On global dynamics of optimal graph reduction

Julia L. Lawall[*]
IRISA
Campus Universitaire de Beaulieu
35042 Rennes Cedex
France

Harry G. Mairson[†]
Computer Science Department
Brandeis University
Waltham, Massachusetts 02254

Abstract

Optimal graph reduction technology for the λ-calculus, as developed by Lamping, with modifications by Asperti, Gonthier, Abadi, and Lévy, has a well-understood *local dynamics* based on a standard menagerie of reduction rules, as well as a *global context semantics* based on Girard's geometry of interaction. However, the *global dynamics* of graph reduction has not been subject to careful investigation. In particular, graphs lose their structural resemblence to λ-terms after only a few graph reduction steps, and little is known about *graph* reduction strategies that maintain efficiency or structure. While the context semantics provides global information about the computation, its use as part of a reduction strategy seems computationally infeasible. We propose a tractable graph reduction strategy that preserves computationally relevant global structure, and allows us to efficiently bound the computational resources needed to implement optimal reduction.

A simple canonical representation for graphs is introduced that we call *fan-normal form.* This normal form allows us to reduce graphs based on efficient identification of β-redexes, rather than being guided by lower-level search for interacting nodes. In addition and perhaps more important, the fan-normal form facilitates a complexity analysis of graph reductions, showing that the number of fan interactions is essentially bounded by a polynomial in the number of unique Lévy labels generated during a labelled reduction. This global analysis of the finitary dynamics of optimal reduction is the first demonstration that a reasonable implementation-independent cost model for the λ-calculus is in fact realized by Lamping's abstract algorithm. It remains to be seen whether similar claims can be made about so-called *bookkeeping* for fan interactions.

1 Introduction

A fundamental distinction between the varieties of functional programming languages can be made by considering respective *evaluation strategies.* In what order are subexpressions evaluated? All such languages are founded on the λ-calculus [Chu41, Bar84]. But this calculus, like everyday algebraic calculation, is *indeterminate*: choosing what subexpression to simplify next is not specified, though the Church-Rosser theorem provides assurance that we always get the same answer, if there is one. The designer of a functional programming language must complete this framework by choosing a deterministic strategy, while balancing issues of efficiency and of semantic clarity.

[*]Work performed at Brandeis University, supported by ONR Grant N00014-93-1-1015 and NSF Grant CDA-9504288, and at IRISA, supported by a NSF International Post-Doctoral Research Fellowship. E-mail: jll@irisa.fr. Phone: (33) 2 99 84 75 36. Paper number 1569.225.

[†]Supported by ONR Grant N00014-93-1-1015, NSF Grant CCR-9216185, and the Tyson Foundation.

In searching for efficient evaluation strategies for the λ-calculus, a natural goal was to minimize the number of *β-reductions*, which implement the application of functions to arguments. However, there are many examples of λ-terms where the minimal reduction sequence depends on specific term structure, and seems inaccessible to any fixed, universal reduction strategy. Instead, attention was drawn to augmenting the syntactic world of the λ-calculus with a *sharing* mechanism, introducing basic data structures allowing *duplicated terms*—the bane of efficient evaluation—to be represented uniquely and reduced without redundancy. A well-known evaluation technique of this sort is due to Wadsworth [Wad71]. While these techniques succeed in the efficient sharing of *arguments*, they suffer from redundancies caused by the necessary copying of shared *functions*.

In foundational research some two decades ago, Jean-Jacques Lévy attempted to formally characterize what an optimally efficient strategy would look like, even if the technology for its implementation was at the time lacking. Lévy's dual goals were *correctness*, so that such a reduction strategy does not diverge when another could produce a normal form, and *optimality*, so that redexes are not duplicated by a reduction, causing a redundancy in later calculation [Lévy78, Lévy80]. The relevant functional programming analogies are that call-by-name evaluation is a correct but not optimal strategy, while call-by-value evaluation is an approximation of an incorrect but optimal strategy. Lévy introduced a variant of λ-calculus where all subterms of an expression are annotated with *labels* that code the history of a computation. He proposed the notion of a *redex family*—redexes in a term with identical labels in the "function" position—to identify similar redexes whose reduction should somehow be evaluated *at once* (via a so-called *parallel β-reduction*) by any efficient scheme.

Recent research by Lamping, and independently Kathail, has shown that there indeed exist λ-calculus evaluators satisfying Lévy's specification [Lam90, Kat90]. Lamping's work, in particular, was followed by variants and simplifications proposed by Asperti, and Gonthier, Abadi, and Lévy, among others [Asp94, GAL92]. The basic framework was to begin with a straightforward graph reduction scheme made up of application (@) and abstraction (λ) nodes, and augment it with *sharing nodes* that lazily duplicate pieces of the graph. Traditional sharing nodes, found in implementations of lazy evaluation, allow the sharing of a single *value* between two different *evaluation contexts*. Lamping's key insight was to introduce a *dual* form of sharing, where a single *evaluation context* is shared by two different *values*. These two forms of sharing nodes allow a single, shared representation of a redex family, and thus provide a means to implement Lévy's notion of parallel reduction. The varied implementations of this fundamental framework differ in the underlying bookkeeping technology used to control interactions between sharing nodes. For a gentle introduction to and summary of these issues, see [LM96].

A distinguishing characteristic of optimal reduction schemes founded upon graph reduction is their description solely in terms of local reduction rules. This approach simplifies the underlying atomic operations, and facilitates implementation. Reducing the graph representation of a λ-term, however, quickly yields intermediate graphs (i.e., representing intermediate terms in the reduction) whose structure is utterly obscure. Formidable problems then exist with respect to *readback* of a term represented by a graph. In fact, the problem becomes so tortuous that [GAL92] largely dispenses with readback, substituting instead the use of *context semantics* to read back the (invariant!) Böhm tree of the term.

Equally intractible seems the problem of simply finding a β-redex in the graph, which is by contrast absurdly trivial when reducing terms. The difficulty is never faced by known implementations, which just look for interacting pairs of nodes, whether or not they are @ and λ nodes. An unfortunate consequence of computation via local rules is that optimal graph reduction evaluators can exponentially duplicate graph structure even though there are no more β-redexes in the graph [LM96]. Because paths between @ and λ nodes may encounter intermediate sharing nodes linked in perverse ways, the computationally prohibitive context semantics of [GAL92] has been the only known way of doing relevant static analysis of graphs.

Optimal graph reduction technology has also obscured the complexity analysis of reduction. In particular, in what sense is the graph technology algorithmically efficient? A recent paper by Frandsen and Sturtivant proposed the cost of a reduction to be the number of parallel β-reduction steps, plus the lengths of the initial and final terms [FS91]. Unfortunately, graph reduction may require interaction of sharing nodes that grows exponentially in the number of parallel β-steps, a result derived independently by Asperti [Asp96], and in a simplified way by the authors [LM96]. As a consequence, initial and final terms can be short, parallel reductions can be very few, yet intermediate forms can grow exponentially in these parameters. Mackie has investigated optimizations of the algorithm, however, his result apply to the static context semantics, rather than to reduction [Mac95].

In summary, while the development of optimal reduction engines has been inspired, at least in spirit, by biologist René Dubos' famous adage to think globally and act locally, it has been damned by the admonishment of Socrates that we know the good things, but we don't do them. It is certainly an open problem to improve the efficiency of implementing reduction in λ-calculus, but it seems unlikely that parallel β-reduction can be effected as a unit-cost operation, since the cost of substitution may vary widely. A better course of action, in our opinion, is to consider more realistic cost models that explain the inherent efficiency of optimal graph reduction in an implementation-independent way.

Technical contributions:

In this paper, we make four fundamental contributions to understanding the finitary dynamics of optimal graph reduction at a global level.

- We define an invariant relating the number of Lévy labels in a term to the structure of the associated graph. This invariant is maintained by reduction.
- We characterize a canonical form for graphs that we call *fan-normal form*. This normal form makes paths explicit, and thus allows the efficient identification of parallel β-redexes, without resorting to the context semantics. Graph reduction is reorganized into canonical β-steps, which reduce a single β-redex and return the graph to fan-normal form.
- We propose a reasonable, implementation-independent cost model to account for Lamping's *abstract algorithm*, counting the work of fan nodes, but not bookkeeping (e.g., bracket and croissant) nodes. Namely, we replace the number of parallel β-reductions in the Frandsen-Sturtivant cost model by the number of unique Lévy labels generated during a labelled reduction.
- By restructuring graph reduction as a sequence of canonical β-steps, we can control the number of fan interactions between ordinary β-reductions. Combining the invariant on path labels with the maintenance of fan-normal form, we show that Lamping's abstract algorithm does satisfy the proposed cost model. This analysis establishes that the graph redundancy introduced by lazy copying can be bounded by a polynomial in the number of unique Lévy labels. Polynomial relatedness is the fundamental equivalence relation inherent in the *invariance thesis* that characterizes "first class" machine models; this notion is the computer scientist's refinement of Church's thesis [vEB90].

In a recent paper, Asperti has advocated the number of fan interactions as *the* defining cost metric for λ-calculus reduction [Asp96]. He intuitively asserts that Lamping's abstract algorithm performs no useless work. We agree with the assessment of the efficiency of the abstract algorithm. However, inspired by both the literature on machine simulation, where cost models are not linked to implementation or simulation, as well as dominant themes in denotational semantics, where the meaning

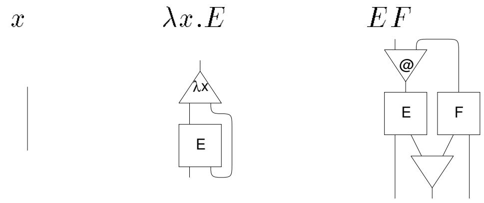

Figure 1: The translation of λ terms

of a language is distinct from any particular implementation technology, we prefer to advocate our implementation-independent metric.

Our analysis is the first demonstration that a reasonable implementation-independent cost model for the λ-calculus can in fact by realized by an efficient machine implementation. By *implementation* here, we mean Lamping's abstract algorithm; we have shown in [LM96] that the further interaction of *bookkeeping nodes* (brackets, croissants, etc.) used to maintain *fan indices* may grow exponentially in the number of fan interactions. It remains an important open research question whether an alternate bookkeeping scheme could have cost polynomial in the number of fan interactions.

2 Preliminaries

We begin with an overview of Lamping's abstract algorithm, and with the Lévy labelled λ-calculus.

2.1 Lamping's abstract algorithm

Lamping's abstract algorithm is a variant of the usual graph implementation of call-by-need evaluation, but in which shared structures are lazily duplicated by the introduction of pairs of sharing nodes. Graphs are composed of fan nodes, which are classified as λ, @, or sharing nodes. The translation of a λ-term into a graph is shown in Figure 1. Wires at the bottom of a subgraph represent free variables.

Fan nodes, whether λ, @, or sharing nodes, can only interact at a distinguished *interaction port*. The rules for the interaction between fan nodes are shown in Figure 2.[1]

To understand the meaning of these graph reduction rules, consider, for example, the duplication of a λ node by a sharing node. The sharing node indicates the use of a λ-expression by two different contexts. In graph reduction, the λ node is copied for each context, and two sharing nodes are introduced. The left sharing node shares the *body* of the λ-expression between the two uses. The right sharing node dually shares the bound variable. If both λ-expressions are later functions in β-redexes, the arguments will share the context given by the bound variable wire.

How sharing nodes interact depends on whether they are *similar* or *different*. Two sharing nodes are said to be similar when they were created as part of the duplication of the same λ-abstraction. Detecting whether two sharing nodes are similar is the responsibility of the bookkeeping nodes, which we do not consider here. In the abstract algorithm, we assume that the identity of sharing nodes is determined by an oracle.

[1] Gonthier, Abadi, and Lévy present these rules more concisely by considering λ and @ nodes as sharing nodes that are the same as each other, but different from all other sharing nodes. Our more verbose presentation coincides with the distinctions made by the analyses later in the paper.

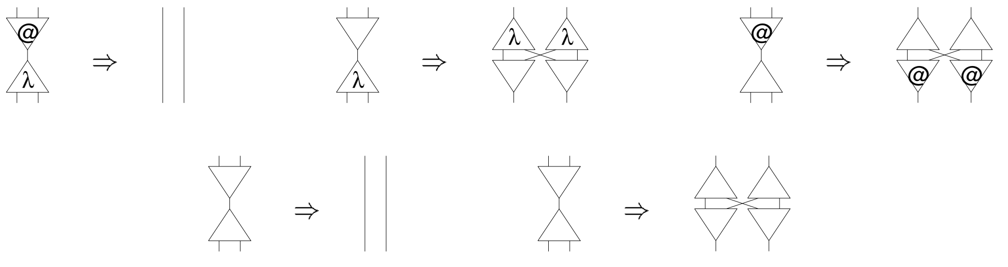

Figure 2: The interactions between fan nodes

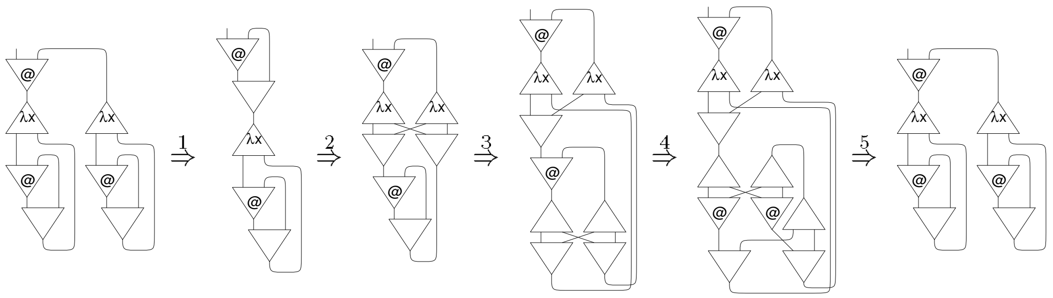

Figure 3: The reduction of $(\lambda x.xx)(\lambda x.xx)$

The beginning of the reduction of $(\lambda x.xx)(\lambda x.xx)$ is shown in Figure 3. In the second step a λ-abstraction begins to be lazily duplicated to enable a β-reduction. A more complete and intuitive introduction to the abstract algorithm is presented in [LM96].

In the rest of the paper, we use the following terminology to describe the structure of graphs.

Definition 2.1 *A* primary *node in a graph is either a λ node or an @ node. A* path *between two primary nodes does not enter or exit any other primary node, so that all intermediate nodes occurring along a path are sharing nodes.*

Definition 2.2 *A* port *is the endpoint of a graph edge incident upon a node. Each node has three ports: an* interaction *port, and* left *and* right *ports identifying the two non-interacting ports.*[2] *Ports are further classified as either* value ports *or* environment ports. *A* value port *of a λ, @, or sharing node supplies, via the connecting path, a value to an accompanying evaluation context. Dually, an* environment port *of a λ, @, or sharing node supplies, via the connecting path, an evaluation context to an accompanying subgraph.*

Definition 2.3 *A* positive *sharing node is one whose interaction port is an environment port, and whose non-interaction ports are value ports. Dually, a* negative *sharing node has an interaction port that is a value port, and non-interaction ports that are environment ports.*

[2] Gonthier, Abadi and Lévy call these *black* and *grey* ports; Lamping calls them *star* and *circle* ports. Rather than give in to either nomenclature, we invent our own.

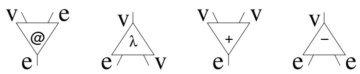

Figure 4: Value and environment ports

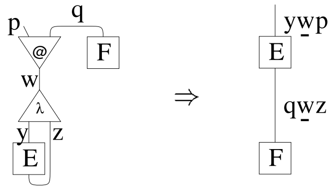

Figure 5: Graph rewriting for parallel labelled β-reduction

Figure 4 illustrates λ nodes, @ nodes, and positive and negative sharing nodes, where **v** and **e** mark value and environment ports. It is a simple observation that an edge between two nodes connects a value port with an environment port: continuations need values, and vice versa.

2.2 Lévy labels

There are a variety of flavors of Lévy labels; in this paper, we have relied on the version in [Fie90]. The set of labels is made up of a countably infinite set of atomic labels (such as integers), and closed under the operations of concatenation and *underlining.* Briefly, each subterm of the initial λ-term is initially annotated with a unique *label.* As reduction occurs, labels are concatenated and underlined, according to arcane rules, so that the labels encode the history of the computation. A labelled λ-term thus codes the reduction history that led to its derivation; labels can also be interpreted in terms of *context semantics* [GAL92] as coding information flow in the graph representation between functions and arguments. This insight is developed in considerable detail by Asperti [Asp95]. The label on the λ-abstraction in a β-redex is called the *family index* of the redex, because the same label may appear on the function position of a *family* of distinct redexes that should be reduced as one. When $(\lambda x.E)F$ is reduced, the label on each free x in E is (reverse) concatenated with the underlining of the family index, and the label of the argument; the label of E is concatenated with the underlining of the family index, and the label of the redex:

$$(\lambda x.E)^\ell F \to E^{\underline{\ell}}[x \mapsto F^{\underline{\ell}}]$$

For example,

$$((\lambda\, x\, .\, (x^1\, x^2)^3)^4\, (\lambda\, x\, .\, (x^5\, x^6)^7)^8)^9 \to\ ((\lambda x.(x^5\, x^6)^7)^{8\underline{4}1}(\lambda x.(x^5\, x^6)^7)^{8\underline{4}2})^{3\underline{4}9}$$

The Lévy-labelled λ-calculus has a direct analogue in terms of sharing graphs, where edges in the graph are annotated with labels. For example, the above labelled reduction rule is mirrored by the labelled graph rewriting rule shown in Figure 5. We also need to specify how labels are shared. When a label is duplicated by a sharing node, it is concatenated by a positive node, and reverse concatenated by a negative node. These rules, shown in Figure 6, preserve the fidelity of representation and of readback.

Figure 6: Commuting labels and sharing nodes

3 An invariant on intermediate graphs

As observed by Lamping [Lam90], it is not the case that every link in the graph is associated with a unique Lévy label. We can state an invariant, however, relating Lévy labels to intermediate graphs. This invariant will be essential in accounting for the cost of reductions.

Path-label Invariant: If there are two distinct paths that have the same ports as endpoints, then the paths have different labels.

Theorem 3.1 *Graph reduction preserves the path-label invariant.*

Proof. We consider the three possible reductions in the abstract algorithm:

1. The interaction between a pair of sharing nodes.
2. The interaction of a sharing node with a primary node.
3. A β-reduction.

The interaction between a pair of sharing nodes has no effect on the paths in the graph. Thus the invariant is trivially preserved.

When a sharing node interacts with a primary node, paths are duplicated. Since the primary node is duplicated as well, creating new endpoints, the invariant is preserved.

The most complicated case is β-reduction, which removes primary nodes, and hence endpoints, from the graph. Parallel β-reduction causes two fusings of paths: one on the *function call* side, and one on the *argument side*. On the function call side the paths ending in the left port of the @ node are fused with the path through the redex and the paths beginning at the left port of the λ node. By induction each path has a unique configuration of labels and endpoints. Because each path ending at the @ or λ shares a common endpoint there, it is the other endpoint, which is preserved by the β reduction, that makes the configurations of labels and endpoints unique. Thus, since the underlined family index acts as a unique demarcator, the configurations of labels and endpoints for the fused paths are pairwise distinct. The configurations of labels and endpoints created on the argument side are similarly pairwise distinct. However, could a label $y\underline{w}p$ on the function side be identical to another label $q\underline{w}z$ on the argument side?

Since the label $\underline{w}$ is new and unique, it acts as an unambiguous demarcator, hence we would have $y \equiv q$ and $p \equiv z$. Were this the case, readback would show that the label p of an application was identical to the label z of a bound variable. However, the following argument shows that such a situation is impossible.

In any labelled reduction, each term in the reduction has the following property: labels on variables are disjoint from labels on lambda abstractions and applications. The claim is certainly true of initial terms, where each subexpression has a unique label; we need only show that the condition is preserved by labelled reduction. We use E, F to denote non-variable terms; ℓ, m, n, p, q to denote labels; and proceed by induction, using an obvious syntactic case analysis:

$(\lambda x.E)^\ell F \Rightarrow E^{\underline{\ell}}[x := F^{\underline{\ell}}]$ No variables are relabelled; expressions that are relabelled all have the new label $\underline{\ell}$ as a subword. The condition is thus preserved.

$(\lambda x.x^\ell)^m F \Rightarrow x^{\ell\underline{m}}[x := F^{\underline{m}}] \equiv F^{\underline{m}\ell\underline{m}}$ No variables are relabelled; the relabelled residual has the new label $\underline{m}$ as a subword. The condition is once more preserved.

$(\lambda x.y^\ell)^m F \Rightarrow y^{\ell\underline{m}}$ Again, the variable is relabelled uniquely.

$((\lambda x.E^\ell)^m y^n)^p \Rightarrow E^{\ell\underline{m}p}[x := y^{n\underline{m}}]$ Only this case uses the inductive hypothesis. The residual gets the label $\ell\underline{m}p$, and any occurrence x^q in E is rewritten to $y^{n\underline{m}q}$.

By the induction hypothesis, we know p and q are different labels, hence so are $\ell\underline{m}p$ and $n\underline{m}q$. (We could as well deduce $\ell\underline{m}p \neq n\underline{m}q$ from the inductive hypothesis that $\ell \neq n$.)

This concludes the proof of Theorem 3.1. ∎

The previous theorem can be used, together with some basic facts about optimality, to give a unique characterization of every path in the graph. This characterization is used in the analysis of Section 5.

Definition 3.2 *For any λ or @ node* $\mathbf{n}$ *in an intermediate graph representing a step in the reduction, we define $\mathcal{I}(\mathbf{n})$, the* canonical identifier *of* $\mathbf{n}$*, as follows:*

- *if* $\mathbf{n}$ *is eliminated via a parallel β-step with family index w, then $\mathcal{I}(\mathbf{n}) = w$.*[3]
- *otherwise,* $\mathbf{n}$ *appears in the final graph: $\mathcal{I}(\mathbf{n})$ is defined as the ordinal for* $\mathbf{n}$*.*

Optimality tells us that for any intermediate form of the graph, at most two nodes can have the same canonical identifier w: the λ and @ node involved in the parallel reduction with index w.

Definition 3.3 *Given a path ρ with label w between the value port v of primary node* $\mathbf{m}$ *and the environment port e of a primary node* $\mathbf{n}$*, we define the* path identifier $\Pi(\rho)$ *to be the tuple* $\langle v, \mathcal{I}(\mathbf{m}), w, e, \mathcal{I}(\mathbf{n})\rangle$.

Lemma 3.4 *No two paths have the same path identifier.*

Proof. By optimality and Theorem 3.1. ∎

4 Fan-normal form

Definition 4.1 *A graph is in* fan-normal form *if it has no sharing nodes facing each other on interaction ports.*

In fan-normal form no interactions *between sharing nodes only* are possible. As a consequence, sharing nodes accumulate in binary trees of all positive or all negative nodes (a positive tree is shown in Figure 7). These trees are "stuck" on a free variable, or on the ports of @ and λ nodes (see Figure 8).

[3] We consider that when a primary node is duplicated, one of the primary nodes produced is the original, and the other is a fresh copy.

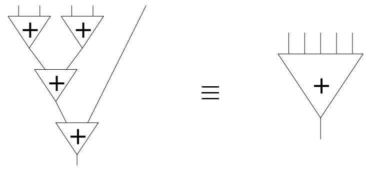

Figure 7: A tree of positive sharing nodes.

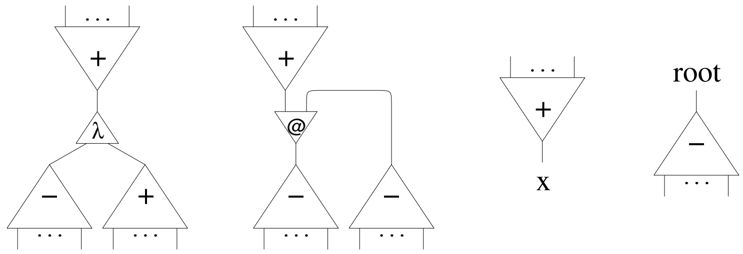

Figure 8: Fan-normal form.

Lemma 4.2 *A β-redex can be found in time proportional to the size of a graph in fan-normal form.*

Proof. Start at a λ node, and search to each of the leaf wires of the (positive) tree above the node: in the case of a redex, the leaf wire will be attached to an @ node, or will be the leaf of a (negative) tree leading to an @ node. We remark that without the fan-normal form, paths from an @ node to a λ node, as directed by the context semantics, can have exponential length, due to multiple entries and exits from sharing nodes along the path; see [LM96] for details. ■

Definition 4.3 *A* canonical β-reduction, *based on the fan-normal form, consists of the following sequence of steps: (1) identify a β-redex; (2) move relevant sharing nodes past the @ and λ node of the redex; (3) reduce the redex; (4) carry out as many interactions as possible between sharing nodes, not allowing the duplication of @ or λ nodes.*

The steps in a canonical β-reduction are shown in Figure 9. When the applied λ-abstraction is $\lambda x.x$, there is an additional interaction between the trees P' and Q'.

Theorem 4.4 *Canonical β-reduction preserves fan-normal form.*

Proof. By induction on the structure of the source term and the number of such reductions. ■

Since the goal of graph reduction is to reduce all the β-redexes, clearly we obtain a graph representation of the normal form of the term by considering only canonical β-redexes. The obvious and fundamental question is then: can fan-normal form be maintained without prohibitive cost while doing reductions? We address this question in the next section.

5 Accounting for fan interactions in β-reduction

We want to show that the total number of graph interactions involving @, λ, and sharing nodes is bounded by a polynomial in the sum of the following two parameters:

1. The number of unique labels generated during the reduction;

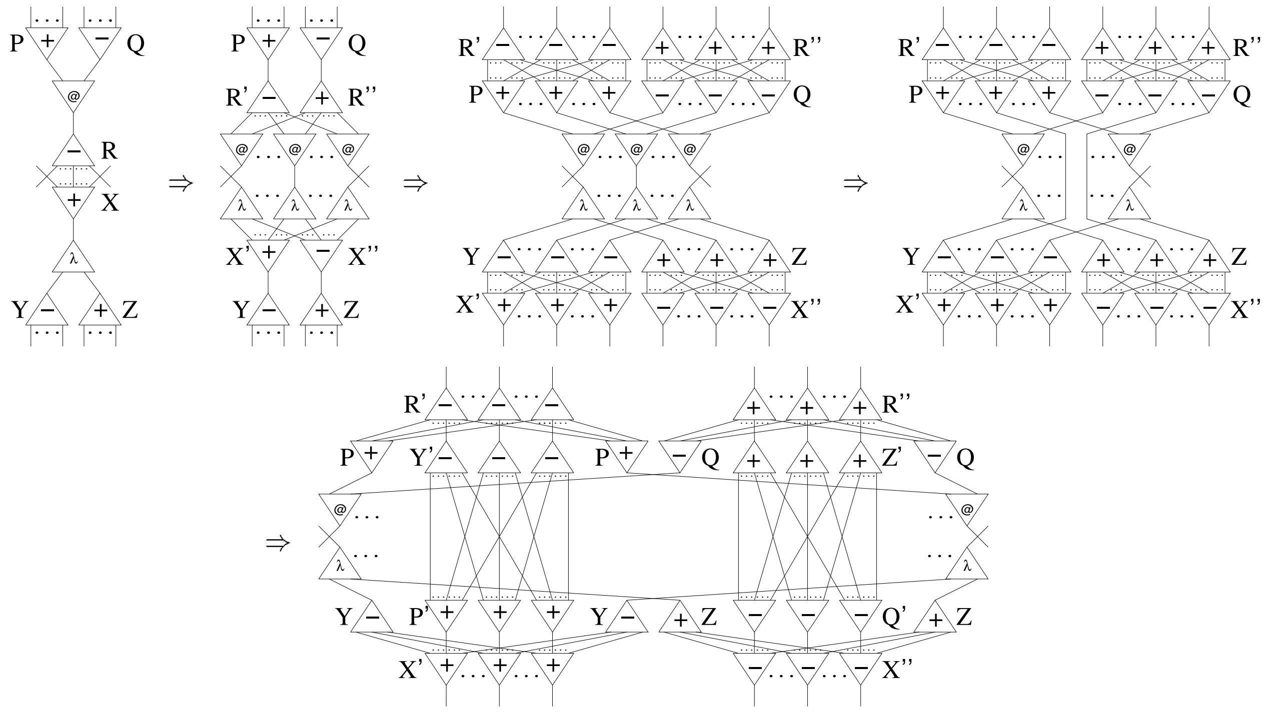

Figure 9: A canonical β reduction.

2. The number of @ and λ nodes in the graph representation of the normal form of the term.

While the second item above seems a peculiar and implementation-dependent parameter, we remark that it is certainly bounded by the size of the output, and that it can furthermore be replaced by a technical and complicated parameter based purely on labels. In this latter case, we need only make precise what is an unsharable pair of applications or abstractions in a labelled term, e.g., using Lamping's idea of *prerequisite chains* [Lam90].

We account for the cost of fan interaction and related parallel β-reduction by *charging* individual fan interactions to tuples made up of a family index and up to three path identifiers. Let n be the sum of the above parameters. The number of family indices is clearly $O(n)$. The number of path identifiers is $O(n^3)$, accounting for the label and the nodes at the two endpoints. Because there are only three kinds of port identifiers (interaction, left, right), the port identifiers only contribute a multiplicative constant factor to the charging scheme.

The bound we give in Theorem 5.4 is thus $O(n^{10})$. This bound is deliberately crude and without doubt can be made tighter, but for the moment we are not interested—the given bound both preserves polynomial relatedness, and facilitates the analysis. A further technical analysis will doubtless reduce the bound.

For tuples of labels $T = \langle t_1, \ldots, t_k \rangle$ and $U = \langle u_1, \ldots, u_\ell \rangle$ we write $T \circ U$ to denote the concatenated tuple $\langle t_1, \ldots, t_k, u_1, \ldots, u_\ell \rangle$.

The substeps of a canonical β-reduction consist of the interaction between a tree and a primary node, or between a tree of positive sharing nodes and a tree of negative sharing nodes. We begin with some observations about the cost of the interaction between trees of sharing nodes.

Lemma 5.1 *In the interaction between a positive tree with m sharing nodes and a negative tree with n sharing nodes, there are at most mn fan interactions, occurring when there are no annihilations* *between interacting fan nodes.*

When annihilations occur between trees of sharing nodes, the number of interactions decreases. Annihilation has at worst a linear effect, since two sharing nodes disappear, but also has the ability to fork trees onto different paths in the graph and thus eliminate any interaction. In a graph in fan-normal form, the number of leaves of a tree coincides with the number of paths through the tree, and is greater than the number of nodes in the tree. Thus we can use path identifiers to account for the number of nodes in a tree.

Definition 5.2 *We write $|T|$ to denote the number of (non-interactive) leaves of a tree T.*

Using this analysis of the cost of the interaction between trees of sharing nodes, we make the following association between the substeps of a canonical β reduction and tuples of path identifiers.

Moving R past the @ node (see Figure 9). Charge 1 to each of the pairs $\langle w\rangle \circ \Pi(r_i)$, where w is the family index of the reduction, and r_i is a leaf wire of R. We write this charge as

$$\sum_{1\leq i\leq |R|} \langle w\rangle \circ \Pi(r_i).$$

Moving R' past P. $|R'| = |R|$, therefore

$$\sum_{\substack{1\leq i\leq |R|\\ 1\leq j\leq |P|}} \langle w\rangle \circ \Pi(r_i) \circ \Pi(p_j).$$

Moving R'' past Q. Similarly, charge

$$\sum_{\substack{1\leq i\leq |R|\\ 1\leq j\leq |Q|}} \langle w\rangle \circ \Pi(r_i) \circ \Pi(q_j).$$

Dual charges for moving sharing nodes past λ-node.

$$\sum_{1\leq i\leq |X|} \langle w\rangle \circ \Pi(x_i) + \sum_{\substack{1\leq i\leq |X|\\ 1\leq j\leq |Y|}} \langle w\rangle \circ \Pi(x_i) \circ \Pi(y_j) + \sum_{\substack{1\leq i\leq |X|\\ 1\leq j\leq |Z|}} \langle w\rangle \circ \Pi(x_i) \circ \Pi(z_j).$$

Parallel β-step. Charge 1 to $\langle w\rangle$.

Moving Y past P. By Theorem 3.1, the $\Pi(y_i)$ and $\Pi(p_j)$ are unique; we then charge

$$\sum_{\substack{1\leq i\leq |Y|\\ 1\leq j\leq |P|}} \langle w\rangle \circ \Pi(y_i) \circ \Pi(p_j).$$

Moving Q past Z. Similarly, charge

$$\sum_{\substack{1\leq i\leq |Q|\\ 1\leq j\leq |Z|}} \langle w\rangle \circ \Pi(q_i) \circ \Pi(z_j).$$

Moving P' past Q'. If the function applied is $\lambda x.x$, we have the additional cost of moving P' past Q'. A single λ-node can represent many copies of the identity function. In particular, the number is bounded by the number of possible connections between the left and right ports of the λ-node, *i.e.*, the minimum of $|Y|$ and $|Z|$. We arbitrarily choose to charge against the paths of Y:

$$\sum_{\substack{1\leq i\leq |Y|\\ 1\leq j\leq |P|\\ 1\leq k\leq |Q|}} \langle w\rangle \circ \Pi(y_i)\circ \Pi(p_j)\circ \Pi(q_k).$$

Lemma 5.3 *The charges made to any tuple $\langle \alpha_1,\ldots,\alpha_k\rangle$ above are bounded by a constant.*

Proof. By appealing to optimality and uniqueness of path identifiers. ∎

Theorem 5.4 *The total number of fan interactions is $O(n^{10})$, where n is the sum of the number of labels generated during reduction, and the number of primary nodes in the final graph.*

Corollary 5.5 *In reduction to constants, the number of fan interactions is bounded by a polynomial in the number of labels generated.*

6 Conclusions

We have made a first attempt at understanding the global dynamics of graph reduction mechanisms for implementing optimal evaluation. We like the straightforward, mundane nature of our arguments, and their reliance on simple invariants. We hope eventually to develop an understanding of optimal evaluation that makes its dynamics and its correctness as easy to understand as quicksort. No category theory is necessary—it's only an algorithm!

A plausible corollary to Theorem 5.4 would be an analogous result that counts the work done by *all* graph operations, including bracket and croissant operators that maintain the fidelity of sharing node indices. We conjecture that a similar theorem holds by replacing the number of unique labels with the sum of their respective *lengths*. The intuition is simply that the duplication of bracket and croissant operators corresponds closely to the commuting of labels across sharing nodes; the metric bounds the bookkeeping interactions because while nodes can annihilate, labels cannot get shorter. We also hope to demystify the readback procedure, based on insight we have about identifying redexes; the ability to do so offers the possibility of controlled introduction of graph operations that destroy optimality if misused, and violate context semantics at the naive level of local rewriting, but obey the semantics at a global level.

Acknowledgments. We thank Jean-Jacques Lévy, Georges Gonthier, and Peter Lee for their encouragement, Martin Abadi for answering some questions we had concerning context semantics, and Olivier Danvy for valuable comments on an earlier draft.

References

[Asp94] Andrea Asperti. *Linear logic, comonads, and optimal reductions.* **Fundamenta Informaticae** 22:1, 1994.

[Asp95] Andrea Asperti. *Paths, computations, and labels in the λ-calculus.* **Theoretical Computer Science** 142:2 (May, 1995), pp. 277–297.

[Asp96] Andrea Asperti. *On the complexity of beta-reduction.* **1996 ACM Symposium on Principles of Programming Languages**, pp. 110–118.

[Bar84] Hendrik Barendregt. **The Lambda Calculus: Its Syntax and Semantics**. North Holland, 1984.

[vEB90] Peter van Emde Boas. *Machine models and simulation.* **Handbook of Theoretical Computer Science**, volume A, pp. 1–66. North Holland, 1990.

[Chu41] Alonzo Church. **The Calculi of Lambda-conversion**. Princeton University Press, 1941.

[Fie90] John Field. *On laziness and optimality in lambda interpreters: tools for specification and analysis.* **1990 ACM Symposium on Principles of Programming Languages**, pp. 1–15.

[FS91] Gudmund S. Frandsen and Carl Sturtivant. *What is an efficient implementation of the λ-calculus?* **1991 ACM Conference on Functional Programming and Computer Architecture** (J. Hughes, ed.), LNCS 523, pp. 289–312.

[GAL92] Georges Gonthier, Martin Abadi, and Jean-Jacques Lévy. *The geometry of optimal lambda reduction.* **1992 ACM Symposium on Principles of Programming Languages**, pp. 15–26.

[Gue95] Stefano Guerrini. *Sharing-graphs, sharing-morphisms, and (optimal) λ-graph reductions (draft).* Unpublished manuscript. June 16, 1995.

[Kat90] Vinod Kathail. *Optimal interpreters for lambda-calculus based functional languages.* Ph.D. Thesis, MIT, May 1990.

[Lam90] John Lamping. *An algorithm for optimal lambda calculus reduction.* **1990 ACM Symposium on Principles of Programming Languages**, pp. 16–30.

[LM96] Julia L. Lawall and Harry G. Mairson. *Optimality and inefficiency: what isn't a cost model of the lambda calculus?* **1996 ACM International Conference on Functional Programming**, pp. 92–101.

[Lévy78] Jean-Jacques Lévy. *Réductions correctes et optimales dans le lambda-calcul.* Thèse d'Etat, Université Paris 7, 1978.

[Lévy80] Jean-Jacques Lévy. *Optimal reductions in the lambda-calculus.* **To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism**, (Jonathan P. Seldin and J. Roger Hindley, editors), pp. 159–191. Academic Press, 1980.

[Mac95] Ian Mackie. *The geometry of interaction machine.* **1995 ACM Symposium on Principles of Programming Languages**, pp. 198–208.

[Wad71] Christopher Wadsworth. *Semantics and pragmatics of the λ-calculus.* Ph. D. Thesis, Programming Research Group, Oxford University.