Information and Computation **150**, 153–186 (1999)
Article ID inco.1998.2773, available online at http://www.idealibrary.com on IDEAL®

Discrimination by Parallel Observers: The Algorithm

Mariangiola Dezani-Ciancaglini*

Department of Informatics, University of Torino, c.Svizzera 185, 10149 Turino, Italy
E-mail: dezani@di.unito.it

and

Jerzy Tiuryn† and Paweł Urzyczyn†, ‡

Institute of Informatics, University of Warsaw, Banacha 2, 02-097 Warsaw, Poland
E-mail: tiuryn@mimuw.edu.pl; urzy@mimuw.edu.pl

The main result of the paper is a constructive proof of the following equivalence: two pure λ-terms are observationally equivalent in the lazy concurrent λ-calculus *iff* they have the same Lévy–Longo trees. An algorithm which allows to build a context discriminating any two pure λ-terms with different Lévy–Longo trees is described. It follows that contextual equivalence coincides with behavioural equivalence (bisimulation) as considered by Sangiorgi. Another consequence is that the discriminating power of concurrent lambda contexts is the same as that of Boudol–Laneve's contexts with multiplicities. © 1999 Academic Press

1. INTRODUCTION

The aim of this paper is to improve our understanding of what is the "meaning" of a term in the *lazy λ-calculus*. To explain our result let us begin with the following few observations borrowed from the paper [2] of Abramsky and Ong.

In the ordinary λ-calculus, the most natural understanding of evaluation to a "value" is reduction to a normal form. It is however, well known that this interpretation cannot be taken as a basis of a consistent equational theory. Indeed, equating all λ-terms without normal forms results in equating everything: the equality $\lambda x.xN\mathbf{\Omega} = \lambda x.xM\mathbf{\Omega}$ must hold for each M, N (see [7, p. 39]), where $\mathbf{\Omega} \equiv (\lambda x.xx)(\lambda x.xx)$.

* Partly supported by ESPRIT BRA7232 GENTZEN and by NATO Grant HTECH.LG960875.

† Partly supported by NSF Grant CCR-9417382, KBN Grant 8 T11C 034 10, ESPRIT BRA7232 GENTZEN, and by NATO Grant HTECH.LG960875.

‡ Corresponding author.

One can consider head normal forms instead, obtaining a consistent theory, which equates unsolvable terms. The "meaning" of a λ-term is then understood as its equivalence class of the observational equivalence $\sim$ defined so that $M \sim N$ holds iff

$$C[M]\downarrow \Leftrightarrow C[N]\downarrow \qquad \text{for all contexts } C[\ ],$$

where the notation "$P\downarrow$" reads "P has a value."

This meaning of a λ-term can equivalently be expressed by the notion of Böhm trees, that is we have

$$M \sim N \qquad \text{iff} \quad \mathrm{BT}(M) \cong \mathrm{BT}(N),$$

where $\cong$ stands for equality of Böhm trees modulo possibly infinite eta-reductions.

The reader is referred to [2] for a discussion why this understanding of a "meaning" is not satisfactory when λ-calculus is meant to be used as a prototypical functional language. In short, a lazy language may evaluate an expressions to a "prompt" (an abstraction) rather than to a head normal form. This is better represented by assuming that any abstraction is a "value." This idea gives rise to the lazy λ-calculus. The lazy λ-calculus is a type free calculus, with the same syntax as pure λ-calculus and a reduction relation over closed terms, with just two rules:

$$(\lambda x.M)N \to M[N/x], \qquad \frac{M \to M'}{MN \to M'N}.$$

Clearly, the structure of lazy λ-calculus is finer than that of the "standard theory." Terms like $\mathbf{\Omega}$ and $\lambda x.\mathbf{\Omega}$ are now distinguished, and the contextual equivalence $\sim$ has smaller equivalence classes. Thus, obviously, Böhm trees are no longer adequate to describe the "meaning" of λ-terms.

Here comes the idea of a refinement of the Böhm tree approach for the lazy evaluation: *the Lévy–Longo trees*. Indeed Longo [24] and Ong [27, 28] prove that Lévy–Longo trees characterize the λ-theory of the *lazy Plotkin–Scott–Engeler models*. These trees were introduced by Longo in [24] following ideas of Lévy [23].

The Lévy–Longo trees are defined in a similar way to Böhm trees with the following modifications. First is that one distinguishes between $\bot$ and $\lambda x_1 ... x_n . \bot$, for all n. In addition, there is a new symbol $\top$ used to represent an "infinite lambda abstraction" obtained when a term can be reduced to weak head normal forms $\lambda x_1 ... x_n . M_n$, for all n and suitable M_n. (An example of such a term is $(\lambda xy.xx)(\lambda xy.xx)$.) The Lévy–Longo tree of a term M is denoted by $LL(M)$.

It should be clear that equality of Lévy–Longo trees suffices for contextual equivalence. However, the converse does not hold even for Böhm trees. Take as an example the two terms $M \equiv \lambda x.xx$ and $N \equiv \lambda x.x(\lambda y.xy)$. We have $M \sim N$, but $\mathrm{BT}(M) \neq \mathrm{BT}(N)$.

We shall however contend that Lévy–Longo trees provide an appropriate understanding of the actual "meaning" of a λ-term with respect to lazy evaluation, even if, as shown by the above example, they are of stronger discriminating power than the ordinary notion of contextual equivalence. This is because the notion of lazy contextual equivalence may change when the lazy λ-calculus is embedded into a richer language. Then the finer structure of lazy λ-terms becomes visible in the extended contexts.

As noticed in [33], this is interesting mainly when we add parallel and nondeterministic features to the lazy λ-calculus, in view of the integration between functional and concurrent programming languages. We want to know when a functional procedure can replace another one, leaving unchanged the observable behaviour of a process which uses it.

Notice that an equivalence finer than that induced by Lévy–Longo trees seems unreasonable. In fact [33] proves that enriching the lazy λ-calculus with *well-formed operators* (w.r.t. a natural definition inspired by [22]), we never distinguish λ-terms with the same Lévy–Longo tree.

In the literature we can find essentially two different scenarios which induce the same equivalence relation on pure λ-terms as Lévy–Longo trees.

In [33], Sangiorgi considers the embedding of lazy λ-calculus in some concurrent calculi. First, Milner's encoding of lazy λ-calculus in π-*calculus* is studied. A slight variant of this encoding gives rise to a λ-calculus model whose theory is again that of Lévy–Longo trees [34]. Then the lazy λ-calculus is enriched with a simple nondeterministic operator, which, when applied to an argument, either gives the argument itself or diverges. In both cases the processes are compared using bisimulation. So we can conclude that "*nondeterminism + bisimulation*" has the discriminating power of Lévy–Longo trees.

On the other side, Boudol and Laneve [12] introduce a "resource conscious" refinement of λ-calculus, in which every argument comes with a *multiplicity*. The reduction process (which uses *explicit substitution* in an essential way) remains deterministic, but a *deadlock* can appear. The terms are compared by means of the standard observational equivalence. The result is that also "*deadlock + observational equivalence*" discriminate as Lévy–Longo trees. Notably Boudol and Laneve use in [11] the λ-calculus of multiplicities to show that the preorder on terms induced by the encoding in π-calculus when processes are compared using *may testing* coincide with the preorder on terms induced by their Lévy–Longo tree representation.

We consider the behaviour of pure λ-terms inside contexts of the concurrent λ-calculus as defined in [18]. This calculus is obtained from the pure λ-calculus (with call-by-value and call-by-name variables) by adding a non-deterministic choice operator $+$ and a parallel operator $\|$, whose main reduction rules are

$$M + N \to M, \qquad M + N \to N$$

$$\frac{M \to M' \quad N \to N'}{M \parallel N \to M' \parallel N'}$$

$$\frac{M \to M' \quad N \not\to}{M \parallel N \to M' \parallel N, \quad N \parallel M \to N \parallel M'},$$

where $\to$ stands for one-step reduction and $\not\to$ means irreducibility. The reader can find more on this calculus in Section 4 of this paper.

The main technical tool we use is a type assignment system for union and intersection types (see Section 3 for details). Let $\vdash$ denote the derivation in this system. Besides the logic $Type_{\vee\wedge}$ of intersection and union types, we will consider also the logic $Type_{\wedge}$ of intersection types only, and the set of type schemes $Type_{\wedge t}$ obtained by adding type variables to $Type_{\wedge}$. We stress that we are interested in results concerning $Type_{\vee\wedge}$ and that the introduction of the other two languages is only a device for showing these results.

The following theorem is proved in [18].

THEOREM 1 [18]. *For pure λ-terms M, N, the following conditions are equivalent*:

(i) *For every basis Γ and type σ,*

$$\Gamma \vdash M : \sigma \qquad \textit{iff} \quad \Gamma \vdash N : \sigma;$$

(ii) *M and N are observationally equivalent in the concurrent λ-calculus.*

The main technical result of our paper is as follows:

THEOREM 2. *For pure λ-terms M, N, the following conditions are equivalent*:

(i) *For every basis Γ and type σ,*

$$\Gamma \vdash M : \sigma \qquad \textit{iff} \quad \Gamma \vdash N : \sigma;$$

(ii) *M and N have the same Lévy–Longo trees.*

This result nicely complements the characterization obtained in [18]. Theorems 1 and 2 together give the following fact

COROLLARY 3. *Two pure λ-terms are observationally equivalent in the concurrent λ-calculus iff they have the same Lévy–Longo trees.*

Really, Theorems 1 and 2 hold for broader sets of terms. In fact Theorem 1 was stated in [18] for the terms of the concurrent λ-calculus $\Lambda_{+\|}$ (introduced in Section 4) and we will prove Theorem 2 for the set of terms $\Lambda_{\bot,\top}$, obtained by adding the constants $\bot, \top$ to the pure λ-terms (cf. Section 3). The intersection between $\Lambda_{+\|}$ and $\Lambda_{\bot,\top}$ is the set of pure λ-terms, so we get Corollary 3 for them. Notice that the observation contexts which give us the discriminating power of Lévy–Longo trees belong to the concurrent λ-calculus.

Corollary 3 justifies the title of the paper: we can exhibit a discriminating context belonging to the concurrent λ-calculus whenever we get two pure λ-terms with different Lévy–Longo trees.

Hence the third scenario of the same discriminating power as the Lévy–Longo trees is "*concurrent λ-calculus + observational equivalence*". We want to justify the interest in this third scenario by comparing it with the previous ones.

From one side, observational equivalence seems a more appropriate tool than bisimulation. First, the notion of observational equivalence is simpler: indeed bisimulation amounts to verifying the behaviours of processes at intermediate steps of computation rather than just the input–output relation. Then, as pointed out in [9], there are various notions of bisimulation (see, for example [25, 26, 32]) and there is no consensus on which is the proper one. Notably Sangiorgi proves in [34] that these notions of bisimulation coincide on the sublanguage of π-calculus which suffices for encoding lazy λ-calculus.

On the other side, the λ-calculus of multiplicities of [12] does not seem to be of independent interest, while systems similar to the concurrent λ-calculus have been studied in different papers, like [10, 29, 30, 3, 17, 18].

A last remark is that we give a very simple algorithm for building discriminating contexts. Also the proofs of [32, 12] are constructive, so they can be used for obtaining discriminating contexts, through suitable variants of the Böhm-out technique [7, Section 10.3].

As a by-product of our main result we have that the λ-theory of the logical model of [18] coincides with the equality of Lévy–Longo trees, proving a conjecture of [18]. The domain of this logical model is the initial solution of the domain equation $D = \mathscr{P}^{\#}([D \to D]_{\perp})$ in the category of prime algebraic lattices, where $\mathscr{P}^{\#}$ is the upper powerdomain functor, $[\cdot \to \cdot]$ is the space of continuous functions and $(\cdot)_{\perp}$ is the lifting operator. The lazy Plotkin–Scott–Engeler models [31] of lazy λ-calculus are solutions of the domain equations $D = [D \to D]_{\perp} \times \mathscr{P}(A)$, where $\mathscr{P}$ is the powerset constructor and A is an enumerable set, in the same category. In spite of this difference, Lévy–Longo trees characterize the λ-theory of all these models, when we restrict to pure λ-terms. Notice that in this case the powerdomain semantics uses only singleton sets.

We want to stress that we describe a discrimination algorithm. Given two λ-terms with different Lévy–Longo trees, we can determine a concurrent context which separates these terms. A proof without explicit construction of a concurrent context was given in [20].

This paper is organized as follows. In Section 2 we define the intersection and union types $Type_{\vee\wedge}$ and an auxiliary language $Type_{\wedge}$ of intersection types. Section 3 gives the language $\Lambda_{\perp,\top}$, and introduces the type assignment system $\vdash$. Section 4 contains an introduction to the concurrent λ-calculus. In Section 5 we define Lévy–Longo trees and approximants. The latters serve as the main tool in proving the main result of this paper. Section 6 introduces the notion of principal pairs and discusses their properties. The main result of the paper (Theorem 20) and the discrimination algorithm are given in Section 7. We also illustrate there the technique of finding discriminating typings and concurrent contexts for two λ-terms with different Lévy–Longo trees. The concluding remarks section discusses the relation with Abramsky–Ong development [2]. The Appendix contains proofs of the auxiliary technical results which are used during the proof of the main theorem of the paper.

2. SUBTYPE PREORDERS

In this section we define the set of types $Type_{\vee\wedge}$ used in our type assignment system, and a subset $Type_{\wedge}$ of it, and we state some basic properties of the subtype relations we need. We remark that the logic $Type_{\vee\wedge}$ is central to our development, while the logic $Type_{\wedge}$ is only useful in some proofs given in the Appendix. In spite of this, we present here both logics since we think the conservativity result (Proposition 4) is of independent interest.

The set $Type_{\vee\wedge}$ is defined by the grammar

$$\sigma ::= \omega \mid \sigma \to \sigma \mid \sigma \wedge \sigma \mid \sigma \vee \sigma,$$

and the set $Type_{\wedge}$ is defined by

$$\sigma ::= \omega \mid \sigma \to \sigma \mid \sigma \wedge \sigma.$$

Clearly $Type_{\wedge} \subseteq Type_{\vee\wedge}$ and the inclusion is proper. A notational convention is that $\wedge$ and $\vee$ take precedence over $\to$.

Now we define our subtyping relations. The relation $\leqslant$ over $Type_{\vee\wedge}$ is the smallest preorder such that

1. $\langle Type_{\vee\wedge}, \leqslant \rangle$ is a distributive lattice in which $\wedge$ is the meet, $\vee$ is the join and ω is the top. (Strictly speaking it becomes a lattice after taking the quotient by the equivalence relation induced by the preorder $\leqslant$.) That is, the following axioms and rules axiomatize $\wedge$, $\vee$, and ω:

(A1) $\sigma \leqslant \sigma$;

(A2) $\sigma \leqslant \omega$;

(A3) $\sigma \wedge \tau \leqslant \sigma$;

(A4) $\sigma \wedge \tau \leqslant \tau$;

(A5) $\sigma \leqslant \sigma \vee \tau$;

(A6) $\tau \leqslant \sigma \vee \tau$;

(A7) $\sigma \wedge (\tau \vee \rho) \leqslant (\sigma \wedge \tau) \vee (\sigma \wedge \rho)$;

(R1) $\sigma \leqslant \rho,\ \rho \leqslant \tau \Rightarrow \sigma \leqslant \tau$;

(R2) $\rho \leqslant \sigma,\ \rho \leqslant \tau \Rightarrow \rho \leqslant \sigma \wedge \tau$;

(R3) $\sigma \leqslant \rho,\ \tau \leqslant \rho \Rightarrow \sigma \vee \tau \leqslant \rho$;

2. the arrow satisfies

(A8) $\sigma \to \omega \leqslant \omega \to \omega$;

(A9) $(\sigma \to \rho) \wedge (\sigma \to \tau) \leqslant \sigma \to \rho \wedge \tau$;

(R4) $\sigma \geqslant \sigma',\ \tau \leqslant \tau' \Rightarrow \sigma \to \tau \leqslant \sigma' \to \tau'$.

The relation $\leqslant$ over $Type_{\wedge}$ is the smallest preorder such that

1. $\langle Type_{\wedge}, \leqslant \rangle$ is a semi-lattice, in which $\wedge$ is the meet, and ω is the top, i.e. the axioms (A1) through (A4) and rules (R1) and (R2) above axiomatize $\wedge$ and ω.

2. the arrow satisfies (A8), (A9), and (R4) above.

We will use the symbols $\leqslant_{\wedge}$ and $\leqslant_{\vee\wedge}$, for subtyping in $Type_{\wedge}$ and $Type_{\vee\wedge}$, respectively. The symbol $\sim_{\wedge}$ denotes the equivalence relation induced by $\leqslant_{\wedge}$.

Similarly, $\sim_{\vee\wedge}$ denotes the equivalence relation induced by $\leqslant_{\vee\wedge}$. In fact, as it follows from the next result, we do not have to be careful when using subscripts, since the preorder over $Type_{\vee\wedge}$ is conservative over $Type_{\wedge}$. We will sometimes identify $\sim$-equivalent types, especially we identify types σ and $\sigma \wedge \omega$.

Proposition 4 (Conservativity). *For all $\sigma, \tau \in Type_{\wedge}$,*

$$\sigma \leqslant_{\vee\wedge} \tau \qquad \textit{iff} \quad \sigma \leqslant_{\wedge} \tau.$$

We defer the proof of Proposition 4 for the Appendix. In the remainder of this section we collect some useful properties of the subtyping relations.

Lemma 5. *For types in $Type_{\vee\wedge}$ we have:*

1. *If $\bigwedge_{i\in I}(\sigma_i \to \tau_i) \leqslant_{\vee\wedge} \rho \vee \mu$, then either $\bigwedge_{i\in I}(\sigma_i \to \tau_i) \leqslant_{\vee\wedge} \rho$ or $\bigwedge_{i\in I}(\sigma_i \to \tau_i) \leqslant_{\vee\wedge} \mu$.*
2. *If $\wedge$ does not occur in σ and in τ_i for all $i \in I$ and $\bigwedge_{i\in I}\tau_i \leqslant_{\vee\wedge} \sigma$, then there is $i \in I$ such that $\tau_i \leqslant_{\vee\wedge} \sigma$.*

For types in $Type_{\vee\wedge}$ or $Type_{\wedge}$ we have:

3. *If $\bigwedge_{i\in I}(\mu_i \to \nu_i) \leqslant \sigma \to \tau$, where $\tau \not\sim \omega$, then for some $J \subseteq I$ it holds that $\sigma \leqslant \bigwedge_{j\in J}\mu_j$ and $\bigwedge_{j\in J}\nu_j \leqslant \tau$.*
4. *If $I_1 \subseteq I$ then $\bigwedge_{i\in I}(\sigma_i \to \sigma'_i) \leqslant (\bigwedge_{i\in I_1}\sigma_i) \to (\bigwedge_{i\in I_1}\sigma'_i)$.*

Proof. Part (1) follows from the irreducibility of arrow types (as shown in Section 3.1 of [18]). Part (2) follows from Proposition 3.8(iii) in [18]. Part (3) is Lemma 3.9(ii) in [18]. For part (4), let us observe that $\bigwedge_{i\in I}(\sigma_i \to \sigma'_i) \leqslant \bigwedge_{i\in I_1}(\sigma_i \to \sigma'_i) \leqslant \bigwedge_{i\in I_1}((\bigwedge_{k\in I_1}\sigma_k) \to \sigma'_i) \leqslant (\bigwedge_{i\in I_1}\sigma_i) \to (\bigwedge_{i\in I_1}\sigma'_i)$. The last inequality follows from the subtyping axiom (A9). ▮

3. LAMBDA-TERMS AND TYPE ASSIGNMENT

The set Λ of terms of the untyped λ-calculus (λ-terms) is defined as usual according to the following grammar

$$M := x \mid MM \mid \lambda x.M,$$

where x ranges over an infinite set of term variables. We refer to [7, Definition 2.1.6] for the standard notions of free and bound variables. We use $FV(M)$ to denote the set of free variables in M (according to Definition 2.1.7 of [7]).

We will abbreviate some λ-terms as

$$\begin{array}{ll}
\mathbf{I} =^{Def} \lambda x.x, & \mathbf{\Delta} =^{Def} \lambda x.xx \\
\mathbf{K} =^{Def} \lambda xy.x, & \mathbf{O} =^{Def} \lambda xy.y \\
\mathbf{\Omega} =^{Def} \mathbf{\Delta\Delta}, & \mathbf{Y} =^{Def} \lambda y.(\lambda x.y(xx))(\lambda x.y(xx)).
\end{array}$$

We consider the language $\Lambda_{\bot,\top}$ obtained from Λ by adding two new constants: $\bot$ and $\top$. Therefore the set $\Lambda_{\bot,\top}$ is defined by

$$M := x \mid \bot \mid \top \mid MM \mid \lambda x.M.$$

Usually in the literature only $\bot$ is added to Λ, both in the case of classical λ-calculus [7, p. 366] and of lazy λ-calculus [23]. We need the additional constant $\top$ to account for an "infinite abstraction." It will become clearer after we introduce Lévy–Longo trees in Section 5.

In the lazy λ-calculus $\bot$ plays the role of an unsolvable term which never reduces to an abstraction, and therefore it could be replaced by $\mathbf{\Omega}$. Analogously, $\top$ plays the role of an unsolvable term which reduces to terms with arbitrary numbers of initial abstractions, and therefore it could be replaced by **YK**. We think that our choice (which is the standard one) is better since it allows us to approximate λ-terms by constants rather than by λ-terms (cf. Section 5).

The symbol $\equiv$ denotes the syntactic equality of terms up to the renaming of bound variables.

Head normal forms are terms of the form $\lambda x_1 \cdots x_n.yM_1 \cdots M_m$, where $n, m \geqslant 0$. The variable y is called the *head variable* of such a head normal form.

On $\Lambda_{\bot,\top}$ we consider the reduction rules

$$\bot M \to \bot$$

$$\top M \to \top$$

$$\lambda x.\top \to \top$$

for all x and all M. So by *reduction* we will mean the contextual, reflexive and transitive closure of these rules, plus the standard beta rule. By $=_{\beta}^{\bot,\top}$ we will denote the symmetric closure of this reduction relation.

Now we present the definitions and basic properties of our type assignment system. A basis Γ is a partial mapping from term variables to types. We derive assertions $\Gamma \vdash M\colon \tau$, where $M \in \Lambda_{\bot,\top}$, and $\tau \in Type_{\vee\wedge}$, and all types in Γ are in $Type_{\vee\wedge}$. In the next section we will extend this system to terms of the concurrent λ-calculus. For that extension the subject reduction property (Theorem 8) is no longer true; this is the reason why we first consider the present system.

The axioms and rules of our system are the following:

$$(\mathrm{Ax}) \quad \Gamma \vdash x\colon \Gamma(x) \qquad (\omega) \quad \Gamma \vdash M\colon \omega \qquad (\top) \quad \Gamma \vdash \top\colon \sigma$$

$$(\to\mathrm{I}) \quad \frac{\Gamma, x\colon \sigma \vdash M\colon \tau}{\Gamma \vdash \lambda x.M\colon \sigma \to \tau}, \qquad (\to\mathrm{E}) \quad \frac{\Gamma \vdash M\colon \sigma \to \tau \quad \Gamma \vdash N\colon \sigma}{\Gamma \vdash MN\colon \tau}$$

$$(\wedge\mathrm{I}) \quad \frac{\Gamma \vdash M\colon \sigma \quad \Gamma \vdash M\colon \tau}{\Gamma \vdash M\colon \sigma \wedge \tau}, \qquad (\leqslant) \quad \frac{\Gamma \vdash M\colon \sigma \quad \sigma \leqslant \tau}{\Gamma \vdash M\colon \tau}.$$

We write $\Gamma, x\colon \sigma$ for the mapping $\Gamma'(y) = \sigma$ if $y \equiv x$ and $\Gamma'(y) = \Gamma(y)$, otherwise. The notation "$\Gamma \vdash M\colon \tau$" means "$\Gamma \vdash M\colon \tau$ is derivable."

None of the logical rules use disjunction explicitly. But types containing disjunction can be derived with the help of rule ($\leqslant$).

Note that, in general, the condition $\Gamma \vdash M: \tau$ does not imply that all free variables of M are in the domain of Γ. This is because of rule (ω). Thus, for uniformity, we assume that $\Gamma(x) = \omega$, whenever $x \notin \mathrm{Dom}(\Gamma)$.

In the following we use the notation

$$\Gamma \uplus \Gamma' = \{x: \sigma \wedge \tau \mid \Gamma(x) = \sigma \text{ and } \Gamma'(x) = \tau\}.$$

Accordingly we define:

$$\Gamma \Subset \Gamma' \quad \Leftrightarrow \quad \exists \Gamma''.\Gamma \uplus \Gamma'' = \Gamma'.$$

The type assignment rules can be "reversed," as stated in the following theorem.

Theorem 6. (Generation Theorem).

1. *If* $\Gamma \vdash x: \tau$ *then* $\Gamma(x) \leqslant \tau$.
2. *If* $\Gamma \vdash \lambda x.M: \tau$ *then there are types* $\mu_1, ..., \mu_n$, $\nu_1, ..., \nu_n$, *such that* $\bigwedge_{i=1}^{n} (\mu_i \to \nu_i) \leqslant \tau$ *and* $\Gamma, x: \mu_i \vdash M: \nu_i$, *for all* $i \leqslant n$.
3. *If* $\Gamma \vdash \lambda x.M: \sigma \to \tau$ *then* $\Gamma, x: \sigma \vdash M: \tau$, *in addition if* $\sigma \sim \omega$ *then* $\Gamma \vdash M: \tau$.
4. *If* $\Gamma \vdash MN: \tau$ *with* $\tau \not\sim \omega$, *then* $\Gamma \vdash M: \sigma \to \tau$ *and* $\Gamma \vdash N: \sigma$, *for some* σ.

Proof. The proof of (1) and (2) is by a routine induction with respect to the size of derivations. Part (3) follows from part (2) and Lemma 5(3). Part (4) is proved again by induction, with Lemma 5(4) used for case ($\wedge$I).

Following [18], we will use the notation $\omega^n \to \sigma$, defined by

$$\omega^0 \to \sigma = \sigma;$$
$$\omega^{n+1} \to \sigma = \omega \to (\omega^n \to \sigma).$$

We will also use the abbreviation $\omega^n \to \omega = \omega_n$.

The types ω_n for suitable n are "better than" all other types, as shown in the following proposition (proved in [18]).

Proposition 7. *For all* σ, *there exists* n *such that* $\omega_n \leqslant \sigma$.

An essential property of the system $\vdash$ is the *subject conversion property*; the types are invariant w.r.t. $=_\beta^{\perp, \top}$ conversion. This property is fairly well known for ordinary systems with intersection types w.r.t. beta conversion (see, e.g. [13]) and can be shown for our system using the same approach. Let us stress here that our system $\vdash$ should not be confused with systems of union types with a "union elimination" rule, (cf., e.g. [5]) which often do not have even the subject reduction property. As shown in [18], union elimination is unsound for the concurrent λ-calculus, since the terms of this calculus contain the nondeterministic choice operator (defined in Section 4). For example, in absence of the union elimination rule we cannot derive $\vdash \lambda xy.xy: (\sigma \to \tau) \vee (\rho \to \tau) \to (\sigma \wedge \rho) \to \tau$.

Theorem 8 (Subject conversion). *If* $\Gamma \vdash M: \tau$ *and* $M =_\beta^{\perp, \top} M'$, *then* $\Gamma \vdash M': \tau$.

Proof. For beta equality the proof is a straightforward adaptation of the proof given in [13]. For the conversion rules involving $\top$, notice that $\Gamma \vdash \lambda x.\top : \tau$ holds for all Γ, τ and x. In fact for a fixed τ by Proposition 7 we can always find n such that $\omega_n \leqslant \tau$. Now by axiom $(\top)$ we can deduce $\Gamma \vdash \top : \omega_n$, so by rules $(\rightarrow \mathrm{I})$ and $(\leqslant)$ we are done, since $\omega_{n+1} \leqslant \omega_n$. Moreover, $\Gamma \vdash \top M : \tau$ for all Γ, τ, and M follows from $\Gamma \vdash \top : \omega \rightarrow \tau$ and $\Gamma \vdash M : \omega$ using rule $(\rightarrow \mathrm{E})$.

About $\perp$, notice that only types equivalent to ω can be derived for $\perp$ and $\perp M$. ▮

Observe that types are not preserved by η-reduction. Indeed, we have, e.g. $\vdash \lambda y.xy : \omega \rightarrow \omega$, but $\not\vdash x : \omega \rightarrow \omega$.

4. THE CONCURRENT λ-CALCULUS

We extend the syntax of pure λ-calculus with a nondeterministic choice operator $+$ and a parallel operator $\|$. We allow two sorts of variables, namely the set Vn of call-by-name variables, ranged over by x, y, z and the set Vv of call-by-value variables, ranged over by v, w. The terms of the concurrent λ-calculus are defined by the grammar

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

We denote by $\Lambda_{+\|}$ this set of terms. In writing terms, we assume that abstraction and application take precedence over the symbols $+$ and $\|$. For any $M \in \Lambda_{+\|}$, the symbol $FV(M)$ stands for the set of free variables of M and $\Lambda^0_{+\|}$ is the set of terms M such that $FV(M) = \varnothing$. Moreover, we shall refer to the following set

$$\mathsf{Par} = \{(M \parallel N) \mid M, N \in \Lambda_{+\|}\}.$$

As discussed in [18], we need to distinguish between partial and total values. We define the set Val of *values* according to the grammar

$$V ::= v \mid \lambda x.M \mid \lambda v.M \mid V \| M \mid M \| V$$

and the set TVal of *total values* as the subset of Val

$$W ::= v \mid \lambda x.M \mid \lambda v.M \mid W \| W.$$

A value V is *partial* iff $V \notin \mathsf{TVal}$. The main difference between the partial and total values concerns the parallel operator. Note that we require both M and N to be total values to ensure that $M \| N$ is a total value, while in general it suffices that either M or N is a value to have that $M \| N$ is a value. For example, $\mathbf{I} \| (\mathbf{K} + \mathbf{O})$ is a partial value, while both $\mathbf{I} \| \mathbf{K}$ and $\mathbf{I} \| \mathbf{O}$ are total values.

We now introduce a reduction relation which is intended to formalize the expected behaviour of a machine which evaluates in a synchronous way parallel compositions, until a value is produced. Partial values can be further evaluated, and this is essential to deal with an application of a call-by-value abstraction. Therefore, in some cases an asynchronous evaluation of parallel composition is permitted.

The reduction relation $\rightarrow$ is the least binary relation over $\Lambda^0_{+\|}$ such that

$$(\beta) \quad (\lambda x.M)\,N \rightarrow M[N/x], \qquad (\beta_v) \quad \frac{W \in \mathsf{TVal}}{(\lambda v.M)\;W \rightarrow M[W/v]}$$

$$(\mu_v) \quad \frac{N \rightarrow N' \quad N \notin \mathsf{Val}}{(\lambda v.M)\;N \rightarrow (\lambda v.M)\;N'}, \qquad (\beta_v \|) \quad \frac{V \rightarrow V' \quad V \in \mathsf{Val}}{(\lambda v.M)\;V \rightarrow M[V/v] \,\|\, (\lambda v.M)\;V'}$$

$$(\|_{app}) \quad (M \,\|\, N)\,L \rightarrow ML \,\|\, NL, \qquad (v) \quad \frac{M \rightarrow M' \quad M \notin \mathsf{Val} \cup \mathsf{Par}}{MN \rightarrow M'N}$$

$$(\|_s) \quad \frac{M \rightarrow M' \quad N \rightarrow N'}{M \,\|\, N \rightarrow M' \,\|\, N'}, \qquad (\|_a) \quad \frac{M \rightarrow M' \quad W \in \mathsf{TVal}}{M \,\|\, W \rightarrow M' \,\|\, W, \quad W \,\|\, M \rightarrow W \,\|\, M'}$$

$$(+) \quad M + N \rightarrow M, \qquad M + N \rightarrow N.$$

We denote by $\xrightarrow{n}$ the n-times self-composition of $\rightarrow$ and by $\rightarrow^*$ the reflexive and transitive closure of $\rightarrow$.

As in [18] a term is convergent if and only if all reduction paths will eventually reach a value. Let $M \in \Lambda^0_{+\|}$; then

$$M \Downarrow \quad \Leftrightarrow \quad \exists n.M \xrightarrow{n} N \Rightarrow N \in \mathsf{Val}.$$

Note that $M\Downarrow$ implies that $M \xrightarrow{n} N \Rightarrow N \in \mathsf{Val}$ holds for almost all n. Also note that $(M+N)\Downarrow$ if and only if both $M\Downarrow$ and $N\Downarrow$. On the other hand, $(M \,\|\, N)\Downarrow$ if and only if either $M\Downarrow$ or $N\Downarrow$ (or both).

Rule $(\beta_v \|)$ asks for some explanation. Given a value V, we cannot decide whether it has been sufficiently evaluated to perform the reduction step $(\lambda v.M)\;V \rightarrow M[V/v]$, or if it is necessary to reduce V further, before contracting the outermost β-redex. We cannot reduce V as long as possible, since this could not terminate. In the meantime, $M[V/v]$ can diverge while $M[V'/v]$ can converge for all V' which are reducts of V. For example, if $M \equiv vv\mathbf{\Omega}\mathbf{I}$, we have that $M[\mathbf{I} \,\|\, (\mathbf{K}+\mathbf{O})/v] \rightarrow^* \mathbf{\Omega}\mathbf{I} \,\|\, \mathbf{\Omega} \,\|\, \mathbf{\Omega}\mathbf{I}$, which will never reduce to a value, while both $M[\mathbf{I} \,\|\, \mathbf{K}/v]$ and $M[\mathbf{I} \,\|\, \mathbf{O}/v]$ reduce (deterministically) to values. On the other hand, any effective description of the operational semantics calls for a definition of a recursive one-step reduction relation. This explains why rule $(\beta_v \|)$ to compute $(\lambda v.M)\;V$ "takes the best" between the terms $M[V/v]$ and $(\lambda v.M)\;V'$, for an arbitrary V' such that $V \rightarrow^* V'$. We realize this by evaluating in parallel (using the operator $\|$) $M[V/v]$ and $(\lambda v.M)\;V'$ for all V' such that $V \rightarrow V'$. Then $(\lambda v.M)\;V$ will have as many reduction paths as V.

Notice that our reduction is highly nondeterministic, and also very sensitive to counting the number of steps. For example the only reductions of the term

$$(\lambda v.v\mathbf{I})(\mathbf{I} \,\|\, \lambda x.(\mathbf{K}+\mathbf{O}))$$

are

$$(\lambda v.v\mathbf{I})(\mathbf{I} \parallel \lambda x.(\mathbf{K}+\mathbf{O})) \to (\mathbf{I} \parallel \lambda x.(\mathbf{K}+\mathbf{O}))\,\mathbf{I} \to \mathbf{II} \parallel (\lambda x.(\mathbf{K}+\mathbf{O}))\,\mathbf{I}$$
$$\to \mathbf{I} \parallel (\mathbf{K}+\mathbf{O}) \to \mathbf{I} \parallel \mathbf{K}$$

and the symmetrical

$$(\lambda v.v\mathbf{I})(\mathbf{I} \parallel \lambda x.(\mathbf{K}+\mathbf{O})) \to^* \mathbf{I} \parallel \mathbf{O}.$$

But adding an application of the identity function we get the possible reduction:

$$(\lambda v.v\mathbf{I})(\mathbf{I} \parallel (\mathbf{I}(\lambda x.(\mathbf{K}+\mathbf{O}))) \to^* \mathbf{I} \parallel \mathbf{K} \parallel \mathbf{I} \parallel \mathbf{O}.$$

Analogously we get the terms $\mathbf{I} \parallel \mathbf{O} \parallel \mathbf{I} \parallel \mathbf{K}$, $\mathbf{I} \parallel \mathbf{K} \parallel \mathbf{I} \parallel \mathbf{K}$ and $\mathbf{I} \parallel \mathbf{O} \parallel \mathbf{I} \parallel \mathbf{O}$.

As usual we say that two terms are operationally equivalent if and only if in all contexts they exhibit the same behaviour with respect to convergence.

Let $M, N \in \Lambda_{+\parallel}$. Then

$$M \simeq^{\mathcal{O}} N \quad \Leftrightarrow \quad \forall C[\ \ ].C[M] \Downarrow \quad \Leftrightarrow \quad C[N] \Downarrow,$$

where $C[M], C[N] \in \Lambda^0_{+\parallel}$.

The operational semantics overcomes the above-discussed anomaly of our reduction relation. As regards to the previous example, it is clear that $\mathbf{I} \parallel \mathbf{K} \simeq^{\mathcal{O}} \mathbf{I} \parallel \mathbf{K} \parallel \mathbf{I} \parallel \mathbf{K}$ and $\mathbf{I} \parallel \mathbf{O} \simeq^{\mathcal{O}} \mathbf{I} \parallel \mathbf{O} \parallel \mathbf{I} \parallel \mathbf{O}$. Instead every context that converges when filled with $\mathbf{I} \parallel \mathbf{K}$ or $\mathbf{I} \parallel \mathbf{O}$, converges also when filled with $\mathbf{I} \parallel \mathbf{K} \parallel \mathbf{I} \parallel \mathbf{O}$ or $\mathbf{I} \parallel \mathbf{O} \parallel \mathbf{I} \parallel \mathbf{K}$, but the opposite is not always true. Since we require that all reduction paths out of a term reach a value for assuring convergence, we obtain that

$$(\lambda v.v\mathbf{I})(\mathbf{I} \parallel \lambda x.(\mathbf{K}+\mathbf{O})) \simeq^{\mathcal{O}} (\lambda v.v\mathbf{I})(\mathbf{I} \parallel (\mathbf{I}(\lambda x.(\mathbf{K}+\mathbf{O}))).$$

As in [18] we can extend the type assignment system $\vdash$ to terms in $\Lambda_{+\parallel}$ by adding the following rules[1]:

$$(\to \mathrm{I}_v) \quad \frac{\Gamma \vdash \lambda v.M\colon (\sigma \to \rho) \wedge (\tau \to \rho)}{\Gamma \vdash \lambda v.M\colon \sigma \vee \tau \to \rho}$$

$$(+\mathrm{I}) \quad \frac{\Gamma \vdash M\colon \sigma \quad \Gamma \vdash N\colon \tau}{\Gamma \vdash M+N\colon \sigma \vee \tau}$$

$$(\parallel \mathrm{I}) \quad \frac{\Gamma \vdash M\colon \sigma \quad \Gamma \vdash N\colon \tau}{\Gamma \vdash M \parallel N\colon \sigma \wedge \tau}.$$

We write $\vdash_{\mathscr{E}}$ to denote derivability in this extended system. The so-obtained typing enjoys the subject reduction property, as proved in [18]. Instead the subject expansion fails. For example, $\vdash_{\mathscr{E}} \mathbf{I}\colon \omega \to \omega$, but $\nvdash_{\mathscr{E}} \mathbf{I} + \Omega\colon \omega \to \omega$.

[1] The present definition simplifies that of [18].

For the present development, we are mainly interested in the full abstraction result of [18], which here can be rephrased as follows.

THEOREM 9. *For all terms $M, N \in \Lambda_{+\|}$, the condition $M \simeq^{\mathcal{O}} N$ holds if and only if*

$$\Gamma \vdash_{\mathscr{E}} M : \sigma \qquad \textit{iff} \quad \Gamma \vdash_{\mathscr{E}} N : \sigma \quad \textit{for all } \Gamma \textit{ and } \sigma.$$

Notice that Theorem 1 is the restriction of Theorem 9 to the case $M, N \in \Lambda$.

5. LÉVY–LONGO TREES AND APPROXIMANTS

In this section we consider only terms in $\Lambda_{\bot,\top}$. It is easy to verify that the set of normal forms $\mathscr{A} \subseteq \Lambda_{\bot,\top}$ with respect to the reduction relation introduced in Section 3 is the least set satisfying:

1. $\bot, \top \in \mathscr{A}$;
2. $A_1, ..., A_n \in \mathscr{A} \Rightarrow xA_1 \cdots A_n \in \mathscr{A}$ $(n \geqslant 0)$;
3. $A \in \mathscr{A}, A \not\equiv \top \Rightarrow \lambda x.A \in \mathscr{A}$.

The elements of $\mathscr{A}$ are called *approximate normal forms.*

We define two preorder relations on approximate normal forms. The first preorder generalizes that of [23], making $\top$ the top element of the approximate normal forms of the shape $\lambda\vec{x}.\bot$. In the second preorder $\top$ is the top of the whole $\mathscr{A}$, and an η-redex is always less than its contractum.

1. The relation $\preccurlyeq$ is the least preorder in $\mathscr{A}$ such that:
   (a) $\bot \preccurlyeq A$;
   (b) $A \preccurlyeq A', A' \not\equiv \top \Rightarrow \lambda x.A \preccurlyeq \lambda x.A'$;
   (c) $A_1 \preccurlyeq A'_1, ..., A_n \preccurlyeq A'_n \Rightarrow xA_1 \cdots A_n \preccurlyeq xA'_1 \cdots A'_n$;
   (d) $\lambda\vec{x}.\bot \preccurlyeq \top$.
2. The relation $\preccurlyeq_\eta$ is the least preorder in $\mathscr{A}$, such that it satisfies clauses (a), (b), (c) above, and moreover
   (d') $A \preccurlyeq_\eta \top$;
   (e) $\lambda y.xA_1 \cdots A_n y \preccurlyeq_\eta xA_1 \cdots A_n$, where $y \notin FV(xA_1 \cdots A_n)$.

The following lemma gives a characterization of $\preccurlyeq_\eta$.

LEMMA 10. *The condition $A \preccurlyeq_\eta B$ is equivalent to the disjunction of the following conditions (vectors of variables and terms may be empty):*

— $A \equiv \lambda\vec{x}.\bot$ *and* $B \equiv \lambda\vec{x}\vec{y}.\bot$;

— $A \equiv \lambda\vec{x}.\bot$ *and B is a head normal form*;

— $B \equiv \top$;

— $A \equiv \lambda\vec{x}\vec{z}.y\vec{C}\vec{Z}$ *and* $B \equiv \lambda\vec{x}.y\vec{D}$, *with* $\vec{C} \preccurlyeq_\eta \vec{D}$ *and* $\vec{Z} \preccurlyeq_\eta \vec{z}$, *componentwise.*

Proof. ($\Rightarrow$) The proof is by induction with respect to the definition of $\leqslant_\eta$. The only nontrivial case is transitivity. Details are left to the reader.

($\Leftarrow$) The only nontrivial observation is $\lambda\vec{x}\vec{y}.\bot \leqslant_\eta \lambda\vec{x}\vec{y}.z\vec{A}\vec{y} \leqslant_\eta \lambda\vec{x}.z\vec{A}$. ∎

For each term $M \in \Lambda_{\bot,\top}$, we define the set $\mathscr{A}(M)$ of its approximants. (The goal is to express properties of pure λ-terms with the help of their approximants, but for uniformity we need also to define approximants of terms involving $\bot$ and $\top$.) A natural way to state this definition would be to first define a *direct approximant* of a term M, denoted $\varpi(M)$, and then to take $\mathscr{A}(M) = \{\varpi(N) \mid N =_\beta^{\bot,\top} M\}$ as the set of approximants of M. Let us recall that according to Lévy [23] we have

— $\varpi(xM_1 \cdots M_m) = x\varpi(M_1) \cdots \varpi(M_m)$, for $m \geqslant 0$;

— $\varpi(\lambda x.M) = \lambda x.\varpi(M)$;

— $\varpi((\lambda x.N)\, PM_1 \cdots M_m) = \bot$, for $m \geqslant 0$.

Unfortunately, this does not reflect the possibility of an "infinite abstraction," caused by $\top$. Thus, we have to use the definition by structural induction on the set of approximate normal forms. A term A is an *approximant* of a term M if and only if one of the following holds (vectors of variables and terms may be empty):

— M has a head normal form $\lambda\vec{x}.y\vec{P}$, and A is of the form $\lambda\vec{x}.y\vec{A}$, where $\vec{A}$ are approximants of $\vec{P}$;

— M reduces to $\lambda\vec{x}.Q$, and A is of the form $\lambda\vec{x}.\bot$;

— M reduces to $\lambda\vec{x}.Q$ for arbitrarily long $\vec{x}$, or to $\top$, and $A \equiv \top$;

— M reduces to $\top$, and A is of the form $\lambda\vec{y}.\bot$.

Clearly, $\bot \in \mathscr{A}(M)$ for all M.

We say that a term M has a *weak head normal form* iff it has a head normal form or it is equal to an abstraction, in the sense of $=_\beta^{\bot,\top}$. Clearly, M has a weak head normal form iff $\mathscr{A}(M) \neq \{\bot\}$.

Some properties of sets of approximants follow straightforwardly.

LEMMA 11.

1. *If* $M =_\beta^{\bot,\top} N$ *then* $\mathscr{A}(M) = \mathscr{A}(N)$.
2. *If* $\lambda x.A \in \mathscr{A}(M)$ *then* $M =_\beta^{\bot,\top} \lambda x.N$, *where* $A \in \mathscr{A}(N)$.
3. *If* $x\vec{A} \in \mathscr{A}(M)$ *then* $M =_\beta^{\bot,\top} x\vec{N}$, *where* $\vec{A} \in \mathscr{A}(\vec{N})$ *componentwise.*

Proof. 1. Easy induction with respect to the length of approximants.

2–3. Immediate from the definition. ∎

As in the case of the standard definition [23], the set $\mathscr{A}(\ )$ turns out to be an ideal with respect to $\leqslant$.

LEMMA 12. *For each M, the set $\mathscr{A}(M)$ is an ideal* (*that is, a downward closed upper semi-lattice*) *with respect to* $\leqslant$.

Proof. It is easy to verify that $\mathscr{A}(M)$ is downward closed, so we prove that it is directed. If M does not have a head normal form, then the set $\mathscr{A}(M)$ is linearly (pre)ordered with respect to $\leqslant$. Otherwise, if M has a head normal form $\lambda\vec{x}.y\vec{N}$, then the proof is by induction on the length of approximants. ∎

Since $\mathscr{A}(M)$ is *not* downward closed with respect to $\leqslant_\eta$, we also need to consider the set

$$\mathscr{A}^*(M) = \{A \in \mathscr{A} \mid A \leqslant_\eta B \text{ for some } B \in \mathscr{A}(M)\}.$$

Note that if $\top \in \mathscr{A}^*(M)$ then $\mathscr{A}^*(M)$ is the whole set of approximate normal forms, i.e. $\mathscr{A}^*(M) = \mathscr{A}$.

The inclusion $\mathscr{A}(M) \subseteq \mathscr{A}(N)$ implies $\mathscr{A}^*(M) \subseteq \mathscr{A}^*(N)$. The converse does not hold (take, e.g., $M \equiv \lambda x.yx$ and $N \equiv y$), but we have a slightly weaker property.

LEMMA 13. *If $\mathscr{A}^*(M) = \mathscr{A}^*(N)$, then $\mathscr{A}(M) = \mathscr{A}(N)$.*

Proof. First note that if $\mathscr{A}^*(M) = \mathscr{A}^*(N)$ and one of the terms M, N has a head normal form $\lambda\vec{x}.y\vec{P}$, then the other one must also have a head normal form $\lambda\vec{x}.y\vec{Q}$ (with the same $\vec{x}$, y, and the same length of $\vec{P}$ and $\vec{Q}$). Indeed, if M has a head normal form as above, then it has an approximant $A \equiv \lambda\vec{x}.y\vec{\bot}$. There is $B \in \mathscr{A}(N)$ with $A \leqslant_\eta B$. By Lemma 10, the term B (which is in head normal form) must either be of shape $\lambda\vec{x}.y\vec{C}$, or have a lambda prefix shorter than that of A. In the former case, A is an approximant of N, in the latter case we have another approximant of M, say A', such that $A \leqslant_\eta B \leqslant_\eta A'$. But then, the lambda prefix of A' would also have to be shorter than the vector $\vec{x}$. But this is impossible for an approximant of M in head normal form.

Having observed the above, we prove the following claim, by induction with respect to the size of A:

For all M and N, if $\mathscr{A}^*(M) = \mathscr{A}^*(N)$ and $A \in \mathscr{A}(M)$, then $A \in \mathscr{A}(N)$.

Assume first that both M and N have head normal forms, say $\lambda\vec{x}.yP_1 \cdots P_n$ and $\lambda\vec{x}.yQ_1 \cdots Q_n$, respectively. Then, for all $i = 1, ..., n$, we have $\mathscr{A}^*(P_i) = \mathscr{A}^*(Q_i)$. Indeed, if $D \in \mathscr{A}(P_i)$, then we have an approximant of M of the form $\lambda\vec{x}.y\bot \cdots \bot D \bot \cdots \bot$, with D on the ith position. There must be $E \in \mathscr{A}(N)$ such that $\lambda\vec{x}.y\bot \cdots \bot D \bot \cdots \bot \leqslant_\eta E$. By Lemma 10, we must have $E \equiv \lambda\vec{x}.yF_1 \cdots F_n$, where $D \leqslant_\eta F_i \in \mathscr{A}(Q_i)$. Thus, $\mathscr{A}(P_i) \subseteq \mathscr{A}^*(Q_i)$, which implies $\mathscr{A}^*(P_i) \subseteq \mathscr{A}^*(Q_i)$. The converse inclusion follows from the symmetry.

Assume that $A \in \mathscr{A}(M)$. If $A \equiv \lambda\vec{y}.\bot$ then $A \in \mathscr{A}(N)$. Otherwise, we have $A \equiv \lambda\vec{x}.yD_1 \cdots D_n$. For each $i = 1, ..., n$, we apply the induction hypothesis to D_i, to obtain that $D_i \in \mathscr{A}(Q_i)$. This gives $A \in \mathscr{A}(N)$.

Now suppose that neither N nor M has a head normal form. If $\top \in \mathscr{A}(N)$, then $\mathscr{A}(N)$ includes $\top$ and all terms of the form $\lambda\vec{z}.\bot$, among them A. Otherwise, we have $A \equiv \lambda\vec{y}.\bot$, and if $B \in \mathscr{A}(N)$ is such that $A \leqslant_\eta B$ then B must be of the form $\lambda\vec{y}\vec{x}.\bot$. Thus $N =_\beta \lambda\vec{y}\vec{x}.Q$, for some Q, and we conclude that $A \in \mathscr{A}(N)$. ∎

We prove also other features of $\mathscr{A}^*(\ )$.

LEMMA 14.

1. *If $\mathscr{A}(M) \neq \mathscr{A}(N)$, then there is $A \in \mathscr{A}(M)$ such that $A \notin \mathscr{A}^*(N)$ or vice versa.*

2. *If M has a weak head normal form and $z \notin FV(M)$, then $\mathscr{A}^*(\lambda z.Mz) \subseteq \mathscr{A}^*(M)$.*

3. *If $\vec{A} \in \mathscr{A}^*(\vec{M})$ componentwise, then $x\vec{A} \in \mathscr{A}^*(x\vec{M})$.*

Proof. 1. It is enough to observe that $\mathscr{A}(M) \subseteq \mathscr{A}^*(N)$ and $\mathscr{A}(N) \subseteq \mathscr{A}^*(M)$ imply $\mathscr{A}^*(M) \subseteq \mathscr{A}^*(N)$ and $\mathscr{A}^*(N) \subseteq \mathscr{A}^*(M)$. So we are done by Lemma 13.

2. It suffices to show $\mathscr{A}(\lambda z.Mz) \subseteq \mathscr{A}^*(M)$. The case $M =_\beta^{\perp,\top} \lambda x.M'$ is trivial. Let $M =_\beta^{\perp,\top} x\vec{N}$. By definition $A \in \mathscr{A}(\lambda z.Mz)$ implies one of the following alternatives: $A \equiv \perp$, $A \equiv \lambda z.\perp$, or $A \equiv \lambda z.x\vec{A}Z$, where $\vec{A} \in \mathscr{A}(\vec{N})$ componentwise, and $Z \in \mathscr{A}(z)$. Therefore $x\vec{A} \in \mathscr{A}(M)$ and Z is either $\perp$ or z. In all cases we can conclude that $A \preccurlyeq_\eta x\vec{A}$.

3. If $\vec{A} \in \mathscr{A}^*(\vec{M})$ then there are $\vec{A}' \in \mathscr{A}(\vec{M})$ such that $\vec{A} \preccurlyeq_\eta \vec{A}'$. By definition this implies $x\vec{A}' \in \mathscr{A}(x\vec{M})$ and $x\vec{A} \preccurlyeq_\eta x\vec{A}'$. ∎

Recall that a weak head normal form is a head normal form or an abstraction. The *Lévy–Longo tree* $LL(M)$ of a term $M \in \Lambda_{\perp,\top}$ (see [24]) is defined by cases on M:

(1) If $M =_\beta^{\perp,\top} \top$, or M reduces to $\lambda\vec{x}.Q$, for arbitrarily long $\vec{x}$, then

$$LL(M) = \cdot \top.$$

(2) If $M =_\beta^{\perp,\top} \lambda\vec{x}.Q$, where Q has no weak head normal form, then

$$LL(M) = \cdot \lambda\vec{x}.\perp.$$

(3) If M has a head normal form $\lambda\vec{x}.yN_1 \cdots N_n$, then

$$LL(M) = \begin{tikzcd} & \lambda\vec{x}.y \arrow[dl, dash] \arrow[dr, dash] & \\ LL(N_1) & \cdots & LL(N_n). \end{tikzcd}$$

Some examples of Lévy–Longo trees are shown in Fig. 1.

We get immediately that $\top \in \mathscr{A}(M)$ (and $\top \in \mathscr{A}^*(M)$) iff $LL(M) = \cdot \top$. With the standard definition of approximants, $LL(M) = \cdot \top$ implies only that $\lambda\vec{x}.\perp \in \mathscr{A}(M)$ for arbitrarily long $\vec{x}$.

From the above it follows easily that equality of Lévy–Longo trees coincides with equality of sets of approximants $\mathscr{A}(M)$. This property holds also for the standard definition of sets of approximants [24].

Lemma 15. *For all $M, N \in \Lambda_{\perp,\top}$, the conditions $LL(M) = LL(N)$ and $\mathscr{A}(M) = \mathscr{A}(N)$ are equivalent.*

The approximation theorem allows us to relate the sets of types of a term to the set of its approximants, and therefore to its Lévy–Longo tree.

Theorem 16 (Approximation theorem). *$\Gamma \vdash M : \sigma$ iff there is $A \in \mathscr{A}(M)$ such that $\Gamma \vdash A : \sigma$.*

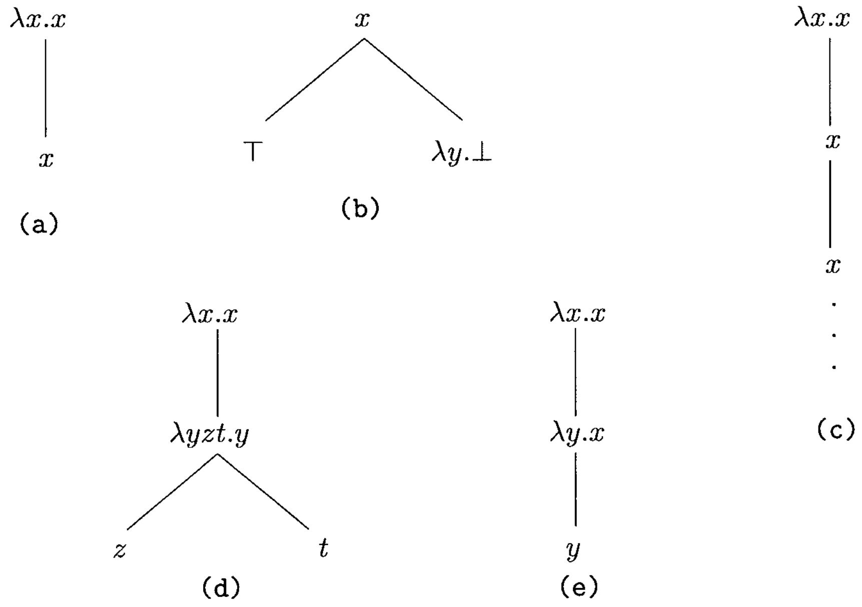

FIG. 1. (a) $LL(\mathbf{\Delta})$; (b) $LL(x(\mathbf{YK})(\mathbf{K\Omega}))$; (c) $LL(\mathbf{Y})$; (d) $LL(\lambda x.x(\lambda yzt.yzt))$; (e) $LL(\lambda x.(\lambda y.xy))$.

We defer the proof of Theorem 16 to the Appendix. The following is an immediate consequence of the above result.

THEOREM 17. *Let $\mathscr{A}(N) \subseteq \mathscr{A}(M)$. Then $\Gamma \vdash N: \sigma$ implies $\Gamma \vdash M: \sigma$.*

We derive one more consequence of Theorem 16, which shows how the types characterize properties of terms.

THEOREM 18. *M has a weak head normal form iff there is a basis Γ and a type $\sigma \not\sim \omega$ such that $\Gamma \vdash M: \sigma$.*

Proof. It is easy to verify that for every term A in approximate normal form we have $A \not\equiv \bot$ iff there is a basis Γ and a type $\sigma \not\sim \omega$ such that $\Gamma \vdash A: \sigma$. On the other hand, M has a weak head normal form iff M has an approximant $A \not\equiv \bot$. Hence the conclusion follows from Theorem 16. ∎

6. PRINCIPAL PAIRS

In this section, we consider type schemes, built out of types in $Type_{\wedge}$ by adding type variables. The set of all type schemes is denoted by $Type_{\wedge t}$. Notice that we do not consider a logic with variables; we only use the language $Type_{\wedge t}$ to define the principal pairs of approximate normal forms. To simplify our treatment, we split the set of type variables in two disjoint subsets $V = \{t_i : i \in \mathsf{N}\}$ and $V' = \{t'_i : i \in \mathsf{N}\}$. We use respectively t and t' (without indices) as (meta)variables ranging over V and V', respectively. So the set of type schemes is defined by

$$\sigma ::= t \mid t' \mid \omega \mid \sigma \to \sigma \mid \sigma \wedge \sigma.$$

We consider also basis schemes whose predicates are type schemes. In what follows, the notation $TV(\langle\Gamma;\sigma\rangle)$ stands for the set of type variables which occur in the basis scheme Γ or in the type scheme σ.

For each $A\in\mathscr{A}$, we define the *principal pair*, denoted $pp(A)$ by induction:

1. $pp(\bot)=\langle\varnothing;\omega\rangle$;
2. $pp(\top)=\langle\varnothing;t'\rangle$, where $t'\in V'$;
3. if $pp(A_i)=\langle\Gamma_i;\sigma_i\rangle$, $TV(\langle\Gamma_i;\sigma_i\rangle)\cap TV(\langle\Gamma_j;\sigma_j\rangle)=\varnothing$ for $1\leqslant i\neq j\leqslant n$ and $t\in V$ is fresh, then

$$pp(xA_1\cdots A_n)=\left\langle\left(\biguplus_{i\leqslant n}\Gamma_i\right)\uplus\{x\colon\sigma_1\to\cdots\to\sigma_n\to t\};t\right\rangle\qquad(n\geqslant 0);$$

4. if $pp(A)=\langle\Gamma,x\colon\tau;\sigma\rangle$, then $pp(\lambda x.A)=\langle\Gamma;\tau\to\sigma\rangle$;
5. if $pp(A)=\langle\Gamma;\sigma\rangle$ and $x\notin\mathrm{Dom}(\Gamma)$, then $pp(\lambda x.A)=\langle\Gamma;\omega\to\sigma\rangle$.

The set of all principal pairs is denoted by Π. A type scheme σ is *principal* iff $\langle\Gamma;\sigma\rangle\in\Pi$, for some basis scheme Γ. A basis scheme Γ is *principal* iff $\langle\Gamma;\sigma\rangle\in\Pi$ for some type scheme σ. We assume that principal pairs are taken up to renaming of their type variables (agreeing with the splitting between V and V'), so that we may have a unique principal pair for each approximate normal form.

As an immediate consequence of the definition we get the following observation. Let $pp(A)=\langle\Gamma;\sigma\rangle$, and let s be an arbitrary substitution mapping type variables into types. Then $s(\Gamma)\vdash A\colon s(\sigma)$. Note that the converse does not hold, i.e., our principal pair is not principal with respect to substitutions. Indeed, $pp(\lambda xy.xy)=\langle\varnothing;(t_1\to t_2)\to t_1\to t_2\rangle$ but $\vdash\lambda xy.xy\colon\omega\to\omega\to\omega$. Principal pairs for intersection and union types are studied in [4].

Let $1\leqslant n\leqslant m$. We will define a substitution $s_{m,n}$. First for every $j\leqslant 2n$ we define

$$\beta_j=\omega^{m+j-1}\to\omega_1\to\omega^{m+2n-j+1}\to\omega_1.$$

Using β_j's we define for every $i\leqslant n$,

$$\alpha_i=\beta_i\vee\beta_{n+i}.$$

Now, we define the substitution $s_{m,n}$ as

- $s_{m,n}(t_i)=\alpha_i$ for $i\leqslant n$ and $t_i\in V$;
- $s_{m,n}(t_i)=\omega$ for $i>n$ and $t_i\in V$;
- $s_{m,n}(t'_i)=\omega_{3m+2n+3}$ for all $t'_i\in V'$.

We extend $s_{m,n}$ to type schemes and base schemes in the usual way.

Next we define the auxiliary measures, where $M\in\Lambda_{\bot,\top}$ and $A\in\mathscr{A}$,

- order(M,h) is the maximum branching of nodes at height less than or equal to h in $LL(M)$;

- degree(M, h) is the maximum number of consecutive abstractions in labels of nodes at height less than or equal to h in $LL(M)$;
- $v(A)$ is the number of type variables from V (i.e., nonprimed versions) which occur in $pp(A)$;
- $\kappa(M, A) = \max\{a + \text{degree}(M, h), a + \text{order}(M, h), v(A)\}$, where a is the number of arrows which occur in $pp(A)$ and h is the height of $LL(A)$.

The following is the main tool which provides the discriminating power. We defer the proof of this theorem to the Appendix.

THEOREM 19. (Principal pair theorem). *Let $M \in \Lambda_{\perp,\top}$, $A \in \mathscr{A}$ and $pp(A) = \langle \Gamma; \sigma \rangle$. Let $n = v(A)$ and $m = \kappa(M, A)$. If $s_{m,n}(\Gamma) \vdash M : s_{m,n}(\sigma)$, then $A \in \mathscr{A}^*(M)$.*

Note that the converse implication in Theorem 19 is not true: take $A \equiv \lambda x.\perp$ and $M \equiv y$; then $\not\vdash y : \omega \to \omega$. Moreover we really need to consider types with union and intersection to obtain the result of Theorem 19. In fact, Theorem 19 does not hold if we do not allow $\vee$ in the range of substitutions. For example, we have $pp(x) = \langle x : t; t \rangle$ and for all $\sigma \in Type_\wedge$ we get $x : \sigma \vdash \lambda y.xy : \sigma$, but $x \notin \mathscr{A}^*(\lambda y.xy)$.

7. MAIN RESULT—A DISCRIMINATION ALGORITHM

This section contains our main result.

THEOREM 20. *For all terms $M, N \in \Lambda_{\perp,\top}$, $LL(M) = LL(N)$ if and only if for all Γ and σ,*

$$\Gamma \vdash M : \sigma \qquad \textit{iff} \quad \Gamma \vdash N : \sigma.$$

Proof. $(\Rightarrow)$ From Lemma 15 and Theorem 17.

$(\Leftarrow)$ Assume $LL(M) \neq (N)$. Then by Lemma 15 and Lemma 14(1) there is $A \in \mathscr{A}(M)$ such that $A \notin \mathscr{A}^*(N)$ (or vice versa). Let $pp(A) = \langle \Gamma; \sigma \rangle$, $m = \kappa(N, A)$, and $n = v(A)$. By Theorem 16, we have $s_{m,n}(\Gamma) \vdash M : s_{m,n}(\sigma)$. If we assume $s_{m,n}(\Gamma) \vdash N : s_{m,n}(\sigma)$, then by Theorem 19 we get $A \in \mathscr{A}^*(N)$, which is a contradiction. So we can conclude that

$$s_{m,n}(\Gamma) \vdash M : s_{m,n}(\sigma) \qquad \text{and} \qquad s_{m,n}(\Gamma) \not\vdash N : s_{m,n}(\sigma);$$

i.e. $\langle s_{m,n}(\Gamma); s_{m,n}(\sigma) \rangle$ discriminates between M and N. ∎

It should be clear that the above proof, together with that of Theorem 19, describes a simple algorithm to discriminate by typing two terms with different Lévy–Longo trees. Since the Lévy–Longo trees are possibly infinite, the inputs of our algorithm are the trees of M and N cut at a height sufficiently big to be different. This allows us to find an approximate normal form A such that $A \in \mathscr{A}(M)$ and $A \notin \mathscr{A}^*(N)$. We compute:

$h =$ the height of $LL(A)$;
$o = \text{order}(N, h)$;
$d = \text{degree}(N, h)$;
$\langle \Gamma; \sigma \rangle = pp(A)$;
$n = v(A)$;
$a =$ the number of arrows in $\langle \Gamma; \sigma \rangle$;
$m = \max\{a + d, a + o, n\}$.

Now we have

$$s_{m,n}(\Gamma) \vdash M : s_{m,n}(\sigma) \qquad \text{and} \qquad s_{m,n}(\Gamma) \nvdash N : s_{m,n}(\sigma).$$

EXAMPLE 21. Consider $M \equiv x$ and $N \equiv \lambda y.xy$. The corresponding trees are respectively subtrees of the trees (a) and (e) in Fig. 1 of Section 5. Clearly $M \notin \mathscr{A}^*(N)$.

We have $h = 0$, $o = \text{order}(N, 1) = 1$ and $d = \text{degree}(N, 1) = 1$. We get $pp(M) = \langle \{x : t_1\}; t_1 \rangle$, $n = 1$, $a = 0$. Then $m = 1$, $s_{1,1}(t_1) = \alpha_1 = \beta_1 \vee \beta_2$, *where*

$$\beta_1 \equiv \omega \to \omega_1 \to \omega^3 \to \omega_1,$$
$$\beta_2 \equiv \omega^2 \to \omega_1 \to \omega^2 \to \omega_1.$$

To get $x : \alpha_1 \vdash \lambda y.xy : \alpha_1$ we need either $x : \alpha_1 \vdash \lambda y.xy : \beta_1$ or $x : \alpha_1 \vdash \lambda y.xy : \beta_2$ by Theorem 6(2) and Lemma 5(1). So we would need by Theorem 6(1, 3, 4), either $\alpha_1 \leqslant \omega \to \beta_1$ *or* $\alpha_1 \leqslant \omega \to \beta_2$, but they are both false. Instead it is easy to verify that $x : \alpha_1 \vdash x : \alpha_1$.

EXAMPLE 22. Consider $M \equiv \mathbf{\Delta}$ and $N \equiv \lambda x.x(\lambda y.xy)$. The corresponding trees are respectively the trees (a) and (e) in Fig. 1 of Section 5. Clearly $M \notin \mathscr{A}^*(N)$.

We have $h = 1$, $o = \text{order}(N, 1) = 2$, and $d = \text{degree}(N, 1) = 1$. Let $\sigma \equiv t_1 \wedge (t_1 \to t_2) \to t_2$. We get $pp(M) = \langle \varnothing; \sigma \rangle$, $n = 2$, $a = 2$. Then $m = 4$, $s_{4,2}(\sigma) = \tau \to \alpha_2$, where

$$\tau \equiv \alpha_1 \wedge (\alpha_1 \to \alpha_2),$$
$$\alpha_1 \equiv \beta_1 \vee \beta_3, \alpha_2 \equiv \beta_2 \vee \beta_4, \quad \beta_1 \equiv \omega \to \omega_1 \to \omega^8 \to \omega_1,$$
$$\beta_2 \equiv \omega^5 \to \omega_1 \to \omega^7 \to \omega_1, \quad \beta_3 \equiv \omega^6 \to \omega_1 \to \omega^6 \to \omega_1,$$
$$\beta_4 \equiv \omega^7 \to \omega_1 \to \omega^5 \to \omega_1.$$

To have $\vdash N : \tau \to \alpha_2$ we need by Theorem 6(3) that $x : \tau \vdash x(\lambda y.xy) : \alpha_2$. We get by Theorem 6(1, 4) that $x : \tau \vdash \lambda y.xy : \alpha_1$, which requires either $x : \tau \vdash \lambda y.xy : \beta_1$ or $x : \tau \vdash \lambda y.xy : \beta_3$ by Theorem 6(2) and Lemma 5(1). So we would need by Theorem 6(1, 3, 4), either $\tau \leqslant \omega \to \beta_1$ or $\tau \leqslant \omega \to \beta_3$, but they are both false. Instead, it is easy to verify that $\vdash M : \tau \to \alpha_2$.

To find a suitable context $C[\ ]$ such that $C[M]\Downarrow$ and $C[N]\Uparrow$ we define by induction on τ the "test term" T_τ as

$$\begin{array}{ll}
\mathsf{R}_{\omega} \equiv \mathbf{\Omega}; & \mathsf{T}_{\omega} \equiv \lambda xy.y; \\
\mathsf{R}_{\sigma\to\tau} \equiv \lambda x.\mathsf{T}_{\sigma} x \,\mathsf{R}_{\tau}; & \mathsf{T}_{\sigma\to\tau} \equiv \lambda v.\mathsf{T}_{\tau}(v\, \mathsf{R}_{\sigma}); \\
\mathsf{R}_{\sigma\wedge\tau} \equiv \mathsf{R}_{\sigma} \parallel \mathsf{R}_{\tau}; & \mathsf{T}_{\sigma\wedge\tau} \equiv \lambda x.(\mathsf{T}_{\sigma} x + \mathsf{T}_{\tau} x); \\
\mathsf{R}_{\sigma\vee\tau} \equiv \mathsf{R}_{\sigma} + \mathsf{R}_{\tau}. & \mathsf{T}_{\sigma\vee\tau} \equiv \lambda v.(\mathsf{T}_{\sigma} v \parallel \mathsf{T}_{\tau} v), \qquad \text{where} \quad \sigma \vee \tau \neq \omega.
\end{array}$$

Now we can use the following result from [18].

THEOREM 23. *Let* $M \in \Lambda^{0}_{+\parallel}$. *Then* $\mathsf{T}_{\tau} M \Downarrow \;\Leftrightarrow\; \vdash_{\mathscr{E}} M : \tau$.

Let $M, N \in \Lambda$ and assume $\Gamma \vdash M : \sigma$ and $\Gamma \not\vdash N : \sigma$, where $FV(MN) = \{x_i \mid 1 \leqslant i \leqslant n\}$, and $\Gamma(x_i) = \tau_i$ for $1 \leqslant i \leqslant n$. Take $\tau = \tau_1 \to \cdots \to \tau_n \to \sigma$; then we have $\vdash \lambda x_1 \cdots x_n.M : \tau$ and $\not\vdash \lambda x_1 \cdots x_n.N : \tau$ by Theorem 6(3). Therefore, choosing $C[\;\;] = \mathsf{T}_{\tau}(\lambda x_1 \cdots x_n.[\;\;])$, we have by Theorem 23 that $C[M] \Downarrow$ and $C[N] \Uparrow$. So really the algorithm gives us a discriminating context.

EXAMPLE 24. 1. If M, N are as in Example 21, we get the following discriminating context $C[\;\;] \equiv \mathsf{T}_{\alpha_1}[\;\;] \equiv \lambda v.(\mathsf{T}_{\beta_1} v \parallel \mathsf{T}_{\beta_2} v) \equiv \lambda v.(\mathsf{T}_{\omega_1}(v\mathbf{\Omega}(\lambda x.\mathbf{\Omega})\,\mathbf{\Omega}^3) \parallel (\mathsf{T}_{\omega_1}(v\mathbf{\Omega}^2(\lambda x.\mathbf{\Omega})\,\mathbf{\Omega}^2))$, where $\mathsf{T}_{\omega_1} \equiv \lambda vy.y$.

2. If M, N are as in Example 22, we get the discriminating context $C[\;\;] \equiv \mathsf{T}_{\tau\to\alpha_2}[\;\;] \equiv \lambda v.\mathsf{T}_{\alpha_2}(v\mathsf{R}_{\tau})[\;\;]$, where $\mathsf{R}_{\tau} \equiv \mathsf{R}_{\alpha_1} \parallel \mathsf{R}_{\alpha_1\to\alpha_2} \equiv \mathsf{R}_{\alpha_1} \parallel \lambda x.\mathsf{T}_{\alpha_2} x \mathsf{R}_{\alpha_1}$, $\mathsf{T}_{\alpha_2} \equiv \lambda v.(\mathsf{T}_{\beta_2} v \parallel \mathsf{T}_{\beta_4} v) \equiv \lambda v.(\mathsf{T}_{\omega_1}(v\mathbf{\Omega}^5(\lambda x.\mathbf{\Omega})\,\mathbf{\Omega}^7) \parallel (\mathsf{T}_{\omega_1}(v\mathbf{\Omega}^7(\lambda x.\mathbf{\Omega})\,\mathbf{\Omega}^5))$, $\mathsf{R}_{\alpha_1} \equiv \mathsf{R}_{\beta_1} \parallel \mathsf{R}_{\beta_3} \equiv \lambda x_1 \cdots x_5.\mathsf{T}_{\omega_1} x_5(\lambda x_6 \cdots x_{14}.\mathbf{\Omega}) \parallel \lambda x_1 \cdots x_7.\mathsf{T}_{\omega_1} x_7(\lambda x_8 \cdots x_{14}.\mathbf{\Omega})$, and T_{ω_1} is as in the previous example.

8. CONCLUDING REMARKS

The literature related to the present work has mostly been quoted in the Introduction. Here we want to compare our development (based on [18]) with that of [2].

First we consider the logic $Type_{\vee\wedge}$ of intersection and union types, while Abramsky and Ong consider the logic $Type_{\wedge}$ of intersection types only. This is reflected in the solved domain equations: $D = \mathscr{P}^{\#}([D \to D]_{\perp})$ in [18] and $D = [D \to D]_{\perp}$ in [2].

Also the techniques to find these solutions are different; [2] uses domain prelocales following [1]. Instead, [18] uses a (simplified) version of information systems [35]: the filter model approach introduced in [8]. Domain prelocales allow us to describe the category of SFP domains, while filter models only allow us to describe complete ω-algebraic lattices. But in many cases filter models are sufficient, also for different languages; for example [21, 15] build fully abstract filter models of the π-calculus.

Lastly, there is a difference in the choice of the language: [2] builds a model for the lazy λ-calculus, while [18] builds a model for the concurrent λ-calculus introduced in Section 4. We can compare the local structure of these models by restricting ourselves to pure λ-terms. Let us consider x and $\lambda y.xy$; we have $\not\vdash x : \omega \to \omega$, while $\vdash \lambda y.xy : \omega \to \omega$. One can prove that with types in $Type_{\wedge}$ we get $\Gamma \vdash \lambda y.xy : \tau$

whenever $\Gamma \vdash x : \tau$. Therefore, in the model of [2] the interpretation of x is properly included in that of $\lambda y.xy$. But they are incomparable in the model of [18]. Indeed, in Example 21 we showed that $x : \alpha_1 \vdash x : \alpha_1$ while $x : \alpha_1 \nvdash \lambda y.xy : \alpha_1$, for a suitable α_1 containing $\vee$.

If we consider $\lambda x.xx$ and $\lambda x.x(\lambda y.xy)$, Sangiorgi proves in [33] that these terms have the same types in $Type_{\wedge}$. Therefore, they have the same interpretation in the Abramsky–Ong model. As a consequence that model equates λ-terms with different Lévy–Longo trees, as already noticed in [27]. Example 22 exhibits a type $\rho \in Type_{\vee\wedge}$ such that $\vdash x : \rho$, and $\nvdash \lambda y.xy : \rho$. We can show that $\lambda x.xx$ possesses all types in $Type_{\vee\wedge}$ derivable for $\lambda x.x(\lambda y.xy)$. Therefore, in the model of [18] the interpretation of $\lambda x.x(\lambda y.xy)$ is properly included in that of $\lambda x.xx$.

To sum up, we proved that intersection and union types discriminate as strictly as Lévy–Longo trees do. Using the results of [18] we can then build a discriminating context in the concurrent λ-calculus. For the classical λ-calculus the more common way of comparing terms is to consider their Böhm trees [7, Chap. 10]. So a natural question is what can be added to the pure λ-calculus in order to give it the discriminating power of Böhm trees. The paper [6] gives a type assignment system in which two λ-terms have the same types iff they have the same Böhm trees. Starting from this result, [16] proves that adding to the pure λ-calculus a nondeterministic choice operator and an adequate numeral system we obtain a language which internally discriminates two λ-terms if and only if they have different Böhm trees.

APPENDIX

In this appendix we will provide the missing proofs of the following three results.

- Conservativity (Proposition 4);
- Approximation (Theorem 16);
- Principal pair (Theorem 19).

Notice that we consider only terms in $\Lambda_{\perp,\top}$.

A.1. Proof of the Conservativity Property

We start with the following measure. The *arrow-degree* of a type in $Type_{\vee\wedge}$, i.e., the maximal number of consecutive external arrows, is inductively defined by

1. $ad(\omega) = 0$;
2. $ad(\sigma \to \tau) = ad(\tau) + 1$;
3. $ad(\sigma \wedge \tau) = \max(ad(\sigma), ad(\tau))$;
4. $ad(\sigma \vee \tau) = \min(ad(\sigma), ad(\tau))$.

We have the following simple lemma.

LEMMA 25. *If $\sigma \leqslant_{\wedge\vee} \tau$ then $ad(\tau) \leqslant ad(\sigma)$.*

Proof. The proof is by easy induction w.r.t. the definition of $\leqslant_{\wedge\vee}$.

Our first goal is to show that in fact $Type_\wedge$ has least upper bounds of finite non-empty sets. First let us observe that every element in $Type_\wedge$ is either equal to ω (after some obvious normalization) or is of the form $\bigwedge_{i \in I}(\sigma_i \to \sigma'_i)$, with $I \neq \emptyset$.

For $\sigma, \tau \in Type_\wedge$ we define a binary operation $\sigma \sqcup \tau$ by induction on $\min\{ad(\sigma), ad(\tau)\} = \#(\sigma, \tau)$. If $\#(\sigma, \tau) = 0$, then we set $\sigma \sqcup \tau = \omega$. Otherwise, let $\sigma \equiv \bigwedge_{i \in I}(\sigma_i \to \sigma'_i)$ and $\tau \equiv \bigwedge_{j \in J}(\tau_j \to \tau'_j)$. Then

$$\sigma \sqcup \tau = \bigwedge_{i \in I,\, j \in J} ((\sigma_i \wedge \tau_j) \to (\sigma'_i \sqcup \tau'_j)).$$

Since $ad(\sigma) > ad(\sigma'_i)$, for all $i \in I$, and $ad(\tau) > ad(\tau'_j)$, for all $j \in J$, it follows that $\#(\sigma, \tau) > \#(\sigma'_i, \tau'_j)$, for all $i \in I$ and $j \in J$. Hence the definition is well-founded and gives a total operation $\sqcup$. We show several properties of this operation.

Lemma 26. *For all $\sigma, \tau, \rho \in Type_\wedge$,*

1. $\sigma \leqslant_\wedge \sigma \sqcup \tau$;
2. $\sigma \sqcup \tau \sim_\wedge \tau \sqcup \sigma$;
3. $\sigma \sqcup (\tau \wedge \rho) \sim_\wedge (\sigma \sqcup \tau) \wedge (\sigma \sqcup \rho)$.

Proof. The proofs of parts 1 and 2 are by induction on $ad(\sigma)$. We prove only 1, the proof of 2 being similar. For $\sigma \sim_\wedge \omega$ or $\tau \sim_\wedge \omega$ the conclusion is obvious. So take $\sigma \equiv \bigwedge_{i \in I}(\sigma_i \to \sigma'_i)$ and $\tau \equiv \bigwedge_{j \in J}(\tau_j \to \tau'_j)$. Since $ad(\sigma'_i) < ad(\sigma)$ for $i \in I$, it follows by the induction hypothesis that for all $j \in J$ we have

$$\sigma_i \to \sigma'_i \leqslant_\wedge (\sigma_i \wedge \tau_j) \to (\sigma'_i \sqcup \tau'_j).$$

Hence $\sigma \leqslant_\wedge \sigma \sqcup \tau$.

For 3, we can assume without loss of generality that all σ, τ, and ρ are not equivalent to ω. Let $\sigma \equiv \bigwedge_{i \in I}(\sigma_i \to \sigma'_i)$, $\tau \equiv \bigwedge_{j \in J}(\tau_j \to \tau'_j)$, and $\rho \equiv \bigwedge_{k \in K}(\rho_k \to \rho'_k)$. We may assume that the sets J and K are pairwise disjoint. Let $L = J \cup K$. Then the left-hand side of 3 equals, by definition,

$$\bigwedge_{i \in I,\, l \in L} (\sigma_i \wedge \xi_l) \to (\sigma'_i \sqcup \xi'_l), \tag{4}$$

where ξ_l is τ_j, provided $j \in J$, and ρ_k otherwise, and similar notation for ξ'_l. The right-hand side of 3 is equal to

$$\bigwedge_{i \in I,\, j \in J} ((\sigma_i \wedge \tau_j) \to (\sigma'_i \sqcup \tau'_j)) \wedge \bigwedge_{i \in I,\, k \in K} ((\sigma_i \wedge \rho_k) \to (\sigma'_i \sqcup \rho'_k)),$$

which is clearly the same (up to $\sim_\wedge$) as the formula (4). ∎

Corollary 27. *For types in $Type_\wedge$ we have*

$$\sigma \wedge (\tau \sqcup \rho) \leqslant_\wedge (\sigma \wedge \tau) \sqcup (\sigma \wedge \rho).$$

Proof. By Lemma 26 we have

$$\begin{aligned}\sigma \wedge (\tau \sqcup \rho) &\sim_\wedge \sigma \wedge \sigma \wedge \sigma \wedge (\tau \sqcup \rho)\\ &\leqslant_\wedge (\sigma \sqcup \sigma) \wedge (\sigma \sqcup \rho) \wedge (\tau \sqcup \sigma) \wedge (\tau \sqcup \rho)\\ &\sim_\wedge (\sigma \wedge \tau) \sqcup (\sigma \wedge \rho). \quad ▮\end{aligned}$$

LEMMA 28. *For types in* $Type_\wedge$ *we have that if* $\sigma \leqslant_\wedge \rho$ *and* $\tau \leqslant_\wedge \rho$ *then* $\sigma \sqcup \tau \leqslant_\wedge \rho$.

Proof. We prove the lemma by induction on $\#(\sigma, \tau)$. If $\#(\sigma, \tau) = 0$, then $\sigma \sim_\wedge \omega$ or $\tau \sim_\wedge \omega$ and therefore $\rho \sim_\wedge \omega$ and we are done.

Assume $\#(\sigma, \tau) > 0$ and let $\sigma \equiv \bigwedge_{i \in I} (\sigma_i \to \sigma'_i)$ and $\tau \equiv \bigwedge_{j \in J} (\tau_j \to \tau'_j)$ with $I \neq \emptyset \neq J$. Without loss of generality we may assume that ρ is of the form $\rho_1 \to \rho_2$ with $\rho_2 \not\sim_\wedge \omega$. Hence, by Lemma 5(3) there exist sets $I_1 \subseteq I$ and $J_1 \subseteq J$ such that

$$\rho_1 \leqslant_\wedge \bigwedge_{i \in I_1} \sigma_i, \qquad \bigwedge_{i \in I_1} \sigma'_i \leqslant_\wedge \rho_2$$

and

$$\rho_1 \leqslant_\wedge \bigwedge_{j \in J_1} \tau_j, \qquad \bigwedge_{j \in J_1} \tau'_j \leqslant_\wedge \rho_2.$$

Hence,

$$\rho_1 \leqslant_\wedge \bigwedge_{i \in I_1, j \in J_1} (\sigma_i \wedge \tau_j). \tag{5}$$

By induction hypothesis we get

$$\left(\bigwedge_{i \in I_1} \sigma'_i\right) \sqcup \left(\bigwedge_{j \in J_1} \tau'_j\right) \leqslant_\wedge \rho_2.$$

By Lemma 26(3) the above formula is equivalent to

$$\bigwedge_{i \in I_1, j \in J_1} (\sigma'_i \sqcup \tau'_j) \leqslant_\wedge \rho_2. \tag{6}$$

Now, by formulas (5), (6) and rule (R4) we have

$$\sigma \sqcup \tau \sim_\wedge \bigwedge_{i \in I, j \in J} (\sigma_i \wedge \tau_j) \to (\sigma'_i \sqcup \tau'_j) \leqslant_\wedge \bigwedge_{i \in I_1, j \in J_1} (\sigma_i \wedge \tau_j) \to (\sigma'_i \sqcup \tau'_j) \leqslant_\wedge \rho_1 \to \rho_2.$$

This completes the proof. ▮

Our next goal is to show that $Type_\wedge$ is a retract of $Type_{\vee\wedge}$. From this a conservativity result will follow. Let us define a function $G\colon Type_{\vee\wedge} \to Type_\wedge$ by induction as

$$G(\omega)=\omega;$$

$$G(\sigma \to \tau)=G(\sigma) \to G(\tau);$$

$$G(\sigma \wedge \tau)=G(\sigma) \wedge G(\tau);$$

$$G(\sigma \vee \tau)=G(\sigma) \sqcup G(\tau).$$

It is clear that G restricted to $Type_{\wedge}$ is identity. Hence, by the next lemma it will follow that G is a retraction.

LEMMA 29. *G is monotone, i.e., for $\sigma, \tau \in Type_{\vee\wedge}$, if $\sigma \leqslant_{\vee\wedge} \tau$ then $G(\sigma) \leqslant_{\wedge} G(\tau)$.*

Proof. The lemma is proved by induction on the length of the derivation of $\sigma \leqslant_{\vee\wedge} \tau$. The case of the axioms (A1) through (A4), (A8), and (A9) and of the rules (R1), (R2), and (R4) is obvious since G preserves ω, $\to$, and $\wedge$. Axioms (A5) and (A6) follow from Lemma 26(2, 3). Axiom (A7) follows from Corollary 2, while rule (R3) follows from Lemma 28. ▮

Now we can conclude the proof of Proposition 4; i.e., we can show that for all $\sigma, \tau \in Type_{\wedge}$,

$$\sigma \leqslant_{\vee\wedge} \tau \quad \text{iff} \quad \sigma \leqslant_{\wedge} \tau.$$

The implication ($\Rightarrow$) follows directly from Lemma 29 and the fact that G is the identity on $Type_{\wedge}$. The opposite implication is obvious.

The reader should be warned that even though $Type_{\wedge}$ is a lattice and it is a retract of $Type_{\vee\wedge}$ as a semi-lattice, the least upper bounds in $Type_{\wedge}$ and in $Type_{\vee\wedge}$ do not coincide. For example, $(\omega_1 \to \omega_1) \vee (\omega_2 \to \omega_2)$ is strictly smaller in $Type_{\vee\wedge}$ than $(\omega_1 \to \omega_1) \sqcup (\omega_2 \to \omega_2) = \omega_2 \to \omega_1$. A similar phenomenon occurs for the set T of pure arrow types, i.e., the types of $Type_{\wedge}$ which do contain $\wedge$. It can be shown, using a definition of $\sqcup$ and $\sqcap$ in T by mutual recursion, that T is a lattice with the ordinary subtype preorder. However, lub's and glb's in T, in general, do not coincide with those in $Type_{\vee\wedge}$, nor with those in $Type_{\wedge}$.

A.2. Proof of the Approximation Theorem

The approximation theorem is proved by means of a variant of Tait's "computability" technique, in the style of [19]. We define sets of "approximable" and "computable" terms. The computable terms are defined by induction on types, and every computable term is shown to be approximable (Lemma 32(3)). Using induction on type derivations, we show that every term is computable for the appropriate type (Lemma 34).

For $M \in \Lambda_{\perp,\top}$, we define two predicates $\mathsf{App}(\Gamma, \sigma, M)$ and $\mathsf{Comp}(\Gamma, \sigma, M)$ as

1. $\mathsf{App}(\Gamma, \sigma, M) \Leftrightarrow \exists A \in \mathscr{A}(M).\Gamma \vdash A : \sigma$;
2. (a) $\mathsf{Comp}(\Gamma, \omega, M)$ is always true;

   (b) If $\tau \sim \omega$ then $\mathsf{Comp}(\Gamma, \sigma \to \tau, M) \Leftrightarrow \mathsf{App}(\Gamma, \sigma \to \tau, M)$;

(c) If $\tau \not\sim \omega$ then $\mathsf{Comp}(\Gamma, \sigma \to \tau, M)$ holds iff $\mathsf{Comp}(\Gamma', \sigma, N)$ implies $\mathsf{Comp}(\Gamma \uplus \Gamma', \tau, MN)$, for each Γ' and N;

(d) $\mathsf{Comp}(\Gamma, \sigma \wedge \tau, M) \Leftrightarrow \mathsf{Comp}(\Gamma, \sigma, M)$ and $\mathsf{Comp}(\Gamma, \tau, M)$;

(e) $\mathsf{Comp}(\Gamma, \sigma \vee \tau, M)$ holds iff $\mathsf{Comp}(\Gamma, \sigma, M)$ or $\mathsf{Comp}(\Gamma, \tau, M)$ or $[M =_{\beta}^{\perp, \top} x\vec{N}$ and $\mathsf{App}(\Gamma, \sigma \vee \tau, M)]$.

We can easily prove by induction on types that Comp is invariant under beta conversion and that it is always true for terms of the shape $\top\vec{M}$.

Lemma 30. 1. *$\mathsf{Comp}(\Gamma, \sigma, M)$ and $M =_\beta N$ imply $\mathsf{Comp}(\Gamma, \sigma, N)$.*

2. *$\mathsf{Comp}(\Gamma, \sigma, \top\vec{M})$ is true for all $\Gamma, \sigma, \vec{M}$.*

Proof. For part (2), if $\sigma \equiv \sigma_1 \to \sigma_2$ and $\sigma_2 \sim \omega$ we use Lemma 11(1). If $\sigma \equiv \sigma_1 \to \sigma_2$ and $\sigma_2 \not\sim \omega$, we have by induction $\mathsf{Comp}(\Gamma \uplus \Gamma', \sigma_2, \top\vec{M}N)$ for all Γ', N, so we get $\mathsf{Comp}(\Gamma, \sigma, \top\vec{M})$. ∎

Below we show some properties of type assignments deducible for approximate normal forms.

Lemma 31. 1. *If $\Gamma \vdash A : \sigma$ and $A \leqslant A'$ then $\Gamma \vdash A' : \sigma$.*

2. *Let $z \notin FV(M)$, $\tau \not\sim \omega$ and $\Gamma' = \Gamma, z : \sigma$. Then $\mathsf{App}(\Gamma', \tau, Mz)$ implies $\mathsf{App}(\Gamma, \sigma \to \tau, M)$.*

Proof. Part (1) follows by induction with respect to the definition of $\leqslant$ using Theorem 6. For part (2), let $A \in \mathscr{A}(Mz)$ be such that $\Gamma' \vdash A : \tau$. We show that there exists $\hat{A} \in \mathscr{A}(M)$ such that $\Gamma \vdash \hat{A} : \sigma \to \tau$. If M is beta equal to an abstraction, then $\lambda z.Mz =_\beta M$ and we can choose $\hat{A} \equiv \lambda z.A$. Otherwise, either $A \equiv \top$, and we can take $\hat{A} \equiv \top$, or A must be of the form $xA_1 \cdots A_n Z$. In the latter case we can choose $\hat{A} \equiv xA_1 \cdots A_n$. Indeed, since $\Gamma' \vdash xA_1 \cdots A_n Z : \tau$, we have by Theorem 6(4) that $\Gamma' \vdash xA_1 \cdots A_n : \rho \to \tau$, for some ρ with $\Gamma' \vdash Z : \rho$. As an approximant of z, the term Z is either z or $\perp$, and in both cases it must be $\sigma \leqslant \rho$. Thus $\Gamma \vdash xA_1 \cdots A_n : \sigma \to \tau$, as desired.

We can now show that computability implies approximability.

Lemma 32. *For all Γ, σ, $\vec{L}$, and M:*

1. *If $\mathsf{App}(\Gamma, \sigma, x\vec{L})$ then $\mathsf{Comp}(\Gamma, \sigma, x\vec{L})$;*
2. *If $\mathsf{Comp}(\Gamma, \sigma, M)$ then $\mathsf{App}(\Gamma, \sigma, M)$.*

Proof. Conditions (1) and (2) are proved by a simultaneous induction on σ.

1. Most cases are easy, so we only consider the case $\sigma \equiv \sigma_1 \to \sigma_2$, with $\sigma_2 \not\sim \omega$. Let $\mathsf{App}(\Gamma, \sigma, x\vec{L})$. This means that there is an approximant $A \in \mathscr{A}(x\vec{L})$ with $\Gamma \vdash A : \sigma$. By inspecting the definition of approximants, we easily find out that A must be of the form $x\vec{A}$, where $\vec{A}$ is a vector of approximants of $\vec{L}$.

We show that $\mathsf{Comp}(\Gamma, \sigma, x\vec{L})$. Assume $\mathsf{Comp}(\Gamma', \sigma_1, N)$. By the induction hypothesis, part 2, we have $\mathsf{App}(\Gamma', \sigma_1, N)$, that is, there is an approximant $B \in \mathscr{A}(N)$ of type σ_1 in the context Γ'. The term $AB \equiv x\vec{A}B$ is an approximant of

$x\vec{L}N$, and we have $\Gamma \uplus \Gamma' \vdash AB : \sigma_2$. Thus $\mathsf{Comp}(\Gamma \uplus \Gamma', \sigma_2, x\vec{L}N)$ follows from part 1 of the induction hypothesis.

2. Assume first that $\sigma \equiv \sigma_1 \to \sigma_2$ and $\sigma_2 \not\sim \omega$. Let $\Gamma' = \Gamma, z : \sigma_1$, where $z \notin FV(M)$, and suppose $\mathsf{Comp}(\Gamma, \sigma_1 \to \sigma_2, M)$. Then we have

$$\begin{array}{lll} & \mathsf{Comp}(\{z : \sigma_1\}, \sigma_1, z) & \text{by part 1} \\ \Rightarrow & \mathsf{Comp}(\Gamma', \sigma_2, Mz) & \text{by definition} \\ \Rightarrow & \mathsf{App}(\Gamma', \sigma_2, Mz) & \text{by induction} \\ \Rightarrow & \mathsf{App}(\Gamma, \sigma_1 \to \sigma_2, M) & \text{by Lemma 31(2).} \end{array}$$

Now let $\sigma \equiv \sigma_1 \wedge \sigma_2$, and assume $\mathsf{Comp}(\Gamma, \sigma_1 \wedge \sigma_2, M)$. Then we have $\mathsf{Comp}(\Gamma, \sigma_1, M)$ and $\mathsf{Comp}(\Gamma, \sigma_2, M)$. By part 2 of the induction hypothesis, there is an approximant for M of type σ_1 and another of type σ_2. Using Lemmas 12 and Lemma 31(1), we obtain one approximant that has both these types. The remaining cases are easy. ∎

Let us notice that without the last clause of part (e) in the definition of Comp, Lemma 32 would be false. Consider, for example, that $x : \sigma \vee \tau \vdash x : \sigma \vee \tau$; then we get $\mathsf{App}(\{x : \sigma \vee \tau\}, \sigma \vee \tau, x)$ and this implies $\mathsf{Comp}(\{x : \sigma \vee \tau\}, \sigma \vee \tau, x)$ by Lemma 32(1). But we can derive neither $x : \sigma \vee \tau \vdash x : \sigma$ nor $x : \sigma \vee \tau \vdash x : \tau$, and therefore, $\mathsf{App}(\{x : \sigma \vee \tau\}, \sigma, x)$ and $\mathsf{App}(\{x : \sigma \vee \tau\}, \tau, x)$ are both false. We conclude that also $\mathsf{Comp}(\{x : \sigma \vee \tau\}, \sigma, x)$ and $\mathsf{Comp}(\{x : \sigma \vee \tau\}, \tau, x)$ are both false by Lemma 32(2).

The following lemma states that computability agrees with the typing rule $(\leqslant)$.

LEMMA 33. *If $\sigma \leqslant \tau$ and $\mathsf{Comp}(\Gamma, \sigma, M)$ then $\mathsf{Comp}(\Gamma, \tau, M)$.*

Proof. The proof is by induction with respect to the definition of $\leqslant$. For example, for axiom (A7), assume that $\mathsf{Comp}(\Gamma, \sigma \wedge (\tau \vee \rho), M)$. If $M =_\beta^{\perp,\top} x\vec{N}$ for some x and $\vec{N}$, then by Lemma 32(2) we have $\mathsf{App}(\Gamma, \sigma \wedge (\tau \vee \rho), M)$. Thus $\mathsf{App}(\Gamma, (\sigma \wedge \tau) \vee (\sigma \wedge \rho), M)$, and this implies $\mathsf{Comp}(\Gamma, (\sigma \wedge \tau) \vee (\sigma \wedge \rho), M)$ by Lemma 32(1).

Otherwise, we have $\mathsf{Comp}(\Gamma, \sigma, M)$ and $\mathsf{Comp}(\Gamma, \tau \vee \rho, M)$. But now we can assume that M is not $=_\beta^{\perp,\top}$ equal to any $x\vec{N}$, and thus either $\mathsf{Comp}(\Gamma, \tau, M)$ or $\mathsf{Comp}(\Gamma, \rho, M)$ by definition. In a routine way we obtain $\mathsf{Comp}(\Gamma, (\sigma \wedge \tau) \vee (\sigma \wedge \rho), M)$.

Consider the case of rule (R3). Suppose we derive $\sigma \vee \tau \leqslant \rho$ from $\sigma \leqslant \rho$ and $\tau \leqslant \rho$. Assume $\mathsf{Comp}(\Gamma, \sigma \vee \tau, M)$. To show $\mathsf{Comp}(\Gamma, \rho, M)$, we consider three cases, according to the definition of $\mathsf{Comp}(\Gamma, \sigma \vee \tau, M)$. For the first two cases, we use the induction hypothesis, and for the third one, we apply Lemma 32. The remaining cases are easy. ∎

LEMMA 34. *Let $\Gamma = \{x_1 : \sigma_1, ..., x_n : \sigma_n\}$ and let $\Gamma \vdash M : \tau$. Assume that the condition $\mathsf{Comp}(\Gamma_i, \sigma_i, N_i)$ holds for each $i \leqslant n$ and take $\Gamma' = \biguplus_{i=1}^n \Gamma_i$. Then*

$$\mathsf{Comp}(\Gamma', \tau, M[N_1/x_1, ..., N_n/x_n]).$$

Proof. Induction on the derivation of $\Gamma \vdash M : \tau$. Cases (Ax) and (ω) are immediate. Case ($\top$) follows from Lemma 30(2). Cases ($\to$E) and ($\wedge$I) follow from the induction hypothesis. Case ($\leqslant$) follows from the induction hypothesis and Lemma 33.

For the proof in case ($\to$I), let $M \equiv \lambda y.P$ and $\tau \equiv \tau_1 \to \tau_2$ and suppose that $\Gamma, y : \tau_1 \vdash P : \tau_2$ has been derived. If $\tau_2 \not\sim \omega$ and $\mathsf{Comp}(\Gamma'', \tau_1, Q)$ then from the induction hypothesis

$$\mathsf{Comp}(\Gamma' \uplus \Gamma'', \tau_2, P[Q/y, \vec{N}/\vec{x}]).$$

There is no loss of generality in assuming that $y \notin FV(\vec{N})$, so that

$$P[Q/y, \vec{N}/\vec{x}] \equiv P[\vec{N}/\vec{x}][Q/y], \qquad (\lambda y.P[\vec{N}/\vec{x}])\, Q \equiv ((\lambda y.P)[\vec{N}/\vec{x}])\, Q.$$

By the invariance of computability under beta conversion (Lemma 30(1)), it follows that $\mathsf{Comp}(\Gamma' \uplus \Gamma'', \tau_2, ((\lambda y.P)[\vec{N}/\vec{x}])\, Q)$, and hence,

$$\mathsf{Comp}(\Gamma', \tau_1 \to \tau_2, (\lambda y.P)[\vec{N}/\vec{x}]),$$

since the computable term Q was arbitrary.

If $\tau_2 \sim \omega$, then $\lambda y.\bot$ is an approximant of $\lambda y.P[\vec{N}/\vec{x}]$ of type $\tau_1 \to \omega$, and thus, $\mathsf{Comp}(\Gamma', \tau_1 \to \omega, M[\vec{N}/\vec{x}])$ holds. ∎

Now we can complete the proof of the approximation theorem (Theorem 16); i.e. we show

$$\Gamma \vdash M : \sigma \quad \text{iff} \quad \exists A \in \mathscr{A}(M) \quad \text{such that} \quad \Gamma \vdash A : \sigma. \tag{7}$$

($\Rightarrow$) If $\top \in \mathscr{A}(M)$ it is trivial. Otherwise, since $\mathsf{App}(\{x : \tau\}, \tau, x)$ holds for any variable x and type τ, then by Lemma 32(1), we have $\mathsf{Comp}(\{x : \tau\}, \tau, x)$. Thus we can apply Lemma 34 for the identity substitution, to obtain $\mathsf{Comp}(\Gamma, \sigma, M)$. The hypothesis follows from Lemma 32(2).

($\Leftarrow$) The proof is by induction with respect to $\Gamma \vdash A : \sigma$. Cases ($\wedge$I), ($\leqslant$), and (ω) are obvious. Case ($\to$I) follows from Lemma 11(2), and cases ($\to$E) and (Ax) from Lemma 11(3). The most interesting case is when the last rule is ($\top$). Now if $M =_{\beta}^{\bot,\top} \top$ it follows by subject conversion (Theorem 8). Otherwise, $\top \in \mathscr{A}(M)$ implies that M reduces to $\lambda x_1 \cdots x_n . Q_n$ for suitable Q_n and all n. We have $\Gamma \vdash \lambda x_1 \cdots x_n . Q_n : \omega_n$ for all n, so we conclude $\Gamma \vdash M : \sigma$ for all Γ, σ, by Proposition 7 and Theorem 8. This completes the proof of (7). ∎

A.3. Proof of the Principal Pair Theorem

We start with the following result which collects some of the properties of our definition of a principal pair.

LEMMA 35. *Let* $\langle \Gamma; \sigma \rangle \in \Pi$. *Then*:

1. *Each type variable* $t \in V$ *occurs twice in* Γ, σ, *and at most once as a tail variable of a type scheme in* Γ.

2. *Each type variable $t' \in V'$ occurs at most once in Γ, σ, and never as a tail variable of a type scheme in Γ.*

3. *Each type scheme σ which occurs in Γ as a predicate is of the form $\sigma_1 \wedge \sigma_2 \wedge \cdots \wedge \sigma_k$, where $k \geqslant 1$, and each σ_i equals some $\rho_1^i \to \cdots \to \rho_{n_i}^i \to t_i$ with $n_i \geqslant 0$.*

4. *If $x: \tau_1 \to \cdots \to \tau_n \to \mu \in \Gamma$, then for all $1 \leqslant i \leqslant n$ there is $\Gamma_i \Subset \Gamma$ such that $\langle \Gamma_i; \tau_i \rangle \in \Pi$.*

5. *If $\sigma \equiv \mu \to \tau$ and $\mu \not\sim \omega$, then $\langle \Gamma, x: \mu; \tau \rangle \in \Pi$ for every x.*

6. *If $\sigma \equiv \omega \to \tau$ then $\langle \Gamma\ ; \tau \rangle \in \Pi$.*

Proof. Easy induction. ▌

A basis scheme Γ is *semi-principal* iff it satisfies conditions (1), (2), (3), and (4) of the above Lemma.

We define a "labeled partial order" $\Delta \sqsubseteq_\sigma \Gamma$, where $\langle \Delta; \sigma \rangle$ must be a principal pair and Γ is semi-principal. The definition of $\sqsubseteq_\sigma$ is by induction. As a base step we take $\Delta \sqsubseteq_\sigma \Delta$. Further, if $\Delta \sqsubseteq_\sigma \Gamma$ holds, then:

1. If $\Delta = (\biguplus_{i \leqslant n} \Delta_i) \uplus \{x: \tau_1 \to \cdots \to \tau_m \to t\}$ and $\sigma \equiv t$, then $\Delta_i \sqsubseteq_{\tau_i} \Gamma$, for each i;

2. If $\sigma \equiv \rho_1 \to \rho_2$, and $\rho_1 \not\sim \omega$, then $\Delta, x: \rho_1 \sqsubseteq_{\rho_2} \Gamma$, $x: \rho_1$, where x is a fresh variable;

3. If $\sigma \equiv \omega \to \rho_2$, then $\Delta \sqsubseteq_{\rho_2} \Gamma$.

One can easily see that $\Delta \sqsubseteq_\sigma \Gamma$ implies $\Delta \Subset \Gamma$. Recall that the symbol $\Subset$ denotes the inclusion between bases defined above Theorem 6.

Before we start proving the Principal Pair Theorem we have to show how to build types in $Type_{\vee\wedge}$ which essentially behave like different type variables with respect to $\leqslant_{\vee\wedge}$.

First we define a mapping F: $Type_{\vee\wedge} \times \mathsf{N} \to Type_{\vee\wedge}$, as

1. $F(\sigma, 0) = \sigma$;
2. $F(\omega, n+1) = \omega$;
3. $F(\sigma \to \tau, n+1) = F(\tau, n)$;
4. $F(\sigma \wedge \tau, n+1) = F(\sigma, n+1) \wedge F(\tau, n+1)$;
5. $F(\sigma \vee \tau, n+1) = F(\sigma, n+1) \vee F(\tau, n+1)$.

Let us fix numbers n and m with $1 \leqslant n \leqslant m$, and let $h = m + 2n + 1$. Let us recall the definition of types α_i from Section 6,

$$\alpha_i = \beta_i \vee \beta_{n+i},$$

where β_j for $j \leqslant 2n$ is defined as

$$\beta_j = \omega^{m+j-1} \to \omega_1 \to \omega^{h-j} \to \omega_1.$$

We have the following lemma.

Lemma 36. *For each $I \subseteq \{1, ..., n\}$, each $p_i, q_i \leqslant m$, and $j \leqslant n$:*

$$\bigwedge_{i \in I} (\omega^{p_i} \to F(\alpha_i, q_i)) \leqslant_{\vee\wedge} \alpha_j \quad \textit{iff} \quad j \in I \qquad \textit{and} \qquad p_j = q_j = 0.$$

Proof. If $\bigwedge_{i \in I} (\omega^{p_i} \to F(\alpha_i, q_i)) \leqslant_{\vee\wedge} \alpha_j$, then from Lemma 5(2) we have $\omega^{p_i} \to F(\alpha_i, q_i) \leqslant_{\vee\wedge} \alpha_j$ for some $i \in I$.

By definition $ad(\alpha_i) = ad(\beta_i) = ad(\beta_{n+i}) = h + m + 1$ for all i. On the other hand, $ad(\omega^{p_i} \to F(\alpha_i, q_i)) = p_i + h + m + 1 - q_i$. An immediate consequence of Lemma 25 is that $p_i \geqslant q_i$.

Case 1. $p_i = 0$. Then also $q_i = 0$, and we actually have $\alpha_i = F(\alpha_i, 0) \leqslant_{\vee\wedge} \alpha_j$. Lemma 5 gives us $G(\alpha_i) \leqslant_{\vee\wedge} G(\alpha_j)$, i.e.,

$$\omega^{m+i-1} \to \omega_1 \to \omega^{n-1} \to \omega_1 \to \omega^{h-n-i} \to \omega_1$$
$$\leqslant_{\wedge} \omega^{m+j-1} \to \omega_1 \to \omega^{n-1} \to \omega_1 \to \omega^{h-n-j} \to \omega_1.$$

Thus, the positions of ω_1 must match, and we have $i = j$.

Case 2: $p_i > 0$. By Lemma 5(1), the condition $\omega^{p_i} \to F(\alpha_i, q_i) \leqslant_{\vee\wedge} \alpha_j$ implies either $\omega^{p_i} \to F(\alpha_i, q_i) \leqslant_{\vee\wedge} \beta_j$ or $\omega^{p_i} \to F(\alpha_i, q_i) \leqslant_{\vee\wedge} \beta_{n+j}$. Assume the first possibility. Then, from Lemma 29, we obtain

$$G(\omega^{p_i} \to F(\alpha_i, q_i)) \leqslant_{\wedge} G(\beta_j) = \beta_j.$$

That is, we have

$$\omega^{p_i + m - q_i + i - 1} \to \omega_1 \to \omega^{n-1} \to \omega_1 \to \omega^{h-n-i} \to \omega_1$$
$$\leqslant_{\wedge} \omega^{m+j-1} \to \omega_1 \to \omega^{h-j} \to \omega_1.$$

The right-hand side of the above inequality has only one occurrence of ω_1 and the left-hand side has two. To satisfy the inequality, the first ω_1 must match the occurrence at the right-hand side, and the second one must occur not earlier than the $(h+m+2)$nd argument (to be "covered" by the target ω). It follows that $h + m + 2 \leqslant p_i + m - q_i + i + n - 1 \leqslant p_i + m + i + n \leqslant 2m + 2n$, which is a contradiction. Similarly we get that the second possibility is contradictory. ∎

We need one more technical lemma for the proof of Theorem 19. Let us recall that the arrow-degree $ad(-)$ is defined in Section A.1 for the proof of the conservativity property.

Lemma 37. *If $\Gamma \vdash \lambda x_1 \cdots x_n . y M_1 \cdots M_m : \sigma$ and $\Gamma(y) = \tau$, then $ad(\sigma) \leqslant ad(\tau) + n - m$.*

Proof. The proof is by induction w.r.t. the length of derivations. Case (Ax) follows from Theorem 6(1) and Lemma 25. Case $(\leqslant)$ follows from Lemma 25. ∎

Now we are ready to start proving Theorem 19. Recall that we have to show that for $M \in \Lambda_{\perp,\top}$, $A \in \mathscr{A}$, and $pp(A) = \langle \Gamma; \sigma \rangle$, if we take $n = v(A)$ and $m = \kappa(M, A)$, then $s_{m,n}(\Gamma) \vdash M : s_{m,n}(\sigma)$ implies $A \in \mathscr{A}^*(M)$. Definitions of measures υ and κ are given in Section 6.

We show a stronger claim, from which our theorem follows:

Let $M \in \Lambda_{\perp,\top}$, $A \in \mathscr{A}$, $\langle \Delta; \sigma \rangle = pp(A)$, and $\Delta \sqsubseteq_\sigma \Gamma$. Let n be the number of type variables from V (i.e., nonprimed versions) which occur in $\langle \Gamma; \sigma \rangle$, and let $m \geqslant \{a + \text{degree}(M, h), a + \text{order}(M, h), n\}$, where a is the number of arrows which occur in $\langle \Gamma; \sigma \rangle$ and h is the height of $LL(A)$. Then $s_{m,n}(\Gamma) \vdash M : s_{m,n}(\sigma)$ implies $A \in \mathscr{A}^*(M)$.

The case $LL(M) = \cdot \top$ is trivial, since we get $\mathscr{A}(M) = \mathscr{A}$. In the other cases the proof is by induction with respect to $\langle \Delta; \sigma \rangle$. Whenever $\sigma \not\sim \omega$, by Theorem 18 the term M must have a weak head normal form:

Case 1. If $\langle \Delta; \sigma \rangle = \langle \emptyset; \omega \rangle$ then $A \equiv \perp$.

Case 2. If $\langle \Delta; \sigma \rangle = \langle \emptyset; t' \rangle$ then $s_{m,n}(t') = \omega_{3m+2n+3}$. Let $M \equiv \lambda x_1 \cdots x_{d'}.M'$ where M' does not have a weak head normal form. Then a type deducible for M is always $\geqslant \omega_{d'}$. Now $\omega_{d'} \leqslant \omega_{3m+2n+3}$ is impossible by Lemma 25, since $ad(\omega_{d'}) = d' < ad(\omega_{3m+2n+3}) = 3m + 2n + 3$.

Assume $M \equiv \lambda x_1 \cdots x_{d'}.y\vec{N}$. By definition of semi-principal basis and of $s_{m,n}(\ )$ a type which occurs in $s_{m,n}(\Gamma)$ is an intersection of arrow types of the shape $\tau_1 \to \cdots \to \tau_{a'} \to \alpha_i$ for some $a' \leqslant a$ and $i \leqslant n$. Notice that $ad(\alpha_i) = 2m + 2n + 2$, then a being the total number of arrows in $\langle \Gamma; t' \rangle$ we have $ad(\rho) \leqslant a + 2m + 2n + 2$ for all ρ which occur in $s_{m,n}(\Gamma)$. By Lemma 37 we get that $s_{m,n}(\Gamma) \vdash M : \tau$ implies $ad(\tau) \leqslant d' + a + 2m + 2n + 2 \leqslant 3m + 2n + 2$. Therefore, we cannot have $\tau \leqslant \omega_{3m+2n+3}$, since this would contradict Lemma 25. So we conclude that either $M =_\beta^{\perp,\top} \top$ or $M =_\beta^{\perp,\top} \lambda\vec{x}.Q$ for arbitrarily long $\vec{x}$. In both cases $\mathscr{A}^*(M) = \mathscr{A}$.

Case 3. Let $\sigma \equiv \sigma_1 \to \sigma_2$ in $\langle \Delta; \sigma \rangle$. Then $A \equiv \lambda x.B$, and $\langle \Delta, x : \sigma_1; \sigma_2 \rangle = pp(B)$ if $\sigma_1 \not\sim \omega$; $\langle \Delta; \sigma_2 \rangle = pp(B)$ otherwise. If $M =_\beta^{\perp,\top} \lambda x.M'$, then $s_{m,n}(\Gamma) \vdash M : s_{m,n}(\sigma)$ implies $s_{m,n}(\Gamma, x : \sigma_1) \vdash M' : s_{m,n}(\sigma_2)$ and we apply the induction hypothesis. If $M =_\beta^{\perp,\top} y\vec{N}$, then $s_{m,n}(\Gamma) \vdash M : s_{m,n}(\sigma)$ implies $s_{m,n}(\Gamma) \vdash \lambda z.Mz : s_{m,n}(\sigma)$, where $z \notin FV(M)$. Notice that $\text{degree}(\lambda z.Mz, h) \leqslant d + 1$ and $\text{order}(\lambda z.Mz, h) \leqslant o + 1$, but the number of arrows in $\langle \Gamma, x : \sigma_1; \sigma_2 \rangle$ is $a - 1$, so the hypothesis on m is still valid. Therefore, from the previous case $A \in \mathscr{A}^*(\lambda z.Mz)$ and by Lemma 14(2), we get $A \in \mathscr{A}^*(M)$.

Case 4. Let $\langle \Delta; \sigma \rangle = \langle (\biguplus_{j \leqslant k} \Delta_j) \uplus \{x : \sigma_1 \to \cdots \to \sigma_k \to t\}; t \rangle$. Then $A \equiv xA_1 \cdots A_k$ and $s_{m,n}(t) = \alpha_l$ for some $l \leqslant n$.

Let $M \equiv \lambda x_1 \cdots x_{d'}.M'$ where M' does not have a weak head normal form. Then a type deducible for M is always $\geqslant \omega_{d'}$. Now $\omega_{d'} \leqslant \alpha_l$ is impossible by Lemma 25, since $ad(\omega_{d'}) = d' < ad(\alpha_l) = 2m + 2n + 2$.

Assume $M =_\beta^{\perp,\top} \lambda y_1 \cdots y_{d'}.z\vec{N}$. By Theorem 6(1, 3, 4) and the assumptions that a type for M will have either the shape

$$\rho_1 \to \cdots \to \rho_{d'} \to \tau_1 \to \cdots \to \tau_{a'} \to \alpha_i$$

or the shape

$$\rho_1 \to \cdots \to \rho_{d'} \to F(\alpha_i, o')$$

for some $a'+d', o' \leqslant m$, and $i \leqslant n$, or it is an intersection of types of the above shapes. Clearly,

$$\omega^{a'+d'} \to \alpha_i \leqslant \rho_1 \to \cdots \to \rho_{d'} \to \tau_1 \to \cdots \to \tau_{a'} \to \alpha_i$$

and

$$\omega^{d'} \to F(\alpha_i, o') \leqslant \rho_1 \to \cdots \to \rho_{d'} \to F(\alpha_i, o').$$

Therefore, we need to satisfy

$$\bigwedge_{i \in I} (\omega^{p_i} \to F(\alpha_i, q_i)) \leqslant \alpha_l$$

for some $I \subseteq \{1, ..., n\}$ and $p_i, q_i \leqslant m$. Lemma 36 gives $a' = d' = o' = 0$ and $i = l$. By construction, the only type in $s_{m,n}(\Gamma)$ which ends by α_l is $s_{m,n}(\sigma_1 \to \cdots \to \sigma_k \to t)$, so we get $M =_{\beta}^{\perp, \top} xN_1 \cdots N_k$ and $s_{m,n}(\Gamma) \vdash N_i : s_{m,n}(\sigma_i)$ for $1 \leqslant i \leqslant k$ by Theorem 6. This implies by induction that $A_i \in \mathscr{A}^*(N_i)$ for $1 \leqslant i \leqslant k$, so we conclude $A \in \mathscr{A}^*(M)$ by Lemma 14(3). This completes the proof of the claim and now Theorem 19 follows.

ACKNOWLEDGMENTS

The final form of this paper differs deeply from the initial one. This is essentially due to the constructive criticism of the referees of the LICS paper [20] and of the journal version. Therefore we feel strongly indebted to them. During the elaboration of an intermediate version of this paper Mariangiola Dezani-Ciancaglini was visiting the Tokyo Institute of Technology. She thanks her host Masako Takahashi–Horai and the whole Department of Mathematical and Computing Sciences for the ideal working conditions they provided.

Received September 15, 1997; final manuscript received October 5, 1998

REFERENCES

1. Abramsky, S. (1991), Domain theory in logical form, *Ann. Pure Appl. Logic* **51**, 1–77.
2. Abramsky, S., and Ong, C.-H. L. (1993), Full abstraction in the lazy lambda calculus, *Inform. and Comput.* **105**, 159–267.
3. Alessi, F., Dezani-Ciancaglini, M., and de'Liguoro, U., (1997), A convex powerdomain over lattices: Its logic and λ-calculus, *Fund. Inf.* **32**(3–4), 193–250.
4. Aoun, A., Barbanera, F., Dezani-Ciancaglini, M., and Mirasyedioğlu, S. (1997), Principal typing for parallel and non-deterministic lambda calculus, *CIT J.* **57**(2), 129–138.
5. Barbanera, F., Dezani-Ciancaglini, M., and de'Liguoro, U. (1995), Intersection and union types: Syntax and semantics, *Inform. and Comput.* **119**, 202–230.

6. Barbanera, F., Dezani-Ciancaglini, M., and de Vries, F. J. (1998), Types for trees, *in* "Proc. PROCOMET'98" (D. Gries and W. P. de Rover, Eds.), pp. 11–29, Chapman & Hall, London.
7. Barendregt, H. P. (1984), "The Lambda Calculus: Its Syntax and Semantics," 2nd ed., North-Holland, Amsterdam.
8. Barendregt, H. P., Coppo, M., and Dezani-Ciancaglini, M. (1983), A filter lambda model and the completeness of type assignment, *J. Symbolic Logic* **48**, 931–940.
9. Boreale, M., and De Nicola, R. (1995), Testing equivalence for mobile processes, *Inform. and Comput.* **120**, 279–303.
10. Boudol, G. (1994), A lambda calculus for (strict) parallel functions, *Inform. and Comput.* **108**, 51–127.
11. Boudol, G., and Laneve, C. (1995), "λ-Calculus, Multiplicities and the π-Calculus," Rapport de Recherche 2581, INRIA, Sophia-Antipolis.
12. Boudol, G., and Laneve, C. (1996), The discriminating power of multiplicities in the λ-calculus, *Inform. and Comput.* **126**(1), 83–102.
13. Cardone, F., and Coppo, M. (1990), Two extensions of Curry's type inference system, *in* "Logic and Computer Science" (P. Odifreddi, Ed.), pp. 19–75, Academic Press, New York.
14. Coppo, M., Dezani-Ciancaglini, M., and Zacchi, M. (1987), Type theories, normal forms, and D_∞-lambda-models, *Inform. and Comput.* **72**, 85–116.
15. Damiani, F., Dezani-Ciancaglini, M., and Giannini, P., A filter model for mobile processes, *Math. Struc. Comput. Sci.*, to appear.
16. Dezani-Ciancaglini, M., Intrigila, B., and Venturini-Zilli, M. (1998), Böhm's theorem for Böhm trees, *in* "Proc. ICTCS'98" (P. Degano, U. Vaccaro, and G. Pirillo, Eds.), World Scientific, Oxford.
17. Dezani-Ciancaglini, M., de'Liguoro, U., and Piperno, A. (1996), Filter models for conjunctive-disjunctive λ-calculi, *Theoret. Comput. Sci.* **170**(1–2), 83–128.
18. Dezani-Ciancaglini, M., de'Liguoro, U., and Piperno, A. (1998), A filter model for concurrent λ-calculi, *SIAM J. of Comput.* **27**(5), 1376–1419.
19. Dezani-Ciancaglini, M., and Margaria, I. (1986), A characterization of F-complete type assignments, *Theoret. Comput. Sci.* **45**, 121–157.
20. Dezani-Ciancaglini, M., Tiuryn, J., and Urzyczyn, P. (1997), Discrimination by parallel observers, *in* "Proc. 12th IEEE Symp. LICS'97," pp. 396–407, IEEE Comput. Soc. Press, Los Alamitos, CA.
21. Hennessy, M. C. B. (1994), A fully abstract denotational model for higher-order processes, *Inform. and Comput.* **112**, 55–95.
22. Groote, J. F., and Vaandrager, F. W. (1992), Structured operational semantics and bisimulation as a congruence, *Inform. and Comput.* **100**, 202–260.
23. Lévy, J. J. (1976), An algebraic interpretation of the λ-β-**K**-calculus and an application of the labelled λ-calculus, *Theoret. Comput. Sci.* **2**(1), 97–114.
24. Longo, G. (1983), Set theoretical models of lambda calculus: Theory, expansions and isomorphisms, *Ann. Pure Appl. Logic* **24**, 153–188.
25. Milner, R., Parrow, J., and Walker, D. (1992), A calculus of mobile processes, I, II, *Inform. and Comput.* **100**(1), 1–77.
26. Milner, R., Parrow, J., and Walker, D. (1993), Modal logics for mobile processes, *Theoret. Comput. Sci.* **114**, 149–171.
27. Ong, C.-H. L. (1988), "The Lazy λ-Calculus: An Investigation into the Foundations of Functional Programming," Ph.D. thesis, University of London. [Also Prize Fellowship Dissertation, Trinity College, Cambridge.]
28. Ong, C.-H. L. (1992), Lazy lambda calculus: Theories, models and local structure characterization, *in* "Proc. ICALP'92" (W. Kuich, Ed.), pp. 487–498, Lecture Notes in Computer Science, Vol. 623, Springer-Verlag, Berlin.
29. Ong, C.-H. L. (1992), Concurrent lambda calculus and a general precongruence theorem for applicative bisimulations, Draft, Cambridge University.

30. Ong, C.-H. L. (1993), Non-determinism in a functional setting, *in* "Proc. 8th IEEE Symp. LICS'93," pp. 275–286, IEEE Comput. Soc. Press, Los Alamitos, CA.

31. Plotkin, G. (1993), Set-theoretical and other elementary models of the λ-calculus, *Theoret. Comput. Sci.* **121**, 351–409.

32. Sangiorgi, D. (1993), A theory of bisimulation for the π-calculus, *in* "Proc. CONCUR'93" (E. Best, Ed.), pp. 121–136, Lecture Notes in Computer Science, Vol. 715, Springer-Verlag, Berlin.

33. Sangiorgi, D. (1994), The lazy λ-calculus in a concurrency scenario, *Inform. and Comput.* **111**(1), 120–153.

34. Sangiorgi, D. (1995), "The Lazy Functions and Mobile Processes," Rapport de Recherche 2515, INRIA, Sophia-Antipolis.

35. Scott, D. (1982), Domains for denotational semantics, *in* "Proc. ICALP'82" (M. Nielsen and E. M. Schmidt, Eds.), pp. 577–613, Lecture Notes in Computer Science, Vol. 140, Springer-Verlag, Berlin.