Linearization by Program Transformation

Sandra Alves and Mário Florido

University of Porto
Department of Computer Science & LIACC
R. do Campo Alegre 823, 4150-180 Porto, Portugal
e-mail: {sandra,amf}@ncc.up.pt

Abstract. We identify a restricted class of terms of the lambda calculus, here called weak linear, that includes the linear lambda-terms keeping their good properties of strong normalization, non-duplicating reductions and typability in polynomial time. The advantage of this class over the linear lambda-calculus is the possibility of transforming general terms into weak linear terms with the same normal form. We present such transformation and prove its correctness by showing that it preserves normal forms.

1 Introduction

Linear programs are simple. Concerning implementation issues, for linear programs one may safely inline the term bound to any variable, or safely update structures in place. Every linear term in the λ-calculus is β-strongly normalizing (i.e., there is no infinite β-reduction sequence starting from M), every β-reduction of a linear term is non-duplicating and every closed linear term is typable in the simple type system [11], thus it is typable in polynomial time.

Here linear programs are modeled by *linear* λ-terms: λ-terms M such that for each subterm $\lambda x.P$ of M, x occurs free in P at most once.

Now consider the following question: is there a way of simulating the standard λ-calculus by the linear λ-calculus? If simulation means transforming the original standard term into a linear term with the same normal form then it is not possible, in general, to define such transformation. This is shown by the next simple example: it is not possible to transform the term (in normal form) $\lambda x.xx$ into a linear term with the same normal form. This happens for any non-linear normal form.

In this paper we address the following problem: is there a restricted class of λ-terms, with the same nice properties of the linear λ-calculus and such that we can simulate the standard λ-calculus by terms of that class?

We show that there is a restriction with these properties, which we here call the *weak linear λ-calculus*. A λ-term M is *weak linear* if in any reduction sequence of M, when there is a contraction of a β-redex $(\lambda x.P)Q$, then x occurs free in P at most once, i.e., when a function $\lambda x.P$ is applied, its formal parameter x must occur at most once in the function body. For example the term $\lambda x.xx$ is weak linear because it is a non-linear λ-abstraction which is never applied. The term $(\lambda x.xx)I$ is not weak linear because it has a redex where the function is not linear. For weak linear terms, only functions that can be applied to an argument in the reduction process are required to be linear.

Notice that our definition does not refer only to β-redexes $(\lambda x.P)Q$ that are subterms of the original term M, but to abstractions $\lambda x.P$ that are going to be the function part of a β-redex in the reduction of M. For example the term $M \equiv ((\lambda x.x)(\lambda x.xx))k$ is not weak linear although it does not have any subterm of the form $(\lambda x.P)Q$ with x occurring more than once in P. The problem is that there is a redex of this form (in this case, $(\lambda x.xx)k$), in a reduction sequence from M.

The main contributions of this paper are the following:

- A restricted class of λ-terms, here called the *weak linear λ-calculus* with the same basic properties of the linear λ-calculus. Here we show that weak linear terms are strong normalizing and typable in polynomial time.

– A transformation of general terms into weak linear terms preserving normal forms. To deal with transformation of redexes which will appear during the reduction process (the *virtual redexes*) our transformation uses *legal paths*, [4, 5], because this notion provides a formal characterization of the intuitive notion of *virtual redex*. This contribution is also significant for the methodology it develops. What we set up is a new use of legal paths for complex term transformation.

Let us quickly review the existing literature on the question. A linearization of the λ-calculus was made by Kfoury [13]. He embedded the λ-calculus into a larger calculus, denoted $\Lambda^\wedge$ with a new notion of reduction, denoted $\beta^\wedge$. In the new calculus $\Lambda^\wedge$, in every function application, an argument is used at most once. He also defined the notion of *contraction* of a term in the new calculus, giving a λ-term. This last notion gives a way of transforming terms in the new calculus into terms of the λ-calculus, however, it was not presented a direct definition of a transformation of λ-terms into terms of the new calculus. The relation between the two calculus was made indirectly saying that the *well-formed* terms of the new calculus are the ones for which there is a *contraction* in the λ-calculus. Our algorithm is defined directly as a transformation from λ-terms to *weak linear* terms and simulating the λ-calculus by a subset of the λ-calculus and not by a non-standard calculus. The problem was also discussed in [9] where it was established a relation, not a transformation, between terms typable in an intersection type system and linear terms. Types played a central role is the definition of this relation and, because the mapping was on the linear λ-calculus, β-normal forms were not preserved.

The type system used in our paper was used (with minor differences) before in [14] and [1], as a restricted form of intersection type inference for terms resulting from a simplification process of arbitrary terms. Finally we remark that our definition of *linear* term follows [13], but some people call this class of terms *affine*, and use the word *linear* for terms $\lambda x.M$ where x occurs free exactly once in M. The work presented here was previously presented in [2]. A complete report with detailed proofs of every theorem can be found in [3].

In the rest of the paper we assume that the reader is familiar with the λ-calculus. A standard reference for this area is [6]. A good survey on the application of λ-calculus to programming language technology can be found in [7].

We start in Section 2 with the definition of weak linear lambda terms. In section 3 we present a type system and a type inference algorithm for the weak linear calculus. In section 4 we present a transformation from λ-terms into weak linear λ-terms, and prove that it is correct in the sense that it preserves normal forms. Finally we conclude and outline some future work in section 5.

2 The Weak Linear Lambda Calculus

We start this section by giving some brief notions on λ-calculus.

Definition 1. *Let x range over an infinite set of variables. The set of λ-terms, Λ is defined as:*

$$M, N \in \Lambda ::= x \mid \lambda x.M \mid MN$$

As usual the set of λ-terms is quotiented by α-conversion. We use the usual notation of reduction

$$(\lambda x.M)N \rightarrow_\beta M[N/x]$$

As usual, $FV(M)$ denotes the set of free variables of M. In the rest of the paper we follow the Barendregt variable naming convention, that is, we assume that no variable is bound more than once, and that it is impossible for a variable to occur both free and bound in a term.

Definition 2. *The length of a term M is defined as follows:*
– $\mathit{length}(x) = 1$;
– $\mathit{length}(MN) = \mathit{length}(M) + \mathit{length}(N) + 1$;
– $\mathit{length}(\lambda x.M) = \mathit{length}(M) + 1$.

Definition 3 (Weak linear terms). *A λ-term M is* weak linear *iff every redex $(\lambda x.P)Q$, in the reduction graph of M (consisting of every reduction sequence from M), is such that x occurs at most once in P.*

Definition 4. *If M is strongly normalizable, the maximal depth of a derivation from M is called the reduction depth of M, and denoted depth(M).*

Lemma 1. *For weak linear λ-terms, contracting a redex reduces the length of a term.*

Corollary 1 (Strong normalization). *Let M be a weak linear λ-term. Then depth(M) $\leq$ length(M). Thus M is strongly normalizable.*

3 Type Inference for the Weak Linear Lambda Calculus

Here we present a type system that types every weak linear term. Note that closed linear terms are typed in the Simple Type System [12], but the same does not happen with weak linear terms. In this case functions which are never applied may not be linear. For example the term $\lambda x.xx$ is weak linear and it is not typable in the Simple Type System. The type system described in this section, here called T_W type system, is based on intersection types [8]. Note that the previous term, $\lambda x.xx$, is typed by intersection types with type $((\alpha \to \beta) \cap \alpha) \to \beta$. We use intersections only to type abstractions, and when typing applications the function part cannot have a domain denoted by an intersection. This is enough to type every weak linear term keeping the decidability of the type inference problem. The type system can also be used to establish decidability for weak linear terms.

3.1 A Type System for Weak Linear Terms

Definition 5. *An infinite sequence of type-variables is assumed to be given.* Intersection types *are expressions defined thus:*

1. *each type-variable is a type;*
2. *if σ and $\tau_1 \ldots \tau_n$ are types (for $n \geq 1$) then $(\tau_1 \cap \cdots \cap \tau_n \to \sigma)$ is a type.*

In the previous definition $\cap$ is assumed to be an associative, commutative and idempotent operator.

Definition 6. *A* type environment *is a finite set of pairs of the form $x : \tau$, where x is a term variable and τ is a type.*

Definition 7. *The T_W system is defined by:*

$$VAR \qquad \{x : \sigma\} \vdash x : \sigma$$

$$ABS\text{-}I \quad \frac{A \cup \{x : \tau_1, \ldots, x : \tau_n\} \vdash M : \sigma}{A \vdash \lambda x.M : \tau_1 \cap \cdots \cap \tau_n \to \sigma} \quad \text{if } x \in FV(M) \quad (a)$$

$$ABS\text{-}K \quad \frac{A \vdash M : \sigma}{A \vdash \lambda x.M : \tau \to \sigma} \quad \text{if } x \notin FV(M)$$

$$APP \quad \frac{A_1 \vdash M : \tau \to \sigma \quad A_2 \vdash N : \tau}{A_1 \cup A_2 \vdash MN : \sigma}$$

(a) If $x : \tau_1, \ldots, x : \tau_n$ are all and nothing but statements about x on which $A \cup \{x : \tau_1, \ldots, x : \tau_n\} \vdash M : \sigma$ depends. The symbol $\cup$ denotes the usual set union operation.

$M : \sigma$ is derivable from an environment A in the T_W type system, notation $A \vdash M : \sigma$, if and only if it is obtained using the previous rules. Note that in T_W intersections only appear in the *ABS-I* rule, thus in type derivations, intersection types can only appear in the types of abstractions which are not applied.

Example 1. In system T_W we have $\vdash (\lambda x.xx) : (\alpha \cap (\alpha \to \beta)) \to \beta$ but $(\lambda x.xx)(\lambda x.x)$ is not typable.

Theorem 1. *Every weak linear term M is typable in system T_W.*

Note that the converse does not hold. For example, $(\lambda fx.f(fx))(\lambda x.x)$ is typable but it is not weak linear.

3.2 A Type Inference Algorithm

The type inference algorithm presented here is a generalization of the Hindley's algorithm for Simple Types ([10], [12]). A brief sketch of a similar system was presented before in [14]. The main difference to the Simple Type System is that type declarations for the same variable in the environment may not be unique, and abstractions are typable even when types of their formal parameter do not unify.

Definition 8. *Let UNIFY be Robinson's unification algorithm [16]. Given a λ term M, we define the function $\mathcal{I}(M) = (\Gamma, \tau)$, where Γ is a type environment and τ is a type, thus:*
1. *If $M = x$ then $\mathcal{I}(M) = (\{x : \alpha\}, \alpha)$ where α is a type variable.*
2. *If $M = \lambda x.N$ then:*
   - *If $\mathcal{I}(N) = (\Gamma', \tau)$ and $x \notin Subjects(\Gamma')$, then $\mathcal{I}(\lambda x.N) = (\Gamma', \alpha \to \tau)$ where α is a new type variable.*
   - *If $\mathcal{I}(N) = (\Gamma', \tau)$ and $\Gamma'(x) = \{\tau_1, \ldots, \tau_n\}$, then $\mathcal{I}(\lambda x.N) = (\Gamma'_x, \tau_1 \cap \cdots \cap \tau_n \to \tau)$.*
3. *If $M = M_1 M_2$ then $\mathcal{I}(M) = (S(\Gamma_1 \cup \Gamma_2), S(\alpha))$ where:*
   - *$\mathcal{I}(M_1) = (\Gamma_1, \tau_1)$;*
   - *$\mathcal{I}(M_2) = (\Gamma_2, \tau_2)$;*
   - *$S = UNIFY(\tau_1, \tau_2 \to \alpha)$ (α is a new type variable).*

Theorem 2.
1. *$\mathcal{I}(M) = (\Gamma, \sigma)$ if and only if M is typable in system T_W.*
2. *$\mathcal{I}(M)$ terminates in time polynomial in the size of M.*

Theorem 3. *Give a λ-term M it is decidable to know if M is weak linear.*

4 Transformation into Weak Linear Terms

Let us see what means to transform a λ-term into a *weak linear* term. Consider the following example: suppose one wants to define the weak linear version of the term $(\lambda xy.xy)(\lambda z.zz)(\lambda w.w)$. In this term the only variable which occurs more than once is z. Thus we must linearize $(\lambda z.zz)$ to get the term $(\lambda z_1 z_2.z_1 z_2)$. But $(\lambda z.zz)$ will have y as an argument after one reduction step. Thus y in $(\lambda xy.xy)$ has to be copied and we get the term:

$$(\lambda xy.xyy)(\lambda z_1 z_2.z_1 z_2)(\lambda w.w)$$

Now, a variable which occurred once in the original term occurs twice in the new term, thus the linearization process has to go on, linearizing $(\lambda xy.xyy)$ to obtain $(\lambda xy_1 y_2.xy_1 y_2)$. Notice that y will be replaced by $\lambda w.w$ in the original version of the term, thus, as y was replaced by two new parameters, we have to duplicate $(\lambda w.w)$ in the resulting term to get the final term:

$$(\lambda xy_1 y_2.xy_1 y_2)(\lambda z_1 z_2.z_1 z_2)(\lambda w.w)(\lambda w.w)$$

Notice that y will be replaced by $\lambda w.w$ after one reduction step. Thus the transformation algorithm has to know in advance that subterms of the form $(\lambda x.M)$ and N are going to be the function and argument part of a redex in the future, i.e. $(\lambda x.M)N$ is a *virtual redex*. To deal with this complicated issue we make use of *legal paths*. Legal paths were introduced by Asperti and Laneve [5] as a characterization based on paths of Lévy's redex families in the context of optimal reductions for the λ-calculus. They provide a static characterization of virtual redexes and revealed to be quite useful for program transformation. As far as we know this is the first work relating the two subjects.

4.1 The Labeled λ-Calculus and Legal Paths

The Labeled λ-Calculus The labeled λ-calculus is an extension of the λ-calculus, proposed by Lévy in [15]. In the rest of the paper, we will use $x, y, z, \ldots$ to range over variables, $a, b, c, \ldots$ to range over atomic labels, $l, l_1, l_2, \ldots$ to range over labels, and $\varphi, \psi, \phi, \ldots$ to range over paths.

Definition 9. *Let a range over an infinite set of atomic labels. The set of labels, L is defined as:*

$$l_1, l_2 \in L ::= a \mid l_1 l_2 \mid \underline{l_1} \mid \overline{l_2}$$

Definition 10. *Let x range over an infinite set of variables $\mathcal{V}$, and l over an infinite set of labels L. The set of labelled λ-terms, $\Lambda^L_{\mathcal{V}}$ is defined as:*

$$M, N \in \Lambda^L_{\mathcal{V}} ::= x^l \mid (MN)^l \mid (\lambda x.M)^l$$

Labeled β-reduction is the following rule (note that $l_0 \cdot (T)^{l_1} = (T)^{l_0 l_1}$):

$$((\lambda x.M)^{l_0} N)^{l_1} \to l_1 \cdot \overline{l_0} \cdot M[\underline{l_0} \cdot N/x]$$

where the label l_0 is the *degree* of the redex $((\lambda x.M)^{l_0} N)^{l_1}$

Labels provide an approach to the notion of computation as a travel along a path. Note that every label trivially defines a path in the syntactic tree of the term.

Definition 11. *If l is a label of an edge generated along some reduction from M, the* path *of l in M is inductively defined as follows:*

$$\begin{aligned} path(a) &= a \\ path(l_1 l_2) &= path(l_1) \cdot path(l_2) \\ path(\overline{l}) &= path(l) \\ path(\underline{l}) &= (path(l))^r \end{aligned}$$

where $\varphi_1 \cdot \varphi_2$ means concatenation of the two paths (in the following we will sometimes omit $\cdot$), and φ^r is the path obtained by reversing φ.

Example 2. $M = ((\lambda x.((x^e(\lambda y.(y^h y^i)^g)^f)^d v^j)^c)^b(\lambda w.(\lambda z.(w^n(w^p z^q)^o)^m)^l)^k)^a$ has the graph representation given by Figure 1.

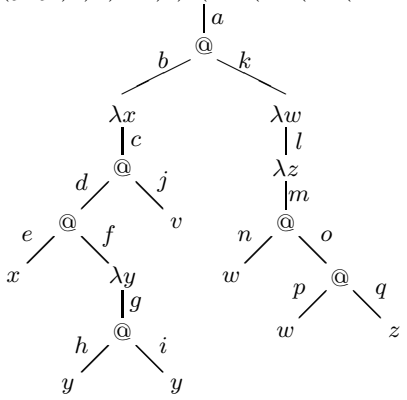

Fig. 1. Labeled λ-term

Legal Paths Different degrees of redexes correspond to different paths in the original term. Legal paths are a characterization of paths yield by degrees. Legal paths are obtained by suitably constraining another type of paths which are the well balanced paths (wbp). (All the definitions and results concerning legal paths, presented in this sub-section, can be found in [5].)

Definition 12. *Well balanced paths are inductively defined in the following way (see Figure 2):*

- (***base case***)*The function edge of any application, is a* well balanced path.
- (***λ-composition***) *Let ψ be a wbp of type @-x whose ending variable is bound to a λ-node **c** and φ be a wbp of type @-λ coming into **c**. Then $\psi.(\varphi)^r.u$ is a wbp, where u is the argument edge of the initial node of φ;*
- (***@-composition***)*Let ψ be a wbp of type @-@ ending into a node **d** and φ be of type @-λ leading from **d** to some λ-node **c**. Then $\psi.\varphi.u$ is a wbp, where u outgoes **c** towards its body.*

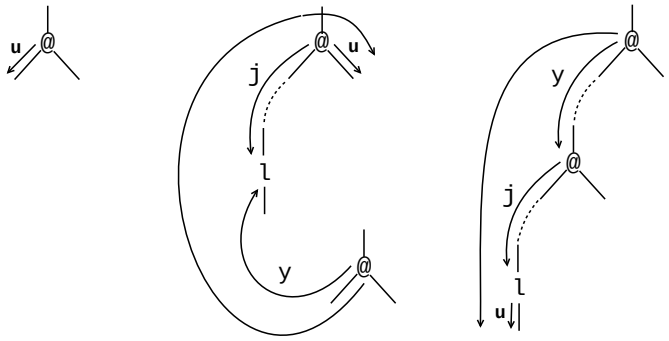

Fig. 2. Well balanced paths

The type @-? where ? can be λ, @ or x (variable) is determined by the type of the node where the wbp ends.

Example 3. The set of wbp (of the form *path* : *type*) of the λ-term M of Figure 1 is given by:
Initial paths:

$$\{b : @\text{-}\lambda, d : @\text{-}@, e : @\text{-}x, h : @\text{-}x, n : @\text{-}x, p : @\text{-}x\}$$

Applying (λ-*composition*) to the initial paths, and (@-*composition*) to the new paths we get,

$$\{e \cdot b \cdot k : @\text{-}\lambda, d \cdot e\, b\, k \cdot l : @\text{-}\lambda\}$$

Again by (λ-*composition*) and (@-*composition*) we get

$$\{n \cdot k\, b\, e \cdot f : @\text{-}\lambda, p \cdot k\, b\, e \cdot f : @\text{-}\lambda\}$$

After one more iteration

$$\{h \cdot f\, e\, b\, k\, n \cdot o : @\text{-}x, h \cdot f\, e\, b\, k\, p \cdot q : @\text{-}x\}$$

Finally

$$\{h\, f\, e\, b\, k\, p\, q \cdot l\, k\, b\, e\, d \cdot j : @\text{-}x\}$$

Thus, the set of wbp of type @-λ is given by:

$$\{b : @\text{-}\lambda, e \cdot b \cdot k : @\text{-}\lambda, d \cdot e\, b\, k \cdot l : @\text{-}\lambda, n \cdot k\, b\, e \cdot f : @\text{-}\lambda, p \cdot k\, b\, e \cdot f : @\text{-}\lambda\}$$

Note that, if we imagine that bound variables (x) are explicitly connected to their binders (λx), these paths are actual paths in the graph representation of M.

We now present two other kinds of paths in a term needed to define legal paths.

Definition 13. *Let φ be a wbp.*

- (v-***cycles***) *Let* v *be the label of a variable edge. A v-cycle (over* v*) is a cyclic subpath of the form* $\mathrm{v}\lambda(\varphi)^r @\psi @\varphi\lambda\mathrm{v}$ *where φ is a wbp and ψ is a @-cycle.*
- (@-***cycles***) *A @-cycle, over an @ node with argument subterm N, is a subpath ψ that starts and ends with the argument edge of the @-node, and composed of subpaths internal to the argument N and v-cycles over free variables of N. A particular case of @-cycle is a cycle starting from and ending to the argument edge p of a @-node (the negative auxiliary port), and internal to the argument N of the application (i.e. not traversing variables which are free in N).*

Example 4. The wbp $h\, f\, e\, b\, k\, p\, q\, l\, k\, b\, e\, d\, j$ of type @-x of example 3 has a v-cycle $e\lambda b@k\, p\, q\, l\, k@b\lambda e$ and a @-cycle $@k\, p\, q\, l\, k@$.

Proposition 1. *Let φ be a wbp with a @-cycle $@\psi@$. Then φ can be uniquely decomposed as*

$$\zeta_1\lambda\zeta_2@\psi@(\zeta_3)^r\lambda\zeta_4$$

where both ζ_2 and ζ_3 are wbp's. The paths ζ_2 and ζ_3 are called the call *and* return *paths of the @-cycle ψ. The last label of ζ_1 and the first label of ζ_4 are named the* discriminants *of the call and return paths, respectively.*

Definition 14. *A wbp is a* legal path *if and only the call and return paths of* any @*-cycle are one the reversed of the other and their discriminants are equal.*

Example 5. The wbp's of example 3 are all legal. Notice that in the only path having a @-cycle $h\,f\,e\,b\,k\,p\,q\,l\,k\,b\,e\,d\,j$, the call and return paths of the cycle are both b (thus one is the reversed of the other), and the discriminants are both e.

Theorem 4. *Every path yield by the degree of a redex is a legal path.*

Theorem 5. *For any legal path φ of type @-λ in a term M, there exists a degree l of a redex originated along some reduction of M such that $path(l) = \varphi$.*

4.2 Term Transformation

Here we present a transformation from arbitrary λ-terms into weak linear terms. We first present some lemmas needed when showing the correctness of the transformation.

Lemma 2. *M is a strongly normalizable term, iff the set $\mathcal{LP}$ of legal paths of type @-λ in M is a finite set.*

Definition 15. *Let M be a λ-term, $\mathcal{LP}$ be the set of legal paths of M, and S_1 the paths corresponding to redexes in M. We define the chain of dependences between legal paths of type @-λ in $\mathcal{LP}$ as $S_1 \gg S_2 \gg \cdots \gg S_n \gg \cdots$ such that the legal paths in S_i are build by* λ-composition *from the paths in the sets $S_1, \ldots, S_{i-1}$ and* @-composition *from the paths in S_i.*

Definition 16. *Let M be a λ-term, and $\mathcal{LP}$ be the set of legal paths of M. The function*

$$next\_non\_linear(M)$$

returns a pair of labels (l, k), where l is the abstraction node where the next non linear legal path, of type @-λ, in the chain of legal paths built from $\mathcal{LP}$, ends and k is the function edge of the application node where it starts. If every legal path is linear, $next\_non\_linear(M) = \perp$.

Basically this function returns the labels which identify the next non-linear virtual redex.

Example 6. For the λ-term M in Figure 1, with the set $\mathcal{LP}$, of legal paths of type @-λ, given in example 3 the chain of legal paths is $\{b\} \gg \{e \cdot b \cdot k,\ d \cdot e\,b\,k \cdot l\} \gg \{n \cdot k\,b\,e \cdot f,\ p \cdot k\,b\,e \cdot f\}$, thus the result of $next\_non\_linear(\mathcal{LP})$ is (k, e).

Lemma 3. *If M is a λ-term and $(l, k) = next\_non\_linear(M)$, then the only application node such that there is a legal path φ of type @-λ ending in l is the application with function edge, labeled k.*

Example 7. Consider the chain of legal paths of type @-λ in example 6. We have $next\_non\_linear(M)$ $= (k, e)$, and the application node whose function edge is e is the only application node for which there is a legal path φ ending in k.

Definition 17 (one step of the transformation). *Let M be a λ-term. We define the term $\mathcal{L}(M)$, that results from linearizing the next non linear abstraction $(\lambda x.P)^l$ of M. Let n be the number of occurrences of x in P:*

$$\mathcal{L}(M) = \begin{cases} M & \text{if } next\_non\_linear(M) = \perp \\ M_l & \text{otherwise} \end{cases}$$

where

- $(l, k) = next\_non\_linear(M)$;
- $M' = replace(l, M)$, *where* $replace(l, M)$ *replaces* $(\lambda x.P)^l$ *in* M *by* $(\lambda x_1 \ldots (\lambda x_n.P^*)^{l_n} \cdots)^{l_1}$, *and* P^* *results from replacing the* i^{th} *occurrence of* x *in* P *by the fresh variable* x_i $(i = 1, \ldots, n)$.
- $M_l = replace\_n(k, n, M')$, *where* $replace\_n(k, n, M')$ *is the function that replaces the term* $(Q^k N^m)^j$ *by* $((\cdots(Q^{k_1} \underbrace{N^{m_1})\cdots)^{k_n} N^{m_n})^j}_{n \text{ times}}$ *(see Figure 3).*

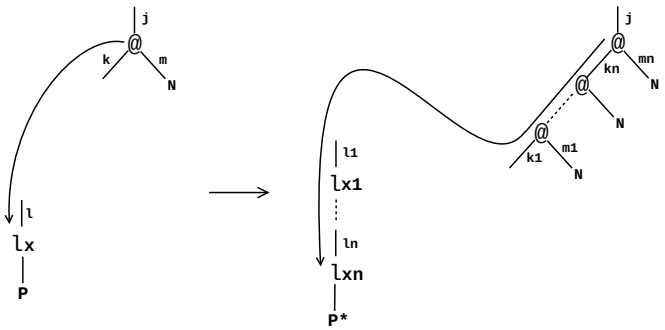

Fig. 3. Expansion of an abstraction

Remark: Note that the labels in each copy of N are such that, if N has a label c in M, there is a label c_i in the i^{th} copy of N in M_l.
Example 8. Let M be the λ-term in Figure 1. The term that results from expanding the first non linear path $(e \cdot b \cdot k)$ in M, $\mathcal{L}(M)$ is given by Figure 4.

Definition 18 (Transformation into weak linear terms). *Let M be a λ-term, and $\mathcal{LP}$ the set of legal paths in M. We define the following function:*

$$\mathcal{T}(M) = \begin{cases} M & \text{if } all\_linear(M) \\ \mathcal{T}(\mathcal{L}(M)) & \text{otherwise} \end{cases}$$

The function $all\_linear(M)$ returns true *if all the legal paths of type @-λ in M, end in a linear abstraction, and* false *otherwise.*

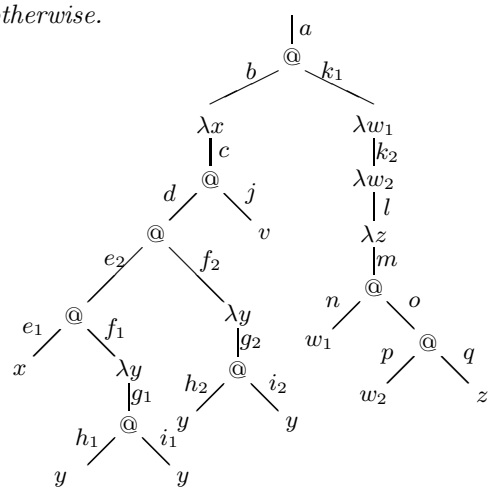

Fig. 4. Term M, of Figure 1, after one step of transformation

Example 9. Let $\Delta = \lambda y.yy$ and $D = \lambda y_1 y_2 . y_1 y_2$. Let $M = (\lambda x.x\Delta v)(\lambda wz.w(wz))$ be the term represented in Figure 1. Let us follow the transformation in some detail. We will omit the labels of the terms, except the ones needed to follow the example. We first linearize the abstraction $(\lambda wz.w(wz))^k$ and duplicate the argument of x^e corresponding to the first non-linear abstraction on the chain of paths in example 6, $(e \cdot b \cdot k)$, to get the term in Figure 4.

$$\mathcal{T}(M) = \mathcal{T}((\lambda x.x\Delta^{f_1}\Delta^{f_2}v)(\lambda w_1.(\lambda w_2 z.w_1(w_2 z))^{k_2})^{k_1})$$

After one more step, linearizing Δ^{f_1}, we get:

$$\mathcal{T}((\lambda x.xD\Delta v)(\lambda w_1 w_2 z.w_1(w_2 z)(w_2 z)))$$

Now the abstractions labeled by k_2 and l in $(\lambda w_2.(\lambda z.w_1(w_2 z)(w_2 z))^l)^{k_2}$ which were linear, became non-linear thus, after a one more step we get:

$$\mathcal{T}((\lambda x.xD\Delta\Delta v)(\lambda w_1 w_2 w_3 z.w_1(w_2 z)(w_3 z)))$$

Linearizing $\lambda z.w_1(w_2 z)(w_3 z)$ we get

$$\mathcal{T}((\lambda x.xD^{f_1}\Delta^{f_{21}}\Delta^{f_{22}}vv)(\lambda w_1 w_2 w_3 z_1 z_2.w_1(w_2 z_1)(w_3 z_2)))$$

The next non-linear abstraction is the copy of Δ labeled by f_{21}, thus we get:

$$\mathcal{T}((\lambda x.xDD\Delta vv)(\lambda w_1 w_2 w_3 z_1 z_2.w_1(w_2 z_1 z_1)(w_3 z_2)))$$

The abstraction $(\lambda z_1 z_2.w_1(w_2 z_1 z_1)(w_3 z_2)))$ becomes non-linear, thus, after a few more steps, we get:

$$\begin{aligned}
&= \mathcal{T}((\lambda x.xDD\Delta vvv)(\lambda w_1 w_2 w_3 z_1 z_3 z_2.w_1(w_2 z_1 z_3)(w_3 z_2))) \\
&= \mathcal{T}((\lambda x.xDDDvvvv)(\lambda w_1 w_2 w_3 z_1 z_3 z_2 z_4.w_1(w_2 z_1 z_3)(w_3 z_2 z_4))) \\
&= (\lambda x.xDDDvvvv)(\lambda w_1 w_2 w_3 z_1 z_2 z_3 z_4.w_1(w_2 z_1 z_2)(w_3 z_3 z_4))
\end{aligned}$$

Now all the legal paths of type @-λ are linear, thus the transformation terminates. Note that $nf(M) = (vv)(vv) = nf(\mathcal{T}(M))$. As we shall prove latter, this happens for every term.

4.3 Correctness

Here we show that the transformation preserves normal forms. This relies on the next lemma, which basically, relates the legal paths of the original term and of the transformed term.

Lemma 4. *If $\mathcal{L}(M) = M_{\mathcal{L}}$, let $(l,k) = next\_non\_linear(M)$ and N the subterm argument of the application node having k as function edge, then the set $P_{\mathcal{L}}$ of legal paths of $M_{\mathcal{L}}$ is such that:*

1. *If φ is a legal path internal to N, then φ_i $(i = 1 \ldots n)$ is a legal path in the i^{th} copy of N in $M_{\mathcal{L}}$;*
2. *If φ is a legal path with a @-cycle in N (starting and ending outside N), then φ' is a legal path with a cycle in a copy of N in $M_{\mathcal{L}}$;*
3. *If φ is a legal path not internal but starting and ending in N, then φ' is a legal path starting and ending in a copy of N in $M_{\mathcal{L}}$;*
4. *If φ is a legal path starting/ending in N, then φ' is a legal path starting/ending in a copy of N in $M_{\mathcal{L}}$, and ending/starting in the same edge;*
5. *If $\varphi = k\psi l$ is the legal path in M of type @-λ ending in $(\lambda x.P)^l$, then in $M_{\mathcal{L}}$, there are n paths of type @-λ ending respectively in $(\lambda x_1 \ldots x_n.P^*)^{l_1}$, $(\lambda x_2 \ldots x_n.P^*)^{l_2}, \ldots, (\lambda x_n.P^*)^{l_n}$, and starting respectively in $k_1, k_2, \ldots, k_n$, $(n \geq 1)$;*
6. *If φ is a legal path in M of any type, external to N, then there is a legal path of the same type in $M_{\mathcal{L}}$, starting and ending in the same edges.*

Example 10. Let M be the labeled term in Figure 1, and $M_{\mathcal{L}}$ (the term obtained from M after one step of transformation) in Figure 4. Given the legal path of M, h, we have the corresponding legal paths h_1 and h_2 in $M_{\mathcal{L}}$. This illustrates point 1 of lemma 4. For point 5 of the same lemma, notice that given the legal path $e\,b\,k$ of M, we have two legal paths $e_1\,b\,k_1$ and $e_2\,e_1\,b\,k_1\,k_2$ is $M_{\mathcal{L}}$.

Theorem 6. *If $\mathcal{T}(M) = N$, then N is weak linear.*

Lemma 5. *If $\mathcal{L}(M) = N$, and both N and M have a normal form, then $nf(M) = nf(N)$.*

Theorem 7. *If $\mathcal{T}(M) = N$, then $nf(M) = nf(N)$.*

Theorem 8. *If $\mathcal{T}(M) = N$, then M is strongly normalizable.*

Note that the transformation may not terminate. In every example tested, the non-termination of $\mathcal{T}(M)$ arises when the reduction of M itself may not terminate.

Example 11. Let $\Delta = \lambda x.xx$, $D = \lambda x_1 x_2 . x_1 x_2$, and $\Omega = \Delta\Delta$. We have:
$\mathcal{T}(\Omega) = \mathcal{T}(D\Delta\Delta) = \mathcal{T}(\lambda x_1 x_2 . x_1 x_2 x_2) D\Delta) = \mathcal{T}(\lambda x_1 x_2 x_3 . x_1 x_2 x_3) D\Delta\Delta) =$
$= \mathcal{T}(\lambda x_1 x_2 x_3 . x_1 x_2 x_3 x_3) DD\Delta) = \mathcal{T}(\lambda x_1 x_2 x_3 x_4 . x_1 x_2 x_3 x_4) DD\Delta\Delta) = \cdots$

Since the set of legal paths of Ω is not finite, $\mathcal{T}(\Omega)$ never terminates. This is somehow expected since, apparently, the only cause of non-termination of the transformation process is the existence of an infinite number of legal paths (which can only happen when the term is not strongly normalizable).

5 Final Remarks

In this paper we present a transformation of general terms into weak linear terms. It is necessary to find out sufficient conditions for the initial terms, under which we can guarantee that the transformation terminates. Based on what we said at the end of section 4, we conjecture that our transformation terminates if and only if the term is strongly normalizable. The left-to-right implication is proved in Theorem 8. The right-to-left implication requires a more detailed analysis of the interaction between reductions of the initial term and of the transformed term, and it is left for future work.

Acknowledgements We thank Laurent Regnier for his helpful comments on some aspects of legal paths. The work presented in this paper has been partially supported by funds granted to *LIACC* through the *Programa de Financiamento Plurianual, Fundação para a Ciência e Tecnologia* and *Programa POSI*.

References

1. Sandra Alves and Mario Florido. On the relation between rank 2 intersection types and simple types. In *Joint Conference on Declarative Programming (AGP'2002)*, 2002.
2. Sandra Alves and Mario Florido. Linearization by program transformation. In *"Logic Based Program Synthesis and Transformation - LOPSTR 2003, Revised Selected Papers"*, LNCS, 2003.
3. Sandra Alves and Mario Florido. Linearization by program transformation. Technical report, DCC-FC, LIACC, University of Porto, 2003. (available from www.dcc.fc.up.pt/∼sandra/papers/report2003.ps).
4. Andrea Asperti, Vincent Danos, Cosimo Laneve, and Laurent Regnier. Paths in the lambda-calculus. In *Logic in Computer Science*, pages 426–436, 1994.
5. Andrea Asperti and Cosimo Laneve. Paths, computations and labels in the lambda-calculus. *Theoretical Computer Science*, 142(2):277–297, 1995.
6. Henk Barendregt. *The Lambda Calculus. Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1984.
7. Henk Barendregt. The impact of the lambda calculus. *Bulletin of Symbolic Logic*, 3(2):181–215, 1997.
8. M. Coppo and M. Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame Journal of Formal Logic*, 21(4):685–693, 1980.
9. Mario Florido and Luis Damas. Linearization of the lambda-calculus and its relation with intersection type systems. *To appear in Journal of Functional Programming*, 2004.
10. J. R. Hindley. The principal type-scheme of an object in combinatory logic. *Trans. American Math. Soc.*, 146:29–60, 1969.
11. J. R. Hindley. BCK-Combinators and linear lambda-terms have types. *Theoretical Computer Science*, 64(1):97–105, 1989.
12. J. R. Hindley. *Basic Simple Type Theory*. Cambridge University Press, 1997.
13. Assaf J. Kfoury. A linearization of the lambda-calculus. *Journal of Logic and Computation*, 10(3), 2000.
14. Assaf J. Kfoury, Harry G. Mairson, Franklyn A. Turbak, and J. B. Wells. Relating typability and expressiveness in finite-rank intersection type systems. In *International Conference on Functional Programming*, pages 90–101, 1999.
15. J.J. Lévy. *Réductions correctes et optimales dans le lambda calcul*. PhD thesis, Université Paris VII, 1978.
16. J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. Assoc. for Computing Machinery*, 12:23–41, 1965.