Optimality and inefficiency : what isn't a cost model of the lambda calculus?

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

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

Abstract

We investigate the computational efficiency of the *sharing graphs* of Lamping [Lam90], Gonthier, Abadi, and Lévy [GAL92], and Asperti [Asp94], designed to effect so-called *optimal evaluation*, with the goal of reconciling optimality, efficiency, and the clarification of reasonable cost models for the λ-calculus. Do these graphs suggest reasonable cost models for the λ-calculus? If they are optimal, are they efficient?

We present a brief survey of these optimal evaluators, identifying their common characteristics, as well as their shared failures. We give a lower bound on the efficiency of sharing graphs by identifying a class of λ-terms that are normalizable in $\Theta(n)$ time, and require $\Theta(n)$ "fan interactions," but require $\Omega(2^n)$ bookkeeping steps. For [GAL92], we analyze this anomaly in terms of the dynamic maintenance of deBruijn indices for intermediate terms. We give another lower bound showing that sharing graphs can do $\Omega(2^n)$ work (via fan interactions) on graphs that have no β-redexes. Finally, we criticize a proposed cost model for λ-calculus given by Frandsen and Sturtivant [FS91], showing by example that the model does not take account of the size of intermediate forms. Our example is a term requiring $\Theta(2^n)$ steps while having proposed cost $\Theta(n)$. We propose some cost models that both reflect this parameter, and simultaneously reconcile key concepts from optimal reduction.

1 Introduction

The dual fields of *algorithmics* and *semantics* have matured with very little intercommunication, although both are devoted to the foundations of computation. Universal computation, as developed by Turing, Church, Herbrand, Gödel, Post, among others, divides naturally into those models of computation that are *machine based* (Turing machines, register machines, and their variants), and those that are *language based* (λ-calculus, computing with equations *à la* Herbrand-Gödel, etc.). From this fundamental division one sees the algorithmic focus on machines, as well as the semantic focus on languages.

Optimal evaluation: *Optimal reduction* [Lévy78, Lévy80], the idea of evaluating programs "correctly" (producing a normal form if there is one) while not "duplicating work," attempted to simultaneously achieve the goals of computational efficiency and semantic clarity. These goals sit squarely between the two cultures of algorithmics and semantics. But how efficient is optimal reduction? To understand its efficiency, we also need to consider reasonable cost models for λ-calculus.

It was an open question for some fifteen years whether optimal reduction strategies exist, answered affirmatively by Lamping [Lam90], Kathail [Kat90], and in a simplified way by Gonthier, Abadi, and Lévy [GAL92] (henceforth, GAL), as well as Asperti [Asp94]. The solution of GAL was appealing because it also gave a static semantics to λ-calculus in the spirit of Girard's *geometry of interaction* [Gir88]; this semantics has also been recently investigated as an implementation technique by Mackie [Mac95]. All these solutions are contributions to compiler technology since they are just rarefied forms of graph reduction. In this paper, we compare and analyze the computational features of proposed optimal evaluators, explaining what design features are common, and what computational resources are required to realize these design features. While these schemes are optimal, we show that they are all inefficient.

Machine models: Compiler and interpreter technology relate the "two cultures" of machines and languages at a *theoretical* level as well as at a *practical* one. There is a substantial theory literature on *machine simulations* (see, e.g., the survey paper [vEB90]), founded on the *Invariance Thesis*, the modern-day version of Church's Thesis: *"Reasonable" universal machines can simulate each other within a polynomially-bounded overhead in time and a constant-factor overhead in space.* Machine simulation is just the theorist's version of hardware emulation, not really different from compiler and interpreter technology except in the complexity of the source language. How efficient are optimal evaluators—do they simulate machine models well? What are the relevant cost models? More generally, can we speak of the complexity of a functional program in a machine-independent way?

[*]Work performed at Brandeis University, supported by ONR Grant N00014-93-1-1015 and NSF Grant CDA-9504288.

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

These kinds of questions were asked in a provocative paper by Frandsen and Sturtivant [FS91], who proposed various implementation-independent cost models for the λ-calculus, and showed that several well-known implementation techniques (Turner combinators [Tur79], Hughes supercombinators [Hug82]) are too inefficient to satisfy these cost models. They leave as an open question whether their cost models, while conscientiously justified, are indeed attainable. This paper continues their work by analyzing optimal evaluators in the same vein. We discover that optimal reduction fails to be efficient in the sense of Frandsen and Sturtivant, even though their cost models are based on the idea of *parallel reduction*, borrowed from the optimal evaluation literature. While [FS91] implicitly endorses parallel reduction as a good idea with respect to efficiency, it does not analyze whether any additional computation (for example, bookkeeping to facilitate parallel reductions) is computationally prohibitive. Rather than dispense with optimal reduction as hopelessly inefficient, we propose modifications to their cost model that we believe to be more realistic. Furthermore, we argue that realizing this revised model is intimately related to the bookkeeping problems inherent in proposed optimal reduction schemes.

Technical contributions: We present optimal reduction *à la* Lamping, GAL, and Asperti, making the following observations:

- We present a taxonomy of operators from each scheme, indicating which of myriad interactions are computationally explosive. We link the problematic form of bracket interaction to deBruijn indices (i.e., lexical addresses), and to a beginning analysis of the cost of substitution in optimal reduction.
- We present a lower bound on the efficiency of Lamping, GAL, and Asperti, by identifying classes of terms requiring $\Omega(2^n)$ graph reduction steps, with only $\Theta(n)$ interactions of so-called *fan nodes*, where we regard such fan interactions, which include β-reduction, as the "essential work" being performed. The *additional* required bookkeeping overhead required swamps the real work, and for an infinite class of terms approaches the efficiency of normal order evaluation. Whether such bookkeeping can be made efficient remains the biggest open question in this area.
- If we further divide the "real work" done by optimal reduction strategies into parallel β-steps and complementary fan interactions, we present another lower bound on efficiency, showing that such fan interactions may grow exponentially in the number of parallel β-steps, even when the corresponding term is in normal form. Knowing whether a graph-encoded term is in normal form requires a *global* analysis that local reduction rules seem ill-equipped to provide. Sharing is undone via duplication in optimal reduction to facilitate β-reduction: how can this duplication be controlled?
- We analyze the cost model of Frandsen and Sturtivant based on parallel reduction, and provide concrete evidence that it is not met by any of the optimal reduction schemes. We further answer a conjecture of Frandsen and Sturtivant concerning the polynomial relatedness of two of their proposed cost models, showing they are not so related. The essence of these analyses is the identification of a class of terms that require $\Theta(2^n)$ fan interactions to reduce the corresponding term to normal form, while requiring only $\Theta(n)$ parallel β-steps. A very similar result was recently and independently derived by Asperti [Asp96] in his analysis of the inherent complexity of β-reduction.
- We observe that normal order evaluation is an acceptable machine model in the sense of the orthodox Invariance Thesis. Thus the debate over cost models is largely fine structure, or the thesis is too crude. The key observation is that sharing and machine simulation have little in common.
- We propose two cost models for the λ-calculus based on the *labelled* λ-calculus, and we use them to motivate further work in this area.

While there is a developing literature on implementations of optimal evaluation based on graph reduction, we explore its finitary dynamics from an algorithmic perspective. Because a lot of this literature is technical, we also provide a gentle introduction to optimal reduction that we hope is digestible to the general reader. While [GAL92] parodied Lamping as "TV Digest," we found their presentation, however brilliant, an impenetrable synthesis of the traditions of Bourbaki and Hunter Thompson. As much as possible, we return to the TV Digest vernacular.

2 Sharing graphs for λ-calculus

Traditional technology for evaluating functional programs can be categorized according to the *efficiency* and *safety* of β-reduction. When the value of an expression is multiply used, efficiency requires that it not be evaluated more than once. When the value of an expression is never used, safety requires that it not be evaluated. Graph reduction (see, e.g., [PJ87]), used to implement lazy functional languages, compromises between these two goals: β-reduction binds *all* occurrences of the parameter to a *single* shared graph representing the argument. When the value of some occurrence of the parameter is requested, the argument is evaluated. Sharing causes all other occurrences of the parameter to return a result immediately.

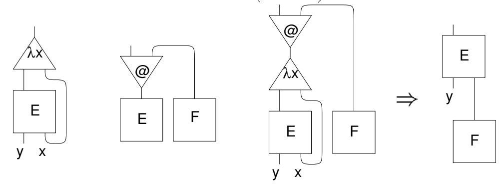
This strategy works well until the shared value is a λ-abstraction and is applied. Since each use of the λ-abstraction may apply it to a different argument, the body of the λ-abstraction cannot be shared after β-reduction. When a λ-abstraction is applied, a new *copy* of its body is instantiated with the argument. This copying may be excessive, since other computations in the λ-body not depending on the argument could still be shared.

Lamping [Lam90] had the great idea of economizing via *partial sharing* of the body. In other words, instead of copying the whole body, just copy the λ-node at the top of the graph representing the function, and continue to share the body of the function. This scheme introduces the sharing of *evaluation contexts* (multiple values evaluated concisely in the same shared context) as well as the more obvious sharing of *values* (one value evaluated concisely in multiple contexts), and makes these two kinds of sharing dually complementary to effect duplication. Since no redex is ever duplicated, Lamping's evaluator is thus efficient in the number of β-reductions and safe, or *optimal.* The algorithmic variations on his fundamental observation (e.g, [GAL92, Asp94]) consist of alternative low-level implementations of his *sharing nodes.* We now briefly sketch Lamping's basic idea, and variations on its implementation.

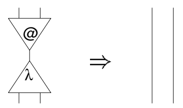
2.1 First version: no sharing allowed

For a moment, forget about sharing, and consider a first version that implements a λ-calculus where bound variables occur at most once. We see two kinds of ternary nodes: λ nodes and @ nodes.[1] A value is given by a wire. The terms $\lambda x.E$, EF, and $(\lambda x.E)F$ are represented by the following graphs, where graphs for E and F are similarly defined:

$\lambda x.E$ EF $(\lambda x.E)F$



The "bound variable wire" for x in $\lambda x.E$ wraps around to the right side of the λ node in the graph; a free variable wire to y also appears. This "wrap around" feature allows β-reduction to be defined by a simple rewrite rule, as shown by example in the reduction $(\lambda x.E)F \Rightarrow E[F/x]$, where the λ and @ nodes annihilate, and their wires are fused. The rule is shown below:



2.2 Second version: sharing with omniscient fan nodes

Now let's add *fan nodes* to implement sharing, for example the term $\Delta \equiv \lambda x.xx$ that has two references to x. Figure 1 shows the reduction of $\Delta\Delta$. After step 1, the value of the second Δ is shared by a fan node.

A new graph rule is introduced to push the sharing node past the λ node, *partially* sharing the λ-body (value sharing) as well as the context of its bound variable wire (context sharing). (Dually, one can partially share an application.) In thus unsharing the λ node, we create *two* λ nodes—one for each use—and two sharing nodes. The left *positive* node shares the body of the λ-expression; the right *negative* node shares the context provided by the body with two uses (step 2). The pair of nodes serves as a kind of switch, where setting both to "left" gives $F(M)$, and setting both "right" gives $F(N)$. The context semantics given in [GAL92] is an elaborate mechanism for iteratively setting these switches.

[1] We annotate λ-nodes with the name of the bound variable for clarity.

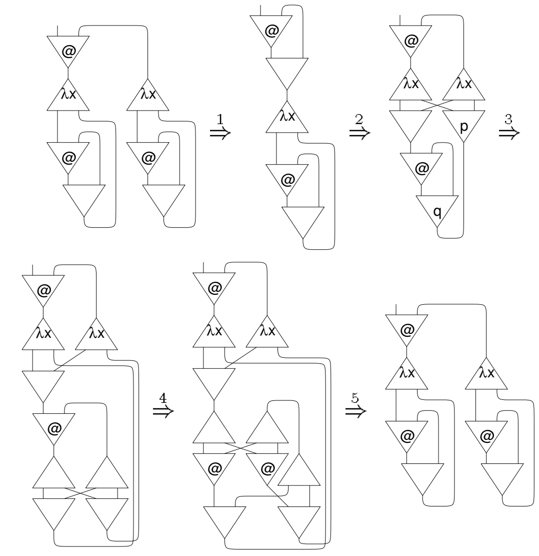
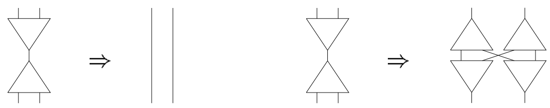


Figure 1. The reduction of $\Delta\Delta$

Next, we consider the interaction of fan nodes (p) and (q) above. When fan nodes interact that are (omnisciently) the *same*, they annihilate each other, while when *different*, they duplicate each other:



In this case, they are different (step 3). The fan node under the @ then duplicates the application, and its copies annihilate the fan nodes on the other side (steps 4 and 5).

How do fan nodes facing each other know if they are the same or different? This question is the key technical implementation issue to be resolved in optimal reduction. Its answer is what makes optimal reduction schemes so hard to understand: the respective *tours de force* of Lamping, GAL, and Asperti each resolve it differently. *They all agree which fans are the same and different; they only disagree in how this information is determined.* The different solutions are all variations on the same theme: a numerical *index* is maintained by each fan node, and two nodes are the same if they have the same index. In each implementation, new graph operators are introduced whose sole purpose is *bookkeeping*: the dynamic mutation of fan node indices.

2.3 Third version: sharing with bookkeeping nodes

In the *bus system*, the single wires of the previous diagrams are replaced by a bus of wires, where graph operators travel on particular wires of a bus. The bus gives a physical interpretation to the idea of *index* mentioned above—the index is just the position of the operator on the bus. Additionally, new graph operators, called *croissant* and *bracket*, are introduced to mutate indices of other operators. Croissant and bracket nodes mark the boundary "ports" of regions of the sharing graph. As a sharing node enters such a region, they modify its index appropriately. Each of the different implementations—GAL, Asperti, Lamping—chooses particular kinds of regions to be annotated with croissant and bracket nodes at port positions.

Now for the fine print: A fan node joins two buses. Fan nodes implement sharing, as well as λ and @. A bracket combines two wires into one. A croissant creates a new wire.

Nodes can appear in any orientation. In each case we have drawn the node with its *interaction port* facing downwards. Two nodes only interact if their interaction ports are facing each other. The rules for their interaction are shown in Figure 2.

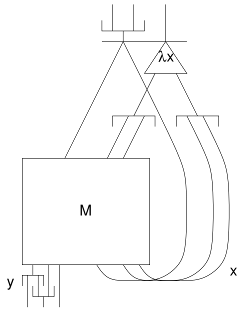
By annotating each node with an index, indicating the number of wires to its right, we can abbreviate the multi-wire bus using a single wire. The rules for the one-wire bus can be derived from the rules for the multi-wire bus. In particular, the index of a node is incremented when it passes to the left of a bracket, and decremented when it passes to the left of a croissant.

We refer to interactions involving brackets and croissants as *bookkeeping*. The complementary *fan interactions* contain a subset consisting of parallel β-steps, since fan nodes can be used to implement λ and @ as well as sharing.

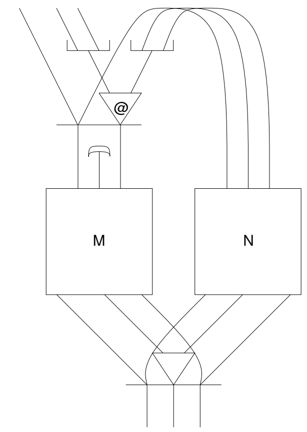
2.3.1 GAL

GAL's translation of lambda terms into graphs is as follows:

$\lambda x.M$



MN



This encoding uses brackets in three quite different ways. *Stack* brackets split the leftmost wire. These do not affect the index of any sharing node; we do not consider them further. *Redex* brackets travel next to λ and @ nodes like pilot fish on a shark. *Free-variable* brackets, which are the most problematic, manipulate an analogue of deBruijn indices. In the initial graph, the number of free-variable brackets equals the deBruijn index of the occurrence of the identifier.

In GAL, the index of a sharing node is the number of nested λ-bodies the sharing node has begun to copy. This index is maintained by the inward-facing redex and free-variable brackets surrounding all entrances to a λ-body, as shown by the translation of a λ-abstraction above. Entrance is via one of the extremal *ports* of the graph, located at the *root*, the bound variable, and any free variables. At the root and the bound-variable ports the redex brackets increment the index of an entering sharing node. When a sharing node enters at a free-variable port, its index is incremented by the free-variable bracket.

The graph simulation of β-reduction must (1) substitute the argument for the parameter, and (2) adjust the deBruijn indices of variables, represented in unary as stacks of free-variable brackets. During β-reduction in graphs, redex brackets at the λ-node and @ node annihilate each other, connecting root to λ-body, and parameter to argument, achieving the substitution. In the λ-calculus, β-reduction of $(\lambda x.E)F$ affects deBruijn indices as follows: the index of every free variable in E is reduced by 1, and for each occurrence of x with index k in $\lambda x.E$, the index of every free variable in the substituted F is incremented by $k-1$. In GAL, the brackets on the free-variable ports of the λ-body are not eliminated during β-reduction, nor does the λ-node have direct access to them; thus, the body is delineated even after annihilation. To adjust the deBruijn indices, a croissant node introduced at the application migrates lazily to the free variable ports, decreasing all deBruijn indices by 1. Brackets at the parameter port migrate lazily to the free-variables ports of the substituted argument, and are appended to the brackets found there.

In the bureaucracy of index management, the "$1-1$" of a croissant hitting a bracket is *not* reduced to zero, even though it has zero effect on the index of passing nodes, since such bureaucratic junk may need to annihilate with other nodes. This anomaly is the essential bottleneck of GAL, and of optimal reduction schemes more generally: details vary with respect to Lamping and Asperti, but the essential problem is the same.

2.3.2 Asperti

In GAL, the index of a sharing node is the number of λ-bodies in which the sharing node is nested. Asperti counts applications instead: the index of a sharing node is the number of argument positions in which the sharing node is nested. Asperti's algorithm begins by assigning an index to each subterm. The index of the entire term is 0. The indices of the subterms are calculated as follows:

$$
\begin{aligned}
\underline{(EF)}_n &= (\underline{E}_n \underline{F}_{n+1})_n \\
\underline{(\lambda x.E)}_n &= (\lambda x.\underline{E}_n)_n \\
\underline{v}_n &= v_n
\end{aligned}
$$

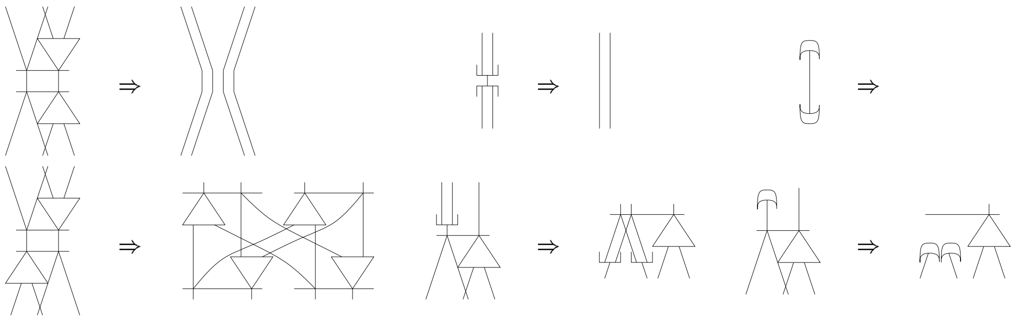
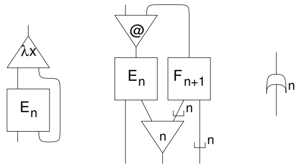


Figure 2. Interaction rules

The translation of an indexed term into a graph is shown below.

$\lambda x.E_n$ $\quad E_n F_{n+1}$ $\quad v_n$



If an occurrence of identifier v has index k, and the λ-abstraction binding v has index ℓ, then the wire coding the occurrence holds a sequence of $k-\ell$ brackets. When the identifier is bound, these brackets propagate over the value, incrementing the indices to reflect the new argument-nesting depth of each occurrence.

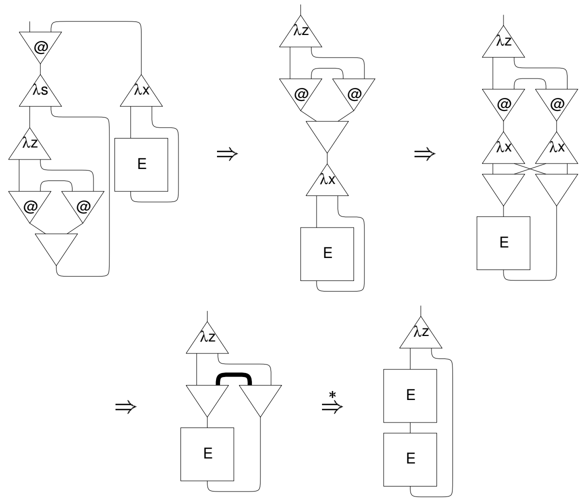
The sequence of brackets representing an identifier ends in a croissant. GAL uses a croissant in an application because a β-reduction causes one λ-abstraction to disappear. Here the croissant reflects the fact that a β-reduction causes one application to disappear. The croissant is directly part of the translation of an identifier, rather than being introduced by an application. It has no effect, however, until the identifier is bound by an application.

3 Lower bounds

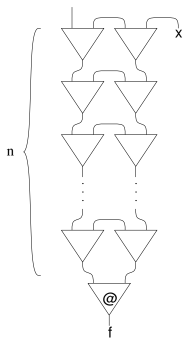
The exponential lower bounds we present on the efficiency of optimal reduction are all based on application of the term $C_n \equiv \lambda x.\overline{2}(\overline{2}(\ldots(\overline{2}x)\ldots))$ to different arguments, where the n applications of $\overline{2} \equiv \lambda s.\lambda z.s(sz)$ refer to the familiar Church numeral. We use C_n to construct a term that normalizes in $\Theta(n)$ β-steps, but requires $\Omega(2^n)$ bookkeeping steps, mostly involving bracket and croissant interactions. We then use C_n to construct another term that contains no further β-redexes after $\Theta(n)$, but the graph representation requires $\Omega(2^n)$ further fan interactions to normalize. These examples show that the bookkeeping overhead needed to implement fan interactions (including parallel β-steps) can grow exponentially in the number of such interactions, and that the local reduction rules are not sufficient to efficiently detect normal form.

3.1 Computing an exponential iterator

Consider the reduction, in the 1-wire bus with omniscient fan nodes, of $(\lambda s.\lambda z.s(sz))(\lambda x.E)$, where $\lambda x.E$ is closed and normalized:

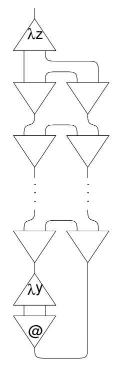


We use $\overset{*}{\Rightarrow}$ to denote a sequence of reductions. In this last sequence, the linked fan nodes *duplicate* the graph E between them; the boldface wire linking the fan nodes says to connect the x port of the first copy to the root of the second copy. This configuration of fan nodes is a key succinctness primitive: for example, here is how to code 2^n applications of f to x in a graph of size $\Theta(n)$:

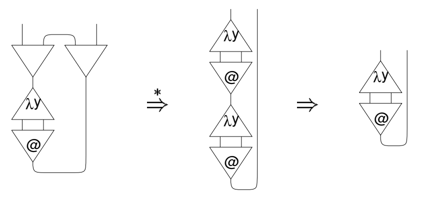


3.2 First analysis: bookkeeping overwhelms fan interaction

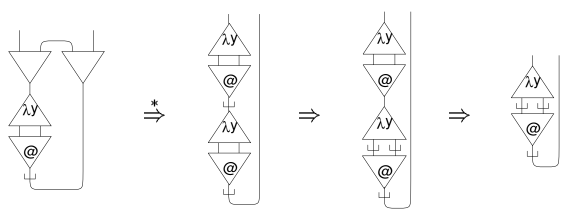
Consider the term $E \equiv C_n(\lambda x.\lambda y.xy)$. Since a simple calculation shows $(\lambda s.\lambda z.s(sz))(\lambda x.\lambda y.xy) \triangleright \lambda z.\lambda y.zy$, clearly E normalizes in $\Theta(n)$ β-steps. The initial graph quickly gets rewritten to:



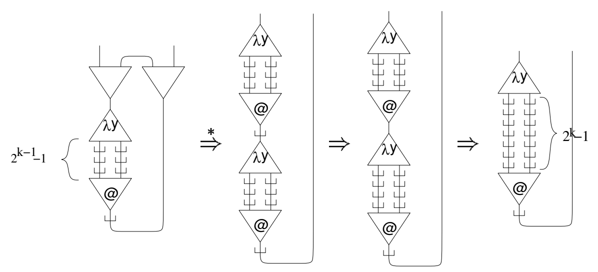
Via the reduction sequence



we see that the graph normalizes in $\Theta(n)$ parallel β-steps as well. The duplicated term has a constant number of @, λ, and fan nodes, suggesting that a constant amount of copying work is done at each of n iterations. This is not the case! The pairs of fans duplicate whatever is between them, *including bookkeeping nodes*. If the duplicates do not annihilate, the copying work doubles on each iteration. The first iteration of GAL looks like this:



The kth iteration looks like:



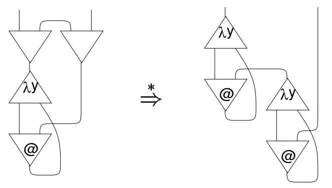
Because Asperti also uses brackets and croissants in the translation of $\lambda x.\lambda y.xy$ (and indeed, unlike GAL, in the translation of $\lambda x.x$) the reduction in Asperti's system behaves similarly.

Theorem 1. *GAL and Asperti take $\Omega(2^n)$ bookkeeping steps to effect n parallel β-steps.*

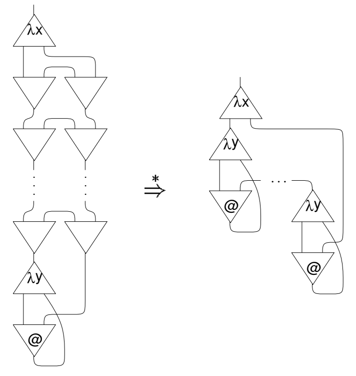
In the reduction of the term $C_n(\lambda x.\lambda y.xy)$, there are complexes of brackets and croissants of size $\Omega(2^n)$ that travel through the graph, even though their effect on wire indices is $O(1)$. The complexes are designed to readjust deBruijn indices; they form a kind of base 1 with no subtraction, where $+$ is a bracket and $-$ is a croissant. We can write, say, 2, as $++$, or $+--++$, or as any string with two more $+$ than $-$, *without* a simplification rule that rewrites substrings $+-$ and $-+$ to ϵ. This redundancy has suggested various optimizations, which we assess in Section 4.

3.3 Second application: locality and unnecessary fan interaction

Consider the difference between putting the pair of fans around the ports of $\lambda y.xy$ and $\lambda y.yx$. The first case is the basis of Section 3.2, and causes the construction of a β-redex. The second case causes duplication, but no β-redex is made:



To iterate this peculiarity, we reduce $C_n(\lambda x.\lambda y.yx)$. The compact encoding we really want is given by the first intermediate form below. This graph represents the term $\lambda y.y(\lambda y.y(\lambda y.y(...)))$, which is in normal form. To normalize the graph, the sharing nodes need to be propagated, with an exponential explosion in graph size:



The point of "unsharing" by propagating fan nodes is to partially duplicate λ and @ nodes in order to create β-redexes. One then needs a *global strategy* for propagating the right fan nodes. The beauty of graph reduction lies in its dependence on *local* rewrite rules, but therein lies a limitation, since it is insufficient to identify useful work. Detecting redexes seems to require a static analysis like GAL's context semantics, which can be very expensive.

Theorem 2. *Optimal evaluators allow $\Omega(2^n)$ useless fan interactions where there are no β-redexes to reduce.*

Asperti has suggested that we avoid this anomaly by considering only terms that normalize to constants [Asp96]. In this case, all functions are used, and all applications become redexes. But what if we remove this assumption—can sharing be implemented with minimal overhead and no implicit foreknowledge that all functions will be applied?

4 Optimizations

The inefficiency in the reduction of $C_n(\lambda x.\lambda y.xy)$ comes from the duplication of the brackets and croissants introduced in the translation of $\lambda x.\lambda y.xy$. Brackets and croissants are only useful to adjust the indices of sharing nodes. Because $\lambda x.\lambda y.xy$ contains no sharing nodes, the bookkeeping nodes serve no purpose. Asperti and Lamping have each proposed rules to eliminate these nodes. While their approaches improve performance for $\lambda x.\lambda y.xy$, they do not eliminate the problem of excess duplication in general.

4.1 Asperti

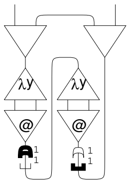
Asperti [Asp95] observed, by way of category theory, that we can replace a croissant atop a bracket by a single wire, when we can guarantee that neither the bracket nor the croissant ever annihilate with any node in the graph. Such brackets and croissants are called *safe*. In the initial graph, every bracket and croissant is safe. A bracket or croissant becomes unsafe when it propagates over a lambda node. It becomes safe again when it reaches a safe node of lower index. So that unsafe nodes can reach safe nodes, new rules should be introduced allow brackets and croissants to move from the root of an application to the function and argument positions (such rules are not mentioned by Asperti). If these optimization rules are applied whenever possible, this system can reduce $C_n(id)$ in $\Theta(n)$ steps, where id is any η-expansion of $\lambda x.x$.

4.2 Lamping

Although Lamping's algorithm [Lam90] preceeded both GAL and Asperti, it can be seen as a variant of GAL with Asperti's optimization rules built in. Like GAL, Lamping uses brackets to mark free-variable ports, and croissants to cancel free-variable brackets when a λ abstraction is applied. Lamping uses two kinds of brackets: round brackets that are safe, and square brackets that are not safe. Lamping's algorithm does contain the rules to allow croissants and unsafe brackets to move from the root of an application to the function and argument positions. While at first glance, the number of Lamping's rules seems daunting, most follow the bus-oriented rules described in Section 2.3, with additional rules for manipulating and optimizing safe brackets. Like Asperti, if the rules are applied in the right order, Lamping can reduce $C_n(id)$ in $\Theta(n)$ steps, where id is any η-expansion of $\lambda x.x$.

4.3 GAL

Although GAL is based on Lamping, it is not clear how to extend the optimization to GAL. In the Lamping translation of source terms, a croissant is placed over the root of any lambda abstraction containing a free variable. This croissant does not move until the lambda abstraction is applied. If there is no sharing, it is then free to move through the term to annihilate the free-variable brackets. In GAL the croissant is introduced by the translation of an application. If in the source term the function position is a free variable, the croissant gets stuck on top of these free-variable brackets. In our example we wind up with the following (omitting some of the bookkeeping nodes).



The boldface croissant should cancel the boldface free-variable bracket, but the free-variable bracket below the croissant is perpetually in the way.

4.4 More trouble

The efficacy of the optimization relies on the ability of the croissant to reach the corresponding free-variable bracket so they can annihilate before being duplicated. However, the optimization, and any sophisticated variant of it (*e.g.*, making the rules for "safety" more complex), is useless if the croissant cannot reach the bracket. For example, in the term $C_n(\lambda x.\lambda y.(\lambda a.aa)(\lambda z.(xy)z))$, after $\Theta(n)$ fan reductions there are no more β steps. Because the croissant representing the application of $(\lambda y....)$ gets stuck on the sharing node of $(\lambda a.aa)$, it cannot cancel with the free-variable brackets, and exponential duplication is required. Thus

Theorem 3. *GAL, Lamping, and Asperti, with and without optimization, take $\Omega(2^n)$ bookkeeping to effect n parallel β-steps.*

4.5 Guerrini

Guerrini [Gue95] shows how Asperti's stacks of brackets and croissants can be represented as integers. Rather than cancelling croissants against brackets, he shows that at the bound variable port of a lambda node it is safe to combine these integers using addition. Unfortunately, as shown by the example in Section 3.2, by the time the brackets and croissants reach a bound variable port of a lambda node, the exponential duplication has already taken place.

5 Cost models

A cost model for the λ-calculus should be implementation independent, so it can be used as a metric to measure the efficiency of different implementations. Taste and aesthetics govern choice of a model, with a "sanity check" given by the Invariance Thesis: *"Reasonable" universal machines can simulate each other within a polynomially-bounded overhead in time and a constant-factor overhead in space* [vEB90].

This sanity check has two basic components: (1) if reducing E requires time t and space s, then there must be an implementation (say, on a Turing machine) using $P(t)$ time and cs space, for some fixed polynomial P and constant c; (2) there must be a coding of Turing machine computations as λ-terms, with accompanying reduction strategy, such that given *any* Turing machine requiring $T(n)$ time and $S(n)$ space to accept or reject an input of length n, the λ-calculus simulation of that Turing machine computation can be so effected in $P(T(n))$ time and $cS(n)$ space.

While the sanity check rules out the truly unrealistic cost models—the unattainable ones, as well as the overly costly—there is a large enough "slop factor" that cost models and strategies, typically regarded as inefficient, nonethless become admissible. For example, we have the following, disconcerting claim:

Theorem 4. *The cost of normal order evaluation, naively implemented, satisfies the invariance thesis.*

Namely, to reduce a term, write it down on a piece of paper. Write down successive terms derived from β-reduction, using a leftmost-outermost strategy: total ink (or paper area) used is the time cost, and maximum ink (or paper) used for any term is the space cost. This metric can be realized (with polynomial slowdown for time, and constant expansion for space) on a Turing machine. More surprisingly, an arbitrary Turing machine can be simulated efficiently, using your favorite coding—[FS91] essentially gives one; [HM94] gives another. The explanation of the efficiency of the simulation is simple: the simulation does not depend in any profound way on sharing.

In an attempt to define a more parsimonious cost model, Frandsen and Sturtivant [FS91] proposed the following: the cost of a reduction to normal form is the length of the the initial and final term, plus the number of parallel β-steps. They point out that several standard combinator-based implementations (e.g., [Tur79, Hug82]) require time exponential in the cost model. We observe that optimal reduction strategies, even without counting bookkeeping, fail to run polynomially in the Frandsen-Sturtivant cost metric.

5.1 Optimal evaluators fail the Frandsen-Sturtivant test

Our evidence against the Frandsen-Sturtivant model is the reduction of the term $F \equiv E(\lambda x.x)$, where $E \equiv C_n(\lambda s.\lambda z.s(sz))$. Since F has length $\Theta(n)$, reduces to the term $\lambda x.x$ of length $\Theta(1)$, and can be so reduced in $5n+1$ parallel β-steps, its Frandsen-Sturtivant cost is $\Theta(n)$.

To reduce F to normal form, an optimal evaluator must normalize E to the Church numeral for 2^{2^n}. During this reduction, the fan sharing the two occurrences of s in $\lambda s.\lambda z.s(sz)$ is duplicated exponentially. This duplication follows the pattern of the exponential free-variable bracket duplication in Section 3.2. This work seems to be essential, since a $\Theta(2^n)$-node graph (hence $\Theta(n2^n)$ bits) is being used to represent the numeral for 2^{2^n}. It does not seem likely that sharing graphs can improve measurably on binary representations of integers.

Theorem 5. *If $\nu(E)$ is the Frandsen-Sturtivant cost metric for evaluating a λ-term E, then the number of fan interactions required by existing optimal evaluators grows as $\Omega(2^{\nu(E)})$.*

This result was derived independently by Asperti [Asp96], using a more complicated term having similar consequences.

The reduction of E to normal form also answers a conjecture posed by Frandsen and Sturtivant [FS91]. In their paper, they identify a family of terms requiring at least $5n$ β-steps to normalize, but only n parallel β-steps. They conjectured that the ratio of ordinary to parallel β-steps could be exponential. The reduction of E shows that the ratio can in fact be doubly exponential.

Observe that the Church numeral for 2^{2^n} is computed by iterating squaring n times: $2^{2^n} = (\cdots(2^2)^2\cdots)^2$. To square k, we reduce $\overline{2}\,\overline{k}$, where $\overline{m}$ is the Church numeral for m. This reduction requires $\Omega(k)$ ordinary β-steps. In reducing E, then, the final squaring requires at least $2^{2^{n-1}}$ ordinary β-steps. As a consequence, we derive the following:

Theorem 6. *There exists a family of λ-terms that normalizes in $2^{\Omega(2^n)}$ ordinary β-steps, but only $\Theta(n)$ parallel β-steps and $\Theta(2^n)$ fan interactions.*

Theorem 5 implies that optimal evaluators are inefficient: do we discard the message and look for a better cost metric, or do we kill the messenger and look for a better evaluation algorithm? We think both are necessary. Not only are efficiency improvements needed for optimal reduction at the bureaucratic level, we also need a more realistic cost model. The lesson of this example is very clear: the model of [FS91] does not take into account the size of intermediate terms. Furthermore, we consider the work of the sharing nodes, even if their work is independent from β-reduction, to be essential. By contrast, the work of brackets and croissants seems more bureaucratic.

5.2 Cost models based on fan interaction

In his recent paper, Asperti has proposed a cost metric where the intrinsic complexity of a λ-term is the number of fan interactions incurred in the normalization of the term [Asp96]. "Intuitively," he writes, "Lamping's abstract algorithm does not seem to perform any useless operation." We comment briefly on this proposal.

First, it seems undesirable to base the inherent difficulty of reducing a term on an implementation. Denotational semantics was invented, in part, to liberate *meaning* from implementation; the same ought to be true of *cost*. In this case, there would be nothing left to prove: Lamping's abstract algorithm is certainly optimal in the cost model. Do we as a consequence know anything new?

Second, the proposal seems premature in the face of the research community's considerable lack of insight into optimal evaluation. Imagine that the subject was instead sorting, and the best proposed algorithm was insertion sort. Surely we would not want to declare insertion sort optimal by fiat—we would be better off learning more about sorting.

Last, the proposal ignores the role of brackets and croissants. While these operators *seem* to cause needless bureacracy, they do play a crucial role in demarcating the boundaries of regions, *e.g.*, in GAL, the bodies of λ-terms. If Asperti's proposal is to be taken seriously, it implies that this concept of boundary is not essential.

5.3 Cost models based on labelled λ-calculus

A more realistic, implementation independent cost model is immediately suggested by the (Lévy)-*labelled λ-calculus*, of which there are many flavors: we have relied on the version in [Fie90]. Briefly, each subterm of the initial λ-term is initially annotated with a unique *label*. As reduction occurs, labels are concatenated according to certain rules, so that the labels encode the history of the computation. When $(\lambda x.E)F$ is reduced, the label on each free x in E is (reverse) concatenated with a unique new label associated 1-1 with the label of the function, and the label of the argument; the label of E is concatenated with the new label, and the label of the redex:

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

For example,

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

Lévy's original idea of *optimality* is that all redexes in the entire computation having identical labels on the "function" part are reduced in one parallel β-step.

The first cost model we propose, providing "more money" than that of Frandsen and Sturtivant, though not at initial glance realizable, is that the cost of reducing a term is the number of unique labels generated in the reduction. A second, more liberal cost metric is one where we sum the length of the labels generated. We conjecture that the number of fan interactions is polynomial in the number of labels, and that the total number of interactions, including bureaucratic ones, is polynomial in the sum of the lengths of the labels.

Frandsen and Sturtivant showed Turner combinators and Hughes supercombinators to be inefficient in terms of their proposed cost model, identifying a family of terms involving sharing that require $\Omega(2^n)$ combinator reduction steps, while normalizing in only $\Theta(n)$ β-steps. Optimal evaluators can reduce this family of terms with only $\Theta(n)$ fan interactions because "different" sharing nodes do not duplicate each other. As a consequence, combinators are not efficient even under these more liberal cost models.

6 Summary; open questions

Many important questions concerning the computational efficiency of optimal reduction remain unanswered, including several we have considered in some detail. They include:

1. Can the exponential lower bounds of Section 3 be made stronger? Are there examples that require, say, $\Omega(2^{2^n})$ bookkeeping interactions to effect n fan node interactions? We conjecture no: sharing nodes copy the body and then come to rest at free-variable ports. Thus corresponding to each fan interaction, there is at most a doubling of the size of some graph. The question is, though, *what* graph: reducing a β-redex in the body of a λ-abstraction can allow a sharing node duplicating the λ-abstraction to also duplicate other nodes.

2. Can the cost models we have proposed be realized by implementations? Such a demonstration is nontrivial.

3. Can the bookkeeping interactions that update de-Bruijn indices be related in a more precise, natural way to the cost of substitution?

4. Can similarity of fan nodes be dynamically maintained using altogether different methods, perhaps relying on more sophisticated data structures without all the pitfalls of current methods?

5. Can completeness bounds be proven on the difficulty of optimal evaluation? For instance, given a λ-term E and a constant k where $\log k$ is polynomial in the length of E, can E be normalized in no more than k parallel steps? Were such a question complete for, say, exponential time, this would be a *coup de grâce* to the entire idea of efficient optimal evaluation.

6. Similarly, might efficient implementation of optimal evaluation imply unlikely hierarchy relations, e.g., if k parallel β-steps can be implemented with overhead polynomial in k, then $P = NP$ (or something like that). Such an approach would involve clever and nontrivial coding of generic problems (time bounded computation, satisfiability, etc.).

After having spent tremendous time and effort understanding optimal reduction mechanisms and their efficiencies, we are left with an overriding impression: few algorithmic insights have followed Lamping's breakthrough. Progress has been made in identifying a static (context) semantics, in understanding how Lévy labels identify paths in the graphs, and in alternate mechanisms for hacking the indices of sharing nodes. However, nobody can do it with appreciably less bookkeeping overhead than Lamping, including us. Further progress in developing these algorithms must come from a better global understanding of the dynamics of the bookkeeping nodes, so that redundant nodes can be eliminated.

Acknowledgments. Thanks to Paris Kanellakis, John Field, Doaitse Swierstra, and Georges Gonthier for several very helpful discussions on complexity aspects of the λ-calculus.

References

[Asp94] Andrea Asperti. *Linear logic, comonads, and optimal reductions.* Unpublished manuscript.

[Asp95] Andrea Asperti. *$\delta \circ ! \epsilon = 1$: optimizing optimal λ-calculus implementations.* Proceedings, RTA, 1995.

[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.), pp. 289–312.

[Gir88] Jean-Yves Girard. *Geometry of interaction I: interpretation of System F.* **Logic Colloquim 1988**, pp. 221-260. Elsevier (North Holland), 1989.

[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.

[Hug82] John Hughes. *Supercombinators: a new implementation method for applicative languages.* **1982 ACM Symposium on Lisp and Functional Programming**, pp. 1–10.

[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.

[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.

[HM94] Harry G. Mairson and Fritz Henglein. *The complexity of type inference for higher-order typed lambda calculi.* **Journal of Functional Programming** 4:4 (October 1994), pp. 435-478.

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

[PJ87] Simon Peyton-Jones. **The Implementation of Functional Programming Languages.** Prentice-Hall, 1987.

[Tur79] David A. Turner. *New implementation techniques for applicative languages.* **Software Practice and Experience** 9 (1979), pp. 31–49.