# An explicit *Eta* rewrite rule

Daniel Briaud

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

# *An explicit Eta rewrite rule*

Daniel BRIAUD

**N° 2417**

Novembre 1994

PROGRAMME 2


ISSN 0249-6399

INRIA
LORRAINE

# An explicit *Eta* rewrite rule

Daniel BRIAUD

Programme 2 — Calcul symbolique, programmation et génie logiciel
Projet Eureca

Rapport de recherche n° 2417 — Novembre 1994 — 20 pages

**Abstract:** In this report, we extend $\lambda$-calculi of explicit substitutions by an *Eta* rule. We do this in the framework of $\lambda v$, a $\lambda$-calculus of explicit substitutions introduced by Lescanne (1994) and thoroughly studied by Lescanne and Rouyer-Degli (1994). The main feature of such a calculus is that the classical $\beta$-contraction is expressed by a first-order term rewriting system. Our main result is the explicitation of the $\eta$-contraction by means of an unconditional, *generic Eta* rewrite rule and of an extension of the substitution calculus, $v$. Previous definitions of *Eta* are due to Hardin (1992) and Ríos (1993) in the framework of $\lambda\sigma$-calculi. The principal difference between $\lambda v$ and $\lambda\sigma$-calculi concerns confluence and strong normalization: $\lambda v$ is ground confluent and its simply typed version is terminating, whenever $\lambda\sigma_{\Uparrow}$-calculus is confluent on open terms but non terminating on typed terms. In their work, Hardin and Ríos present *Eta* rule as an extension of $\eta$-contraction. Their extension is a conditional rewrite rule and does not stick fully to the philosophy of explicit substitutions. In our approach, the *Eta*-rule is a first order rewrite rule which uses an explicit substitution calculus. For that, one needs to introduce a new constant $\perp$ that denotes an unspecified term. Its behaviour is described by a rewrite rule. This report shows, in the one hand, how the *Eta*-rule associated with $\lambda v$ and the rule for $\perp$ provides a correct implementation of the $\eta$-reduction and studies other properties of the $\lambda v\eta$-calculus namely confluence on ground terms and strong normalisation on typed terms. On the other hand, this explicit *Eta* leads to $\eta'$ a very general alternative to the classical $\eta$. Indeed, we prove that $\eta'$ allows confluent contractions which are not captured by classical $\eta$.

**Key-words:** $\lambda$-calculus, explicit substitutions, $\eta$-reduction

*(Résumé : tsvp)*

Unité de recherche INRIA Lorraine
Technôpole de Nancy-Brabois, Campus scientifique,
615 rue de Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY (France)
Téléphone : (33) 83 59 30 30 – Télécopie : (33) 83 27 83 19
Antenne de Metz, technopôle de Metz 2000, 4 rue Marconi, 55070 METZ
Téléphone : (33) 87 20 35 00 – Télécopie : (33) 87 76 39 77

# Une règle de réécriture *Eta* explicite

**Résumé :** Dans ce rapport, nous étendons les $\lambda$-calculs à subsitutions explicites avec une règle *Eta*. Nous l'effectuons dans le cadre de $\lambda\upsilon$, un $\lambda$-calcul à substitution explicite introduit par Lescanne (1994) et étudié en détail par Lescanne et Rouyer-Degli (1994). La caractéristique essentielle d'un tel calcul est l'expression de la $\beta$-contraction classique à l'aide d'un système de réécriture de termes du premier ordre. Notre résultat principal consiste en l'explicitation de la $\eta$-contraction au moyen d'une règle de réécriture *Eta* non-conditionnelle, *générique* et d'une extension du calcul de substitution $\upsilon$. Les précédentes définitions de *Eta* sont dûes à Hardin (1992) et Ríos (1993) dans le cadre des $\lambda\sigma$-calculs. La principale différence entre les calculs $\lambda\upsilon$ et $\lambda\sigma$ concerne la confluence et la forte normalisation : $\lambda\upsilon$ est confluent sur les termes clos et sa version simplement typée termine, alors que $\lambda\sigma_{\Uparrow}$est confluent sur les termes ouverts mais ne termine pas sur les termes typés. Dans leurs travaux, Hardin et Ríos définissent la règle *Eta* comme une extension de la $\eta$-contraction. Leur extension est une règle conditionnelle et ne respecte pas totalement la philosophie des substitutions explicites. Dans notre approche, la règle *Eta* est une règle de réécriture du premier ordre qui emploie un calcul de subsitutions explicites. Pour cela, on introduit une nouvelle constante $\perp$ qui dénote un terme non-spécifié. Son comportement est décrit par une règle de réécriture. Ce rapport montre, d'une part, comment *Eta* associée $\lambda\upsilon$ et à la règle de $\perp$ fournit une implémentation correcte de la $\eta$-réduction, puis étudie des propriétés du $\lambda\upsilon\eta$-calcul, c'est-à-dire, la confluence sur les termes clos et la forte normalisation des termes typés. D'autre part, cette *Eta* explicite conduit à $\eta'$, une alternative à $\eta$ qui se révèle plus générale. En effet, nous prouvons que $\eta'$ autorise des contractions confluentes qui ne sont pas capturées par la $\eta$ classique.

**Mots-clé :** $\lambda$-calcul, substitutions explicites, $\eta$-réduction

# Introduction

The main feature of a $\lambda$-calculus of explicit substitutions is that the classical $\beta$-contraction is expressed by a first-order term rewriting system. In this way, such $\lambda$-calculi concur to achieve Curry's program (Cf appendix A). Indeed, one aim was to find a fully formalized prelogic, constituted of a theory of substitution (not a metatheory) and a theory of types. Upon Curry, there are two forms of such a prelogic : Combinatory Logic which is ultimate and $\lambda$-conversion which is intermediate. We think $\lambda$-calculus of explicit substitutions is as ultimate as Combinatory Logic, but more intuitive. Historically, explicit substitutions were first designed by De Bruijn [dB78] but with a crude terminology. They have been made popular more recently by Abadi, Cardelli, Curien and Lévy [ACCL91].

These authors present $\lambda\sigma$, a $\lambda$-calculus of explicit substitutions which is a result of an almost decade research work. The starting point is CCL, categorical combinator logic [Cur83, Cur86b, Cur86a], a combinatory logic more intuitive than the classical one. Indeed, it is based on $\lambda$-calculus with cartesian products and keeps its structure. Hardin [Har87, Har89] studied confluence on open terms of this calculus. An important contribution toward explicit substitutions is the $\lambda\rho$-calculus, [Cur86b] suited for weak reduction. It has been extended to $\lambda\sigma$ [ACCL91]. Hardin and Levy designed $\lambda\sigma_{\Uparrow}$[HL89], thus achieving an important goal : confluence on open terms. Guided by implementation grounds, this family of calculi rests from the beginning on the concept of composition of substitutions.

In this paper, we will work in $\lambda\upsilon$, a recent $\lambda$-calculus of explicit substitutions introduced in [Les94] and thoroughly studied in [LRD94a]. What characterizes this calculus is that the substitution calculus $\upsilon$ is an orthogonal rewriting system which does not manage composition of substitutions. Like most $\lambda$-calculi of explicit substitutions, $\lambda\upsilon$ uses De Bruijn indices for terms [dB72], beginning at $\underline{1}$. We recall the syntax of $\lambda$-terms with De Bruijn indexes :

**Definition 1 ($\Lambda$)** *The set $\Lambda$ is the set generated by :*

$$\begin{array}{lcll} \textbf{Terms} & a & ::= & \underline{n} \mid aa \mid \lambda a \\ \textbf{Naturals} & n & ::= & n+1 \mid 1. \end{array}$$

*Terms of $\Lambda$ are called pure terms.*

How does $\lambda\upsilon$ break the $\beta$-contraction ? First, the rule $Beta$ creates a substitution stored in a closure denoted by [ ] :

$$Beta \quad (\lambda a)b \ \rightarrow \ a[b/]$$

Then, rules of $\upsilon$ (Cf figure 1) distribute this substitution and apply it to indexes. The rules $\{Beta\} \cup \upsilon$ are defined on the following set of terms $\Lambda\upsilon$:

$$\begin{array}{lcll} \textbf{Terms} & a & ::= & \underline{n} \mid aa \mid \lambda a \mid a[s] \\ \textbf{Substitutions} & s & ::= & a/ \mid \Uparrow(s) \mid \uparrow \\ \textbf{Naturals} & n & ::= & n+1 \mid 1. \end{array}$$

For example, $(\lambda x.\lambda y.xy)z$, denoted $(\lambda(\lambda\underline{2}\ \underline{1}\ ))\underline{1}$ in De Bruijn notation, is contracted by $\upsilon$ as follows :

| | | | |
|---|---|---|---|
| (Beta) | $(\lambda a)b$ | $\rightarrow$ | $a[b/]$ |
| (App) | $(ab)[s]$ | $\rightarrow$ | $a[s]b[s]$ |
| (Lambda) | $(\lambda a)[s]$ | $\rightarrow$ | $\lambda(a[\Uparrow(s)])$ |
| (FVar) | $\underline{1}[a/]$ | $\rightarrow$ | $a$ |
| (RVar) | $\underline{n+1}[a/]$ | $\rightarrow$ | $\underline{n}$ |
| (FVarLift) | $\underline{1}[\Uparrow(s)]$ | $\rightarrow$ | $\underline{1}$ |
| (RVarLift) | $\underline{n+1}[\Uparrow(s)]$ | $\rightarrow$ | $\underline{n}[s][\uparrow]$ |
| (VarShift) | $\underline{n}[\uparrow]$ | $\rightarrow$ | $\underline{n+1}$ |

Figure 1: The rewrite system $\lambda\upsilon$

$$
\begin{array}{lll}
(\lambda(\lambda \underline{2}\ \underline{1}\ ))\underline{1} & \underset{Beta}{\longrightarrow} & (\lambda \underline{2}\ \underline{1}\ )[\underline{1}\ /] \\
& \underset{\upsilon}{\longrightarrow} & \lambda((\underline{2}\ \underline{1}\ )[\Uparrow(\underline{1}\ /)]) \qquad \text{rule } \textit{Lambda} \\
& \underset{\upsilon}{\longrightarrow} & \lambda(\underline{2}\ [\Uparrow(\underline{1}\ /)]\ \underline{1}\ [\Uparrow(\underline{1}\ /)]) \qquad \text{rule } \textit{App} \\
& \underset{\upsilon}{\overset{+}{\longrightarrow}} & \lambda(\underline{1}\ [\underline{1}\ /][\uparrow])\ \underline{1}\ ) \qquad \text{rules } \textit{RVarLift, FVarLift} \\
& \underset{\upsilon}{\longrightarrow} & \lambda(\underline{1}\ [\uparrow])\ \underline{1}\ ) \qquad \text{rule } \textit{FVar} \\
& \underset{\upsilon}{\longrightarrow} & \lambda(\underline{2}\ \underline{1}\ ) \qquad \text{rule } \textit{VarShift}
\end{array}
$$

Its $\upsilon$-normal form, namely $\lambda(\underline{2}\ \underline{1}\ )$, is equivalent to $\lambda y.z\ y$. We may $\eta$-reduce it to $z$. Indeed, the practical reason for defining $\eta$-reduction comes from the natural equality $(\lambda x.fx)a =_\beta fa$ if $x$ has no free occurences in $f$. It comes from the wish to make equal two functions which behave the same, that is which return the same result when applied to the same parameter (extensional equality). This is the role of $\eta$-contraction :

$$(\lambda x.fx) \underset{\eta}{\longrightarrow} f \text{ if } x \text{ has no free occurences in } f.$$

Accordingly, we would like to $\eta$-reduce $\lambda(\underline{2}\ \underline{1}\ )$ to $\underline{1}$ . We observe that in De Bruijn notation, $\eta$-contraction is not as trivial as in the classical formalism: there is some work to do to compute the $\eta$-reduct. Our aim is now to find a first order rewrite rule which explicits the substitution process involved in the $\eta$-contraction and hidden at the meta-level in all the other approaches [Har92, Río93]. In the following, we explicit the $\eta$-reduction by means of an unconditional *Eta* rewrite rule and of an extension of the substitution calculus, $\upsilon$. We show that the rewriting system obtained, denoted by $\lambda\upsilon\eta$, provides a correct implementation of the $\eta$-contraction. Moreover, *Eta* leads to a new definition of the classical $\eta$-contraction. We especially study consequences of this new definition w.r.t. ground confluence. Next, we prove some properties of the $\lambda\upsilon\eta$-calculus namely confluence on ground terms[1] and strong normalisation on typed terms. Finally, we compare our *Eta* rule to previous approaches.

# 1 Definition of $\eta$ and *Eta*

We now work in De Bruijn notation. Ríos [Río93] gives an operational definition of $\eta$ on the set $\Lambda$ of pure terms. Indeed, he proves :

$$\lambda(a\ \underline{1}) \underset{\eta}{\longrightarrow} a_1 \ \textit{if } a_1 \textit{ is defined,}$$

with the partial function $a_n$, defined at the meta-level :

$$(ab)_n = a_n b_n \qquad (\lambda a)_n = \lambda a_{n+1} \qquad \underline{m}_n = \begin{cases} \underline{m-1} & \text{if } m > n \\ undefined & \text{if } m = n \\ \underline{m} & \text{if } m < n \end{cases}$$

**Lemma 1 (Ríos)** *$\eta$ is the rewrite extension on $\Lambda$ of $\lambda(a\ \underline{1}) \underset{\eta}{\longrightarrow} a_1$ if $a_1$ is defined.*

[1] In the following, we write confluent instead of ground confluent

Starting from this and unlike Ríos, we try to define a primitive *Eta* rule. Our reasoning is based on two facts. First, $a_n$ is an effective partial function and should be computed by a rewrite system. Second, the function $a_n$ should look like a / substitution according to the following similarities.

*$a_{n+1}$ leaves unchanged the indexes from $\underline{1}$ to $\underline{n}$, decreases the indexes after $\underline{n+2}$ and rules out $\underline{n+1}$.*

*$a[\Uparrow^n(b/)]$ leaves unchanged the indexes from $\underline{1}$ to $\underline{n}$, decreases the indexes after $\underline{n+2}$ and replaces $\underline{n+1}$ by $b[\uparrow^n]$.*

We see that $a_{n+1}$ and $a[\Uparrow^n(b/)]$ have the same effect if the term $a$ does not contain the index $\underline{n+1}$. If $a$ contains $\underline{n+1}$, its presence is remembered by a special term, namely $\bot$.

Let us define the set of terms $\Lambda\upsilon_\bot$.

**Definition 2** *The set $\Lambda\upsilon_\bot$ is the set generated by:*

$$\begin{array}{lllll} \textbf{Terms} & a & ::= & \underline{n} \mid aa \mid \lambda a \mid a[s] \mid \bot \\ \textbf{Subst} & s & ::= & a/ \mid \Uparrow(s) \mid \uparrow \\ \textbf{Naturals} & n & ::= & n+1 \mid 1. \end{array}$$

Now, we are able to define a rule on $\Lambda\upsilon_\bot$ which we call *Eta*:

**Definition 3** *Let $a, b \in \Lambda\upsilon_\bot$. Eta is the rewrite extension on $\Lambda\upsilon_\bot$ of :*

$$\lambda(a\ \underline{1}) \underset{Eta}{\longrightarrow} a[\bot/]$$

As $\bot$ is a constant, we add the rule:

**Definition 4** *$\upsilon_\bot$ is: $\upsilon \cup \{\bot[s] \rightarrow \bot\}$.*

In appendix B, we extend the properties of $\lambda\upsilon$ proved in [LRD94a] to a $\lambda\upsilon$-calculus with a finite set of constants. Lemmas 16 and 17 prove that $a[\Uparrow^n(\bot/)]$ has the expected behaviour:

**Lemma 2** *Let $m \geq 1$ and $n \geq 0$.*

1. $\underline{m}\,[\Uparrow^n(\bot/)] \underset{\upsilon}{\overset{*}{\longrightarrow}} \underline{m}$ *if* $1 \leq m \leq n$
2. $\underline{n+1}\,[\Uparrow^n(\bot/)] \underset{\upsilon}{\overset{*}{\longrightarrow}} \bot[\uparrow^n] \underset{\upsilon_\bot}{\overset{*}{\longrightarrow}} \bot$
3. $\underline{m}\,[\Uparrow^n(\bot/)] \underset{\upsilon}{\overset{*}{\longrightarrow}} \underline{m-1}$ *if* $m \geq n+2$.

We show that *Eta* is correct on pure terms w.r.t. $\eta$, that is to say, an *Eta*-rewrite followed by $\upsilon$-normalisation is equivalent to an $\eta$-contraction. We introduce another relation $\eta'$ defined on $\Lambda$ and we prove this relation is actually $\eta$:

**Definition 5** *Let $a \in \Lambda$. $\lambda(a\ \underline{1}) \underset{\eta'}{\longrightarrow} b$ iff $\lambda(a\ \underline{1}) \underset{Eta}{\longrightarrow} a[\bot/]$ and $\upsilon(a[\bot/]) = b \in \Lambda$. We extend $\eta'$ by rewrite extension on $\Lambda$.*

**Lemma 3** *Let $a \in \Lambda$, $a_{n+1}$ is defined if and only if $\upsilon(a[\Uparrow^n(\bot/)]) \in \Lambda$ and in that case $a_{n+1} = \upsilon(a[\Uparrow^n(\bot/)])$.*

**Proof:**

1. Let $a \in \Lambda$ and suppose that $a_{n+1}$ is defined. We proceed by structural induction on $a$:
   (a) $a = \underline{m}$
   By hypothesis, $a_{n+1}$ exists, so $a = \underline{m} \neq \underline{n+1}$.
      i. $m > n+1$, $\underline{m}_{n+1} = \underline{m-1}$
      By lemma 2, $\upsilon(\underline{m}\,[\Uparrow^n(\bot/)]) = \underline{m-1} \in \Lambda, = \underline{m}_{n+1}$
      ii. $m < n+1$, $\underline{m}_{n+1} = \underline{m}$
      By lemma 2, $\upsilon(\underline{m}\,[\Uparrow^n(\bot/)]) = \underline{m} \in \Lambda, = \underline{m}_{n+1}$

(b) $a = \lambda b$
As $a_{n+1}$ is defined and $a_{n+1} = (\lambda b)_{n+1} = \lambda b_{n+2}$, $b_{n+2}$ is defined. Then, by induction hypothesis, $\upsilon(b[\Uparrow^{n+1}(\perp/)]) \in \Lambda$ and equals $b_{n+2}$.

$$\begin{array}{rcl} a[\Uparrow^{n}(\perp/)] & = & (\lambda b)[\Uparrow^{n}(\perp/)] \\ & \underset{\upsilon}{\rightarrow} & \lambda(b[\Uparrow^{n+1}(\perp/)]) \\ & \underset{\upsilon}{\overset{*}{\rightarrow}} & \lambda(b_{n+2}) \\ & \in & \Lambda \\ & = & (\lambda b)_{n+1} \\ & = & a_{n+1} \end{array}$$

(c) $a = bc$
Immediate by application of the induction hypothesis to $b$ and $c$.

2. Let $a \in \Lambda$ and suppose $\upsilon(a[\Uparrow^{n}(\perp/)]) \in \Lambda$. $a_{n+1}$ is defined. Indeed, by structural induction on $a$:

(a) $a = \underline{m}$
Suppose $a_{n+1}$ is not defined. Therefore $a = \underline{n+1}$ and by lemma 2, $\upsilon(\underline{n+1}[\Uparrow^{n}(\perp/)]) = \perp[\uparrow^{n}] \notin \Lambda$. This contradicts the hypothesis, so $a_{n+1}$ is defined.

(b) $a = \lambda b$
$\upsilon((\lambda b)[\Uparrow^{n}(\perp/)]) = \lambda\upsilon(b[\Uparrow^{n+1}(\perp/)]) \in \Lambda$. So $\upsilon(b[\Uparrow^{n+1}(\perp/)]) \in \Lambda$, and by induction hypothesis, $b_{n+2}$ is defined. As $a_{n+1} = (\lambda b)_{n+1} = \lambda b_{n+2}$, $a_{n+1}$ is defined.

(c) $a = bc$
Immediate by application of the induction hypothesis to $b$ and $c$.

□

We now prove equivalence of $\eta$ and $\eta'$ contractions for redexes located at the head of terms:

**Lemma 4** *Let $a \in \Lambda$, $\lambda(a\ \underline{1}) \underset{\eta}{\rightarrow} a_1$ if and only if $\lambda(a\ \underline{1}) \underset{\eta'}{\rightarrow} \upsilon(a[\perp/]) = a_1$.*

**Proof:**

1. Suppose $\lambda(a\ \underline{1}) \underset{\eta}{\rightarrow} a_1$. By previous lemma, as $a_1$ is defined, we know: $\upsilon(a[\perp/]) = a_1 \in \Lambda$. So $\lambda(a\ \underline{1}) \underset{Eta}{\rightarrow} a[\perp/] \underset{\upsilon}{\overset{*}{\rightarrow}} a_1$. That is to say, $\lambda(a\ \underline{1}) \underset{\eta'}{\rightarrow} \upsilon(a[\perp/]) = a_1 \in \Lambda$.
2. Suppose $\lambda(a\underline{1}) \underset{\eta'}{\rightarrow} \upsilon(a[\perp/]) \in \Lambda$. By previous lemma, $a_1$ is defined and equals $\upsilon(a[\perp/])$. So, $\lambda(a\ \underline{1}) \underset{\eta}{\rightarrow} \upsilon(a[\perp/])$.

□

More generally, by rewrite extension on $\Lambda$, we get the following proposition: *Eta* followed by the $\upsilon$-normalisation[2] correctly implements the classical $\eta$-reduction.

**Proposition 1 (Correction)** *Let $a, b \in \Lambda$, $a \underset{\eta}{\rightarrow} b$ if and only if $a \underset{\eta'}{\rightarrow} b$.*

So we achieve our aim: we compute the $\eta$-reduct by a first order term rewriting system, as this computation is expressed through explicit substitutions.

*Beta* is now extended to $\Lambda\upsilon_{\perp}$:

**Definition 6** *Let $a, b \in \Lambda\upsilon_{\perp}$. Beta is the rewrite extension on $\Lambda\upsilon_{\perp}$ of: $(\lambda a)b \underset{Beta}{\rightarrow} a[b/]$*

The definition of $\beta'$ w.r.t. [LRD94a] is unchanged:

**Definition 7** *Let $a, b \in \Lambda$. $\beta'$ is the rewrite extension on $\Lambda$ of: $(\lambda a)b \underset{\beta'}{\rightarrow} \upsilon(a[b/])$*

$\beta'$ is correct w.r.t. $\beta$, as already proved in [LRD94a].

[2] We do not need the constant rule $\perp[s] \rightarrow \perp$

## 2 Confluence of $\beta\eta'$

The *Eta* rule leads us to a new view of the classical $\eta$-contraction, i.e. $\eta$ expressed in classical formalism (without De Bruijn notation). Indeed, we define a new $\eta$-contraction in the classical $\lambda$-calculus, denoted by $\eta'$ as :

$$\lambda x.(a\ x) \underset{\eta'}{\longrightarrow} a\{\bot/x\}$$

Notice that $\eta'$ is unconditional. As shown by correction, this rule coincides with classical $\eta$ in the case $x$ is not a free variable of $a$. In the other case, classical $\eta$ is not allowed, but $\eta'$ is. In some cases, it makes sense to forget the precondition on the application of classical $\eta$. The following examples show that some $\eta'$-contractions, classically forbidden, can in fact be allowed and keep the whole calculus confluent.

1. $(\lambda xy.x)(\lambda x.x)(\lambda x.x\ x) \underset{\eta'}{\longrightarrow} (\lambda xy.x)(\lambda x.x)\bot \underset{\beta}{\overset{*}{\longrightarrow}} (\lambda x.x)$
   $(\lambda xy.x)(\lambda x.x)(\lambda x.x\ x) \underset{\beta}{\overset{*}{\longrightarrow}} (\lambda x.x)$
2. $\lambda x.((\lambda y.z)x\ x) \underset{\eta'}{\longrightarrow} (\lambda y.z)\bot \underset{\beta}{\longrightarrow} z$
   $\lambda x.((\lambda y.z)x\ x) \underset{\beta}{\longrightarrow} \lambda x.(z\ x) \underset{\eta'}{\longrightarrow} z$
3. $(\lambda u.(\lambda x.uxx))(\lambda y.z) \underset{\eta'}{\longrightarrow} (\lambda u.u\bot)(\lambda y.z) \underset{\beta}{\longrightarrow} (\lambda y.z)\bot \underset{\beta}{\longrightarrow} z$
   $(\lambda u.(\lambda x.uxx))(\lambda y.z) \underset{\beta}{\longrightarrow} (\lambda x.(\lambda y.z)xx) \underset{\beta}{\longrightarrow} (\lambda x.zx) \underset{\eta'}{\longrightarrow} z$

The rest of this section is devoted to the study of this unconditional $\eta'$. To be consistent with our older definitions, we go back to De Bruijn notation, but the following statements hold in classical $\lambda$-calculus. First, we extend $\Lambda$ to $\Lambda_\bot$, the set of pure terms that may contain $\bot$ constant occurences. We define unconditional $\eta'$ on this set. Next, we look for a set $\Lambda m$ on which the relation $\beta\eta'$ is confluent. To prove this confluence, we first show the postponement of $\eta'$-steps w.r.t. $\beta$-steps.

We define $\beta'$ and $\eta'$ on the set $\Lambda_\bot$ :

**Definition 8 ($\Lambda_\bot$)** *The set $\Lambda_\bot$ is the set generated by:*

$$\begin{array}{lllll} \textbf{Terms} & a & ::= & \underline{n} \mid aa \mid \lambda a \mid \bot \\ \textbf{Naturals} & n & ::= & n+1 \mid 1. \end{array}$$

**Definition 9** *Let $a, b \in \Lambda_\bot$.*

1. *$\beta'$ is the rewrite extension on $\Lambda_\bot$ of: $(\lambda a)b \underset{\beta'}{\longrightarrow} v_\bot(a[b/])$*
2. *$\eta'$ is the rewrite extension on $\Lambda_\bot$ of: $\lambda(a\ \underline{1}) \underset{\eta'}{\longrightarrow} v_\bot(a[\bot/])$*

In appendix B, we show that $\beta'$ and classical $\beta$ coincide on the set $\Lambda_\bot$, as $\bot$ is a constant. So we will write $\beta$ for both.

$\beta\eta'$ is clearly not confluent on $\Lambda_\bot$, i.e. on the set of terms containing $\bot$ occurences, as shown by the critical pair between $\beta$ and $\eta'$ :

$$ab \underset{\beta'}{\longleftarrow} (\lambda(a\ \underline{1}))b \underset{\eta'}{\longrightarrow} a[\bot/]b$$

In $\Lambda_\bot$, we have the counter-example :

$$\underline{1}\ \underline{1} \underset{\beta'}{\longleftarrow} (\lambda(\underline{1}\ \underline{1}))\ \underline{1} \underset{\eta'}{\longrightarrow} \bot\ \underline{1}$$

But by restricting $\eta'$ to a reasonably large subset of $\Lambda_\bot$, we can make $\beta\eta'$ confluent. This set is $\Lambda m$ :

**Definition 10**

$$\Lambda m = \{a \in \Lambda_\bot \mid \exists b \in \Lambda\ :\ a \underset{\beta\eta'}{\overset{*}{\longrightarrow}} b\}$$

This set is larger than $\Lambda$ and than the set used by Hardin and Ríos in the sense that $\eta'$ contains more reductions and $\Lambda m$ contains terms with occurences of $\bot$ (See fig. 2). For example, the terms $(\lambda(\underline{1}\ \underline{1}))\ \underline{1}$ and $\underline{1}\ \underline{1}$ belong to $\Lambda m$, but $\bot\ \underline{1}$ does not.

As already noticed, an $\eta'$-contraction does not necessarily coincide with a classical $\eta$-contraction. We qualify such an $\eta'$-contraction as biased. In the following, we show that on the set $\Lambda m$, we can either postpone a

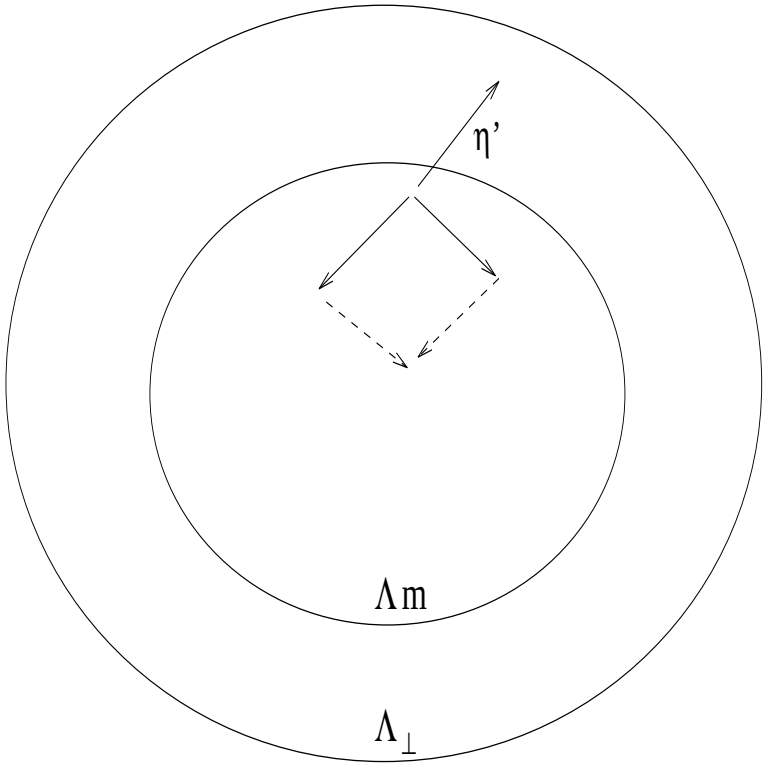


Figure 2: Relations between $\Lambda m$ and $\Lambda_\perp$

biased $\eta'$-contraction and so transform it in a classical $\eta$ one, or eliminate it. This way, confluence of $\beta\eta'$ follows immediately. Consider the reduction :

$$a \underset{\eta'}{\longrightarrow} b \underset{\beta}{\longrightarrow} c$$

We successively prove :

1. the below case : when the $\eta'$-redex is strictly contained in the $\beta$-redex, it can be postponed and may be duplicated (or eliminated).
2. the upon case : when the $\eta'$-redex strictly contains the $\beta$-redex, it can be postponed.
3. the critical case : when there is a critical pair, the $\eta'$-contraction can be eliminated.

The first two cases hold in $\Lambda_\perp$, the last one holds only in $\Lambda m$. Due to potential duplication, there is a technicality for the below case.

First, we show the below case. To treat it formally and according to the philosophy of explicit substitution, we need what is called projection lemmas. In the case of *Eta*, this lemma says that an *Eta*-rewrite between terms of $\Lambda\upsilon_\perp$ translates in a $\eta'$-reduction between their $\upsilon_\perp$-normal forms. If the *Eta*-rewrite takes place inside a closure [ ] we call it internal, if it takes place outside each closure, we call it external. External and internal are abbreviated by the superscripts *ext* and *int*.

**Lemma 5 (Projection lemmas on $\Lambda_\perp$)**

1. *If* $a, b \in \Lambda\upsilon_\perp$ *such that* $a \underset{Eta}{\overset{ext}{\longrightarrow}} b$ *then* $\upsilon_\perp(a) \underset{\eta'}{\longrightarrow} \upsilon_\perp(b)$.

2. *If* $a, b \in \Lambda\upsilon_\perp$ *such that* $a \underset{Eta}{\overset{int}{\longrightarrow}} b$ *then* $\upsilon_\perp(a) \overset{*}{\underset{\eta'}{\longrightarrow}} \upsilon_\perp(b)$

The proof of this lemma is in appendix C. The *Beta* projection lemma is similar, its proof extends the one given in [LRD94a] (Cf appendix B).

**Lemma 6 (The case $\eta'$ below $\beta$)** *Let* $a, b, c \in \Lambda$.

1. *If* $(\lambda a)b \underset{\eta'}{\longrightarrow} (\lambda c)b \underset{\beta}{\longrightarrow} \upsilon_\perp(c[b/])$ *then* $(\lambda a)b \underset{\beta}{\longrightarrow} \upsilon_\perp(a[b/]) \underset{\eta'}{\longrightarrow} \upsilon_\perp(c[b/])$.
2. *If* $(\lambda b)a \underset{\eta'}{\longrightarrow} (\lambda b)c \underset{\beta}{\longrightarrow} \upsilon_\perp(b[c/])$ *then* $(\lambda b)a \underset{\beta}{\longrightarrow} \upsilon_\perp(b[a/]) \overset{*}{\underset{\eta'}{\longrightarrow}} \upsilon_\perp(b[c/])$.

The proof comes directly from the projection lemma 5.

**Proof:**

1. We suppose :

$$(\lambda\lambda(a\ \underline{1}))b \underset{\eta'}{\longrightarrow} (\lambda\upsilon_\perp(a[\perp/]))b \underset{\beta}{\longrightarrow} \upsilon_\perp(\upsilon_\perp(a[\perp/][b/])) = \upsilon_\perp(a[\perp/][b/])$$

We observe :

$$(\lambda\lambda(a\ \underline{1}))b \underset{Beta}{\overset{ext}{\longrightarrow}} (\lambda(a\ \underline{1}))[b/] \underset{Eta}{\overset{ext}{\longrightarrow}} a[\perp/][b/]$$

By external projection lemmas, we get :

$$(\lambda\lambda(a\ \underline{1}))b \underset{\beta}{\longrightarrow} \upsilon_\perp((\lambda(a\ 1))[b/]) \underset{\eta'}{\longrightarrow} \upsilon_\perp(a[\perp/][b/])$$

The key part, i.e. Barendregt style substitution lemma [Bar84], is inside the projection lemma :

$$a[\Uparrow(b/)][\perp/] \overset{*}{\underset{\upsilon_\perp}{\longleftrightarrow}} a[\Uparrow(b/)][\perp[b/]/] \overset{*}{\underset{\upsilon_\perp}{\longleftrightarrow}} a[\perp/][b/]$$

2. We suppose :

$$(\lambda b)(\lambda(a\ \underline{1})) \underset{\eta'}{\longrightarrow} (\lambda b)(\upsilon_\perp(a[\perp/])) \underset{\beta}{\longrightarrow} \upsilon_\perp(b[\upsilon_\perp(a[\perp/])/]) = \upsilon_\perp(b[a[\perp/]/])$$

We observes :

$$(\lambda b)(\lambda(a\ \underline{1})) \underset{Beta}{\overset{ext}{\longrightarrow}} b[(\lambda(a\ \underline{1}))/] \underset{Eta}{\overset{int}{\longrightarrow}} b[a[\perp/]/]$$

By external and internal projection lemmas, we get :

$$(\lambda b)(\lambda(a\ \underline{1})) \underset{\beta}{\longrightarrow} \upsilon_\perp(b[(\lambda(a\ \underline{1}))/]) \overset{*}{\underset{\eta'}{\longrightarrow}} \upsilon_\perp(b[a[\perp/]/])$$

□

**Lemma 7 (The case $\eta'$ above $\beta$)** *Let* $a, c \in \Lambda_\perp$. *If* $\lambda(a\ \underline{1}) \underset{\eta'}{\longrightarrow} \upsilon_\perp(a[\perp/]) \underset{\beta}{\longrightarrow} c$ *then there exists* $d \in \Lambda_\perp$ *such that* $\lambda(a\ \underline{1}) \underset{\beta}{\longrightarrow} d \underset{\eta'}{\longrightarrow} c$.

**Proof:** We translate the hypothesis

$$\lambda(a\ \underline{1}) \underset{\eta'}{\longrightarrow} \upsilon_\perp(a[\perp/]) \underset{\beta}{\longrightarrow} c$$

into a $\beta$-reduction :

$$(\lambda\ a)\perp \underset{\beta}{\longrightarrow} \upsilon_\perp(a[\perp/]) \underset{\beta}{\longrightarrow} c$$

Since $\perp$ is not an abstraction, the first $\beta$-contraction does not create a $\beta$-redex in $\upsilon_\perp(a[\perp/])$. So, the structure of the $\beta$-redex in $\upsilon_\perp(a[\perp/])$ is already in $a$. Hence, there exists a $d \in \Lambda_\perp$ such that :

$$a \underset{\beta}{\longrightarrow} d$$

As we have :

$$(\lambda\ a)\perp \underset{\beta}{\longrightarrow} \upsilon_\perp(a[\perp/]) \underset{\beta}{\longrightarrow} c$$

$$(\lambda\ a)\perp \underset{\beta}{\longrightarrow} (\lambda\ d)\perp \underset{\beta}{\longrightarrow} \upsilon_\perp(d[\perp/])$$

By the substitution lemma for $\beta$ [HS86], we get :

$$c = v_\perp(d[\perp/])$$

Back to $\eta'$-contraction, we get the postponement :

$$\lambda(a\ \underline{1}) \underset{\beta}{\rightarrow} \lambda(d\ \underline{1}) \underset{\eta'}{\rightarrow} v_\perp(d[\perp/]) = c$$

□

The previous lemmas imply :

**Lemma 8** *Let $a, b, c \in \Lambda_\perp$. If $a \underset{\eta'}{\overset{*}{\rightarrow}} b \underset{\beta}{\overset{*}{\rightarrow}} c$ when $\eta'$ is below or above $\beta$ then there exists $d \in \Lambda_\perp$ such that: $a \underset{\beta}{\overset{*}{\rightarrow}} d \underset{\eta'}{\overset{*}{\rightarrow}} b$.*

The proof is by a double induction on, first, the length of the $\eta'$-reduction, and, second, the length of the $\beta$-reduction.

The following lemma shows what becomes of the classical commutation of classical $\beta$ and $\eta$.

**Lemma 9 (The critical case)** *Let $M, N \in \Lambda_\perp$. If $(\lambda(M\ \underline{1}))N \underset{\eta'}{\rightarrow} (v_\perp(M[\perp/]))N \underset{\beta}{\overset{*}{\rightarrow}} P \in \Lambda$ then $(\lambda(M\ \underline{1}))N \underset{\beta}{\overset{*}{\rightarrow}} P$.*

**Proof:** We prove it with the classical formalism. Here, we treat $\perp$ as a free variable and we rename $\perp$ by $\top$, a fresh variable, in $M$ and $N$ :
$N' \equiv N[\top/\perp]$ and $N' \equiv N[\top/\perp]$. We have :

$$(\lambda x.M'\ x)N' \underset{\eta'}{\rightarrow} (M'[\perp/x])N' \underset{\beta}{\overset{*}{\rightarrow}} P \in \Lambda$$

By stability of $\beta$ (or substitution lemma [HS86]) :

$$((M'[\perp/x])N')[N'/\perp] \underset{\beta}{\overset{*}{\rightarrow}} P[N'/\perp]$$

As $\perp \notin FV(N'P)$, $P[N'/\perp] = P$ and $N'[N'/\perp] = N'$.

$$(M'[\perp/x][N'/\perp])N' \underset{\beta}{\overset{*}{\rightarrow}} P$$

And $M'[\perp/x][N'/\perp] = M'[N'/x]$,

$$(M'[N'/x])N' \underset{\beta}{\overset{*}{\rightarrow}} P$$

By $\beta$-expansion :

$$(\lambda x.M'\ x)N' \underset{\beta}{\overset{*}{\rightarrow}} P$$

By renaming, as $\top \notin FV(MN)$,

$$(\lambda x.M\ x)N \underset{\beta}{\overset{*}{\rightarrow}} P$$

□

**Lemma 10 (Postponement of $\eta'$-contractions)** *Let $a \in \Lambda m$ and $c \in \Lambda$ such that: $a \underset{\beta\eta'}{\overset{*}{\rightarrow}} c$. Then there exists $d \in \Lambda$ : $a \underset{\beta}{\overset{*}{\rightarrow}} d \underset{\eta}{\overset{*}{\rightarrow}} c$.*

**Proof:** We proceed by induction on the number of $\eta'$-steps. We consider the last $\eta'$ step. If there is no $\eta'$-step, we get the result. If this step is a critical case, by lemma 9, we eliminate it. If this step is an upon or below case, by lemma 8, it is postponed and may be duplicated (or eliminated). As only $\beta$-steps eliminate $\perp$ occurences, final $\eta'$-steps are in fact classical $\eta$-steps : $a \underset{\beta\eta'}{\overset{*}{\rightarrow}} b \underset{\eta}{\overset{*}{\rightarrow}} c$ and $b \in \Lambda$, so that we apply the induction hypothesis on $a \underset{\beta\eta'}{\overset{*}{\rightarrow}} b$. □

**Theorem 1** *$\beta\eta'$ is confluent on $\Lambda m$.*

**Proof:** Let $a, b, c \in \Lambda m$ such that $a \underset{\beta\eta'}{\overset{*}{\rightarrow}} b$ and $a \underset{\beta\eta'}{\overset{*}{\rightarrow}} c$. As $b, c \in \Lambda m$, there exists $b', c' \in \Lambda$ such that $b \underset{\beta\eta'}{\overset{*}{\rightarrow}} b'$ and $c \underset{\beta\eta'}{\overset{*}{\rightarrow}} c'$. So :

$$a \underset{\beta\eta'}{\overset{*}{\rightarrow}} b' \quad a \underset{\beta\eta'}{\overset{*}{\rightarrow}} c'$$

By previous lemma, we associate to these two reductions two classical ones :

$$a \underset{\beta\eta}{\overset{*}{\rightarrow}} b' \quad a \underset{\beta\eta}{\overset{*}{\rightarrow}} c'$$

As classical $\beta\eta$ is confluent on $\Lambda$ plus a constant, there exists $d \in \Lambda$ ($d \notin \Lambda_\perp$ because $\eta$-steps are correct and $b', c' \in \Lambda$) :

$$b' \underset{\beta\eta}{\overset{*}{\rightarrow}} d \quad c' \underset{\beta\eta}{\overset{*}{\rightarrow}} d$$

By correction, $\eta'$ can simulate $\eta$ :

$$b' \underset{\beta\eta'}{\overset{*}{\rightarrow}} d \quad c' \underset{\beta\eta'}{\overset{*}{\rightarrow}} d$$

□

The definition of $\Lambda m$ is very general and we do not know a syntactic characterization for it (we conjecture there is none). If one wants to implements an $\eta'$ reduction strategy, one may wish to know if one stays in $\Lambda m$. The absence of a syntactic characterization seems to prevent providing such a criterion. Even a smaller set, like,

$$\{a \in \Lambda_\perp | \exists b \in \Lambda \ : a \underset{\beta}{\overset{*}{\rightarrow}} b\}$$

is of no more help. Nevertheless, $\eta'$ sheds more light on the relation between $\eta$ and $\beta$.

# 3 Confluence and strong normalization of $\lambda\upsilon\eta$

In this section, we study two properties of $\lambda\upsilon\eta$, the rewrite system $\{Beta, Eta\} \cup \upsilon_\perp$, namely confluence and strong normalization. The proof of these properties are straightforward extension of $\lambda\upsilon$ ones, so we just sketch them.

## 3.1 Confluence of $\lambda\upsilon\eta$

We prove confluence of $\lambda\upsilon\eta$, the theory $\lambda\upsilon$ augmented with the rule $Eta$ on a smaller and handy set :

**Definition 11**

$$\Lambda\upsilon\eta = \{a \in \Lambda\upsilon_\perp | \exists b \in \Lambda\upsilon \ : a \underset{\upsilon}{\overset{*}{\rightarrow}} b\}$$

We prove the confluence of $\lambda\upsilon\eta$ by the interpretation method [HL89]. We review it for our particular case. Consider the relation $\lambda\upsilon\eta$ defined on the set $\Lambda\upsilon\eta$ and such that :

1. $\lambda\upsilon\eta = R \cup \upsilon_\perp$ with $R = \{Beta, Eta\}$
2. $\upsilon_\perp$ is convergent, that is to say confluent and strongly normalizing, on the set $\Lambda$ of $\upsilon_\perp$-normal forms of $\Lambda\upsilon\eta$.
3. $\beta\eta \subset (\lambda\upsilon\eta)^*$

If

1. $\beta\eta$ is defined on the set $\Lambda$ of $\upsilon_\perp$-normal forms of $\Lambda\upsilon\eta$, and
2. the projection lemmas on $\Lambda$ hold : a $Beta$ (resp. $Eta$) contraction between terms of $\Lambda\upsilon\eta$ translates in a $\beta$ (resp. $\eta$) reduction between their $\upsilon_\perp$-normal forms.

then

$$\beta\eta \text{ is confluent on } \Lambda \text{ if and only if } \lambda\upsilon\eta \text{ is confluent on } \Lambda\upsilon\eta.$$

So the confluence of $\lambda\upsilon\eta$ on $\Lambda\upsilon\eta$ relies on the projection lemmas of $Beta$ and $Eta$ on $\Lambda$. In a first step, we recall the projection lemmas concerning terms of $\Lambda\upsilon_\perp$ :

**Lemma 11 (Projection lemmas on $\Lambda_\perp$)** *If $a, b \in \Lambda\upsilon_\perp$ such that $a \underset{Beta}{\longrightarrow} b$ (resp. $a \underset{Eta}{\longrightarrow} b$) then $\upsilon_\perp(a) \overset{*}{\underset{\beta}{\longrightarrow}} \upsilon_\perp(b)$ (resp. $\upsilon_\perp(a) \overset{*}{\underset{\eta'}{\longrightarrow}} \upsilon_\perp(b)$).*

As a subcase, as $\upsilon_\perp(b) \in \Lambda$ for $b \in \Lambda\upsilon\eta$, we get :

**Corollary 1 (Projection lemmas on $\Lambda$)** *If $a, b \in \Lambda\upsilon\eta$ such that $a \underset{Beta}{\longrightarrow} b$ (resp. $a \underset{Eta}{\longrightarrow} b$) then $\upsilon_\perp(a) \overset{*}{\underset{\beta}{\longrightarrow}} \upsilon_\perp(b)$ (resp. $\upsilon_\perp(a) \overset{*}{\underset{\eta'}{\longrightarrow}} \upsilon_\perp(b)$) and $\upsilon_\perp(a), \upsilon_\perp(b) \in \Lambda$.*

**Theorem 2** *The reduction $\lambda\upsilon\eta$ is confluent on $\Lambda\upsilon\eta$.*

So, we prove the ground confluence of $\lambda\upsilon\eta$ on a set equivalent to, on the one hand, the classical one, on the other hand, the Hardin *ground* one. Unlike Hardin, we can not achieve a stronger result, i.e. confluence on terms containing variables of type **Term** or **Substitution**. Indeed, there is a counter-example to such a confluence, which is already present in $\lambda\upsilon$ :

$$a[b/][s] \underset{Beta}{\longleftarrow} ((\lambda a)b)[s] \overset{+}{\underset{\upsilon}{\longrightarrow}} (\lambda a[\Uparrow(s)])(b[s]) \underset{Beta}{\longrightarrow} a[\Uparrow(s)][b[s]/]$$

If we consider $a$, $b$ and $s$ as variables, we get two distinct $\upsilon$-normal forms. But for each ground instance of $a$, $b$ and $s$, we have (Cf appendix B) :

$$a[b/][s] \overset{*}{\underset{\upsilon}{\longleftrightarrow}} a[\Uparrow(s)][b[s]/]$$

That is to say, it is a theorem of the inductive theory, not of the equational theory of $\upsilon$. On the contrary, $\lambda\sigma_\Uparrow$ -calculus is confluent on open terms [HL89]. As a counterpart, typed $\lambda\upsilon$ is strongly normalizing, an interesting property which does not hold in $\lambda\sigma_\Uparrow$ nor in $\lambda\sigma$ [Mel95].

## 3.2 Strong normalization of typed $\lambda\upsilon\eta$

We now look at the preservation of strong normalisation of $\lambda\upsilon\eta$ on $\Lambda\upsilon_\perp$ terms. Then, we deduce strong normalisation of $\lambda\upsilon\eta$ on simply typed terms.

We adapt the $\lambda\upsilon$ strong normalization preservation proof [LRD94a] and comment it. The main ideas are : use the strong normalization of $\beta\eta'$ and the fact that $b/$ substitutions, with $b \neq \perp$, come from *Beta* rewrites. We emphasize the fact that this last property of $\upsilon$ is essential to this proof of strong normalization of $\lambda\upsilon$ and $\lambda\upsilon\eta$. The following lemma formalizes this property.

**Lemma 12** *Let $a_0 \in \Lambda_\perp$ such that $a_0 \overset{*}{\underset{\lambda\upsilon\eta}{\longrightarrow}} a_n \equiv C\{d[\Uparrow^i(c/)]\}$ with $c \neq \perp$. Then there exists $a_i$, $0 \leq i \leq n$ such that : $a_i \equiv D\{(\lambda e)b\}$ and $b \overset{*}{\underset{\lambda\upsilon\eta}{\longrightarrow}} c$.*

The proof can be found in [BBLRD94]. This lemma does not hold in the $\sigma$ calculus because of the rule $(a \cdot s) \circ t \ \rightarrow \ a[t] \cdot (s \circ t)$. Indeed, one observes that it creates a closure [ ] and so it may create $[b \cdot id]$, the equivalent in $\lambda\sigma$ of $[b/]$ [Mel95].

We now state the second key point of this normalization proof : this lemma isolates the sources of the potentially infinite derivations in closures [ ].

**Lemma 13** *Let $a \in \Lambda\upsilon_\perp$ such that $\upsilon_\perp(a)$ is strongly normalizing. In a $\lambda\upsilon\eta$ derivation starting from $a$, there exists a rank $N$ such that each $\lambda\upsilon\eta$-step following $N$ is internal.*

That property, proved in [LRD94a], depends only on $\upsilon$, not on *Beta*, $\perp$ or *Eta*. It is not shared by the $\sigma$ substitution calculus [ACCL91] because of the same rule. Indeed, in the $\sigma$-derivation :

$$\underline{1}[(a \cdot s) \circ t] \ \rightarrow^{int} \ \underline{1}[a[t] \cdot (s \circ t)] \ \rightarrow^{ext} a[t]$$

the external redex $\underline{1}[\,\cdot\,]$ produced by the internal $\sigma$-rewrite $(a \cdot s) \circ t \ \rightarrow \ a[t] \cdot (s \circ t)$ can not be pushed up.

By the two previous lemmas, we get :

**Theorem 3** *Let $a \in \Lambda\upsilon_\perp$ such that $\upsilon_\perp(a)$ is $\beta\eta'$-strongly normalizing. Then, $a$ is $\lambda\upsilon_\perp$-strongly normalizing.*

The proof follows closely the $\lambda\upsilon$ case, described in [BBLRD94]. It is based on a minimal counter-example, more precisely a minimal $\lambda\upsilon\eta$-derivation.

So $\lambda\upsilon\eta$ preserves strong normalization. As a consequence, we derive strong normalization of a simply typed version of $\lambda\upsilon\eta$ on $\Lambda\upsilon_\perp$. For this, first, we need a typing system, second, we have to show $\beta\eta'$ is strongly normalizing on simply typed pure terms.

Thus, we enlarge $\lambda\upsilon$ simply typed terms [LRD94a] to $\lambda\upsilon\eta$ ones. This part heavily relies on the simply typed version of $\lambda\upsilon$-calculus described in [LRD94a]. To introduce a typed *Eta* rule, we have to type terms of $\Lambda\upsilon_\perp$. For this, instead of a single constant $\perp$, we need, for each simple type $A$, a typed constant $\perp_A$ and a rule $\perp_A[s] \rightarrow \perp_A$. We just add an axiom scheme to the typing system of $\lambda\upsilon$ in order to type $\perp_A$ and terms containing occurences of $\perp_A$.

The grammar of the pseudo-terms is:

| | | | |
|---|---|---|---|
| **Type** | $A$ | ::= | $\mathsf{A}_1 \mid \ldots \mid \mathsf{A}_n \mid A \Rightarrow B$ |
| **Naturals** | $n$ | ::= | $1 \mid n+1$ |
| **Terms** | $a$ | ::= | $\underline{n} \mid ab \mid \lambda A.a \mid a[s] \mid \perp_A$ |
| **Substitutions** | $s$ | ::= | $a/ \mid \uparrow \mid \Uparrow(s)$ |
| **Context** | $\Gamma$ | ::= | $[\,] \mid A \cdot \Gamma$ |

where $\mathsf{A}_1 \ldots \mathsf{A}_n$ is a family of atomic types. The set of $\lambda\upsilon\eta$-simply typed terms is noted $\Lambda\upsilon^{\rightarrow}_{\perp}$ and is produced by the typing system :

**Terms**

$$\frac{}{\Gamma \vdash \perp_A : A}$$

$$\frac{\Gamma \vdash a : A \Rightarrow B \quad \Gamma \vdash b : A}{\Gamma \vdash ab : B} \qquad \frac{A \cdot \Gamma \vdash a : B}{\Gamma \vdash \lambda A.a : A \Rightarrow B}$$

$$\frac{\Gamma \vdash a : A \quad \Delta \vdash s : \Gamma}{\Delta \vdash a[s] : A} \qquad \frac{}{A \cdot \Gamma \vdash \underline{1} : A} \qquad \frac{\Gamma \vdash \underline{n} : A}{B \cdot \Gamma \vdash \underline{n+1} : A}$$

**Substitutions**

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash a/ : A \cdot \Gamma} \qquad \frac{}{A \cdot \Gamma \vdash \uparrow : \Gamma} \qquad \frac{\Gamma \vdash s : \Delta}{A \cdot \Gamma \vdash \Uparrow(s) : A \cdot \Delta}$$

We define a typed *Eta* rule accordingly :

**Lemma 14 (Subject reduction theorem)** *Let $a \in \Lambda\Upsilon_\perp$. The rewrite extension on $\Lambda\Upsilon_\perp$ of $\lambda A.(a\ \underline{1}) \underset{Eta}{\rightarrow} a[\perp_A/]$, that is to say typed Eta, preserves types.*

**Proof:**

$$\frac{\dfrac{A \cdot \Gamma \vdash a : A \Rightarrow B \quad A \cdot \Gamma \vdash \underline{1} : A}{A \cdot \Gamma \vdash a\ \underline{1} : B}}{\Gamma \vdash \lambda A.(a\ \underline{1}) : A \Rightarrow B} \qquad \frac{A \cdot \Gamma \vdash a : A \Rightarrow B \quad \dfrac{\Gamma \vdash \perp_A : A}{\Gamma \vdash \perp_A/ : A \cdot \Gamma}}{\Gamma \vdash a[\perp_A/] : A \Rightarrow B}$$

□

So, having defined simply typed terms, it remains to show $\beta\eta'$ strong normalization on simply typed pure terms. We note the set of simply typed pure terms $\Lambda^{\rightarrow}_{\perp}$; it is a subset of $\Lambda\upsilon^{\rightarrow}_{\perp}$.

**Lemma 15** *Let $a \in \Lambda^{\rightarrow}_{\perp}$, $a$ is $\beta\eta'$-strongly normalizing.*

The adaptation of the $\beta\eta$ case [HS86] is quite straightforward. It relies on an easy "reverse" substitution lemma: let $M, L$ be pure terms and $x, y$ be variables. if $M\{y/x\} \underset{\beta\eta'}{\rightarrow} L$ then there is a pure term $N$ such that: $M \underset{\beta\eta'}{\rightarrow} N$ and $L \equiv N\{y/x\}$.

The conditions of the preservation theorem 3 are fulfilled , so :

**Corollary 2** *Let $a \in \Lambda\upsilon^{\rightarrow}_{\perp}$, $a$ is $\lambda\upsilon\eta$-strongly normalizing.*

# Conclusion

Hardin and Ríos definition[3] of $Eta$ [Har92, Río93] in the framework of $\lambda\upsilon$, is:

$$\lambda(a\ \underline{1}) \underset{Eta}{\longrightarrow} b \text{ if } \upsilon(a) = \upsilon(b[\uparrow]) \qquad (HR)$$

which raises few comments. That rule does not make the $\eta$-reduct computation explicit since it uses an $\upsilon$-matching [JK91] instead of our $\upsilon_\perp$-normalization. More precisely, imagine $Eta$ applied on a term $\lambda(a\ \underline{1})$. To apply rule $(HR)$, one must solve, modulo the theory $\upsilon$, the equation $a =_\upsilon b[\uparrow]$ where $b$, the $Eta$-reduct, is the unknown (the variable to instantiate). This computation is what we call $\upsilon$-matching. Clearly, $\upsilon$-matching is more complex than $\upsilon_\perp$-normalization: we do not even know whether $\upsilon$-matching is decidable or not and since $\upsilon$-matching may produce several solutions, rule $(HR)$ may produce several reducts for the same $Eta$-redex, among them the classical $\eta$-reduct. Consequently, rule $(HR)$ is less operational than our $Eta$ rule.

Moreover, our $Eta$ rule is generic. Indeed, in this report we apply our definition to $\lambda\upsilon$. But all we need in order to define $Eta$ is a term rewriting system that computes $\beta$-contraction; for instance we do not use renaming operators like $\uparrow$. Hence, $Eta$ can be adapted to every $\lambda$-calculus of explicit substitution, with explicit names or not. For instance, in $\lambda\sigma_{\Uparrow}$ [HL89], we would write:

$$\lambda(a\ \underline{1}) \underset{Eta}{\longrightarrow} a[\perp \cdot id]$$

and in $\lambda\chi$ [LRD94b]:

$$\lambda x_i.(a\ x_i) \underset{Eta}{\longrightarrow} a[\perp/x_i]_0$$

Last, our rule is unconditional. We have seen that this lead to a very general alternative of the classical $\eta$, namely $\eta'$ which does not require De Bruijn notation. The condition of application of the classical $\eta$ rule is too strong and as we have shown, there are other confluent reductions which are not captured by this rule. This examplifies our conviction that explicit substitutions help to a deeper understanding of $\lambda$-calculus, not only of its $\beta$-reduction aspect but also of other aspects like $\eta$-reduction. In that manner, according to Curry [CF58], explicit substitutions concur to precise the fundamentals of logic.

**Acknowledgments.** We would like to thank Pierre Lescanne and Jocelyne Rouyer-Degli for their constant support, and Philippe de Groote and Roberto Amadio for their remarks.

[3] To be consistent with our notations, we take Ríos syntax.

# References


[ACCL91] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, 1991.

[Bar84] H. P. Barendregt. *The Lambda-Calculus, its syntax and semantics.* Studies in Logic and the Foundation of Mathematics. Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1984. Second edition.

[BBLRD94] Z. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. Submitted, December 1994.

[CF58] H. B. Curry and Feys. *Combinatory Logic*, volume 1. Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1958.

[Cur83] P.-L. Curien. *Combinateurs catégoriques, algorithmes séquentiels et programmation applicative.* Thèse de Doctorat d'Etat, Université Paris 7, 1983.

[Cur86a] P.-L. Curien. Categorical combinators. *Information and Control*, 69:188–254, 1986.

[Cur86b] P.-L. Curien. *Categorical Combinators, Sequential Algorithms and Functional Programming.* Pitman, 1986.

[dB72] N. G. de Bruijn. Lambda calculus with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Proc. Koninkl. Nederl. Akademie van Wetenschappen*, 75(5):381–392, 1972.

[dB78] N. G. de Bruijn. A namefree lambda calculus with facilities for internal definition of expressions and segments. TH-Report 78-WSK-03, Technological University Eindhoven, Netherlands, Department of Mathematics, 1978.

[Har87] Th. Hardin. *Résultats de confluence pour les règles fortes de la logique combinatoire catégorique et liens avec les lambda-calculs.* Thèse de Doctorat d'Etat, Université Paris 7, 1987.

[Har89] Th. Hardin. Confluence results for the pure strong categorical combinatory logic CCL: $\lambda$-calculi as subsystems of CCL. *Theoretical Computer Science*, 65:291–342, 1989.

[Har92] Th. Hardin. Eta-conversion for the languages of explicit substitutions. In *3rd ALP, LNCS 632*, Volterra, Italy, 1992.

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

[HS86] J. Roger Hindley and Johnathan P. Seldin. *Introduction to Combinators and Lambda-calculus.* Cambridge University, 1986.

[JK91] J.-P. Jouannaud and Claude Kirchner. Solving equations in abstract algebras: a rule-based survey of unification. In Jean-Louis Lassez and G. Plotkin, editors, *Computational Logic. Essays in honor of Alan Robinson*, chapter 8, pages 257–321. The MIT press, Cambridge (MA, USA), 1991.

[Les94] P. Lescanne. From $\lambda\sigma$ to $\lambda\upsilon$, a journey through calculi of explicit substitutions. In Hans Boehm, editor, *Proceedings of the 21st Annual ACM Symposium on Principles Of Programming Languages, Portland (Or., USA)*, pages 60–69. ACM, 1994.

[LRD94a] P. Lescanne and J. Rouyer-Degli. The calculus of explicit substitutions $\lambda\upsilon$. Technical Report RR-2222, INRIA-Lorraine, January 1994.

[LRD94b] P. Lescanne and J. Rouyer-Degli. Explicit substitutions with de Bruijn's levels, August 1994.

[Mel95] P.-A. Melliès. Typed $\lambda$-calculi with explicit substitutions may not terminate. In M. Dezani, editor, *Int. Conf. on Typed Lambda Calculus and Applications*, 1995.

[Río93] A. Ríos. *Contributions à l'étude des $\lambda$-calculs avec des substitutions explicites.* Thèse de Doctorat d'Université, U. Paris VII, 1993.

# A Quotation from Curry and Feys

We quote Curry and Feys [CF58] : 'Combinatory logic is a branch of mathematical logic which concerns itself with the ultimate foundations. Its purpose is the analysis of certain notions of such basic character that they are ordinarily taken for granted. These include the processes of substitution usually indicated by the use of variables; and also the classification of entities constructed by these processes into types or categories, which in many systems has to be done intuitively before the theory can be applied. It has been observed that these notions, although generally presupposed are not simple; they constitute a prelogic, so to speak, whose analysis is by no means trivial. [...] It is the synthetic theory[4] which gives the ultimate analysis of substitution in terms of a system of extreme simplicity. The theory of lambda-conversion is intermediate in character between the synthetic theory and ordinary logics. Although its analysis is in some ways less profound - many of the complexities in regard to variables are still unanalysed there - yet it is none the less significant; and it has the advantage of departing less radically from our intuitions.'

# B Properties of applied $\lambda\upsilon$-calculus

We add a finite set of constants to $\Lambda\upsilon$, forming $\Lambda\upsilon_c$, and their associated constant rules $c[s] \to c$ to $\upsilon$, then

**Theorem 4 (Convergence of $\upsilon_c$)**

1. *$\upsilon_c$ terminates.*
2. *$\upsilon_c$ is orthogonal, therefore confluent.*

**Remark 1 (well-foundedness of $(\underset{\upsilon_c}{\longrightarrow} \cup \sqsupset)^+$)** *The termination of $\upsilon_c$ is proved with a simplification ordering, $>$. So it contains $\sqsupset$ and $\underset{\upsilon_c}{\longrightarrow}$, and therefore the transitive closure of their union.*

**Remark 2 (application and abstraction structure is preserved)** *As no left hand side of $\upsilon_c$ rules is an application or an abstraction, for $a, b \in \Lambda_c$,*

1. $\upsilon_c(ab) = \upsilon_c(a)\upsilon_c(b)$
2. $\upsilon_c(\lambda a) = \lambda\upsilon_c(a)$

**Remark 3 (Church-Rosser consequence)** *As $\upsilon_c$ is Church-Rosser, for $a, s \in \Lambda_c$, $\upsilon_c(a[s]) = \upsilon_c(\upsilon_c(a)[\upsilon_c(s)])$*

**Lemma 16** *For $n \geq 1$ and $i \geq 0$, $\underline{n}\,[\Uparrow^{n+i}(s)] \underset{\upsilon_c}{\overset{*}{\longrightarrow}} \underline{n}$.*

**Lemma 17** *For $n \geq 1$ and $i \geq 0$, $\underline{n+i}\,[\Uparrow^i(s)] \underset{\upsilon_c}{\overset{*}{\longrightarrow}} \underline{n}[s][\uparrow^i]$.*

**Corollary 3 (Barendregt style Substitution Lemma)** $a[b/][s] \underset{\upsilon_c}{\overset{*}{\longleftrightarrow}} a[\Uparrow(s)][b[s]/]$.

**Theorem 5**

1. *the projection lemma for Beta holds: a Beta-rewrite between terms of $\Lambda\upsilon_c$ translates in a $\beta$-reduction between their $\upsilon_\perp$-normal forms.*
2. *Beta is confluent on $\Lambda\upsilon_c$, by previous item.*

# C *Eta* projection lemma on $\Lambda_\perp$

In the next lemma, we prove that an external (in the term part) *Eta*-rewrite corresponds to exactly one $\eta'$-reduction. In particular, an external *Eta*-redex structure does not tamper with $\upsilon_\perp$-normalization. (basic case : 4 b)

**Lemma 18 (External Projection Lemma)** *Let $a, b \in \Lambda\upsilon_\perp$,*
*if $a \underset{Eta}{\longrightarrow}^{ext} b$ then $\upsilon_\perp(a) \underset{\eta'}{\overset{+}{\longrightarrow}} \upsilon_\perp(b)$.*

[4] what we now call combinatory logic

**Proof:** By noetherian induction with the well-founded order $\underset{v_\perp}{\longrightarrow}$.

1. $a = \lambda(c\ \underline{1})$ : by Remark 2 and Remark 3, $v_\perp(a) = v_\perp(\lambda(c\ \underline{1})) = \lambda(v_\perp(a)\ \underline{1}) \underset{\eta'}{\longrightarrow} v_\perp(v_\perp(a)[\perp/]) = v_\perp(a[\perp/]) = v_\perp(b)$
2. $a = \lambda c \underset{Eta}{\longrightarrow}^{ext} \lambda c'$
   As $c \underset{Eta}{\longrightarrow}^{ext} c'$, by induction hypothesis : $v_\perp(c) \underset{\eta'}{\longrightarrow} v_\perp(c')$.
   By compatibility of $\eta'$ : $\lambda v_\perp(c) \underset{\eta'}{\longrightarrow} \lambda v_\perp(c')$.
   And $v_\perp$-normalisation left unchanged the abstraction structure, so : $v_\perp(\lambda c) \underset{\eta'}{\longrightarrow} v_\perp(\lambda c')$.
3. $a = cd$ : as previous case.
4. $a = c[s] \underset{Eta}{\longrightarrow}^{ext} c'[s]$. We can not apply the same reasoning, because $\eta$ is not defined on $\Lambda v_\perp$, so we have not the compatibility step. We must look at the structure of c :
   (a) $a = de[s] \underset{Eta}{\longrightarrow}^{ext} d'e[s] = b$. We have :

   $$a = de[s] \underset{v}{\longrightarrow} d[s]e[s]$$

   By compatility of $Eta$ on $\Lambda v_\perp$,

   $$d[s]e[s] \underset{Eta}{\longrightarrow} d'[s]e[s]$$

   By induction hypothesis,

   $$v_\perp(d[s]e[s]) \underset{\eta'}{\longrightarrow} v_\perp(d'[s]e[s])$$

   And $v_\perp$-normalisation left unchanged the application structure, so :

   $$v_\perp(de[s]) \underset{\eta'}{\longrightarrow} v_\perp(d'e[s])$$

   $$v_\perp(a) \underset{\eta'}{\longrightarrow} v_\perp(b)$$

   (b) $a = \lambda(d\ \underline{1})[s] \underset{Eta}{\longrightarrow} e[s] = b$

   $$\begin{array}{rcl} v_\perp(a) = v_\perp((\lambda(d\ \underline{1}))[s]) & = & v_\perp(\lambda(d[\Uparrow(s)]\ \underline{1}[\Uparrow(s)]) \\ & = & \lambda(v_\perp(d[\Uparrow(s)])\ \underline{1}) \\ & \underset{\eta'}{\longrightarrow} & v_\perp(v_\perp(d[\Uparrow(s)])[\perp/]) \\ & = & v_\perp(d[\Uparrow(s)][\perp/]) \\ & = & v_\perp(d[\Uparrow(s)][\perp[s]/]) \\ & = & v_\perp(d[\perp/][s]) = v_\perp(b) \end{array}$$

   (c) $a = (\lambda\ d)[s] \underset{Eta}{\longrightarrow} (\lambda\ e)[s] = b$ and $d \underset{Eta}{\longrightarrow} e$
   We have : $(\lambda\ d)[s] \underset{v_\perp}{\longrightarrow} \lambda(d[\Uparrow(s)])$ so $(\lambda\ d)[s] >_{(\underset{v_\perp}{\longrightarrow}\sqsupset)} \lambda(d[\Uparrow(s)])$ By compatibility of $Eta$,

   $$\lambda(d[\Uparrow(s)]) \underset{Eta}{\longrightarrow} \lambda(e[\Uparrow(s)])$$

   By induction hypothesis, $v_\perp(\lambda(d[\Uparrow(s)]) \underset{\eta'}{\longrightarrow} v_\perp(\lambda(e[\Uparrow(s)]))$

   $$v_\perp((\lambda\ d)[s])) \underset{\eta'}{\longrightarrow} v_\perp((\lambda\ e)[s])$$

   $$v_\perp(a) \underset{\eta'}{\longrightarrow} v_\perp(b)$$

   (d) $a = d[t][s] \underset{Eta}{\longrightarrow}^{ext} e[t][s] = b$ and $d \underset{Eta}{\longrightarrow} e$.
   $d[t]$ is a subterm of $d[t][s]$, so we can apply the induction hypothesis on it :

   $$v_\perp(d[t]) \underset{\eta'}{\longrightarrow} v_\perp(e[t])$$

   By definition of $\eta'$ there exists $g$ such that :

   $$v_\perp(d[t]) \underset{Eta}{\longrightarrow} g \underset{v_\perp}{\longrightarrow} v_\perp(e[t])$$

By compatibility of $Eta$ on $\Lambda_\perp$ :

$$v_\perp(d[t])[s] \underset{Eta}{\longrightarrow} g[s] \underset{v_\perp}{\longrightarrow} v_\perp(e[t])[s]$$

As a closure is not in $v_\perp$-nf, $v_\perp(d[t])[s] <_{(\underset{v_\perp}{\longrightarrow} \sqsupset)} d[t][s]$, so we can apply the induction hypothesis :

$$v_\perp(v_\perp(d[t])[s]) \underset{\eta'}{\longrightarrow} v_\perp(v_\perp(e[t])[s])$$

And by convergence of $v_\perp$ :

$$v_\perp(d[t][s]) \underset{\eta'}{\longrightarrow} v_\perp(e[t][s])$$

$$v_\perp(a) \underset{\eta'}{\longrightarrow} v_\perp(b)$$

□

In the following lemma, we prove that an internal $Eta$-rewrite corresponds to zero, one or several $\eta$-contractions. We point out where the $Eta$-redexes are eliminated or duplicated.

**Lemma 19** *Let $a,b \in \Lambda v_\perp$, if $a \underset{Eta}{\longrightarrow}^{int} b$ then $v_\perp(a) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(b)$.*

**Proof:** By noetherian induction with the well-founded order $(\underset{v_\perp}{\longrightarrow} \sqsupset)$.

1. $a = \lambda c \underset{Eta}{\longrightarrow}^{int} \lambda c = b$. By induction hypothesis on $c$ and compatibility of $\eta$ on $\Lambda$.
2. $a = cd$. Idem.
3. $a = c[s]$
   (a) $c[s] \underset{Eta}{\longrightarrow}^{int} c[t]$. By next 'lemma'.
   (b) $c[s] \underset{Eta}{\longrightarrow}^{int} d[s]$.
   As $c \underset{Eta}{\longrightarrow}^{int} d$, by induction hypothesis :

$$v_\perp(c) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(d)$$

By correction, $Eta$ simulates $\eta'$ on pure terms,

$$v_\perp(c) \underset{Eta}{\longrightarrow}^{ext} c' \underset{v_\perp}{\longrightarrow} v_\perp(c') \dots v_\perp(d)$$

The $Eta$-rewrite are external because pure terms contain no closure. By compatibility of $Eta$ and $v_\perp$ :

$$v_\perp(c)[s] \underset{Eta}{\longrightarrow}^{ext} c'[s] \underset{v_\perp}{\longrightarrow} v_\perp(c')[s] \dots v_\perp(d)[s]$$

By zero or more applications of the external projection lemma,

$$v_\perp(v_\perp(c)[s]) \underset{\eta'}{\longrightarrow} v_\perp(c'[s]) = v_\perp(v_\perp(c')[s]) \dots v_\perp(v_\perp(d)[s])$$

By remark 3,

$$v_\perp(c[s]) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(d[s])$$

□

We extract a particular case of the previous proof :

**Lemma 20** *Let $a,b \in \Lambda v_\perp$,*
*if $a[s] \underset{Eta}{\longrightarrow}^{int} a[t]$ then $v_\perp(a) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(b)$.*

**Proof:** We suppose : $s \underset{Eta}{\longrightarrow} t$

1. $a = bc$ and $(bc)[s] \underset{Eta}{\longrightarrow} (bc)[t]$

$$bc[s] \underset{v_\perp}{\longrightarrow} b[s]c[s] \sqsupset c[s] \text{ so } bc[s] >_{(\underset{v_\perp}{\longrightarrow} \sqsupset)} b[s]$$

As
$$b[s] \underset{Eta}{\longrightarrow} b[t]$$
by induction hypothesis :
$$v_\perp(b[s]) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(b[t])$$
Similarly,
$$v_\perp(c[s]) \underset{\eta}{\overset{*}{\longrightarrow}} v_\perp(c[t])$$
By compatibility of $\eta$,
$$v_\perp(b[s])v_\perp(c[s]) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(b[t])v_\perp(c[s])$$
$$v_\perp(b[t])v_\perp(c[s]) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(b[t])v_\perp(c[t])$$
By transitivity of $\eta$,
$$v_\perp(b[s])v_\perp(c[s]) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(b[t])v_\perp(c[t])$$
By remark 2,
$$v_\perp(bc[s]) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(bc[t])$$
Here, the $Eta$-redex is duplicated. (We use twice the induction hypothesis.)

2. $a = \lambda c$ and $(\lambda c)[s] \underset{Eta}{\longrightarrow} (\lambda c)[t]$

$$(\lambda c)[s] \underset{v_\perp}{\longrightarrow} \lambda(c[\Uparrow (s)]) \text{ so } (\lambda c)[s] >_{(\underset{v_\perp}{\longrightarrow} \sqsupset)} \lambda(c[\Uparrow (s)])$$

By compatibility of $Eta$ on $\Lambda v_\perp$ :
$$\lambda(c[\Uparrow (s)]) \underset{Eta}{\longrightarrow} \lambda(c[\Uparrow (t)])$$
By induction hypothesis on $\lambda(c[\Uparrow (s)])$,
$$v_\perp(\lambda(c[\Uparrow (s)])) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(\lambda(c[\Uparrow (t)])$$
By remark 2,
$$v_\perp((\lambda c)[s]) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp((\lambda c)[t])$$

3. $a = c[u]$ and $c[u][s] \underset{Eta}{\longrightarrow} c[u][t]$

$$c[u][s] \underset{v_\perp}{\overset{+}{\longrightarrow}} v_\perp(c[u])[s] \text{ so } c[u][s] >_{(\underset{v_\perp}{\longrightarrow} \sqsupset)} v_\perp(c[u])[s]$$

By compatibility of $Eta$ on $\Lambda v_\perp$ :
$$v_\perp(c[u])[s] \underset{Eta}{\longrightarrow} v_\perp(c[u])[t]$$
By induction hypothesis on $v_\perp(c[u])[s]$,
$$v_\perp(v_\perp(c[u])[s]) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(v_\perp(c[u])[t])$$
By remark 3,
$$v_\perp(c[u][s]) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(c[u][t])$$

4. $a = m$ and $\underline{n}\,[s] \underset{Eta}{\longrightarrow} \underline{m}\,[t]$. The form of s is necessarily : $\Uparrow^i(c/)$. So, we suppose :$\underline{n}\,[\Uparrow^i(c/)] \underset{Eta}{\longrightarrow} \underline{n}\,[\Uparrow^i(d/)]$ with $c \underset{Eta}{\longrightarrow} d$.

(a) $m = \underline{1}, i = 0$ and $\underline{1}\,[c/] \underset{Eta}{\longrightarrow} \underline{1}\,[d/]$

$$\underline{1}\,[c/] \underset{v_\perp}{\longrightarrow} \underline{1} \text{ so } \underline{1}\,[c/] >_{(\underset{v_\perp}{\longrightarrow}\sqsupset)} \underline{1}$$

i. $c \underset{Eta}{\longrightarrow}^{int} d$. By induction hypothesis :

$$v_\perp(c) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(d)$$

ii. $c \underset{Eta}{\longrightarrow}^{ext} d$. By external projection lemma:

$$v_\perp(c) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(d)$$

As $v_\perp(\underline{1}\,[c/]) = v_\perp(c)$,

$$v_\perp(\underline{1}\,[c/]) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(\underline{1}\,[d/])$$

$$v_\perp(\underline{m}\,[s]) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(\underline{m}\,[t])$$

Notice that we need the external projection lemma to prove the internal one (but not conversely).

(b) $m = n+1, i = 0$ and $\underline{n+1}\,[c/] \underset{Eta}{\longrightarrow} \underline{n+1}\,[d/]$

As, $v_\perp(\underline{n+1}\,[c/]) = \underline{n} = v_\perp(\underline{n+1}\,[d/])$, trivially:

$$v_\perp(\underline{n+1}\,[c/]) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(\underline{n+1}\,[d/])$$

$$v_\perp(\underline{m}\,[s]) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(\underline{m}\,[t])$$

Here, we see that a $Eta$-redex is eliminated by the rule $RVar$

(c) $m = 1, i = j+1$ and $1[\Uparrow(\Uparrow^j(c/))] \underset{Eta}{\longrightarrow} 1[\Uparrow(\Uparrow^j(d/))]$

As, $v_\perp(1[\Uparrow(\Uparrow^j(c/))]) = 1 = v_\perp(1[\Uparrow(\Uparrow^j(d/))])$, trivially:

$$v_\perp(1[\Uparrow(\Uparrow^j(d/))]) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(1[\Uparrow(\Uparrow^j(d/))])$$

$$v_\perp(\underline{m}\,[s]) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(\underline{m}\,[t])$$

Here, we see that a $Eta$-redex is eliminated by the rule $FVarLift$

(d) $m = n+1, i = j+1$ and $\underline{n+1}\,[\Uparrow(\Uparrow^j(c/))] \underset{Eta}{\longrightarrow} \underline{n+1}\,[\Uparrow(\Uparrow^j(d/))]$

By compatibility of $Eta$ on $\Lambda v_\perp$,

$$\underline{n}\,[\Uparrow^j(c/)][\uparrow] \underset{Eta}{\longrightarrow}^{int} \underline{n}\,[\Uparrow^j(d/)][\uparrow]$$

$$\underline{n+1}\,[\Uparrow(\Uparrow^j(c/))] \underset{v_\perp}{\longrightarrow} \underline{n}\,[\Uparrow^j(c/)][\uparrow] \text{ so } \underline{n+1}\,[\Uparrow(\Uparrow^j(c/))] >_{(\underset{v_\perp}{\longrightarrow}\sqsupset)} \underline{n}\,[\Uparrow^j(c/)][\uparrow]$$

By induction hypothesis :

$$v_\perp(\underline{n}\,[\Uparrow^j(c/)][\uparrow]) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(\underline{n}\,[\Uparrow^j(d/)][\uparrow])$$

As $v_\perp(\underline{n}\,[\Uparrow^j(c/)][\uparrow]) = v_\perp(\underline{n+1}\,[\Uparrow(\Uparrow^j(c/))])$,

$$v_\perp(\underline{n+1}\,[\Uparrow(\Uparrow^j(c/))]) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(\underline{n+1}\,[\Uparrow(\Uparrow^j(d/))])$$

$$v_\perp(\underline{m}\,[s]) \underset{\eta'}{\overset{*}{\longrightarrow}} v_\perp(\underline{m}\,[t])$$

□

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399