λ-calculi with explicit substitutions and weak composition which preserve β-strong normalization

Maria C. F. Ferreira [*] Delia Kesner [‡] Laurence Puel [‡]

Abstract

This paper studies preservation of β-strong normalization by two different confluent λ-calculi with explicit substitutions defined in [Kes96]; the particularity of these calculi, called λ_d and λ_{dn} respectively, is that both have a (weak) composition operator for substitutions. We develop an abstract simulation technique allowing to reduce preservation of β-strong normalization of one calculus to that of another one. We apply this abstract technique to reduce preservation of β-strong normalization of λ_d and λ_{dn} to that of another calculus, called λ_f, having no composition operator. Then, preservation of β-strong normalization of λ_f is shown using the same technique as in [BBLRD95]. As a consequence, λ_d and λ_{dn} become the first λ-calculi with explicit substitutions having (weak) composition and preserving β-strong normalization. As an aside, we also show how to apply our technique to reduce preservation of β-strong normalization of the calculus λ_v in [Les94] to that of λ_f.

[*] Dep. de Informática, Fac. de Ciências e Tecnologia, Univ. Nova de Lisboa, Quinta da Torre, 2825 Monte da Caparica, Portugal, cfdi.fct.unl.pt.

[‡] CNRS and Laboratoire de Recherche en Informatique, Bat 490, Université de Paris-Sud, 91405 Orsay Cedex, France, {kesner,puel}lri.fr.

1 Introduction

The λ-calculus is a convenient framework to study functional programming, where the evaluation process is modeled by the β-reduction rule $(\lambda x.M)N \longrightarrow_\beta M\{x \leftarrow N\}$.

The main mechanism used to perform β-reduction is the operation between brackets, called *substitution*, which consists of the replacement of formal parameters by actual arguments. This operation is treated as an atomic step in the λ-calculus, and is defined in a meta level by other functions that are external to the language. However, substitutions need to be treated explicitly when implementing functional languages, and a mechanism to deal with them becomes necessary.

In λ-calculi with explicit substitutions, this is possible since the substitution operation is handled by symbols and reduction rules which belong to the syntax of the calculus. Thus, one can delay the application of a substitution to a term by considering terms with partially applied substitutions. Most of the calculi with explicit substitutions also use natural numbers to represent the names of variables (de Bruijn's indices), avoiding in this way the systematic renaming of bound variables (α-conversion) which is necessary in classical λ-calculus to guarantee the correctness of substitution.

The λ_σ calculus was introduced in [ACCaL91] as a bridge between the classical λ-calculus and concrete implementations of functional programming languages. It is similar to, and inspired by, Categorical Combinatory Logic (CCL) [Cur86], and is very useful to derive machines for the λ-calculus [HMP95], or to implement higher-order unification [DHK95]. We find in the λ_σ-calculus the main features which characterize a calculus of explicit substitutions: substitutions are incorporated to the language and manipulated explicitly, objects are divided in two sorts - terms themselves and substitutions - , and β-reduction is simulated in two stages, first by the application of the *Beta* rule which activates the calculus of substitutions, then by propagation of the substitution until variables are reached. Substitutions can also be combined by a composition operator allowing many interactions between them. Strong normalization of the calculus of explicit substitutions alone, called the σ-calculus, is proved in [CHR92], but the σ-calculus together with the *Beta* rule does not preserve β-strong normalization [Mel95]. The λ_σ-calculus is also confluent on closed terms but is no longer confluent when open terms and substitutions are also considered. To overcome this problem, Hardin and Lévy introduced the $\lambda_{\sigma_{\Uparrow}}$-calculus [HL89] which allows to recover the confluence property on the set of open terms [CHL92]. However, the $\lambda_{\sigma_{\Uparrow}}$-calculus, as the λ_σ-calculus, does not preserve β-strong normalization either [Mel94].

The λ_υ-calculus [Les94] is, in some sense, the *minimal* calculus of explicit substitutions that one can expect: it correctly implements the λ-calculus and it has as few forms of substitutions as possible, eliminating in this way the possibility of composing two different substitutions. The λ_υ-calculus preserves β-strong normalization (see [BBLRD95] for details) but it is not confluent on open terms. Another calculus with explicit substitutions, the λ_s-calculus [KR95b], is inspired by the λ-calculus à la de Bruijn as it explicitly handles the meta-operators used to define substitutions in de Bruijn's calculus; thus, there are no composition operators for substitutions. The λ_s-calculus preserves β-strong normalization, however it is not confluent on open terms; an extension of it exists [KR95a] which is confluent on open terms, but for which the preservation of β-strong normalization is still an open question. It turns out that these well-known calculi preserving β-strong normalization and simulating one-step β reduction are isomorphic to $\lambda\mathtt{x}$ [Lins86, Lins92, Rose92, BR95], which is a calculus with explicit substitutions and named variables notation.

Recently, another calculus with explicit substitutions was proposed in [Muñ96]; this calculus is confluent on open terms and preserves β-strong normalization, however, the calculus does not only lack a composition operator for substitutions, but it implements a particular strategy of the λ-calculus: the distribution of substitutions to applications is only possible if the head term is a variable. Thus, there are some β-reduction sequences in classical λ-calculus which can not be simulated in the calculus in [Muñ96].

Apparently, there is a choice between confluence on open terms, composition and preservation of β-strong normalization, which is an interesting challenge that we partially answer in this paper: composition is possible in calculi preserving β-strong normalization and being confluent on closed terms.

We study two different calculi with explicit substitutions initially defined in [Kes96]; these calculi, named λ_d and λ_{dn} respectively, do have (weak) composition, are confluent on closed terms and preserve β-strong normalization. The λ_d-calculus is inspired by the λ_τ-calculus from [Río93], which is in turn inspired by [Ehr88], the difference being that the $MapEnv$ rule of the λ_τ-calculus, namely:

$$\#(a) \circ s \longrightarrow \Uparrow(s) \circ \#(a[s])$$

is replaced here by the two following rules:

$$\begin{array}{lcl} \underline{1}[\#(a) \circ s] & \longrightarrow & a[s] \\ \uparrow \circ (\#(a) \circ s) & \longrightarrow & s \end{array}$$

This may seem superfluous, but it is essential to prove confluence and strong normalization of the d-calculus, in contrast with the case of the τ-calculus, for which confluence remains still as a conjecture, and for which strong normalization does not hold, as was shown by Pierre Lescanne [Les96]. However, the rule $MapEnv$ (or a similar one) is essential if one wants to recover the confluence property on open terms, and is at the same time, as explained in [Mel95], the cause of non-termination in many λ-calculi with explicit substitutions.

We show that preservation of β-strong normalization of λ_d and λ_{dn} reduces to that of another calculus, called λ_f. The λ_f-calculus, which can be seen as the λ_υ-calculus of [Les94] without variables of the form $\underline{n+1}$, arose from transformations of the original calculi λ_d and λ_{dn}. The transformations applied to both calculi work at two levels. First composition of substitutions is eliminated (and in the case of λ_{dn}, the representation of variables is standardized), giving rise to another intermediate calculus, the λ_e-calculus. Then, we eliminate the identity substitution from λ_e obtaining the λ_f-calculus. Those transformations are defined by complete rewrite systems. Reduction sequences in λ_d (resp. λ_{dn}) are translated, via a simulation technique, to reduction sequences in λ_f, in such a way that λ_f-strong normalization is preserved. Consequently, if λ_f preserves β-strong normalization, so do λ_d and λ_{dn} (and the intermediate calculus λ_e). It remains then to see that λ_f does itself preserve β-strong normalization. This is done applying the same technique used in [BBLRD95]. Interestingly, it turns out that the simulation technique presented here can also be applied to show that preservation of β-strong normalization of λ_υ reduces to that of λ_f.

Before ending the section, we make some remarks on the solution presented. It would be possible to completely skip the λ_f-calculus (thus keeping the identity substitution) and prove preservation of β-strong normalization of the λ_e-calculus using the technique presented in [BBLRD95]. But we prefer to work directly with the λ_f-calculus, which is, in some sense, the minimal calculus of substitution that one can expect to preserve β-strong normalization and to be confluent on closed terms. In particular, λ_f can be seen as the calculus λ_υ *without* constants of the form $\underline{n+1}$, so that translating λ_υ into λ_f is straightforward. Also, once one decides to get rid of the identity substitution, both transformations, elimination of the composition operator and elimination of the identity substitution, respectively, could have been made in one step, thus suppressing the intermediate calculus λ_e, but we believe that the separation of the transformations as presented in this paper allows us to break down the problem into much simpler ones. We will make however some more technical considerations about the possibility of skipping the λ_e-calculus in section 5.

The paper is organized as follows. In Section 2 we introduce some notations and definitions and we give the syntax of all the calculi we are going to deal with.

In Section 3 we explain the simulation technique that will be used along the paper to show that our different calculi preserve β-strong normalization. Section 4 is devoted to the translations used by our simulation technique. More precisely, we define four translations: from λ_d to λ_e, from λ_{dn} to λ_e, from λ_e to λ_f and from λ_v to λ_f. We also give some properties satisfied by these translations. In Section 6 we apply the simulation technique introduced in section 3 to show that λ_d and λ_{dn} preserve λ_e strong normalization, while Section 7 is devoted to show that both λ_e and λ_v preserve λ_f strong normalization. In Section 8 we show that λ_f preserves β-strong normalization using the technique in [BBLRD95]. This result is essential, since it allows us to conclude that λ_d, λ_{dn}, λ_e and λ_v also preserve β-strong normalization.

Finally, in Section 9 we show our main results, namely, preservation of β-strong normalization by λ_d and λ_{dn}. We conclude with some remarks in Section 10.

2 Definitions and notations

A classical way to avoid α-conversion problems in λ-calculus, is to use the de Bruijn's notation [dB72, dB78] for λ-terms, where names of variables are replaced by natural numbers. Hence, the set of λ-terms (also called *pure terms*) is defined as follows:

Definition 2.1 (λ-terms)

$$\begin{array}{llll} Naturals & n & ::= & 1 \mid n+1 \\ PureTerms & a & ::= & \underline{n} \mid (aa) \mid \lambda a \end{array}$$

Parenthesis in application terms associate to the left, we omit them when no confusion arises.

A given occurrence u of a variable, say x, is replaced by the number of symbols λ whose occurrences are between the binder of this x and u. For example, $\lambda x.\lambda y.x(\lambda z.zx)y$ is written as $\lambda(\lambda(\underline{2}(\lambda(\underline{1}\ \underline{3}))\underline{1}))$. Note that the same variable can be translated to different integers (as the variable x is translated to $\underline{2}$ and $\underline{3}$), and that different variables can be translated to the same integer (as the variables z and y are both translated to $\underline{1}$). Representation of free variables is retrieved from a given context of variables in such a way that an occurrence of a variable $\underline{n}$ represents the $(n-m)$-th free variables of the context when $n > m$ and there are m symbols λ surrounding this occurrence n. For example, the term $\lambda x.(y\ z)$ is written as $\lambda(\underline{2}\ \underline{3})$ with respect to the context y, z.

In λ-calculi with explicit substitutions, the objects denoting substitutions are incorporated to the language in an explicit way, as well as the manipulation of substitutions, which is incorporated via a set of rewrite rules. This syntactic presentation can be defined, for all the calculi we are going to deal with in this paper, in the following way:

Definition 2.2 (Substitution signatures) Let us consider two distinguished symbols $\mathbf{T}$ (for terms) and $\mathbf{S}$ (for substitutions). A *substitution declaration* is a (possibly empty) word over the alphabet $\{\mathbf{T}, \mathbf{S}\}$. A *substitution signature* is defined to be a signature Σ (a set of symbols) such that every symbol in Σ is equipped with an arity n and a declaration of length n. We usually write $\xi : \langle n, \sigma_1 \ldots \sigma_n \rangle$ if ξ has arity n and declaration $\sigma_1 \ldots \sigma_n$.

Definition 2.3 (Substitution language) The set of objects over a substitution signature Σ is defined to be the union of objects of sort $\mathbf{T}$ and $\mathbf{S}$. They are constructed in the following way :

- If n is a natural number, then $\underline{n}$, called a *variable*, is of sort $\mathbf{T}$.

- If a and b are of sort $\mathbf{T}$, then $(a\ b)$, called an *application*, is of sort $\mathbf{T}$.
- If a is of sort $\mathbf{T}$, then $\lambda(a)$, called an *abstraction*, is of sort $\mathbf{T}$.
- If a is of sort $\mathbf{T}$ and s is of sort $\mathbf{S}$, then $a[s]$, called a *closure*, is of sort $\mathbf{T}$.
- If every f_i is of sort σ_i, and $\xi : \langle n, \sigma_1 \ldots \sigma_n \rangle$, then $\xi(f_1, \ldots, f_n)$, called a *substitution*, is of sort $\mathbf{S}$.

For each substitution language W, the set of objects of sort $\mathbf{T}$ is denoted by $\mathcal{T}_W$, and called *the set of terms of* W, and the set of objects of sort $\mathbf{S}$ is denoted by $\mathcal{S}_W$, and called *the set of substitutions of* W. We write $\mathcal{O}_W$ for $\mathcal{T}_W \cup \mathcal{S}_W$.

Notation 2.1 Let W be a substitution language and let s be a substitution of W. We note $[s]^n$ the concatenation of substitutions $\underbrace{[s]\ldots[s]}_{n\ \ times}$, assuming that $[s]^0$ denotes the empty word. We also use the following notation for multiple applications of a unary substitution operator:

$$\begin{array}{lcl} \xi^0(s) & = & s \\ \xi^{n+1}(s) & = & \xi(\xi^n(s)) \end{array}$$

We now introduce the rewriting rules of our different calculi λ_d, λ_{dn}, λ_e and λ_f. We write λ_W for the set of rewriting rules of the λ-calculus W, and simply W for the set $\lambda_W - Beta$. The notations $\longrightarrow_{\mathcal{R}}$ and $=_{\mathcal{R}}$ are used to denote respectively, $\mathcal{R}$-reduction and $\mathcal{R}$-equality, where $\mathcal{R}$ can be either a set of rewrite rules or a single rule.

For all calculi and reductions considered, to simplify the notation, we omit the substitution σ used in the reduction step and use other objects corresponding to the substitution application; for example instead of writing $(\lambda a^\sigma)b^\sigma \longrightarrow_{Beta} a^\sigma[\#(b^\sigma)]$, we write $(\lambda a_1)a_2 \longrightarrow_{Beta} a_1[\#(a_2)]$. We will also use the notation $C\langle\!|\ |\!\rangle$ to denote arbitrary linear contexts C, instead of the more usual notation $C[\ ]$; this is so to avoid confusion with substitution application. Finally we note that both reduction rules and reductions we are interested in, apply to objects of the appropriate sort, the ones constructed by our grammars.

The λ_d-calculus

The signature associated to the substitutions of the λ_d-calculus is given by the set $\{\#, \Uparrow, \circ, \uparrow, id\}$, where $\# : \langle 1, \mathcal{T} \rangle$, $\Uparrow: \langle 1, \mathcal{S} \rangle$, $\circ : \langle 2, \mathcal{SS} \rangle$, $\uparrow: \langle 0, \epsilon \rangle$ and $id : \langle 0, \epsilon \rangle$. Reduction rules of the λ_d-calculus are given in table 1. This set of rules can be decomposed into two disjoint sets of rewriting rules: A and B, shown in tables 2 and 3 respectively.

The λ_{dn}-calculus

The signature associated to the λ_{dn}-calculus is the same as before. Reduction rules of the λ_{dn}-calculus are given in table 4 and are decomposed into two disjoint sets of rewriting rules: A_n and B_n, shown in tables 5 and 6, respectively.

The λ_e-calculus

We no longer have a composition operator in λ_e, so the signature associated to the substitution language of the λ_e-calculus is the set $\{\#, \Uparrow, \uparrow, id\}$, where $\# : \langle 1, \mathcal{T} \rangle$, $\Uparrow: \langle 1, \mathcal{S} \rangle$, $\uparrow: \langle 0, \epsilon \rangle$ and $id : \langle 0, \epsilon \rangle$.

Reduction rules of the λ_e-calculus are given in table 7. Note that the system is infinite since i ranges over $\mathbb{N}$, the natural numbers including 0.

$(Beta)$	$(\lambda a)b$	$\longrightarrow$	$a[\#(b)]$
(App)	$(a\ b)[s]$	$\longrightarrow$	$(a[s]\ b[s])$
$(Lambda)$	$(\lambda a)[s]$	$\longrightarrow$	$\lambda(a[\Uparrow(s)])$
$(Clos)$	$(a[s])[t]$	$\longrightarrow$	$a[s\circ t]$
$(FVar1)$	$\underline{1}[\#(a)]$	$\longrightarrow$	a
$(FVar2)$	$\underline{1}[\#(a)\circ s]$	$\longrightarrow$	$a[s]$
$(FVarLift1)$	$\underline{1}[\Uparrow(s)]$	$\longrightarrow$	$\underline{1}$
$(FVarLift2)$	$\underline{1}[\Uparrow(s)\circ t]$	$\longrightarrow$	$\underline{1}[t]$
(Ass)	$(s_1\circ s_2)\circ s_3$	$\longrightarrow$	$s_1\circ(s_2\circ s_3)$
$(Shift1)$	$\uparrow\circ\#(a)$	$\longrightarrow$	id
$(Shift2)$	$\uparrow\circ(\#(a)\circ s)$	$\longrightarrow$	s
$(ShiftLift1)$	$\uparrow\circ\Uparrow(s)$	$\longrightarrow$	$s\circ\uparrow$
$(ShiftLift2)$	$\uparrow\circ(\Uparrow(s)\circ t)$	$\longrightarrow$	$s\circ(\uparrow\circ t)$
$(Lift1)$	$\Uparrow(s)\circ\Uparrow(t)$	$\longrightarrow$	$\Uparrow(s\circ t)$
$(Lift2)$	$\Uparrow(s)\circ(\Uparrow(t)\circ u)$	$\longrightarrow$	$\Uparrow(s\circ t)\circ u$
(IdL)	$id\circ s$	$\longrightarrow$	s
(IdR)	$s\circ id$	$\longrightarrow$	s
$(LiftId)$	$\Uparrow(id)$	$\longrightarrow$	id
(Id)	$a[id]$	$\longrightarrow$	a

Table 1: Reduction rules of the λ_d-calculus.

The reduction rules of the λ_e-calculus originate two disjoint infinite sets of rewriting rules C and D, shown in tables 8 and 9, respectively. In both cases, i and j range over $\mathbb{N}$.

Even though the calculus λ_e does not coincide, syntactically speaking, with the calculus $C\cup D$, it is not difficult to see that when we consider reductions on objects constructed by our grammar, the reduction relations defined by both calculi are the same, i. e.,

$$\text{If } p \text{ and } p' \text{ are terms in } \mathcal{T}_e, \text{ then } p \longrightarrow_{\lambda_e} p' \text{ if and only if } p \longrightarrow_{C\cup D} p'. \tag{1}$$

As a consequence, from now on we will indistinctly use λ_e or $C\cup D$.

The λ_f-calculus

We no longer have composition, nor identity in λ_f, so that the signature associated to the substitution language of the λ_f-calculus is the set $\{\#,\Uparrow,\uparrow\}$, where again $\#:\langle 1,\mathcal{T}\rangle$, $\Uparrow:\langle 1,\mathcal{S}\rangle$ and $\uparrow:\langle 0,\epsilon\rangle$. Reduction rules of the λ_f-calculus are given in table 10, where i ranges over $\mathbb{N}$. It is worth noticing that any variable $\underline{n+1}$ $(n\geq 1)$ is represented by:

- the term $\underline{1}[\underbrace{\uparrow\circ(\uparrow\circ(\ldots\circ\uparrow))}_{n\ times}]$ in λ_d, and
- the term $\underline{1}\underbrace{[\uparrow]\ldots[\uparrow]}_{n\ times}$ in λ_e and λ_f,

so that in every case we can assume the same set of pure terms, given by definition 2.1.

The λ_υ-calculus

The λ_υ-calculus [Les94] has as signature $\{\#,\Uparrow,\uparrow\}$, where again $\#:\langle 1,\mathcal{T}\rangle$, $\Uparrow:\langle 1,\mathcal{S}\rangle$ and $\uparrow:\langle 0,\epsilon\rangle$.

Reduction rules of the λ_υ-calculus are given in table 11.

$(Beta)$	$(\lambda a)b$	$\longrightarrow$	$a[\#(b)]$
(App)	$(a\ b)[s]$	$\longrightarrow$	$(a[s]\ b[s])$
$(Lambda)$	$(\lambda a)[s]$	$\longrightarrow$	$\lambda(a[\Uparrow(s)])$
$(FVar1)$	$\underline{1}[\#(a)]$	$\longrightarrow$	a
$(FVar2)$	$\underline{1}[\#(a)\circ s]$	$\longrightarrow$	$a[s]$
$(FVarLift1)$	$\underline{1}[\Uparrow(s)]$	$\longrightarrow$	$\underline{1}$
$(FVarLift2)$	$\underline{1}[\Uparrow(s)\circ t]$	$\longrightarrow$	$\underline{1}[t]$
$(Shift1)$	$\uparrow\circ\#(a)$	$\longrightarrow$	id
$(Shift2)$	$\uparrow\circ(\#(a)\circ s)$	$\longrightarrow$	s
$(ShiftLift1)$	$\uparrow\circ\Uparrow(s)$	$\longrightarrow$	$s\circ\uparrow$
$(ShiftLift2)$	$\uparrow\circ(\Uparrow(s)\circ t)$	$\longrightarrow$	$s\circ(\uparrow\circ t)$
(IdL)	$id\circ s$	$\longrightarrow$	s
(IdR)	$s\circ id$	$\longrightarrow$	s
$(LiftId)$	$\Uparrow(id)$	$\longrightarrow$	id
(Id)	$a[id]$	$\longrightarrow$	a

Table 2: Reduction rules of the A-calculus.

$(Clos)$	$(a[s])[t]$	$\longrightarrow$	$a[s\circ t]$
(Ass)	$(s_1\circ s_2)\circ s_3$	$\longrightarrow$	$s_1\circ(s_2\circ s_3)$
$(Lift1)$	$\Uparrow(s)\circ\Uparrow(t)$	$\longrightarrow$	$\Uparrow(s\circ t)$
$(Lift2)$	$\Uparrow(s)\circ(\Uparrow(t)\circ u)$	$\longrightarrow$	$\Uparrow(s\circ t)\circ u$

Table 3: Reduction rules of the B-calculus.

3 The simulation technique

We give in this section the abstract simulation technique that allows us to show that preservation of β-strong normalization of λ_d, λ_v and λ_{dn} reduces to that of λ_f

We first introduce some abstract definitions and results that will later be used in the context of our calculi.

Definition 3.1 Let $A = \langle \mathcal{O}, R\rangle$ and $B = \langle \mathcal{O}', R'\rangle$ be two abstract reduction systems such that $\mathcal{O}' \subseteq \mathcal{O}$. We say that A preserves B-strong normalization (or that R preserves R'-strong normalization), if every $a \in \mathcal{O}'$ which is B-strongly normalizing (R'-strongly normalizing), is also A-strongly normalizing (R-strongly normalizing).

Theorem 3.1 Let $R = \langle \mathcal{O}, R_1 \cup R_2\rangle$ be an abstract reduction system such that

- R_2 is strongly normalizing;
- there exists a reduction system $S = \langle \mathcal{O}', R'\rangle$ and a translation $\mathcal{T}$ from $\mathcal{O}$ to $\mathcal{O}'$ such that
  - $\mathcal{O}' \subseteq \mathcal{O}$,
  - $\mathcal{T}(a) = a$, for any $a \in \mathcal{O}'$,
  - $a \longrightarrow_{R_1} b$ implies $\mathcal{T}(a) \longrightarrow^+_{R'} \mathcal{T}(b)$,
  - $a \longrightarrow_{R_2} b$ implies $\mathcal{T}(a) = \mathcal{T}(b)$.

$(Beta)$	$(\lambda a)b$	$\longrightarrow$	$a[\#(b)]$
(App)	$(a\ b)[s]$	$\longrightarrow$	$(a[s]\ b[s])$
$(Lambda)$	$(\lambda a)[s]$	$\longrightarrow$	$\lambda(a[\Uparrow(s)])$
$(Clos)$	$(a[s])[t]$	$\longrightarrow$	$a[s \circ t]$
$(FVar1)$	$\underline{1}[\#(a)]$	$\longrightarrow$	a
$(FVar2)$	$\underline{1}[\#(a) \circ s]$	$\longrightarrow$	$a[s]$
$(FVarLift1)$	$\underline{1}[\Uparrow(s)]$	$\longrightarrow$	$\underline{1}$
$(FVarLift2)$	$\underline{1}[\Uparrow(s) \circ t]$	$\longrightarrow$	$\underline{1}[t]$
(Ass)	$(s_1 \circ s_2) \circ s_3$	$\longrightarrow$	$s_1 \circ (s_2 \circ s_3)$
$(Shift1)$	$\uparrow \circ \#(a)$	$\longrightarrow$	id
$(Shift2)$	$\uparrow \circ (\#(a) \circ s)$	$\longrightarrow$	s
$(ShiftLift1)$	$\uparrow \circ \Uparrow(s)$	$\longrightarrow$	$s \circ \uparrow$
$(ShiftLift2)$	$\uparrow \circ (\Uparrow(s) \circ t)$	$\longrightarrow$	$s \circ (\uparrow \circ t)$
$(Lift1)$	$\Uparrow(s) \circ \Uparrow(t)$	$\longrightarrow$	$\Uparrow(s \circ t)$
$(Lift2)$	$\Uparrow(s) \circ (\Uparrow(t) \circ u)$	$\longrightarrow$	$\Uparrow(s \circ t) \circ u$
(IdL)	$id \circ s$	$\longrightarrow$	s
(IdR)	$s \circ id$	$\longrightarrow$	s
$(LiftId)$	$\Uparrow(id)$	$\longrightarrow$	id
(Id)	$a[id]$	$\longrightarrow$	a
$(VarShift1)$	$\underline{n}[\uparrow]$	$\longrightarrow$	$\underline{n+1}$
$(VarShift2)$	$\underline{n}[\uparrow \circ s]$	$\longrightarrow$	$\underline{n+1}[s]$
$(RVar1)$	$\underline{n+1}[\#(a)]$	$\longrightarrow$	$\underline{n}$
$(RVar2)$	$\underline{n+1}[\#(a) \circ s]$	$\longrightarrow$	$\underline{n}[s]$
$(RVarLift1)$	$\underline{n+1}[\Uparrow(s)]$	$\longrightarrow$	$\underline{n}[s \circ \uparrow]$
$(RVarLift2)$	$\underline{n+1}[\Uparrow(s) \circ t]$	$\longrightarrow$	$\underline{n}[s \circ (\uparrow \circ t)]$

Table 4: Reduction rules of the λ_{dn}-calculus.

Then $R_1 \cup R_2$ preserves R'-strong normalization.

Proof. We proceed by contradiction. Suppose that $R_1 \cup R_2$ does not preserve R'-strong normalization and let $a \in \mathcal{O}'$ be such that a is R'-strongly normalizing but not $R_1 \cup R_2$-strongly normalizing. First note that any infinite reduction in $R_1 \cup R_2$ has to contain infinitely many steps of R_1: if it would contain only finitely many steps of R_1, then the remaining of the infinite reduction sequence would be an infinite sequence in R_2, contradicting the fact that R_2 is strongly normalizing. With this observations, the infinite sequence (with respect to $R_1 \cup R_2$) starting at a can be written as

$$a \longrightarrow^*_{R_2} a_{i_1} \longrightarrow^+_{R_1} a_{i_2} \longrightarrow^*_{R_2} a_{i_3} \longrightarrow^+_{R_1} a_{i_4} \longrightarrow^*_{R_2} \cdots$$

But this sequence translates to the following infinite sequence in $\langle \mathcal{O}', R' \rangle$:

$$\mathcal{T}(a) = a = \mathcal{T}(a_{i_1}) \longrightarrow^+_{R'} \mathcal{T}(a_{i_2}) = \mathcal{T}(a_{i_3}) \longrightarrow^+_{R'} \mathcal{T}(a_{i_4}) = \mathcal{T}(a_{i_5}) \longrightarrow^+_{R'} \cdots$$

contradicting R'-strong normalization of the term a.

We should note that in the result above, some conditions can be weakened. Namely instead of requiring that for any $a \in \mathcal{O}'$, $\mathcal{T}(a) = a$, we can require that $\mathcal{T}(\mathcal{O}') \subseteq \mathcal{O}'$ and that $\mathcal{T}$ respects strong normalization, i. e., if $a \in \mathcal{O}'$ is R'-strongly normalizing, then $\mathcal{T}(a)$ is also R'-strongly normalizing.

$(Beta)$	$(\lambda a)b$	$\longrightarrow$	$a[\#(b)]$
(App)	$(a\ b)[s]$	$\longrightarrow$	$(a[s]\ b[s])$
$(Lambda)$	$(\lambda a)[s]$	$\longrightarrow$	$\lambda(a[\Uparrow(s)])$
$(FVar1)$	$\underline{1}[\#(a)]$	$\longrightarrow$	a
$(FVar2)$	$\underline{1}[\#(a)\circ s]$	$\longrightarrow$	$a[s]$
$(FVarLift1)$	$\underline{1}[\Uparrow(s)]$	$\longrightarrow$	$\underline{1}$
$(FVarLift2)$	$\underline{1}[\Uparrow(s)\circ t]$	$\longrightarrow$	$\underline{1}[t]$
$(Shift1)$	$\uparrow\circ\#(a)$	$\longrightarrow$	id
$(Shift2)$	$\uparrow\circ(\#(a)\circ s)$	$\longrightarrow$	s
$(ShiftLift1)$	$\uparrow\circ\Uparrow(s)$	$\longrightarrow$	$s\circ\uparrow$
$(ShiftLift2)$	$\uparrow\circ(\Uparrow(s)\circ t)$	$\longrightarrow$	$s\circ(\uparrow\circ t)$
(IdL)	$id\circ s$	$\longrightarrow$	s
(IdR)	$s\circ id$	$\longrightarrow$	s
$(LiftId)$	$\Uparrow(id)$	$\longrightarrow$	id
(Id)	$a[id]$	$\longrightarrow$	a
$(RVar1)$	$\underline{n+1}[\#(a)]$	$\longrightarrow$	$\underline{n}$
$(RVar2)$	$\underline{n+1}[\#(a)\circ s]$	$\longrightarrow$	$\underline{n}[s]$
$(RVarLift1)$	$\underline{n+1}[\Uparrow(s)]$	$\longrightarrow$	$\underline{n}[s\circ\uparrow]$
$(RVarLift2)$	$\underline{n+1}[\Uparrow(s)\circ t]$	$\longrightarrow$	$\underline{n}[s\circ(\uparrow\circ t)]$

Table 5: Reduction rules of the A_n-calculus.

$(Clos)$	$(a[s])[t]$	$\longrightarrow$	$a[s\circ t]$
(Ass)	$(s_1\circ s_2)\circ s_3$	$\longrightarrow$	$s_1\circ(s_2\circ s_3)$
$(Lift1)$	$\Uparrow(s)\circ\Uparrow(t)$	$\longrightarrow$	$\Uparrow(s\circ t)$
$(Lift2)$	$\Uparrow(s)\circ(\Uparrow(t)\circ u)$	$\longrightarrow$	$\Uparrow(s\circ t)\circ u$
$(VarShift1)$	$\underline{n}[\uparrow]$	$\longrightarrow$	$\underline{n+1}$
$(VarShift2)$	$\underline{n}[\uparrow\circ s]$	$\longrightarrow$	$\underline{n+1}[s]$

Table 6: Reduction rules of the B_n-calculus.

4 The translations

In this section we present the transformations that allow us to reduce preservation of β-strong normalization of λ_d, λ_{dn} and λ_v to that of λ_f. For that, we first translate the λ_d and λ_{dn} calculi into λ_e, then λ_e and λ_v will be simulated by λ_f. The first transformation is defined by the complete rewrite system de and allows us to eliminate the composition operator from the calculus λ_d. The second one, called dne, allows to eliminate not only the composition operator but also all the variables $\underline{n+1}$ from the calculus λ_{dn}. Both transformations are defined in such a way that terms and reductions in λ_d (resp. λ_{dn}) are translated into terms and reductions in λ_e. The third transformation is defined by the also complete rewrite system ef, and allows us to eliminate the identity substitution from the calculus λ_e. Finally, vf eliminates variables from λ_v. These two last transformations allows us to simulate terms and reductions of both λ_e and λ_v by terms and reductions of the more atomic calculus λ_f.

We consider the following infinite rewrite systems:

$(Beta)$	$(\lambda a)b$	$\longrightarrow$	$a[\#(b)]$
(App)	$(a\ b)[s]$	$\longrightarrow$	$(a[s]\ b[s])$
$(Lambda)$	$(\lambda a)[s]$	$\longrightarrow$	$\lambda(a[\Uparrow(s)])$
$(FVar)$	$\underline{1}[\#(a)]$	$\longrightarrow$	a
$(FVarLift)$	$\underline{1}[\Uparrow(s)]$	$\longrightarrow$	$\underline{1}$
$(Shift)$	$a[\Uparrow^i(\uparrow)][\Uparrow^i(\#(b))]$	$\longrightarrow$	$a[\Uparrow^i(id)]$
$(ShiftLift)$	$a[\Uparrow^i(\uparrow)][\Uparrow^{i+1}(s)]$	$\longrightarrow$	$a[\Uparrow^i(s)][\Uparrow^i(\uparrow)]$
$(LiftId)$	$a[\Uparrow^{i+1}(id)]$	$\longrightarrow$	$a[\Uparrow^i(id)]$
(Id)	$a[id]$	$\longrightarrow$	a

Table 7: Reduction rules of the λ_e-calculus

$(Beta)$	$(\lambda a)b$	$\longrightarrow$	$a[\#(b)]$
$(\widehat{App})$	$(a\ b)[\Uparrow^i(\uparrow)]$	$\longrightarrow$	$(a[\Uparrow^i(\uparrow)]\ b[\Uparrow^i(\uparrow)])$
	$(a\ b)[\Uparrow^i(\#(c))]$	$\longrightarrow$	$(a[\Uparrow^i(\#(c))]\ b[\Uparrow^i(\#(c))])$
$(\widehat{Lambda})$	$(\lambda a)[\Uparrow^i(\uparrow)]$	$\longrightarrow$	$\lambda(a[\Uparrow^{i+1}(\uparrow)])$
	$(\lambda a)[\Uparrow^i(\#(c))]$	$\longrightarrow$	$\lambda(a[\Uparrow^{i+1}(\#(c))])$
$(FVar)$	$\underline{1}[\#(a)]$	$\longrightarrow$	a
$(\widehat{FVarLift})$	$\underline{1}[\Uparrow^{i+1}(\uparrow)]$	$\longrightarrow$	$\underline{1}$
	$\underline{1}[\Uparrow^{i+1}(\#(a))]$	$\longrightarrow$	$\underline{1}$
$(Shift)$	$a[\Uparrow^i(\uparrow)][\Uparrow^i(\#(b))]$	$\longrightarrow$	$a[\Uparrow^i(id)]$
$(\widehat{ShiftLift})$	$a[\Uparrow^i(\uparrow)][\Uparrow^{i+j+1}(\uparrow)]$	$\longrightarrow$	$a[\Uparrow^{i+j}(\uparrow)][\Uparrow^i(\uparrow)]$
	$a[\Uparrow^i(\uparrow)][\Uparrow^{i+j+1}(\#(b))]$	$\longrightarrow$	$a[\Uparrow^{i+j}(\#(b))][\Uparrow^i(\uparrow)]$

Table 8: Reduction rules of the C-calculus

(de)	$a[\Uparrow^i(s \circ t)]$	$\longrightarrow$	$a[\Uparrow^i(s)][\Uparrow^i(t)]$	$(i \geq 0)$
(dne)	$a[\Uparrow^i(s \circ t)]$	$\longrightarrow$	$a[\Uparrow^i(s)][\Uparrow^i(t)]$	$(i \geq 0)$
	$\underline{n+1}$	$\longrightarrow$	$\underline{1}[\uparrow]^n$	$(n \geq 1)$
(ef)	$a[\Uparrow^i(id)]$	$\longrightarrow$	a	$(i \geq 0)$
(vf)	$\underline{n+1}$	$\longrightarrow$	$\underline{1}[\uparrow]^n$	$(n \geq 1)$

The reduction relation de is well-defined on $\mathcal{T}_d$ in the sense that $s \in \mathcal{T}_d$ and $s \longrightarrow_{de} s'$ implies $s' \in \mathcal{T}_d$; indeed if the reduction was at the root, we must have that a is a term in the left-hand side and so the whole right-hand side is a term (in $\mathcal{T}_d$), and if the reduction occurred within a context, the same argument applies plus the fact that replacing terms by terms yields a term (of the appropriate syntax). Exactly the same occurs with dne, ef and vf, so from now, to simplify our statements, we will not precise the set of terms for which the reduction applies. We state some properties which are satisfied by the systems de, dne, ef and vf.

Lemma 4.1 The systems de, dne, ef and vf are confluent and strongly normalizing.

$(\widetilde{App})$	$(a\ b)[\Uparrow^i (id)]$	$\longrightarrow$	$(a[\Uparrow^i (id)]\ b[\Uparrow^i (id)])$
$(\widetilde{Lambda})$	$(\lambda a)[\Uparrow^i (id)]$	$\longrightarrow$	$\lambda(a[\Uparrow^{i+1} (id)])$
$(F\widetilde{VarLift})$	$\underline{1}[\Uparrow^{i+1} (id)]$	$\longrightarrow$	$\underline{1}$
$(Shi\widetilde{ftLift})$	$a[\Uparrow^i (\uparrow)][\Uparrow^{i+j+1} (id)]$	$\longrightarrow$	$a[\Uparrow^{i+j} (id)][\Uparrow^i (\uparrow)]$
$(LiftId)$	$a[\Uparrow^{i+1} (id)]$	$\longrightarrow$	$a[\Uparrow^i (id)]$
(Id)	$a[id]$	$\longrightarrow$	a

Table 9: Reduction rules of the D-calculus

$(Beta)$	$(\lambda a)b$	$\longrightarrow$	$a[\#(b)]$
(App)	$(a\ b)[s]$	$\longrightarrow$	$(a[s]\ b[s])$
$(Lambda)$	$(\lambda a)[s]$	$\longrightarrow$	$\lambda(a[\Uparrow (s)])$
$(FVar)$	$\underline{1}[\#(a)]$	$\longrightarrow$	a
$(FVarLift)$	$\underline{1}[\Uparrow (s)]$	$\longrightarrow$	$\underline{1}$
$(Shift)$	$a[\Uparrow^i (\uparrow)][\Uparrow^i (\#(b))]$	$\longrightarrow$	a
$(ShiftLift)$	$a[\Uparrow^i (\uparrow)][\Uparrow^{i+1} (s)]$	$\longrightarrow$	$a[\Uparrow^i (s)][\Uparrow^i (\uparrow)]$

Table 10: Reduction rules of the λ_f-calculus

Proof. All the systems are orthogonal and therefore confluent [Hue80]. Strong normalization of de (resp. ef, vf) can be shown using the fact that whenever $a \longrightarrow_{de} b$ (resp. $a \longrightarrow_{ef} b$, $a \longrightarrow_{vf} b$), then the total number of occurrences of the symbol $\circ$ (resp. id, $\underline{n+1}$) in b is strictly smaller than that of a. Strong normalization of dne can be shown using the fact that $a \longrightarrow_{dne} b$ implies that the number of occurrences of the symbol $\circ$ plus the number of occurrences of constants of the form $\underline{n+1}$ (with $n \geq 1$) in b is strictly smaller than the corresponding number in a.

Lemma 4.1 tells us that the systems de, dne, ef and vf define functions, namely the functions that associate to each object its unique normal form with respect to the translation system. From now on we use the notations $de(s)$ (resp. $dne(s)$, $ef(s)$, $vf(s)$) meaning the normal form of s with respect to the system de (resp. dne, ef, vf). The domain of the function de (resp. dne, ef, vf) is $\mathcal{O}_d$ (resp. $\mathcal{O}_{dn}$, $\mathcal{O}_e$, $\mathcal{O}_v$) and its co-domain is $\mathcal{O}_e$ for de and dne, and $\mathcal{O}_f$ for ef and vf. We will write $Domain(\mathcal{T})$ and $Co\text{-}Domain(\mathcal{T})$ to refer to the domain and co-domain of a given translation $\mathcal{T}$. We first enunciate some other properties of the translation functions.

$(Beta)$	$(\lambda a)b$	$\longrightarrow$	$a[\#(b)]$
(App)	$(a\ b)[s]$	$\longrightarrow$	$(a[s]\ b[s])$
$(Lambda)$	$(\lambda a)[s]$	$\longrightarrow$	$\lambda(a[\Uparrow (s)])$
$(FVar)$	$\underline{1}[\#(a)]$	$\longrightarrow$	a
$(RVar)$	$\underline{n+1}[\#(a)]$	$\longrightarrow$	$\underline{n}$
$(FVarLift)$	$\underline{1}[\Uparrow (s)]$	$\longrightarrow$	$\underline{1}$
$(RVarLift)$	$\underline{n+1}[\Uparrow (s)]$	$\longrightarrow$	$\underline{n}[s][\uparrow]$
$(VarShift)$	$\underline{n}[\uparrow]$	$\longrightarrow$	$\underline{n+1}$

Table 11: Reduction rules of the λ_v-calculus.

Lemma 4.2 (Some properties of the translations) Let $\mathcal{T}$ be any of the functions de, dne, ef and vf. Then, the following properties hold:

1. $\mathcal{T}(a[s]) = \mathcal{T}(\mathcal{T}(a)[s]) = \mathcal{T}(a[\mathcal{T}(s)]) = \mathcal{T}(\mathcal{T}(a)[\mathcal{T}(s)])$.
2. $\mathcal{T}(\underline{1}) = \underline{1}$.
3. $\mathcal{T}(a\ b) = (\mathcal{T}(a)\ \mathcal{T}(b))$.
4. $\mathcal{T}(\lambda a) = \lambda(\mathcal{T}(a))$.
5. If $a \in Co\text{-}Domain(\mathcal{T})$ then $\mathcal{T}(a) = a$.
6. Let $\mathcal{S}$ be one of the translations dne or vf. Then for any $n \geq 1$, $\mathcal{S}(\underline{n+1}) = \underline{1}[\uparrow]^n$.
7. For any $i \geq 0$, $\mathcal{T}(a[\Uparrow^i(\uparrow)]) = \mathcal{T}(a)[\Uparrow^i(\uparrow)]$ and $\mathcal{T}(a[\Uparrow^i(\#(b))]) = \mathcal{T}(a)[\Uparrow^i(\#(\mathcal{T}(b)))]$.
8. Let $\mathcal{S}$ be one of the translations de or dne. Then for any $i \geq 0$, $\mathcal{S}(a[\Uparrow^i(id)]) = \mathcal{S}(a)[\Uparrow^i(id)]$ and $\mathcal{S}(a[\Uparrow^i(s_1 \circ s_2)]) = \mathcal{S}(\mathcal{S}(a[\Uparrow^i(s_1)])[\Uparrow^i(s_2)])$.
9. $ef(a[\Uparrow^i(id)]) = ef(a)$, for any term $a \in \mathcal{T}_e$.

Proof.

1. Note that using any complete (i. e., confluent and strongly normalizing) rewrite system, if a rewrites (or is equal) to a', then the normal forms of a and a' coincide. Since $a[s]$ rewrites to $\mathcal{T}(a)[s]$, $a[\mathcal{T}(s)]$ and $\mathcal{T}(a)[\mathcal{T}(s)]$, the result follows.
2. Obviously the term $\underline{1}$ is in normal form with respect to $\mathcal{T}$ so the result follows.
3. Consequence of the form of the rewrite systems.
4. Consequence of the form of the rewrite systems.
5. For any $a \in Co\text{-}Domain(\mathcal{T})$, the term a cannot contain any redex for the system $\mathcal{T}$ because it is in normal form with respect to $\mathcal{T}$ and so $\mathcal{T}(a) = a$.
6. Consequence of the form of the rewrite systems.
7. For $a[\Uparrow^i(\uparrow)]$, due to the form of the rules in $\mathcal{T}$, reductions in $a[\Uparrow^i(\uparrow)]$ can only occur in the term a, and for $a[\Uparrow^i(\#(b))]$, reductions in the term can only occur in a or b, so the result holds in both cases.
8. Since $a[\Uparrow^i(s_1 \circ s_2)] \longrightarrow^*_{de,dne} a[\Uparrow^i(s_1)][\Uparrow^i(s_2)]$ and due to property 1, we can write $de(a[s]) = de(de(a[s_1])[s_2])$ and $dne(a[s]) = dne(dne(a[s_1])[s_2])$ so the result holds.
9. Since $a[\Uparrow^i(id)] \longrightarrow^*_{ef} a$, the result follows.

Lemma 4.3 For every substitution $s \in \mathcal{S}_d$ (resp. $\mathcal{S}_{dn}$), there exist unique substitutions $u_1 \ldots u_k \in \mathcal{S}_e$, for some $k \geq 1$, such that for every term $a \in \mathcal{T}_d$ (resp. $\mathcal{T}_{dn}$) and every $i \geq 0$, we have

$$\mathcal{T}(a[\Uparrow^i(s)]) = \mathcal{T}(a)[\Uparrow^i(u_1)]\ldots[\Uparrow^i(u_k)]$$

where $\mathcal{T}$ is either de or dne.

Proof. We show the property for $s \in \mathcal{S}_d$, the other case being similar. We proceed by induction on the structure of s.

- If $s = \uparrow$ or $s = id$, then by lemma 4.2, $de(a[\Uparrow^i (s)]) = de(a)[\Uparrow^i (s)]$, for all terms a and all $i \geq 0$, and the result follows with $k = 1$ and $u_1 = s$ (since $\uparrow$ and id are in $\mathcal{S}_e$).

- If $s = \Uparrow (s')$ then for any term a and any $i \geq 0$:

$$\begin{array}{ll} de(a[\Uparrow^i (s)]) & = \\ de(a[\Uparrow^i (\Uparrow (s'))]) & = \\ de(a[\Uparrow^{i+1} (s')) & = \text{ (by induction hypothesis)} \\ de(a)[\Uparrow^{i+1} (v_1)] \ldots [\Uparrow^{i+1} (v_k)] & = \\ de(a)[\Uparrow^i (\Uparrow (v_1))] \ldots [\Uparrow^i (\Uparrow (v_k))] & \end{array}$$

  for some substitutions $v_1, \ldots, v_k \in \mathcal{S}_e$, $k \geq 1$. The result follows with $u_i = \Uparrow (v_i)$, for all $1 \leq i \leq k$.

- If $s = \#(b)$ then for any term a and any $i \geq 0$, lemma 4.2 gives

$$de(a[\Uparrow^i (\#(b))]) = de(a)[\Uparrow^i (\#(de(b)))]$$

  The result follows with $k = 1$ and $u_1 = \#(de(b))$.

- If $s = s_1 \circ s_2$, then for any term a and any $i \geq 0$:

$$\begin{array}{ll} de(a[\Uparrow^i (s_1 \circ s_2)]) & = \text{ (reduction step in } de) \\ de(a[\Uparrow^i (s_1)][\Uparrow^i (s_2)]) & = \text{ (by induction hypothesis)} \\ (de(a[\Uparrow^i (s_1)]))[\Uparrow^i (v_1)] \ldots [\Uparrow^i (v_n)] & = \text{ (by induction hypothesis)} \\ de(a)[\Uparrow^i (w_1)] \ldots [\Uparrow^i (w_m)][\Uparrow^i (v_1)] \ldots [\Uparrow^i (v_n)] & \end{array}$$

  The result then follows with $2 \leq k = m + n$, $u_i = w_i (1 \leq i \leq m)$ and $u_l = v_{l-m} (m + 1 \leq l \leq m + n)$.

From property 7 in lemma 4.2, a similar result can be derived for ef.

Corollary 4.4 If $s = \uparrow$ or $s = \#(c)$, then there exists a unique substitution $u \in \mathcal{S}_f$ such that for every term $a \in \mathcal{T}_e$ and every $i \geq 0$, $ef(a[\Uparrow^i (s)]) = ef(a)[\Uparrow^i (u)]$.

Note that, as for lemma 4.3, the substitution u obtained in corollary 4.4, is unique and is in normal form with respect to the system ef.

Lemma 4.5 If $a \in \mathcal{T}_d$ (resp. $\mathcal{T}_{dn}$), then $de(a)$ (resp. $dne(a)$) is in $\mathcal{T}_e$ and if $a \in \mathcal{T}_e$ (resp. $\mathcal{T}_v$) then $ef(a)$ (resp. $vf(a)$) is in $\mathcal{T}_f$.

Proof. We present the proof for $\mathcal{T}_d$ and de, being the proofs of the other statements similar. We proof, by induction on the structure of p that if $p \in \mathcal{O}_d$ then:

1. if $p \in \mathcal{T}_d$ then $de(p) \in \mathcal{T}_e$;

2. for all $q \in \mathcal{T}_e$, if $q[p] \in \mathcal{T}_d$ then for all $i \geq 0$, $de(q[\Uparrow^i (p)]) \in \mathcal{T}_e$.

The result then coincides with 1.

Let us start by the case where p is a term, for which we only check property 1 since property 2 holds vacuously.

- If $p = \underline{1}$, then $de(p) = \underline{1} \in \mathcal{T}_e$.

- If $p = (a\ b)$ with $a, b \in \mathcal{T}_d$ then $de(a\ b) = (de(a)\ de(b))$, by property 3 of lemma 4.2. By induction hypothesis (1), $de(a), de(b) \in \mathcal{T}_e$, which implies, by definition of $\mathcal{T}_e$, that $(de(a)\ de(b)) \in \mathcal{T}_e$.

- If $p = \lambda a$, with $a \in \mathcal{T}_d$, then $de(\lambda a) = \lambda(de(a))$, by property 4 of lemma 4.2, and again by induction hypothesis (1), $de(a) \in \mathcal{T}_e$ which implies, by definition of $\mathcal{T}_e$, that $\lambda(de(a)) \in \mathcal{T}_e$.

- If $p = a[s]$, with $a \in \mathcal{T}_d$, $s \in \mathcal{S}_d$, then $de(a[s]) = de(de(a)[s])$, by property 1 of lemma 4.2. By induction hypothesis (1) applied to a we have that $de(a) \in \mathcal{T}_e$ and then we can apply the induction hypothesis (2) to s and conclude that $de(de(a)[s]) \in \mathcal{T}_e$.

Now, if p is a substitution, we only check property 2 since property 1 holds vacuously.

- If $p = \uparrow$ or $p = id$, let q be any term in $\mathcal{T}_e$ such that $q[p] \in \mathcal{T}_d$ and fix $i \geq 0$ arbitrary. Then $de(q[\Uparrow^i (p)]) = q[\Uparrow^i (p)]$ by properties 5, 7 and 8 in lemma 4.2, so $de(q[\Uparrow^i (p)])$ is in $\mathcal{T}_e$ by definition of $\mathcal{T}_e$.

- If $p = \#(b)$, let q be any term in $\mathcal{T}_e$ such that $q[p] \in \mathcal{T}_d$ and fix $i \geq 0$ arbitrary. Then $de(q[\Uparrow^i (\#(b))]) = q[\Uparrow^i (\#(de(b)))]$ by properties 5 and 7 lemma 4.2, and by induction hypothesis (1), $de(b) \in \mathcal{T}_e$; therefore $\#(de(b)) \in \mathcal{S}_e$ and $\Uparrow^i (\#(de(b))) \in \mathcal{S}_e$. Since $q \in \mathcal{T}_e$, the result holds.

- If $p = \Uparrow (s)$, let q be any term in $\mathcal{T}_e$ such that $q[p] \in \mathcal{T}_d$ and fix $i \geq 0$ arbitrary. Since $q[\Uparrow (s)] \in \mathcal{T}_d$ we must have that $s \in \mathcal{S}_d$ so $q[s] \in \mathcal{T}_d$ and by induction hypothesis (2), we conclude that for all $j \geq 0$, $de(q[\Uparrow^j (s)]) \in \mathcal{T}_e$. In particular $de(q[\Uparrow^i (\Uparrow (s))]) \in \mathcal{T}_e$.

- If $p = s_1 \circ s_2$, then let q be any term in $\mathcal{T}_e$ such that $q[p] \in \mathcal{T}_d$ and fix $i \geq 0$ arbitrary. We have that $q[\Uparrow^i (s_1 \circ s_2)] \longrightarrow_{de} q[\Uparrow^i (s_1)][\Uparrow^i (s_2)]$. Since $\Uparrow^i (s_1) \in \mathcal{S}_d$, we can apply the induction hypothesis (2) to s_1 and conclude that $de(q[\Uparrow^i (s_1)]) \in \mathcal{T}_e$, and applying the induction hypothesis (2) to s_2 we conclude that $de(de(q[\Uparrow^i (s_1)])[\Uparrow^i (s_2)]) \in \mathcal{T}_e$. The result now follows from lemma 4.2, property 1.

The previous lemma is just a way of stating that de and dne-normal forms of terms do not contain composition of substitutions, dne and vf-normal forms of terms do not contain constants $\underline{n+1}$, and ef-normal forms of terms do not contain the substitution id.

5 Some considerations about the transformations

We make some relevant comments on the strategy used. Our aim is to prove β-strong normalization of the λ_d-calculus. One of the things we do is to transform this calculus by eliminating both composition of substitutions and the identity substitution, and this is done in two steps:

- via the complete rewrite system de we eliminate the composition operator, and generate an intermediate calculus λ_e, and

- via the complete rewrite system ef, we eliminate the identity substitution (from λ_d), thus obtaining a simpler calculus λ_f.

It is possible to put the complete rewrite systems de and ef together to form a system, say $\mathcal{DF}$, that turns out to be also complete. Therefore this system $\mathcal{DF}$ would define a function we could use to translate the calculus λ_d into the calculus λ_f. The problem is that instead of obtaining a simple partition of λ_d like the systems A and B, we obtain a complicated partition that is constituted by two conditional rewrite systems, making the problem much more difficult. To see why this is so, consider for example the rule (App) from λ_d. According to the system $\mathcal{DF}$, this rule has to be unfolded in a conditional way, i. e., there is no way of a priori deciding whether the rule is going to be an element of the subsystem A or B. In fact we get two rules $(App1)$ and $(App2)$, very much in the flavor of rules $(\widehat{App})$ and $(\widetilde{App})$ of the C and D-calculus, respectively, but with the important difference that both $(App1)$ and $(App2)$ would be conditional rules. More specifically the rules would be

$$\begin{array}{llll} (App1) & (a\ b)[s] & \longrightarrow & (a[s]\ b[s]) \ \text{ if } \mathcal{DF}((a\ b)[s]) \neq \mathcal{DF}(a[s]\ b[s]) \\ (App2) & (a\ b)[s] & \longrightarrow & (a[s]\ b[s]) \ \text{ if } \mathcal{DF}((a\ b)[s]) = \mathcal{DF}(a[s]\ b[s]) \end{array}$$

and then $(App1)$ would end up in the system A and the rule $(App2)$ would end up in the system B. But we would have to deal with two conditional rewrite systems, since we cannot say beforehand when the transformations of both the left-hand and right-hand sides of the rule (App) will be equal or different.

By separating the transformation in two, thus first eliminating composition and then the identity, we obtain one more calculus but the solution is cleaner and definitively simpler, since no conditional systems are involved.

6 λ_d and λ_{dn} preserve λ_e-strong normalization

In order to show that our translations allow us to go from λ_d and λ_{dn} to λ_f, we will first prove that both calculi preserve λ_e-strong normalization. Combining this result with the facts that λ_e preserves λ_f-strong normalization and λ_f preserves β-strong normalization we will be able to conclude that λ_d and λ_{dn} preserve β-strong normalization. Let us start by checking all the properties needed in order to apply theorem 3.1.

Proposition 6.1 The B-calculus is strongly normalizing.

Proof. B is a sub-calculus of $\sigma_{\Uparrow}$, which is proved to be strongly normalizing in [HL89].

Lemma 6.2 Let p be an object in $\mathcal{O}_d$ such that $p \longrightarrow_{\lambda_d} p'$.

1. If p is a term, then
   $p \longrightarrow_A p'$ implies $de(p') \longrightarrow^+_{\lambda_e} de(p')$
   $p \longrightarrow_B p'$ implies $de(p') = de(p')$

2. For every term d in $\mathcal{T}_e$ such that $d[p]$ is a term in $\mathcal{T}_d$, and every $i \geq 0$, we have
   $p \longrightarrow_A p'$ implies $de(d[\Uparrow^i(p)]) \longrightarrow^+_{\lambda_e} de(d[\Uparrow^i(p')])$
   $p \longrightarrow_B p'$ implies $de(d[\Uparrow^i(p)]) = de(d[\Uparrow^i(p')])$

Proof. We can reason according to the two following cases: the reduction takes place either at the root or strictly below the root.

- If $p \longrightarrow_{\lambda_d} p'$ takes place at the root of p, we proceed by case analysis, showing only property 1 (resp. 2) if p is a term (resp. a substitution) since the other one holds vacuously. We only show here the more interesting cases, the other ones being similar.
  - •• If $p = (\lambda a)b \longrightarrow_{Beta} a[\#(b)] = p'$, then

$$\begin{array}{ll} de((\lambda a)b) & =_{lemma\ 4.2} \\ (\lambda de(a))de(b) & \longrightarrow_{\lambda_e} \\ de(a)[\#(de(b))] & =_{lemma\ 4.2} \\ de(a[\#(b)]) & \end{array}$$

•• If $p = (a\ b)[s] \longrightarrow_{App} (a[s]\ b[s]) = p'$, then

$$\begin{array}{ll}
de((a\ b)[s]) & =_{lemmas\ 4.2,4.3} \\
de((a\ b))[u_1]\ldots[u_k] & =_{lemma\ 4.2} \\
(de(a)\ de(b))[u_1]\ldots[u_k] & \longrightarrow^+{}_{\lambda_e} \\
(de(a)[u_1]\ldots[u_k])\ de(b)[u_1]\ldots[u_k] & =_{lemma\ 4.3} \\
(de(a[s])\ de(b[s])) & =_{lemma\ 4.2} \\
de(a[s]\ b[s]) &
\end{array}$$

•• If $p = \underline{1}[\#(a)\circ s] \longrightarrow_{FVar2} a[s] = p'$, then

$$\begin{array}{ll}
de(\underline{1}[\#(a)\circ s]) & = \\
de(\underline{1}[\#(a)][s]) & =_{lemmas\ 4.2,4.3} \\
\underline{1}[\#(de(a))][u_1]\ldots[u_k] & \longrightarrow_{\lambda_e} \\
de(a)[u_1]\ldots[u_k] & =_{lemma\ 4.3} \\
de(a[s]) &
\end{array}$$

•• If $p = \underline{1}[\Uparrow(s)\circ t] \longrightarrow_{FVarLift2} \underline{1}[t] = p'$, then

$$\begin{array}{ll}
de(\underline{1}[\Uparrow(s)\circ t]) & = \\
de(\underline{1}[\Uparrow(s)][t]) & =_{lemmas\ 4.2,4.3} \\
\underline{1}[\Uparrow(u_1)]\ldots[\Uparrow(u_k)][v_1]\ldots[v_z] & \longrightarrow^+{}_{\lambda_e} \\
\underline{1}[v_1]\ldots[v_z] & =_{lemma\ 4.3} \\
de(\underline{1}[t]) &
\end{array}$$

•• If $p = (s_1\circ s_2)\circ s_3 \longrightarrow_{Ass} s_1\circ(s_2\circ s_3) = p'$, then

$$\begin{array}{ll}
de(d[\Uparrow^i((s_1\circ s_2)\circ s_3)]) & = \\
de(d[\Uparrow^i(s_1\circ s_2)][\Uparrow^i(s_3)]) & = \\
de(d[\Uparrow^i(s_1)][\Uparrow^i(s_2)][\Uparrow^i(s_3)]) & = \\
de(d[\Uparrow^i(s_1)][\Uparrow^i(s_2\circ s_3)]) & = \\
de(d[\Uparrow^i(s_1\circ(s_2\circ s_3))]) &
\end{array}$$

•• If $p = \uparrow\circ(\#(a)\circ s) \longrightarrow_{Shift2} s$, then

$$\begin{array}{ll}
de(d[\Uparrow^i(\uparrow\circ(\#(a)\circ s))]) & = \\
de(d[\Uparrow^i(\uparrow)][\Uparrow^i(\#(a))][\Uparrow^i(s)]) & =_{lemmas\ 4.2,4.3} \\
de(d)[\Uparrow^i(\uparrow)][\Uparrow^i(\#(de(a)))][\Uparrow^i(u_1)]\ldots[\Uparrow^i(u_k)] & \longrightarrow_{\lambda_e} \\
de(d)[\Uparrow^i(id)][\Uparrow^i(u_1)]\ldots[\Uparrow^i(u_k)] & \longrightarrow^+{}_{\lambda_e} \\
de(d)[\Uparrow^i(u_1)]\ldots[\Uparrow^i(u_k)] & =_{lemma\ 4.2} \\
de(d[\Uparrow^i(s)]) &
\end{array}$$

•• If $p = \uparrow\circ\Uparrow(s) \longrightarrow_{ShiftLift1} s\circ\uparrow = p'$, then

$$\begin{array}{ll}
de(d[\Uparrow^i(\uparrow\circ\Uparrow(s))]) & = \\
de(d[\Uparrow^i(\uparrow)][\Uparrow^{i+1}(s)]) & =_{lemmas\ 4.2,4.3} \\
de(d)[\Uparrow^i(\uparrow)][\Uparrow^{i+1}(u_1)]\ldots[\Uparrow^{i+1}(u_k)] & \longrightarrow^+{}_{\lambda_e} \\
de(d)[\Uparrow^i(u_1)]\ldots[\Uparrow^i(u_k)][\Uparrow^i(\uparrow)] & =_{lemma\ 4.3} \\
de(d[\Uparrow^i(s)][\Uparrow^i(\uparrow)]) & = \\
de(d[\Uparrow^i(s\circ\uparrow)]) &
\end{array}$$

•• If $p = \uparrow \circ (\Uparrow(s) \circ t) \longrightarrow_{ShiftLift2} s \circ (\uparrow \circ t) = p'$, then

$$\begin{array}{ll}
de(d[\Uparrow^i(\uparrow \circ (\Uparrow(s) \circ t))]) & = \\
de(d[\Uparrow^i(\uparrow)][\Uparrow^{i+1}(s)][\Uparrow^i(t)]) & =_{lemmas\ 4.2,4.3} \\
de(d)[\Uparrow^i(\uparrow)][\Uparrow^{i+1}(u_1)]\ldots[\Uparrow^{i+1}(u_k)][\Uparrow^{i+1}(v_1)]\ldots[\Uparrow^{i+1}(v_z)] & \longrightarrow^+_{\lambda_e} \\
de(d)[\Uparrow^i(u_1)]\ldots[\Uparrow^i(u_k)][\Uparrow^i(\uparrow)][\Uparrow^{i+1}(v_1)]\ldots[\Uparrow^{i+1}(v_z)] & =_{lemma\ 4.3} \\
de(d[\Uparrow^i(s)][\Uparrow^i(\uparrow)][\Uparrow^i(t)]) & = \\
de(d[\Uparrow^i(s \circ (\uparrow \circ t))]) &
\end{array}$$

•• If $p = \Uparrow(s) \circ \Uparrow(t) \longrightarrow_{Lift1} \Uparrow(s \circ t) = p'$, then

$$\begin{array}{ll}
de(d[\Uparrow^i(\Uparrow(s) \circ \Uparrow(t))]) & = \\
de(d[\Uparrow^{i+1}(s)][\Uparrow^{i+1}(t)]) & = \\
de(d[\Uparrow^i(\Uparrow(s \circ t))]) &
\end{array}$$

•• If $p = \Uparrow(id) \longrightarrow_{LiftId} id = p'$, then

$$\begin{array}{ll}
de(d[\Uparrow^i(\Uparrow(id))]) & =_{lemma\ 4.2} \\
de(d)[\Uparrow^{i+1}(id)] & \longrightarrow_{\lambda_e} \\
de(d)[\Uparrow^i(id)] & =_{lemma\ 4.2} \\
de(d[\Uparrow^i(id)]) &
\end{array}$$

- If $p \longrightarrow_{\lambda_d} p'$ does not take place at the root of p, one can easily proceed by induction on the structure of p, using lemmas 4.2 and 4.3.

Corollary 6.3 Let a be a term in $\mathcal{T}_d$. If $a \longrightarrow_A b$, then $de(a) \longrightarrow^+_{\lambda_e} de(b)$, and if $a \longrightarrow_B b$, then $de(a) = de(b)$.

Taking $\mathcal{O}$ as $\mathcal{T}_d$, R_1 as A, R_2 as B, $\mathcal{O}'$ as $\mathcal{T}_e$, R' as λ_e and $\mathcal{T}$ as de in theorem 3.1, we can conclude that :

Corollary 6.4 λ_d preserves λ_e-strong normalization.

Proof. By lemma 4.2, for every term t in $\mathcal{T}_e$ we have $de(t) = t$. Combining this with proposition 6.1 and corollary 6.3 allows us to use theorem 3.1 (with $\mathcal{O}$ as $\mathcal{T}_d$, R_1 as A, R_2 as B, $\mathcal{O}'$ as $\mathcal{T}_e$, R' as λ_e and $\mathcal{T}$ as de), to conclude that λ_d preserves λ_e-strong normalization.

Lemma 6.5 Let p be an object in $\mathcal{O}_{dn}$ such that $p \longrightarrow_{\lambda_{dn}} p'$. Then

1. If p is a term, then

   $p \longrightarrow_{A_n} p'$ implies $dne(p) \longrightarrow^+_{\lambda_e} dne(p')$
   $p \longrightarrow_{B_n} p'$ implies $dne(p) = dne(p')$

2. For every term $d \in \mathcal{T}_e$ such that $d[p]$ is a term in $\mathcal{T}_{dn}$, and every $i \geq 0$ we have

   $p \longrightarrow_{A_n} p'$ implies $dne(d[\Uparrow^i(p)]) \longrightarrow^+_{\lambda_e} dne(d[\Uparrow^i(p')])$
   $p \longrightarrow_{B_n} p'$ implies $dne(d[\Uparrow^i(p)]) = dne(d[\Uparrow^i(p')])$

Proof. The proof proceeds exactly as in lemma 6.2, using lemmas 4.2 and 4.3.

Corollary 6.6 Let a be a term in $\mathcal{T}_{dn}$. If $a \longrightarrow_{A_n} b$, then $dne(a) \longrightarrow^+_{\lambda_e} dne(b)$, and if $a \longrightarrow_{B_n} b$, then $dne(a) = dne(b)$.

In order to see that the λ_{dn}-calculus preserves λ_e-strong normalization, we still need to show that the the B_n-calculus is strongly normalizing.

Proposition 6.7 The B_n-calculus is strongly normalizing.

Proof. We show that the B_n-calculus is strongly normalizing by defining a mapping from terms in B_n to a well-founded set such that reductions in the B_n-calculus are mapped to descending chains.

Let $\mathcal{A} = \mathbb{N}_1 \times \mathbb{N}_1$, where $\mathbb{N}_1$ is the set of natural numbers ≥ 1, ordered by $>_{lex}$, the lexicographic extension of $>$, the usual order on $\mathbb{N}$.

Now, for every object o in the B_n-calculus we define a measure $\mathcal{M}(o)$ by induction on the structure of a as follows:

$$
\begin{array}{lcll}
\mathcal{M}(\underline{n}) & = & \langle 1, 1 \rangle & \text{for any } n \geq 1 \\
\mathcal{M}(\uparrow) & = & \langle 1, 1 \rangle & \\
\mathcal{M}(id) & = & \langle 1, 1 \rangle & \\
\mathcal{M}(a\ b) & = & \langle a_1 + b_1, a_2 + b_2 \rangle & \text{where } \mathcal{M}(a) = \langle a_1, a_2 \rangle \text{ and } \mathcal{M}(b) = \langle b_1, b_2 \rangle \\
\mathcal{M}(u \circ w) & = & \langle u_1 + w_1, 2 \cdot u_2 + w_2 \rangle & \text{where } \mathcal{M}(u) = \langle u_1, u_2 \rangle \text{ and } \mathcal{M}(w) = \langle w_1, w_2 \rangle \\
\mathcal{M}(a[s]) & = & \langle 2 \cdot a_1 + s_1, a_2 + s_2 \rangle & \text{where } \mathcal{M}(a) = \langle a_1, a_2 \rangle \text{ and } \mathcal{M}(s) = \langle s_1, s_2 \rangle \\
\mathcal{M}(\Uparrow (s)) & = & \langle s_1 + 1, s_2 \rangle & \text{where } \mathcal{M}(s) = \langle s_1, s_2 \rangle
\end{array}
$$

Now, in order to show that the B_n-calculus is strongly normalizing, we only need to see that for every reduction $a \longrightarrow_{B_n} b$ we have $\mathcal{M}(a) >_{lex} \mathcal{M}(b)$. It is not difficult to see that this measure is stable under substitution and monotone with respect to contexts, so in order to see that $a \longrightarrow_{B_n} b$ implies $\mathcal{M}(a) >_{lex} \mathcal{M}(b)$ it suffices to check this property for the reduction rules of B_n. We just show here two cases to illustrate the principle of the proof:

- For the rewriting rule $(c[s])[t] \longrightarrow c[s \circ t]$; we have

$$
\begin{array}{ll}
\mathcal{M}((c[s])[t]) & = \\
\langle 4 \cdot c_1 + 2 \cdot s_1 + t_1, c_2 + s_2 + t_2 \rangle & >_{lex} \\
\langle 2 \cdot c_1 + s_1 + t_1, c_2 + 2 \cdot s_2 + t_2 \rangle & = \\
\mathcal{M}(c[s \circ t]) &
\end{array}
$$

  where $\mathcal{M}(c) = \langle c_1, c_2 \rangle$, $\mathcal{M}(s) = \langle s_1, s_2 \rangle$ and $\mathcal{M}(t) = \langle t_1, t_2 \rangle$.

- For the rewriting rule $(s \circ t) \circ u \longrightarrow s \circ (t \circ u)$; we have

$$
\begin{array}{ll}
(\langle s_1, s_2 \rangle \circ \langle t_1, t_2 \rangle) \circ \langle u_1, u_2 \rangle & = \\
\langle s_1 + t_1 + u_1, 4 \cdot s_2 + 2 \cdot t_2 + u_2 \rangle & >_{lex} \\
\langle s_1 + t_1 + u_1, 2 \cdot s_2 + 2 \cdot t_2 + u_2 \rangle & = \\
\langle s_1, s_2 \rangle \circ (\langle t_1, t_2 \rangle \circ \langle u_1, u_2 \rangle) &
\end{array}
$$

  where $\mathcal{M}(s) = \langle s_1, s_2 \rangle$, $\mathcal{M}(t) = \langle t_1, t_2 \rangle$ and $\mathcal{M}(u) = \langle u_1, u_2 \rangle$.

Taking $\mathcal{O}$ as $\mathcal{T}_{dn}$, R_1 as A_n, R_2 as B_n, $\mathcal{O}'$ as $\mathcal{T}_e$, R' as λ_e and $\mathcal{T}$ as dne in theorem 3.1, we can conclude that :

Corollary 6.8 The λ_{dn}-calculus preserves λ_e-strong normalization.

Proof. By lemma 4.2, for every term t in $\mathcal{T}_e$ we have $dne(t) = t$. Combining this with proposition 6.7 and corollary 6.6 allows us to use theorem 3.1 (with $\mathcal{O}$ as $\mathcal{T}_{dn}$, R_1 as A_n, R_2 as B_n, $\mathcal{O}'$ as $\mathcal{T}_e$, R' as λ_e and $\mathcal{T}$ as dne), to conclude that λ_{dn} preserves λ_e-strong normalization.

7 λ_e and λ_v preserve λ_f-strong normalization

Following the approach of section 6, we will actually prove that the $(C \cup D)$-calculus preserves λ_f-strong normalization. As remarked in section 2, whenever p and p' are terms in $\mathcal{T}_e$, then $p \longrightarrow_{\lambda_e} p'$ if and only if $p \longrightarrow_{C \cup D} p'$, so we can then immediately conclude that λ_e preserves λ_f-strong normalization.

Proposition 7.1 The D-calculus is strongly normalizing.

Proof. We show that the D-calculus is strongly normalizing by defining a mapping from terms of D to a well-founded set such that reductions in the D-calculus are mapped to descending chains.

Let $\mathcal{A} = \mathbb{N}_2 \times \mathbb{N}_2$, where $\mathbb{N}_2$ is the set of natural numbers ≥ 2, ordered by $>_{lex}$, the lexicographic extension of $>$, the usual order on $\mathbb{N}$.

Now, for every object o in the D-calculus we define a measure $\mathcal{M}(o)$ by induction on the structure of a as follows:

$$
\begin{array}{lcll}
\mathcal{M}(\underline{1}) & = & \langle 2, 2\rangle & \\
\mathcal{M}(id) & = & \langle 2, 2\rangle & \\
\mathcal{M}(\uparrow) & = & \langle 2, 2\rangle & \\
\mathcal{M}(a\ b) & = & \langle a_1 + b_1 + 1, a_2 + b_2 + 1\rangle & \text{where } \mathcal{M}(a) = \langle a_1, a_2\rangle \text{ and } \mathcal{M}(b) = \langle b_1, b_2\rangle \\
\mathcal{M}(\lambda a) & = & \langle a_1 + 2, a_2 + 2\rangle & \text{where } \mathcal{M}(a) = \langle a_1, a_2\rangle \\
\mathcal{M}(a[s]) & = & \langle a_1 \cdot s_1, a_2 \cdot s_2\rangle & \text{where } \mathcal{M}(a) = \langle a_1, a_2\rangle \text{ and } \mathcal{M}(s) = \langle s_1, s_2\rangle \\
\mathcal{M}(\Uparrow(s)) & = & \langle s_1, s_2 + 1\rangle & \text{where } \mathcal{M}(s) = \langle s_1, s_2\rangle \\
\mathcal{M}(\#(a)) & = & \langle a_1, a_2\rangle & \text{where } \mathcal{M}(a) = \langle a_1, a_2\rangle
\end{array}
$$

In order to show that the D-calculus is strongly normalizing, we need to see that for every reduction $a \longrightarrow_D b$ we have $\mathcal{M}(a) >_{lex} \mathcal{M}(b)$. But it is not difficult to see that this measure is stable under substitution and monotone with respect to contexts, so as remarked in the proof of proposition 6.7, due to the stability and monotonicity of the measure, it suffices to show the property for the rewriting rules. We just show here one case to illustrate the principle of the proof:

- For the rule $(c\ d)[\Uparrow^i(id)] \longrightarrow_{\widetilde{App}} (c[\Uparrow^i(id)]\ d[\Uparrow^i(id)])$, we have

$$
\begin{array}{lr}
\mathcal{M}((c\ d)[\Uparrow^i(id)]) & = \\
\langle (c_1 + d_1 + 1) \cdot 2, (c_2 + d_2 + 1) \cdot (s + i)\rangle & >_{lex} \\
\langle 2 \cdot c_1 + 2 \cdot d_1 + 1, (c_2 + d_2) \cdot (2 + i) + 1\rangle & = \\
\mathcal{M}(c[\Uparrow^i(id)]\ d[\Uparrow^i(id)]) &
\end{array}
$$

  where $\mathcal{M}(c) = \langle c_1, c_2\rangle$ and $\mathcal{M}(d) = \langle d_1, d_2\rangle$.

- For the rule $a[\Uparrow^{i+1}(id)] \longrightarrow_{LiftId} a[\Uparrow^i(id)]$; for any $i \geq 0$, we have:

$$
\begin{array}{lclcll}
\langle a_1, a_2\rangle[\Uparrow^{i+1}(id)] & = & \langle a_1, a_2\rangle[\langle 2, 3 + i\rangle] & = & \langle 2 \cdot a_1, a_2 \cdot (3 + i)\rangle & >_{lex} \\
\langle 2 \cdot a_1, a_2 \cdot (2 + i)\rangle & = & \langle a_1, a_2\rangle[\langle 2, 2 + i\rangle] & = & \langle a_1, a_2\rangle[\Uparrow^i(\langle 2, 2\rangle)] & = \langle a_1, a_2\rangle[\Uparrow^i(id)]
\end{array}
$$

  where $\mathcal{M}(a) = \langle a_1, a_2\rangle$.

Lemma 7.2 Let p be a term in $\mathcal{T}_e$ such that $p \longrightarrow_{\lambda_e} p'$. Then,
$p \longrightarrow_C p'$ implies $ef(p) \longrightarrow^+_{\lambda_f} ef(p')$
$p \longrightarrow_D p'$ implies $ef(p) = ef(p')$

Proof. We can reason according to the two following cases: the reduction takes place at the root or strictly below the root.

- If $p \longrightarrow_{\lambda_e} p'$ takes place at the root of p, we proceed by case-analysis:
  - If $p \longrightarrow_{Beta,FVar,Shift} p'$, then $ef(p) \longrightarrow_{\lambda_f} ef(p')$ immediately holds using lemma 4.2.
  - If $p = (a\ b)[\Uparrow^i(s)] \longrightarrow_{\widehat{App}} (a[\Uparrow^i(s)]\ b[\Uparrow^i(s)]) = p'$, where $s = \uparrow$ or $s = \#(c)$, then

$$\begin{array}{ll} ef((a\ b)[\Uparrow^i(s)]) & =_{corollary\ 4.4} \\ (ef(a)\ ef(b))[\Uparrow^i(u)] & \longrightarrow^+_{\lambda_f} \\ (ef(a)[\Uparrow^i(u)]\ ef(b)[\Uparrow^i(u)]) & =_{corollary\ 4.4} \\ (ef(a[\Uparrow^i(s)])\ ef(b[\Uparrow^i(s)])) & =_{lemma\ 4.2} \\ ef(a[\Uparrow^i(s)]\ b[\Uparrow^i(s)]) & \end{array}$$

  - If $p = (\lambda a)[\Uparrow^i(s)] \longrightarrow_{\widehat{Lambda}} \lambda(a[\Uparrow^{i+1}(s)]) = p'$, where $s = \uparrow$ or $s = \#(c)$, then

$$\begin{array}{ll} ef((\lambda a)[\Uparrow^i(s)]) & =_{corollary\ 4.4} \\ ef(\lambda a)[\Uparrow^i(u)] & =_{lemma\ 4.2} \\ (\lambda ef(a))[\Uparrow^i(u)] & \longrightarrow^+_{\lambda_f} \\ \lambda(ef(a)[\Uparrow^{i+1}(u)]) & =_{corollary\ 4.4} \\ \lambda ef(a[\Uparrow^{i+1}(s)]) & =_{lemma\ 4.2} \\ ef(\lambda(a[\Uparrow^{i+1}(s)])) & \end{array}$$

  - If $p = \underline{1}[\Uparrow^{i+1}(s)] \longrightarrow_{\widehat{FVarLift}} \underline{1} = p'$, where $s = \uparrow$ or $s = \#(a)$, then

$$ef(\underline{1}[\Uparrow^{i+1}(s)]) =_{corollary\ 4.4} \underline{1}[\Uparrow^{i+1}(u)] \longrightarrow^+_{\lambda_f} \underline{1} = ef(\underline{1})$$

  - If $p = a[\Uparrow^i(\uparrow)][\Uparrow^{i+j+1}(s)] \longrightarrow_{\widehat{ShiftLift}} a[\Uparrow^{i+j}(s)][\Uparrow^i(\uparrow)] = p'$, where $s = \uparrow$ or $s = \#(c)$, then

$$\begin{array}{ll} ef(a[\Uparrow^i(\uparrow)][\Uparrow^{i+j+1}(s)]) & =_{corollary\ 4.4} \\ ef(a)[\Uparrow^i(\uparrow)][\Uparrow^{i+j+1}(u)] & \longrightarrow_{\lambda_f} \\ ef(a)[\Uparrow^{i+j}(u)][\Uparrow^i(\uparrow)] & =_{corollary\ 4.4} \\ ef(a[\Uparrow^{i+j}(s)][\Uparrow^i(\uparrow)]) & \end{array}$$

  - If $p \longrightarrow_{\widetilde{App},\widetilde{Lambda},\widetilde{FVarLift},\widetilde{ShiftLift},LiftId,Id} p'$, then $ef(p) = ef(p')$ immediately holds by lemma 4.2.
- If $p \longrightarrow_{\lambda_e} p'$ does not take place at the root of p, one can proceed by induction on the structure of p, using lemma 4.2 and corollary 4.4.

As remarked in section 2, fact 1, whenever p and p' are terms in $\mathcal{T}_e$, then $p \longrightarrow_{\lambda_e} p'$ if and only if $p \longrightarrow_{C \cup D} p'$. As a consequence, taking $\mathcal{O}$ as $\mathcal{T}_e$, R_1 as C, R_2 as D, $\mathcal{O}'$ as $\mathcal{T}_f$, R' as λ_f and $\mathcal{T}$ as ef in theorem 3.1, we can conclude that:

Corollary 7.3 The calculus λ_e preserves λ_f-strong normalization.

Proof. By lemma 4.2, for every term t in $\mathcal{T}_f$ we have $ef(t) = t$. This combined with proposition 7.1 and lemma 7.2 gives us that all conditions of theorem 3.1 are satisfied (with $\mathcal{O}$ as $\mathcal{T}_e$, R_1 as C, R_2 as D, $\mathcal{O}'$ as $\mathcal{T}_f$, R' as λ_f and $\mathcal{T}$ as ef) and therefore from application of theorem 3.1, we conclude that $C \cup D$ preserves λ_f-strong normalization. Due to the observation above, the result follows.

The previous lemma is just a way of stating that vf-normal forms of terms from $\mathcal{T}_v$ do not contain variables different from $\underline{1}$.

Lemma 7.4 Let p be a term in $\mathcal{T}_v$ such that $p \longrightarrow_{\lambda_v} p'$. Then,
$p \longrightarrow_{\lambda_v - VarShift} p'$ implies $vf(p') \longrightarrow^{+}_{\lambda_f} vf(p')$
$p \longrightarrow_{VarShift} p'$ implies $vf(p') = vf(p')$

Proof. The proof proceeds as in lemma 7.2, using lemma 4.2.

Corollary 7.5 Let a be a term in $\mathcal{T}_v$. If $a \longrightarrow_{\lambda_v - VarShift} b$, then $vf(a) \longrightarrow^{+}_{\lambda_f} vf(b)$, and if $a \longrightarrow_{VarShift} b$, then $vf(a) = vf(b)$.

We remark that the system containing the rule $VarShift : \underline{n}[\uparrow] \longrightarrow \underline{n+1}$ is strongly normalizing because whenever $a \longrightarrow_{VarShift} b$, the number of occurrences of the constant $\uparrow$ in a is strictly greater than the corresponding number in b. Thus, taking $\mathcal{O}$ as $\mathcal{T}_v$, R_1 as $\lambda_v - VarShift$, R_2 as $VarShift$, $\mathcal{O}'$ as $\mathcal{T}_f$, R' as λ_f and $\mathcal{T}$ as vf in theorem 3.1, we can conclude that:

Corollary 7.6 λ_v preserves λ_f-strong normalization.

Proof. By definition of vf, for every term t in $\mathcal{T}_f$ we have $vf(t) = t$. Therefore, corollary 8.16, lemma 4.1 and corollary 7.5 allow us to conclude, using theorem 3.1, that λ_v preserves λ_f-strong normalization.

8 λ_f preserves β-strong normalization

In this section we show that the λ_f-calculus preserves β-strong normalization for pure terms. The proof technique used is a minimality construction similar to the one used in [BBLRD95].

Lemma 8.1 Let a, c, d be terms in $\mathcal{O}_f$. If $a \longrightarrow_{\lambda_f} C\langle\!\langle d[\Uparrow^i (\#(c))]\rangle\!\rangle$, for some context C and some $i \geq 0$, then either

- $a = C'\langle\!\langle d'[\Uparrow^j (\#(c'))]\rangle\!\rangle$, for some context C', some term d', some index j, and some term c', such that $c' \longrightarrow_{\lambda_f} c$ or $c' = c$; or
- $a = C\langle\!\langle (\lambda d)c \rangle\!\rangle$.

Proof. Let b be the object $C\langle\!\langle d[\Uparrow^i (\#(c))]\rangle\!\rangle$. If $a \longrightarrow_{\lambda_f} b$, then the reduction can take place either at the root or within a non-trivial context. If the reduction step $a \longrightarrow_{\lambda_f} b$ takes place at the root of the object a, then we proceed by case-analysis as follows. First of all note that whenever $d[\Uparrow^i (\#(c))]$ occurs in a subterm of b which is also a subterm of a, then the first case of the statement trivially holds for some context C', $d' = d$, $j = i$ and $c' = c$.

Now, if $a \longrightarrow_{FVar, FVarLift, Shift} b$, then no occurrence of $d[\Uparrow^i (\#(c))]$ has been *created* by this reduction step so the property vacuously holds. Let us then study all the remaining cases.

- If $a \longrightarrow_{Beta} b$, then $a = (\lambda a_1)a_2$ and $b = a_1[\#(a_2)]$. We must have $C = \Box$, $a_1 = d$, $a_2 = c$, $i = 0$, and the second case of the statement holds.
- If $a \longrightarrow_{App} b$, then $a = (a_1\ a_2)[s]$ and $b = (a_1[s]\ a_2[s])$. We have that d is a_k (for $k = 1$ or $k = 2$) and s is $\Uparrow^i (\#(c))$, so the first case of the statement holds with $C' = \Box$, $d' = (a_1\ a_2)$, $j = i$ and $c' = c$.

- If $a \longrightarrow_{Lambda} b$, then $a = (\lambda a_1)[s]$ and $b = (\lambda a_1[\Uparrow (s)])$. We have that d is a_1 and s is $\Uparrow^{i-1} (\#(c))$, so the first case of the statement holds with $C' = \Box$, $d' = (\lambda a_1)$, $j = i - 1$ and $c' = c$.

- If $a \longrightarrow_{ShiftLift} b$, then $a = a_1[\Uparrow^k (\uparrow)][\Uparrow^{k+1} (s)]$ and $b = a_1[\Uparrow^k (s)][\Uparrow^k (\uparrow)]$. We have that d is a_1 and s is $\Uparrow^{i-k} (\#(c))$, so the first case of the statement holds with $C' = \Box$, $d' = a_1[\Uparrow^k (\uparrow)]$, $j = i + 1$ and $c' = c$.

Suppose now that the reduction does not occur at the root. We proceed by induction on the structure of the object a.

- In the cases $a = \underline{1}$ and $a = \uparrow$, the property holds vacuously.

- If $a = (a_1\ a_2)$, then either $b = (b_1\ a_2)$ or $b = (a_1\ b_2)$ with $a_i \longrightarrow_{\lambda_f} b_i$. We only consider the first case as the second one can be treated similarly. Now, the subterm $d[\Uparrow^i (\#(c))]$ is either in b_1 or in a_2. In the first case the property follows by induction hypothesis, in the second one the first statement of the property trivially holds.

- If $a = \lambda p_1$ (resp. $a = \#(p_1)$, $a = \Uparrow (p_1)$), then $b = \lambda p_2$ (resp. $b = \#(p_2)$, $b = \Uparrow (p_2)$) with $p_1 \longrightarrow_{\lambda_f} p_2$, and $d[\Uparrow^i (\#(c))]$ in p_2. The property then follows by induction hypothesis.

- If $a = a_1[s_1]$, then there are two cases to consider:

  - $b = a_2[s_1]$ with $a_1 \longrightarrow_{\lambda_f} a_2$. If $d[\Uparrow^i (\#(c))]$ is in a_2, then the property follows by induction hypothesis, and if it is in s_1 the first case of the statement trivially holds. So the only case that remains is when the term $d[\Uparrow^i (\#(c))]$ overlaps with b and is *created* by the reduction step $a_1 \longrightarrow_{\lambda_f} a_2$. As a consequence a_2 must be d and s_1 must be the substitution $\Uparrow^i (\#(c))$, so the first statement of the property holds for $C' = \Box$, $d' = a_1$, $j = i$ and $c' = c$.
  - $b = a_1[s_2]$ with $s_1 \longrightarrow_{\lambda_f} s_2$. If $d[\Uparrow^i (\#(c))]$ is in s_2, then the property follows by induction hypothesis and if it is in a_1 the first case of the statement trivially holds. So the only case that remains is when the term $d[\Uparrow^i (\#(c))]$ overlaps with b and is *created* by the reduction step $s_1 \longrightarrow_{\lambda_f} s_2$. As a consequence a_1 must be d, s_2 must be the substitution $\Uparrow^i (\#(c))$ and $s_1 = \Uparrow^i (\#(e))$ such that $e \longrightarrow_{\lambda_f} c$. Therefore the first statement of the property holds with $C' = \Box$, $d' = a_1$, $j = i$ and $c' = e$.

Lemma 8.2 Let $a_1, \ldots a_{n+1} \in \mathcal{T}_f$ such that

$$a_1 \longrightarrow_{\lambda_f} a_2 \longrightarrow_{\lambda_f} \ldots \longrightarrow_{\lambda_f} a_{n+1} = C\langle\!\langle d[\Uparrow^i (\#(c))]\rangle\!\rangle$$

then either

- $a_1 = C'\langle\!\langle d'[\Uparrow^j (\#(c'))]\rangle\!\rangle$, for some context C', some term d', some index j, and some term c' such that $c' \longrightarrow^*_{\lambda_f} c$; or

- there exists $1 \leq i \leq n$, such that $a_i = C'\langle\!\langle(\lambda d')c'\rangle\!\rangle$, for some context C', some term d' and some term c', such that $c' \longrightarrow^*_{\lambda_f} c$.

Proof. By induction on n and using lemma 8.1.

Theorem 8.3 The f-calculus is locally confluent.

Proof. This can be done automatically, for example by using CIME [Mar93].

Theorem 8.4 The f-calculus is strongly normalizing.

Proof. We proceed exactly as in the proof of proposition 7.1, using the same measure.

By Newman's lemma we obtain:

Corollary 8.5 The f-calculus is confluent.

Theorem 8.4 and Corollary 8.5 say that the system f defines a function, namely the function that given any object in $\mathcal{O}_f$ associates to it its unique normal form with respect to the f-calculus. The function f satisfies:

Lemma 8.6 (Some Properties of λ_f) The function f has the following properties:

1. $f(a[s]) = f(f(a)[s])$, for any term $a \in \mathcal{T}_f$ and substitution $s \in \mathcal{S}_f$;
2. $f(\underline{1}) = \underline{1}$;
3. $f(a\ b) = (f(a)\ f(b))$, for any terms $a, b \in \mathcal{T}_f$;
4. $f(\lambda a) = \lambda(f(a))$, for any term $a \in \mathcal{T}_f$;

Proof. Similar observations as in lemma 4.2.

Now, we are ready to prove the projection lemma:

Theorem 8.7 (Projection lemma) Let a and b be terms in $\mathcal{T}_f$. If $a \longrightarrow_{Beta}\ b$, then $f(a) \longrightarrow^*_\beta f(b)$.

Proof. If we prove that f is a *basic calculus fitting the general scheme* as defined in [Kes96], then the statement is a consequence of lemma 3.6 in [Kes96]. We have to show the following properties:

- Basic Conditions:
  - f is strongly normalizing: by theorem 8.4.
  - f is confluent: by corollary 8.5.
  - f-normal forms of terms are pure terms: by corollary 8.15.
  - $f(a\ b) = (f(a)\ f(b))$ and $f(\lambda a) = \lambda f(a)$, which hold by lemma 8.6.
  - For every substitution s, we have $\underline{1}[\Uparrow(s)] =_f \underline{1}$: this is true by the $FVarLift$ reduction rule.
  - $\forall m \geq 1$ and for every substitution s, we have $\underline{m+1}[\Uparrow(s)] =_f \underline{m}[s][\uparrow]$: indeed,

$$\underline{m+1}[\Uparrow(s)] = \underline{1}[\uparrow]^{m-1}[\uparrow][\Uparrow(s)] =_{ShiftLift} \underline{1}[\uparrow]^{m-1}[s][\uparrow] = \underline{m}[s][\uparrow]$$

  - $\forall m \geq 1$ and for every term k, $\underline{m+1}[\#(k)] =_f \underline{m}$:

$$\underline{m+1}[\#(k)] = \underline{1}[\uparrow]^{m-1}[\uparrow][\#(k)] =_{Shift} \underline{1}[\uparrow]^{m-1} = \underline{m}$$

  - For every term k, $\underline{1}[\#(k)] =_f k$: this holds by the $FVar$ rule.

– $\forall m \geq 1$, $\underline{m}[\uparrow] =_f \underline{m+1}$: indeed, we have:

$$\underline{m}[\uparrow] = \underline{1}[\uparrow]^{m-1}[\uparrow] = \underline{m+1}$$

- General Scheme: we have to verify that for every variable $\underline{m} \in \mathcal{T}_f$ and every substitution constructor ξ of arity q, one of the following conditions holds:
  - There exist a variable $\underline{n}$, and indices $i_1, \ldots, i_p$ ($p \geq 0$), and substitutions $u_1, \ldots, u_k$ ($k \geq 0$), such that
    * $1 \leq i_1, i_2, \ldots, i_p \leq q$ and all the i_j's are distinct
    * For all $s_1, \ldots, s_q$, we have $\underline{m}[\xi(s_1, \ldots, s_q)] =_f \underline{n}[s_{i_1}] \ldots [s_{i_p}][u_1] \ldots [u_k]$.
  - There exists an index i ($1 \leq i \leq q$), such that for all $s_1, \ldots, s_q$, we have

$$\underline{m}[\xi(s_1, \ldots, s_q)] =_f s_i$$

  So let us analyze each constructor in detail:

  - For $\uparrow$: for every variable $\underline{m}$ we have shown that $f(\underline{m}[\uparrow]) = \underline{m+1}$ so the first case of the definition of general scheme applies for $\underline{n} = \underline{m+1}$, $p = 0$ and $k = 0$.
  - For $\Uparrow$:
    * If the variable is $\underline{1}$, then $f(\underline{1}[\Uparrow(s)]) = \underline{1}$ by the $FVarLift$ rule, so the first case of the definition of general scheme applies for $\underline{n} = \underline{1}$, $p = 0$ and $k = 0$.
    * If the variable is $\underline{m+1}$, we have $f(\underline{m+1}[\Uparrow(s)]) = f(\underline{m}[s][\uparrow])$ as shown before, so the first case of definition of general scheme applies for $\underline{n} = \underline{m}$, $p = 1$, $i_1 = 1$, $k = 1$ and $u_1 = \uparrow$.
  - For $\#$:
    * If the variable is $\underline{1}$, then for every term a we have $f(\underline{1}[\#(a)]) = f(a)$ by the $FVar$ rule, so the second case of the definition of general scheme applies for $i = 1$.
    * If the variable is $\underline{m+1}$, then for every term a we have $f(\underline{m+1}[\#(a)]) = \underline{m}$ as shown before, so the first case of the definition of general scheme applies for $\underline{n} = \underline{m}$, $p = 0$ and $k = 0$.

Definition 8.1 A *position* in an object $o \in \mathcal{O}_f$ is a (possibly empty) word over $\{1, 2\}$ satisfying:

- $o_{|_\epsilon} = o$ (ϵ is the empty word);
- if $o_{|_p} = (ab)$ then $o_{|_{p\cdot 1}} = a$ and $o_{|_{p\cdot 2}} = b$;
- if $o_{|_p} = \lambda a$ then $o_{|_{p\cdot 1}} = a$;
- if $o_{|_p} = a[s]$ then $o_{|_{p\cdot 1}} = a$ and $o_{|_{p\cdot 2}} = s$;
- if $o_{|_p} = \#(a)$ then $o_{|_{p\cdot 1}} = a$;
- if $o_{|_p} = \Uparrow(s)$ then $o_{|_{p\cdot 1}} = s$.

The set of all positions of an object o is denoted by $Pos(o)$.

Within the positions in a term, we want to distinguish between *external* and *internal* positions. Intuitively, internal positions are the positions that occur inside [ ] symbols; external positions are the positions occurring outside the outermost [ ] symbol.

Definition 8.2 Given a term $t \in \mathcal{T}_f$, the set of its external positions is denoted by $Ext(t)$ and is inductively defined as follows:

- $Ext(\underline{1}) = \epsilon$;
- $Ext(a\ b) = \{1 \cdot p |\ p \in Ext(a)\} \cup \{2 \cdot p |\ p \in Ext(b)\} \cup \{\epsilon\}$;
- $Ext(\lambda a) = \{1 \cdot p |\ p \in Ext(a)\} \cup \{\epsilon\}$;
- $Ext(a[s]) = \{1 \cdot p |\ p \in Ext(a)\} \cup \{\epsilon\}$.

Definition 8.3 Given a term $t \in \mathcal{T}_f$, the set of its internal positions is denoted by $Int(t)$ and is inductively defined as follows:

- $Int(\underline{1}) = \emptyset$;
- $Int(a\ b) = \{1 \cdot p |\ p \in Int(a)\} \cup \{2 \cdot p |\ p \in Int(b)\}$;
- $Int(\lambda a) = \{1 \cdot p |\ p \in Int(a)\}$;
- $Int(a[s]) = \{1 \cdot p |\ p \in Int(a)\} \cup \{2 \cdot p |\ p \in Pos(s)\}$.

It is not difficult to see that a position in a term is either internal or external, i. e.:

Fact 8.8 For any term $t \in \mathcal{T}_f$, $Pos(t) = Ext(t) \cup Int(t)$.

The notion of external/internal can be extended to reduction steps.

Definition 8.4 Let $a_1, a_2 \in \mathcal{T}_f$ such that $a_1 \longrightarrow a_2$. We say that the reduction is external (respectively internal) and denote it by $a_1 \stackrel{ext}{\longrightarrow} a_2$ (respectively $a_1 \stackrel{int}{\longrightarrow} a_2$) if it occurs at an external (respectively internal) position in a_1. We just write $a \underset{p}{\longrightarrow} b$ to denote a reduction step which occurs at position p of a.

The following lemmas are not difficult to check.

Proposition 8.9 Let a and b be terms in $\mathcal{T}_f$ such that $a \stackrel{ext}{\longrightarrow}_{Beta} b$. Then $f(a) \longrightarrow^{+}_{\beta} f(b)$.

Proof. By induction on the structure of a.

Lemma 8.10 The reduction relation $\stackrel{int}{\longrightarrow}_{\lambda_f}$ has the following properties:

1. For any $a, b \in \mathcal{T}_f$, if $a \stackrel{int}{\longrightarrow}_{\lambda_f} \lambda b$, then $a = \lambda a_1$, for some $a_1 \in \mathcal{T}_f$ such that $a_1 \stackrel{int}{\longrightarrow}_{\lambda_f} b$.
2. For any $a, c, d \in \mathcal{T}_f$, if $a \stackrel{int}{\longrightarrow}_{\lambda_f} (c\ d)$, then $a = (e\ f)$, for some $e, f \in \mathcal{T}_f$ such that either
   - $e \stackrel{int}{\longrightarrow}_{\lambda_f} c$, and $f = d$; or
   - $e = c$, and $f \stackrel{int}{\longrightarrow}_{\lambda_f} d$.
3. For any $a \in \mathcal{T}_f$, $a \stackrel{int}{\longrightarrow}_{\lambda_f} \underline{1}$ is impossible.
4. For any $a, b \in \mathcal{T}_f$ and for any $i \geq 0$, if $a \stackrel{int}{\longrightarrow}_{\lambda_f} b[\Uparrow^i (\uparrow)]$ then $a = c[\Uparrow^i (\uparrow)]$, for some $c \in \mathcal{T}_f$, such that $c \stackrel{int}{\longrightarrow}_{\lambda_f} b$.

Lemma 8.11 (Commutation Lemma) Let a and b be two terms in $\mathcal{T}_f$ such that $f(a)$ is β-strongly normalizing and $f(a) = f(b)$. If $a \xrightarrow{int}_{\lambda_f} . \xrightarrow{ext}_f b$, then $a \overset{ext}{\longrightarrow}{}^{+}_{f} . \xrightarrow{int}_{\lambda_f} b$.

Proof. We proceed by induction on the structure of a.

- if $a = \underline{1}$ then the property holds vacuously.
- if $a = (a_1\ a_2)$ then we have to consider several possibilities:
  - •• $(a_1\ a_2) \xrightarrow{int}_{\lambda_f} (a_1'\ a_2) \xrightarrow{ext}_f (a_1''\ a_2) = b$, with $a_1 \xrightarrow{int}_{\lambda_f} a_1' \xrightarrow{ext}_f a_1''$; in order to apply the induction hypothesis to a_1 we need to see that $f(a_1)$ is β-strongly normalizing and that $f(a_1) = f(a_1'')$. Since f satisfies $f(c\ d) = (f(c)\ f(d))$, then we have that $f(a_1\ a_2) = (f(a_1)\ f(a_2))$ and so β-strong normalization of $f(a_1)$ comes from that of $f(a)$. For the second property, we have that

    $$(f(a_1)\, f(a_2)) = f(a_1\, a_2) = f(a_1''\, a_2) = (f(a_1'')\, f(a_2))$$

    and consequently $f(a_1) = f(a_1'')$. So we can apply the induction hypothesis to conclude that $a_1 \overset{ext}{\longrightarrow}{}^{+}_{f} . \xrightarrow{int}_{\lambda_f} a_1''$ and so $a \overset{ext}{\longrightarrow}{}^{+}_{f} . \xrightarrow{int}_{\lambda_f} b$.
  - •• $(a_1\ a_2) \xrightarrow{int}_{\lambda_f} (a_1\ a_2') \xrightarrow{ext}_f (a_1\ a_2'') = b$, with $a_2 \xrightarrow{int}_{\lambda_f} a_2' \xrightarrow{ext}_f a_2''$; this case is symmetrical to the previous one.
  - •• $(a_1\ a_2) \xrightarrow{int}_{\lambda_f} (a_1'\ a_2) \xrightarrow{ext}_f (a_1'\ a_2')$, with $a_1 \xrightarrow{int}_{\lambda_f} a_1'$ and $a_2 \xrightarrow{ext}_f a_2'$; in this case the internal and external positions are parallel so the order of the rewrites can be commuted.
  - •• $(a_1\ a_2) \xrightarrow{int}_{\lambda_f} (a_1\ a_2') \xrightarrow{ext}_f (a_1'\ a_2')$, with $a_2 \xrightarrow{int}_{\lambda_f} a_2'$ and $a_1 \xrightarrow{ext}_f a_1'$; this case is symmetrical to the previous one.
- if $a = \lambda a_1$ then the only possibility is

  $$a = \lambda a_1 \xrightarrow{int}_{\lambda_f} \lambda a_1' \xrightarrow{ext}_f \lambda a_1'' = b$$

  with $a_1 \xrightarrow{int}_{\lambda_f} a_1' \xrightarrow{ext}_f a_1''$; in order to apply the induction hypothesis to a_1, we need to see that $f(a_1)$ is β-strongly normalizing and that $f(a_1) = f(a_1'')$. Since f satisfies $f(\lambda d) = \lambda f(d)$, we also have $f(\lambda a_1) = \lambda f(a_1)$ and since $f(\lambda a_1)$ is β-strongly normalizing, $f(a_1)$ has to be β-strongly normalizing. By hypothesis $f(a) = f(b)$, so

  $$\lambda f(a_1) = f(\lambda a_1) = f(\lambda a_1'') = \lambda f(a_1'')$$

  and consequently $f(a_1) = f(a_1'')$. So we can apply the induction hypothesis to a_1 to conclude that $a_1 \overset{ext}{\longrightarrow}{}^{+}_{f} . \xrightarrow{int}_{\lambda_f} a_1''$ and therefore that $a \overset{ext}{\longrightarrow}{}^{+}_{f} . \xrightarrow{int}_{\lambda_f} b$.
- if $a = a_1[s]$ then we have several possibilities:
  - •• $a = a_1[s] \xrightarrow{int}_{\lambda_f} a_1[s'] \xrightarrow{ext}_f a_1'[s'] = b$, with $s \longrightarrow_{\lambda_f} s'$ and $a_1 \xrightarrow{ext}_f a_1'$; in this case the external and internal positions used in the rewrite steps are parallel and therefore the order of the rewrites can be commuted.
  - •• $a = a_1[s] \xrightarrow{int}_{\lambda_f} a_1'[s] \xrightarrow{ext}_f b$, with $a_1 \xrightarrow{int}_{\lambda_f} a_1'$; this case can be subdivided into two sub-cases, depending on how the external rewrite takes place, namely

- $a = a_1[s] \xrightarrow{int}_{\lambda_f} a_1'[s] \xrightarrow{ext}_f a_1''[s]$, with $a_1 \xrightarrow{int}_{\lambda_f} a_1' \xrightarrow{ext}_f a_1''$; in order to apply the induction hypothesis to a_1 we need to see that $f(a_1) = f(a_1'')$ and that $f(a_1)$ is β-strongly normalizing. First note that $f(a_1') = f(a_1'')$.
  Suppose that the internal rewrite does not use the rule $(Beta)$, i. e., we have $a_1 \xrightarrow{int}_f a_1' \xrightarrow{ext}_f a_1''$, then obviously $f(a_1) = f(a_1') = f(a_1'')$. Furthermore, $f(a) = f(a_1[s]) = f(f(a_1)[s])$, so if $f(a_1)$ is not β-strongly normalizing then $f(a)$ cannot be β-strongly normalizing either. So in this case we can apply the induction hypothesis to conclude that $a_1 \overset{ext}{\longrightarrow}{}^{+}_{f} \ . \xrightarrow{int}_{\lambda_f} a_1''$ and consequently we have $a \overset{ext}{\longrightarrow}{}^{+}_{f} \ . \xrightarrow{int}_{\lambda_f} b$.
  If the internal rewrite is a $(Beta)$-application, then by the Projection Lemma (cf. lemma 8.7), we have that $f(a_1) \longrightarrow^*_\beta f(a_1')$ and since $f(a_1') = f(a_1'')$, we can write $f(a_1) \longrightarrow^*_\beta f(a_1'')$. If $f(a_1) = f(a_1'')$, by an argument similar to the previous one, we are done. If it is the case that $f(a_1) \longrightarrow^+_\beta f(a_1'')$ then one can show by induction on the structure of a_1 that $f(f(a_1)[s]) \longrightarrow^+_\beta f(f(a_1'')[s])$. Therefore

  $$f(a) = f(a_1[s]) = f(f(a_1)[s]) \longrightarrow^+_\beta f(f(a_1'')[s]) = f(a_1''[s]) = f(b) = f(a)$$

  contradicting the fact that $f(a)$ is β-strongly normalizing.
- $a = a_1[s] \xrightarrow{int}_{\lambda_f} a_1'[s] \xrightarrow{ext}_{\epsilon,f} b$, with $a_1 \xrightarrow{int}_{\lambda_f} a_1'$ and the external reduction being a root reduction; we now proceed by case analysis on the rule used in the external root rewrite.
  - (App) Then $a_1' = (c\ d)$ and $b = (c[s]\ d[s])$. Since $a_1 \xrightarrow{int}_{\lambda_f} (c\ d)$, we must have $a_1 = (\overline{c}\ \overline{d})$, by lemma 8.10, and either $\overline{c} \xrightarrow{int}_{\lambda_f} c$ and $\overline{d} = d$ or vice-versa (i. e., $\overline{c} = c$ and $\overline{d} \xrightarrow{int}_{\lambda_f} d$). We assume without loss of generality that we are in the first case (the other one is symmetrical). Then

    $$a = (\overline{c}\ d)[s] \xrightarrow{ext}_f (\overline{c}[s]\ d[s]) \xrightarrow{int}_{\lambda_f} (c[s]\ d[s]) = b$$

    so the result holds.
  - $(Lambda)$ Then $a_1' = \lambda c$ and $b = \lambda(c[\Uparrow(s)])$. Since $a_1 \xrightarrow{int}_{\lambda_f} \lambda c$, by lemma 8.10, we must have $a_1 = \lambda\overline{c}$ with $\overline{c} \xrightarrow{int}_{\lambda_f} c$. But then

    $$a = (\lambda\overline{c})[s] \xrightarrow{ext}_f \lambda(\overline{c}[\Uparrow(s)]) \xrightarrow{int}_{\lambda_f} \lambda(c[\Uparrow(s)]) = b$$

    and the result holds.
  - $(FVar)$ In this case we have $a_1' = \underline{1}$ and so $a_1 \xrightarrow{int}_{\lambda_f} \underline{1}$. Due to lemma 8.10, this case cannot occur.
  - $(FVarLift)$ In this case we also have $a_1' = \underline{1}$, so like the previous one, this case cannot occur.
  - $(Shift)$ Then $a_1' = c[\Uparrow^i(\uparrow)]$, and $s = \Uparrow^i(\#(d))$, for some terms $c, d \in \mathcal{T}_f$, and some index $i \geq 0$, and $b = c$. Since $a_1 \xrightarrow{int}_{\lambda_f} a_1'$, by lemma 8.10, we have that $a_1 = \overline{c}[\Uparrow^i(\uparrow)]$, for some $\overline{c} \in \mathcal{T}_f$, such that $\overline{c} \xrightarrow{int}_{\lambda_f} c$. Consequently we have

    $$a = \overline{c}[\Uparrow^i(\uparrow)][\Uparrow^i(\#(d))] \xrightarrow{ext}_f \overline{c} \xrightarrow{int}_{\lambda_f} c = b$$

    and the result holds.
  - $(ShiftLift)$ This case is similar to the previous one.

The following lemma is taken from [BBLRD95].

Lemma 8.12 (Iterative Commutation Lemma) Let $a_0, \cdots, a_n$ be $n+1$ terms in $\mathcal{T}_f$, such that:

- $f(a_0)$ is β-strongly normalizing;
- $f(a_i) = f(a_j)$, for all $0 \leq i, j \leq n$;
- $a_i \overset{int}{\longrightarrow}{}^*_{\lambda_f} \cdot \overset{ext}{\longrightarrow}_f \ a_{i+1}$, for all $0 \leq i \leq n-1$.

Then $a_0 \overset{n,ext}{\longrightarrow}_f \ \cdot (\overset{int}{\longrightarrow}_{\lambda_f} \cup \overset{ext}{\longrightarrow}_f)^* a_n$.

Proof. As in [BBLRD95], with λ_υ and υ replaced by respectively λ_f and f and using the following result on abstract reductions (whose proof can also be found in [BBLRD95]).

Let R, S be two abstract reduction systems such that $R \circ S \subseteq S^+ \circ R^*$. Then

1. $(R \cup S)^* \circ S \subseteq S \circ (R \cup S)^*$;
2. $\forall n \geq 0 : \ (R \cup S)^* \circ S^n \subseteq S^n \circ (R \cup S)^*$.

Lemma 8.13 Let a be a β-strongly normalizing pure term. For every infinite λ_f-derivation $a \longrightarrow_{\lambda_f} b_1 \longrightarrow_{\lambda_f} b_2 \longrightarrow_{\lambda_f} \ldots$ there exists N such that for $i \geq N$ all the reduction steps $b_i \longrightarrow_{\lambda_f} b_{i+1}$ are internal.

Proof. Suppose that a is a β-strongly normalizing pure term and consider an arbitrary infinite λ_f-derivation of a. Since a is pure and due to the form of the rules in the λ_f-calculus, the first rewrite has to be an application of the rule $(Beta)$ at an external position (pure terms have no internal positions). Since the f-calculus is strongly normalizing, any infinite derivation of a in the λ_f-calculus has to contain infinitely many applications of the rule $(Beta)$, i. e., we can write

$$a \overset{ext}{\longrightarrow}_{Beta} \ a_1 \longrightarrow^*_f a_{i_1} \longrightarrow_{Beta} \ a_{i_1+1} \longrightarrow^*_f a_{i_2} \longrightarrow_{Beta} \ a_{i_2+1} \cdots \tag{2}$$

Consequently (note that $a = f(a)$, since a is pure), using the Projection lemma (lemma 8.7) and proposition 8.9, we have:

$$f(a) = a \longrightarrow^*_\beta f(a_1) = f(a_{i_1}) \longrightarrow^*_\beta f(a_{i_1+1}) = f(a_{i_2}) \longrightarrow^*_\beta f(a_{i_3}) \cdots \tag{3}$$

Furthermore by proposition 8.9, only finitely many $(Beta)$-rewrites in (2) can be external, for otherwise we could replace the respective $\longrightarrow^*_\beta$ steps in (3) by $\longrightarrow^+_\beta$ steps, thus creating an infinite β-reduction for a. So after some index $I \geq 1$, all $(Beta)$ applications in (2) are internal. Furthermore, since $a = f(a)$ is β-strongly normalizing and (3) is infinite, we can conclude that there is another index $I' \geq 1$ such that for $i, j \geq I'$, $f(a_i) = f(a_j)$. So, for $P \geq \ max\{I, I'\}$, we can write:

$$a_P \overset{int}{\longrightarrow}{}^*_{\lambda_f} \ a_{P_1} \overset{ext}{\longrightarrow}_f \ a'_{P_1} \overset{int}{\longrightarrow}{}^*_{\lambda_f} \ a_{P_2} \overset{ext}{\longrightarrow}_f \ a'_{P_2} \cdots \tag{4}$$

Note that this infinite sub-sequence has to contain infinitely many $\overset{int}{\longrightarrow}_{Beta}$-steps, for otherwise we would get an infinite sequence in the f-calculus. Also in this sub-sequence we can find an index $P' \geq P$ after which all rewrite steps are internal; we see why this is so.

Since the f-calculus is strongly normalizing, for any infinite λ_f-sequence starting at a_P, there is a natural number K_P such that the maximal number of f-rewrites starting at a_P is less or equal to K_P. If in the infinite sequence we can find infinitely many $\stackrel{ext}{\longrightarrow}_f$ -steps, there are at least K_P+1 of them , i. .e, we can write:

$$a_P \stackrel{int}{\longrightarrow}^*_{\lambda_f} \cdot \stackrel{ext}{\longrightarrow}_f a_{R_1} \stackrel{int}{\longrightarrow}^*_{\lambda_f} \cdot \stackrel{ext}{\longrightarrow}_f a_{R_2} \cdots \stackrel{int}{\longrightarrow}^*_{\lambda_f} \cdot \stackrel{ext}{\longrightarrow}_f a_{R_{K_P+1}} \cdots \tag{5}$$

and using the Iterative Commutation Lemma (lemma 8.12) (note that $f(a_i) = f(a_j)$, for any $i, j \geq P$ and $f(a_P)$ is β-strongly normalizing), we get

$$a_P \stackrel{K_P+1, ext}{\longrightarrow}_f \cdot (\stackrel{ext}{\longrightarrow}_f \cup \stackrel{int}{\longrightarrow}_{\lambda_f})^* a_{R_{K_P+1}} \cdots \tag{6}$$

contradicting the definition of K_P.

From now on, $a \underset{p}{\longrightarrow} b$ denotes

Definition 8.5 (Derivation Ordering) Let a_1 be a term and $\mathcal{D}$ and $\mathcal{D}'$ be two derivations starting at a_1.

$$\mathcal{D} = a_1 \underset{p_1}{\longrightarrow} a_2 \underset{p_2}{\longrightarrow} \ldots a_n \underset{p_n}{\longrightarrow} a_{n+1} \ldots$$

$$\mathcal{D}' = a_1 \underset{q_1}{\longrightarrow} a_2 \underset{q_2}{\longrightarrow} \ldots a_n \underset{q_n}{\longrightarrow} b_{n+1} \ldots$$

Then $\mathcal{D}$ is said to be *smaller* than $\mathcal{D}'$ if $p_i = q_i$ for every $i < n$ and q_n is a strict prefix of p_n.

Definition 8.6 (Minimal Derivation) An infinite derivation $\mathcal{D}$ starting at a_1 is *minimal* if there is no infinite derivation starting at a_1 which is smaller than $\mathcal{D}$.

Thus, a minimal derivation has the property that if one rewrites at least one of its steps at a lower position, then an infinite tail is impossible.

Proposition 8.14 For every object p in $\mathcal{O}_f$

1. If p is a term, then $f(p)$ is a pure term.
2. For every pure term d such that $d[p]$ is a term, we have that for every $i \geq 0$, $f(d[\Uparrow^i (p)])$ is a pure term.

Proof. By induction on the structure of the object p.

- If p is a variable $\underline{1}$, the first property trivially holds, while the second one holds vacuously.
- If $p = \uparrow$, then the first property vacuously holds; for the second one we proceed by induction on the structure of pure terms.
  - •• If $d = \underline{1}$, then $f(\underline{1}[\Uparrow^i (\uparrow)]) = \underline{1}$, if $i > 0$, and $f(\underline{1}[\Uparrow^i (\uparrow)]) = \underline{1}[\uparrow]$, if $i = 0$, both of which are pure terms.
  - •• If $d = \underline{1}[\uparrow]^j$ $(j > 0)$, we consider two cases:
    - ••• If $i = 0$, then $f(\underline{1}[\uparrow]^j[\uparrow]) = \underline{1}[\uparrow]^{j+1}$, which is a pure term.
    - ••• If $i > 0$, then we distinguish again two cases:
      - ••••• If $j < i$, then

$$f(\underline{1}[\uparrow]^j[\Uparrow^i (\uparrow)]) =_{ShiftLift} f(\underline{1}[\Uparrow^{i-j} (\uparrow)][\uparrow]^j) =_{FVarLift} f(\underline{1}[\uparrow]^j) = \underline{1}[\uparrow]^j$$

- - - - If $j \geq i$, then

$$f(\underline{1}[\uparrow]^j[\Uparrow^i(\uparrow)]) =_{ShiftLift} f(\underline{1}[\uparrow]^{j+1}) = \underline{1}[\uparrow]^{j+1}$$

  - If $d = (d_1\ d_2)$, then $f((d_1\ d_2)[\Uparrow^i(\uparrow)]) = f(d_1[\Uparrow^i(\uparrow)])\ f(d_2[\Uparrow^i(\uparrow)])$, so the property holds by induction hypothesis applied to d_1 and d_2.
  - If $d = \lambda d_1$, then $f((\lambda d_1)[\Uparrow^i(\uparrow)]) = \lambda f(d_1[\Uparrow^{i+1}(\uparrow)])$, so the property holds by induction hypothesis applied to d_1.

- If $p = (a\ b)$, then $f(a)$ and $f(b)$ are pure terms by induction hypothesis (1), so the first property holds because $f(a\ b) = (f(a)\ f(b))$; the second one holds vacuously.

- If $p = \lambda a$, then $f(a)$ is a pure term by induction hypothesis (1), so the first property holds because $f(\lambda a) = \lambda f(a)$, while the second one holds vacuously.

- If $p = a[s]$, then $f(a)$ is a pure term by induction hypothesis (1), and the second property holds for s by induction hypothesis (2) so the first property holds for p since $f(a[s]) = f(f(a)[s])$ turns out to be a pure term. The second property holds vacuously.

- If $p = \Uparrow(s)$. The first property holds vacuously; for the second one we proceed as follows. By induction hypothesis (2) we know that for every pure term d and every $i \geq 0$, the term $f(d[\Uparrow^i(s)])$ is pure; in particular $f(d[\Uparrow^j(\Uparrow(s))])$, $j \geq 0$, is pure, giving our result.

- If $p = \#(a)$. The first property vacuously holds; for the second one we proceed by induction on the structure of pure terms.

  - If $d = \underline{1}$, then $f(\underline{1}[\Uparrow^i(\#(a))]) = f(a)$, if $i = 0$, and $f(\underline{1}[\Uparrow^i(\#(a))]) = \underline{1}$, if $i > 0$. In the first case the property holds by induction hypothesis (1) applied to a, while the second case is trivial.
  - If $d = \underline{1}[\uparrow]^j$ $(j > 0)$, then we consider two cases:
    - If $i = 0$, then $f(\underline{1}[\uparrow]^j[\#(a)]) =_{Shift} f(\underline{1}[\uparrow]^{j-1}) = \underline{1}[\uparrow]^{j-1}$.
    - If $i > 0$, then we distinguish again two cases:
      - If $j < i$, then

$$\begin{array}{ll} f(\underline{1}[\uparrow]^j[\Uparrow^i(\#(a))]) & =_{ShiftLift} \\ f(\underline{1}[\Uparrow^{i-j}(\#(a))][\uparrow]^j) & =_{FVarLift} \\ f(\underline{1}[\uparrow]^j) & = \\ \underline{1}[\uparrow]^j & \end{array}$$

      - If $j \geq i$, then

$$f(\underline{1}[\uparrow]^j[\Uparrow^i(\#(a))]) =_{ShiftLift} f(\underline{1}[\uparrow]^{j-i}[\#(a)][\uparrow]^i)$$

        Now, if $j - i > 0$, then $f(\underline{1}[\uparrow]^{j-i}[\#(a)][\uparrow]^i) =_{ShiftLift} f(\underline{1}[\uparrow]^{j-i-1}[\uparrow]^i) = \underline{1}[\uparrow]^{j-1}$. Otherwise, $j - i = 0$, and $f(\underline{1}[\#(a)][\uparrow]^i) =_{FVar} f(a[\uparrow]^i)$. By induction hypothesis (1) applied to a we know that $f(a)$ is a pure term, and by i applications of the induction hypothesis(2) applied to $\uparrow$ we can conclude that $f(a[\uparrow]^i) = f(f(f(a)[\uparrow])\ldots[\uparrow])$ is a pure term.
  - If $d = (d_1\ d_2)$, then $f((d_1\ d_2)[\Uparrow^i(\#(a))]) = (f(d_1[\Uparrow^i(\#(a))])\ f(d_1[\Uparrow^i(\#(a))]))$, so the property holds by induction hypothesis applied to d_1 and d_2.
  - If $d = \lambda d_1$, then $f((\lambda d_1)[\Uparrow^i(\#(a))]) = \lambda f(d_1[\Uparrow^{i+1}(\#(a))])$, so the property holds by induction hypothesis applied to d_1.

We can conclude that

Corollary 8.15 f-normal forms of terms are pure terms.

Definition 8.7 (Skeleton) The *skeleton* of a term in $\mathcal{T}_f$ is defined by induction as follows:

$$\begin{array}{lcl} Sk(\underline{n}) & = & \underline{n} \\ Sk(a\, b) & = & Sk(a)\, Sk(b) \\ Sk(\lambda a) & = & \lambda Sk(a) \\ Sk(a[s]) & = & Sk(a)[.] \end{array}$$

Intuitively, the skeleton of a term is the the part where external reductions are possible.

The occurrences of [.] inside a skeleton are called *closures*. Remark that $a \xrightarrow{int}_{\lambda_f} b$ implies $Sk(a) = Sk(b)$.

Theorem 8.16 If a is a β-strongly normalizing pure term, then it is λ_f-strongly normalizing.

Proof. Suppose that a is a β-strongly normalizing pure term, which is not λ_f-strongly normalizing. Let us consider a minimal infinite λ_f-derivation

$$\mathcal{D}: \; a_0 = a \longrightarrow_{\lambda_f} a_1 \longrightarrow_{\lambda_f} a_2 \ldots$$

By lemma 8.13 there exists N such that for every $i \geq N$, the reduction step $a_i \longrightarrow_{\lambda_f} a_{i+1}$ is internal. As a consequence, by the previous remark, $Sk(a_i) = Sk(a_{i+1})$ for every $i \geq N$. As there are only a finite number of closures in $Sk(a_N)$ and the reductions within these closures are independent, an infinite sub-derivation of $\mathcal{D}$ must take place within the same closure in $Sk(a_N)$ (which is necessarily of the form $\Uparrow^i(b)$), and evidently, this sub-derivation is also minimal. Let us call it $\mathcal{D}'$ and let C be the context such that $a_N = C\langle\!|d[\Uparrow^i(b)]|\!\rangle$ and $\Uparrow^i(b)$ is the term appearing inside the closure we have mentioned before. Therefore we have

$$\mathcal{D}': \; a_N = C\langle\!|d[\Uparrow^i(b)]|\!\rangle \xrightarrow{int}_{\lambda_f} C\langle\!|d[\Uparrow^i(b_1)]|\!\rangle \xrightarrow{int}_{\lambda_f} C\langle\!|d[\Uparrow^i(b_2)]|\!\rangle \xrightarrow{int}_{\lambda_f} \ldots$$

Since a is a pure term, lemma 8.2 ensures the existence of $0 \leq I \leq N$ such that

$$a_I = C'\langle\!|(\lambda d')b'|\!\rangle \longrightarrow_{\lambda_f} C'\langle\!|d'[\#(b')]|\!\rangle \text{ and } b' \longrightarrow^*_{\lambda_f} b$$

Let us consider the following infinite derivation $\mathcal{D}''$:

$$a \longrightarrow^*_{\lambda_f} a_I = C'\langle\!|(\lambda d')b'|\!\rangle \longrightarrow^*_{\lambda_f} C'\langle\!|(\lambda d')b|\!\rangle \longrightarrow_{\lambda_f} C'\langle\!|(\lambda d')b_1|\!\rangle \longrightarrow_{\lambda_f} C'\langle\!|(\lambda d')b_2|\!\rangle \longrightarrow_{\lambda_f} \ldots$$

In this infinite derivation the redex in a_I is inside b' which is a proper subterm of $(\lambda d')b'$, whereas in the derivation $\mathcal{D}$ the redex in a_I is the term $(\lambda d')b'$ itself and hence it appears in an upper position. This means that $\mathcal{D}''$ is smaller than $\mathcal{D}$, contradicting the minimality of the infinite derivation $\mathcal{D}$.

We have shown preservation of β-strong normalization of λ_f using a minimality technique from [BBLRD95]. More recently a technique has been proposed in [BG] using semantic labeling [Zan95] and RPO so that preservation of β-strong normalization of λ_f can also be shown using the technique

9 Main Results

We now present the result we were aiming at, namely preservation of β-strong normalization by λ_d, λ_υ and λ_{dn}.

Theorem 9.1 If a is a β-strongly normalizing pure term, then it is λ_d-strongly normalizing

Proof. Let a be a β-strongly normalizing pure term. From theorem 8.16, we know that a is also λ_f-strongly normalizing and since λ_e preserves λ_f-strong normalization (by corollary 7.3), we conclude that a is λ_e-strongly normalizing. Finally since λ_d preserves λ_e-strong normalization (by corollary 6.4), we conclude that a is λ_d-strongly normalizing.

Theorem 9.2 If a is a β-strongly normalizing pure term, then it is λ_{dn}-strongly normalizing.

Proof. Let a be a β-strongly normalizing pure term. From theorem 9.1, we know that a is also λ_e-strongly normalizing and since λ_{dn} preserves λ_e-strong normalization (by corollary 6.8), we conclude that a is λ_{dn}-strongly normalizing.

An alternative proof of the result in [BBLRD95] is the following:

Theorem 9.3 If a is a β-strongly normalizing pure term, then it is λ_υ-strongly normalizing.

Proof. Let a be a β-strongly normalizing pure term. From theorem 8.16, we know that a is also λ_f-strongly normalizing and since λ_υ preserves λ_f-strong normalization (by corollary 7.6), we conclude that a is λ_υ-strongly normalizing.

10 Conclusion

We have shown preservation of β-strong normalization of λ_d and λ_{dn}, two different λ-calculi with explicit substitutions which have composition of substitutions and are confluent on closed terms. For the proof, an abstract simulation technique was developed to reduce β-strong normalization of these calculi to that of another one. We applied this technique to show that, not only preservation of β-strong normalization of λ_d and λ_{dn}, but also that of λ_υ, reduces to preservation of β-strong normalization of λ_f, which is a very simple calculus containing no composition and no identity substitution. Thus, λ_f becomes the minimal calculus of explicit substitutions that one can expect to preserve β-strong normalization and to be confluent on closed terms, while λ_d and λ_{dn} become the first λ-calculi with explicit substitutions having weak composition, being confluent on closed terms and preserving β-strong normalization.

Next challenge concerns the definition of a calculus in the same spirit of λ_d (having full composition and preserving β-strong normalization) but being confluent on *open* terms. We are also interested in a *general scheme* for substitution calculi as the one defined in [Kes96] to study preservation of β-strong normalization from a more abstract point of view. Finally, it would be interesting to investigate which are the abstract functional machines that one can encode using λ_d or λ_{dn}.

Acknowledgments

This work was initiated when Maria Ferreira visited the LRI at the Université Paris-Sud in January 1996. She was then supported by NWO, the Dutch Organization for Scientific Research (under grant 612-316-041), the University of Utrecht, and the Human Mobility Program (under grant #ERBCHRXCT940495). We would like to thank JNICT and CNRS for partially supporting this work.

References

[ACCaL91] Martin Abadi, Luca Cardelli, Pierre Louis Curien, and Jean-Jacques Lévy. Explicit substitutions. *Journal of Functional Programming*, 4(1):375–416, 1991.

[BBLRD95] Zine-El-Abidine Benaissa, Daniel Briaud, Pierre Lescanne, and Jocelyne Rouyer-Degli. λv, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 6(5):699–722, 1996.

[BG] Roel Bloo and Herman Geuvers. Explicit Substitution: on the Edge of Strong Normalization. Theoretical Computer Science, to appear.

[BR95] Roel Bloo and Kristoffer Rose. Preservation of strong normalization in named lambda calculi with explicit substitution and garbage collection. In *Computer Science in the Netherlands (CSN)*, 1995.

[CHL92] Pierre-Louis Curien, Thérèse Hardin, and Jean-Jacques Lévy. Confluence properties of weak and strong calculi of explicit substitutions. Technical Report 1617, INRIA Rocquencourt, 1992.

[CHR92] Pierre-Louis Curien, Thérèse Hardin, and Alejandro Rios. Strong normalisation of substitutions. In *MFCS'92*, LNCS 629, pages 209–218, 1992.

[Cur86] Pierre-Louis Curien. *Categorical combinators, sequential algorithms and functional programming.* Progress in Theoretical Computer Science. Birkhaüser, 1986. First edition.

[dB72] N. de Bruijn. Lambda-calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indag. Mat.*, 5(35):381–392, 1972.

[dB78] N. de Bruijn. Lambda-calculus notation with namefree formulas involving symbols that represent reference transforming mappings. *Indag. Mat.*, (40):384–356, 1978.

[DHK95] Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Higher-order unification via explicit substitutions. In *Proceedings of the Symposium on Logic in Computer Science (LICS)*, 1995.

[Ehr88] Thomas Ehrhard. *Une sémantique catégorique des types dépendants. Application au calcul des constructions.* Thèse de doctorat, Université de Paris VII, 1988.

[HL89] Thérèse Hardin and Jean-Jacques Lévy. A confluent calculus of substitutions. In *France-Japan Artificial Intelligence and Computer Science Symposium*, 1989.

[HMP95] Thérèse Hardin, Luc Maranget, and Bruno Pagano. Functional back-ends and compilers within the lambda-sigma calculus, 1995. Draft.

[Hue80] Gérard Huet. Confluent reductions: abstract properties and applications to term rewriting systems. *Journal of the ACM*, 4(27):797–821, 1980.

[Kes96] Delia Kesner. Confluence properties of extensional and non-extensional λ-calculi with explicit substitutions. In *Proc. of the Seventh International Conference on Rewriting Techniques and Applications (RTA)*, LNCS 1103, pages 184–199, 1996.

[KR95a] Fairouz Kamareddine and Alejandro Ríos. Extending a Lambda-calculus with explicit substitutions which preserves strong normalisation into a confluent calculus on open terms. *Journal of Functional Programming*, To appear.

[KR95b] Fairouz Kamareddine and Alejandro Ríos. A λ-calculus à la de Bruijn with explicit substitutions. In *Proceedings of the International Symposium on Programming Language Implementation and Logic Programming*, LNCS 982, 1995.

[Lan75] D. S. Lankford. Canonical algebraic simplification in computational logic. Technical Report Memo ATP-25, University of Texas. Austin. 1975.

[Les94] Pierre Lescanne. From λ_σ to λ_υ, a journey through calculi of explicit substitutions. In *Ann. ACM Symp. on Principles of Programming Languages (POPL)*, pages 60–69. ACM, 1994.

[Les96] Pierre Lescanne. Personal Communication. 1996.

[Lins86] Raphael Lins. A New Formula for the Execution of Categorical Combinators. In *Proceedings of the 8th International Conference on Automated Deduction*, LNCS 230, pages 89–98, 1986.

[Lins92] Raphael Lins. Partial Categorical Multi-Combinators and Church Rosser Theorems. Technical Report 7/92 of the Computing Laboratory, University of Kent at Canterbury, 1992.

[Mar93] Claude Marché. *Réécriture modulo une théorie présentée par un système convergent et décidabilité des problèmes du mot dans certains classes de théories équationnelles.* PhD thesis, Université Paris-Sud, Orsay, 1993.

[Mel94] Paul-André Mellies. Four typed-lambda calculi with explicit substitutions may not terminate: the first examples, 1994. Draft.

[Mel95] Paul-André Mellies. Typed λ-calculi with explicit substitutions may not terminate. In Mariangiola Dezani-Ciancaglini and Gordon Plotkin, editors, *Proceedings of Int. Conf. on Typed Lambda Calculi and Applications (TLCA)*, LNCS 902, 1995.

[MN70] Z. Manna and S. Ness. On the Termination of Markov Algorithms. Proceedings of the Third Hawaii International Conference on System Science. 1970.

[Muñ96] César Muñoz. Confluence and preservation of strong normalisation in an explicit substitutions calculus. In *Proceedings of the Symposium on Logic in Computer Science (LICS)*, 1996.

[Río93] Alejandro Ríos. *Contribution à l'étude des λ-calculus avec substitutions explicites.* Thèse de doctorat, Université de Paris VII, 1993.

[Rose92] Kristoffer Rose. Explicit cyclic substitutions. In *CTRS* 1992, LNCS 656, 1993.

[Zan94] Hans Zantema. Termination of term rewriting: interpretation and type elimination. *Journal of Symbolic Computation*, 17:23–50, 1994.

[Zan95] Hans Zantema. Termination of Term Rewriting by Semantic Labelling. *Fundamenta Informaticae*, 24:89–105, 1995.