The Weak Normalization of the Simply Typed λs_e-calculus

ARIEL ARBISER, *Department of Computer Science, University of Buenos Aires, Pabellón I - Ciudad Universitaria (1428) Buenos Aires, Argentina. E-mail: arbiser@dc.uba.ar*

FAIROUZ KAMAREDDINE, *School of Mathematical and Computer Sciences, Heriot-Watt University, Riccarton, Edinburgh EH14 4AS, Scotland. E-mail: fairouz@macs.hw.ac.uk*

ALEJANDRO RIOS, *Department of Computer Science, University of Buenos Aires, Pabellón I - Ciudad Universitaria (1428) Buenos Aires, Argentina. E-mail: rios@dc.uba.ar*

Abstract

In this paper, we show the weak normalization (WN) of the simply-typed λs_e-calculus with open terms where abstractions are decorated with types, and metavariables, de Bruijn indices and updating operators are decorated with environments. We show a proof of WN using the $\lambda\omega_e$-calculus, a calculus isomorphic to $\lambda\overrightarrow{s}_e$. This proof is strongly influenced by Goubault-Larrecq's proof of WN for the $\lambda\sigma$-calculus but with subtle differences which show that the two styles require different attention. Furthermore, we give a new calculus $\lambda\omega'_e$ which works like λs_e but which is closer to $\lambda\sigma$ than $\lambda\omega_e$. For both $\lambda\omega_e$ and $\lambda\omega'_e$ we prove WN for typed semi-open terms (i.e. those which allow term variables but no substitution variables), unlike the result of Goubault-Larrecq which covered all $\lambda\sigma$ open terms.

Keywords: lambda calculus, explicit substitution, weak normalization, simple typing.

1 Introduction

Classical λ-calculus has the meta-substitution operator which behaves as an atomic or primitive operation. Since there are many interesting questions regarding the way substitutions are "executed", calculi of explicit substitution for the λ-calculus were created. The aim is to fill the gap between theory and implementation of the λ-calculus.

During the last fifteen years, many lambda calculi with explicit substitution have been proposed and studied. Among all the known calculi, $\lambda\sigma$ [1] has been historically the first one as well as a model for others to compare with. This pioneering calculus reflects in its choice of operators and rules the calculus of categorical combinators (cf. [3]). The main innovation of the $\lambda\sigma$-calculus is the division of terms in two sorts: the sort `term` and the sort `substitution`. λs_e [8] departs from this style of explicit substitutions in two ways. First, it keeps the classical and unique sort `term` of the λ-calculus. Second, it does not use some of the categorical operators, especially those which are not present in the λ-calculus. λs_e introduces two new sets of operators which reflect the substitution and updating that are only present in the meta-language of the λ-calculus. By doing so, the λs_e-calculus can be said to be closer to the λ-calculus from an intuitive point of view, rather than a categorical one. The λs_e-calculus, like the $\lambda\sigma$-calculus, does not preserve strong normalization and its simply typed version is not strongly normalizing [7].

The object of this paper is to study the weak normalization of typable terms in a simply typed de Bruijn lambda calculus with explicit substitution, and some variants. We begin by giving here a brief survey of the simply typed versions of the λ-calculus with de Bruijn indices [5], and of λs, λs_e and their respectively isomorphic (in a sense to be defined later) calculi $\lambda\omega$ and $\lambda\omega_e$ as presented in [9]. From the point of view of syntax the only difference in those presentations from the type free framework is that abstractions are decorated with types. In the calculi we study here we also have metavariables, de Bruijn indices and updating operators decorated with environments. We will present these extensions in Section 2.

A result of weak normalization for typed λs_e with open terms is quite interesting, since on the one hand λs_e is a calculus enjoying most good properties, and on the other it is an open problem whether the s_e-calculus is strongly normalizing. Moreover, we give strategies to calculate normal forms. For the proof of WN we will use the method of Goubault-Larrecq [6] already employed for $\lambda\sigma$, although a non-trivial adaptation is necessary for λs_e as we will see.

We assume familiarity with de Bruijn notation and meta-substitution (cf. [5]), and we assume the usual conventions about parentheses. Throughout the article, we use $a, b, \ldots$ to range over terms of any of the calculi considered, $s, t, \ldots$ to range over substitutions of any of the calculi having substitutions as a sort, and $m, n, \ldots$ to range over $I\!N$ (positive natural numbers). Moreover, $a = b$ means that a and b are syntactically identical, and $\rightarrow^+$ and $\twoheadrightarrow$ denote the transitive and the reflexive transitive closures of a reduction notion $\rightarrow$.

Types and environments are defined for all the calculi discussed in this paper, as follows:

DEFINITION 1.1
The syntax for types and environments is given by:

$$\textbf{Types}\quad \mathcal{T} \ ::= \ T \mid \mathcal{T} \to \mathcal{T} \qquad\qquad \textbf{Environments}\quad \mathcal{E} \ ::= \ nil \mid \mathcal{T}, \mathcal{E}$$

where T is a set of basic types. We let A, B, etc. range over $\mathcal{T}$ and E, F, etc. range over $\mathcal{E}$.

The following notation for environments will be frequently used. For an environment $E = A_1, A_2, \ldots, A_n$, we denote with E_i the ith type of the environment, i.e. A_i, with $E_{\geq i}$ the environment $A_i, A_{i+1}, \ldots, A_n$ and with $E_{>i}$ the environment $A_{i+1}, A_{i+2}, \ldots, A_n$. Similarly, $E_{\leq i}$ denotes the environment $A_1, A_2, \ldots, A_i$; $E_{<i}$ is the environment $A_1, A_2, \ldots, A_{i-1}$, and $E_{<i,B,\geq i}$ is the environment

$$A_1, A_2, \ldots, A_{i-1}, B, A_i, \ldots, A_n$$

i.e. the result of adding type B before position i to the environment E.

The simply typed λ-calculus in de Bruijn notation is given as follows:

$(\mathbf{L1} - var)$	$A, E \vdash \mathtt{1} : A$	$(\mathbf{L1} - \lambda)$	$\dfrac{A, E \vdash b : B}{E \vdash \lambda A.b : A \to B}$
$(\mathbf{L1} - varn)$	$\dfrac{E \vdash \mathtt{n} : B}{A, E \vdash \mathtt{n+1} : B}$	$(\mathbf{L1} - app)$	$\dfrac{E \vdash b : A \to B \quad E \vdash a : A}{E \vdash b\,a : B}$

FIG. 1. The typing rules of the simply typed λ-calculus

DEFINITION 1.2

1. The syntax of the *simply typed λ-calculus* is given by:

$$\Lambda \quad ::= \quad I\!N \mid (\Lambda\Lambda) \mid (\lambda\mathcal{T}.\Lambda).$$

2. We say that a reduction $\to$ is *compatible on* Λ when for all a, b, $c \in \Lambda$, if $a \to b$ then $a\,c \to b\,c$, $c\,a \to c\,b$ and $\lambda A.a \to \lambda A.b$.

3. *β-reduction* is the smallest compatible reduction on Λ generated by:
   (β-rule) $\qquad (\lambda A.a)\,b \to_\beta a\{\!\{1 \leftarrow b\}\!\}$
   where $\bullet\{\!\{\bullet \leftarrow \bullet\}\!\}$ is the usual meta-substitution for the de Bruijn terms [5].

   The *λ-calculus (à la de Bruijn)*, is the reduction system whose only rewriting rule is β.

4. The typing rules for the simply typed λ-calculus in de Bruijn notation are defined in Figure 1. (We call the typing system **L1**).

2 Typing lambda calculi by marking abstractions and operators

In this section we give the typed versions of the calculi we are going to study. We recall the calculus $\lambda\vec{\omega}_e$ from [8], and we remark that when we define this calculus as well as $\lambda\vec{s}_e$ they differ from the previously studied versions (see [8, 9]) in that not only abstractions are decorated with types (as in Church-style formulations of typed λ-calculi), but also metavariables, de Bruijn indices and updating operators are decorated with environments. These new decorations should allow one to type each typable term in a unique way, in the sense that given a typable term, there is a unique environment and a unique type for it. This will allow us to talk about *the type* of a term. For instance, the type of $n_{A_1\ldots A_m}$ will be A_n in the environment $A_1 \ldots A_m$ (for $m \geq n$), whereas undecorated terms would have a type depending on the given environment. Nevertheless we remark that all the results to follow also hold when erasing all type decorations.

$\overrightarrow{\sigma}$*-gen*	$(\lambda A.a)\, b$	$\longrightarrow$	$a\,\sigma^1\, b$
$\overrightarrow{\sigma}$*-λ-tr*	$(\lambda A.a)\,\sigma^j b$	$\longrightarrow$	$\lambda A.(a\,\sigma^{j+1}\, b)$
$\overrightarrow{\sigma}$*-app-tr*	$(a_1\, a_2)\,\sigma^j b$	$\longrightarrow$	$(a_1\,\sigma^j b)\,(a_2\,\sigma^j b)$
$\overrightarrow{\sigma}$*-des*	$\mathtt{n}_{E_{<j},B,E_{\geq j}}\,\sigma^j b$	$\longrightarrow$	$\begin{cases} \mathtt{n}-\mathtt{1}_E & \text{if } n>j \\ \varphi_0^{j,E}\, b & \text{if } n=j \\ \mathtt{n}_E & \text{if } n<j \end{cases}$
$\overrightarrow{\varphi}$*-λ-tr*	$\varphi_k^{i,E}(\lambda A.a)$	$\longrightarrow$	$\lambda A.(\varphi_{k+1}^{i,E}\, a)$
$\overrightarrow{\varphi}$*-app-tr*	$\varphi_k^{i,E}(a_1\, a_2)$	$\longrightarrow$	$(\varphi_k^{i,E}\, a_1)\,(\varphi_k^{i,E}\, a_2)$
$\overrightarrow{\varphi}$*-des*	$\varphi_k^{i,E_{>k}}\,\mathtt{n}_{E_{\leq k},E_{\geq k+i}}$	$\longrightarrow$	$\begin{cases} \mathtt{n}+\mathtt{i}-\mathtt{1}_E & \text{if } n>k \\ \mathtt{n}_E & \text{if } n\leq k \end{cases}$

FIG. 2. The rewriting rules of the simply typed λs-calculus

2.1 The simply typed λs and λs_e-calculi

DEFINITION 2.1
The set of *ground simply typed λs-terms*, denoted $\Lambda\overrightarrow{s}$, and the set of *open simply typed λs-terms*, denoted $\Lambda\overrightarrow{s}_{op}$ are given as follows :

$$\textbf{Gr. Terms } \Lambda\overrightarrow{s} \;::=\; I\!N_{\mathcal{E}} \mid \Lambda\overrightarrow{s}\;\Lambda\overrightarrow{s} \mid \lambda\mathcal{T}.\Lambda\overrightarrow{s} \mid \Lambda\overrightarrow{s}\;\sigma^j\Lambda\overrightarrow{s} \mid \varphi_k^{i,\mathcal{E}}\Lambda\overrightarrow{s}$$
$$\textbf{Op. Terms } \Lambda\overrightarrow{s}_{op} ::= \mathbf{V}_{\mathcal{E};\mathcal{T}} \mid I\!N_{\mathcal{E}} \mid \Lambda\overrightarrow{s}_{op}\;\Lambda\overrightarrow{s}_{op} \mid \lambda\mathcal{T}.\Lambda\overrightarrow{s}_{op} \mid \Lambda\overrightarrow{s}_{op}\;\sigma^j\Lambda\overrightarrow{s}_{op} \mid \varphi_k^{i,\mathcal{E}}\Lambda\overrightarrow{s}_{op}$$

where $i,j\geq 1$, $k\geq 0$ and $\mathbf{V}$ stands for a set of variables with pairs of environments and types as sub-indices, denoted as $X_{E,T}$, $Y_{E,T}$, etc. For each $\mathtt{n}\in I\!N$, we assume that $\mathtt{n}_E$ can be formed only when the length of E is greater than or equal to n. Sometimes, when no ambiguity could arise, the subscripts will be omitted. We call *pure terms* the terms that do not contain metavariables (i.e. elements of $\mathbf{V}_{\mathcal{E};\mathcal{T}}$), σ- or φ-operators.

DEFINITION 2.2

1. A *closure* is a term of the form $a\,\sigma^j b$. A *pure term* does not contain σ's nor φ's. *Compatibility* on Λs is extended by adding: if $a\to b$ then $a\,\sigma^j c\to b\,\sigma^j c$, $c\,\sigma^j a\to c\,\sigma^j b$ and $\varphi_k^i a\to\varphi_k^i b$.
2. The simply typed λs*-calculus* is given by the rewriting rules in Figure 2, i.e. its reduction relation is the smallest compatible relation generated by those rules. We use $\lambda\overrightarrow{s}$ to denote this set of rules. The $\overrightarrow{s}$-calculus is the rewriting system given by the set of rules $\overrightarrow{s}=\lambda\overrightarrow{s}\setminus\{\overrightarrow{\sigma}\textit{-gen}\}$. The typing rules are given by the typing system $\mathbf{L}\,\overrightarrow{\mathbf{s1}}$ of Figure 3. We will use the symbol $\vdash_{\overrightarrow{\mathbf{Ls1}}}$ for this typing relation.
3. We say that $a\in\Lambda\overrightarrow{s}$ is a *well typed* term, or *typed* for short, if there exists an environment E and a type A such that $E\vdash_{\overrightarrow{\mathbf{Ls1}}} a : A$.

$(\mathbf{L}\,\vec{\mathbf{s1}} - var)$	$A, E \vdash 1_{A,E} : A$	$(\mathbf{L}\,\vec{\mathbf{s1}} - \lambda)$	$\dfrac{A, E \vdash b : B}{E \vdash \lambda A.b : A \to B}$
$(\mathbf{L}\,\vec{\mathbf{s1}} - varn)$	$\dfrac{E \vdash \mathtt{n}_E : B}{A, E \vdash \mathtt{n} + 1_{A,E} : B}$	$(\mathbf{L}\,\vec{\mathbf{s1}} - app)$	$\dfrac{E \vdash b : A \to B \quad E \vdash a : A}{E \vdash b\,a : B}$
$(\mathbf{L}\,\vec{\mathbf{s1}} - \sigma)$	$\dfrac{E_{\geq i} \vdash b : B \quad E_{<i}, B, E_{\geq i} \vdash a : A}{E \vdash a\,\sigma^i b : A}$	$(\mathbf{L}\,\vec{\mathbf{s1}} - \varphi)$	$\dfrac{E_{\leq k}, E_{\geq k+i} \vdash a : A}{E \vdash \varphi_k^{i,E_{>k}}\,a : A}$

FIG. 3. The typing rules of the simply typed λs-calculus

$\vec{\sigma}$-$\vec{\sigma}$-*tr*	$(a\,\sigma^i b)\,\sigma^j\,c$	$\longrightarrow$	$(a\,\sigma^{j+1}\,c)\;\sigma^i\,(b\,\sigma^{j-i+1}\,c)$	if	$i \leq j$
$\vec{\sigma}$-$\vec{\varphi}$-*tr 1*	$(\varphi_k^{i,E_{<j-k},B,E_{\geq j-k}}\,a)\,\sigma^j\,b$	$\longrightarrow$	$\varphi_k^{i-1,E}\;a$	if	$k < j < k+i$
$\vec{\sigma}$-$\vec{\varphi}$-*tr 2*	$(\varphi_k^{i,E_{<j-k},B,E_{\geq j-k}}\,a)\,\sigma^j\,b$	$\longrightarrow$	$\varphi_k^{i,E}(a\,\sigma^{j-i+1}\,b)$	if	$k+i \leq j$
$\vec{\varphi}$-$\vec{\sigma}$-*tr*	$\varphi_k^{i,E}(a\,\sigma^j\,b)$	$\longrightarrow$	$(\varphi_{k+1}^{i,E}\,a)\,\sigma^j\,(\varphi_{k+1-j}^{i,E}\,b)$	if	$j \leq k+1$
$\vec{\varphi}$-$\vec{\varphi}$-*tr 1*	$\varphi_k^{i,E_{>k-l}}(\varphi_\ell^{j,E_{\leq k-l},E_{\geq k+i-l}}\;a)$	$\longrightarrow$	$\varphi_\ell^{j,E}\;(\varphi_{k+1-j}^{i,E_{>k-l}}\;a)$	if	$l+j \leq k$
$\vec{\varphi}$-$\vec{\varphi}$-*tr 2*	$\varphi_k^{i,E_{>k-l}}(\varphi_\ell^{j,E_{\leq k-l},E_{\geq k+i-l}}\;a)$	$\longrightarrow$	$\varphi_\ell^{j+i-1,E}\;a$	if	$l \leq k < l+j$

FIG. 4. The new rewriting rules of the simply typed λs_e-calculus

DEFINITION 2.3

1. The set of rules $\lambda\vec{s}_e$ is obtained by adding the rules in Figure 4 to the rules of the $\lambda\vec{s}$-calculus given in Figure 2. The *$\lambda\vec{s}_e$-calculus* is the reduction system $(\Lambda\vec{s}_{op}, \to_{\lambda\vec{s}_e})$ where $\to_{\lambda\vec{s}_e}$ is the smallest compatible reduction on $\Lambda\vec{s}_{op}$ generated by the set of rules $\lambda\vec{s}_e$.

   The *$\vec{s}_e$-calculus* is the rewriting system generated by the set of rules $\vec{s}_e = \lambda\vec{s}_e \setminus \{\vec{\sigma}\text{-}gen\}$. Remark that the typing rules for $\lambda\vec{s}_e$ are exactly the same as the typing rules for $\lambda\vec{s}$ given in Figure 3. We only need to add rules to type metavariables:

$$(L\,\vec{s1}\ -Metav) \qquad E \vdash X_{E,A} : A.$$

   We further assume that for each context E and type A there are infinitely many metavariables X, such that $E \vdash X : A$.

2. We say that $a \in \Lambda\vec{s}_{op}$ is a *well typed* term, or *typed* for short, if there exists an environment E and a type A such that $E \vdash_{\vec{\mathbf{Ls1}}} a : A$.

We denote with λs and λs_e the respective untyped calculi, i.e. the calculi where the decorations have been erased.

2.2 The simply typed $\lambda\omega$ and $\lambda\omega_e$-calculi

In order to express λs-terms in the $\lambda\sigma$-style, we take the approach of [9]. We split the closure operator of $\lambda\sigma$ (denoted in a semi-infix notation as $-[-]$) into a family of closure operators that were denoted also with a semi-infix notation as $-[-]_i$, where i ranges over the set of natural numbers. We also admit as basic operators the iterations of $\uparrow$ and therefore have a countable set of basic substitutions $\uparrow^n$, where n ranges over the set of natural numbers. By doing so, the updating operators of λs become available as $-[\uparrow^n]_i$. Finally, we use a *slash* operator of sort `term` $\to$ `substitution` which transforms a term a into a substitution $a/$. This operator may be considered as *consing with id* (in the $\lambda\sigma$-jargon) and has been first exploited in the $\lambda\upsilon$-calculus (cf. [2]).

DEFINITION 2.4
The set of terms of the $\lambda\vec{\omega}$-calculus, noted $\Lambda\vec{\omega}$, is defined as $\Lambda\,\vec{\omega}^t \cup \Lambda\,\vec{\omega}^s$, where $\Lambda\,\vec{\omega}^t$ and $\Lambda\,\vec{\omega}^s$ are mutually defined as follows :

Terms	$\Lambda\,\vec{\omega}^t ::= I\!N_{\mathcal{E}} \mid \Lambda\,\vec{\omega}^t\,\Lambda\,\vec{\omega}^t \mid \lambda\mathcal{T}.\Lambda\,\vec{\omega}^t \mid \Lambda\,\vec{\omega}^t\,[\Lambda\,\vec{\omega}^s]_j$ where $j \geq 1$
Substitutions	$\Lambda\,\vec{\omega}^s ::= \uparrow^i_{\mathcal{E}} \mid \Lambda\,\vec{\omega}^t\,/$ where $i \geq 0$

The set, denoted $\lambda\vec{\omega}$, of rules of the $\lambda\vec{\omega}$*-calculus* is given in Figure 5 . The set of rules of the $\vec{\omega}$*-calculus* is the set $\vec{\omega} = \lambda\vec{\omega}\setminus\{\vec{\sigma}-gen\}$. *Closures* are terms of the form $a[s]_i$, *pure terms* are terms without substitutions, and *compatibility* is defined in the usual way.

The typing system for the $\lambda\vec{\omega}$-calculus is called $\mathbf{L}\vec{\omega}\mathbf{1}$. The rules $\mathbf{L}\vec{\omega}\mathbf{1}$-*var*, $\mathbf{L}\vec{\omega}\mathbf{1}$-*varn*, $\mathbf{L}\vec{\omega}\mathbf{1}$-$\lambda$ and $\mathbf{L}\vec{\omega}\mathbf{1}$-*app* are exactly the same as $\mathbf{L}\vec{s}\,\mathbf{1}$-*var*, $\mathbf{L}\vec{s}\,\mathbf{1}$-*varn*, $\mathbf{L}\vec{s}\,\mathbf{1}$-$\lambda$ and $\mathbf{L}\vec{s}\,\mathbf{1}$-*app*, respectively, as given in Figure 3. The new rules are given in Figure 6 . We will use the symbol $\vdash_{\mathbf{L}\vec{\omega}\mathbf{1}}$ for this typing relation.

Just like the $\lambda\vec{s}$-calculus, the $\lambda\vec{\omega}$-calculus is not even locally confluent on open terms. By *open terms* in this new syntax we mean terms which admit variables (usually called metavariables) of sort `term` and of sort `substitution`.

Now, we define formally what we mean by open terms in our new syntax and give the rewriting rules of $\lambda\vec{\omega}_e$:

DEFINITION 2.5
The set of *open terms*, noted $\Lambda\vec{\omega}_{op}$ is defined as $\Lambda\,\vec{\omega}^t_{op} \cup \Lambda\,\vec{\omega}^s_{op}$, where $\Lambda\,\vec{\omega}^t_{op}$ and $\Lambda\,\vec{\omega}^s_{op}$ are mutually defined as follows :

Op. Terms	$\Lambda\,\vec{\omega}^t_{op} ::= \mathbf{V}_{\mathcal{E};\mathcal{T}} \mid I\!N_{\mathcal{E}} \mid \Lambda\,\vec{\omega}^t_{op}\,\Lambda\,\vec{\omega}^t_{op} \mid \lambda\mathcal{T}.\Lambda\,\vec{\omega}^t_{op} \mid \Lambda\,\vec{\omega}^t_{op}\,[\Lambda\,\vec{\omega}^s_{op}]_j$
Op. Subst.	$\Lambda\,\vec{\omega}^s_{op} ::= \mathbf{W}_{\mathcal{E};\mathcal{E}} \mid \uparrow^i_{\mathcal{E}} \mid \Lambda\,\vec{\omega}^t_{op}\,/$

where $j \geq 1$ and $i \geq 0$, and $\mathbf{V}_{\mathcal{E};\mathcal{T}}$ stands for a set of term variables, denoted $X_{E,T}$, $Y_{E,T}$, etc. and $\mathbf{W}_{\mathcal{E};\mathcal{E}}$ stands for a set of substitution variables denoted $x_{E,E'}$, $y_{E,E'}$,

$$
\begin{array}{llcl}
\vec{\sigma}\textit{-gen} & (\lambda A.a)\, b & \longrightarrow & a\,[b/]_1 \\
\vec{\sigma}\textit{-app-tr} & (a\, b)[s]_j & \longrightarrow & (a\,[s]_j)\,(b\,[s]_j) \\
\vec{\sigma}\textit{-}\lambda\textit{-tr} & (\lambda A.a)[s]_j & \longrightarrow & \lambda A.(a[s]_{j+1}) \\
\vec{\sigma}\textit{-/-des} & \mathtt{n}_{E_{<j},B,E_{\geq j}}[a/]_j & \longrightarrow & \begin{cases} \mathtt{n}-\mathtt{1}_E & \text{if } n > j \\ a[\uparrow_E^{j-1}]_1 & \text{if } n = j \\ \mathtt{n}_E & \text{if } n < j \end{cases} \\
\vec{\sigma}\textit{-}\uparrow\textit{-des} & \mathtt{n}_{E_{<j},E_{\geq i+j}}[\uparrow^i_{E_{\geq j}}]_j & \longrightarrow & \begin{cases} \mathtt{n}+\mathtt{i}_E & \text{if } n \geq j \\ \mathtt{n}_E & \text{if } n < j \end{cases}
\end{array}
$$

FIG. 5. The rewriting rules of the simply typed $\lambda\omega$-calculus

etc. We take s, t, $\ldots$ to range over $\Lambda\, \vec{\omega}^s_{op}$. Closures, pure terms and compatibility are defined as for $\lambda\vec{\omega}$.

The set, denoted $\lambda\vec{\omega}_e$, of rules of the $\lambda\vec{\omega}_e$*-calculus* is obtained by adding to the set of rules $\lambda\vec{\omega}$ given in Figure 5, the rules given in Figure 7. The set of rules of the $\vec{\omega}_e$*-calculus* is $\vec{\omega}_e = \lambda\vec{\omega}_e \setminus \{\vec{\sigma}-gen\}$.

The typing rules for $\lambda\vec{\omega}_e$ are those of $\lambda\vec{\omega}$ together with rules to type metavariables:

$$(L\,\vec{s1}\ -MetavT) \quad E \vdash X_{E,A} : A \qquad\qquad (L\,\vec{s1}\ -MetavS) \quad E \vdash x_{E,E'} \triangleright E'$$

We further assume that for each context E and type A there are infinitely many metavariables X, such that $E \vdash X : A$. We also assume that for each pair of contexts E, E' there are infinitely many metavariables x, such that $E \vdash x \triangleright E'$.

We also define $\Lambda\, \vec{\omega}^t_{sop}$ as the set of *semi-open terms*, i.e. those open terms without substitution variables (it is a proper subset of $\Lambda\, \vec{\omega}^t_{op}$), and $\Lambda\, \vec{\omega}^s_{sop}$ as the set of *semi-open substitutions*, i.e. those open substitutions without substitution variables (it is a proper subset of $\Lambda\, \vec{\omega}^s_{op}$).

$$
\begin{array}{l}
\Lambda\, \vec{\omega}^t_{sop} ::= \mathbf{V}_{\mathcal{E};\mathcal{T}} \mid I\!N_{\mathcal{E}} \mid \Lambda\, \vec{\omega}^t_{sop}\, \Lambda\, \vec{\omega}^t_{sop} \mid \lambda\mathcal{T}.\Lambda\, \vec{\omega}^t_{sop} \mid \Lambda\, \vec{\omega}^t_{sop}\, [\Lambda\, \vec{\omega}^s_{sop}]_j \\
\Lambda\, \vec{\omega}^s_{sop} ::= \uparrow^i_{\mathcal{E}} \mid \Lambda\, \vec{\omega}^t_{sop}\, /
\end{array}
$$

Last, we denote with $\lambda\omega$ and $\lambda\omega_e$ the respective untyped calculi, i.e. the calculi where the decorations have been erased.

In the rest of the paper unless explicitly stated we will restrict $\lambda\vec{\omega}_e$ to the set $\Lambda\, \vec{\omega}^t_{sop}$.

$$
\begin{array}{llll}
(\mathbf{L}\vec{\omega}\mathbf{1} - \mathit{id}) & E \vdash\, \uparrow^0_E\, \triangleright E & (\mathbf{L}\vec{\omega}\mathbf{1} - \mathit{slash}) & \dfrac{E \vdash a : A}{E \vdash a/ \triangleright A, E} \\[2ex]
(\mathbf{L}\vec{\omega}\mathbf{1} - \mathit{shift}) & \dfrac{E \vdash\, \uparrow^i_E\, \triangleright E'}{A, E \vdash\, \uparrow^{i+1}_{A,E}\, \triangleright E'} & (\mathbf{L}\vec{\omega}\mathbf{1} - \mathit{clos}) & \dfrac{E_{\geq j} \vdash s \triangleright E' \quad E_{<j}, E' \vdash a : A}{E \vdash a[s]_j : A} \\[2ex]
(\mathbf{L}\vec{\omega}\mathbf{1} - \mathit{MetavT}) & E \vdash X_{E,A} : A & (\mathbf{L}\vec{\omega}\mathbf{1} - \mathit{MetavS}) & E \vdash x_{E,E'} \triangleright E'
\end{array}
$$

FIG. 6. The new typing rules of the simply typed $\lambda\omega_e$-calculus

$$
\begin{array}{lrcl}
\vec{\sigma}\text{-}/\text{-}tr & a\,[b/]_k[s]_j & \longrightarrow & a\,[s]_{j+1}[b[s]_{j-k+1}/]_k \quad \text{if } \; k \leq j \\[2ex]
/\text{-}\!\uparrow\!\text{-}tr & a\,[\uparrow^i_{E_{\leq j-k}, B, E_{>j-k}}]_k[b/]_j & \longrightarrow & \begin{cases} a[b/]_{j-i}[\uparrow^i_E]_k & \text{if } \; k+i \leq j \\ a[\uparrow^{i-1}_E]_k & \text{if } \; k \leq j < k+i \end{cases} \\[2ex]
\uparrow\!\text{-}\!\uparrow\!\text{-}tr & a\,[\uparrow^i_{E_{\leq j-k}, E_{>j+l-k}}]_k[\uparrow^l_{E_{>j-k}}]_j & \longrightarrow & \begin{cases} a[\uparrow^l_{E_{>j-k}}]_{j-i}[\uparrow^i_E]_k & \text{if } \; k+i < j \\ a[\uparrow^{i+l}_E]_k & \text{if } \; k \leq j \leq k+i \end{cases}
\end{array}
$$

FIG. 7. The new rewriting rules of the simply typed $\lambda\omega_e$-calculus

3 The isomorphism

The untyped versions of $\lambda\vec{s}$, $\lambda\vec{s}_e$, $\lambda\vec{\omega}$ and $\lambda\vec{\omega}_e$ are obtained by deleting every type and environment information of terms (cf. [9]). For the untyped calculi we proved (cf. [9]) that the term restriction of the $\lambda\vec{\omega}$ and $\lambda\vec{\omega}_e$-calculi on $\Lambda\,\vec{\omega}^t_{sop}$ are "isomorphic" respectively to the $\lambda\vec{s}$ and $\lambda\vec{s}_e$-calculi. In this section, we state that the isomorphism can be adapted for the typed versions of these calculi and furthermore, that the new isomorphism preserves types.

DEFINITION 3.1
The functions $T : \Lambda\vec{s}_{op} \rightarrow \Lambda\,\vec{\omega}^t_{sop}$ and $S : \Lambda\,\vec{\omega}^t_{sop} \rightarrow \Lambda\vec{s}_{op}$ are defined inductively:

$$
\begin{array}{ll}
T(X_{E,A}) = X_{E,A} & S(X_{E,A}) = X_{E,A} \\
T(\mathtt{n}_E) = \mathtt{n}_E & S(\mathtt{n}_E) = \mathtt{n}_E \\
T(a\,b) = T(a)T(b) & S(a\,b) = S(a)S(b) \\
T(\lambda A.a) = \lambda A.T(a) & S(\lambda A.a) = \lambda A.S(a) \\
T(a\,\sigma^j b) = T(a)[T(b)/]_j & S(a\,[b/]_j) = S(a)\,\sigma^j S(b) \\
T(\varphi^{i,E}_k a) = T(a)[\uparrow^{i-1}_E]_{k+1} & S(a\,[\uparrow^i_E]_k) = \varphi^{i+1,E}_{k-1}(S(a))
\end{array}
$$

We make an "abus de notation" and use the same names T and S for the trivial restrictions of these functions to ground terms. The context will make it clear which is meant in every case.

THEOREM 3.2
The following hold:

1. Let $a, b \in \Lambda\vec{s}_{op}$. If $a \to_{\vec{s}} b$ then $T(a) \to_{\vec{\omega}} T(b)$. If $a \to_{\lambda\vec{s}} b$ then $T(a) \to_{\lambda\vec{\omega}} T(b)$.
2. Let $a, b \in \Lambda\vec{s}_{op}$. If $a \to_{\vec{s}_e} b$ then $T(a) \to_{\vec{\omega}_e} T(b)$. If $a \to_{\lambda\vec{s}_e} b$ then $T(a) \to_{\lambda\vec{\omega}_e} T(b)$.
3. Let $a, b \in \Lambda\,\vec{\omega}^t_{sop}$. If $a \to_{\vec{\omega}} b$ then $S(a) \to_{\vec{s}} S(b)$.
   If $a \to_{\lambda\vec{\omega}_e} b$ then $S(a) \to_{\lambda\vec{s}_e} S(b)$.
4. Let $a, b \in \Lambda\,\vec{\omega}^t_{sop}$. If $a \to_{\vec{\omega}_e} b$ then $S(a) \to_{\vec{s}_e} S(b)$.
   If $a \to_{\lambda\vec{\omega}_e} b$ then $S(a) \to_{\lambda\vec{s}_e} S(b)$.

PROOF: By induction on a. If the reduction is internal, the induction hypothesis applies; otherwise, the theorem must be checked for each rule. As an example, we illustrate for item 2. the case of reduction at the root with the rules *σ-φ-tr 1* and *σ-φ-tr 2*:
Suppose $k < j < k+i$. Then $(\varphi^i_k a)\sigma^j b \to_{\sigma-\varphi-tr1} \varphi^{i-1}_k a$, and
$T(\varphi^i_k a)\sigma^j b) = T(\varphi^i_k a)[T(b)/]_j$
$= T(a)[\uparrow^{i-1}]_{k+1}[T(b)/]_j \to_{/-\uparrow-tr} T(a)[\uparrow^{i-2}]_{k+1}$ (since $k+i \geq j \geq k+1 > k$)
$= T(\varphi^{i-1}_k a)$.
Suppose $k+i \leq j$. Then $(\varphi^i_k a)\sigma^j b \to_{\sigma-\varphi-tr2} \varphi^i_k(a\sigma^{j-i+1}b)$, and
$T((\varphi^i_k a)\sigma^j b) = T(\varphi^i_k a)[T(b)/]_j$
$= T(a)[\uparrow^{i-1}]_{k+1}[T(b)/]_j \to_{/-\uparrow-tr} T(a)[T(b)/]_{j-i+1}[\uparrow^{i-1}]_{k+1}$ (since $k+i \leq j$)
$= T(a\sigma^{j-i+1}b)[\uparrow^{i-1}]_{k+1} = T(\varphi^i_k(a\sigma^{j-i+1}b))$.
The other cases are simpler. ⊠

We verify finally that T and S are in fact inverses of each other.

THEOREM 3.3
The following hold:

1. For all $a \in \Lambda\,\vec{\omega}^t$, we have $T(S(a)) = a$. For all $a \in \Lambda\vec{s}$, we have $S(T(a)) = a$.
2. For all $a \in \Lambda\,\vec{\omega}^t_{sop}$, we have $T(S(a)) = a$. For all $a \in \Lambda\vec{s}_{op}$, we have $S(T(a)) = a$.

PROOF: By an easy induction on a. ⊠

COROLLARY 3.4
$\lambda\vec{s}$ is isomorphic to $\lambda\vec{\omega}$ restricted to $\Lambda\,\vec{\omega}^t$, and $\lambda\vec{s}_e$ is isomorphic to $\lambda\vec{\omega}_e$ restricted to $\Lambda\,\vec{\omega}^t_{sop}$.

We end this section by stating that the isomorphism preserves types, which will be used to obtain subject reduction for $\lambda\vec{s}$ and $\lambda\vec{s}_e$ from the corresponding results for $\lambda\vec{\omega}$ and $\lambda\vec{\omega}_e$.

LEMMA 3.5
Let E be an environment, A a type, $a \in \Lambda\vec{s}_{op}$ and $b \in \Lambda\,\vec{\omega}^t_{sop}$.

1. If $E \vdash_{\mathbf{L}\vec{\mathbf{s}}\mathbf{1}} a : A$ then $E \vdash_{\mathbf{L}\vec{\omega}\mathbf{1}} T(a) : A$.
2. If $E \vdash_{\mathbf{L}\vec{\omega}\mathbf{1}} b : A$ then $E \vdash_{\mathbf{L}\vec{\mathbf{s}}\mathbf{1}} S(b) : A$.

PROOF: By induction on the structure of a and b, respectively. ⊠

4 Subject Reduction

This section is devoted to establish Subject Reduction for our four calculi. We prove first subject reduction for $\lambda\overrightarrow{\omega}$ and $\lambda\overrightarrow{\omega}_e$ and then we use the isomorphisms given in the previous section to obtain Subject Reduction for $\lambda\overrightarrow{s}$ and $\lambda\overrightarrow{s}_e$.

THEOREM 4.1 (Subject Reduction for $\lambda\overrightarrow{\omega}$)
Let $a, b \in \Lambda\overrightarrow{\omega}^t_{sop}$ and $s, t \in \Lambda\overrightarrow{\omega}^s_{sop}$.

1. If $E \vdash a : A$ and $a \to_{\lambda\overrightarrow{\omega}} b$ then $E \vdash b : A$.
2. If $E \vdash s \rhd F$ and $s \to_{\lambda\overrightarrow{\omega}} t$ then $E \vdash t \rhd F$.

PROOF: By simultaneous induction on the structure of a and s. If the reduction is internal it is enough to apply the inductive hypothesis. If the reduction is at the root then each rule must be examined. We check for instance the rule $\overrightarrow{\sigma}$*-/-des* for the case $n = j$.
Let us assume $E \vdash \mathtt{n}_{F_{<j},B,F_{\geq j}}[a/]_j : A$. Therefore there exists an environment E' such that $E_{\geq j} \vdash a/ \rhd E'$ and $E_{<j}, E' \vdash \mathtt{n}_{F_{<j},B,F_{\geq j}} : A$. Hence $E_{<j} = F_{<j}$ and $E' = B, F_{\geq j}$ and therefore $E_{\geq j} = F_{\geq j}$, hence $E = F$. From $E_{\geq j} \vdash a/ \rhd E'$ we deduce $E_{\geq j} \vdash a : B$ and, since $A = (E_{<j}, B, E_{\geq j})_n$ and $n = j$, we have $A = B$. Therefore, $E_{\geq j} \vdash a : A$ and, because $E \vdash \uparrow_E^{j-1} \rhd E_{\geq j}$, we can apply the *clos*-rule (remember $E = E_{\geq 1}$ and, by convention, $E_{<1} = nil$) to obtain $E \vdash a[\uparrow_E^{j-1}]_1 : A$. ⊠

THEOREM 4.2 (Subject Reduction for $\lambda\overrightarrow{\omega}_e$)
Let $a, b \in \Lambda\overrightarrow{\omega}^t_{sop}$ and $s, t \in \Lambda\overrightarrow{\omega}^s_{sop}$.

1. If $E \vdash a : A$ and $a \to_{\lambda\overrightarrow{\omega}_e} b$ then $E \vdash b : A$.
2. If $E \vdash s \rhd F$ and $s \to_{\lambda\overrightarrow{\omega}_e} t$ then $E \vdash t \rhd F$.

PROOF: By simultaneous induction on the structure of a and s. The proof is analogous to the previous proof, only the new rules must be checked now. As an example we study $\overrightarrow{\sigma}$*-/-tr*. Assume $E \vdash a[b/]_k[s]_j : A$ and $k \leq j$. Therefore, there exists an environment E' such that

$$E_{\geq j} \vdash s \rhd E' \tag{4.1}$$

and $E_{<j}, E' \vdash a[b/]_k : A$. Since $k \leq j$, by $\mathbf{L}\overrightarrow{\omega}\mathbf{1}$-*clos* there exists an environment E'' such that

$$E_{<k}, E'' \vdash a : A \tag{4.2}$$

and $E_k, \ldots, E_{j-1}, E' \vdash b/ \rhd E''$. Therefore, $E'' = B, E_k, \ldots, E_{j-1}, E'$ and

$$E_k, \ldots, E_{j-1}, E' \vdash b : B \tag{4.3}$$

Applying the *clos* rule, from 4.1 and 4.2 we get

$$E_{<k}, B, E_{\geq k} \vdash a[s]_{j+1} : A \tag{4.4}$$

and from 4.1 and 4.3, $E_{\geq k} \vdash b[s]_{j-k+1} : B$, and a further application of *slash* gives

$$E_{\geq k} \vdash b[s]_{j-k+1}/ \rhd B, E_{\geq k} \tag{4.5}$$

Finally, applying *clos* to 4.4 and 4.5, we conclude $E \vdash a[s]_{j+1}[b[s]_{j-k+1}/]_k : A$. ⊠

We use now the translations to prove Subject Reduction for $\lambda\vec{s}$ and $\lambda\vec{s}_e$.

THEOREM 4.3 (Subject Reduction for $\lambda\vec{s}$ and $\lambda\vec{s}_e$)
Let $a, b \in \Lambda\vec{s}$ and $c, d \in \Lambda\vec{s}_{op}$.

1. If $E \vdash a : A$ and $a \to_{\lambda\vec{s}} b$ then $E \vdash b : A$.
2. If $E \vdash c : A$ and $c \to_{\lambda\vec{s}_e} d$ then $E \vdash d : A$.

PROOF: We just check the 1st item (the 2nd is similar). If $E \vdash a : A$ then, by Lemma 3.5.1, $E \vdash T(a) : A$. On the other hand, if $a \to_{\lambda\vec{s}} b$ then, by Theorem 3.2.2, $T(a) \to_{\lambda\vec{\omega}} T(b)$. Now, by Theorem 4.1.1, $E \vdash T(b) : A$, and by Lemma 3.5.2, we get $E \vdash S(T(b)) : A$, and we are done because $S(T(b)) = b$, by Theorem 3.3. ⊠

5 Weak Normalization of $\vec{\omega}_e$

In this section we prove weak normalization for $\vec{\omega}_e$, the calculus of substitutions associated to $\lambda\vec{\omega}_e$, by reducing the problem to the untyped calculus. Weak normalization of $\vec{\omega}_e$ will be needed in the next section to obtain weak normalization of $\lambda\vec{\omega}_e$.

DEFINITION 5.1
We define *type erasure* for $\lambda\vec{\omega}$-terms as follows:

$$\begin{array}{ll} |X_{E,A}| = X_{E,A} & |x_{E,F}| = x_{E,F} \\ |n_E| = n & |\uparrow^i_E| = \uparrow^i \\ |a\,b| = |a|\,|b| & |a/| = |a|/ \\ |\lambda A.a| = \lambda|a| & |a[s]_j| = |a|[|s|]_j \end{array}$$

LEMMA 5.2
Let $a, b \in \Lambda\,\vec{\omega}^t_{sop}$ and $s, t \in \Lambda\,\vec{\omega}^s_{sop}$.

1. If $a \to_{\lambda\vec{\omega}_e} b$ then $|a| \to_{\lambda\omega_e} |b|$.
2. If $s \to_{\lambda\vec{\omega}_e} t$ then $|s| \to_{\lambda\omega_e} |t|$.

PROOF: By an easy induction on the structure of terms and substitutions. ⊠

THEOREM 5.3
$\vec{\omega}_e$-calculus is weakly normalizing for semi-open terms.

PROOF: In [8] it is shown that every innermost strategy terminates in the s_e-calculus, i.e. the untyped version of $\vec{s}_e$. Here, we prove that every innermost strategy must also terminate for $\vec{\omega}_e$. The proof is by contradiction. Let us consider an innermost infinite reduction path beginning with:

- a term a, i.e. $a \to a_1 \to \ldots \to a_n \to \ldots$. Now, using the previous lemma and remarking that erasing the types does not change the character of the strategy, we get an innermost infinite derivation in ω_e:
  $|a| \to |a_1| \to \ldots \to |a_n| \to \ldots$
  and then, applying the translation S (cf.[9]) from untyped $\lambda\vec{\omega}$-terms into untyped $\lambda\vec{s}_e$-terms, which does not change the character of the strategy either, we get the innermost infinite s_e-derivation:
  $S(|a|) \to S(|a_1|) \to \ldots \to S(|a_n|) \to \ldots$
  which contradicts the above mentioned result in [8].

- a substitution s, then $s = a/$ (because $s =\uparrow^i_E$ is a normal form) and the infinite reduction must occur within a. Hence, by the previous item, this is also a contradiction.

$\boxtimes$

6 Soundness and simulation

We have shown that $\vec{\omega}_e$ is WN for semi-open terms. Therefore for every term a in the corresponding language we can define the normal forms $\vec{s}\ (a)$, $\vec{s}_e\ (a)$, $\vec{\omega}_e\ (a)$ as usual. In this section we show that these calculi enjoy the expected soundness and simulation properties with respect to de Bruijn λ-calculus.

These calculi are sound in the following sense:

Proposition 6.1 (Soundness of $\lambda\vec{s}$, $\lambda\vec{s}_e$, $\lambda\vec{\omega}$ and $\lambda\vec{\omega}_e$)
The following hold:

1. Let $a,\ b \in \Lambda\vec{s}$. If $a \to_{\lambda\vec{s}} b$ then $\vec{s}\ (a) \twoheadrightarrow_\beta\ \vec{s}\ (b)$.
2. Let $a,\ b \in \Lambda\vec{s}_{op}$. If $a \to_{\lambda\vec{s}_e} b$ then $\vec{s}_e\ (a) \twoheadrightarrow_\beta\ \vec{s}_e\ (b)$.
3. Let $a,\ b \in \Lambda\vec{\omega}$. If $a \to_{\lambda\vec{\omega}} b$ then $\vec{\omega}\ (a) \twoheadrightarrow_\beta\ \vec{\omega}\ (b)$.
4. Let $a,\ b \in \Lambda\vec{\omega}_{sop}$. If $a \to_{\lambda\vec{\omega}_e} b$ then $\vec{\omega}_e\ (a) \twoheadrightarrow_\beta\ \vec{\omega}_e\ (b)$.

Proof: All items can be proved by induction on the position where the reduction takes place. See more details in [8]. $\boxtimes$

Also, they simulate the β-reduction in the de Bruijn λ-calculus with or without open terms. We recall next the definition of the de Bruijn λ-calculus with the addition of metavariables.

Definition 6.2
We define the de Bruijn open terms by:

$$\textbf{Op.Terms} \quad \Lambda_{op} ::= \mathbf{V}_{\mathcal{E};\mathcal{T}}\ \mid I\!N_{\mathcal{E}}\ \mid (\Lambda_{op}\Lambda_{op})\ \mid (\lambda\mathcal{T}.\Lambda_{op})$$

The de Bruijn λ-calculus on open terms has as its only rule *β-reduction*, which is be the smallest compatible reduction on Λ_{op} based on the schema in Definition 1.2. As before we may assume that for each context E and type A there are infinitely many metavariables X, such that $E \vdash X : A$. The typing rules are the usual ones plus the term metavariable typing rule as given before.

Proposition 6.3 (Simulation of β-reduction for $\lambda\vec{s}$, $\lambda\vec{s}_e$, $\lambda\vec{\omega}$ and $\lambda\vec{\omega}_e$)

1. Let $a,\ b \in \Lambda$ be typed terms. If $a \to_\beta b$, then the following hold:
   (a) there exists $c \in \vec{\Lambda s}$ such that $a \to_{\vec{\sigma}-gen} c \twoheadrightarrow_{\vec{s}}\ b$.
   (b) there exists $c \in \vec{\Lambda\omega}$ such that $a \to_{\vec{\sigma}-gen} c \twoheadrightarrow_{\vec{\omega}}\ b$.
2. Let $a,\ b$ be open λ-calculus typed terms. If $a \to_\beta b$, then the following hold:
   (a) there exists $c \in \vec{\Lambda s}_{op}$ such that $a \to_{\vec{\sigma}-gen} c \twoheadrightarrow_{\vec{s}_e}\ b$.

(b) there exists $c \in \Lambda\vec{\omega}_{sop}$ such that $a \to_{\vec{\sigma}-gen} c \twoheadrightarrow_{\vec{\omega}_e} b$.

PROOF: All items can be proved by induction on the position where the reduction takes place. ⊠

7 Weak Normalization of $\lambda\vec{\omega}_e$ and $\lambda\vec{s}_e$

The main technical tool in the proof of weak normalization of $\lambda\vec{\omega}_e$ is the translation similar to the one given in [6] of typed terms into functions whose arguments are λ-terms (or lists of them) and whose results are λ-terms (or lists of them). Although the idea is the same, the translation has to be carefully adapted. Let $\mathtt{T}$ be a given type. In order to define this translation we associate every term variable $X_{E,A}$ with a variable of the λ-calculus that we denote

$$\hat{X}_{A_1 \to \ldots \to A_n \to \mathtt{T} \to A},$$

where $E = A_1, \ldots, A_n$. We also associate every substitution variable $x_{E,F}$ with a list of classical variables denoted

$$\hat{x}_{A_1 \to \ldots \to A_n \to \mathtt{T} \to B_1}; \cdots ; \hat{x}_{A_1 \to \ldots \to A_n \to \mathtt{T} \to B_m}, \hat{x}_{A_1 \to \ldots \to A_n \to \mathtt{T} \to \mathtt{T}},$$

where $F = B_1, \ldots, B_m$. The indices show the types of the associated classical variables.

The translation maps every $\lambda\vec{\omega}$-term u such that $A_1, \ldots, A_n \vdash u : A$ into a function $[\![u]\!]$ whose arguments are lists of $n+1$ terms $t_1; \ldots; t_{n+1}$ of respective types $A_1, \ldots, A_n$ and $\mathtt{T}$ and which returns a term of type A. The translation of a $\lambda\vec{\omega}$-substitution s such that $A_1, \ldots, A_n \vdash s \rhd B_1, \ldots, B_m$ is a function $[\![s]\!]$ whose arguments are lists of $n+1$ terms $t_1; \ldots; t_{n+1}$ of respective types $A_1, \ldots, A_n$ and $\mathtt{T}$ and which returns a list of $m+1$ λ-terms of types $B_1, \ldots, B_m$ and $\mathtt{T}$, respectively.

Essentially the translation reduces the term to substitution normal form, suspending substitutions on variables. Then, roughly speaking, substitution steps map to vacuous beta-reductions and $\vec{\sigma}$ $-gen$ (i.e. Beta) steps on substitution normal forms map to non-empty λ-calculus β reductions, yielding the desired result by a simulation argument.

DEFINITION 7.1
The translation $[\![\bullet]\!]$ is given as follows, where $\bar{t}$ denotes the list of terms $t_1; \ldots; t_n; t_{n+1}$ and $E = A_1, \ldots, A_n$, $F = B_1, \ldots, B_m$ and $E \to A = A_1 \to \ldots \to A_n \to \mathtt{T} \to A$.

1. $[\![X_{E,A}]\!](\bar{t}) = \hat{X}_{E \to A}\, t_1 \, \ldots \, t_{n+1}$
2. $[\![x_{E,F}]\!](\bar{t}) = (\hat{x}_{E \to B_1}\, t_1 \, \ldots \, t_{n+1}); \cdots ; (\hat{x}_{E \to B_m}\, t_1 \, \ldots \, t_{n+1}); (\hat{x}_{E \to \mathtt{T}}\, t_1 \, \ldots \, t_{n+1})$
3. $[\![\mathrm{k}_E]\!](\bar{t}) = t_k$ where $k \leq n$
4. $[\![\lambda A.u]\!](\bar{t}) = \lambda z.([\![u]\!](z; \bar{t}))$ with z fresh of type A
5. $[\![u\, v]\!](\bar{t}) = ([\![u]\!](\bar{t}))([\![v]\!](\bar{t}))$
6. $[\![u[s]_i]\!](\bar{t}) = [\![u]\!](t_1; \ldots; t_{i-1}; [\![s]\!](t_i; \ldots; t_{n+1}))$
7. $[\![\uparrow^k_E]\!](\bar{t}) = t_{k+1}; \ldots; t_{n+1}$ where $k \leq n+1$
8. $[\![u/]\!](\bar{t}) = ([\![u]\!](\bar{t})); \bar{t}$

In clause (4), z fresh means $z \notin FV(\overline{t})$ and $z \neq \hat{X}$, $\hat{x}$ for every free variable X and x in u. If we assume that our countable set of variables is ordered then we may take z as the first variable satisfying the previous conditions.

In clause (6), the arguments list $(t_1; \ldots; t_{i-1}; [\![s]\!](t_i; \ldots; t_{n+1}))$ should be interpreted as the concatenation between the list $t_1; \ldots; t_{i-1}$ and the list which results from $[\![s]\!](t_i; \ldots; t_{n+1})$.

LEMMA 7.2
If $z \neq \hat{X}$,$\hat{x}$ for every X,x in u, then $([\![u]\!](\overline{t}))[a/z] \equiv_\alpha [\![u]\!](\overline{t}[a/z])$ where by $\overline{t}[a/z]$ we mean the list $t_1[a/z]; \ldots; t_n[a/z]$ if $\overline{t} = t_1; \ldots; t_n$.

PROOF: By an easy induction on u. ⊠

The next lemma is important, stating that $[\![]\!]$ is invariant under all the rules of the substitution calculus.

LEMMA 7.3
Let $f, g \in \Lambda\overrightarrow{\omega}_{op}$, if $f \to_{\overrightarrow{\omega}_e} g$ then $[\![f]\!] = [\![g]\!]$.

PROOF: By induction on the structure of f. If the reduction is internal, use the induction hypothesis. We only give the case where $f = a[s]_i$, $g = a[s']_i$ and $s \to s'$, since the application and abstraction cases are analogous.
Let $\overline{t} = t_1; \ldots; t_{n+1}$ with the right length. Then $[\![a[s]_i]\!](\overline{t}) =$
$[\![a]\!](t_1; \ldots; t_{i-1}; [\![s]\!](t_i; \ldots; t_{n+1})) = [\![a]\!](t_1; \ldots; t_{i-1}; [\![s']\!](t_i; \ldots; t_{n+1}))$
(by the induction hypothesis)
$= [\![a[s']_i]\!](\overline{t})$.

If the reduction is at the root, then we must study each rule. In all cases, let $\overline{t} = t_1; \ldots; t_{n+1}$ with the right length.

For the $\overrightarrow{\sigma}$-$/$-*des* rule,
- if $k > j$, then $[\![k[a/]_j]\!](t_1; \ldots; t_{n+1}) =$
  $[\![k]\!](t_1; \ldots; t_{j-1}; [\![a/]\!](t_j; \ldots; t_{n+1})) =$
  $[\![k]\!](t_1; \ldots; t_{j-1}; [\![a]\!](t_j; \ldots; t_{n+1}); t_j; \ldots; t_{n+1}) =$
  $t_{k-1} = [\![k-1]\!](t_1; \ldots; t_{n+1})$
- if $k = j$, then $[\![k[a/]_j]\!](t_1; \ldots; t_{n+1}) =$
  $[\![k]\!](t_1; \ldots; t_{j-1}; [\![a/]\!](t_j; \ldots; t_{n+1})) =$
  $[\![k]\!](t_1; \ldots; t_{j-1}; [\![a]\!](t_j; \ldots; t_{n+1}); t_j; \ldots; t_{n+1}) =$
  $[\![a]\!](t_j; \ldots; t_{n+1}) =$
  $[\![a]\!]([\![\uparrow^{j-1}]\!](t_1; \ldots; t_{n+1})) =$
  $[\![a[\uparrow^{j-1}]_1]\!](t_1; \ldots; t_{n+1})$
- if $k < j$, then $[\![k[a/]_j]\!](t_1; \ldots; t_{n+1}) =$
  $[\![k]\!](t_1; \ldots; t_{j-1}; [\![a/]\!](t_j; \ldots; t_{n+1})) =$
  $t_k = [\![k]\!](t_1; \ldots; t_{n+1})$.

For the $\overrightarrow{\sigma}$-λ-*tr* rule,
$[\![(\lambda A.a)[s]_j]\!](t_1; \ldots; t_{n+1}) =$
$[\![(\lambda A.a)]\!](t_1; \ldots; t_{j-1}; [\![s]\!](t_j; \ldots; t_{n+1})) =$
$\lambda z.[\![a]\!](z; t_1; \ldots; t_{j-1}; [\![s]\!](t_j; \ldots; t_{n+1})) =$
$\lambda z.[\![a[s]_{j+1}]\!](z; t_1; \ldots; t_{n+1}) =$

$[\![\lambda A.(a[s]_{j+1})]\!](\overline{t})$.

For the $\overrightarrow{\sigma}$-*app-tr* rule,
$[\![(ab)[s]_j]\!](t_1;\ldots;t_{n+1}) =$
$[\![(ab)]\!](t_1;\ldots;t_{j-1};[\![s]\!](t_j;\ldots;t_{n+1})) =$
$[\![a]\!](t_1;\ldots;t_{j-1};[\![s]\!](t_j;\ldots;t_{n+1}))[\![b]\!](t_1;\ldots;t_{j-1};[\![s]\!](t_j;\ldots;t_{n+1})) =$
$[\![a[s]_j]\!](t_1;\ldots;t_{n+1})[\![b[s]_j]\!](t_1;\ldots;t_{n+1}) =$
$[\![a[s]_j b[s]_j]\!](t_1;\ldots;t_{n+1})$.

For the $\overrightarrow{\sigma}$-$\uparrow$-*des* rule,
- if $k \geq j$, then $[\![k[\uparrow^i]_j]\!](t_1;\ldots;t_{n+1}) =$
  $[\![k]\!](t_1;\cdots:t_{j-1}[\![\uparrow^i]\!](t_j;\ldots;t_{n+1})) =$
  $[\![k]\!](t_1;\cdots:t_{j-1};t_{j+i};\ldots;t_{n+1}) =$
  $t_{j+i+k-1-j+1} = t_{k+i} = [\![k+i]\!](t_1;\ldots;t_{n+1})$
- if $k < j$, then $[\![k[\uparrow^i]_j]\!](t_1;\ldots;t_{n+1}) =$
  $[\![k]\!](t_1;\cdots:t_{j-1}[\![\uparrow^i]\!](t_j;\ldots;t_{n+1})) =$
  $t_k = [\![k]\!](t_1;\ldots;t_{n+1})$.

For the $\overrightarrow{\sigma}$-/-*tr* rule, let $k \leq j$, $f = a[b/]_k[s]_j$ and $g = a[s]_{j+1}[b[s]_{j-k+1}/]_k$. Let $B = [\![b]\!](t_k;\ldots;t_{j-1};[\![s]\!](t_j;\ldots;t_{n+1}))$. Then
$[\![a[b/]_k[s]_j]\!](\overline{t}) =$
$[\![a[b/]_k]\!](t_1;\ldots;t_{j-1};[\![s]\!](t_j;\ldots;t_{n+1})) =$
$[\![a]\!](t_1;\ldots;t_{k-1};B;t_k;\ldots;t_{j-1};[\![s]\!](t_j;\ldots;t_{n+1})) =$
$[\![a[s]_{j+1}]\!](t_1;\ldots;t_{k-1};B;t_k;\ldots;t_{n+1}) =$
$[\![a[s]_{j+1}]\!](t_1;\ldots;t_{k-1};[\![b[s]_{j-k+1}]\!](t_k;\ldots;t_{n+1})) =$
$[\![a[s]_{j+1}[b[s]_{j-k+1}/]_k]\!](\overline{t})$.

For the /-$\uparrow$-*tr* rule,
- if $k+i \leq j$, then $[\![a[\uparrow^i]_k[b/]_j]\!](\overline{t}) =$
  $[\![a[\uparrow^i]_k]\!](t_1;\ldots;t_{j-1};[\![b]\!](t_j;\ldots;t_{n+1});t_j\ldots;t_{n+1}) =$
  $[\![a]\!](t_1;\ldots;t_{k-1};[\![\uparrow^i]\!](t_k;\ldots;t_{j-1};[\![b]\!](t_j\ldots;t_{n+1});t_j\ldots;t_{n+1})) =$
  $[\![a]\!](t_1;\ldots;t_{k-1};t_{k+i};\ldots;t_{j-1};[\![b]\!](t_j\ldots;t_{n+1});t_j\ldots;t_{n+1}) =$
  $[\![a[b/]_{j-i}]\!](t_1;\ldots;t_{k-1};t_{k+i};\ldots;t_{n+1}) =$
  $[\![a[b/]_{j-i}]\!](t_1;\ldots;t_{k-1};[\![\uparrow^i]\!](t_k;\ldots;t_{n+1})) =$
  $[\![a[b/]_{j-i}[\uparrow^i]_k]\!](\overline{t})$
- if $k \leq j < k+i$, then $[\![a[\uparrow^i]_k[b/]_j]\!](\overline{t}) =$
  $[\![a[\uparrow^i]_k]\!](t_1;\ldots;t_{j-1};[\![b]\!](t_j;\ldots;t_{n+1});t_j\ldots;t_{n+1}) =$
  $[\![a]\!](t_1;\ldots;t_{k-1};[\![\uparrow^i]\!](t_k;\ldots;t_{j-1};[\![b]\!](t_j\ldots;t_{n+1});t_j\ldots;t_{n+1})) =$
  $[\![a]\!](t_1;\ldots;t_{k-1};t_{k+i-1};\ldots;t_{n+1}) =$
  $[\![a]\!](t_1;\ldots;t_{k-1};[\![\uparrow^{i-1}]\!](t_k;\ldots;t_{n+1});t_k;\ldots;t_{n+1}) =$
  $[\![a[\uparrow^{i-1}]_k]\!](\overline{t})$.

For the $\uparrow$-$\uparrow$-*tr* rule,
- if $k+i < j$, then $[\![a[\uparrow^i]_k[\uparrow^l]_j]\!](\overline{t}) =$
  $[\![a[\uparrow^i]_k]\!](t_1;\ldots;t_{j-1};[\![\uparrow^l]\!](t_j;\ldots;t_{n+1})) =$
  $[\![a[\uparrow^i]_k]\!](t_1;\ldots;t_{j-1};t_{j+l};\ldots;t_{n+1}) =$
  $[\![a]\!](t_1;\ldots;t_{k-1};[\![\uparrow^i]\!](t_k;\ldots;t_{j-1};t_{j+l};\ldots;t_{n+1})) =$
  $[\![a]\!](t_1;\ldots;t_{k-1};t_{k+i};\ldots;t_{j-1};t_{j+l};\ldots;t_{n+1}) =$

$[\![a]\!](t_1;\ldots;t_{k-1};t_{k+i};\ldots;t_{j-i-1+i};[\![\uparrow^l]\!](t_k;\ldots;t_{n+1})) =$
$[\![a[\uparrow^l]_{j-i}]\!](t_1;\ldots;t_{k-1};[\![\uparrow^i]\!](t_k;\ldots;t_{n+1})) =$
$[\![a[\uparrow^l]_{j-i}[\uparrow^i]_k]\!](\overline{t})$

- if $k \le j \le k+i$, then $[\![a[\uparrow^i]_k[\uparrow^l]_j]\!](\overline{t}) =$
$[\![a[\uparrow^i]_k]\!](t_1;\ldots;t_{j-1};[\![\uparrow^l]\!](t_j;\ldots;t_{n+1})) =$
$[\![a]\!](t_1;\ldots;t_{k-1};[\![\uparrow^i]\!](t_k;\ldots;t_{j-1};t_{j+l};\ldots;t_{n+1})) =$
$[\![a]\!](t_1;\ldots;t_{k-1};t_{k+i+l};\ldots;t_{n+1}) =$
$[\![a]\!](t_1;\ldots;t_{k-1};[\![\uparrow^{i+l}]\!](t_k;\ldots;t_{n+1})) =$
$[\![a[\uparrow^{i+l}]_k]\!](\overline{t})$

⊠

DEFINITION 7.4
Given the terms $u, v \in \Lambda\vec{\omega}_{op}$ we say that u and v *have the same type* if both are well typed and there is a context E and type A such that $E \vdash_{\vec{\mathbf{Ls1}}} u : A$ and $E \vdash_{\vec{\mathbf{Ls1}}} v : A$.

DEFINITION 7.5
We define the quasi-order $>$ on $\Lambda\vec{\omega}_{op}$ terms as follows: $u > v$ if u and v have the same type and $[\![u]\!](\overline{t}) \overset{+}{\to}_\beta [\![v]\!](\overline{t})$ for every list of λ-terms $\overline{t}$ of the right length and the right types (to be called *right* $\overline{t}$ from now onwards).

It follows immediately that $>$ is a strict order (i.e. irreflexive and transitive), which is also compatible with taking closures as stated next.

LEMMA 7.6
1. Let $a, b \in \Lambda\, \vec{\omega}^t_{sop}$, $j \ge 1$ and $s \in \Lambda\, \vec{\omega}^s_{sop}$. If $a > b$, then $a[s]_j > b[s]_j$.
2. Let $a, b \in \Lambda\, \vec{\omega}^t_{sop}$, $i_1, \ldots, i_k \ge 1$ and $s_1, \ldots, s_k \in \Lambda\, \vec{\omega}^s_{sop}$. If $a > b$, then $a[s_1]_{i_1} \ldots [s_k]_{i_k} > b[s_1]_{i_1} \ldots [s_k]_{i_k}$.

PROOF:
1. For every right $\overline{t}$,
$[\![a[s]_j]\!](\overline{t}) = [\![a]\!](t_1, \ldots, t_{j-1}, [\![s]\!](t_j, \ldots, t_n)) \to^+_\beta [\![b]\!](t_1, \ldots, t_{j-1}, [\![s]\!](t_j, \ldots, t_n))$
$= [\![b[s]_j]\!](\overline{t})$.
2. Iterating the previous result.

⊠

The following Lemma is technically important and will be used in the proof of Lemma 7.8.

LEMMA 7.7
Let $k \ge 1$, $i_1 \ge i_2 \ge \cdots \ge i_k \ge 1$, $s_1, \ldots, s_k \in \Lambda\, \vec{\omega}^s_{sop}$ and let X be a term variable. Then for every right $\overline{t}$, there exist $r \ge 1, q_1, \ldots, q_r \in \Lambda\, \vec{\omega}^t_{sop}$ which do not depend on $t_1, \ldots, t_{i_k-1}$ such that $[\![X[s_1]_{i_1} \ldots [s_k]_{i_k}]\!](\overline{t}) = \hat{X}\ t_1\ t_2\ \ldots\ t_{i_k-1}\ q_1\ \ldots\ q_r$.

PROOF: By induction on k.
- For $k = 1$ we have:
  - $s_1 = d/$, then $[\![X[s_1]_{i_1}]\!](\overline{t}) = \hat{X}\ t_1\ \ldots\ t_{i_1-1}\ [\![d]\!](t_{i_1}, \ldots, t_n)\ t_{i_1}\ \ldots\ t_n$, thus take $(q_1, \ldots, q_r) = ([\![d]\!](t_{i_1}, \ldots, t_n), t_{i_1}, \ldots, t_n)$ which clearly do not depend on the prescribed terms.
  - $s_1 = \uparrow^m$ with $m \ge 0$, then $[\![X[s_1]_{i_1}]\!](\overline{t}) = \hat{X}\ t_1\ \ldots\ t_{i_1-1}\ t_{i_1+m}\ \ldots\ t_n$, thus take $(q_1, \ldots, q_r) = (t_{i_1+m}, \ldots, t_n)$ which do not depend on the prescribed terms either.

- For the inductive case we have:
  - $s_{k+1} = d/$, then $[\![X[s_1]_{i_1} \dots [s_k]_{i_k} [s_{k+1}]_{i_{k+1}}]\!](\bar{t})$
    $= [\![X[s_1]_{i_1} \dots [s_k]_{i_k}]\!](t_1, \dots, t_{i_{k+1}-1} [\![d]\!](t_{i_{k+1}}, \dots, t_n), t_{i_{k+1}}, \dots, t_n)$
    $= \hat{X}\ t_1\ \dots\ t_{i_{k+1}-1}\ [\![d]\!](t_{i_{k+1}}, \dots, t_n)\ t_{i_{k+1}}\ \dots\ t_{i_k-2}\ q_1\ \dots\ q_r$ (by the induction hypothesis) where $q_1, \dots, q_r$ do not depend on the previous terms in the list.
    Taking $(q'_1, \dots, q'_{r'}) = ([\![d]\!](t_{i_{k+1}}, \dots, t_n), t_{i_{k+1}}, \dots, t_{i_k-2}, q_1, \dots, q_r)$, no q'_j depends on $t_1, \dots, t_{i_{k+1}-1}$, and
    $[\![X[s_1]_{i_1} \dots [s_{k+1}]_{i_{k+1}}]\!](\bar{t}) = \hat{X}\ t_1\ t_2\ \dots\ t_{i_{k+1}-1}\ q'_1\ \dots\ q'_{r'}$.
  - $s_{k+1} = \uparrow^m$ with $m \geq 0$, then $[\![X[s_1]_{i_1} \dots [s_k]_{i_k} [s_{k+1}]_{i_{k+1}}]\!](\bar{t})$
    $= [\![X[s_1]_{i_1} \dots [s_k]_{i_k}]\!](t_1, \dots, t_{i_{k+1}-1}, t_{i_{k+1}+m}, \dots, t_n)$
    $= \hat{X}\ t_1\ \dots\ t_{i_{k+1}-1}\ t_{i_{k+1}+m}\ \dots\ t_{i_k-1+m}\ q_1\ \dots\ q_r$ (by the induction hypothesis) where $q_1, \dots, q_r$ do not depend on the previous terms in the list. Taking $(q'_1, \dots, q'_{r'}) = (t_{i_{k+1}+m}, \dots, t_{i_k-1+m}, q_1, \dots, q_r)$, no q'_j depends on $t_1, \dots, t_{i_{k+1}-1}$, and
    $[\![X[s_1]_{i_1} \dots [s_{k+1}]_{i_{k+1}}]\!](\bar{t}) = \hat{X}\ t_1\ t_2\ \dots\ t_{i_{k+1}-1}\ q'_1\ \dots\ q'_{r'}$.

⊠

Now we can give the key result, for semi-open terms.

LEMMA 7.8
Let $a, b \in \Lambda \overset{\rightarrow t}{\omega}_{sop}$ where a is an $\vec{\omega}_e$-normal form. If $a \to_{\vec{\sigma}-gen} b$ then $a > b$.

PROOF: By induction on the position of the $\vec{\sigma}-gen$ redex in the term a.
If the reduction is at the root, i.e. $a = (\lambda A.c)d$ and $b = c[d/]_1$, then
$[\![(\lambda A.c)d]\!](\bar{t}) = ([\![(\lambda A.c)]\!](\bar{t}))([\![d]\!](\bar{t})) = (\lambda z.[\![c]\!](z;\bar{t}))([\![d]\!](\bar{t})) \to_\beta [\![c]\!](z;\bar{t}))[([\![d]\!](\bar{t}))/z]$
$= [\![c]\!]([\![d]\!](\bar{t});\bar{t}) = [\![c[d/]_1]\!](\bar{t})$ (by Lemma 7.2). Remark that, when Lemma 7.2 has been used, since z should be chosen such that $z \notin FV(\bar{t})$, then $\bar{t}[([\![d]\!](\bar{t}))/z] = \bar{t}$.
Else we have the following cases:

- $a = n$ or $a = X$ a term variable, the result holds vacuously since there is no $\vec{\sigma}-gen$ redex.
- $a = cd$ then if the reduction occurs in c, say $c \to_{\vec{\sigma}-gen} c'$, we have
  $[\![a]\!](\bar{t}) = [\![c]\!](\bar{t})[\![d]\!](\bar{t}) \to^+_\beta [\![c']\!](\bar{t})[\![d]\!](\bar{t}) = [\![(c'd)]\!](\bar{t}) = [\![b]\!](\bar{t})$ using the induction hypothesis; and the situation is analogous if the reduction occurs in d.
- $a = \lambda A.c$ then the reduction occurs in c, say $c \to_{\vec{\sigma}-gen} c'$, then
  $[\![a]\!](\bar{t}) = \lambda z.[\![c]\!](\bar{t}) \to^+_\beta \lambda z.[\![c']\!](\bar{t}) = [\![\lambda A.c']\!](\bar{t}) = [\![b]\!](\bar{t})$ using the induction hypothesis.
- a is a closure, then a will necessarily have the form $d[s_1]_{i_1} \dots [s_m]_{i_m}$ where d is not a closure; d cannot be an abstraction, nor an application, nor an index (or else a would not be a nf). Then $d = X$ a term variable, and $i_1 > i_2 > \dots > i_m$ (or else a would not be a nf). Then we have that there exists $k \geq 1$ such that $a = X[s_1]_{i_1} \dots [s_{k-1}]_{i_{k-1}} [s_k]_{i_k} [s_{k+1}]_{i_{k+1}} \dots [s_m]_{i_m}$ where $s_k = e/$ with $e \to_{\vec{\sigma}-gen} e'$, for some terms e, e', and
  $a \to_{\vec{\sigma}-gen} X[s_1]_{i_1} \dots [s_{k-1}]_{i_{k-1}} [e'/]_{i_k} [s_{k+1}]_{i_{k+1}} \dots [s_m]_{i_m} = b$.

Suppose first $k \geq 2$. In what follows Lemma 7.7 will be used twice; the non dependence of the terms $q_1, \dots, q_r$ on the $i_{k-1} - 1$ terms explicited in the proof guarantees that the $q_1, \dots, q_r$ which appear after the 2nd. equality also ensure that the 3rd.

equality holds. We have that
$[\![X[s_1]_{i_1}\ldots[s_{k-1}]_{i_{k-1}}[e/]_{i_k}]\!](\overline{t})$
$= [\![X[s_1]_{i_1}\ldots[s_{k-1}]_{i_{k-1}}]\!](t_1,\ldots,t_{i_k-1},[\![e]\!](t_{i_k},\ldots,t_n),t_{i_k},\ldots,t_n)$
$= \hat{X}\ t_1\ t_2\ \ldots\ t_{i_k-1}\ [\![e]\!](t_{i_k}\ldots t_n)\ t_{i_k}\ \ldots\ t_{i_{k-1}-1}\ q_1\ \ldots\ q_r$ (by Lemma 7.7)
$\rightarrow^+_\beta \hat{X}\ t_1\ t_2\ \ldots\ t_{i_k-1}\ [\![e']\!](t_{i_k}\ldots t_n)\ t_{i_k}\ \ldots\ t_{i_{k-1}-1}\ q_1\ \ldots\ q_r$
(by the induction hypothesis and compatibility).

On the other hand,
$[\![X[s_1]_{i_1}\ldots[s_{k-1}]_{i_{k-1}}[e'/]_{i_k}]\!](\overline{t})$
$= [\![X[s_1]_{i_1}\ldots[s_{k-1}]_{i_{k-1}}]\!](t_1,\ldots,t_{i_k-1},[\![e']\!](t_{i_k},\ldots,t_n),t_{i_k},\ldots,t_n)$
$= \hat{X}\ t_1\ t_2\ \ldots\ t_{i_k-1}\ [\![e']\!](t_{i_k}\ldots t_n)\ t_{i_k}\ \ldots\ t_{i_{k-1}-1}\ q'_1\ \ldots\ q'_r$
and because of the fact that $q'_1,\ldots,q'_r$ do not depend on the terms before q'_1, we have that $q_i = q'_i$ for $1 \le i \le r$.

Thus, by Lemma 7.6(2),
$[\![a]\!](\overline{t}) = [\![X[s_1]_{i_1}\ldots[s_{k-1}]_{i_{k-1}}[e/]_{i_k}[s_{k+1}]_{i_{k+1}}\ldots[s_m]_{i_m}]\!](\overline{t})$
$\rightarrow^+_\beta [\![X[s_1]_{i_1}\ldots[s_{k-1}]_{i_{k-1}}[e'/]_{i_k}[s_{k+1}]_{i_{k+1}}\ldots[s_m]_{i_m}]\!](\overline{t}) = [\![b]\!](\overline{t})$.

The case $k = 1$ follows directly by Definition 7.1:
$[\![X[e/]_{i_1}]\!](\overline{t})$
$= \hat{X}\ t_1\ t_2\ \ldots\ t_{i_1-1}\ [\![e]\!](t_{i_1}\ldots t_n)\ t_{i_1}\ \ldots\ t_n$
$\rightarrow^+_\beta \hat{X}\ t_1\ t_2\ \ldots\ t_{i_1-1}\ [\![e']\!](t_{i_1}\ldots t_n)\ t_{i_1}\ \ldots\ t_n$
(by the induction hypothesis and compatibility)
$= [\![X[e'/]_{i_1}]\!](\overline{t})$ and use Lemma 7.6(2) similarly.

⊠

Even though it is an adaptation of the technique in [6], the technical result in Lemma 7.7 was required. In Lemma 7.8 special care is necessary for handling closures. For the case that a is a closure, if it were any open term (i.e. having substitution variables), it might have a more complicated form than just $d[s_1]_{i_1}\ldots[s_m]_{i_m}$ with the conditions above mentioned, and here is where the method would not work. The condition $i_1 > \cdots > i_m$ becomes strictly necessary, otherwise the β-redex could be lost in the $[\![\bullet]\!]$ transform. The result does not hold for full open terms, taking for instance the normal form $a = 2[x]_2[d/]_1$ as a counterexample, for $d \rightarrow_{\vec{\sigma}-gen} d'$, since $a \rightarrow_{\vec{\sigma}-gen} 2[x]_2[d'/]_1$ but
$[\![a]\!](\overline{t}) = [\![2[x]_2]\!]([\![d/]\!](t_1;\ldots;t_n)) =$
$[\![2[x]_2]\!]([\![d]\!](t_1;\ldots;t_n);t_1;\ldots;t_n) =$
$[\![2]\!]([\![d]\!](t_1;\ldots;t_n);[\![x]\!](t_1;\ldots;t_n)) =$
$[\![2]\!]([\![d]\!](t_1;\ldots;t_n);\hat{x}t_1\ldots t_n;\ldots;\hat{x}t_1\ldots t_n) =$
$\hat{x}t_1\ldots t_n$
which may not have β-redexes. This counterexample is critical. So far, Lemma 7.8 could not be extended for the full open term set using the same technique; it does not seem that some simple or intuitive change in the $[\![\bullet]\!]$ function definition could help. Problems also are caused by terms like $1[x]_2$, which could be partially fixed redefining the $\lambda\vec{\omega}_e$-calculus in such a way that terms of the form $k[x]_j$ for $k < j$ would not be in normal form. Recall that if $a, b \in \Lambda\vec{\omega}$ are two typed terms where a is an

$\vec{\omega_e}$-normal form, we need to certify that, when $a \rightarrow_{\vec{\sigma}-gen} b$, then for every right $\bar{t}$, $[\![a]\!](\bar{t}) \overset{+}{\rightarrow}_\beta [\![b]\!](\bar{t})$, i.e. the possibility of β reduction in the simply typed λ-calculus must be created. For this reason we could add to $\lambda\vec{\omega_e}$ the following rule:

$$(\vec{\sigma} - des) \quad n[s]_j \quad \rightarrow \quad n \quad if\ n < j$$

This rule subsumes rules ($\vec{\sigma}$-/-des) and ($\vec{\sigma}$-$\uparrow$-des) for the case $n < j$, and it is consistent with Lemma 7.3. The addition of this rule forces terms like $n[x]_j$ for $n < j$ not to be $\vec{\omega_e}$-normal forms. But note that terms having closures with substitution variables are not in the domain of the translation S.

Nevertheless, the addition of this new rule does not fix the problem for examples like a above, since the condition is $n = j$ and there is no similar rule to be added in order to force $n[x]_n$ to reduce.

In virtue of the main result in [6], a natural question is: what happens in $\lambda\omega_e$ which can make the difference with $\lambda\sigma$, since the latter is WN on all typed open terms? The reason we found is that, when the counterexample is translated to $\lambda\sigma$, it is not a σ-normal form and hence it does not represent a counterexample. Remark that the statement of Lemma 7.8 is for normal forms. More precisely, $a = 2[x]_2[d/]_1$ translates to the $\lambda\sigma$ term $1[\uparrow][1.(x \circ \uparrow)][d.id]$ (and to the $\lambda\sigma_\Uparrow$ term $1[\uparrow][\Uparrow (x)][d.id]$), which is clearly not a σ-normal form.

The problem forced the statement of Lemma 7.8 to refer to semi-open terms. It is worth mentioning, however, that due to the above isomorphism it suffices to consider this restricted set of terms in order to obtain WN of $\lambda\vec{s_e}$. This was our original goal.

Definition 7.9
A *canonical strategy* for $\lambda\vec{\omega}_e$ is a strategy which applies the $\vec{\sigma}$ $-gen$ rule only to $\vec{\sigma}$ $-gen$-redexes in $\vec{\omega_e}$-normal forms and whose $\vec{\omega}_e$-reductions are normalizing.

As an example, take a strategy which applies the $\vec{\sigma}$ $-gen$ rule only to $\vec{\omega_e}$-normal forms and whose $\vec{\omega}_e$-reductions are leftmost-innermost (li). Hence, given a term a_1, such a canonical reduction sequence will be:

$$a_1 \overset{li}{\twoheadrightarrow}_{\vec{\omega}_e} \vec{\omega}_e\ (a_1) \rightarrow_{\vec{\sigma}-gen} a_2 \overset{li}{\twoheadrightarrow}_{\vec{\omega}_e} \vec{\omega}_e\ (a_2) \rightarrow_{\vec{\sigma}-gen} \cdots$$

where $\overset{li}{\twoheadrightarrow}_{\vec{\omega}_e}$ stands for li $\vec{\omega}_e$-reduction and $\vec{\omega}_e\ (a_i)$ is the $\vec{\omega}_e$-normal form of a_i.

Theorem 7.10
Every canonical strategy for $\lambda\vec{\omega}_e$ is strongly normalizing and therefore the $\lambda\vec{\omega}_e$-calculus is WN for semi-open terms.

Proof: If there is an infinite reduction sequence

$$a_1 \twoheadrightarrow_{\vec{\omega}_e} \vec{\omega}_e\ (a_1) \rightarrow_{\vec{\sigma}-gen} a_2 \twoheadrightarrow_{\vec{\omega}_e} \vec{\omega}_e\ (a_2) \rightarrow_{\vec{\sigma}-gen} \cdots$$

then by Lemmas 7.3 and 7.8, for every right $\bar{t}$, we get a contradiction through the infinite reduction sequence in the typed λ-calculus:

$$[\![a_1]\!](\bar{t}) = [\![\vec{\omega}_e\ (a_1)]\!](\bar{t}) \rightarrow_\beta^+ [\![a_2]\!](\bar{t}) = [\![\vec{\omega}_e\ (a_2)]\!](\bar{t}) \rightarrow_\beta^+ \cdots$$

⊠

Now, the isomorphism presented in Section 3, gives:

THEOREM 7.11
The $\lambda\vec{s}_e$-calculus is weakly normalizing for open terms.

8 The $\lambda\omega'_e$-calculus.

We extend the previous result to a new calculus, $\lambda\vec{\omega}'_e$, derived from $\lambda\vec{\omega}_e$. In this section we omit typing decorations for notation simplicity, therefore $\lambda\vec{\omega}'_e$ will be just written $\lambda\omega'_e$.

$\lambda\omega'_e$ is written in the style of $\lambda\sigma$ and has 1 as the sole de Bruijn index, while the others are constructed as in $\lambda\sigma$. We will show that typed $\lambda\omega'_e$ is WN on semi-open terms.

A good reason to use this calculus is to show the power of the composition rules, which indeed emulates the behavior of the other indices. Thus with a smaller language one will have in some sense the same reduction possibilities.

Remark that the problem which forced us to restrict Lemma 7.8 still holds. Up to now, we do not know whether typed $\lambda\omega'_e$ is WN on all open terms.

DEFINITION 8.1
The set of open terms and substitutions of the $\lambda\omega'_e$-calculus, noted $\Lambda\omega'_{op}$, is defined as $\Lambda^t\omega'_{op} \cup \Lambda^s\omega'_{op}$, where $\Lambda^t\omega'_{op}$ and $\Lambda^s\omega'_{op}$ are mutually defined as follows:

$$\begin{array}{lcll} \Lambda^t\omega'_{op} & ::= & \mathbf{V} \mid 1 \mid \lambda\Lambda^t\omega'_{op} \mid \Lambda^t\omega'_{op}\Lambda^t\omega'_{op} \mid \Lambda^t\omega'_{op}[\Lambda^s\omega'_{op}]_j & j \geq 1 \\ \Lambda^s\omega'_{op} & ::= & \mathbf{W} \mid \uparrow^k \mid \Lambda^t\omega'_{op}/ & k \geq 0 \end{array}$$

and the set of semi-open terms and substitutions of the $\lambda\omega'_e$-calculus is defined as $\Lambda^t\omega'_{sop} \cup \Lambda^s\omega'_{sop}$, where $\Lambda^t\omega'_{sop}$ and $\Lambda^s\omega'_{sop}$ are mutually defined as follows:

$$\begin{array}{lcll} \Lambda^t\omega'_{sop} & ::= & \mathbf{V} \mid 1 \mid \lambda\Lambda^t\omega'_{sop} \mid \Lambda^t\omega'_{sop}\Lambda^t\omega'_{sop} \mid \Lambda^t\omega'_{sop}[\Lambda^s\omega'_{sop}]_j & j \geq 1 \\ \Lambda^s\omega'_{sop} & ::= & \uparrow^k \mid \Lambda^t\omega'_{sop}/ & k \geq 0 \end{array}$$

The rules of the $\lambda\omega'_e$-calculus are given in Figure 8

As with $\lambda\omega_e$, all rules except *(σ-gen')* conform ω'_e.

Note that the *(σ-des')* rule, in the presence of the *(σ-/-des')* rule, subsumes the following two possible rules:

$$\begin{array}{lll} (\sigma - / - des'') & 1[a/]_j & \to \left\{ \begin{array}{ll} a[\uparrow^0]_1 & j = 1 \\ 1 & j > 1 \end{array} \right. \\ (\sigma - \uparrow - des'') & 1[\uparrow^i]_j & \to \quad 1 \;\; j > 1 \end{array}$$

Note also that for all $i \geq 1$, the term $1[\uparrow^i]_1$ is an ω'_e-normal form representing the de Bruijn index $i+1$. In fact, $1[\uparrow^i]_1[\uparrow^l]_1 \to_{\omega'_e} 1[\uparrow^{i+l}]_1$.

Now we wish to relate $\lambda\omega'_e$ and $\lambda\omega_e$ by means of a translation.

$$\begin{array}{llcll}
(\sigma\text{-}gen') & (\lambda a)b & \longrightarrow & a[b/]_1 & \\
(\sigma\text{-}app\text{-}tr') & (ab)[s]_j & \longrightarrow & a[s]_j b[s]_j & \\
(\sigma\text{-}\lambda\text{-}tr') & (\lambda a)[s]_j & \longrightarrow & \lambda(a[s]_{j+1}) & \\
(\sigma\text{-}/\text{-}des') & 1[a/]_1 & \longrightarrow & a[\uparrow^0]_1 & \\
(\sigma\text{-}\uparrow\text{-}des') & 1[\uparrow^0]_1 & \longrightarrow & 1 & \\
(\sigma\text{-}des') & 1[s]_j & \longrightarrow & 1 & j > 1 \\
(\sigma\text{-}/\text{-}tr') & a[b/]_k[s]_j & \longrightarrow & a[s]_{j+1}[b[s]_{j-k+1}/]_k & k \leq j \\
(/\text{-}\uparrow\text{-}tr') & a[\uparrow^i]_k[b/]_j & \longrightarrow & \begin{cases} a[b/]_{j-i}[\uparrow^i]_k & k+i \leq j \\ a[\uparrow^{i-1}]_k & k \leq j < k+i \end{cases} & \\
(\uparrow\text{-}\uparrow\text{-}tr') & a[\uparrow^i]_k[\uparrow^l]_j & \longrightarrow & \begin{cases} a[\uparrow^l]_{j-i}[\uparrow^i]_k & k+i < j \\ a[\uparrow^{i+l}]_k & k \leq j \leq k+i \end{cases} &
\end{array}$$

FIG. 8. The rewriting rules of the simply typed $\lambda\omega'_e$-calculus

DEFINITION 8.2
For open terms and substitutions we define a translation $|\bullet| : \Lambda\omega_{op} \to \Lambda\omega'_{op}$ by:

$$\begin{array}{lcllcl}
|X| & = & X & |x| & = & x \\
|1| & = & 1 & |n+1| & = & 1[\uparrow^n]_1 \quad (n \geq 1) \\
|\lambda a| & = & \lambda|a| & |ab| & = & |a||b| \\
|\uparrow^k| & = & \uparrow^k & |a/| & = & |a|/ \\
|a[s]_j| & = & |a|[|s|]_j & & &
\end{array}$$

Note that the translation of an index greater than 1 yields a term of the form $1[\uparrow^n]$, while 1 is translated as 1.

We give a Simulation Proposition which will be used in the subsequent results of the section.

PROPOSITION 8.3 (Simulation)
Let $a, b \in \Lambda\omega_{op}$.

1. If $a \to_{\sigma-gen} b$, then $|a| \to_{\sigma-gen'} |b|$.
2. If $a \to_{\omega_e} b$, then $|a| \twoheadrightarrow_{\omega'_e} |b|$.
3. If $a \twoheadrightarrow_{\omega_e} b$, then $|a| \twoheadrightarrow_{\omega'_e} |b|$.
4. If $a \twoheadrightarrow_{\lambda\omega_e} b$, then $|a| \twoheadrightarrow_{\lambda\omega'_e} |b|$.

PROOF:
1. By induction on a. If the reduction is at the root where $(\lambda c)d \to_{\sigma-gen} c[d/]_1$, then $|a| = (\lambda|c|)|d| \to_{\sigma-gen'} |c|[|d|/]_1 = |b|$. For internal reductions, the proof is straightforward.
2. By induction on a. If the reduction is at the root, we analyze every possible ω_e-rule applied.
   - $a = n[c/]_j \to_{\sigma-/-des} n-1 = b$ with $n > j \geq 1$, then $n \geq 2$. If $n > 2$, then $|a| = |n|[|c|/]_j = 1[\uparrow^{n-1}]_1[|c|/]_j \to_{/-\uparrow-tr'} 1[\uparrow^{n-2}]_1 = |n-1| = |b|$, since $1 = k \leq j < k+i = n > 2$. If $n = 2$, $1[\uparrow^{n-2}]_1 = 1[\uparrow^0]_1 \to_{\sigma-\uparrow-des'} 1 = |1| = |b|$.

- $a = n[c/]_j \to_{\sigma-/-des} c[\uparrow^{j-1}]_1 = b$ with $n = j \geq 1$.
  If $n = 1$, then $|n[c/]_j| = 1[|c|/]_j \to_{\sigma-/-des'} |c|[\uparrow^0]_1 = |c[\uparrow^0]_1|$ and we are done.
  If $n > 1$, then $|a| = |n|[|c|/]_j = 1[\uparrow^{n-1}]_1[|c|/]_j \to_{/-\uparrow-tr'} 1[|c|/]_{j-n+1}[\uparrow^{n-1}]_1$
  $\to_{\sigma-\uparrow-des'} |c|[\uparrow^0]_1[\uparrow^{n-1}]_1 \to_{\uparrow-\uparrow-tr'} |c|[\uparrow^{n-1}]_1 = |b|$ since
  $1 = k \leq k + i = n = j$ thus $j - n + 1 = 1$.
- $a = n[c/]_j \to_{\sigma-/-des} n = b$ with $1 \leq n < j$. If $n = 1$, we are done by rule (σ-/-*des*$'$) since $j > 1$. Else, $|a| = |n|[|c|/]_j = 1[\uparrow^{n-1}]_1[|c|/]_j \to_{/-\uparrow-tr'} 1[|c|/]_{j-n+1}[\uparrow^{n-1}]_1 \to_{\sigma-des'} 1[\uparrow^{n-1}]_1 = |b|$, since $j > n = k + i \leq j$ thus $j - n + 1 > 1$.
- $a = n[\uparrow^l]_j \to_{\sigma-\uparrow-des} n + l = b$, with $n \geq j \geq 1$. We have the following cases
  If $n = 1$ (thus $j = 1$) and $l = 0$, then $|a| = 1[\uparrow^0]_1 \to_{\sigma-\uparrow-des'} 1 = |b|$.
  If $n = 1$ (thus $j = 1$) and $l > 0$, then $|a| = 1[\uparrow^l]_1 = |1 + l| = |b|$
  If $n \geq 2$ then $|a| = |n|[\uparrow^l]_j = 1[\uparrow^{n-1}]_1[\uparrow^l]_j \to_{\uparrow-\uparrow-tr'} 1[\uparrow^{n+l-1}]_1 = |n + l| = |b|$
  since $1 = k \leq j \leq k + i = n$.
- $a = n[\uparrow^l]_j \to_{\sigma-\uparrow-des} n = b$ with $1 \leq n < j$. We have the following cases:
  If $n = 1$, then $|a| = 1[\uparrow^l]_j \to_{\sigma-des'} 1 = |b|$ since $j > 1$.
  If $n \geq 2$ then $|a| = 1[\uparrow^{n-1}]_1[\uparrow^l]_j \to_{\uparrow-\uparrow-tr'} 1[\uparrow^l]_{j-n+1}[\uparrow^{n-1}]_1 \to_{\sigma-des'} 1[\uparrow^{n-1}]_1$
  $= |n| = |b|$ since $1 \leq n < j$ thus $j - n + 1 > 1$.
- The other rules are straightforward.

For internal reductions, the proof is straightforward.

3. Consequence of the second item.
4. Consequence of the previous items.

⊠

As it can be seen in the proof of Proposition 8.3, the ω'_e-rules (σ-/-tr'), (/-$\uparrow$-tr') and ($\uparrow$-$\uparrow$-tr') can handle closures over indices thus simulating the behavior of the ω_e-rules (σ-/-des) and (σ-$\uparrow$-des).

Remark 8.4
Let $u, v \in \Lambda\omega'_{sop}$.

1. If $u \to_{\lambda\omega'_e} v$, then $u \to_{\lambda\omega_e} v$.
2. $|u| = u$.

Proof: Both 1. and 2. can be proved by an easy induction on u. ⊠

The second assertion above means that the translation is onto and invariant for the set $\Lambda\omega'_{sop}$.

Corollary 8.5 (Confluence)
$\lambda\omega'_e$ and ω'_e are confluent on semi-open terms.

Proof: To prove the confluence of $\lambda\omega'_e$, let $a \in \Lambda\omega'_{sop}$, and suppose $a \twoheadrightarrow_{\lambda\omega'_e} a_1$, $a \twoheadrightarrow_{\lambda\omega'_e} a_2$. By Remark 8.4, both derivations are also $\lambda\omega_e$-derivations. Since $\lambda\omega_e$ is isomorphic to λs_e, it is confluent on semi-open terms [8], thus there exists $b \in \Lambda\omega_{sop}$ such that $a_1 \twoheadrightarrow_{\lambda\omega_e} b$ and $a_2 \twoheadrightarrow_{\lambda\omega_e} b$. By the fourth item of the Simulation Proposition, $|a_1| \twoheadrightarrow_{\lambda\omega'_e} |b|$ and $|a_2| \twoheadrightarrow_{\lambda\omega'_e} |b|$. Since $a = |a|$ by Remark 8.4, this closes the diagram.
The confluence of ω'_e is proved analogously, by using the third item of the Simulation Proposition. ⊠

We define the typing rules of $\lambda\omega'_e$ in a straightforward manner analogously to $\overset{\rightarrow}{\lambda\omega_e}$. Moreover, we have:

LEMMA 8.6 (Typability preservation)
For all $a \in \Lambda\omega'_{sop}$, if a is typed in $\lambda\omega'_e$, then a is typed in $\lambda\omega_e$.

PROOF: By induction on a. ⊠

8.1 *Weak normalization of typed $\lambda\omega'_e$*

In order to prove WN of typed $\lambda\omega'_e$, we will relate the $\lambda\omega_e$-calculus with the $\lambda\omega'_e$-calculus. We first give a grammar for the set of $\lambda\omega_e$ open terms in ω_e-normal form and another grammar for the set of $\lambda\omega_e$ open terms in $\lambda\omega_e$-normal form. These grammars will specify conditions associated to some of their rules (strictly speaking, they can be seen as grammar schemas or conditional grammars.)

We denote with NF_{ω_e}, $NF_{\lambda\omega_e}$, $NF_{\omega'_e}$ and $NF_{\lambda\omega'_e}$ the sets of normal forms of the respective calculi untyped open terms.

DEFINITION 8.7
We call ω_e-syntactic normal forms the terms NS_{ω_e} generated by the following syntax with start symbol M:

$$
\begin{array}{lll l}
M & ::= & M_1 \dots M_n \mid c \mid \lambda M & \text{where } n \geq 1 \\
c & ::= & c_1 \mid c_2 & \\
c_1 & ::= & m[s_1]_{i_1} \cdots [s_n]_{i_n} & \text{where } m \geq 1, n \geq 0, \forall 1 \leq k \leq n, i_k \geq 1, \\
 & & & \forall 1 \leq k < n, (i_k \leq i_{k+1} \Rightarrow (s_k \in \mathbf{W} \ \textit{or} \ (s_k = \uparrow^t \ \textit{and} \ s_{k+1} \in \mathbf{W}))), \\
 & & & n \geq 1 \Rightarrow s_1 \in \mathbf{W} \\
c_2 & ::= & X[s_1]_{i_1} \cdots [s_n]_{i_n} & \text{where } n \geq 0, \forall 1 \leq k < n, i_k \geq 1, \\
 & & & \forall 1 \leq k < n, (i_k \leq i_{k+1} \Rightarrow (s_k \in \mathbf{W} \ \textit{or} \ (s_k = \uparrow^t \ \textit{and} \ s_{k+1} \in \mathbf{W}))) \\
s & ::= & M/ \mid x \mid \uparrow^k & \text{where } k \geq 0
\end{array}
$$

DEFINITION 8.8
We call $\lambda\omega_e$-syntactic normal forms the terms $NS_{\lambda\omega_e}$ generated by the following syntax with start symbol N:

$$
\begin{array}{lll l}
N & ::= & cN_1 \dots N_n \mid \lambda N & \text{where } n \geq 0 \\
c & ::= & c_1 \mid c_2 & \\
c_1 & ::= & m[s_1]_{i_1} \cdots [s_n]_{i_n} & \text{where } m \geq 1, n \geq 0, \forall 1 \leq k \leq n, i_k \geq 1, \\
 & & & \forall 1 \leq k < n, (i_k \leq i_{k+1} \Rightarrow (s_k \in \mathbf{W} \ \textit{or} \ (s_k = \uparrow^t \ \textit{and} \ s_{k+1} \in \mathbf{W}))), \\
 & & & n \geq 1 \Rightarrow s_1 \in \mathbf{W} \\
c_2 & ::= & X[s_1]_{i_1} \cdots [s_n]_{i_n} & \text{where } n \geq 0, \forall 1 \leq k < n, i_k \geq 1, \\
 & & & \forall 1 \leq k < n, (i_k \leq i_{k+1} \Rightarrow (s_k \in \mathbf{W} \ \textit{or} \ (s_k = \uparrow^t \ \textit{and} \ s_{k+1} \in \mathbf{W}))) \\
s & ::= & N/ \mid x \mid \uparrow^k & \text{where } k \geq 0
\end{array}
$$

LEMMA 8.9
The ω_e-syntactic normal forms are exactly the ω_e-normal forms.

PROOF: We prove $NS_{\omega_e} \subseteq NF_{\omega_e}$ by checking that in each clause no rhs term contains any ω_e-redex.
Now we prove $NF_{\omega_e} \subseteq NS_{\omega_e}$. Let $t \in NF_{\omega_e}$. We prove that $t \in NS_{\omega_e}$ by induction on t. If $t = n$, it is clear. The same if $t = X$, $t = \lambda b$ or $t = t_1 t_2$. If t is a closure, then let $t = u[s_1]_{\imath_1} \dots [s_n]_{\imath_n}$, where u is not a closure. Then u cannot be λv, nor $t_1 t_2$, otherwise t would not be an ω_e-nf. It can only be a de Bruijn index or a metavariable. In either case, t is generated by the c_1 or c_2 clause respectively, and in each case the conditions should hold or else t would not be an ω_e-nf. ⊠

LEMMA 8.10
The $\lambda\omega_e$-syntactic normal forms are exactly the $\lambda\omega_e$-normal forms.

PROOF: The proofs of both inclusions are analogous to the ones given in the previous Lemma. ⊠

We also give grammars for the set of $\lambda\omega'_e$ open terms in ω'_e-normal form and for the set of $\lambda\omega'_e$ open terms in $\lambda\omega'_e$-normal form, specifying conditions in some of their rules.

DEFINITION 8.11
We call ω'_e-syntactic normal forms the terms $NS_{\omega'_e}$ generated by the following syntax with start symbol M:

$$\begin{array}{lcll}
M & ::= & M_1 \ldots M_n \mid c \mid \lambda M & \text{where } n \geq 1 \\
c & ::= & c_1 \mid c_2 & \\
c_1 & ::= & 1[s_1]_{i_1} \ldots [s_n]_{i_n} & \text{where } n \geq 0, \forall 1 \leq k \leq n, i_k \geq 1, \\
 & & & \forall 1 \leq k < n, (i_k \leq i_{k+1} \Rightarrow (s_k \in \mathbf{W} \textit{ or } (s_k = \uparrow^t \textit{ and } s_{k+1} \in \mathbf{W}))), \\
 & & & n \geq 1 \Rightarrow (i_1 = 1 \textit{ and } (s_1 \in \mathbf{W} \textit{ or } s_1 = \uparrow^t,\ t > 0)) \\
c_2 & ::= & X[s_1]_{i_1} \ldots [s_n]_{i_n} & \text{where } n \geq 0, \forall 1 \leq k \leq n, i_k \geq 1, \\
 & & & \forall 1 \leq k < n, (i_k \leq i_{k+1} \Rightarrow (s_k \in \mathbf{W} \textit{ or } (s_k = \uparrow^t \textit{ and } s_{k+1} \in \mathbf{W}))) \\
s & ::= & M/ \mid x \mid \uparrow^k & \text{where } k \geq 0
\end{array}$$

DEFINITION 8.12
We call $\lambda\omega'_e$-syntactic normal forms the terms $NS_{\lambda\omega'_e}$ generated by the following syntax with start symbol N:

$$\begin{array}{lcll}
N & ::= & cN_1 \ldots N_n \mid \lambda N & \text{where } n \geq 0 \\
c & ::= & c_1 \mid c_2 & \\
c_1 & ::= & 1[s_1]_{i_1} \ldots [s_n]_{i_n} & \text{where } n \geq 0, \forall 1 \leq k \leq n, i_k \geq 1, \\
 & & & \forall 1 \leq k < n, (i_k \leq i_{k+1} \Rightarrow (s_k \in \mathbf{W} \textit{ or } (s_k = \uparrow^t \textit{ and } s_{k+1} \in \mathbf{W}))), \\
 & & & n \geq 1 \Rightarrow (i_1 = 1 \textit{ and } (s_1 \in \mathbf{W} \textit{ or } s_1 = \uparrow^t,\ t > 0)) \\
c_2 & ::= & X[s_1]_{i_1} \ldots [s_n]_{i_n} & \text{where } n \geq 0, \forall 1 \leq k \leq n, i_k \geq 1, \\
 & & & \forall 1 \leq k < n, (i_k \leq i_{k+1} \Rightarrow (s_k \in \mathbf{W} \textit{ or } (s_k = \uparrow^t \textit{ and } s_{k+1} \in \mathbf{W}))) \\
s & ::= & N/ \mid x \mid \uparrow^k & \text{where } k \geq 0
\end{array}$$

LEMMA 8.13
The ω'_e-syntactic normal forms are exactly the ω'_e-normal forms.

PROOF: We prove $NS_{\omega'_e} \subseteq NF_{\omega'_e}$ by checking that in each clause no rhs term contains any ω'_e-redex.
Now we prove $NF_{\omega'_e} \subseteq NS_{\omega'_e}$. Let $t \in NF_{\omega'_e}$. We prove that $t \in NS_{\omega'_e}$ by induction on t. If $t = 1$, it is clear. The same if $t = X$, $t = \lambda b$ or $t = t_1 t_2$. If t is a closure, then let $t = u[s_1]_{i_1} \ldots [s_n]_{i_n}$, where u is not a closure. Then u cannot be λv, nor $t_1 t_2$, otherwise t would not be an ω_e-nf. It can only be 1 or a metavariable. In either case, t is generated by the c_1 or c_2 clause respectively, and in each case the conditions should hold or else t would not be an ω'_e-nf. ⊠

LEMMA 8.14
The $\lambda\omega'_e$-syntactic normal forms are exactly the $\lambda\omega'_e$-normal forms.

PROOF: The proofs of both inclusions are analogous to the ones given in the previous Lemma. ⊠

Remark that all these grammars generate all the respective normal forms including untypable normal forms (eg. such as 11).

LEMMA 8.15
If $a \in NF_{\omega_e}$ then $|a| \in NF_{\omega'_e}$.

PROOF: We use induction on a. In virtue of Lemma 8.9, we have the following cases:
1. if $a = M_1 \ldots M_n$ or $a = \lambda M$, i.e. the M clause was used, it is straightforward, since $|a| = |M_1| \ldots |M_n|$ or $|a| = \lambda |M|$, so in both cases $|a|$ has no internal ω_e-redexes by the induction hypothesis.
2. analogous for the $c2$ clause.
3. analogous for the s clause.
4. for the $c1$ clause, a will have the form $m[s_1]_{i_1} \ldots [s_n]_{i_n}$ where the mentioned conditions hold. Then:
   - If $n = 0$, then
     (a) either $m = 1$, then $|a| = 1$ which is an ω'_e-normal form
     (b) or $m \geq 2$, then $|a| = 1[\uparrow^{m-1}]_1$ which is also an ω'_e-normal form.
   - Else $n \geq 1$, then $s_1 \in \mathbf{W}$, and we have two cases:
     (a) if $m = 1$, $|a| = 1[|s_1|]_{i_1} \ldots [|s_n|]_{i_n}$, and since $|s_1| = s_1 \in \mathbf{W}$, there are no ω'_e-redexes by the induction hypothesis, thus $|a| \in NF_{\omega'_e}$
     (b) if $m \geq 2$, $|a| = 1[\uparrow^{m-1}]_1[|s_1|]_{i_1} \ldots [|s_n|]_{i_n}$, and since $|s_1| = s_1 \in \mathbf{W}$, there are no ω'_e-redexes by the induction hypothesis, thus $|a| \in NF_{\omega'_e}$.

⊠

LEMMA 8.16
If $a \in NF_{\lambda\omega_e}$ then $|a| \in NF_{\lambda\omega'_e}$.

PROOF: We use induction on a. In virtue of Lemma 8.10, we have the following cases:
1. if $a = cN_1 \ldots N_n$ or $a = \lambda N$, i.e. the N clause was used, it is straightforward, since $|a| = |c||N_1| \ldots |N_n|$ or $|a| = \lambda |N|$, so in both cases $|a|$ has no internal $\lambda\omega_e$-redexes by the induction hypothesis.

Cases 2., 3. and 4. are analogous to items 2., 3. and 4. of the previous lemma. ⊠

COROLLARY 8.17 (Weak normalization of typed ω'_e)
Typed ω'_e is weakly normalizing for semi-open terms.

PROOF: Let $a \in \Lambda\omega'_{sop}$ be a typed semi-open term. By Theorem 5.3, a has an ω_e-normal form $\overrightarrow{\omega_e}(a)$. By Simulation and Remark 8.4, $a = |a| \twoheadrightarrow_{\omega'_e} |\overrightarrow{\omega_e}(a)|$. Last, $|\overrightarrow{\omega_e}(a)|$ is an ω'_e-normal form by Lemma 8.15. ⊠

We will state a necessary result about li-strategies:

LEMMA 8.18 (leftmost-innermost character preservation)
Via simulation, every li-strategy applied to a term $a \in \Lambda\omega_{sop}$ projects into a li-strategy applied to the term $|a| \in \Lambda\omega'_{sop}$.

PROOF: We can prove that if $a \rightarrow_{\omega_e} b$ is a li-step, then $|a| \twoheadrightarrow_{\omega'_e} |b|$ is a sequence of li-steps, by induction on the position where the reduction takes place. As an illustration, we analyze the case of the *σ-/-des* rule for the case $n = j$:
$n[a/]_n \rightarrow_{\omega_e} a[\uparrow^{n-1}]_1$, so we suppose $a \in NF_{\omega_e}$ since this is a li-step, then:
$|n[a/]_n| = 1[\uparrow^{n-1}]_1[|a|/]_n$
$\rightarrow_{\omega'_e} 1[|a|/]_1[\uparrow^{n-1}]_1$
$\rightarrow_{\omega'_e} |a|[\uparrow^0]_1[\uparrow^{n-1}]_1$
$\rightarrow_{\omega'_e} |a|[\uparrow^{n-1}]_1 = |a[\uparrow^{n-1}]_1|$.

Note that all $\rightarrow_{\omega'_e}$ steps in this sequence are li, in particular the second and third steps are li because $|a| \in NF_{\omega'_e}$ by Lemma 8.15 thus it does not contain ω'_e-redexes.

The rest of the cases require similar or less considerations. ⊠

Combining the previous lemmas and Theorem 7.10 we get

THEOREM 8.19 (Weak normalization of typed $\lambda\omega'_e$)
Every canonical strategy for $\lambda\omega'_e$ with li ω'_e-steps is strongly normalizing and therefore the simply typed $\lambda\omega'_e$-calculus is WN for semi-open terms.

PROOF: Let $a \in \Lambda\omega'_{sop}$. Then, since $\Lambda\omega'_{sop} \subset \Lambda\omega_{sop}$, by Theorem 7.10 there exists $b \in NF_{\lambda\omega_e}$ such that $a \twoheadrightarrow_{\lambda\omega_e} b$ and this derivation is a canonical strategy. Then by Remark 8.4 and Lemma 8.18, we have that $a = |a| \twoheadrightarrow_{\lambda\omega'_e} |b|$ and this derivation is a canonical strategy, and by Lemma 8.16 $|b| \in NF_{\lambda\omega'_e}$, thus $a \in WN_{\lambda\omega'_e}$. ⊠

9 Conclusion

The main purpose of this paper is to present a proof of weak normalization for simply typed λs_e, inspired by the technique of [6] for proving weak normalization of simply typed $\lambda\sigma$. We prove not only that typed terms are WN but we also give a strategy for reaching the normal forms.

A main feature to emphasize is that the behavior of $\lambda\omega_e$ differs when analyzing weak normalization of open and semi-open terms. The same applies to $\lambda\omega'_e$. It is important to notice that this question for $\lambda\omega_e$ on open terms emerged when analyzing λs open terms, a calculus in which there is no distinction between semi-open and open terms since it is one-sorted.

We introduced a new calculus, $\lambda\omega'_e$, to which we transferred the same result. This calculus is closer to $\lambda\sigma$ than the calculus $\lambda\omega_e$ (which is isomorphic to $\lambda\overrightarrow{s}_e$ yet written in the $\lambda\sigma$ style), in the sense that the only de Bruijn index it uses is 1. It is a good example which shows that a calculus may not need more than a single index, if it has adequate composition rules. Thus such a new calculus has a smaller set of terms when compared to its parent. We showed that $\lambda\omega'_e$ enjoys the same good properties as $\lambda\omega_e$, by relating their respective sets of normal forms. For that purpose we provided conditional context-free grammars to describe the normal forms, this being a useful tool.

Future work includes a possible characterization of the properties that make it possible to carry over this result to other calculi. Also, it will be interesting to analyze weak normalization (possibly in a different line from [6]) for typed $\lambda\omega_e$ and $\lambda\omega'_e$ on full open terms. Another interesting line of research is the study of other typing systems for these calculi, such as intersection and higher-order typing.

10 Acknowledgment

We are grateful to the anonymous referees for their careful and detailed comments and corrections.

References

[1] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit Substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[2] Z. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. λv, a calculus of explicit substitutions which preserves strong normalisation. *Functional Programming*, 6(5), 1996.

[3] P.-L. Curien. Categorical Combinators, Sequential Algorithms and Functional Programming. Pitman, 1986. Revised edition, Birkhäuser (1993).

[4] P.-L. Curien, T. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. Technical Report RR 1617, INRIA, Rocquencourt, 1992.

[5] N. de Bruijn. Lambda-Calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser Theorem. *Indag. Mat.*, 34(5):381–392, 1972.

[6] J. Goubault-Larrecq. A proof of weak termination of the simply typed $\lambda\sigma$-calculus. Proc. Int. Workshop on Types for Proofs and Programs (TYPES'96). *LNCS*, 1512:134–151, 1998.

[7] B. Guillaume. *Un calcul des substitutions avec étiquettes.* PhD thesis, Université de Savoie, Chambéry, France, 1999.

[8] F. Kamareddine and A. Ríos. Extending a λ-calculus with explicit substitution which preserves strong normalisation into a confluent calculus on open terms. *Journal of Functional Programming*, 7(4):395–420, 1997.

[9] F. Kamareddine and A. Ríos. Relating the $\lambda\sigma$- and λs-styles of explicit substitutions. *Logic and Computation*, 10(3):349–380, 2000.

[10] A. Ríos. *Contribution à l'étude des λ-calculs avec substitutions explicites.* PhD thesis, Université de Paris 7, 1993.

Received 24 November 2004. Revised 23 May 2005.