Labels, paths and linearization of the λ-calculus

Pedro Cunha Sandra Alves Mário Florido
pedro.cunha@fc.up.pt sandra@fc.up.pt amflorid@fc.up.pt
Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
Porto, Portugal

Linearization is essentially a program transformation in whose output all uses of input-bound variables correspond to different inputs, and which preserves the computational content of the original program. It can be syntactic, when applied to every subterm, or semantic, when applied only to functions which are used during reduction. One known example of semantic linearization is Weak Linearization, based on the maximal reduction strategy and which may not terminate when the program loops. The question we intend to answer here is this: how can we make such a linearization finer, so that we can get partial linearizations which work even in cases where the term is not strongly normalizing? In this paper we define a new linearization technique that answers the previous question using a linearization parametrized by a finite sequence of redexes, which is based on the idea of approaching the term as a tree.

1 Introduction: Linearization and its State-of-the-art

Linearization, also referred to as linear expansion (or just expansion, with an implicit linear qualification), is essentially a program transformation in whose output all uses of input-bound variables correspond to different inputs, and which preserves the computational content of the original program. In terms of the λ-calculus, this means all bound variables are bound by a distinct λ. More precisely, we can define *affine* functions, if we maintain that every λ binds **at most one** variable; or *strictly linear* functions, if we maintain that every λ binds **exactly one** variable. These are not equivalent; throughout this abstract, as in many previous works, we will use linear in the first sense. [1] Thus, the term $\delta = \lambda x.xx$ is not linear, whereas the term $\lambda x_1 x_2 . x_1 x_2$ is.

The first attempt at tackling this linearization problem was by Kfoury [7], which was characterized by his goal of "simulating the standard λ-calculus by a non-standard λ-calculus where we enforce the linearity condition on function evaluation", which he would do by "[embedding] the standard λ-calculus Λ in a bigger calculus" that allows several arguments for an application and then restricting the β-reduction so that it can only take effect whenever the number of free occurrences of the applied function matches the number of arguments. In this work, an expanded term is precisely a term that has several arguments, as opposed to a standard λ-calculus term, which may only have one; Kfoury then shows how to expand a λ-term into a term of this expanded λ-calculus (Definition 2.16).

Following this, Florido and Damas defined the notion of linear expansion [6], whose first relevant difference from the previous is that it "stays" in the standard λ-calculus. Its basis are intersection types: given a term and its intersection type, this linearization obtains its (syntactical) linear form. This reveals an interesting aspect of linearization: since intersection types type exactly the strongly normalising (SN) terms, linear expansion is well-defined exactly for SN terms. So, for example, the linearization of δ is $\lambda x_1 x_2 . x_1 x_2$, whereas $\Delta = \delta\delta$ cannot be linearized through this process, since it is not SN.

[1] Note, however, that this approach works for both and can also be adapted to output affine and linear terms.

Submitted to:
TERMGRAPH 2026

In this line, we also have the Weak Linearization of Alves and Florido [1], which becomes finer by linearizing only subterms that participate in some redex. This is done by utilizing legal paths, introduced by Asperti and Laneve [3] as a one-to-one correspondence with Lévy's redex families [9]; these are paths in the term when seen as a tree, which can be interpreted as computation paths. This further emphasizes the relation between computation and linearization; in fact, by Theorem 56 of [1], we know that when a term is not SN, it does not have an expansion. So, for example, the weak linearization of δ is still δ, since there is no redex (whence the "weak" linearization); however, Δ and $(\lambda xy.x)\,I\,\Delta$ have no weak linearization, since they are not SN. More interestingly, the weak linearization of $(\lambda xy.xx)I(\delta I)$ would be $(\lambda x_1x_2y.x_1x_2)\,II\,[(\lambda x_1x_2.x_1x_2)\,II]$, since it works with *all* legal paths, i.e., all redexes (families).

The present work borrows some inspiration from previous works on linearization of the λ-calculus [1, 5, 6, 7, 10] and aims to contribute to this line of research providing a uniform framework for addressing linearization related problems.

2 Parametrized Linearization

The first linearization presented in the previous section is a syntactic linearization, in the sense that it linearizes *all* the non-linear subterms of the original term; it also depends on the typing of the term, whose inference algorithm is highly non-trivial (see, e.g., [4] and [8]).

Weak Linearization, on the other hand, relies on the *totality* of the legal paths of a term to terminate, and whenever there are infinite such paths, the linearization will not terminate (which is, again, the meaning of Theorem 56 of [1]). Besides this, since it works with all redexes, it will also expand terms that do not contribute to the normal form of the term, such as in the given example $(\lambda xy.xx)\,I\,(\delta\,I)$.

In the spirit of keeping in the λ-calculus and refining this process, the question we intend to answer is this: how can we refine such a linearization so that we can get partial linearizations (even when the term is not SN)? It seems reasonable to expect, given the strong relation with computation pointed at in the introduction, that there is a way in which we can do this, insofar as we regulate the interaction had with redexes; and so we answer thus: by a linearization parametrized by a (finite) sequence of redexes.

2.1 Redexes and Other Preliminary Definitions

Since we are parametrizing by a list of redexes, we must describe them in some (unique) way: this is the first challenge. In this matter, the work of Lévy, Asperti and Laneve is key ([9], [3]); we will particularly use labels. These are inductively defined as

$$\alpha := L \mid \overline{\alpha} \mid \underline{\alpha} \mid \alpha_1\alpha_2$$

where L is a set of atomic labels. We will use a specific type of atomic labels that depend on the node and on the tree; this labeling is done as follows (by "order", we mean "order visited by depth-first search"):

- Mark all λ nodes with positive integers in increasing order, starting at 1;
- Mark all variables bound by some λ node n as $n.k$, where k is their order ($n = 0$ for free variables);
- Mark all application nodes as $M@N$, where M is the label of the left subterm and N of the right.

A labeled term can be defined as $\Lambda := x^\alpha \mid (\lambda x.\Lambda)^\alpha \mid (\Lambda\Lambda)^\alpha$, where x is taken from a set of variables and α is a label. These terms can be seen as trees when we consider variables as leaves, λs as nodes with one child and applications (@) as nodes with two children, in which each node[2] is annotated with its corresponding label (such as that in Fig. 2, left). The tree rewriting rule for reduction is in Fig. 1.

[2] In the cited works, only edges are labeled, but there is a direct bijection between the labeled edges and the labeled nodes.

Thus, since we can label each node as mentioned we can also associate computations in that term with specific labels (encoded by the reduction rule), which we can translate to paths in the original term-tree (Definition 5.1 of [3]). Since there is a correspondence between legal paths and these label yielded paths (as Theorem 8.11 of [3] asserts), working with either is equivalent when talking about redexes.

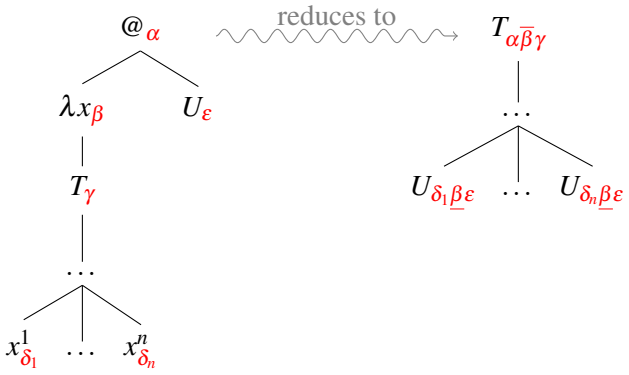

Figure 1: Labeled reduction rule. The degree of the redex is β.

However, it must be noted that these uniquely represent redex *degrees*, which are unique *up to redex families* (Theorem 6.1 of [3]; p. 159 of [2]). In practice, this means that we do not have a guarantee of unicity of representations for every specific redex in an execution; e.g., the term $T = (\lambda x.xx)(\lambda xy.(\lambda z.z)xy)$ would yield the labeled term in Fig. 2 after reducing its outermost redex.

Note that there are two specific redexes with degree 4. So, when we need to reference specific redexes in a reduction sequence, degrees, and thus labels and legal paths, are not enough.

Still, specific redexes of the same family arise only when they are copied in some reduction sequence; thus, since computations are "stored" in the tree's nodes and structure, if we refer to the redexes as pairs of *the shortest paths from the root to the application and λ node* in an appropriate tree (i.e., obtained by a sequence of reductions from the original term, where the latter is a left subchild of the former), we have a unique characterization of such redexes.

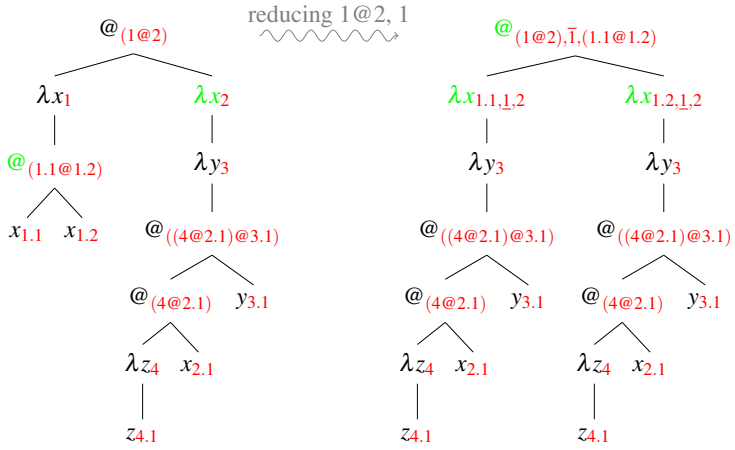

Figure 2: T after reducing leftmost-outermost redex, yielding $(\lambda xy.(\lambda z.z)xy)\,(\lambda xy.(\lambda z.z)xy)$.

Following are the required (preliminary) definitions and proposition; [3] unless specifically stated, by "term" we will always mean a labeled term.

Proposition. *In a term T', obtained by reducing any number of redexes of a term T, no two different nodes have the same shortest path from the root.*

Definition (Label Path)**.** *The label path of a node N in a term T is the (shortest) path from T's root to N.*

Definition (Direct Redex)**.** *A direct redex of a term T is a pair $(r_@, r_\lambda)$, where $r_@, r_\lambda$ are the label paths of two nodes whose label ends in $M@N, n$, respectively, in T and n is the left subchild of $M@N$.*

Definition (Virtual Redex)**.** *A virtual redex of a term T is a direct redex in a term T' obtained by reducing some (non-zero, finite) amount of direct redexes.*

Definition (Redex Duality)**.** *For some redex $r = (r_@, r_\lambda)$, we say that $r_@$ is the dual of r_λ, written $d(r_\lambda)$, and vice-versa.*

[3] Proofs can be found in https://github.com/pcunha1274/Proofs-for-TERMGRAPH.

Definition (List of Redexes)**.** *A list of redexes of a term T is a list of either direct or virtual redexes.*

The following sequentiality condition of a list of redexes guarantees that a redex list is sequential in the intuitive sense, by forcing the dependency of a redex on the previous ones.

Definition (Sequential List of Redexes/Reduction Sequence)**.** *Let $r_1,\ldots,r_n$ be a list of redexes of a term T. We say it is sequential - or a reduction sequence - whenever each r_i is a direct redex of the term T' obtained by reducing $r_1,\ldots,r_{i-1}$.*

Thus, we can talk about a sequence of redexes in a sense more general than that of a strategy of evaluation (as in, e.g., p. 83 of [1]); this will be our parameter.

For instance, in Fig. 2 the label path of the left child of the root is $1@2,1$; thus, $r=(1@2;1@2,1)$ is a direct redex, which we will write $(1@2\,|\,1)$, since the path of the @-node is always a prefix of the path of the λ-node; and $r_v=(1@2,\overline{1},1.1@1.2\,|\,1.1,\underline{1},2)$ is a virtual redex (note that it can be mapped in T). Moreover, the list of redexes r,r_v is a reduction sequence, whereas r_v,r is not.

During the expansion, some nodes (along with their labels) will be copied, which will form clusters. So, when traversing these trees, clusters of λ-nodes are taken as one node and we specify which @-node we should go to with annotated paths.

Definition (Cluster)**.** *A cluster of (λ- or @-)nodes in T is a set of nodes that form a subtree of T and share the same label.*

Definition (Annotated path)**.** *An annotated path is a label path where application nodes that are followed by their right child in the path (i.e., that form the right edge of that application node) have annotations, more precisely, these nodes are pairs of the form $(\ell,\pi\to n)$, where π is some annotated path and n is a positive integer.*

Definition (Multiplicity of a node/cluster)**.** *The multiplicity of a λ-node is the number of its bound variables; of a λ-cluster is the sum of the multiplicities of its λ-nodes; of a @-cluster is the number of nodes in it (i.e., the number of arguments). $\mu(\pi)$ stands for the multiplicity of π, which is the path of a node (cluster).*

Thus, in Fig. 4 (left) we have a λ-node cluster with path $1@3,1$ and multiplicity 2 (examples of annotated paths can be found in the next subsection).

We now start the accessory definitions for our linearization function.

Definition (Local interpretation of label paths)**.** *The local interpretation of a label α, in a term T, with previous path p and list of redexes l_r is given by the function $\texttt{interpret}_{\texttt{local}}$, defined as follows:*

1. $\texttt{interpret}_{\texttt{local}}(\alpha,T,p,l_r)=(T,p+\alpha,l_r)$*, with atomic α;*
2. $\texttt{interpret}_{\texttt{local}}(\overline{\alpha},T,p,l_r)=(T,p+\overline{\alpha},l_r)$*;*
3. $\texttt{interpret}_{\texttt{local}}(\underline{\alpha},T,p,l_r)=$
   - $(T,p+\underline{(\alpha,d(\pi_{\lambda n})\to i)},l_r)$*, if α is atomic.*
     i is the position of $n.k$, which is the node that p leads to (by Proposition 5.3 of [3]), relative to the other bound variables of $\pi_{\lambda n}$ in T (in DFS order); the path π_λ is obtained by removing from p the sequence starting at $n.k$ backwards until the first occurrence of n (exclusively), while its dual is found in l_r.
   - $\texttt{interpret}_{\texttt{local}}((\alpha_3,d(\pi_{\lambda n})\to i),T,p+\underline{p''},l'_r)$*, if $\alpha=\alpha_1\alpha_2\alpha_3$, with α_1,α_3 atomic and α_2 non-atomic.*
     $p''=p'-p$ and $(T,p',l'_r)=\texttt{interpret}_{\texttt{local}}(\alpha_2,\texttt{interpret}_{\texttt{local}}(\alpha_1,T,p,l_r))$; i and π_λ are obtained as in the previous case.

4. $\texttt{interpret}_{\texttt{local}}(\alpha_1\,\alpha_2, T, p, l_r) = \texttt{interpret}_{\texttt{local}}(\alpha_2, \texttt{interpret}_{\texttt{local}}(\alpha_1, T, p, l_r))$, *where* α_1 *is atomic.*

Definition (Interpretation (global))**.** *The (global) interpretation of a label path* $\alpha_1 \ldots \alpha_n$*, in a term* T*, with previous path* p *and list of redexes* l_r *is given by the function* `interpret`*, defined as follows:*

$$\texttt{interpret}(\varepsilon, T, p, l_r) = p$$
$$\texttt{interpret}(\alpha_1 \ldots \alpha_n, T, p, l_r) = \texttt{interpret}(\alpha_2 \ldots \alpha_n, \texttt{interpret}_{\texttt{local}}(\alpha_1, T, p, l_r))$$

Proposition (Injectivity of `interpret`)**.** `interpret` *is injective over label paths.*

This means we lose no information regarding redexes when translating to annotated paths.

Definition (Local expansion)**.** *The local expansion of a redex* r' *(expressed as an annotated path) in a term* T *is given by the function* `expand`*, which is defined as*

$$\texttt{expand}(r', T) = T'$$

where T' *is obtained by the tree rewriting rule in Fig. 3.*

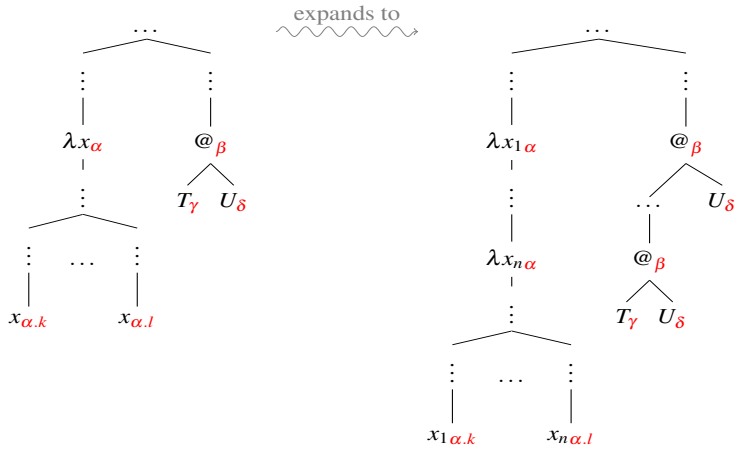

Figure 3: Tree-rewriting definition of `expand`. $d(\lambda x_\alpha) = @_\beta$, which have paths $r'_\lambda, r'_@$; $n = \mu(\lambda x_\alpha)$, and there are n $@_\beta$ nodes; we must have $\mu(\lambda x_\alpha) \geq 1$. Since they are analogous, we omit the other cases for space management reasons.

Definition (Balancing of a term)**.** *Balancing of a term* T*, for a reduction sequence* $r_1, \ldots, r_n$*, is given by the function* `balance`*, which is defined as:*

$$\texttt{balance}(T, r_1 \ldots r_n) = \begin{cases} T & \text{if } \mu(r_{@i}) = \mu(r_{\lambda i}), \forall i.\mu(r_{\lambda i}) \geq 1 \\ \texttt{expand}(r'_n, \ldots \texttt{expand}(r'_1, T)) & \text{otherwise} \end{cases}$$

where $r'_i = \texttt{interpret}(r_i, T, \varepsilon, l_{r_i})$ *and* $l_{r_i} = (r'_1, \ldots, r'_{i-1})$.

Thus, we arrive at our main definition.

Definition ((Parametrized) Linearization)**.** *Let* T *be a term and* $r_i, \ldots, r_n$*, as well as* l_r *(which will be the list of evaluated redexes), a sequential list of redexes. The function* $\texttt{linearize}(r_i \ldots r_n, T, l_r)$ *is defined as*

$$\texttt{linearize}(\varepsilon, T, l_r) = T$$
$$\texttt{linearize}(r_i \ldots r_n, T, l_r) = \texttt{linearize}(r_{i+1} \ldots r_n, \texttt{balance}(\texttt{expand}(r'_i, T), l'_r), l'_r)$$

where $r'_i = \texttt{interpret}(r_i, T, \varepsilon, l_r)$, $l'_r = l_r + r'_i$ *and* + *is list concatenation.*
The parametrized linearization of a term T *with redex sequence* $r_1, \ldots, r_n$ *is* $\texttt{linearize}(r_1 \ldots r_n, T, \varepsilon)$

Conjecture (Correctness)**.**
Let T *be a term,* $r_1, \ldots, r_n$ *a reduction sequence of* T *and* $E = \texttt{linearize}(r_1 \ldots r_n, T, \varepsilon)$*. If* $T \to_{r_1} \cdots \to_{r_n} T'$ *and* `balance` *terminates, then there exists a reduction sequence of* E *such that* $E \to_\beta^* T'$*.*

This conjecture would imply that, for a SN term T such that $N = \texttt{nf}(T)$, if `balance` terminates, then there exists a reduction sequence $r_1, \ldots, r_n$ such that $E = \texttt{linearize}(r_1 \ldots r_n, T, \varepsilon)$ and $\texttt{nf}(E) = T'$.

2.2 Examples

Consider the term 2 2, where 2 is the Church numeral $2 = \lambda f x.f(fx)$. This is a SN term and we can normalize it in (at least) two different ways: by always reducing the leftmost, outermost redex or by reducing the rightmost, innermost one. Naturally, they will yield the same term (and thus, by the previous conjecture, will have the same final linearization when parametrized by one or the other), but their partial linearizations will not be equal.

For the leftmost-outermost strategy, consider only the first two redexes, $r_1 = (1@3 \,|\, 1)$ and $r_2 = (1@3, \overline{1}, 2, 1.1@(1.2@2.1) \,|\, 1.1, \underline{1}, 3)$.

The iteration for r_1 is $\texttt{balance}(\texttt{expand}(r_1, T), [r_1]) = T'$, which is represented in Fig. 4 (left; for space management reasons, single nodes with multiple arguments represent clusters of @-nodes).

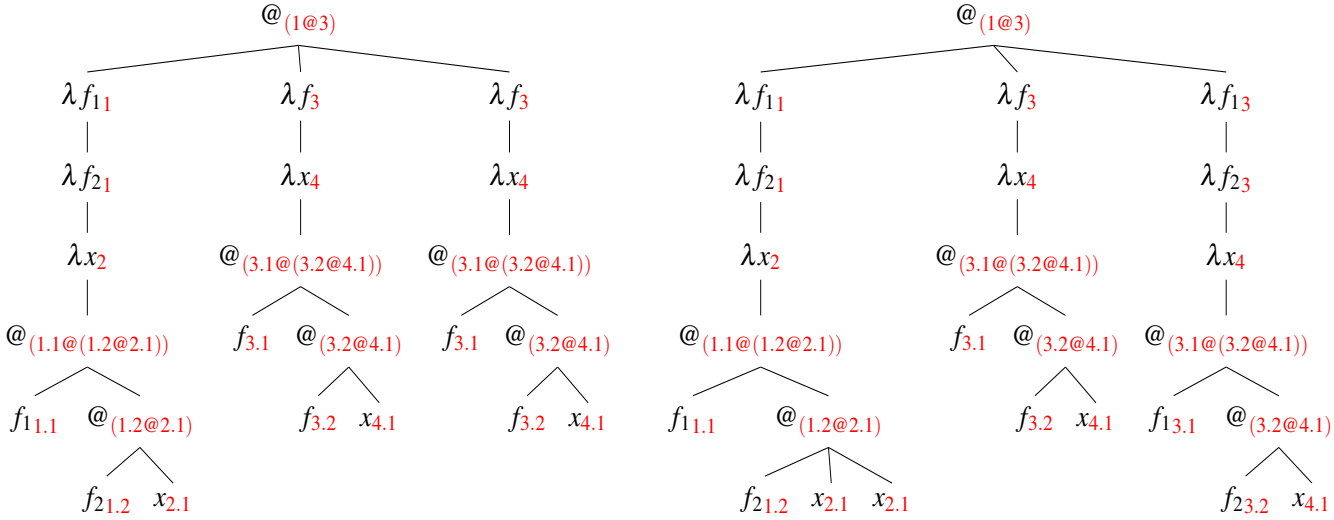

Figure 4: Left: T after expanding r_1. Right: Expanding the next rightmost-innermost redex of T.

For r_2, we get $\texttt{balance}(\texttt{expand}(r'_2, T'), l_r)$, with $l_r = [(1@3 \,|\, 1), r'_2]$, which yields:

1. $r'_2 = \texttt{interpret}([1@3, \overline{1}, 2; 1.1@(1.2@2.1) \,|\, 1.1, \underline{1}, 3], T', \varepsilon, l_r) =$
   $1@3, \overline{1}, 2; 1.1@(1.2@2.1) \,|\, 1.1, \underline{(1, 1@3 \to 1)}, 3 = \pi_{@2} \,|\, 1.1, \underline{(1, 1@3 \to 1)}, 3$
2. $\texttt{expand}(r'_2, T') = T''$, where T'' is as in Fig. 5.
3. $\texttt{balance}(T'', [(1@3 \,|\, 1), (\pi_{@2} \,|\, 1.1, \underline{(1, 1@3 \to 1)}, 3)]) =$
   $\texttt{balance}(\texttt{expand}(r'_2, \texttt{expand}(r'_1, T'')), [r'_1, r'_2]) =$
   $\texttt{balance}(\texttt{expand}(r'_2, T'''), [r'_1, r'_2]) = \texttt{balance}(T'''', [r'_1, r'_2]) = T''''$, where T'''' is as in Fig. 6.

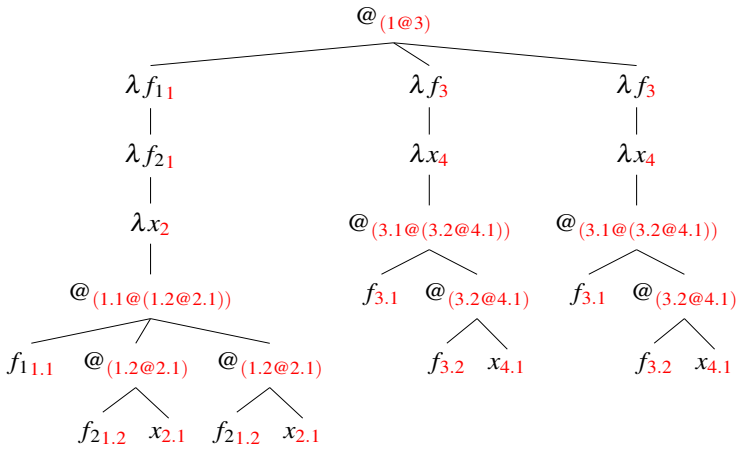

Figure 5: T'': 2nd iteration, after local expansion, before balancing

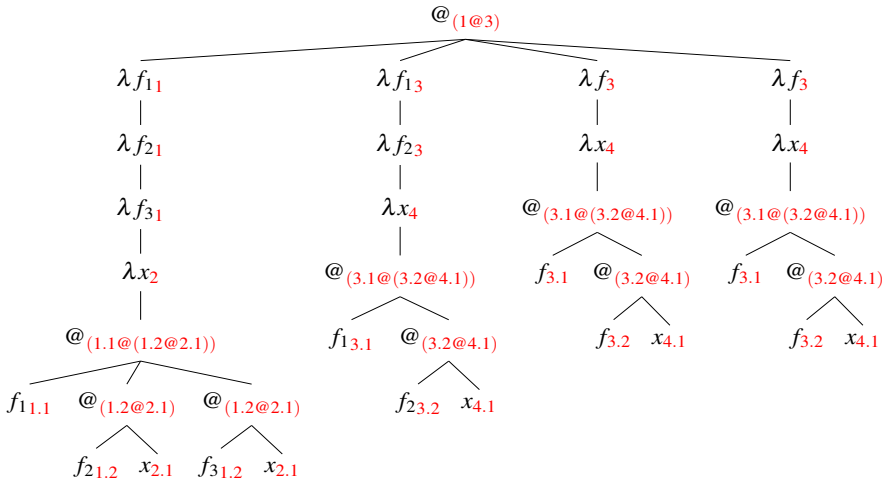

Figure 6: T'''': 2nd iteration, after local expansion and balancing

Note that we have $2\ 2 \to^*_{r_1,r_2} \lambda fx.(2f)[(2f)x]$ and that the term in Fig. 6 also reduces to this term.

For the first two redexes of the rightmost-innermost strategy (r_1 is also the first), we would have, instead, the expanded term in Fig. 4 (on the right). However, the two expansions would yield the same "final" result.

Another interesting example is whenever there are redexes that do not matter for the normal form: e.g., the normal form of $(\lambda xy.xx)I(\delta I)$ is $I = \lambda x.x$, and the inner redex δI is, in a sense, irrelevant to it. However, in this case, standard weak linearization will give $(\lambda x_1x_2y.x_1x_2)II[(\lambda x_1x_2.x_1x_2)II]$, which would be the same output if our linearization was parametrized by all the possible redexes. Still, if we started with $(1@(3@4)\,|\,1)$ (the leftmost-outermost redex) and never reduced the inner redex, we would still get the normal form and would have the expansion $(\lambda x_1x_2y.x_1x_2)II[\delta I]$. This can only happen for terms with nodes that erase subterms (i.e., with multiplicity 0 and, thus, which are affine).

Notably, it is also possible to linearize non-SN terms. If they terminate, i.e., if they are weakly normalizing (WN), then it suffices to choose a reduction sequence that normalizes the term. For example,

the term $(\lambda xy.xx)I\Delta$, parametrized by the redex $(1@3\,|\,1)$, would yield $(\lambda x_1x_2y.x_1x_2)II\Delta$; note, however, that this would not be possible in the simple weak linearization. A more interesting WN term is the factorial of some number, `fact`n, which can be recursively defined by using the fixpoint operator, $Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$. Since `fact` has the fixpoint combinator as a subtree, it necessarily has a non-terminating reduction sequence; however, we know that, e.g., `fact` 2 also reduces to 2; a linearization of this term would also be possible by choosing as parameter the reduction sequence that yields its normal form. Finally, if the term is simply non-terminating, like Δ, it is still possible to (partially) linearize it, if we take a finite list of redexes as our parameter (which also applies in the previous examples, that is, we can still partly linearize those WN terms by choosing other reduction sequences as our parameter).

3 Ongoing/Future Work

We are finishing the proof of correctness for the linearization; the core of the idea starts by defining a relation between redexes: intuitively, we say r_2 (potentially) unbalances r_1 if $@_{r_2}$ is in the subtree of λ_{r_1} and has, in its right subtree, a variable bound by the latter. From here, it follows that `expanding` some redex r will only unbalance terms that it relates to, and only if it is not linear already. Thus, the only way that `balance` cannot terminate is if(f) there is always some redex unbalancing another, or, put in simpler terms, if(f) there is a cycle in the directed graph that represents this relation (since our redex list is finite). The reason for such a thing not being possible, we think, is because otherwise it would be possible to have a reduction sequence where the **same** redex is reduced twice. This would guarantee that `linearize` terminates and that the resulting term is weak linear (for the redexes `expanded`). To prove correctness, we need to establish a relation between the redexes of the original term and those of its linearized version, which we do via clusters. Thus, a redex in the original term will "create" $n-1$ new redexes, where n is the multiplicity of the cluster (after linearization). It may seem that the "output" of the original and new redexes is not the same, since the multiplicity of some λ in the original term is not necessarily the same as that of its corresponding cluster in the expanded term, but since the multiplicity of the latter accounts for all of the copying that will be done in the reduction sequence, at the end of said reduction, and necessarily only then, the "outputs" will match.

Next, we intend to establish that the presented linearization, when parametrized by the maximal strategy, is equivalent to the Weak Linearization of [1]. For this, we need to generate the redexes corresponding to the maximal strategy and show that they are in correspondence with the ones "retrieved" by `next_non_linear`, which also requires a correspondence between label/annotated paths and legal paths (which should be direct, since label paths are just labels with prefixes). By showing that `expand` works in a way similar to $\mathscr{L}$, since we are expanding the same redexes, the final term must be the same. If we succeed, with the help of the previous results, we can solve the conjecture in [1], the converse of Theorem 56 (if a term is SN, then it has a weak linearization), by noticing that a term T being SN implies that the maximal strategy always has a determinate number of redexes; it then follows, from the previous result, that T has a (simple) weak linearization.

From here, the natural way is seeing how this relates to syntactical linearization, e.g., whether it is possible to change the procedure to match the syntactical linearization and what is required; relations with other linearizations, specifically with the Taylor expansion [5], should also be convenient to explore at this stage. Finally, the strong match between syntactical linearization and intersection types, as well as the granularity and parametric character of the presented linearization, prompts us to look at possible relations with Carlier's work [4]: a type inference algorithm that gradually infers the intersection type of a term based on the evaluation of redexes.

Acknowledgements

This work was financially supported by: UID/00027/2025 of the LIACC - Artificial Intelligence and Computer Science Laboratory with DOI https://doi.org/10.54499/UID/00027/2025, funded by Fundação para a Ciência e a Tecnologia, I.P./ MECI through the national funds.

References

[1] S. Alves and M. Florido. Weak linearization of the lambda calculus. *Theoretical Computer Science*, 342(1):79–103, Sept. 2005.

[2] A. Asperti and S. Guerrini. *The optimal implementation of functional programming languages*. Cambridge University Press, 1998.

[3] A. Asperti and C. Laneve. Paths, computations and labels in the lambda-calculus. *Theoretical Computer Science*, 142(2):277–297, May 1995.

[4] S. Carlier and J. B. Wells. Type inference with expansion variables and intersection types in system E and an exact correspondence with beta-reduction. In *Proc. of the 6th ACM SIGPLAN Int. Conf. on Principles and Practice of Declarative Programming*. ACM, Aug. 2004.

[5] T. Ehrhard and L. Regnier. Uniformity and the Taylor expansion of ordinary lambda-terms. *Theoretical Computer Science*, 403(2-3):347–372, Aug. 2008.

[6] M. Florido and L. Damas. Linearization of the lambda-calculus and its relation with intersection type systems. *Journal of Functional Programming*, 14(5):519–546, Sept. 2004.

[7] A. J. Kfoury. A linearization of the lambda-calculus and consequences. *Journal of Logic & Computation*, 10(3), 2000.

[8] A. J. Kfoury and J. B. Wells. Principality and decidable type inference for finite-rank intersection types. In *Proc. of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM, Jan. 1999.

[9] J.-J. Levy. Optimal reductions in the lambda calculus. *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, 01 1980.

[10] D. Mazza, L. Pellissier, and P. Vial. Polyadic approximations, fibrations and intersection types. *Proceedings of the ACM on Programming Languages*, 2(POPL):6:1–6:28, 2018.