$

as the annotated term. Again

$$
P(\omega) 
\equiv \left[ \nu x \text{stau}_{\omega} [\pi_i \text{stau}_{\mu_i}] \nu x_1 \ldots \nu x_n \right]_{\pi_i \mu_i} 
\mid \sum_{i=1}^{n} \eta_i(x_i, [P_i, \mu_i]) 
\mid \sum_{i=1}^{n} \delta_i(x_i, [\nu x_1 \ldots x_n], (\text{Pev} \mu_i \gamma_i)) 
\mid \sum_{i=1}^{n} \delta_i(x_i, [\nu x_1 \ldots x_n], (\text{Pev} \mu_i \gamma_i)) 
\mid \sum_{i=1}^{n} \alpha_i(x_i, \nu x_1 \ldots x_n), (\text{Eval} \gamma_i)
$$

Write $P'$ for the term in (21). $\sum_{i=1}^{n} \pi_i P(\omega_i) \downarrow_{\alpha_i}$, $1 \leq i \leq n$. We have to show that this property is still verified under reduction. Since $\sum_{i=1}^{n} \pi_i P_i$ is annotated static, the process $\Psi \equiv \sum_{i=1}^{n} \pi_i \Gamma_i$ occurs in $\Gamma$. Since $\omega$ is well-annotated, $\Gamma \vdash P': \omega$ for some $\Gamma$ such that $\Gamma \rightarrow^* \emptyset$. Thus

$$
\Gamma \vdash \Psi \vdash P': \omega' \text{ and } \Psi \vdash P': \omega_i, \text{ for some } i < n .
$$

Therefore, $P$ communicates on $\alpha_i$. The partial evaluation process is not blocked and we obtain

$$
P(\omega) 
\simeq \sum_{i=1}^{n} [P_i, \mu_i] 
\parallel (\text{Pev} \mu_i \gamma_i) 
\parallel (\text{Eval} \gamma_i) 
\simeq P(\omega_i)
$$

which is structurally equivalent to $\sum_{i=1}^{n} \pi_i P(\omega_i)$ after communication on $\pi_i$. Additionally $\omega_i$ is well-annotated since $\Gamma \rightarrow \Gamma_i$ if we consider that a communication on $\alpha_i$ occurs.
Proposition 14 (Correctness of partial evaluation) Let $P$ be a $\pi$-term. The following property holds:

$$\vdash P : \omega \Rightarrow P(\omega) \approx P$$

(22)

Remember that $\vdash P : \omega$ means that $\Gamma \vdash P : \omega$ for some $\Gamma$ such that $\Gamma \rightarrow^* \emptyset$. Proposition 14 states that if a $\pi$-term $P$ is annotated under assumption $\Gamma$, and if $\Gamma \rightarrow^* \emptyset$, then the evaluation using Eval of the residual program produced by partial evaluation of $\omega$ is weakly equivalent to the original program $P$.

Proof
The proof is by induction on the structure of the annotated term $\omega$ and is similar to the one of Proposition 9. The hypothesis $\vdash P : \omega$ allows the use of Lemma 13.

$\square$

7 Discussion

First, note we have considered the asynchronous $\pi$-calculus. Otherwise, if we consider synchronous communications, Pev may be a lazy partial evaluator in the sense that continuations are recursively partially evaluated only when required. For instance, let us examine the parallel composition of two processes $P_1$ and $P_2$. If communications on $\pi_0$ are synchronous, the process is stopped as long as $\text{Eval}(\pi_0)$ is not present. Partial evaluations of $P_1$ and $P_2$ will be done only when required by $\text{Eval}(\pi_0)$. The same behavior arises for $P_1$ and $(\pi_0)P$. When a dynamic communication $\pi$ is encountered, its encoding is sent onto $\pi_0$ and the partial evaluation of the continuation will be deferred until $\pi$ is evaluated. In this last case, only the continuation $P_1$ of the selected term is partially evaluated. $P_j$, $j \neq i$ is never specialized.

However, Pev sketches the scheme of partial evaluation for a parallel language and any implementation of a specializer based on it would enforce continuations to be computed statically. In addition, Pev satisfies the property

$$P \rightarrow^* \emptyset \Rightarrow P(P) \rightarrow^* \emptyset$$

(23)

This means that Pev does not generate useless processes that may stay in the environment after partial evaluation. It is possible to write an eager partial evaluator for $\pi$-calculus by computing the continuations concurrently to the communications on $\pi_0$ and $\pi_0$. For instance $B$ would roughly be of the form

$$B \equiv s.\mu(\mu_1.\mu_2.(\text{uncl}(\text{code}),\overline{\text{f}(\text{code})}(\pi_1\pi_2)})$$

with respect to the new encoding:

$$[P_1 || P_2, \mu_1] = (\text{runshw})[\overline{\text{f}(\text{code})}(\pi_1\pi_2)]$$

$$[\overline{\text{f}(\mu_1\mu_2)}] || [P_1, \mu_1] || [P_2, \mu_2]$$

However this eager partial evaluator may generate processes that will never be reduced. These processes are unobservable since they communicate on reserved channels that nobody else knows of. Therefore, proofs of correctness are still valid since they are based on observability. Equation (23) becomes

$$P \rightarrow^* \emptyset \Rightarrow P(P) \rightarrow^* Q$$

such that $Q \approx \emptyset$.

8 Conclusion

In this paper, we discussed self-applicable partial evaluation for pure $\pi$-calculus. This language models concurrent behavior of parallel and object-oriented systems. We used the three-steps methodology consisting of writing a meta-interpreter, introducing an abstract analysis and exhibiting a self-applicable partial evaluator. We proved the correctness of Pev w.r.t. the weak reduction equivalence.

First of all, let us remark that every operator in the language is required in order to obtain reflection. A self-interpreter cannot be written for a strict subset of the $\pi$-calculus. Parallel composition allows application of the evaluator to the encoding of the terms. Replication is needed for recursion. Restriction is required for observability, and sums allow conditional choices.

Second, note that Eval is a parallel interpreter. In addition, our approach is compatible with Futamura's projections. Thus, a compiler Comp can be automatically generated, Comp compiles parallel programs, and the compiled process itself is parallel. In addition, Comp's correctness is a corollary of Eval's and Pev's ones. A compiler generator Cogen can also be produced. Its inputs is a parallel interpreter, i.e., the parallel semantics of any language. It is transformed into a parallel compiler.

Now, our considerations focus on an implementation yielding Cogen. This implementation could be achieved using Pict [15], a language based on $\pi$-calculus. However, this is not clear that Pict's strong type system will be well suited for reflection.

Finally, we believe that the principles of Pev could successfully be applied to the kernel of a real concurrent language with distinct variables and constants. Difficulties are to define such a reflective language. In addition, the BTA should produce two different kinds of informations. First, the static communications that may be executed at compile-time. In this context, variables are unknown either if they depend on a dynamic expression or if their assignments are related to dynamic communications.

References


