A decompilation of the pi-calculus and its application to termination

Roberto Amadio

▶ **To cite this version:**

Roberto Amadio. A decompilation of the pi-calculus and its application to termination. 2011. hal-00565222

HAL Id: hal-00565222
https://hal.science/hal-00565222

Submitted on 11 Feb 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A decompilation of the π-calculus and its application to termination

Roberto M. Amadio
Université Paris Diderot
(UMR CNRS 7126)

February 11, 2011

Abstract

We study the correspondence between a concurrent lambda-calculus in administrative, continuation passing style and a pi-calculus and we derive a termination result for the latter.

1 Introduction

There are two complementary explanations of the π-calculus. The first one is to regard it as an extension of a rather standard process calculus such as CCS while the second one is to present it as an intermediate language for compiling higher-order languages including various imperative/concurrent extensions of λ-calculi and object-oriented calculi. The first view was put forward in the original presentation [10] and explains the transfer of effective operational semantics techniques from CCS to the π-calculus. The second view gradually emerged through a series of encodings starting from, *e.g.*, [8] and it explains the expressivity of the calculus while providing guidance in selecting its essential aspects.

Taking the second view, it has been stressed (see, *e.g.*, [4]) that the translations from the λ-calculus to the π-calculus can be understood as the *composition* of two familiar compilation techniques. In the first step, the λ-term is put in an *administrative form* (AF) where all values are explicitly named and in the second one a *continuation passing style (CPS) translation* is applied so that the evaluation contexts are passed explicitly as an argument.

We note that neither the notion of administrative form nor that of CPS translation are canonical. Our purpose here is to provide a concrete presentation of this approach for a call-by-value λ-calculus and then for a parallel and concurrent extension of it. We show that the two compilation steps commute nicely with the reduction relations and the typing disciplines. Moreover, we identify languages which contain the image of the compilation and are isomorphic to natural fragments of the π-calculus. The situation is summarized in table 1 where the λ-calculi in administrative form play a prominent role since on one hand the ordinary λ-calculi can be regarded as a *retraction* of the administrative ones (symbol $\lhd$) and on the other hand they contain sub-calculi (symbol $\supset$) in CPS form which are isomorphic (symbol $\cong$) to π-calculi. Also it is at this level, that it is natural to introduce a notion of concurrent access to a resource, *i.e.*, a definition. In this framework, one can *decompile* terms of the π-calculus into familiar λ-terms. As an application of the correspondence, we show that the termination of a (concurrent) λ-calculus entails the termination of a corresponding

	$\lambda-$ notation		Adm. Form (AF)		AF in CPS style		$\pi-$ notation
Functional	λ	$\lhd$	λ^a	$\supset$	λ^{ak}	$\cong$	π^f
$\cap$							
Concurrent	$\lambda_{\parallel}$	$\lhd$	$\lambda^a_{\parallel}$	$\supset$	$\lambda^{ak}_{\parallel}$	$\cong$	π

Table 1: Overview of calculi and their relationships

(concurrent) π-calculus. Section 2 covers the functional case, section 3 generalizes it to the parallel and concurrent case, and section 4 makes explicit the correspondence with the π-calculus. Omitted specifications, some concurrent programming examples, and proof sketches of the main results are available in appendices A, B, and C, respectively.

Related work This work arises out of a long term effort of presenting the π-calculus to an audience familiar with the λ-calculus but not necessarily with process calculi. We tried this first in the context of a book [3] and then more recently in the context of a graduate course [2]. The application to termination appears as a natural test for the presented compilation techniques and can be regarded as a natural continuation of recent work on the termination of (higher-order) concurrent calculi. In this respect, we find it remarkable that in the presented approach one can drop completely the notion of *stratified region* [5, 1, 13, 6]. As far as the π-calculus is concerned, we believe the presented approach complements those described in [14, 12] in that we reduce the termination of a fragment of the π-calculus to the termination of the λ-calculus by 'elementary' means. In particular, we do not need to develop reducibility candidates/logical relations techniques for the π-calculus, nor do we require specific knowledge of the operational semantics of the π-calculus.

Requirements The reader is supposed to be acquainted with the simply typed λ-calculus, see, *e.g.*, [7], its evaluation strategies and continuation passing style translations, see, *e.g.*, [11], and to have some familiarity with the syntax of the π-calculus, see, *e.g.*, [9] and its reduction semantics. We shall make no use of the so called labelled transition systems and related notions of bisimulation.

Renaming In the following calculi, all terms are manipulated up to α-renaming of bound names. Whenever a structural congruence or a reduction rule is applied, it is assumed that terms have been renamed so that all binders use distinct variables and these variables are distinct from the free ones. Similar conventions are applied when performing a substitution, say $[T/x]T'$, of a term T for a variable x in a term T'. We denote with $FV(T)$ the set of variables occurring free in a term T.

Syntax, Structural congruence, Reduction, and Typing For each calculus, we specify the syntactic categories, the notion of structural congruence, the reduction rules, and the typing rules. In all calculi, we assume a syntactic category *id* of identifiers (or variables) which we denote with $x, y, z, \ldots$. Structural congruence is the least equivalence relation, denoted with $\equiv$, which is induced by the displayed equations, α-renaming (which is always left implicit), and closed under the operators of the language. The reduction relation is the least binary relation $\rightarrow$ that includes the pairs given by the rewriting rules and such that $M \rightarrow N$ if $M \equiv M' \rightarrow N' \equiv N$. The basic judgment of the typing rules assigns a type to

SYNTAX

$$
\begin{array}{lll}
V & ::= * \mid (\lambda id.M) \mid id & \text{(values)} \\
M & ::= V \mid (MM) & \text{(terms)} \\
E & ::= [\,] \mid E[[\,]M] \mid E[V[\,]] & \text{(eval. contexts)} \\
A & ::= 1 \mid (A \to A) & \text{(types)}
\end{array}
$$

REDUCTION RULE

$$(\beta_V) \qquad E[(\lambda x.M)V] \to E[[V/x]M]$$

TYPING

$$(id)\ \frac{x : A \in \Gamma}{\Gamma \vdash x : A} \qquad (*)\ \frac{}{\Gamma \vdash * : 1}$$

$$(\lambda)\ \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B} \qquad (@)\ \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

Table 2: A simply typed, call-by-value λ-calculus: λ

a term in a context Γ. The latter is a function mapping a finite set of variables to types. When writing $\Gamma, x : A$ it is assumed that x is not in the domain of definition of Γ. In the parallel/concurrent extensions we distinguish between value types and types. The latter are composed of value types plus a distinct behavior type b. Terms of a behavior type do not return a result. As such a behavior type cannot occur in a context or as the type of the argument of a function.

Vectorial notation We shall write X^+ (X^*) for a non-empty (possibly empty) finite sequence $X_1, \ldots, X_n$ of symbols. By extension, $\lambda x^+.M$ stands for $\lambda x_1 \ldots \lambda x_n.M$, $A^+ \to B$ stands for $(A_1 \to \cdots \to (A_n \to B) \cdots)$, $[V^+/x^+]M$ stands for $[V_1/x_1](\cdots [V_n/x_n]M \cdots)$, $x^+ : A^+$ stands for $x_1 : A_1, \ldots, x_n : A_n$, $\Gamma \vdash M^+ : A^+$ stands for $\Gamma \vdash M_1 : A_1, \ldots, \Gamma \vdash M_n : A_n$, and $\mathsf{let}\ (x = V)^+\ \mathsf{in}\ M$ stands for $\mathsf{let}\ x_1 = V_1\ \mathsf{in}\ \cdots \mathsf{let}\ x_n = V_n\ \mathsf{in}\ M$.

2 The functional case

In this section we explore the functional case, namely the upper part of table 1. This has at least two advantages: first one can get an idea of the approach in a simple familiar framework and second it clarifies the additions to be made to achieve parallelism and concurrency.

A λ-calculus To start with we introduce in table 2 a standard, simply typed, call-by-value λ-calculus (λ). We specify, syntax, reduction and typing rules. We follow this pattern in the following calculi too, possibly adding the specification of a structural congruence.

A λ-calculus in administrative form A corresponding calculus in *administrative form* (λ^a) is presented in table 3. The basic idea is to attribute a name to each value and to compute by replacing names with names. Notice that in λ^a we restrict the values in a let definition to be either abstractions or constants '$*$'. Beyond values and terms, we introduce a new syntactic category of 'declarations' which are terms possibly preceded by a list of value declarations. Strictly speaking the application only applies to terms, however if $D_i = \mathsf{let}\ (x_i =$

SYNTAX

$$
\begin{array}{lll}
V & ::= * \mid (\lambda id^{+}.D) & \text{(values)} \\
D & ::= \mathsf{let}\ (id = V)^{*}\ \mathsf{in}\ M & \text{(declarations)} \\
M & ::= id \mid @(M, M^{+}) & \text{(terms)} \\
E & ::= \mathsf{let}\ (id = V)^{*}\ \mathsf{in}\ [\,] \mid E[@(id^{*}, [\,], M^{*})] & \text{(eval. contexts)} \\
A & ::= \mathcal{N}(1) \mid \mathcal{N}(A^{+} \to A) & \text{(types)}
\end{array}
$$

STRUCTURAL CONGRUENCE

$$
\begin{array}{l|l}
(eq_1)\quad \mathsf{let}\ x_1 = V_1\ \mathsf{in}\ \mathsf{let}\ x_2 = V_2\ \mathsf{in}\ D \equiv \mathsf{let}\ x_2 = V_2\ \mathsf{in}\ \mathsf{let}\ x_1 = V_1\ \mathsf{in}\ D \quad \text{if } x_1 \notin FV(V_2), x_2 \notin FV(V_1) & (eq_2)\quad \mathsf{let}\ x = V\ \mathsf{in}\ D \equiv D \quad \text{if } x \notin FV(D)
\end{array}
$$

REDUCTION RULE

$$
(\beta_V^a) \qquad E[\mathsf{let}\ x = \lambda y^{+}.D\ \mathsf{in}\ E'[@(x, z^{+})]] \to E[\mathsf{let}\ x = \lambda y^{+}.D\ \mathsf{in}\ E'[[z^{+}/y^{+}]D]]
$$

TYPING

$$
(id^a)\quad \frac{x : A \in \Gamma}{\Gamma \vdash^a x : A} \qquad (*^a)\quad \frac{\Gamma, x : \mathcal{N}(1) \vdash D : A}{\Gamma \vdash^a \mathsf{let}\ x = *\ \mathsf{in}\ D : A}
$$

$$
(\lambda^a)\quad \frac{\Gamma, y^{+} : A^{+} \vdash^a D' : C \quad \Gamma, x : \mathcal{N}(A^{+} \to C) \vdash^a D : B}{\Gamma \vdash^a \mathsf{let}\ x = \lambda y^{+}.D'\ \mathsf{in}\ D : B} \qquad (@^a)\quad \frac{\Gamma \vdash^a M : \mathcal{N}(A^{+} \to B) \quad \Gamma \vdash^a N^{+} : A^{+}}{\Gamma \vdash^a @(M, N^{+}) : B}
$$

Table 3: The λ-calculus in administrative form: λ^a

$V_i)^*$ in M_i, for $i = 1, \ldots, n$, then we regard $@(D_0, D_1, \ldots, D_n)$ as an abbreviation for let $(x_0 = V_0)^*$ in $\cdots$ let $(x_n = V_n)^*$ in $@(M_0, \ldots, M_n)$ (the order of the declarations is immaterial up to structural congruence). Similar care is needed when substituting a declaration $D = \mathsf{let}$ $(x = V)^*$ in M in an evaluation context E. We remark that E can be written as let $(x' = V')^*$ in E' where E' does not start with a let declaration. Then by $E[D]$ we mean the declaration let $(x = V)^*$ in let $(x' = V')^*$ in $E'[M]$ where it is intended that the names x'^* do not occur free in E'. [1]

We use $\mathcal{N}(A)$ for the type of names carrying values of type A; as the reader might have guessed these names correspond to the channel names of the π-calculus. The λ^a-calculus is *polyadic* in the sense that the application may have any finite, positive number of arguments. This choice allows to represent directly the following CPS translation as a translation from the λ^a calculus to a fragment of the λ^a calculus in 'CPS form' and moreover the latter corresponds directly to a *polyadic* π-calculus. Nevertheless, we notice that the λ^a-calculus contains a *monadic sub-calculus* that we denote with λ^{am} where the types are restricted as follows:

$$
A ::= \mathcal{N}(1) \mid \mathcal{N}(A \to A) \qquad \text{(monadic types)}
$$

This sub-calculus is closed under reduction (provided we consider typable terms) and it suffices to encode the simply typed λ-calculus.

Back and forth between λ and λ^a In table 4, we introduce a translation of λ-terms to administrative forms along with a readback translation. Clearly, there are λ^a-terms which are

[1] There is an alternative presentation where declarations and applications can be inter-mixed freely; we found the current presentation more handy for our purposes.

Translation in AF

$$\underline{1} = \mathcal{N}(1) \qquad \underline{A \to B} = \mathcal{N}(\underline{A} \to \underline{B})$$
$$\underline{x} = x \qquad \underline{*} = \mathsf{let}\ x = * \ \mathsf{in}\ x \qquad \underline{\lambda x.M} = \mathsf{let}\ x = \lambda x.\underline{M}\ \mathsf{in}\ x \qquad \underline{MN} = @(\underline{M}, \underline{N})$$

Readback

$$\mathcal{N}(1)^o = 1 \qquad \mathcal{N}(A_1 \to \cdots \to A_k \to B)^o = A_1^o \to \cdots \to A_k^o \to B^o$$
$$*^o = * \qquad (\lambda y^+.D)^o = \lambda y^+.D^o$$
$$x^o = x \qquad (\mathsf{let}\ x = V\ \mathsf{in}\ D)^o = [V^o/x]D^o \qquad @(M, N_1, \ldots, N_k)^o = M^o N_1^o \cdots N_k^o$$

Table 4: Translation in administrative form and readback

not structurally equivalent but are mapped to the same λ-term by the readback translation. For instance, taking: $V \equiv \lambda z.z$, $M \equiv \mathsf{let}\ x = V\ \mathsf{in}\ @(x,x)$, and $N \equiv \mathsf{let}\ x = V\ \mathsf{in}\ \mathsf{let}\ y = V\ \mathsf{in}\ @(x,y)$, we have that $M \not\equiv N$ in λ^a but $M^o \equiv N^o$ in λ. Thus we can regard the administrative forms as *notations* for λ terms which differ in the amount of value sharing. Conversely, given a λ-term such as VV, we can associate with it an administrative form $\underline{VV} = @(\mathsf{let}\ x = V\ \mathsf{in}\ x, \mathsf{let}\ y = V\ \mathsf{in}\ y) \equiv N$. Thus the translation $\underline{\ }$ in administrative form makes no effort to *share* identical values. The translation and the related readback function form a *retraction pair* which is 'compatible' with typing and reduction in a sense that is made formal below. Thus, we can look at the λ-calculus as a *retract* of the λ^a calculus.

In establishing the simulations between λ and λ^a we *cannot* work directly with the translation $\underline{\ }$. For instance, taking $M = (\lambda x.x(xy))(\lambda z.z)$, we have $M \to M'$ in λ and $\underline{M} \not\to \underline{M'}$. Indeed in M' the term $\lambda z.z$ is duplicated while its translation is shared in $\underline{M'}$. We get around this difficulty by working with the readback operation.

Theorem 1 (read-back translation, functional case) *The following properties hold.*

1. *If $D_1 \equiv D_2$ in λ^a then $D_1^o \equiv D_2^o$ in λ.*
2. *If M is a term in λ then $\underline{M}^o \equiv M$.*
3. *If $\Gamma \vdash M : A$ in λ then $\underline{\Gamma} \vdash^{am} \underline{M} : \underline{A}$ in λ^a.*
4. *If $\Gamma \vdash^a D : A$ in λ^a then $\Gamma^o \vdash D^o : A^o$ in λ.*
5. *If $\Gamma \vdash^a D_1 : A$, $M_1 \equiv D_1^o$ and $D_1 \to D_2$ in λ^a then $M_1 \xrightarrow{+} M_2$ in λ and $M_2 \equiv D_2^o$.*
6. *If $\Gamma \vdash^{am} D_1 : A$, $M_1 \equiv D_1^o$ and $M_1 \to M_2$ in λ then $D_1 \to D_2$ in λ^{am} and $M_2 \equiv D_2^o$.*

The first property states that the readback translation is invariant under structural congruence and the second that it is the left inverse of the function that puts a λ-term in administrative form. The third and fourth properties state that typing is preserved by the translations. The fifth property shows that any reduction in λ^a corresponds to a *positive* number of reductions of the readback in λ (hence the following corollary). Finally, the sixth property guarantees that the monadic administrative forms are indeed enough to simulate the λ-calculus.

Corollary 2 *The λ^a-calculus terminates.*

RESTRICTED CPS SYNTAX

$$
\begin{array}{lll}
V & ::= * \mid (\lambda id^{+}.D) & \text{(values)} \\
D & ::= \mathsf{let}\ (id = V)^{*}\ \mathsf{in}\ M & \text{(declarations)} \\
M & ::= @(id, id^{+}) & \text{(terms)} \\
E & ::= \mathsf{let}\ (id = V)^{*}\ \mathsf{in}\ [\,] & \text{(eval. contexts)} \\
A & ::= \mathcal{N}(1) \mid \mathcal{N}(A^{+} \to R) & \text{(types)}
\end{array}
$$

SPECIALIZED TYPING RULES

$$
(*^{a})\quad \frac{\Gamma, x : \mathcal{N}(1) \vdash D : R}{\Gamma \vdash^{a} \mathsf{let}\ x = *\ \mathsf{in}\ D : R} \qquad (@^{a})\quad \frac{x : \mathcal{N}(A^{+} \to R), y^{+} : A^{+} \in \Gamma}{\Gamma \vdash^{a} @(x, y^{+}) : R}
$$

$$
(\lambda^{a})\quad \frac{\Gamma, y^{+} : A^{+} \vdash^{a} D' : R \quad \Gamma, x : \mathcal{N}(A^{+} \to R) \vdash^{a} D : R}{\Gamma \vdash^{a} \mathsf{let}\ x = \lambda y^{+}.D'\ \mathsf{in}\ D : R}
$$

Table 5: Administrative forms in CPS style: λ^{ak}

$$
\begin{array}{ll}
\overline{\mathcal{N}(1)} & = \mathcal{N}(1) \\
\overline{\mathcal{N}(A_1 \to \cdots \to A_n \to B)} & = \mathcal{N}(\overline{A_1} \to \cdots \to \overline{A_n} \to K(B) \to R) \\
 & \text{where: } K(B) = \mathcal{N}(\overline{B} \to R) \\
 & \\
\psi(*) & = * \\
\psi(\lambda y^{+}.D) & = \lambda y^{+}.\lambda k.(D : k) \\
\mathsf{let}\ (x = V)^{*}\ \mathsf{in}\ M : k & = \mathsf{let}\ (x = \psi(V))^{*}\ \mathsf{in}\ M : k \\
@(x^{*}, @(M, M^{+}), N^{*}) : k & = \mathsf{let}\ k' = \lambda y.@(x^{*}, y, N^{*}) : k\ \mathsf{in}\ @(M, M^{+}) : k' \\
@(x, x^{+}) : k & = @(x, x^{+}, k) \\
x : k & = @(k, x)
\end{array}
$$

Table 6: An optimized CPS translation on the administrative forms

Administrative forms in CPS style In table 5 we consider a restriction of the syntax of the terms where we drop variables (functions are always applied to their arguments) and the operator @ is only applied to variables. Evaluation contexts are then restricted accordingly. The definition of structural congruence and reduction are inherited from λ^{a} and are omitted. We restrict the syntax of types too by requiring, as it is common in CPS translations, that there is a fixed type of results called R. The resulting language is called λ^{ak} and it is a subsystem of λ^{a} which inherits from λ^{a} the reduction and typing rules. In particular, the terms in λ^{ak} terminate because those in λ^{a} do.

CPS translation In table 6, we describe a CPS translation of the administrative language λ^{a} into λ^{ak}. The reader familiar with CPS translations, may appreciate the fact that the translation has been optimized so as to have a simple statement and proof of the simulation property. In particular, notice that the case for application is split into two cases.

Theorem 3 (CPS translation, functional case) *The following properties hold.*

1. *If* $\Gamma \vdash^{a} D : A$ *then* $\overline{\Gamma}, k : K(A) \vdash^{a} (D : k) : R$.
2. *If* $\Gamma \vdash^{a} D : A$ *and* $D \to D'$ *in* λ^{a} *then* $(D : k) \xrightarrow{+} (D' : k)$ *in* λ^{ak}.

Syntax

V	$::= * \mid (\lambda id.M) \mid id$	(values)
M	$::= V \mid (MM) \mid (M \mid M)$	(terms)
E	$::= [\,] \mid E[[\,]M \mid E[V[\,]] \mid E[[\,] \mid M] \mid E[M \mid [\,]]$	(eval. contexts)
A	$::= 1 \mid (A \to \alpha)$	(value types)
α	$::= A \mid b$	(types)

New typing rule

$$(|) \quad \frac{\Gamma \vdash M_i : b \quad i = 1, 2}{\Gamma \vdash (M_1 \mid M_2) : b}$$

Table 7: Sketch of a parallel λ-calculus: $\lambda_{\|}$

3 Parallel and Concurrent extensions

In this section, we explore the more general concurrent case, namely the lower part of table 1.

A parallel λ-calculus In table 7, we introduce a parallel version of the λ-calculus (cf. appendix A, table 10). This amounts to introduce a binary parallel composition operator on terms along with a special *behavior* type b which is attributed to terms running in parallel. The reduction rule and the typing rules (id), $(*)$, (λ), $(@)$ are omitted since they are similar to the ones in table 2. Terms running in parallel are not supposed to return a value. The typing guarantees that they cannot occur under an application (neither as a function nor as an argument). Notice that in this language terms running in parallel are not really competing for the resources, *i.e.*, the value declarations. This is because values are always available and stateless (for this reason we call this calculus parallel rather than concurrent). An interesting remark is that the termination of the parallel λ-calculus can be derived from the termination of the simply typed λ-calculus.

Proposition 4 *The $\lambda_{\|}$-calculus terminates.*

A concurrent λ-calculus in administrative form In table 8 we sketch an extension of the administrative λ-calculus to accommodate the parallelism already introduced in the λ-calculus (cf. appendix A, table 11). Thus once again we introduce parallel terms and a special behavior type b. In order to have a form of concurrency or competition among the parallel threads we associate a usage u with each declaration. A usage u varies over the set $\{\infty, 1, 0\}$. The (familiar) idea is that a declaration with usage ∞ is always available, one with usage 1 can be used at most once, and one with usage 0 cannot be used at all. We take the convention that when the usage is omitted the intended usage is ∞. We define an operator $\downarrow$ to *decrease* usages as follows: $\downarrow \infty = \infty$, $\downarrow 1 = 0$, and $\downarrow 0$ is undefined. Modulo this enrichment of the declarations with usages, the structural congruence is defined as in the functional fragment of the language (table 3) and it is omitted. Notice that a reduction is possible only if the usage of the corresponding definition is not 0 and in this case the effect of the reduction is to decrease the usage. We omit the typing rules (id^a), $(*^a)$, and $(@^a)$ which are similar to the ones in table 3.

SYNTAX

V	$::= * \mid (\lambda id^+.D)$	(values)
D	$::= \mathsf{let}_u\ (id = V)^*\ \mathsf{in}\ M$	(declarations)
M	$::= id \mid @(M, M^+) \mid (M \mid M)$	(terms)
E	$::= \mathsf{let}_u\ (id = V)^*\ \mathsf{in}\ [\,] \mid E[@(id^*, [\,], M^*)] \mid E[[\,] \mid M] \mid E[M \mid [\,]]$	(eval. contexts)
A	$::= \mathcal{N}(1) \mid \mathcal{N}(A^+ \to \alpha)$	(value types)
α	$::= A \mid b$	(types)

REDUCTION RULE

$$(\beta_V^a) \qquad E[\mathsf{let}_u\ x = \lambda y^+.D\ \mathsf{in}\ E'[@(x, z^+)]] \to E[\mathsf{let}_{\downarrow u}\ x = \lambda y^+.D\ \mathsf{in}\ E'[[z^+/y^+]D]]$$

NEW TYPING RULES

$$(\lambda^a) \quad \frac{u \neq 0 \quad \Gamma, y^+ : A^+ \vdash^a D' : \alpha' \Gamma, x : \mathcal{N}(A^+ \to \alpha') \vdash^a D : \alpha}{\Gamma \vdash^a \mathsf{let}_u\ x = \lambda y^+.D'\ \mathsf{in}\ D : \alpha}$$

$$(\lambda_0^a) \quad \frac{V \neq * \quad \Gamma, x : \mathcal{N}(A^+ \to \alpha') \vdash^a D : \alpha}{\Gamma \vdash^a \mathsf{let}_0\ x = V\ \mathsf{in}\ D : \alpha} \qquad (|^a) \quad \frac{\Gamma \vdash^a M_i : b \quad i = 1, 2}{\Gamma \vdash^a (M_1 \mid M_2) : b}$$

Table 8: Sketch of a concurrent λ-calculus in administrative form: $\lambda_{\parallel}^a$

In the typing, we require that in $\mathsf{let}_u\ x = *\ \mathsf{in}\ D$, u is ∞. Also notice that in $\mathsf{let}_0\ x = V\ \mathsf{in}\ D$ we disregard the typing of the value V. Given a declaration D, we can obtain a declaration D' by replacing all the usages with the usage ∞. It is clear that all reductions D may perform can be simulated by D'. If the transformation must respect typing then we can just replace the possibly ill-typed values in let_0 by some well-typed value. Then, as far as termination is concerned, it is enough to consider the sub-calculus where all usages are ∞; we denote this calculus with $\lambda_{\parallel,\infty}^a$. The administrative translation and the related readback translation described in table 4 are extended to provide a retraction pair between the $\lambda_{\parallel}$ and the $\lambda_{\parallel,\infty}^a$ calculi. The translations are the identity on the behavior type b and distribute over parallel compositions (cf. appendix A, table 12):

$\underline{b}$	$= b$	$\underline{M \mid M'}$	$= \underline{M} \mid \underline{M'}$	(translation in AF)
b^o	$= b$	$\overline{(M_1 \mid M_2)}^o$	$= M_1^o \mid M_2^o$	(readback)

Theorem 5 (read-back translation, parallel case) *The following properties hold.*

1. *If $D_1 \equiv D_2$ in $\lambda_{\parallel,\infty}^a$ then $D_1^o \equiv D_2^o$ in λ.*
2. *If M is a term in $\lambda_{\parallel}$ then $\underline{M}^o \equiv M$.*
3. *If $\Gamma \vdash M : \alpha$ in $\lambda_{\parallel}$ then $\underline{\Gamma} \vdash^{am} \underline{M} : \underline{\alpha}$ in $\lambda_{\parallel}^{am}$.*
4. *If $\Gamma \vdash^a D : \alpha$ in $\lambda_{\parallel,\infty}^a$ then $\Gamma^o \vdash D^o : \alpha^o$ in $\lambda_{\parallel}$.*
5. *If If $\Gamma \vdash^a D_1 : \alpha$, $M_1 \equiv D_1^o$ and $D_1 \to D_2$ in $\lambda_{\parallel,\infty}^a$ then $M_1 \xrightarrow{+} M_2$ in $\lambda_{\parallel}$ and $M_2 \equiv D_2^o$.*
6. *If $\Gamma \vdash^{am} D_1 : \alpha$ in $\lambda_{\parallel,\infty}^{am}$, $M_1 \equiv D_1^o$ and $M_1 \to M_2$ in $\lambda_{\parallel}$ then $D_1 \to D_2$ in $\lambda_{\parallel,\infty}^{am}$ and $M_2 \equiv D_2^o$.*

Restricted CPS Syntax

$$
\begin{array}{lll}
V & ::= * \mid (\lambda id^{+}.D) & \text{(values)} \\
D & ::= \mathsf{let}_u\ (id = V)^*\ \mathsf{in}\ M & \text{(declarations)} \\
M & ::= @(id, id^{+}) \mid (M \mid M) & \text{(terms)} \\
E & ::= \mathsf{let}_u\ (id = V)^*\ \mathsf{in}\ [\,] \mid E[[\,] \mid M] \mid E[M \mid [\,]] & \text{(eval. contexts)} \\
A & ::= \mathcal{N}(1) \mid \mathcal{N}(A^{+} \to b) & \text{(types)}
\end{array}
$$

New specialized typing rules

$$(\lambda^a)\quad \frac{\Gamma, y^{+} : A^{+} \vdash^a D' : b \quad \Gamma, x : \mathcal{N}(A^{+} \to b) \vdash^a D : b}{\Gamma \vdash^a \mathsf{let}_u\ x = \lambda y^{+}.D'\ \mathsf{in}\ D : b}$$

$$(\lambda_0^a)\quad \frac{V \neq *\quad \Gamma, x : \mathcal{N}(A^{+} \to b) \vdash^a D : b}{\Gamma \vdash^a \mathsf{let}_0\ x = V\ \mathsf{in}\ D : b} \qquad (|^a)\quad \frac{\Gamma \vdash M_i : b \quad i = 1, 2}{\Gamma \vdash (M_1 \mid M_2) : b}$$

Table 9: Sketch of the concurrent administrative forms in CPS style: $\lambda_{\parallel}^{ak}$

By theorem 5(5), a reduction of a term in $\lambda_{\parallel,\infty}^{a}$ corresponds to a *positive* number of reductions of the readback in $\lambda_{\parallel}$. Hence, by recalling proposition 4, we obtain the following corollary (cf. appendix B for concurrent programming examples).

Corollary 6 *The $\lambda_{\parallel}^{a}$-calculus terminates.*

Concurrent λ-calculus in administrative, CPS form In table 9, we introduce the concurrent λ-calculus in administrative and CPS form (cf. appendix A, table 13). The typing rules $(*^a)$ and $(@^a)$ are omitted as they are similar to the ones in table 5. The CPS translation given in table 6 is extended to a translation from $\lambda_{\parallel}^{a}$ to $\lambda_{\parallel}^{ak}$. The behavior type is mapped to itself, $\overline{b} = b$, with a continuation type $K(b)$ which is conventionally taken to be $\mathcal{N}(1)$, while the optimized term translation distributes over parallel composition $(M \mid M') : k = (M : k) \mid (M' : k)$ (cf. appendix A, table 14). Then typing and reduction are preserved as follows.

Theorem 7 (CPS translation, concurrent case) *The following properties hold.*

1. *If $\Gamma \vdash^a D : \alpha$ in $\lambda_{\parallel}^{a}$ then $\overline{\Gamma}, k : K(\alpha) \vdash^a (D : k) : b$ in $\lambda_{\parallel}^{ak}$.*
2. *If $\Gamma \vdash^a D : \alpha$ and $D \to D'$ in $\lambda_{\parallel}^{a}$ then $(D : k) \xrightarrow{+} (D' : k)$ in $\lambda_{\parallel}^{ak}$.*

4 Correspondence with the π-calculus

In this section we consider the correspondence between the $\lambda_{\parallel}^{ak}$-calculus and the π-calculus. To this end, we introduce the syntax of a π-calculus, π, where we write $\{\ldots\}$ to mean that the symbols between curly brackets are optional. The 'functional' version of this calculus (called π_f in table 1) is obtained by dropping the (non-replicated) input prefix from the syntax and it corresponds to the functional administrative CPS forms (table 5).

$$
\begin{array}{lll}
D & ::= \nu id\ D \mid \nu id\ (id(id^{+}).D \mid D) \mid \nu id\ (!id(id^{+}).D \mid D) \mid M & \text{(declarations)} \\
M & ::= \overline{id}(id^{+}) \mid (M \mid M) & \text{(terms)} \\
E & ::= [\,] \mid E[\nu id([\,])] \mid E[\nu id(\{!\} id(id^{+}.D \mid [\,]) \mid E[[\,] \mid M] \mid E[M \mid [\,]] & \text{(eval. contexts)} \\
A & ::= Ch(1) \mid Ch(A^{+}) & \text{(types)}
\end{array}
$$

The following table provide a correspondence (a bidirectional translation) between $\lambda_{\|}^{ak}$ and π. Notice that declarations of the shape $\mathsf{let}_\infty\ x = *\ \mathsf{in}\ D$ and $\mathsf{let}_0\ x = V\ \mathsf{in}\ D$ both correspond to declarations $\nu x\ D$. The structural congruence, the reduction rules, and the typing rules of π_r are exactly those of $\lambda_{\|}^{ak}$ modulo this correspondence (cf. appendix A, table 15).

	λ^{ak}	π
Types	$\mathcal{N}(1)$	$Ch(1)$
	$\mathcal{N}(A^+ \to b)$	$Ch(A^+)$
Terms	$\mathsf{let}_\infty\ x = \lambda y^+.D\ \mathsf{in}\ D'$	$\nu x\ (!x(y^+).D \mid D')$
	$\mathsf{let}_1\ x = \lambda y^+.D\ \mathsf{in}\ D'$	$\nu x\ (x(y^+).D \mid D')$
	$\mathsf{let}_0\ x = V\ \mathsf{in}\ D$	$\nu x\ D$
	$\mathsf{let}_\infty\ x = *\ \mathsf{in}\ D$	$\nu x\ D$
	$M \mid M'$	$M \mid M'$
	$@(x, y^+)$	$\overline{x}y^+$

When applying the translation from π_r to $\lambda_{\|}^{ak}$ it is intended that: (i) D does not contain a (possibly replicated) input prefix on the declared name x and (ii) the typing determines the appropriate translation (recall that in $\lambda_{\|}^{a}$ the typing of V is disregarded). The reader familiar with the π-calculus will recognize that communication is asynchronous (there is no output prefix) and polyadic (we send vectors of names) and that for every (channel) name x there is at most one associated definition (a process, possibly replicated, ready to input on x). However these constraints are not enough to guarantee termination. For instance, consider the following looping processes of the (untyped) π-calculus: $P_1 \equiv \nu x\ (!x(y).\overline{x}y \mid \overline{x}z)$, $P_2 \equiv \nu x, x'\ (!x(y).\overline{x'}y \mid !x'(y).\overline{x}y \mid \overline{x}y)$. In the presented system, both processes are rejected, the first because the definition of x refers to itself and the second because the definitions of x and x' are mutually recursive. The syntactic constraints and the typing rules guarantee that the definitions can be linearly ordered so that each definition may only refer to previously defined names.

Corollary 8 *The typed π-calculus π described above terminates (cf. appendix A, table 15).*

5 Conclusion

We have introduced a simply typed concurrent λ-calculus in administrative form, and shown that a fragment of this calculus in continuation passing style corresponds to a simply typed π-calculus. As an application of the correspondence, we have derived a termination result for the π-calculus. We expect that the current framework can be extended in various directions including: (i) polymorphic (second order) types, (ii) refinements towards linear logic/type systems, and (iii) a synchronous/timed variants of the concurrency model.

References

[1] R.M. Amadio. On stratified regions. In Proc. *APLAS*, Springer LNCS 5905: 210-225, 2009.

[2] R.M. Amadio. Lectures on extensions of basic process calculi. MPRI course *Concurrency*, 2010, Université Paris-Diderot.

[3] R.M. Amadio, P.-L. Curien. *Domains and lambda calculi.* Cambridge Tracts in Theoretical Computer Science 46. 1998.

[4] G. Boudol. The π-calculus in direct style. In Proc. *ACM-POPL*: 228-241, 1997.

[5] G. Boudol. Typing termination in a higher-order concurrent imperative language. In Proc. *CONCUR*, Springer LNCS 4703:272-286, 2007.

[6] R. Demangeon, D. Hirschkoff, D. Sangiorgi. Termination in impure concurrent languages. Proc. CONCUR 2010, SLNCS 6269: 328-342, 2010.

[7] J.-Y. Girard. *Proofs and types*. Cambridge University Press. 1989.

[8] R. Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2): 119-141, 1992.

[9] R. Milner. *Communicating and mobile systems: the pi calculus*. Cambridge University Press. 1999.

[10] R. Milner, J. Parrow, D. Walker. A calculus of mobile processes, parts 1-2. *Information and Computation*, 100(1):1–77, 1992.

[11] G. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1(2):125-159, 1975.

[12] D. Sangiorgi. Termination of processes. *Math. Struct. in Comp. Sci.*, 16:1-39, 2006.

[13] P. Tranquilli. Translating types and effects with state monads and linear logic. Manuscript, ENS Lyon, January 2010,

[14] N. Yoshida, M. Berger, K. Honda. Strong normalisation in the π-calculus. *Information and Computation*, 191(2):145-202, 2004.

Syntax

$$
\begin{array}{lll}
V & ::= * \mid (\lambda id.M) \mid id & \text{(values)} \\
M & ::= V \mid (MM) \mid (M \mid M) & \text{(terms)} \\
E & ::= [\,] \mid E[[\,]M] \mid E[V[\,]] \mid E[[\,] \mid M] \mid E[M \mid [\,]] & \text{(eval. contexts)} \\
A & ::= 1 \mid (A \to \alpha) & \text{(value types)} \\
\alpha & ::= A \mid b & \text{(types)}
\end{array}
$$

Reduction rule

$$(\beta_V) \qquad E[(\lambda x.M)V] \to E[[V/x]M]$$

Typing

$$(id)\ \frac{x : A \in \Gamma}{\Gamma \vdash x : A} \qquad (*)\ \frac{}{\Gamma \vdash * : 1}$$

$$(\lambda)\ \frac{\Gamma, x : A \vdash M : \alpha}{\Gamma \vdash \lambda x.M : A \to \alpha} \qquad (@)\ \frac{\Gamma \vdash M : A \to \alpha \quad \Gamma \vdash N : A}{\Gamma \vdash MN : \alpha}$$

$$(|)\ \frac{\Gamma \vdash M_i : b \quad i = 1, 2}{\Gamma \vdash (M_1 \mid M_2) : b}$$

Table 10: A parallel λ-calculus: $\lambda_{\|}$

A Full specifications

This section contains the full specifications of the parallel/concurrent calculi and the related translations as tables 10, 11, 12, 13, 14, and 15.

SYNTAX

$$
\begin{array}{lll}
V & ::= * \mid (\lambda id^{+}.D) & \text{(values)} \\
D & ::= \mathsf{let}_u\ (id = V)^*\ \mathsf{in}\ M & \text{(declarations)} \\
M & ::= id \mid @(M, M^{+}) \mid (M \mid M) & \text{(terms)} \\
E & ::= \mathsf{let}_u\ (id = V)^*\ \mathsf{in}\ [\,] \mid E[@(id^*, [\,], M^*)] \mid E[[\,] \mid M] \mid E[M \mid [\,]] & \text{(eval. contexts)} \\
A & ::= \mathcal{N}(1) \mid \mathcal{N}(A^{+} \to \alpha) & \text{(value types)} \\
\alpha & ::= A \mid b & \text{(types)}
\end{array}
$$

STRUCTURAL CONGRUENCE

$$
(eq_1)\quad \begin{array}{c} \mathsf{let}_{u_1}\ x_1 = V_1\ \mathsf{in}\ \mathsf{let}_{u_2}\ x_2 = V_2\ \mathsf{in}\ D \\ \equiv \mathsf{let}_{u_2}\ x_2 = V_2\ \mathsf{in}\ \mathsf{let}_{u_1}\ x_1 = V_1\ \mathsf{in}\ D \\ \text{if } x_1 \notin FV(V_2), x_2 \notin FV(V_1) \end{array}
\qquad
(eq_2)\quad \begin{array}{c} \mathsf{let}_u\ x = V\ \mathsf{in}\ D \equiv D \\ \\ \text{if } x \notin FV(D) \end{array}
$$

REDUCTION RULE

$$
(\beta_V^a)\qquad E[\mathsf{let}_u\ x = \lambda y^{+}.D\ \mathsf{in}\ E'[@(x, z^{+})]] \to E[\mathsf{let}_{\downarrow u}\ x = \lambda y^{+}.D\ \mathsf{in}\ E'[[z^{+}/y^{+}]D]]
$$

TYPING

$$
(id^a)\quad \frac{x : A \in \Gamma}{\Gamma \vdash^a x : A}
\qquad
(*^a)\quad \frac{\Gamma, x : \mathcal{N}(1) \vdash D : \alpha}{\Gamma \vdash^a \mathsf{let}_\infty\ x = *\ \mathsf{in}\ D : \alpha}
$$

$$
(\lambda^a)\quad \frac{\begin{array}{c} u \neq 0 \quad \Gamma, y^{+} : A^{+} \vdash^a D' : \alpha' \\ \Gamma, x : \mathcal{N}(A^{+} \to \alpha') \vdash^a D : \alpha \end{array}}{\Gamma \vdash^a \mathsf{let}_u\ x = \lambda y^{+}.D'\ \mathsf{in}\ D : \alpha}
\qquad
(\lambda_0^a)\quad \frac{V \neq *\quad \Gamma, x : \mathcal{N}(A^{+} \to \alpha') \vdash^a D : \alpha}{\Gamma \vdash^a \mathsf{let}_0\ x = V\ \mathsf{in}\ D : \alpha}
$$

$$
(@^a)\quad \frac{\begin{array}{c} \Gamma \vdash^a M : \mathcal{N}(A^{+} \to \alpha) \\ \Gamma \vdash^a N^{+} : A^{+} \end{array}}{\Gamma \vdash^a @(M, N^{+}) : \alpha}
\qquad
(|^a)\quad \frac{\Gamma \vdash^a M_i : b \quad i = 1, 2}{\Gamma \vdash^a (M_1 \mid M_2) : b}
$$

Table 11: A concurrent λ-calculus in administrative form: $\lambda_{\parallel}^a$

TRANSLATION IN AF

$$
\underline{1} = \mathcal{N}(1) \qquad \underline{b} = b \qquad \underline{A \to \alpha} = \mathcal{N}(\underline{A} \to \underline{\alpha})
$$

$$
\begin{array}{c}
\underline{x} = x \qquad \underline{*} = \mathsf{let}_\infty\ x = *\ \mathsf{in}\ x \qquad \underline{\lambda x.M} = \mathsf{let}_\infty\ x = \lambda x.\underline{M}\ \mathsf{in}\ x \\
\underline{MN} = @(\underline{M}, \underline{N}) \qquad \underline{M \mid M'} = \underline{M} \mid \underline{M'}
\end{array}
$$

READBACK

$$
\mathcal{N}(1)^o = 1 \qquad b^o = b \qquad \mathcal{N}(A_1 \to \cdots \to A_k \to \alpha)^o = A_1^o \to \cdots \to A_k^o \to \alpha^o
$$

$$
\begin{array}{c}
*^o = * \qquad (\lambda y^{+}.D)^o = \lambda y^{+}.D^o \qquad x^o = x \\
(\mathsf{let}_\infty\ x = V\ \mathsf{in}\ D)^o = [V^o/x]D^o \qquad @(M, N_1, \ldots, N_k)^o = M^o N_1^o \cdots N_k^o \\
(M_1 \mid M_2)^o = M_1^o \mid M_2^o
\end{array}
$$

Table 12: Translation in administrative form and readback: parallel case

RESTRICTED CPS SYNTAX

$$
\begin{array}{lll}
V & ::= * \mid (\lambda id^{+}.D) & \text{(values)} \\
D & ::= \mathsf{let}_u\ (id = V)^*\ \mathsf{in}\ M & \text{(declarations)} \\
M & ::= @(id, id^{+}) \mid (M \mid M) & \text{(terms)} \\
E & ::= \mathsf{let}_u\ (id = V)^*\ \mathsf{in}\ [\,] \mid E[[\,] \mid M] \mid E[M \mid [\,]] & \text{(eval. contexts)} \\
A & ::= \mathcal{N}(1) \mid \mathcal{N}(A^{+} \to b) & \text{(types)}
\end{array}
$$

SPECIALIZED TYPING RULES

$$(*^a)\quad \frac{\Gamma, x : \mathcal{N}(1) \vdash D : b}{\Gamma \vdash^a \mathsf{let}_\infty\ x = *\ \mathsf{in}\ D : b} \qquad (\lambda_0^a)\quad \frac{V \neq *\quad \Gamma, x : \mathcal{N}(A^{+} \to b) \vdash^a D : b}{\Gamma \vdash^a \mathsf{let}_0\ x = V\ \mathsf{in}\ D : b}$$

$$(\lambda^a)\quad \frac{\Gamma, y^{+} : A^{+} \vdash^a D' : b \quad \Gamma, x : \mathcal{N}(A^{+} \to b) \vdash^a D : b}{\Gamma \vdash^a \mathsf{let}_u\ x = \lambda y^{+}.D'\ \mathsf{in}\ D : b} \qquad (@^a)\quad \frac{x : \mathcal{N}(A^{+} \to b), y^{+} : A^{+} \in \Gamma}{\Gamma \vdash^a @(x, y^{+}) : b}$$

$$(|^a)\quad \frac{\Gamma \vdash M_i : b \quad i = 1, 2}{\Gamma \vdash (M_1 \mid M_2) : b}$$

Table 13: Concurrent administrative forms in CPS style: $\lambda_{\parallel}^{ak}$

$$
\begin{array}{ll}
\overline{\mathcal{N}(1)} & = \mathcal{N}(1) \\
\overline{b} & = b \\
\overline{\mathcal{N}(A_1 \to \cdots \to A_n \to \alpha)} & = \mathcal{N}(\overline{A_1} \to \cdots \to \overline{A_n} \to K(\alpha) \to b) \\
 & \text{where: } K(A) = \mathcal{N}(\overline{A} \to b), K(b) = \mathcal{N}(1)
\end{array}
$$

$$
\begin{array}{ll}
\psi(*) & = * \\
\psi(\lambda y^{+}.D) & = \lambda y^{+}.\lambda k.(D : k) \\
\mathsf{let}_u\ (x = V)^*\ \mathsf{in}\ D : k & = \mathsf{let}_u\ (x = \psi(V))^*\ \mathsf{in}\ D : k \\
(M \mid M') : k & = (M : k) \mid (M' : k) \\
@(x^*, @(M, M^{+}), N^*) : k & = \mathsf{let}_u\ k' = \lambda y.@(x^*, y, N^*) : k\ \mathsf{in}\ @(M, M^{+}) : k' \quad (u \neq 0) \\
@(x, x^{+}) : k & = @(x, x^{+}, k) \\
x : k & = @(k, x)
\end{array}
$$

Table 14: An optimized CPS translation for the concurrent case

Syntax

$$
\begin{array}{lll}
D & ::= \nu id\ D \parallel \nu id\ (id(id^+).D \mid D) \parallel \nu id\ (!id(id^+).D \mid D) \parallel M & \text{(declarations)} \\
M & ::= \overline{id}(id^+) \parallel (M \mid M) & \text{(terms)} \\
E & ::= [\,] \parallel E[\nu id([\,])] \parallel E[\nu id(\{!\}id(id^+.D \mid [\,]) \parallel E[[\,] \mid M] \parallel E[M \mid [\,]] & \text{(eval. contexts)} \\
A & ::= Ch(1) \parallel Ch(A^+) & \text{(types)}
\end{array}
$$

Structural Congruence

$$
\begin{array}{l|l}
(eq_1) \quad \nu x_1\ (\{\{!\}x_1(y_1^+).D_1 \mid\}\nu x_2\ (\{\{!\}x_2(y_2^+).D_2 \mid\}D)) & (eq_2) \quad \nu x\ (\{\{!\}x(y^+).D' \mid\}D) \\
\equiv \nu x_2\ (\{\{!\}x_2(y_2^+).D_2 \mid\}\nu x_1\ (\{\{!\}x_1(y_1^+).D_1 \mid\}D)) & \equiv D \\
\text{if } x_1 \notin FV(\lambda y_1^+.D_1), x_2 \notin FV(\lambda y_2^+.D_2) & \text{if } x \notin FV(D)
\end{array}
$$

Reduction Rules

$$
\begin{array}{rcl}
E[\nu x\ (!x(y^+).D \mid E'[\overline{x}z^+])] & \rightarrow & E[\nu x\ (!x(y^+).D \mid E'[[z^+/y^+]D])] \\
E[\nu x\ (x(y^+).D \mid E'[\overline{x}z^+])] & \rightarrow & E[\nu x\ (E'[[z^+/y^+]D])]
\end{array}
$$

Typing Rules

$$
(\nu^\pi) \quad \frac{\Gamma, x : A \vdash D}{\Gamma \vdash^\pi \nu x\ D} \qquad (\nu - in^\pi) \quad \frac{\Gamma, y^+ : A^+ \vdash^\pi D' \quad \Gamma, x : Ch(A^+) \vdash^\pi D}{\Gamma \vdash^\pi \nu x\ (\{!\}x(y^+).D' \mid D)}
$$

$$
(out^\pi) \quad \frac{x : Ch(A^+), y^+ : A^+ \in \Gamma}{\Gamma \vdash^\pi \overline{x}y^+} \qquad (|^\pi) \quad \frac{\Gamma \vdash^\pi M_i \quad i = 1, 2}{\Gamma \vdash^\pi (M_1 \mid M_2)}
$$

Table 15: A concurrent π-calculus: π

B Expressivity

This section illustrates the expressivity of the $\lambda_{\parallel}^{a}$-calculus (and therefore of the related $\lambda_{\parallel}^{ak}$ and π-calculi) as far as the programming of some familiar concepts in concurrent programming is concerned.

B.1 Output prefix

An 'output prefix' $@(x, y).D$ is simulated by the usual continuation passing trick:

$$
\begin{array}{lll}
\mathsf{let}_1 & k = \lambda w.D & \mathsf{in} \\
 & @(x, y, k) &
\end{array}
$$

B.2 Internal choice

We introduce an 'internal choice' operator $\oplus$ by defining $M \oplus N$ as follows (all variables being fresh):

$$
\begin{array}{lll}
\mathsf{let} & x = * & \mathsf{in} \\
\mathsf{let}_1 & y = \lambda k.@(@(k, y), x) & \mathsf{in} \\
\mathsf{let}_1 & k_1 = \lambda w.M & \mathsf{in} \\
\mathsf{let}_1 & k_2 = \lambda w.N & \mathsf{in} \\
 & (@(y, k_1) \mid @(y, k_2)) &
\end{array}
$$

B.3 External choice

An 'external choice' between M and N based on a boolean value (coded as a projection) can be defined as follows:

$$\begin{array}{lll} \mathsf{let} & x = * & \mathsf{in} \\ \mathsf{let}_1 & y = \lambda z.@(@(z, \lambda w.M, \lambda w.N), x) & \mathsf{in} \\ \cdots & & \end{array}$$

B.4 Multiple definitions

One can add to the language the possibility of having multiple definitions of the same name.

$$\begin{array}{lll} \mathsf{let} & x = V_1 & \mathsf{or} \\ & \cdots & \mathsf{or} \\ & x = V_n & \mathsf{in} \cdots \end{array}$$

where $V_1, \ldots, V_n$ do not depend on x. This does not compromise termination because a multiple definition can be simulated by a unique definition that receives its arguments and then performs an internal choice among the n branches.

B.5 Joined definitions

One can also add to the language the possibility of having joined definitions (in the direction of Fournet and Gonthier join-calculus).

$$\begin{array}{lll} \mathsf{let} & x_1 = V_1 & \mathsf{join} \\ & \cdots & \mathsf{join} \\ & x_n = V_n & \mathsf{in} \cdots \end{array}$$

where V_i can only depend on $x_1, \ldots, x_{i-1}$. The intended semantics is that the definitions of $x_1, \ldots, x_n$ can be used only simultaneously. Clearly, a joined definition can be simulated by a usual one and thus the termination property is not compromised.

B.6 Lock/Unlock

One can use the joined definitions to define a lock/unlock mechanism:

$$\begin{array}{lll} \mathsf{let} & x = * & \mathsf{in} \\ \mathsf{let} & \mathit{unlock} = \lambda w. \cdots & \mathsf{join} \\ \mathsf{let} & \mathit{lock} = \lambda k.@(k, \mathit{unlock}) & \mathsf{in} \\ & (@(\mathit{unlock}, x) \mid M[\mathit{lock}]) & \end{array}$$

Here $M[\mathit{lock}]$ is composed of several threads that may invoke the *lock* definition. When the lock is acquired the thread receives the name *unlock* and invoking it amounts to release the lock (this is a rudimentary mechanism and no effort is made to to enforce a correct usage).

B.7 CCS channel manager

Another possible use of the joined definitions is to define a CCS channel manager:

$$\begin{array}{lll} \mathsf{let} & x = * & \mathsf{in} \\ \mathsf{let} & in = \lambda k.@(k,x) & \mathsf{join} \\ \mathsf{let} & out = \lambda k.@(k,x) & \mathsf{in} \\ & M[in, out] & \end{array}$$

Here M is composed of parallel threads trying to synchronize on a channel.

C Proofs

The functional case being a special case of the concurrent one, we focus directly on the proofs of the latter. There is one exception: in the concurrent case in administrative, CPS form we fix the type of results to be the behavior type b rather than an arbitrary (value) type R but this does not affect the structure of the proofs.

C.1 Proof of proposition 4

The simple idea idea is to simulate the $\lambda_{\|}$-calculus in an ordinary simply typed λ-calculus equipped with a distinguished variable p of type $(b \to b) \to b$. More precisely, let λ_p be a simply typed λ-calculus with two basic types 1 and b, a constant $*$, and a distinguished variable p. It is well-known that such calculus terminates under an arbitrary reduction strategy. For our purposes, it suffices to consider a reduction strategy where a call-by-value redex $(\lambda x.M)V$ is reduced in a context that does not cross a λ.

Next let us define a translation $\langle \_ \rangle$ from $\lambda_{\|}$ to to λ_p which is the identity on types ($\langle \alpha \rangle = \alpha$) and type contexts and commutes with all the operators of the terms but on parallel composition where it is defined as follows:

$$\langle M \mid N \rangle = (p\langle M \rangle)\langle N \rangle \ .$$

The translation is also extended to evaluation contexts where in particular:

$$\langle E[[\ ] \mid M] \rangle = \langle E \rangle [(p[\ ])\langle M \rangle] \quad \langle E[M \mid [\ ]] \rangle = \langle E \rangle [(p\langle M \rangle)[\ ]]$$

Then it is easy to check the following properties:

1. If $\Gamma \vdash M : \alpha$ then $\langle \Gamma \rangle, p : b \to (b \to b) \vdash \langle M \rangle : \alpha$.
2. $\langle [V/x]M \rangle = [\langle V \rangle / x]\langle M \rangle$.
3. $\langle E[M] \rangle = \langle E \rangle [\langle M \rangle]$.

It follows that if $M \to N$ in $\lambda_{\|}$ then $\langle M \rangle \to \langle N \rangle$ in λ_p. Thus since λ_p terminates, $\lambda_{\|}$ must terminate too. □

C.2 Proof of theorem 5

(1) As a preliminary remark notice that if V is a value and D a declaration in $\lambda^a_{\parallel}$ then $FV(V^o) \subseteq FV(V)$ and $FV(D^o) \subseteq FV(D)$. Then we proceed by case analysis on the structural congruence by applying the properties of substitutions.

(2) By induction on the structure of M term of $\lambda_{\parallel}$.

(3) By induction on the typing of $\Gamma \vdash M : \alpha$ in $\lambda_{\parallel}$. For instance, suppose we derive $\Gamma \vdash \lambda x.M : A \to \alpha$ from $\Gamma, x : A \vdash M : \alpha$. Then by inductive hypothesis, $\underline{\Gamma}, x : \underline{A} \vdash^{am} \underline{M} : \underline{\alpha}$. Also, $\underline{\Gamma}, x : \underline{A \to \alpha} \vdash^{am} x : \underline{A \to \alpha}$, where by definition $\underline{A \to \alpha} = \mathcal{N}(\underline{A} \to \underline{\alpha})$. Then we can conclude $\underline{\Gamma} \vdash^{am} \mathsf{let}\ x = \lambda x.\underline{M}\ \mathsf{in}\ x : \underline{A \to \alpha}$ as required.

(4) First, we prove a substitution lemma for $\lambda_{\parallel}$, namely: if $\Gamma, x : A \vdash M : \alpha$ and $\Gamma \vdash V : A$ then $\Gamma \vdash [V/x]M : \alpha$. Then we proceed by induction on the typing of $\Gamma \vdash D : \alpha$ in $\lambda^a_{\parallel,\infty}$.

(5) Let F and E be one hole contexts composed of parallel compositions and applications, respectively:

$$F \quad ::= [\ ] \parallel (F \mid M) \parallel (M \mid F), \qquad E \quad ::= [\ ] \parallel @(id^*, E, M^*)\ .$$

If D_1 is well-typed and reduces then it has the shape:

$$\mathsf{let}\ (x = V)^*\ \mathsf{in}\ F[E[@(x, z^+)]]$$

and x is associated with a value $\lambda y^+.D'$. Then $D_1 \to D_2$ where:

$$D_2 \equiv \mathsf{let}\ (x = V)^*\ \mathsf{in}\ F[E[[z^+/y^+]D']]\ .$$

We extend the read-back translation to F and E by defining:

$$[\ ]^o = [\ ] \quad (F \mid M)^o = F^o \mid M^o \qquad (M \mid F)^o = M^o \mid F^o$$
$$@(x_1, \ldots, x_n, E, M_1, \ldots, M_m)^o = (\cdots(((x_1 \cdots x_n)E^o)M_1^o) \cdots M_m^o)\ .$$

Let σ be the (iterated) substitution $[V^o/x]^*$. We notice that:

$$M_1 \equiv D_1^o = \sigma(F^o[E^o[xz^+]]) = (\sigma F^o)[\sigma E^o[\sigma(xz^+)]]\ .$$

Recalling that $(\lambda y^+.D')^o = \lambda y^+.(D')^o$, we have that M_1 performs as many reductions as there are arguments z^+ and reduces to (assuming suitable renaming of bound variables):

$$M_2 \equiv (\sigma F^o)[(\sigma E^o)[[\sigma z^+/y^+](\sigma(D')^o)]\ .$$

On the other hand, we notice that:

$$D_2^o \equiv (\sigma F^o)[(\sigma E^o)[\sigma([z^+/y^+](D')^o)]]\ .$$

Knowing that the variables y^+ do not appear in the domain or codomain of the substitution σ, we apply the properties of substitution to check that:

$$[\sigma z^+/y^+](\sigma(D')^o) \equiv \sigma([z^+/y^+](D')^o)\ .$$

(6) Suppose D_1 is typable in the monadic fragment $\lambda^{am}_{\|}$. The following table describes how the structure of the readback $M_1 \equiv (D_1)^o$ determines D_1 up to structural congruence.

M_1	$D_1 \equiv$
x	x
$*$	$\mathsf{let}\ x = *\ \mathsf{in}\ x$
$\lambda y.M'$	$\mathsf{let}\ (x = V)^*\ \mathsf{in}\ \mathsf{let}\ z = \lambda y.D'\ \mathsf{in}\ z$ and $M' \equiv (\mathsf{let}\ (x = V)^*\ \mathsf{in}\ D')^o$
$M_1 M_2$	$\mathsf{let}\ (x = V)^*\ \mathsf{in}\ @(N_1, N_2)$ and $M_i \equiv (\mathsf{let}\ (x = V)^*\ \mathsf{in}\ N_i)^o, i = 1, 2$
$(M_1 \mid M_2)$	$\mathsf{let}\ (x = V)^*\ \mathsf{in}\ (N_1 \mid N_2)$ and $M_i \equiv (\mathsf{let}\ (x = V)^*\ \mathsf{in}\ N_i)^o, i = 1, 2$

Suppose $M_1 \equiv D_1^o$. Notice that M_1 is typable because the readback translation preserves typing (property 4). Then if M_1 reduces, it must have the shape $M_1 = F[E[(\lambda x.M')V]]$, where $F ::= [\ ] \mid (F \mid M) \mid (M \mid F)$ and $E ::= [\ ] \mid EM \mid VE$. The reduced term is $M_2 = F[E[[V/x]M']]$. Let σ be the (iterated) substitution $[V^o/x]^*$. By the table above, we derive that D_1 must have the shape $\mathsf{let}\ (x = V)^*\ \mathsf{in}\ F'[E'[@(x_1, x_2)]]$ with:

$$\begin{array}{llll} \sigma(F')^o & = F, & \sigma(E')^o & = E, \\ \sigma(x_1) & = \lambda x.M', & \sigma(x_2) & = V\ . \end{array}$$

In particular, we see that the variable x_1 must occur in the list of let declarations and it must be associated with a λ-abstraction, say $\lambda x.D'$ where:

$$(\mathsf{let}\ (x = V)^*\ \mathsf{in}\ D')^o \equiv \sigma(D')^o \equiv M'\ .$$

This means that D_1 can also perform one reduction and reduce to

$$D_2 \equiv \mathsf{let}\ (x = V)^*\ \mathsf{in}\ F'[E'[[x_2/x]D']]\ .$$

We observe that:

$$D_2^o \equiv (\sigma(F')^o)[\sigma(E')^o[\sigma([x_2/x](D')^o)]]$$

Knowing that the variable x does not appear in the domain or codomain of the substitution σ, we apply the properties of substitution to check that:

$$\sigma([x_2/x](D')^o) \equiv [\sigma(x_2)/x](\sigma(D')^o) \equiv [V/x]M'\ .$$

□

C.3 Proof of theorem 7

(1) By induction on the proof of $\Gamma \vdash^a D : \alpha$ and case analysis on the definition of $D : k$. We spell out the following case:

$$\frac{\Gamma \vdash^a @(M, M^+) : \mathcal{N}(A^+ \to \alpha) \quad \Gamma \vdash^a N^+ : A^+}{\Gamma \vdash^a @(@(M, M^+), N^+) : \alpha}$$

We use the following abbreviations:

$$A' \equiv \mathcal{N}(A^+ \to \alpha), \qquad M' \equiv @(M, M^+), \qquad \Gamma' \equiv \overline{\Gamma}, k : K(\alpha)\ .$$

We have to show:

$$\Gamma' \vdash^a \mathsf{let}\ k' = \lambda y.@(y, N^+) : k\ \mathsf{in}\ (M' : k') : b$$

By weakening, we derive $\Gamma, y : A' \vdash^a @(y, N^+) : \alpha$. Then by induction hypothesis we have:

$$\Gamma', y : \overline{A'} \vdash^a (@(y, N^+) : k) : b \ .$$

Also, by induction hypothesis on $\Gamma \vdash^a M' : A'$ we derive:

$$\overline{\Gamma}, k' : K(A') \vdash^a (M' : k') : b \ .$$

Noticing that $K(A') = \mathcal{N}(\overline{A'} \to b)$ we conclude by applying the rule for the let.

(2) As a preliminary remark, we notice that structural congruence is preserved by the CPS translation, namely $D \equiv D'$ in λ^a entails $(D : k) \equiv (D' : k)$. We introduce some additional notation for evaluation contexts. We denote with F a one hole context composed of parallel compositions:

$$F ::= [\ ] \parallel F \mid M \parallel M \mid F \ .$$

We denote with H an elementary applicative context of the shape $@(id^*, [\ ], M^*)$. If the declaration D in $\lambda^a_{\parallel}$ is typable and reduces then it must have the following shape:

$$\mathsf{let}_u \ (x = V)^* \ \mathsf{in} \ F[H_1[\cdots H_m[@(x, z^+)] \cdots]]$$

where the name x is associated with a value $\lambda y^+.D'$. Then the reduced term is:

$$\mathsf{let}_u \ (x = V)^* \ \mathsf{in} \ F[H_1[\cdots H_m[[z^+/y^+]D')] \cdots]]$$

The CPS translation of the declaration D has the following shape:

$$\mathsf{let}_u \ (x = \psi(V))^* \ \mathsf{in} \ (F[H_1[\cdots H_m[@(x, z^+)] \cdots]] : k)$$

Recalling that the CPS translation distributes over parallel composition, we define:

$$[\ ] : k = [\ ], \quad (F \mid M) : k = (F : k) \mid (M : k), \quad (M \mid F) : k = (M : k) \mid (F : k) \ .$$

Then we have:

$$\begin{array}{rl} F[H_1[\cdots H_m[@(x, z^+)] \cdots]] : k & = (F : k)[\ \ \mathsf{let}_1\ k_1 = \lambda y.(H_1[y] : k)\ \mathsf{in} \\ & \qquad \mathsf{let}_1\ k_2 = \lambda y.(H_2[y] : k_1)\ \mathsf{in} \\ & \qquad \cdots \\ & \qquad \mathsf{let}_1\ k_m = \lambda y.(H_m[y] : k_{m-1})\ \mathsf{in} \\ & \qquad @(x, z^+, k_m)\ ] \end{array}$$

Recalling that $\psi(\lambda y^+.D') = \lambda y^+, k.(D' : k)$, we observe that the CPS translation is ready to perform the corresponding reduction. Then we proceed by case analysis on the shape of D' to show that the CPS translation reduces to:

$$\mathsf{let}_u \ (x = \psi(V))^* \ \mathsf{in} \ (F[H_1[\cdots H_m[[z^+/y^+]D'] \cdots]] : k)$$

1. $D' \equiv \mathsf{let}_{u'} \ (x' = V')^* \ \mathsf{in} \ x'$. Then the CPS translation performs a first reduction by replacing $@(x, z^+, k_m)$ with

   $$[z^+/y^+, k_m/k](\mathsf{let}_{u'} \ (x' = \psi(V'))^* \ \mathsf{in} \ @(k, x')) \ .$$

   If there is no enclosing elementary evaluation context, *i.e.* $m = 0$, $k_m = k$, we are done. Otherwise, the CPS translation performs an additional reduction along the continuation k_m. Following this reduction the let definition of k_m can be removed applying the second rule of structural congruence.

2. $D' \equiv \mathsf{let}_{u'}\ (x' = V')^*\ \mathsf{in}\ @(M', M'^+)$. Then the CPS translation performs a reduction replacing $@(x, z^+, k_m)$ with

$$[z^+/y^+, k_m/k](\mathsf{let}_{u'}\ (x' = \psi(V'))^*\ \mathsf{in}\ (@(M', M'^+) : k))\ .$$

3. $D' \equiv \mathsf{let}_{u'}\ (x' = V')^*\ \mathsf{in}\ (M_1 \mid M_2)$. In this case, the typing requires that there are no enclosing elementary evaluations contexts, *i.e.* $m = 0$, $k_m = k$. The CPS translation performs a reduction replacing $@(x, z^+, k_m)$ with

$$[z^+/y^+, k_m/k](\mathsf{let}_{u'}\ (x' = \psi(V'))^*\ \mathsf{in}\ (M_1 : k) \mid (M_2 : k))\ .$$

Acknowledgment

The author acknowledges the financial support of the Future and Emerging Technologies (ET) program within the Seventh Framework Programme for Research of the European Commission, under FET-Open grant number: 243881 (project CerCo).