Origin Functions in 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, focusing especially on functional languages, based on the λ-calculus, and their implementation by term rewriting systems. To implement source-level debugging, one needs to establish a relation between the current state of an execution machine and a position in the source program. In reduction systems this facility is usually provided by the notions of descendance and residuals [Morris68], [Lévy78], [Klop80].

Recent work from Hardin and Lévy [Hardin&Lévy89] and from Abadi, Cardelli, Curien, and Lévy [ACCL90] described term rewriting systems that permit to represent completely the λ-calculus. these results are significant in that previous rewriting systems like CL [Hindley&Seldin86] described only part of the λ-calculus, namely weak reductions. Whereas the papers [Hardin&Lévy89] and [ACCL90] only compare term rewriting systems derivations and λ-calculus derivations in terms of initial and final value, the present work studies these derivations with respect to the notion of residuals.

That a 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 residuals, that is, if the same sub-expressions of the final term are residuals of an expression of the initial term in both settings. In this respect, we show that the two rewriting systems λEnv [Hardin&Lévy89] and $\lambda\sigma$ [ACCL90], which we study, do not represent the λ-calculus faithfully. We show that a simple extension of the notion of residual permits to overcome this difficulty for one of the systems, λEnv and that a more complicated extension permits to have the same result for $\lambda\sigma$. this result provides a way to compare the two systems, that can be added to the investigation provided in work from Curien, Hardin, and Lévy [CHL91].

in: *Colloquium on Trees in Algebra and Programming (CAAP),*
Rennes, France, February, 1992.

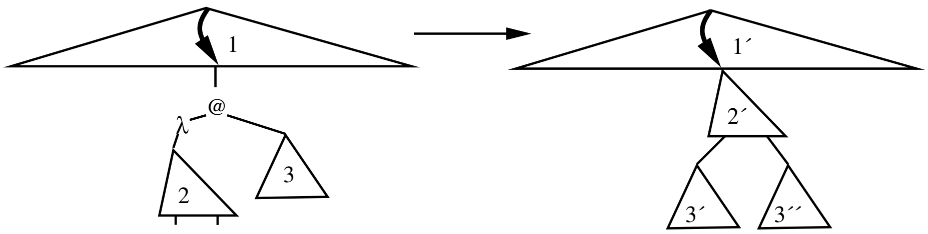

Figure 1. A schematic β-reduction.
(*The stubs below area 2 represent occurrences of the substituted variable*)

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 prefer to use a simple notion of *origin functions* that permits to concentrate on the key aspects of the notions of descendance and residuals which may be hidden by marked derivations: an expression is residual of at most one expression in the initial term, and the residual relation does not depend on the set of names used to mark terms. However, we reuse results that have previously been established using marking terms, for instance [Field90]. Another advantage of origin functions over marked reduction systems is that marks can only give an extensional representation of descendance, whereas one is free to implement origin functions with an intensional representation, as in [Bertot91a]. This may mean significant economies in memory consumption.

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.

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

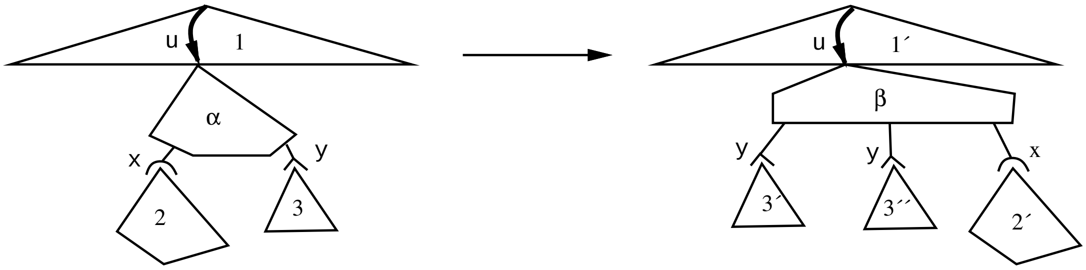

Figure 2. A schematic term rewriting.

appears in α and no variable appears more than once in α. Computing with such rewriting systems 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. The nodes in the pattern β 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 : $\qquad (\lambda(1\ 1))\ a \xrightarrow{\varepsilon} a\ a$

$$D_2: \quad (\lambda(1\ 1))\ a \overset{(\varepsilon, Beta)}{\rightarrow} (1\ 1)[a \cdot id]$$
$$\overset{(\varepsilon, App)}{\rightarrow} (1[a \cdot id]\ 1[a \cdot id])$$
$$\overset{(1, Fvar)}{\rightarrow} a\ 1[a \cdot id]$$
$$\overset{(2, Fvar)}{\rightarrow} a\ a$$

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 $f_1(\varepsilon) = 1.1$ (the empty occurrence, ε, designates the top of a term, whereas 1.1 designates the first child of the first child) and $f_2(\varepsilon)$ 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 step of D_2, whereas the application operator that appears inside the λ-abstraction in the initial term is consumed by the same step. This leads to the following suggestion: why not extend origin functions for term rewritings so that created nodes (inside β in Figure 2) 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 $T \overset{(u,r)}{\longrightarrow} T'$ the rewriting of T at occurrence u using rule $r = \alpha \to \beta$ 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:

$$\begin{aligned} M &::= \lambda M \mid MM \mid n \mid M[s] \\ s &::= s \circ s \mid M \cdot s \mid id \mid \Uparrow(s) \mid \uparrow \end{aligned}$$

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

Informally, the right-hand side of this rule represents the application of the substitution $N \cdot id$ to the term M. The substitution $N \cdot id$ itself represents the operation of replacing the variable index 1 by N and any other variable index n by $n - 1$. All the other rules describe how to compute with 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 $D = T \xrightarrow{S}\!\!\!\!\rightarrow T'$, 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 $D = T \xrightarrow{S}\!\!\!\!\rightarrow T'$ 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 $D' = M \xrightarrow{u.S}\!\!\!\!\rightarrow M[u \leftarrow T']$ 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 $D = (\lambda M)N \xrightarrow{\varepsilon} M[1\backslash N]$, 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 $D = T \xrightarrow{(\varepsilon,r)} T'$ 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 $D = T \xrightarrow{(\varepsilon,r)} T'$ 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

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

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 sketch a proof that these constraints are sufficient (this proof is detailed in [Bertot91b]).

3.1. *A class of Correct Markings.*

The rules of $\mathcal{S}Env$ 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 verifying 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 coinitial and cofinal derivation of λEnv and conversely that for any derivation of λEnv whose initial and final terms are in Λ there also exists a coinitial and cofinal λ-calculus derivation. 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 $$(\lambda 1)((\lambda 1)(\lambda 1)) \xrightarrow{\varepsilon} (\lambda 1)(\lambda 1)$$

D_2 $$(\lambda 1)((\lambda 1)(\lambda 1)) \xrightarrow{2} (\lambda 1)(\lambda 1)$$

The origin functions f_1 and f_2 for these two derivations are different. For example, one has $f_1(\varepsilon) = 2$ and $f_2(\varepsilon) = \varepsilon$. 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.

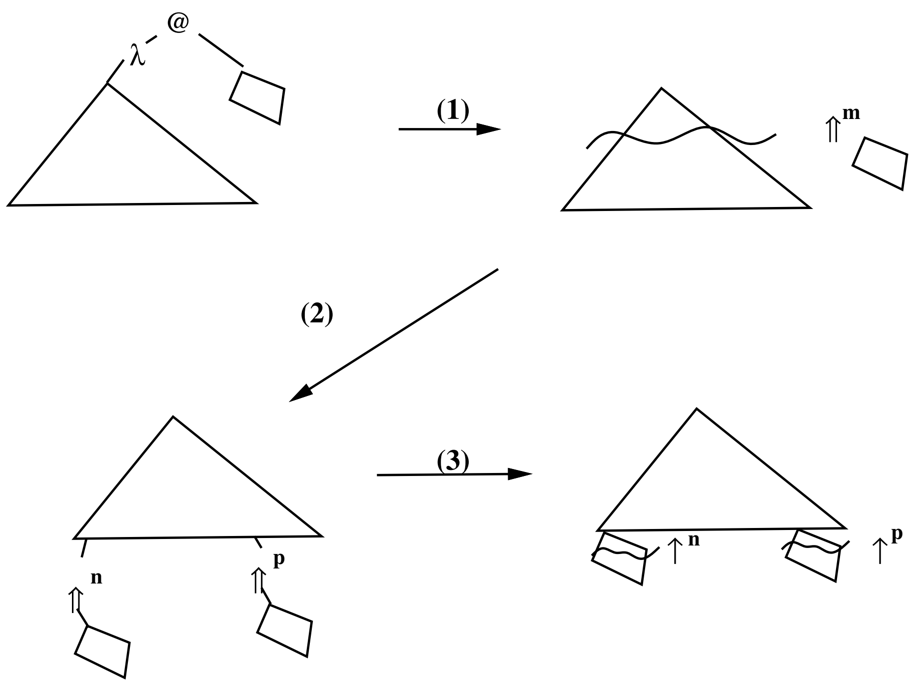

Figure 3. Computation of a β-reduction using a Term Rewriting System.

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 rather easy to establish but its proof is long and tedious. We only sketch this proof here, the exact detail can be found in [Bertot91b]. Basically, to find a derivation of the term rewriting system that performs the same computation as derivation of the λ-calculus, one can simply perform one beta reduction at a time. The computation in that case can be roughly described as in figure 3. It contains three main stages:

(1) One creates a substitution replacing the variable bound in the β-redex by the argument.

(2) One traverses the body of the function part of the β-redex with the substitution, updating this substitution with the $\Uparrow$ operator each time one traverses a λ (rules *App* and *Lambda*). The constraints we have put on the marking ensures that instances of the operators @ and λ in the body are given a correct origin. Variable indexes that do not correspond to the bound variable are either decremented if they are free (rule *RVar*) in the body or restored to their value (rule *FVarLift1*). Here again the constraints put on the marking ensure that these variables are given a correct origin.

(3) On the leaves that correspond the bound variable, the substitution is actually performed, but some updating of the variable indexes corresponding to variables that are free in the argument has to be performed. This updating uses a substitution constructed with the operator $\uparrow$ by the rules *RVarLift1* and *RVarLift2*. We have a second traversal of a term. The constraints put on the marking ensure the origin function restricted to these arguments is the identity function.

Origins for λEnv derivations that perform the same computation as a non elementary λ-calculus derivation are then simply deduced from the fact that origin function of non-elementary derivations are defined the same way for the λ-calculus and for term rewriting systems.

The proof of theorem 2 is simply a proof of confluence, where the confluence has to be extended to origin functions. Problems of confluence have been best studied for term rewriting systems [Huet80]. The simplest way to reuse these results is to define a marked rewriting systems that capture both the semantics of λEnv and the notion of marking functions. This marked rewriting system is defined as follows:

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 $m : Op_\lambda \to Op_\lambda^*$ 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 $erase : \Lambda Env^* \to \Lambda Env$ 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 introduced by the following definition:

Definition. Let λEnv^* be the smallest rewriting system containing λEnv and such that if $r = \alpha \to \beta$ 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 $\alpha' \to \beta'$ is also in λEnv^*. The function $erase$ introduced in the previous definition extends naturally to rules of λEnv^*. For any derivation $D = M \overset{(u,r)}{\to} N$ in λEnv, and any term M' such that $erase(M') = M$, we denote $lift(D, M')$ the derivation $D' = M' \overset{(u,r')}{\to} N'$ 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 $\lambda Env^* - \mathcal{S}Env^*$.

Example. For instance, λEnv^* contains the following rules, where the first two are obtained from *App* and the last two are obtained from *FVar*:

$(ab)[s] \to (a[s]b[s])$
$(ab)^*[s] \to (a[s]b[s])^*$
$1[a \cdot s] \to a$
$1^*[a \cdot s] \to a$

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

Lemma 1. *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:*

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

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 $P \overset{(w,r)}{\to} N$, 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. □

Property P1. *The rewriting system $\mathcal{S}Env^*$ 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 $\mathcal{S}Env$ is noetherian [Hardin&Lévy89] and any derivation of $\mathcal{S}Env^*$ is mapped to a derivation of $\mathcal{S}Env$ 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 $\mathcal{S}Env$ starting from $M[N \cdot id]$ have a marking function equal to the origin function of the corresponding β-reduction.*

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 $\mathcal{S}Env^*$ normalization, then a $Beta^*$ reduction, etc. Corollary 1 simply indicates that derivations in $Sim\beta^*$ have the property needed in theorem 2.

Lemma 2. *any derivation of λEnv^* whose initial and final terms are in Λ^* has the same initial and final term as a derivation in $Sim\beta^*$.*

The proof of this lemma is rather long and technical, and we do not give it here. The precise proof can be found in [Bertot91b]. The following example helps understanding the inherent complication of this proof.

Example. Let us consider the following λEnv^* derivation:

1	$(\lambda(11)^*)((\lambda 1)1)$	$\to (11)^*[((\lambda 1)1^*) \cdot id]$
2		$\to (11)^*[1^* \cdot id]$
3		$\to (1[1^* \cdot id]1[1^* \cdot id])$
4		$\to (1^*1[1^* \cdot id])$
5		$\to (1^*1^*)$

This derivation has the same initial and final term as the following one, which actually is a $Sim\beta^*$ derivation:

1	$(\lambda(11)^*)((\lambda 1)1)$	$\to (11)^*[((\lambda 1)1^*) \cdot id]$
2		$\to (1[((\lambda 1)1^*) \cdot id]1[((\lambda 1)1^*) \cdot id])$
3		$\to (((\lambda 1)1^*)1[((\lambda 1)1^*) \cdot id])$
4		$\to (((\lambda 1)1^*)((\lambda 1)1^*))$
5		$\to (1[1^* \cdot id]((\lambda 1)1^*))$
6		$\to (1^*((\lambda 1)1^*))$
7		$\to (1^*1[1^* \cdot id])$
8		$\to (1^*1^*)$

The second derivation is actually longer than the first one, for various reasons. This makes it difficult to use arguments such as "induction on the length of the derivation" to establish the result. One way to turn around this is to use the notion of parallel derivation of non-overlapping redexes as is already done in [Hardin&Lévy89] to extend the result of confluence from $\mathcal{S}Env^*$ to λEnv^*. This is also the method we use in [Bertot91b] where the exact proof of lemma 2 is presented. by simple transitivity of equality, lemma 2 and corollary 1 give us theorem 2.

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 $$1[M \cdot id] \to M$$

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 $T \overset{(u,FVar)}{\to} T'$, as is shown in the following example.

Example. *Let us consider the two following derivations, where we have used mark to describe origins:*

1 $\quad (\lambda(\lambda 1^*))1) \to (\lambda 1)[1 \cdot id]$
2 $\quad \to \lambda(1^*[1 \cdots ((1 \cdot id) \circ \uparrow)])$
3 $\quad \to \lambda(1^*)$

and

1′ $\quad (\lambda 1)1^* \to 1[1^* \cdot id]$
2′ $\quad 1^*$

The steps 3 and 2′ both use the rule FVar. In the first case, the origin of the result is taken in the variable part of the redex, whereas for the second case, this origin is taken in the value part of the redex.

When computing the marking function for a non-elementary derivation, we need to find a way to choose which function to use 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 $D = M \overset{\emptyset}{\to} M$ then $\mathcal{M}_\sigma(D) = 1_{\mathcal{D}(M)}$.
2. If D, D', and D'' are derivations such that $D = T \overset{S}{\to} T'$, $D' = T \overset{S;(u,r)}{\to} T''$, and $D'' = T' \overset{(u,r)}{\to} T''$ 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:

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

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 theorem 3 is very similar to the proof of theorem 1. We first show that elementary reductions of the λ-calculus and their origin functions can be faithfully represented by derivations of $\lambda\sigma$ and their marking functions. Of course, we have to take care of the two possible uses of the rule *FVar*, but the different cases for the rules *Closure*, *VarShift2*, *RVar*, *VarShift1*, and *VarId* do not appear. The proof of theorem 3 is detailed in [Bertot91b]. Proving theorem 4 can be obtained using a confluence argument identical to the one used for theorem 2. to establish this result we use a more elaborate labeled rewriting system than before. 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.

$$x \text{ or } nil = nil \text{ or } x = x \qquad\qquad (x \text{ or } y) \text{ or } z = x \text{ or } (y \text{ or } z)$$

$$x \text{ or } y = x \qquad \text{if } x \neq nil$$

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 $\mathcal{L} : \mathcal{D}(M) \rightarrow L$, 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 $(M^l)^m = M^{l \text{ or } m}$.

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 $D = M \stackrel{(u,r)}{\rightarrow} N$ of $\lambda\sigma$

and any labeling function $\mathcal{L} : \mathcal{D}(M) \to L$ we note $lift_L(D, \mathcal{L})$ the derivation $D' = \langle M, \mathcal{L}\rangle \overset{(u,r')}{\rightarrow} N'$ 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 $\lambda\sigma^L - \{Beta\}$.

Example. *The following derivation is an example derivation of the system $\lambda\sigma^L$:*

1	$(\lambda^b(\lambda^d(1^f 2^g)^e 1^h)^c 1^i)^a$	$\to (\lambda^d(1^f 2^g)^e 1^h)^c[1^i \cdot id]$
2		$\to (\lambda^d(1^f 2^g)^e[1^i \cdot id]1^h[1^i \cdot id])^c$
3		$\to (\lambda^d((1^f 2^g)^e[1 \cdot ((1^i \cdot id) \circ \uparrow)])1^h[1^i \cdot id])^c$
4		$\to (\lambda^d(1^f[1 \cdot ((1^i \cdot id) \circ \uparrow)]2^g[1 \cdot ((1^i \cdot id) \circ \uparrow)])^e 1^h[1^i \cdot id]$
5		$\to (\lambda^d(1^f 2^g[1 \cdot ((1^i \cdot id) \circ \uparrow)])^e 1^h[1^i \cdot id])^c$
6		$\to (\lambda^d(1^f 1^g[(1^i \cdot id) \circ \uparrow])^e 1^h[1^i \cdot id])^c$
7		$\to (\lambda^d(1^f 1^g[1^i[\uparrow] \cdot (id \circ \uparrow)])^e 1^h[1^i \cdot id])^c$
8		$\to (\lambda^d(1^f(1^i[\uparrow])^g)^e 1^h[1^i \cdot id])^c$
9		$\to (\lambda^d(1^f 2^i)^e 1^h[1^i \cdot id])^c$
10		$\to (\lambda^d(1^f 2^i)^e 1^i)^c$

Note how the meaning attached to the operator or affects the way labels are transmitted in reduction steps 5, 8, 9, and 10.

This labeled rewriting system permits to represent computations of origins for $\lambda\sigma$ exactly as λEnv^* did λEnv. We formally express this property with the following lemma, whose detailed proof is given in [Bertot91b].

Lemma 3. *For any term $M \in \Lambda$, for any labeling function $\mathcal{L} : \mathcal{D}(M) \to \mathcal{A}$, 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:*

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

As for the proof of theorem 2, we need a confluence property to establish theorem 4. Such a property can be found, although it is weaker than for λEnv^*.

Definition. A term of $\Lambda\sigma^L$ is *well labeled* if it contains no occurrence of an unlabeled operator of Op_λ.

Property P2. *The rewriting system σ^L is confluent on well labeled ground terms.*

Most critical pairs of the system are simply confluent. For the others, a simple analysis by cases on variable instances generally gives the result.

We then have the following lemma, that mimics lemma 2 for the rewriting system $\lambda\sigma^L$. This lemma uses a set of derivations $Sim\beta^L$ that corresponds to the set $Sim\beta^*$ used for λEnv^*.

Lemma 4. *Any derivation of $\lambda\sigma^L$ that begins and finishes on terms of Λ^L has the same initial and final term as a derivation of $Sim\beta^L$.*

To prove this result, we have to examine the critical pair introduced by adding the rule *Beta* to σ^L. The problems encountered can be sketched as follows.

We examine the problem of proving that the critical pair between *Beta* and *App* is confluent on ground terms. We recall that these rules have the following form:

Beta	$((\lambda a)^u b)^v \to a[b \cdot id]$
App	$((ab)^u[s])^v \to (a[s]b[s])^{u\ or\ v}$

The critical pair between these rules appears when reducing the term $(((\lambda a)^u b)^v[s])^w$. The two completely reduced outcomes are the terms $a^w[b[s]\cdot s]$ and $a[b[s]\cdot(s\circ id)]$. To establish a confluence result, we have to express that the label w on top of a in the first term will not be used in the final term of σ^L-normalizing derivation and second that $(s\circ id)$ and s are equivalent substitutions. The first statement comes from a lemma given in [Bertot91b] that expresses that any derivation starting with a well labeled term finishes in a well labeled term. For this reason the term a can be normalized in a term a'. Since this term σ^L-normal, its head operator is in Op_λ and since it is well labeled, we have $a'^w = a'$. The second statement can be proved a case analysis on the head operator using a lemma expressing that for any term t, $t[id]$ and t reduce to a same value; this lemma can be proved by induction on the size of t.

Lemma 4 has consequences for $\lambda\sigma^L$ that are equivalent to the consequences of lemma 2 for λEnv^*, and among them comes theorem 4.

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”, *Proceed-*

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

The rewriting system λEnv.

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

The rewriting system $\lambda\sigma$.

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

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