The Origins of lambda-calculus and term rewriting systems

Yves Bertot

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél.: (1) 39 63 55 11

Rapports de Recherche

N° 1543

Programme 2
Calcul Symbolique, Programmation
et Génie logiciel

THE ORIGINS OF λ-CALCULUS AND TERM REWRITING SYSTEMS

Yves BERTOT

Octobre 1991

RR_1543

The Origins of λ-calculus and Term Rewriting Systems

Yves Bertot
INRIA Sophia-Antipolis
e-mail: bertot@mirsa.inria.fr

Abstract

We use *Origin functions* to describe the notions of descendance and residuals in reduction systems such as the λ-calculus and linear term rewriting systems. We compare the origin functions for the λ-calculus and for term rewriting systems that implement this calculus, $\lambda\sigma$ and λEnv. We show that the notions of origin do not correspond exactly, but we describe an extension of the notion of origin that permits the correct computation of λ-calculus origins for derivations of λEnv. We show that this extension is not sufficient to give the same result for $\lambda\sigma$ and we give another extension for that system. This work is interesting as it provides with a distinction between the two term rewriting systems. This work has applications in the debugging of languages based on the λ-calculus or environments.

Keywords: λ-calculus, Term Rewriting Systems, Residuals, Descendance, Origins.

Les Origines du λ-calcul et des Systèmes de Réécriture

Résumé

Nous utilisons des *fonctions d'origine* pour décrire les notions de descendance et de résidu dans des systèmes de réduction tels que le λ-calcul et les systèmes de réécriture linéaires. Nous comparons les fonctions d'origine pour le λ-calcul et pour des systèmes de réécriture qui le représentent, $\lambda\sigma$ et λEnv. Nous montrons que les origines ne se correspondent pas exactement, mais nous décrivons une extension de la notion d'origine qui permet de calculer correctement les origines du λ-calcul pour les derivations de λEnv. Nous montrons que cette extension n'est pas suffisante pour le système $\lambda\sigma$. Nous donnons néanmoins une extension supplémentaire pour résoudre ce problème pour ce système. Ce travail a des applications dans la mise au point des langages basés sur le λ-calcul et les environnements.

Mots-clés : λ-calcul, Systèmes de Réécriture, Résidus, Descendance, Origines.

The Origins of λ-calculus and Term Rewriting Systems

Yves Bertot
INRIA Sophia-Antipolis
e-mail: bertot@mirsa.inria.fr

We use *Origin functions* to describe the notions of descendance and residuals in reduction systems such as the λ-calculus and linear term rewriting systems. We compare the origin functions for the λ-calculus and for term rewriting systems that implement this calculus, $\lambda\sigma$ and λEnv. We show that the notions of origin do not correspond exactly, but we describe an extension of the notion of origin that permits the correct computation of λ-calculus origins for derivations of λEnv. We show that this extension is not sufficient to give the same result for $\lambda\sigma$ and we give another extension for that system. This work is interesting as it provides a distinction between the two term rewriting systems. This work has applications in the debugging of languages based on the λ-calculus or environments.

1. Introduction.

1.1. *Motivation and Related Work.*

The main motivation of this work is source-level debugging of programming languages.

With modern compilers, this is usually achieved by providing a debug option. The produced code is then modified to contain extra instructions and pointers to the source program, for use by a symbolic debugger. This debug option is most often incompatible with code optimization. This problem can be turned around as in [Tolmach&Appel90], where one proposes to implement debugging by source instrumentation. This permits to avoid turning off optimization, but the resulting code is not as optimized, since debugging actions are interspersed in it. With the concept of *origin* functions, we take a different point of view: we avoid interfering with the optimization process, but we observe this process to compute a "map" (the origin function) of the resulting code. We focus our attention on term rewriting systems because optimization is often implemented by rewriting. In this field, origin functions are closely related to the well-known notions of descendance and residuals.

The descendance relation or residual relation associated to a reduction system is usually studied using labeled or marked systems, as in [Morris68], [Lévy78], [Klop80], or [Field90]. We rule out this solution, since it means coming back to modifying the produced code. We prefer to use an origin function that is considered separately from the object code. This has another advantage: whereas marks and labels give an extensional description of the origin function, we can use intensional representations, with possible gains in memory space for the debugger. Examples of such intensional representations are studied in [Bertot91a]. We do not totally reject labeled rewriting systems, which we consider to be good tools for the theoretical study of computation, but we propose origin functions simply as an efficient alternative for the practical implementation of these notions.

In this paper, we are interested in the representation of the λ-calculus by rewriting systems, as this can be useful for studying implementations of functional programming languages. That a

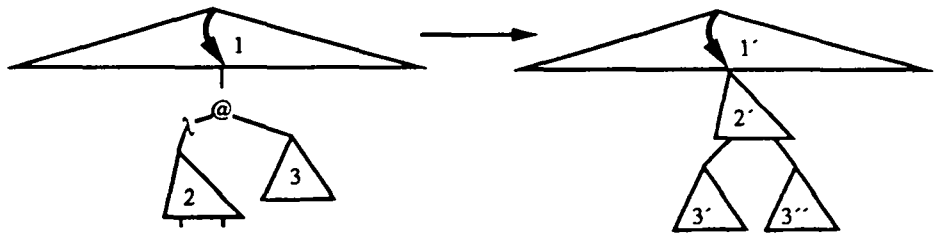


Figure 1. A schematic β-reduction.
(*The stubs below area 2 represent occurrences of the substituted variable*)

rewriting system implements the λ-calculus means that for any λ-calculus derivation we can find a rewriting system derivation that performs the same computations. We want to see if the two derivations also give the same notion of descendance. In particular, we show that the two rewriting systems λEnv and $\lambda\sigma$ which we study do not represent the λ-calculus as faithfully. This work is closely related to previous work of Hardin and Lévy [Hardin&Lévy89] and Curien, Hardin and Lévy [CHL91]. The proofs of correction we provide owe very much to [Hardin&Lévy89] and [Field90]. In particular, our system $\lambda\sigma^L$ is directly inspired from Field's ΛCCL^L.

1.2. *Origins in the λ-calculus and Term Rewriting Systems.*

We consider the $\lambda\beta$-calculus, in its de Bruijn representation [de Bruijn72]. It is based on the following rule, also called the β-rule.

$$(\lambda M)N \rightarrow M[1\backslash N]$$

In this representation variable names are replaced by indices which represent the distance between the variables and their binders. The substitution mechanism represented by the notation $M[1\backslash N]$ requires non-trivial manipulation of these indices, since the distance between a variable and its binder may change in the process. A schematic β-reduction is represented in figure 1.

For the descendance relation, it is obvious that a node at any position in area $1'$ descends from the node at the same position in area 1, whereas a node in area $2'$ descends from the node at the same position in area 2 (note that the position in the entire term changes, since area 2 moves during the β-reduction), and the nodes that are at a same position relative to areas $3'$ and $3''$ descend from the node that at the same position in area 3. Because substitution requires some updating of indices, it may happen that a variable occurrence descends from a variable occurrence that does not have the same index.

We remark that the relation of descendance is simply a relation between occurrences in the final term and occurrences in the initial term. We also notice that the converse of this descendance relation is a function. We call this function an *origin function.*

For multi-step derivations, the origin functions are quite simple to understand. The origin function for an empty derivation is the identity function and the origin of the concatenation of two derivations D and D' is the composition of the two origin functions $f \circ f'$. This simply means that if position u comes from a node position v by the derivation D' (expressed by $v = f'(u)$) and v comes from w by D (expressed by $w = f(v)$) then u comes from w by the concatenated derivation $D; D'$ (expressed by $w = f(f'(u)) = f \circ f'(u)$).

The same notion of descendance also appears in linear term rewriting systems. Such systems are sets of couples of patterns $\alpha \rightarrow \beta$, with the constraint that every variable appearing in β also appears in α and no variable appears more than once in α. Computing with such rewriting systems

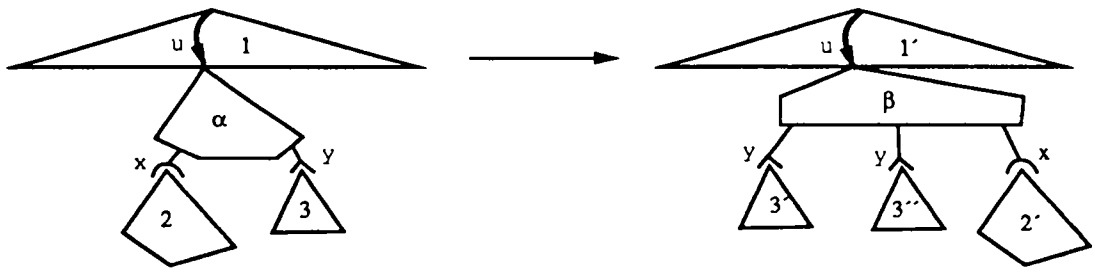


Figure 2. A schematic term rewriting.

means replacing instances of the pattern α by the corresponding instances of the pattern β. A schematic reduction is represented in figure 2.

As far as origin of nodes are concerned, area $1'$ has its origins in area 1 as for the λ-calculus, area $2'$ has its origins in area 2, areas $3'$ and $3''$ have their origins in area 3. The choice of origin for areas $2'$, $3'$, and $3''$ is done using the variables in the patterns α and β: the area under a given variable has its origins in the area that is under the same variable in α. This definition is not ambiguous, since we are dealing with linear systems. For the nodes in the pattern β they do not have any origin; they are "created" by the reduction. Conversely, the nodes inside the pattern α are consumed by the rewriting and do not have any descendants.

1.3. *Relations between Origin Functions.*

Since the discovery of λ-calculus, various attempts have been made to represent it using term rewriting systems, mostly with combinators [Curry&Feys58], [Hindley&Seldin86], [Curien86]. These attempts were all incomplete in the sense that they only presented a part of λ-calculus computations, mostly known as weak computations. Only lately, rewriting systems have been proposed that exactly describe the λ-calculus. These rewriting systems are $\lambda\sigma$ [ACCL90] and λEnv [Hardin&Lévy89], [CHL91].

The main idea in these rewriting systems is to add operators to the λ-calculus to represent "substitution in the making". These operators permit to delay and compose substitution operations, stacking them in environments. As such, these rewriting systems provide a nice tool to study in a common setting both the λ-calculus and its implementation on environment machines (see, for example [Field90], [Maranget91]).

Since we are interested in origins, and this notion exists for both the λ-calculus and term rewriting systems, it is interesting to compare this notion in both settings. More precisely, if we consider a derivation of the λ-calculus and a derivation of one of these term rewriting systems that performs the same computations, do these derivations have the same origin function? The answer is negative and we can rapidly give a counterexample, based on the fact that both $\lambda\sigma$ and λEnv contain the following three rules:

Beta	$(\lambda M)N \to M[N \cdot id]$
App	$(MN)[s] \to M[s]N[s]$
FVar	$1[M \cdot s] \to M$

Counterexample 1. *The origin of the following two derivations differ, if D_1 is a λ-calculus derivation and D_2 is a $\lambda\sigma$ or λEnv derivation (a is any λ-term).*

D_1 : $\quad (\lambda(1\ 1))\ a \xrightarrow{\varepsilon} a\ a$

D_2 :

$$(\lambda(1\ 1))\ a \overset{(\epsilon,Beta)}{\rightarrow} (1\ 1)[a \cdot id]$$
$$\overset{(\epsilon,App)}{\rightarrow} (1[a \cdot id]\ 1[a \cdot id])$$
$$\overset{(1,Fvar)}{\rightarrow} a\ 1[a \cdot id]$$
$$\overset{(2,Fvar)}{\rightarrow} a\ a$$

Let f_1 be the origin function for D_1 and f_2 be the one for D_2. From the informal definition of origins given above, we infer $f_1(\epsilon) = 1.1$ (the empty occurrence, ϵ, designates the top of a term, whereas 1.1 designates the first child of the first child) and $f_2(\epsilon)$ is not defined, because the application node at the top of the resulting term was created during the second rewriting (rule App).

The analysis of this counterexample shows that the application operator in the resulting term is created by the second reduction of D_2, whereas the application operator that appears inside the λ-abstraction in the initial term is consumed by the same reduction. This leads to the following suggestion: why not extend origin functions for term rewritings so that created nodes (inside β) descend from consumed nodes (inside α)? We call such extensions of origin functions *marking functions.*

The main result of this paper is that these extensions are sufficient to represent λ-calculus origins for λEnv, but not for $\lambda\sigma$. The main difference between the two systems lies in the treatment of substitution inside a λ-abstraction. Both introduce an extra operator to express that the variables in the substituent will need some updating, but only λEnv introduces an extra operator to express that the variables bound by the λ-abstraction must be protected from the substitution. The system λEnv contains the following two rules:

Lambda $\quad (\lambda M)[s] \rightarrow \lambda(M[\Uparrow(s)])$

FVarLift1 $\quad 1[\Uparrow(s)] \rightarrow 1$

Whereas the system $\lambda\sigma$ gets away with the following trick, using substitution (rule $FVar$) to recover the correct value for the bound variable:

Lambda' $\quad (\lambda M)[s] \rightarrow \lambda(M[1 \cdot (s \circ \uparrow)])$

Thus, in $\lambda\sigma$, the bound variable is protected because it receives the same value when the substitution is applied. However, the origins are all mixed up, since a substitution actually took place. The problem of computing correctly λ-calculus origins can still be solved, with more complicated extension of origin functions, which we describe in section 4.

2. Definitions.

2.1. *The λ-calculus.*

We denote $\mathcal{N}_+$ the set of strictly positive integers. The set Λ of λ-terms is the set defined by the following syntax:

$$\Lambda ::= \lambda\Lambda \mid \Lambda\Lambda \mid n$$

where n is an element of $\mathcal{N}_+$. We note @ the head operator of a term of the form MN and λ the head operator of a term of the form λM. Thus, Λ is the set of terms built from the operators of $Op_\lambda = \{@, \lambda\} \cup \mathcal{N}_+$.

We use occurrences to denote positions in terms. The set of occurrences $\mathcal{O}$ is the set of words of $\mathcal{N}_+$. We denote the concatenation of occurrences u and v as $u.v$ and the empty occurrence as ε. We note $u \leq v$ whenever there exists an occurrence w such that $v = u.w$ (we note $u < v$ if $u \leq v$ and $u \neq v$) and we denote $|u|$ the length of the occurrence u. The notation M/u denotes the subterm of M at occurrence u, $op(u, M)$ denotes its head operator, and $M[u \leftarrow N]$ denotes M where the subterm at occurrence u is replaced by N. For any term T, the set of occurrences u such that T/u is defined is called its *domain* and is noted $\mathcal{D}(T)$. We also denote $depth(u, M)$ the number $card\{v \mid v < u \quad op(v, M) = \lambda\}$, that is the number of λ operators in M on the path represented by the occurrence u.

Examples. $((MN)P)/1.2 = N$, $op(\varepsilon, (\lambda M)N) = @$, $op(1, (\lambda M)N) = \lambda$, $depth(1.1.2, (\lambda(1\,2)N)) = 1$.

We note $T \xrightarrow{u} T'$ the β-reduction (or elementary λ-derivation) of T at occurrence u. This means that there exist two terms M and N such that $T/u = (\lambda M)N$ and $T' = T[u \leftarrow M[1\backslash N]]$. We denote

$$T \overset{u_1;\cdots;u_n}{\twoheadrightarrow} T' = T \xrightarrow{u_1} \cdots \xrightarrow{u_n} T'$$

and more generally we denote $T \overset{S}{\twoheadrightarrow} T'$ any derivation of the λ-calculus, where S is a list of occurrences. We use ";" to represent the concatenation of lists of occurrences, and we denote the empty list as $\emptyset$. If D and D' are the two derivations $D = T \overset{S}{\twoheadrightarrow} T'$ and $D' = T' \overset{S'}{\twoheadrightarrow} T''$, we note $D; D'$ the derivation $D; D' = T \overset{S;S'}{\twoheadrightarrow} T'$, thus overloading the operator ";". Also, we denote $u.S$ the list of occurrences $u.u_1; \cdots; u.u_n$ when S is the list of occurrences $u_1; \cdots; u_n$.

2.2. *The system λEnv.*

Linear term rewriting systems have already been presented earlier. We denote $T \overset{(u,r)}{\rightarrow} T'$ the rewriting of T at occurrence u using rule $r = \alpha \rightarrow \beta$ to the term T'. This means that there exists a substitution σ such that $T/u = \sigma(\alpha)$ and $T' = T[u \leftarrow \sigma(\beta)]$. We note $T \overset{S}{\twoheadrightarrow} T'$ a derivation of the term rewriting system, where S is a list of couples of the form (u, r) in which u is an occurrence and r is a rule. As for the λ-calculus, we note $D; D'$ the concatenation of two derivations and $\emptyset$ the empty derivation.

The rewriting system λEnv works with the language ΛEnv defined by the following syntax:

$$M ::= \lambda M \mid MM \mid n \mid M[s]$$
$$s ::= s \circ s \mid M \cdot s \mid id \mid \Uparrow(s) \mid \uparrow$$

Thus, ΛEnv is obtained from Λ by adding new operators $[]$, $\circ$, $\cdot$, $\Uparrow$, id, and $\uparrow$. The term rewriting system contains 24 rules that are given in the appendix. One of these rules, *Beta*, is used to represent the β-rule:

Beta $\qquad (\lambda M)N \rightarrow M[N \cdot id]$

All the other rules describe how to compute substitutions. We note $\mathcal{S}Env$ the system obtained by removing rule *Beta* from λEnv.

2.3. *Origin Functions.*

The origin functions for the λ-calculus are given by an origin mapping Ω that maps λ-calculus derivations to functions over occurrences. This mapping is defined by the following properties:

1. For any derivation $D = T \xrightarrow{S} T'$, one has $dom(\Omega(D)) = \mathcal{D}(T')$ and $codom(\Omega(D)) = \mathcal{D}(T)$.
2. For any derivations D and D', one has $\Omega(D; D') = \Omega(D) \circ \Omega(D')$. For any derivation D of the form $D = T \xrightarrow{\emptyset} T$, one has $\Omega(D) = 1_{\mathcal{D}(T)}$.
3. For any derivation $D = T \xrightarrow{S} T'$ such that $\Omega(D) = \omega$ and for any term M such that $M/u = T$, the function $\omega' = \Omega(D')$ where D' is the embedded derivation $D' = M \xrightarrow{u.S} M[u \leftarrow T']$ verifies the following properties:
   a. For any occurrence v such that $u \not\leq v$, one has $\omega'(v) = v$,
   b. For any occurrence $v \in \mathcal{D}(T')$, one has $\omega'(u.v) = u.\omega(v)$.
4. For any elementary derivation $D = (\lambda M)N \xrightarrow{\epsilon} M[1\backslash N]$, the function $\omega = \Omega(D)$ is defined by the following properties:
   a. For any occurrence $u \in \mathcal{D}(M)$ which is not an occurrence of the substituted variable, one has $\omega(u) = 1.1.u$,
   b. For any occurrence $u \in M[1\backslash N]$ such that $u = v.w$ where v is an occurrence of the substituted variable in M and w is an occurrence in $\mathcal{D}(N)$, one has $\omega(u) = 2.w$.

The origin functions for linear term rewriting systems are also given by a mapping Ω_{trs}. To handle the fact that origin functions are only partial functions, we add an element *nil* to $\mathcal{O}$ that represents the lack of origin for some occurrences. Thus, the images of derivations by Ω_{trs} are functions over $\mathcal{O} \cup \{nil\}$ that map *nil* to *nil*. The mapping Ω_{trs} is defined by the same properties 1, 2, 3 as those used for Ω and by the property 4′ given below:

4′. For any elementary derivation $D = T \xrightarrow{(\epsilon,r)} T'$ where r is a rule of the form $\alpha \rightarrow \beta$, the function $\omega = \Omega_{trs}(D)$ is the function defined by the following properties:
   a. For any occurrence $v_x \in \mathcal{D}(\beta)$ such $\beta/v_x = x$ is a variable and for any occurrence w and v such that $w = v_x.v \in T'$, one has $\omega(w) = u_x.v$ where u_x is the only occurrence of x in α (remember that we are working with a linear term rewriting system).
   b. For any occurrence $v \in \mathcal{D}(\beta)$ which is not an occurrence of variable, one has $\omega(v) = nil$.

2.4. *Markings as extensions of Origins.*

A *marking* $\mathcal{M}$ is a function that maps the derivations of a linear term rewriting system to functions over $\mathcal{O} \cup \{nil\}$. It is defined by the same properties as the origin mapping Ω_{trs} for term rewriting systems, except that constraint 4′ is replaced by the following one:

4″. For any elementary derivation $D = T \xrightarrow{(\epsilon,r)} T'$ where $r = \alpha \rightarrow \beta$, the function $f = \mathcal{M}(D)$ is the function defined by the following properties:

a. *same as* 4′.*a, replacing* ω *by* f.

b. For any occurrence v such that $v \in \mathcal{D}(\beta)$, we have $f(v) = f_r(v)$ where $f_r = \mathcal{M}(D_r)$ and

$D_r = \alpha \overset{(\epsilon,r)}{\rightarrow} \beta$.

Note that a marking is entirely specified by the values of all the functions f_r for the rules in the term rewriting system. The properties of markings given above make that we do not need to know the initial and final terms of a derivation to compute the marking function, we only need to know the sequence of elementary steps and the domain of the final term. Also, Ω_{trs} simply appears as a special kind of marking.

3. Correct Markings for λEnv.

In this section, we first describe constraints on markings for the term rewriting system λEnv such that marking functions associated to derivations of λEnv represent correctly the origin functions associated to the λ-calculus derivations they represent, and we then provide a proof that these constraints are sufficient.

3.1. *A class of Correct Markings.*

The rules of $SEnv$ can be classified in two subsets. Some rules describe the interaction of the substitution mechanism with the operators of the λ-calculus (*App*, *Lambda*, *FVar*, *RVar*, *VarShift1*, *VarShift2*, *FVarLift1*, *FVarLift2*, *RVarLift1*, *RVarLift2*); the others describe the interaction of the substitutions with substitutions.

For the rules of the first subset, the left hand side always has the substitution operator $\cdot[\cdot]$ as head operator, the head operator of the first subterm is an operator from Op_λ, and it is the only occurrence of such an operator in the left hand side. In the right hand side, an operator of Op_λ may or may not occur. For the rules of the second subset, no operator of Op_λ occurs.

We consider the markings that verify the extra constraint that for every rule of the first set $r = \alpha \rightarrow \beta$ and any occurrence u such that $op(u,\beta) \in Op_\lambda$, one has $f_r(u) = 1$. Since r is in the first set, one has $op(1,\alpha) \in Op_\lambda$.

From now on, $\mathcal{M}$ will be a marking that verifies this constraint. We prove in the following that the marking functions associated to derivations of λEnv represent correctly the origin functions of the λ-calculus.

3.2. *Proof of Correctness.*

To prove that λEnv actually implements the λ-calculus, the authors of [Hardin&Lévy89] have to prove that for every derivation of the λ-calculus there exists a derivation of λEnv that is coinitial and cofinal, and conversely that for any derivation of λEnv whose initial and final terms are in Λ there exists a λ-calculus derivation that is also coinitial and cofinal. We have to be more careful because two coinitial and cofinal λ-calculus derivations may have different origin functions, as is shown by the derivations D_1 and D_2 below:

D_1 $\qquad (\lambda 1)((\lambda 1)(\lambda 1)) \overset{\epsilon}{\rightarrow} (\lambda 1)(\lambda 1)$

D_2 $\qquad (\lambda 1)((\lambda 1)(\lambda 1)) \overset{2}{\rightarrow} (\lambda 1)(\lambda 1)$

The origin functions f_1 and f_2 for these two derivations are different. For example, one has $f_1(\epsilon) = 2$ and $f_2(\epsilon) = \epsilon$. It is obviously useless to try to compare the marking function associated to a λEnv derivation that implements D_2 with the origin function of D_1.

To turn around this problem, we prefer to state our result using the following two theorems.

Theorem 1. *For any derivation Δ of the λ-calculus, there exists a derivation D of λEnv, that has the same initial and final term as Δ and such that $\mathcal{M}(D) = \Omega(\Delta)$.*

Theorem 2. *For any derivation D of λEnv whose initial and final terms are both in Λ, there exists a derivation Δ of the λ-calculus that has the same initial and final term as D and such that $\mathcal{M}(D) = \Omega(\Delta)$.*

Theorem 1 is a direct consequence of the following lemma, which describes the computation of one substitution at a time. One result of [Hardin&Lévy89] is that the normalization of $M[N \cdot id]$ in $SEnv$ yields the term $M[1\backslash N]$. We extend this result by stating the properties of the corresponding marking function.

Lemma 1. *For any terms M and N of Λ, there exists a derivation D of $SEnv$ normalizing $M[N \cdot id]$ with a marking function $\mathcal{M}(D) = f$ verifying the following properties:*

1. *For any occurrence $u \in \mathcal{D}(M)$ which is not an occurrence of the substituted variable, one has $f(u) = 1.u$,*
2. *For any occurrence $u \in \mathcal{D}(M[1\backslash N])$ such that $u = v.w$ where v is an occurrence of the substituted variable in M and w is an occurrence in $\mathcal{D}(N)$, one has $f(u) = 2.1.w$.*

The proof of this lemma uses the following four properties:

Property P1. *For any term $M \in \Lambda$, any substitution $S \in \Lambda Env$, and any natural n there exists a derivation D leading from $M[S]$ to a term M', with a marking function f such that:*

1. $\forall u \in \mathcal{D}(M) \quad |u| < n \quad op(u, M) \in \{@, \lambda\} \qquad op(u, M') = op(u, M),\ f(u) = 1.u,$
2. $\forall u \in \mathcal{D}(M) \quad M/u = M'' \quad (|u| < n \quad \wedge \quad M'' \in \mathcal{N}) \quad \vee \quad (|u| = n)$

$$M'/u = M''[\overbrace{\Uparrow(\ldots\Uparrow}^{p \text{ times}}(S)\ldots)] \qquad \text{where } p = depth(u, M),$$

$\forall v \in \mathcal{D}(M''), f(u.1.v) = u.v,$

$\forall v \in \mathcal{D}(S), f(u.2.\overbrace{1.\cdots.1}^{p \text{ times}}.v) = 2.v.$

Proof. By induction on n. For $n = 0$ the result is trivial.
Let D_n, f_n, and M_n be the correct derivation, the associated marking function, and the final term of the derivation for a given natural n, let $u_1, \ldots, u_m$ be the occurrences of length n in $\mathcal{D}(M)$ such that $op(u_i, M) \in \{@, \lambda\}$, and let $r_1, \ldots, r_m$ be the rules such that $r_i = App$ if $op(u_i, M) = @$ and $r_i = Lambda$ if $op(u_i, M) = \lambda$. Let s'_n be the sequence $s'_n = (u_1, r_1); \cdots; (u_m, r_m)$ and let D'_n be the derivation $D'_n = M_n \xrightarrow{S'_n} M_{n+1}$. The derivation $D_{n+1} = D_n; D'_n$ is a correct derivation for λEnv. Let f_{n+1} be the associated marking function. The rest of the proof, omitted here, shows that D_{n+1}, f_{n+1}, and M_{n+1} verify the good properties. □

Property P2. *For any number $n \in \mathcal{N}_+$ and any term S built only with the operators $\Uparrow$, $\uparrow$, id and $\circ$, there exist a derivation D starting from $n[S]$, whose final term is a number m, and such that the associated marking function f verifies $f(\varepsilon) = 1$.*

Proof. By induction on the length of the longest derivation for $SEnv$ starting from $n[S]$ (such an induction is licit since $SEnv$ is noetherian). If this length is 1, then the rule applied is either *IdEnv*, *VarShift1*, or *FVarLift1*. The constraints imposed on the marking ensure that $f(\varepsilon) = 1$.

If the length of the derivation is greater than 1, let (u, r) be the first step and f' the corresponding marking function. By induction hypothesis, we only have to verify that $f'(1) = 1$. If one

has $u \neq \varepsilon$, then $2 \leq u$ and this is trivially the case. If one has $u = \varepsilon$, then r is either *VarShift2*, *FVarLift2*, *RVarLift1*, or *RVarLift2*. In all these cases, the constraints we impose on the marking ensure that $f'(\varepsilon) = \varepsilon$. □

Property P3. *For any distinct numbers n and p and any term N there exist a derivation starting from $n[\overbrace{\Uparrow(\cdots\Uparrow}^{p-1 \text{ times}}(N)\cdots)]$ whose final term is a number m and such that the associated marking function f verifies $f(\varepsilon) = 1$.*

Proof. there are two cases, depending on whether $n > p$ or $n < p$. If $n > p$ there exist a derivation D' of the form $D' = n[\Uparrow(\cdots\Uparrow(N)\cdots)] \xrightarrow{s} n'[S]$ where

$$s = (\varepsilon, RVarLift1); \overbrace{(\varepsilon, RVarLift2); \ldots; (\varepsilon, RVarLift2)}^{p-1 \text{ times}}; (\varepsilon, RVar)$$

leading to a term of the form $n'[S]$ where n' is a natural number and S is a term built only with the operators $\Uparrow$, *up*, *id* and $\circ$. The constraints imposed on the marking ensure that the marking function f' associated to D' verifies $f(\varepsilon) = \varepsilon$. the result is then given by P2.

If $n < p$ then there exist a derivation D' of the form $D' = n[\Uparrow(\cdots\Uparrow(N)\cdots)] \xrightarrow{s} n'[S]$ where

$$s = (\varepsilon, RVarLift1); \overbrace{(\varepsilon, RVarLift2); \ldots; (\varepsilon, RVarLift2)}^{n-2 \text{ times}}; (\varepsilon, FVarLift1)$$

. Here again the associated marking function f' is such that $f'(\varepsilon) = \varepsilon$, S is built only with $\Uparrow$, *up*, *id* and $\circ$, and the result is given by property P2. □

Property P4. *For any number n and any term N of Λ there exists a derivation D going from $n[\overbrace{\Uparrow(\cdots\Uparrow}^{n \text{ times}}(N)\cdots)]$ to a term N' of λ such that for any $v \in \mathcal{D}(N)$ one has either $op(v, N) = op(v, N')$ or $op(v, N) \in \mathcal{N}$ and $op(v, N') \in \mathcal{N}$. If f is the associated marking function, then one has $f(v) = 2.\overbrace{1.\cdots.1}^{n \text{ times}}.v$.*

Proof. Let D' be the derivation $D' = n[\Uparrow(\cdots\Uparrow(N)\cdots)] \xrightarrow{s} 1[N[\uparrow \circ \ldots \circ \uparrow] \cdot id]$ where

$$s = (\varepsilon, RVarLift1); \overbrace{(\varepsilon, RVarLift2); \cdots; (\varepsilon, RVarLift2)}^{n \text{ times}}; (1, MapEnv)$$

and let f' be the associated marking function. by the definition of marking functions, one has $f'(2.1.1.v) = f'(2.\overbrace{1.\cdots.1}^{n \text{ times}}.v)$ for any occurrence $v \in \mathcal{D}(N)$. From properties P1 and P2, we obtain that there exist a derivation going from $N[\uparrow \circ \ldots \circ \uparrow]$ to a term N' verifying the good properties and such that the associated derivation f'' verifies $f''(v) = 1.v$. Last, there exists a derivation of the form $1[N' \cdot (\uparrow \circ \ldots \circ \uparrow)] \xrightarrow{(\varepsilon, FVar)} N'$ whose associated marking function f''' is such that $f'''(v) = 2.v$. Altogether, we have constructed a derivation going from $n[\Uparrow(\cdots\Uparrow(N)\cdots)]$ to N' whose marking function f verifies $f(v) = 2.\overbrace{1.\cdots.1}^{n \text{ times}}.v$ for every occurrence $v \in \mathcal{D}(N)$. □

Together properties P1, P2, P3, and P4 give a proof of lemma 1. To find a λEnv derivation D that has the same initial and final term as a λ-reduction $\Delta = M \xrightarrow{u} N$ and such that $\mathcal{M}(D) = \Omega(\Delta)$,

we simply need to perform the rewriting $M \overset{(u,Beta)}{\rightarrow} M'$ and use lemma 1 to find a $\mathcal{S}Env$ derivation leading to N that has the correct marking function. For a non-elementary λ-derivation, we simply need to repeat the operation.

The proof of theorem 2 uses an auxiliary marked rewriting system, for which we prove some confluence properties. The result then comes from the close relation between this marked rewriting system and marking functions. We first introduce a language of marked terms:

Definition. Let $\mathcal{N}_+^*$ be a set of *marked* integers, $\mathcal{N}_+^* = \{1^*, 2^*, \ldots\}$, and let $Op_\lambda^* = \{@^*, \lambda^*\} \cup \mathcal{N}_+^*$ be a set of *marked* operators. Let ΛEnv^* be the set of terms obtained by replacing some occurrence of operators of Op_λ by the corresponding operator of Op_λ^*. Let $m : Op_\lambda \rightarrow Op_\lambda^*$ be the marking bijection such that $m(@) = @^*$, $m(\lambda) = \lambda^*$, and $m(n) = n^*$ for every number n. For any term $t \in \Lambda Env^*$, and any occurrence $u \in \mathcal{D}(t)$ such that $op(u,t) \in Op_\lambda$, we note $m(t,u)$ the term t' such that $op(u,t') = m(op(u,t))$ and $op(w,t') = op(w,t)$ for any $w \neq u$. Let $erase : \Lambda Env^* \rightarrow \Lambda Env$ be the function that maps any term t to the term t' such that $op(u,t') = op(u,t)$, if $op(u,t') \notin Op_\lambda^*$ and $op(u,t') = op$ where $m(op) = op(u,t)$ otherwise.

Stars in a term are simply used to denote occurrences. To compute a descendance relation, we can simply use a rewriting system working on marked terms. The descendance relation we study is simply parameterized by the way we choose to put stars on both sides of a rewriting rule. To study marking functions, we use the rewriting system introduce in the following definition:

Definition. Let λEnv^* be the smallest rewriting system containing λEnv and such that if $r = \alpha \rightarrow \beta$ is a rule in λEnv^*, u is an occurrence such that $\alpha/u \in Op_\lambda$ and v is the occurrence such that $\beta/v \in Op_\lambda$, $\alpha' = m(u,\alpha)$ and $\beta' = m(v,\beta)$ then the rule $\alpha' \rightarrow \beta'$ is also in λEnv^*. The function *erase* introduced in the previous definition extends naturally to rules of λEnv^*. For any derivation $D = M \overset{(u,r)}{\rightarrow} N$ in λEnv, and any term M' such that $erase(M') = M$, we denote $lift(D, M')$ the derivation $D' = M' \overset{(u,r')}{\rightarrow} N'$ where r' is the only rule in λEnv^* such that $erase(r') = r$ and D' is a valid derivation. This function *lift* extends naturally to non-elementary derivations by concatenation. We denote $\mathcal{S}Env^*$ the subset of λEnv^* that contains any rule r such that $erase(r) \in \mathcal{S}Env$ and $Beta^*$ the system $\lambda Env^* - \mathcal{S}Env^*$.

The following lemma shows that λEnv^* enables us to directly use rewriting for studying marking functions.

Lemma 2. *Let $D = M \twoheadrightarrow N$ be a derivation of λEnv, and f be the function $f = \mathcal{M}(D)$. For any occurrences $v \in \mathcal{D}(N)$ and $u \in \mathcal{D}(M)$ such that $op(u,M) \in Op_\lambda$ and $op(v,M) \in Op_\lambda$, and for any terms M' and N' in ΛEnv^* we have the following property:*

$$lift(D, m(M,u)) = M' \twoheadrightarrow N' \quad \Rightarrow \quad op(v,N') \in Op_\lambda^* \Leftrightarrow u = f(v)$$

Proof. By induction on the length of D.
- If D is a derivation of length 0, then the result is trivial.
- If $D = D'; D''$ where D'' is of the form $P \overset{(w,r)}{\rightarrow} N$, let f' and f'' be the function such that $f' = \mathcal{M}(D')$ and $f'' = \mathcal{M}(D'')$ Let P' be the term of $\mathcal{S}Env^*$ such that $lift(D', m(M,u))$. If we have $v = w.v'$ where $v' \in \mathcal{D}(\beta)$ and β/v' is not a variable, then $op(v,N') \in Op_\lambda^*$ if and only if $op(f''(v),P') \in Op_\lambda^*$, from the definition of $\mathcal{S}Env^*$. By induction hypothesis, $op(f''(v),P') \in Op_\lambda^*$ if an only if $f'(f''(v)) = u$. If $v \neq w.v'$ where $v' \in \mathcal{D}(\beta)$ and β/v' is not a variable, then we have directly $op(v,N') = op(f''(v),P')$ and the induction hypothesis also gives the result. □

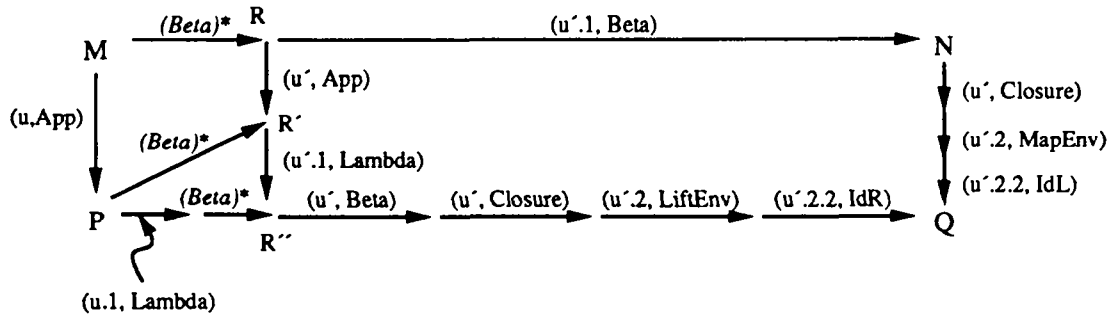


Figure 3. Resolution of critical pairs between $(Beta)^*$ and $SEnv$.

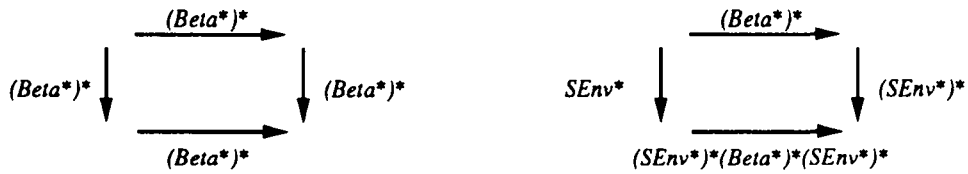
Property P5. *The rewriting system $SEnv^*$ is noetherian and confluent.*

Proof. This rewriting system is locally confluent, as can be checked automatically using the algorithm of Knuth and Bendix [Knuth&Bendix70], and it is also noetherian, since $SEnv$ is noetherian [Hardin&Lévy89] and any derivation of $SEnv^*$ is mapped to a derivation of $SEnv$ of same length by the function *erase*. Therefore, it is confluent [Huet80]. □

Corollary 1. *If M and N are elements of Λ, then all the normalizing derivations for $SEnv$ starting from $M[N \cdot id]$ have the property stated in lemma 1.*

In the following, we shall write $Sim\beta^*$ the set of derivations in λEnv^* that perform first a reduction using one of the rules in $Beta^*$ then a $SEnv^*$ normalization, then a $Beta^*$ reduction, etc. Corollary 1 simply indicates that derivations in $Sim\beta^*$ have the property needed in theorem 2.

Lemma 3. The following diagrams hold, where $(S^*)^*$ represents repetitive uses of the rules in the system S^*.



Proof. Diagram a comes from the fact that $Beta^*$ does not contain any critical pair and that a $Beta^*$-reduction cannot create a $Beta^*$-redex. For this reason, a derivation of $Beta^*$ reduces only residuals of redexes that are all present in the initial term and we can use the possibility to reduce in any order (in parallel) non-overlapping redexes [Boudol85]. Diagram b can be proved by induction on number of redexes reduced in parallel by the derivation on the upper arrow. If the redex reduced on the leftmost arrow does not overlap with any of the redexes reduced by the upper arrow, then the result comes directly from the possibility to reduce in any order non-overlapping redexes. If the redex reduced on the leftmost arrow overlap with one of the redexed reduced by the upper arrow, then the rule r used on the leftmost arrow is such that $m(r) = App$ and we can use diagram a to suppose without loss of generality that the $Beta$ redex that overlaps with (u, r) is reduced last. If we consider the unmarked derivation, we can then draw the diagram given in figure 3.

The same diagram can be drawn with marked derivations, where the appropriate rules for App, $Lambda$, and $Beta$ are chosen depending on the marks carried by M at occurrences $u.1$ and $u.1.2$. The final result Q does not depend on these marks. □

Lemma 4. *any derivation of λEnv^* whose initial and final terms are in Λ^* has the same initial*

and final term as a derivation in $Sim\beta^*$.

Proof. By induction on the number of *Beta** reductions in this derivation. If this number is 0, then the derivation is empty, since a term of Λ^* is in $SEnv^*$ normal form.
Let $D = D_1; D_2$ be a derivation of λEnv^* whose initial and final terms are in Λ^*, where D_1 is a *Beta** reduction followed by a $SEnv^*$ normalization. Let d be a $SEnv^*$-normalization of the initial term of derivation D_2. From lemma 3, diagram c, there exists a derivation D_2' of the form $D_2' = d; d_b; d'$ that has the same initial and final term as D_2, where d_b is a *Beta** derivation and d' is a $SEnv^*$ normalization. We can apply the induction hypothesis to the derivation $D_1; d$. From lemma 3, diagram a, we can suppose that the the redexes reduced in d_b are in decreasing order with respect with the order $\leq$ on occurrences. We can then prove by induction on the length of d_b that the derivation $d_b; d'$ has the same initial and final term as a derivation in $Sim\beta^*$. This is done by separating the first redex from the others. Because this redex appears at an occurrence that is maximal for $\leq$, the normalization of its result does not interfer with the other *Beta* redexes. The detailed proof is omitted here. □

The proof of theorem 2 then comes directly from lemmas 2 and 3. From lemma 2, we obtain that any $Sim\beta$ derivation D has the same initial and final term as a λ-derivation Δ and that we have $\mathcal{M}(D) = \Omega(\Delta)$. From lemma 3, we obtain that it is possible to find a $Sim\beta$ derivation that has the same initial and final term and the same marking function as any derivation of λEnv starting and finishing in Λ.

4. Computing Origins for $\lambda\sigma$.

The rewriting system $\lambda\sigma$ is presented in the appendix. We call σ the subsystem obtained by removing rule *Beta*. the set of terms on which $\lambda\sigma$ works is the subset $\Lambda\sigma$ of ΛEnv that contains all the terms constructed without the operator $\Uparrow$. As we saw in the introduction, $\lambda\sigma$ uses the rule *FVar* both to perform actual substitutions and to protect bound variables from substitution. With respect to origin computations the two usages should be distinguished. We recall the form of this rule:

FVar $\qquad 1[M \cdot id] \rightarrow M$

When an actual substitution is performed, we agree that the origins of nodes in the result M are actually in the subterm M of the left hand side, but when the reduction corresponds to protecting a bound variable, M is actually a term that derives to a number n and we want the origin of the result to be in the subterm 1 of the left hand side. Thus, we have to choose between two possible marking functions for a derivation of the form $T \overset{(u,FVar)}{\rightarrow} T'$. When computing the marking function for a non-elementary derivation, we need to find a way to choose which possible function each time the rule *FVar* is used.

4.1. *A Correct Mapping for $\lambda\sigma$.*

The solution we propose is a mapping $\mathcal{M}_\sigma$ working only for derivations that start from a term of Λ, constructed recursively using the following rules. In these rules we note (u *or* v) the occurrence w such that $w = u$ if $u \neq nil$ and $w = v$ otherwise (*or* behaves like a sequential or).

1. For any term $M \in \Lambda$, if D is the derivation $D = M \overset{\emptyset}{\rightarrow} M$ then $\mathcal{M}_\sigma(D) = 1_{\mathcal{D}(M)}$.

2. If D, D', and D'' are derivations such that $D = T \overset{S}{\rightarrow} T'$, $D' = T \overset{S;(u,r)}{\rightarrow} T''$, and $D'' = T' \overset{(u,r)}{\rightarrow} T''$ then the function $f' = \mathcal{M}_\sigma(D')$ is defined from the function $f = \mathcal{M}_\sigma(D)$ by $f' = f \circ f''$ where f'' is given as follows:

a. If the rule r is *Beta*, *app*, *IdL*, *ShiftId*, *Shift*, *AssEnv*, or *MapEnv*, then $f'' = \mathcal{M}(D'')$ where $\mathcal{M}$ is a mapping verifying the constraints defined previously for λEnv. In all the other cases we have $f'' = \Omega_{trs}(D'')$, almost everywhere, with the following exceptions:

b. If the rule r is *Lambda'*, then we have $f(u) = f(u.1)$. We recall that rule *Lambda'* has the following form:

$$\textit{Lambda'} \qquad (\lambda M)[s] \rightarrow \lambda(M[1 \cdot (s \circ \uparrow)])$$

Note that this definition does not give any origin to the index in the right hand side.

c. If r is the rule $r = FVar$, then we have $f(u) = f'(u.2.1)$ *or* $\big(f'(u.1)$ *or* $f'(u)\big)$.

d. If r is the rule $r = Closure$, then we have $f(u.1) = f'(u.1.1)$ *or* $\big(f'(u.1)$ *or* $f'(u)\big)$.

e. If r is *VarShift2* or *RVar*, then we have $f(u.1) = f'(u.1)$ *or* $f'(u)$.

f. If r is *VarShift1* or *VarId*, then we have $f(u) = f'(u.1)$ *or* $f'(u)$.

4.2. *Proof of Correctness.*

With this mapping $\mathcal{M}_\sigma$ we have the same results for $\lambda\sigma$ as we had earlier for λEnv, that is, we have the following two theorems:

Theorem 3. *For any derivation Δ of the λ-calculus, there exists a derivation D of $\lambda\sigma$, that has the same initial and final term as Δ and such that $\mathcal{M}_\sigma(D) = \Omega(\Delta)$.*

Theorem 4. *For any derivation D of $\lambda\sigma$ whose initial and final terms are both in Λ, there exists a derivation Δ of the λ-calculus that has the same initial and final term as D and such that $\mathcal{M}_\sigma(D) = \Omega(\Delta)$.*

The proof of these theorems is very similar to the proof of theorems 1 and 2. We have to take into account the fact that the mapping $\mathcal{M}_\sigma$ is defined only for derivations starting on terms of Λ. We first use a lemma that reproduces the result of lemma 1:

Lemma 4. *For any terms M, N, and P in Λ there exists a derivation D of the form $D_b; D'$ where $D_b = P[u \leftarrow (\lambda(M)N)] \overset{(u,Beta)}{\rightarrow} P[u \leftarrow M[N \cdot id]]$ and D' is a derivation of σ ending on a term P' of Λ, such that the function $f = \mathcal{M}_\sigma(D)$ verifies the following properties:*
- *$\forall v \in \mathcal{D}(P')$ $u \not\leq v$ $f(v) = v$.*
- *$\forall v \in \mathcal{D}(M)$ v is not an occurrence of the bound variable in M $f(u.v) = u.1.1.v$*
- *$\forall v \in \mathcal{D}(M)$ v is an occurrence of the bound variable, $\forall w \in \mathcal{D}(N)$ $f(u.v.w) = f(u.2.w)$.*

The proof of this lemma uses the following properties, that reproduce the results of properties P1 to P4.

Property P7. *For any term P of Λ , any derivation D starting in P and finishing in P' such that $P'/u = M[S]$, and any natural n there exists a derivation D' starting from P' and ending in a term P'' such that the functions $f = \mathcal{M}(D)$ and $f' = \mathcal{M}(D; D')$ and the term P'' verify the following properties:*

1. $\forall v \quad u \not\leq v \quad f'(v) = f(v), op(v, P'') = op(v, P')$,
2. $\forall v \in \mathcal{D}(M) \quad |v| < n \quad op(v, M) = \{@, \lambda\} \quad op(u.v, P'') = op(v, M), f'(u.v) = f(u.1.v)$,
3. $\forall v \in \mathcal{D}(M) \quad M/v = M' \ (|v| < n \ \wedge \ M' \in \mathcal{N}) \vee (|u| = n)$,

$$P''/u.v = M'[(1 \cdot \ldots (p \cdot S \circ (\overbrace{\uparrow \circ (\ldots (\uparrow \circ \uparrow) \ldots)}^{p \text{ times}})) \ldots)] \qquad \text{where } p = depth(v, M),$$

$\forall w \in \mathcal{D}(M'),\ f'(u.v.1.w) = f(u.v.w), \qquad \forall w \in \mathcal{D}(S),\ f'(u.v.2.\overbrace{2.\cdots.2}^{p \text{ times}}.1.w) = f(u.2.w),$

$\forall q \leq p,\ f(u.v.2.\overbrace{1.\cdots.1}^{q \text{ times}}) = nil.$

Proof. The proof of this property is the same as the proof of property P1, replacing uses of the rule *Lambda* by judiciously chosen sequences of *Lambda'*, *MapEnv*, *VarShift*, and *AssEnv*.

The following property is the equivalent of Property P2:

Property P8. *For any derivation D starting from a term $P \in \Lambda$ and ending in a term P', such that $P'/u = n[S]$ where $n \in \mathcal{N}_+$ and S is a substitution build only with the operators $\cdot$, $\uparrow$, id, $\circ$, $[\,]$, and the elements of $\mathcal{N}_+$, for any derivation D' starting from $n[S]$ and ending in a term $m \in \mathcal{N}_+$, if f and f' are the functions $f = \mathcal{M}_\sigma(D)$ and $f' = \mathcal{M}_\sigma(D; u.D')$ and if $f(u) = nil$ and $f(u.2.v) = nil$ for any occurrence $v \in \mathcal{D}(S)$ such that $S/v \in \mathcal{N}_+$ or $op(v, S) = [\,]$, then we have $f'(u) = f(u.1)$.*

Proof. This proof is done by induction on the length of the longest derivation starting from $n[S]$. It contains a lot of cases, corresponding to the different possible rules used and to the different possible values of the function f. This tedious proof is omitted here. □

The following derivation describes exactly the case when a bound variable is protected from substitution.

Property P9. *For any derivation D starting from a term $P \in \Lambda$ and ending in a term P' such that $P'/u = n[1 \cdot (\ldots p-1 \cdot (N \cdot id) \circ \uparrow \circ (\ldots(\uparrow \circ \uparrow)\ldots)\ldots)]$ where n and p are different numbers, there exists a derivation D' such that $D' = P' \overset{u.s}{\rightarrow} P''$, $P''/u = m \in \mathcal{N}_+$, and if f and f' are the functions $f = \mathcal{M}_\sigma(D)$ and $f' = \mathcal{M}_\sigma(D; D')$, if $f(u) = nil$ and $f(u.\overbrace{2.\cdots.2}^{k \text{ times}}.1) = nil$ for any k such that $1 \leq k \leq p$, then we have $f'(u) = f(u)$.*

Proof. There are two cases. If $n > p$, let s' be the sequence:

$$s' = \overbrace{(\varepsilon, RVar); \ldots; (\varepsilon, RVar)}^{p-1 \text{ times}}; (2, MapEnv); (\varepsilon, RVar)$$

and let D'' be the derivation $D'' = P' \overset{u.s}{\rightarrow} Q$. The term Q is such that $Q/u = n - p[S]$ where S is a substitution built only with the operators id, $\uparrow$, and $\circ$. Let f'' be the function $f'' = \mathcal{M}_\sigma(D; D'')$, we have $f''(u) = nil$, $f''(u.1) = f(u.1)$ *or* $f(u)$. Property P8 then gives the result. If $n < p$ let s be the sequence:

$$s = \overbrace{(\varepsilon, RVar); \ldots; (\varepsilon, RVar)}^{n-1 \text{ times}}; (\varepsilon, FVar)$$

we have $f'(u) = f(u.\overbrace{2.\ldots.2}^{n \text{ times}}.1)$ *or*$(f(u.1)$ *or* $f(u))$. Since $f(u.\overbrace{2.\ldots.2}^{n \text{ times}}.1) = f(u) = nil$ we have $f'(u) = f(u.1)$. □

Property P10. *For any derivation D starting from a term $P \in \Lambda$ and ending in a term P' such that $P'/u = n[1 \cdot \ldots (n-1 \cdot ((N \cdot id) \circ \uparrow \circ (\ldots \circ (\uparrow \circ \uparrow)\ldots)))\ldots]$ where $N \in \Lambda$, there exists a derivation D' such that $D' = P' \overset{u.s}{\rightarrow} P''$, $P''/u = N' \in \Lambda'$, for any occurrence $v \in \mathcal{D}(N)$ one has $op(v, N) = op(v, N')$ or $op(v, N) \in \mathcal{N}_+$ and $op(v, N') \in \mathcal{N}_+$. and if f and f' are the function $f = \mathcal{M}_\sigma(D)$ and $f' = \mathcal{M}_\sigma(D')$ then we have $f'(u.v) = f(u.\overbrace{2.\ldots.2}^{n \text{ times}}.1.v)$ for any occurrence $v \in \mathcal{D}(N)$.*

Proof. Let s' be the sequence

$$s' = \overbrace{(\varepsilon, RVar); \ldots; (\varepsilon, RVar)}^{n-1 \text{ times}}(2, MapEnv); (\varepsilon, FVar)$$

and let D'' be the derivation $P' \overset{u.s'}{\underset{Q}{\twoheadrightarrow}}$. The term Q is such that $Q/u = N[\uparrow \circ (\ldots(\uparrow \circ \uparrow)\ldots)]$. Let f'' be the function $\mathcal{M}_\sigma(D; D'')$. We have $f''(u) = nil$, $f''(u.1.v) = f(u.\overbrace{2.\ldots.2}^{n \text{ times}}.1.v)$ for any occurrence v in $\mathcal{D}(N)$. The properties P7 and P8 then give the result. □

Together, properties P7, P8, P9, and P10 give a proof of lemma 4. This lemma has the same corollaries as lemma 1, and among them comes theorem 3.

To prove theorem 4, we use a more elaborate labeled rewriting system as we did for theorem 2. This rewriting system is based on the following language of labels.

Definition. Given an infinite set $\mathcal{A}$, we consider the language L constructed from $\mathcal{A} \cup \{nil, or\}$ where *nil* and the elements of $\mathcal{A}$ have a null arity and *or* has arity two. Let *or* and *nil* have the following properties, for every x, y, and z in L.

$$x \; or \; nil = nil \; or \; x = x \qquad\qquad (x \; or \; y) \; or \; z = x \; or \; (y \; or \; z)$$

We consider the set $\Lambda\sigma^L$ of terms $\Lambda\sigma$ labeled with elements of L. We assimilate unlabeled terms with labeled terms where all the nodes are labeled with *nil*. Given a term $M \in \Lambda\sigma$, and a function $\mathcal{L} : \mathcal{D}(M) \to L$, we note $\langle M, \mathcal{L}\rangle$ the labeled term $M' \in \Lambda\sigma^L$ that carries the label $\mathcal{L}(u)$ at occurrence u, for any ocurrence $u \in \mathcal{D}(M)$. We also note $M = erase(M')$ and $\mathcal{L} = labels(M')$. labeled terms are represented by writing the label as exponent. For example, if P is the term $P = (\lambda(M)^l N)^m$ and $\mathcal{L} = labels(P)$ we have $\mathcal{L}(\varepsilon) = m$ and $\mathcal{L}(1) = l$. We allow ourselves to add a label l to a term M by writing M^l, with the convention that $(M^l)^m = M^{l \; or \; m}$.

The basic idea behind this labeling is the same as in [Lévy78], [Klop80], [Field90], and [Maranget91]. However, the language of labels is much simpler.

Definition. The labeled rewriting system $\lambda\sigma^L$ is given in appendix. The function *erase* introduced in the previous definition extends naturally to rules of $\lambda\sigma^L$. For any derivation $D = M \overset{(u,r)}{\to} N$ of $\lambda\sigma$ and any labeling function $\mathcal{L} : \mathcal{D}(M) \to L$ we note $lift_L(D, \mathcal{L})$ the derivation $D' = \langle M, \mathcal{L}\rangle \overset{(u,r')}{\to} N'$ such that $erase(r') = r$ (in the following, $erase(r')$ and r will always be noted with the same name, the ambiguity being always solved by the context). As for the function *lift*, this function $lift_L$ extends naturally to non-elementary derivations. We note σ^L the sub-system $\lambda\sigma^L - \{Beta\}$.

This labeled rewriting system permits to represent computations of origins for $\lambda\sigma$ exactly as λEnv^* did λEnv.

Definition. For any term M such that $op(\varepsilon, M) \in Op_\lambda \cup \{[\,]\}$, there exists a *characteristic occurrence* u_c of the form $1. \ldots .1$ such that for any occurrence $v < u_c$ we have $op(v, M) =$"$[\,]$" and $op(u_c, M) \in Op_\lambda$ (this comes directly from the definition of $\Lambda\sigma$). The label $\mathcal{L}(u_c)\ldots\mathcal{L}(1.1)\mathcal{L}(1)$, where $\mathcal{L} = labels(M)$, is called the head-label of M and is noted $head(M)$. We say that M is *head-labeled* if $head(M) \neq nil$ and that M is *well labeled* if for any occurrence u such that $op(u, M) \in \{@, \lambda\}$ one has $\mathcal{L}(M) \neq nil$.

Lemma 5. *If $D = M \twoheadrightarrow N$ is a σ^L derivation and M is well labeled and head-labeled then N is well labeled and head-labeled.*

Proof. Proof by induction on the length of D, omitted here. □

Lemma 6. *For any terms $M \in \Lambda^L$ and $N \in \Lambda\sigma^L$ such that M is well labeled and $D = M \twoheadrightarrow N$ is a σ^L derivation, for any subterm of N of the form $\lambda(P)$, the term P is head-labeled.*

Proof. By induction on the length of the derivation D that goes from M to N. If D is the empty derivation, then the result is trivial.

If $D = D'; D''$ where $D'' = M' \overset{(u,r)}{\rightarrow} N$ we have two cases:

- $M'/u.1 = \lambda(P)$ and $r = Lambda'$. we have $N/u = \lambda(P[S])$ for some term S and $P[S]$ is head labeled because P is head labeled. The result is trivial for any other ocurrence.
- In any other case, the result comes from the induction hypothesis and lemma 6. □

We can now come to the following lemma, which expresses the relation between $\lambda\sigma^L$ and $\mathcal{M}_\sigma$. This reproduces the result of lemma 2.

Lemma 7. *For any term $M \in \Lambda$, for any labeling function $\mathcal{L} : \mathcal{D}(M) \rightarrow \mathcal{A}$, for any derivation $D = M \twoheadrightarrow N$ of $\lambda\sigma$, if $\mathcal{L}'$ is the labeling function such that $lift_L(D, \mathcal{L}) = \langle M, \mathcal{L}\rangle \twoheadrightarrow \langle N, \mathcal{L}'\rangle$, and if f is the function $f = \mathcal{M}_\sigma(D)$, then we have:*

$$\forall v \in \mathcal{D}(N) \quad \mathcal{L}(f(v)) = \mathcal{L}'(v)$$

Proof. By induction on the length of D. If D is the empty derivation, then f is the identity function and $\mathcal{L} = \mathcal{L}'$, the result is trivial.

If we have $D = D'; D''$ where $D'' = M' \overset{(u,r)}{\rightarrow} N$ and $\mathcal{L}''$ is the function such that $lift_L(D, \mathcal{L}) = \langle M', \mathcal{L}''\rangle \rightarrow \langle N, \mathcal{L}'\rangle$, $f' = \mathcal{M}_\sigma(D')$ we only have to compare formally the labeled rewriting system and the definition of $\mathcal{M}_\sigma$. We have two cases:

1. If r is neither *Lambda'* nor *App*, then for any occurrence such that the definition of $\mathcal{M}_\sigma$ imposes $f(w) = f'(w')$ *or* $f'(w'')$ one has $w = u.v$, $w' = u.v'$, $w'' = u.v''$, and $l_\beta(v) = l_\alpha(v')$ *or* $l_\alpha(v'')$ if β/v is not a variable or $l_\beta(v) = l_\alpha(v'')$ and $\beta/v = \alpha/v'$ is a variable. For all the other occurrences, the result comes from the definitions of rewriting and origin functions. We can then apply the induction hypothesis.
   The same kind of reasoning holds when the definition of $\mathcal{M}_\sigma$ imposes a constraint of the form $f(w) = f'(w')$ *or* $f'(w'')$ $f'(w''')$.
2. If r is one of the rules *Lambda'* or *App* then we have $f(u) = f'(u.1)$ and $\mathcal{L}'(u) = \mathcal{L}''(u.1)$ *or* $\mathcal{L}'(u)$. However lemma 5 asserts that $\mathcal{L}''(u.1) \neq nil$ and we have $\mathcal{L}'(u) = \mathcal{L}''(u.1)$. As above, we can then apply the induction hypothesis. □

Property 11. *The rewriting system σ^L is confluent on ground terms.*

Proof. Most critical pairs of σ^L are confluent in the general case. For those critical pairs that are not simply confluent, the confluence can be checked automatically by a systematic treatement of all the possible cases of variable instances. □

This property has the same corollaries as property P5. If we note $Sim\beta_\sigma$ the set of derivations in $\lambda\sigma$, whose initial and final term are in Λ and which perform first a reduction using rule *Beta*, followed by a normalization for σ, followed by a *Beta*-reduction, etc, then all the derivations in $Sim\beta_L$ actually permit to prove theorem 3.

To prove the equivalent of lemma 3 we have to be more careful, and we need extra definition.

Lemma 8. The same diagrams as in lemma 3 hold for $\lambda\sigma^L$ on ground terms, replacing *Beta** by *Beta* and $SEnv^*$ by σ^L.

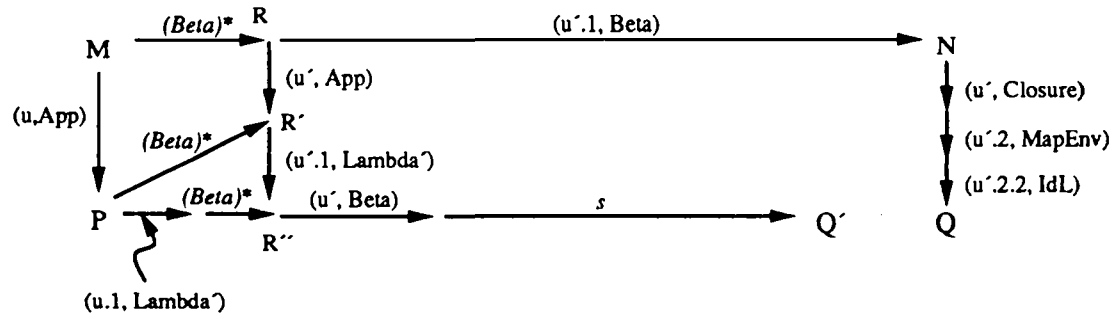


Figure 4. Resolution of critical pairs between $(Beta)^*$ and σ^L.

Proof. Diagram a can be proved exactly the same way as in lemma 3. For diagram c, the case where the σ-redex on the leftmost arrow overlaps with one of *Beta*-redexes is solved by the diagram given in figure 4, where the derivations $(Beta)^*$ represent the derivation that reduces the *Beta* redexes that do not overlap with the reduction in (u, App), u' is the residual of occurrence u', and s is the sequence

$$s = (u', Closure); (u'.2, MapEnv); (u'.2.1, FVar); (u'.2.2, AssEnv); (u'.2.2.2, Shift); (u'.2.2.2, IdL)$$

If the term R is such that $R/u' = ((\lambda(T)^u T')^v[S])^w$ then the terms Q and Q' obtained figure 4 differ only in the sense that $Q/u' = T^w[T'[S] \cdot S]$ and $Q'/u' = T[T'[S] \cdot (S \circ id)]$. We can prove that there exists a term S' such that $S \circ id \twoheadrightarrow S'$ and $S \twoheadrightarrow S'$ for any ground term S. This can be proved by induction on the size of S. Let S' be the normal form of S for σ. If S' is atomic, then we have $S = id$ or $S = \uparrow$, the rules *IdL* and *ShiftId* give the result. If S' is not atomic, then we have necessarily $S' = (t \cdot S'')$ for some terms t and S'', we can then write the following derivation:

$$(t \cdot S'') \circ id \overset{(\epsilon, MapEnv)}{\rightarrow} t[id] \cdot (S'' \circ id)$$

By induction hypothesis, we have $S'' \circ id \twoheadrightarrow S''$. We can prove using the same kind of reasoning that $t[id] \twoheadrightarrow t$. We thus prove that $T[T'[S] \cdot S]$ $T[T'[S] \cdot (S \circ id)]$ reduce to the same term. We still have to prove that $T^w[T'[S] \cdot S]$ and $T[T'[S] \cdot S]$ reduce to a same term. From lemma 6, we have that T is head labeled. Since T is also ground there exists a term T'' such that $T \twoheadrightarrow T''$ is a σ^L derivation and $op(\epsilon, T'') \in Op_\lambda$ since T'' is also head labeled, we have $labels(T'')(\epsilon) \neq nil$ and $labels(T''^w)(\epsilon) = labels(T'')(\epsilon)$. Therefore, the terms $T^w[T'[S] \cdot S]$ and $T[T'[S] \cdot S]$ reduce to the same term $T''[T'[S] \cdot S]$. □

5. Conclusion.

The concept of origin functions we have used in this paper seems a good tool to study the notion of descendance in reduction systems, with a good separation between descendance and other aspects of reduction. With the concept of marking functions, we have extended this notion of descendance in a way that proves fruitful when studying the λ-calculus and the system λEnv.

The difference between λEnv and $\lambda\sigma$, mainly that the former is confluent on all terms whereas the latter is confluent only on ground terms, is studied in detail in [CHL91], along with conditional variants of these systems. Our result shows another difference between the two systems. It would be interesting to understand whether the problem of computing origins and the problem of confluence are related.

On the side of programming language study and implementation, it would be interesting to extend this kind of result to graph rewriting systems, as many new implementations of functional programming languages use graph rewriting.

References.

[**ACCL90**] M. ABADI, L. CARDELLI, P.-L. CURIEN, J.-J. LÉVY, "Explicit Substitutions", *Proceedings of the 17th Annual Symposium on the Principles of Programming Languages*, San Francisco, California, January 1990, pp. 31–46.

[**Bertot91a**] Y. BERTOT, *Une Automatisation du Calcul des Résidus en Sémantique Naturelle*, PhD thesis, University of Nice, France, June 1991.

[**Boudol85**] G. BOUDOL, "Computational Semantics of Term Rewriting Systems", *Algebraic Methods in Semantics*, M. Nivat, J. C. Reynolds (editors), Cambridge University Press, 1985.

[**de Bruijn72**] N. DE BRUIJN, "Lambda-Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem", Indag. Math. 34,5, 1972, pp. 381–392.

[**CHL91**] P.-L. CURIEN, T. HARDIN, J.-J. LÉVY, "Confluence Properties of Weak and Strong Calculi of Explicit Substitutions", Personal communication, July 1991.

[**Curien86**] P.-L. CURIEN, *Categorical Combinators, Sequential Algorithms and Functional Programming*, Pitman, London, 1986.

[**Curry&Feys58**] H. CURRY, R. FEYS, *Combinatory Logic, Vol. I*, North-Holland, 1958.

[**Field90**] J. FIELD, "On Laziness and Optimality in Lambda Interpreters: Tools for Specification and Analysis", *Proceedings of the 17th ACM Symposium on Principles of Programming Languages*, San Francisco, California, January 1990, pp. 1–15.

[**Hardin&Lévy89**] T. HARDIN, J.-J. LÉVY, "A Confluent Calculus of Substitutions", *Proceedings of the France Japan Artificial Intelligence and Computer Science Symposium*, Izu, Japan, November 89.

[**Hindley&Seldin86**] J. HINDLEY, J. SELDIN, *Introduction to Combinators and λ-calculus*, London Mathematical Society Student Texts, Cambridge University Press, 1986.

[**Huet80**] G. HUET, "Confluent reductions: abstract properties and applications to term rewriting systems", Journal of the ACM, 27, 4, pp. 797–821, 1980.

[**Klop80**] J. W. KLOP, *Combinatory Reduction Systems*, PhD Thesis, Mathematisch Centrum Amsterdam, 1980.

[**Knuth&Bendix70**] KNUTH, D. E., BENDIX P. B., "Simple Word Problems in Universal Algebras", *Computational Problems in Abstract Algebra*, J. Leech (editor), Pergamon Press, 1970.

[**Lévy78**] J.-J. LÉVY, *Réductions Correctes et Optimales dans le Lambda Calcul*, PhD Thesis, Université de Paris VII, 1978.

[**Maranget91**] L. MARANGET, "Optimal Derivations in Weak Lambda-Calculi and in Orthogonal Terms Rewriting Systems", *Proceeding of the 18th ACM Symposium on Principles of Programming Languages*, Orlando, Florida, January1991, pp. 255–269.

[**Morris68**] J. MORRIS, *Lambda Calculus Models of Programming Languages*, PhD Thesis, MIT, Cambridge, Massachusetts, 1968.

[**Tolmach&Appel90**] A. P. TOLMACH, A. W. APPEL, "Debugging Standard ML without Reverse Engineering", *Proceedings of the 1990 ACM Conference on Lisp and Functional Programming*, Nice, France, June 1990, pp. 1–12.

Appendix: The Rewriting Systems λEnv and $\lambda\sigma$

(Beta)			$(\lambda a)b$	$\to$	$a[b \cdot id]$		
(App)	$(ab)[s]$	$\to$	$a[s]b[s]$	*(Lambda)*	$(\lambda a)[s]$	$\to$	$\lambda(a[\Uparrow(s)])$
(Closure)	$(a[s])[t]$	$\to$	$a[s \circ t]$	*(VarShift1)*	$n[\uparrow]$	$\to$	$n+1$
(VarShift2)	$n[\uparrow \circ s]$	$\to$	$n+1[s]$	*(FVar)*	$1[a \cdot s]$	$\to$	a
(FVarLift1)	$1[\Uparrow(s)]$	$\to$	1	*(FVarLift2)*	$1[\Uparrow(s) \circ t]$	$\to$	$1[t]$
(RVar)	$n+1[a \cdot s]$	$\to$	$n[s]$	*(RVarLift1)*	$n+1[\Uparrow(s)]$	$\to$	$n[s \circ \uparrow]$
(RVarLift2)	$n+1[\Uparrow(s) \circ t]$	$\to$	$n[s \circ (\uparrow \circ t)]$	*(AssEnv)*	$(s \circ t) \circ u$	$\to$	$s \circ (t \circ u)$
(MapEnv)	$(a \cdot s) \circ t$	$\to$	$a[t] \cdot (s \circ t)$	*(Shift)*	$\uparrow \circ (a \cdot s)$	$\to$	s
(ShiftLift1)	$\uparrow \circ \Uparrow(s)$	$\to$	$s \circ \uparrow$	*(ShiftLift2)*	$\uparrow \circ (\Uparrow(s) \circ t)$	$\to$	$s \circ (\uparrow \circ t)$
(Lift1)	$\Uparrow(s) \circ \Uparrow(t)$	$\to$	$\Uparrow(s \circ t)$	*(Lift2)*	$\Uparrow(s) \circ (\Uparrow(t) \circ u)$	$\to$	$\Uparrow(s \circ t) \circ u$
(LiftEnv)	$\Uparrow(s) \circ (a \cdot t)$	$\to$	$a \cdot (s \circ t)$	*(IdL)*	$id \circ s$	$\to$	s
(IdR)	$s \circ id$	$\to$	s	*(LiftId)*	$\Uparrow(id)$	$\to$	id
(IdEnv)			$a[id]$	$\to$	a		

The rewriting system λEnv.

(Beta)			$(\lambda a)b$	$\to$	$a[b \cdot id]$		
(App)	$(ab)[s]$	$\to$	$(a[s]\ b[s])$	*(VarId)*	$n[id]$	$\to$	n
(FVar)	$1[a \cdot s]$	$\to$	a	*(Closure)*	$a[s][t]$	$\to$	$a[s \circ t]$
(Lambda')	$(\lambda a)[s]$	$\to$	$\lambda(a[1 \cdot (s \circ \uparrow)])$	*(IdL)*	$id \circ s$	$\to$	s
(ShiftId)	$\uparrow \circ id$	$\to$	$\uparrow$	*(Shift)*	$\uparrow \circ (a \cdot s)$	$\to$	s
(AssEnv)	$(s_1 \circ s_2) \circ s_3$	$\to$	$s_1 \circ (s_2 \circ s_3)$	*(MapEnv)*	$(a \cdot s) \circ t$	$\to$	$a[t] \cdot (s \circ t)$
(VarShift1)	$n[\uparrow]$	$\to$	$n+1$	*(VarShift2)*	$n[\uparrow \circ s]$	$\to$	$n+1[s]$
(RVar)			$n+1[a \cdot s]$	$\to$	$n[s]$		

The rewriting system $\lambda\sigma$.

(Beta)			$((\lambda a)^u b)^v$	$\to$	$a[b \cdot id]$		
(App)	$((ab)^u[s])^v$	$\to$	$(a[s]\ b[s])^{u\ or\ v}$	*(VarId)*	$(n^u[id])^v$	$\to$	$n^{u\ or\ v}$
(FVar)	$(1^u[a \cdot s])^v$	$\to$	$a^{u\ or\ v}$	*(Closure)*	$((a[s])^u[t])^v$	$\to$	$a^{u\ or\ v}[s \circ t]$
(Lambda')	$((\lambda a)^u[s])^v$	$\to$	$\lambda(a[1 \cdot (s \circ \uparrow)])^{u\ or\ v}$	*(IdL)*	$id \circ s$	$\to$	s
(ShiftId)	$\uparrow \circ id$	$\to$	$\uparrow$	*(Shift)*	$\uparrow \circ (a \cdot s)$	$\to$	s
(AssEnv)	$(s_1 \circ s_2) \circ s_3$	$\to$	$s_1 \circ (s_2 \circ s_3)$	*(MapEnv)*	$(a \cdot s) \circ t$	$\to$	$a[t] \cdot (s \circ t)$
(VarShift1)	$(n^u[\uparrow])^v$	$\to$	$n+1^{u\ or\ v}$	*(VarShift2)*	$(n^u[\uparrow \circ s])^v$	$\to$	$n+1^{u\ or\ v}[s]$
(RVar)			$n+1^u[a \cdot s]^v$	$\to$	$n^{u\ or\ v}[s]$		

The labeled rewriting system $\lambda\sigma^L$.

ISSN 0249-6399