On the complexity of beta-reduction

Andrea Asperti

Technical Report UBLCS-95-12

July 1995

Department of Computer Science
University of Bologna
Piazza di Porta S. Donato, 5
40127 Bologna (Italy)
UBLCS Technical Report Series

<table>
<thead>
<tr>
<th>Number</th>
<th>Title</th>
<th>Authors</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>94-2</td>
<td>Symbol-Level Requirements for Agent-Level Programming</td>
<td>M. Gaspari, E. Motta</td>
<td>February 1994</td>
</tr>
<tr>
<td>94-3</td>
<td>Extending Prolog with Data Driven Rules</td>
<td>M. Gaspari</td>
<td>February 1994</td>
</tr>
<tr>
<td>94-5</td>
<td>A Shared Dataspace Language and its Compilation</td>
<td>M. Gaspari</td>
<td>March 1994</td>
</tr>
<tr>
<td>94-6</td>
<td>Exploring the Coordination Space with LO</td>
<td>S. Castellani, P. Ciancarini</td>
<td>April 1994</td>
</tr>
<tr>
<td>94-7</td>
<td>Comparative Semantics of LO</td>
<td>S. Castellani, P. Ciancarini</td>
<td>April 1994</td>
</tr>
<tr>
<td>94-8</td>
<td>Distributed Conflicts in Communicating Systems</td>
<td>N. Busi, R. Gorrieri</td>
<td>April 1994</td>
</tr>
<tr>
<td>94-9</td>
<td>Pseudopolar Array Mask Algorithm for Spherical Coordinate Grid Surfaces</td>
<td>A. Amoroso, G. Casciola</td>
<td>May 1994</td>
</tr>
<tr>
<td>94-11</td>
<td>Describing Queueing Systems with MPA</td>
<td>M. Bernardo, L. Donatiello, R. Gorrieri</td>
<td>May 1994</td>
</tr>
<tr>
<td>94-12</td>
<td>Operational GSPN Semantics of MPA</td>
<td>M. Bernardo, L. Donatiello, R. Gorrieri</td>
<td>May 1994</td>
</tr>
<tr>
<td>94-13</td>
<td>Experiments in Distributing and Coordinating Knowledge</td>
<td>P. Ciancarini</td>
<td>May 1994</td>
</tr>
<tr>
<td>94-14</td>
<td>A Comparison of Parallel Search Algorithms Based on Tree Splitting</td>
<td>P. Ciancarini</td>
<td>May 1994</td>
</tr>
<tr>
<td>94-16</td>
<td>Replicated File Management in Large-Scale Distributed Systems</td>
<td>O. Babaoğlu, A. Bartoli, G. Dini</td>
<td>June 1994</td>
</tr>
<tr>
<td>94-17</td>
<td>Parallel Symbolic Computing with the Shared Dataspace Coordination Model</td>
<td>P. Ciancarini, M. Gaspari</td>
<td>July 1994</td>
</tr>
<tr>
<td>94-19</td>
<td>On Group Communication in Large-Scale Distributed Systems</td>
<td>O. Babaoğlu, A. Schiper</td>
<td>July 1994</td>
</tr>
<tr>
<td>94-20</td>
<td>Dynamic Allocation of Signature Files in Multiple-Disk Systems</td>
<td>P. Ciaccia</td>
<td>August 1994</td>
</tr>
<tr>
<td>94-21</td>
<td>Parallel Independent Grid Files Based on a Dynamic Declustering Method Using Multiple Error Correcting Codes</td>
<td>P. Ciaccia</td>
<td>November 1994</td>
</tr>
<tr>
<td>95-1</td>
<td>Performance Preorder and Competitive Equivalence</td>
<td>F. Corradini, R. Gorrieri, M. Roccelli</td>
<td>January 1995</td>
</tr>
<tr>
<td>95-2</td>
<td>Clepsydra Methodology</td>
<td>P. Ciaccia, O. Ciancarini, W. Penzo</td>
<td>January 1995</td>
</tr>
<tr>
<td>95-4</td>
<td>Effective Applicative Structures</td>
<td>A. Asperti, A. Ciabattoni</td>
<td>January 1995</td>
</tr>
<tr>
<td>95-5</td>
<td>An Open Framework for Cooperative Problem Solving</td>
<td>M. Gaspari, E. Motta, A. Stutt</td>
<td>February 1995</td>
</tr>
<tr>
<td>95-7</td>
<td>Modelling Interaction in Agent Systems</td>
<td>A. Dalmonte, M. Gaspari</td>
<td>February 1995</td>
</tr>
<tr>
<td>95-10</td>
<td>Reliable Synchronization Support and Group-Membership Services for Distributed Multimedia Applications</td>
<td>F. Panzieri, M. Roccelli</td>
<td>March 1995</td>
</tr>
<tr>
<td>95-11</td>
<td>The Inherent Cost of Strong-Partial View-Synchronous Communication</td>
<td>O. Babaoğlu, R. Davoli, L.-A. Giachini, P. Sabattini</td>
<td>April 1995</td>
</tr>
</tbody>
</table>
On the complexity of beta-reduction

Andrea Aserti

Technical Report UBLCS-95-12

July 1995

Abstract

We prove that the complexity of Lamping’s optimal graph reduction technique for the λ-calculus can be exponential in the number of Lévy’s family reductions. Starting from this consideration, we propose a new measure for what could be considered as “the intrinsic complexity” of λ-terms.

1. Partially supported by the ESPRIT Basic Research Project 6454 - CONFER.
2. Department of Mathematics, University of Bologna, Piazza di Porta S.Donato 5, Bologna, Italy.
1 Introduction

Twenty years ago, Lévy [Le78] introduced the notion of redex family to formalize the intuitive idea of optimal sharing in the $\lambda$-calculus (see also [Le80, AL93]). As a main consequence, the length of the family reduction would provide a lower bound to the intrinsic complexity of $\lambda$-term reduction, in any possible implementation.

In 1990, Lamping [Lam90] discovered a complex graph reduction technique that was optimal in Lévy’s sense (that is, all sharable redexes had a unique graphical representation, and could be reduced in a single atomical step). However, Lamping did not establish any complexity relation between his algorithm and the length of the corresponding family reduction.

In this paper, we prove that Lamping’s technique can be exponential with respect to the number of redex families reduced along the computation. This fact does not contradict neither the optimality of the algorithm, nor its relevance in view of an actual implementation (as a matter of fact, the examples where Lamping’s algorithm is exponential, are also the examples where it works better with respect to more traditional implementation techniques).

On the contrary, we claim that the length of the family reduction is not a reasonable lower bound to the “intrinsic complexity” of $\lambda$-terms, and we shall propose a different complexity measure.

2 Lamping’s graph reduction technique

Lamping’s graph rewriting rules can be naturally classified in two main groups:

1. the rules involving application, abstraction and sharing nodes (fan), that are responsible for $\beta$-reduction and duplication (we shall call this group of rules the abstract algorithm);
2. some rules involving control nodes (square brackets and croissants), which are merely required for the correct application of the first set of rules.

More precisely, the first set of rules requires an “oracle” to discriminate the correct rule in a critical pair between fan-nodes; the second set of rules can be seen as an effective implementation of this oracle.

This distinction looks particularly appealing since all different translations proposed in the literature after Lamping [GAL92a, GAL92b, As94, As95] differ from each other just in the way the oracle is implemented (in the sense that all of them perform exactly the same set of abstract reductions).

In this paper, we shall prove that Lamping’s technique can be already exponential in its abstract algorithm, without considering the extra work required by the oracle.

For this reason, we shall introduce here Lamping’s technique without mentioning the possible solution to the effective implementation of the oracle.

2.1 Initial translation

Initially, in the optimal graph reduction technique, a $\lambda$-term is essentially represented by its abstract syntax tree (like in ordinary graph reduction). There are two main differences, however:

1. we shall introduce an explicit node for sharing;
2. we shall suppose that variables are explicitly connected to their respective binders.

For instance, the graph in Figure 1(1) is the initial representation of the $\lambda$-term $M = (\text{two } \delta)$, where $\text{two} = \lambda x. \lambda y. (x(y))$ and $\delta = \lambda z. (z z)$.

The triangle (we shall call it fan) is used to express the sharing between different occurrences of a same variable. All variables are connected to their respective binders (we shall always represent this connection on the left of the connection to the body). Since multiple occurrences of a same variable are shared by fans, we shall have a single edge leaving a $\lambda$ towards its variables. So, each node in the graph (@, $\lambda$ and fan) has exactly three distinguished sites (ports) where it will be connected with other ports.
Figure 1. Graph reduction of $\lambda x. y x y$
2.2 Reduction

We shall now illustrate the main ideas of Lamping’s optimal graph reduction technique by showing how a simplified version of the algorithm would work on our sample term \((\mathit{two} \, \delta)\). As we shall see, a crucial issue will remain unresolved. This is exactly where the oracle comes in: however, since the complexity of the oracle is not necessary to prove the exponential nature of Lamping’s algorithm, we shall not discuss this complex topic here.

Lamping’s algorithm consists of a set of local graph rewriting rules. At a given stage of the computation, we can usually have several reducible configurations in the graph. In this case, the choice of next rule to apply is made non-deterministically. This does not matter that much, since, as we shall see, the graph rewriting system satisfies a one-step diamond property (that implies, not only confluence, but also that all reduction sequence to the normal form, if it exists, have the same length). In particular, we shall usually choose the next rule in our example of reduction according to a dydactical criterion (and sometimes for graphical convenience).

The most important of the graph rewriting rules is obviously \(\beta\)-reduction: \((\lambda x . M \, N) \rightarrow M [N/x]\).

In graph reduction, substituting a variable \(x\) for a term \(N\) amounts to explicitly connect the variable to the term \(N\). Moreover, the value returned by the application before the redex is fired (the link above the application) becomes the (instantiated) body \(M\) of the function. Since the portions of graph representing \(M\) and \(N\) do not play any role in the graph reduction corresponding to the \(\beta\)-rule, this reduction can be expressed by the completely local graph rewriting rule in Figure 2.

![Figure 2. the \(\beta\)-rule](image)

By firing the outermost \(\beta\)-redex in \((\mathit{two} \, \delta)\), we get the graph in Figure 1(2). Since the next redex involves a shared \(\lambda\)-expression, we must eventually proceede to the duplication of \(\delta\). In ordinary graph reduction, this duplication would be performed as a unique, global step on the shared piece of graph. On the contrary, the optimal graph reduction technique proceeds in a more lazy way, duplicating the external \(\lambda\) but still sharing its body. However, since the binder has been duplicated, we are forced to introduce another fan on the edge leading from the binder to the variable. In a sense, this fan works as an “unsharing” operator (fan-out, usually depicted upside-down), that is to be “paired” against the fan(-in) sharing the body of the function\(^3\). Since the body of the function \(\lambda x . M\) does not play any role in this reduction, it can be formally expressed as a local interaction between a fan and a \(\lambda\), as described in Figure 3. Let us now proceede in the analysis

![Figure 3. Fan-\(\lambda\) interaction](image)

of our example. By applying the fan-\(\lambda\) interaction rule, we get the graph in Figure 1(3). Now, two

---

3. Although there is no operational distinction between a fan-in and a fan-out, their intuitive semantics is quite different; in particular, keep in mind that a fan-out is always supposed to be paired with some fan-in in the graph, delimiting its scope and annihilating its sharing effect. The way the correct pairing between fans is determined is a crucial point of the optimal graph reduction technique, solved by the “oracle”.

UBLCS-95-12 4
\(\beta\)-redexes have been created, and by their firing we are lead to the graph in Figure 1(5). We have no more \(\beta\)-redexes in the graph, and no fan-\(\lambda\) interactions, so we must proceed in the duplication process, but we must be very carefully here. In particular, the following graph rewriting rule is strictly forbidden, in the optimal imlementation technique (although semantically correct).

\[
\begin{array}{c}
\begin{array}{c}
\text{(1)}
\end{array}
\end{array}
\]

**Figure 4.** not optimal duplication of the application

The intuition should be clear: since the shared application could be involved in some redex, its duplication would imply a double execution of the redex.

The only other possible interaction is between the two fans inside the dotted region. This is another crucial point of the optimal graph reduction technique. As we shall see, this interaction must be handled in a different way form the similar interactions in Figure 1(7). Note in particular that the two fans in Figure 1(5) are not “paired”: the fan-in is a residual of the shared variable of \(\delta\), while the fan-out is a residual of the shared variable of \(\text{two}\), in the process of duplicating \(\delta\).

Since the two fans have nothing to do with each other, they must duplicate each other, according to the rule in Figure 5(2). Now (see Figure 1.(6)), we have a fan-out in front of the function-port of

\[
\begin{array}{c}
\begin{array}{c}
\text{(2)}
\end{array}
\end{array}
\]

**Figure 5.** fan-annihilation rule

the application. In this case, we can apply the following rule: Intuitively, this rule is correct from

\[
\begin{array}{c}
\begin{array}{c}
\text{(3)}
\end{array}
\end{array}
\]

**Figure 6.** fan-@ intersection

the point of view of optimal sharing since such a configuration already implies the existence of two unsharable (class of) redexes for the application.

After firing this rule, we get the graph in Figure 1.(7). In this case, both pairs of fans are paired: they both belong to the same “duplication process”, that has been now (locally) completed. So, the obvious rule, in this case, is to annihilate the paired fans, according to Figure 5.(1).

The problem of deciding which rule to apply when two fans meet each other (that is the question of how their pairing is established) is the crucial point of the optimal implementation technique (solved by the oracle).

By a double application of this rule, we get the graph in Figure 1.(9), that is in normal form w.r.t. Lamping’s algorithm.
3 Complexity

Before discussing the complexity of Lamping’s “abstract” algorithm, we should start by fixing a few preliminary requirements. First of all, a typical feature of optimal techniques is that of anticipating work that could become useful only later on in the computation. A reasonable way to take into account this “extra” work is that of restricting the analysis to λ-terms whose normal form is an atomic constant (or, if you prefer, a variable). This hypothesis also allows us to avoid some obviously degenerate examples. Consider for instance the term \( P = \lambda x. \lambda y. (y x x) \). If we apply \( P \) to a closed term \( M \) in normal form, \( M \) gets fully duplicated, and the “cost” of the \( \beta \)-redex would seem proportional to the size of \( M \). However, this reasoning does not seem convincing, since in duplicating \( M \) we also duplicated all its λ and application nodes (its prerequisite chains, in Lamping’s terminology), which (whenever turned to redexes), would eventually belong to distinguished families! So, we just anticipated work that had to be done in any case\(^4\).

Our second assumption will be to consider only terms of the \( \lambda I \) calculus. The reason, here, is that the correct handling of garbage collection in optimal reduction techniques is still a subject of investigation (in particular, Lamping’s approach does not seem to be completely satisfactory).

We shall now provide an example of \( \lambda \)-term satisfying our assumptions, whose “abstract” reduction (i.e. without considering the extra cost of the oracle) is already exponential in the number of family reductions. Since the example is quite complex, we shall proceed by considering a few auxiliary terms.

Let us start with a simple case. In Figure 7 is a possible representation of the Church Integer \( \text{two} \) in sharing graphs, obtained by reducing the \( \lambda \)-term \( \text{two} = \lambda x. \lambda y. (\lambda z. (x (z y))) \lambda w. (x w) \).

Let us now consider the application of \( \text{two} \) to itself. Recall that the application \( (n m) \) of two church integers \( n \) and \( m \) gives the church integer \( m^n \), so the expected result is (a representation of) the church integer \( \text{four} \). The reduction is shown in Figure 8. The two first reduction steps are \( \beta \)-redex. After these reductions we are left with the term in Figure 8.(3), where the subterm \( \lambda y. (x (z y)) \) is shared by means of the two copies of the fan marked ’a’. Now, this subterm is fully duplicated. This process requires: 2 steps for duplicating the \( \lambda \) and the application; 2 steps for duplicating the fans; 3 steps for effacing all residuals of fans marked with ’a’.

After these 7 steps we are left with the graph in Figure 8.(4), where a new \( \beta \)-redex has been put in evidence. Firing this redex, we obtain the final configuration in Figure 8.(5). Summing up, we executed 3 \( \beta \)-reductions (actually, family reductions), and 7 fan-interactions. Note moreover that the final configuration has the same shape of the initial one.

Let us now generalize the previous example. As should be clear, the church integer \( 2^n \) can be represented by the graph in Figure 9, where we have exactly a sequence of fan-in of length \( n \) and a corresponding sequence of fan-out of the same length.

Let us now apply this term to itself. By firing the two outermost \( \beta \)-redexes we get the term in Figure 10. In the case of two, the portion of graph inside the two fans marked with ’a’ is now fully duplicated. This duplication requires: 2 steps for duplicating the \( \lambda \) and the application; 2n steps for duplicating fans; n+2 steps for effacing fans. This gives a total of \( 4 + 3n \) operations.

After these reductions, we get the configuration in Figure 11. A new \( \beta \)-redex has been created. By firing this redex we obtain the graph in Figure 12. Now, this graph has the same shape of the graph in Figure 10, and we can iterate our reasoning. In particular, the duplication of the innermost part of the graph will now require \( 4 + 3(2n) \) operations. Then, we shall have a new \( \beta \)-redex, and by its firing, we shall get a graph of the same shape of Figure 12 but where the innermost sequences of fans have length \( 4n \) (this length is duplicated at every iteration of the process), while the length of the outermost sequences is decremented by one.

Summing up, the total number of fan-interactions is given by

\[
((4 + 3n) + (4 + 3 \ast (2n)) + (4 + 3 \ast (4n)) + \ldots + (4 + 3 \ast (2^{n-1}n)))
\]

\(^4\) The idea of considering only those rules which are needed to put in evidence new redexes gives some problems due to the unpredictable disposition of fans inside a “virtual” redex: again, if all copies of prerequisite chains generated by the duplication are not fired along the computation, we could perform some (apparently) useless work.
Figure 7. two': a representation of Church's integer two

Figure 8. the reduction of (two' two')
Figure 9. a representation of $2^n$

Figure 10. the reduction of $(2^n, 2^n)$
Figure 11. the reduction of $2^n \ 2^n$
Figure 12. The reduction of \((2^n 2^n)\)

\[
= 4n + 3n \sum_{i=0}^{n-1} 2^i = 4n + 3n (2^n - 1) = n (3 \cdot 2^n + 1)
\]

In contrast, we have executed just \(n\) \(\beta\)-reductions in the main loop, plus two at the very beginning, for a total of \(n + 2\) family reductions.

Our final problem consists in providing an example based on the terms above which satisfies our auxiliary assumptions mentioned at the beginning of this section (i.e. it should be a term of the \(\lambda\text{-I}\)-calculus, whose normal form is an atomic constant).

The term we shall consider is: \(g = \lambda n. (n \delta \text{twod} \ I \ c)\), where I is the identity, \(c\) is some constant, \(\delta = \lambda x. (xx)\), and \(\text{twod} = \lambda x. \lambda y. (\lambda z. (z(x y)) \ \lambda w. (x w))\). I is the identity and \(c\) is some constant.

If \(n\) is a church integer, \((g \ n)\) obviously reduces to \(c\). The term \((n \delta \text{twod})\) is a real “monster”, from the complexity point of view. As a function of \(n\), it corresponds to the church integer \(a_n\), in the succession

\[
a_0 = 2; a_{i+1} = a_i \cdot 2^{a_i}\.
\]

For instance, \(a_3 = 256^{256}\). Let us now consider the number of family reductions. When we apply \(g\) to the Church integer 0, we perform 9 family reductions (one for the application of \(g\), two for the application of 0, three internal to \(\text{twod}'\), and three for the extra-identities). These operations are constant for each input \(n\) of the function \(g\). Let us now compute the cost for each application of \((\delta \ a_i)\). This is 1 plus the number of family reductions for \((a_i \ a_i)\) computed in the previous section, namely \(2 + \log(a_i)\). Note that the succession \(b_i = \log(a_i)\) can be equally defined as \(b_0 = 1; b_{i+1} = b_i \cdot 2^{b_i}\).

Summing up, the number of family reductions \(f(n)\) for the term \((g \ n)\) is

\[
f(n) = 9 + 3 \cdot n + \sum_{i=0}^{n-1} b_i
\]

Finally, let us consider the number of fan-interactions. For each application of \((\delta \ a_i)\), we have \(1 + 4 + 3 \cdot b_i\) interactions for duplicating \(a_i\), plus \(b_i + 3 \cdot b_{i+1}\) operations in the reduction of \((a_i \ a_i)\) (recall that \(b_{i+1} = b_i \cdot 2^{b_i}\)). Moreover, we have one single operation internal to \(\text{twod}'\), \(5 \cdot (n - 1)\) operations for creating all copies of \(\delta\), and \(b_n\) final operations of fan-effacement when we apply the extra-identities.
Summing up, the number \( c(n) \) of fan-interactions (for \( n > 0 \)) is given by the formula

\[
c(n) = 10*n - 4 + 4 \sum_{i=0}^{n-1} b_i + 3 \sum_{i=1}^{n} b_i + b_n
\]

\[
= 10*n + 7 \sum_{i=1}^{n-1} b_i + 4 * b_n
\]

It is easy to show that, for any \( n \), \( c(n) \leq 2^{f(n)} \). Note first that \( 2^{\sum_{i=0}^{n-1} b_i} = b_n \). Then

\[
2^{f(n)} = 2^{9 + 3 * n + \sum_{i=0}^{n-1} b_i} > 2^9 * b_n
\]

On the other side, \( \sum_{i=0}^{n-1} b_i \leq b_n \), and obviously \( n \leq b_n \), so

\[
c(n) < 21 * b_n < 2^9 * b_n < 2^{f(n)}
\]

The previous formulas have been experimentally confirmed by our prototype implementation of (a variant of) Lamping’s algorithm: the Bologna Optimal Higher-Order Machine (BOHM)\(^5\). BOHM has been especially designed to support the user with a large number of experimental data relative to each computation. The results of the computation of the function \( g \) are shown in figure 13. The four columns in the table are, respectively, the user time required by the computation (on a Sparc-station 5), the total number of interactions (comprising the “oracle”), the length of the family reduction (app-lambda interactions), and the total number of fan-interactions.

<table>
<thead>
<tr>
<th>Input</th>
<th>user</th>
<th>tot. inter.</th>
<th>families</th>
<th>fan-inter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(g zero)</td>
<td>0.00 s.</td>
<td>38</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>(g one)</td>
<td>0.00 s.</td>
<td>64</td>
<td>13</td>
<td>18</td>
</tr>
<tr>
<td>(g two)</td>
<td>0.00 s.</td>
<td>200</td>
<td>18</td>
<td>66</td>
</tr>
<tr>
<td>(g three)</td>
<td>15.90 s.</td>
<td>2642966</td>
<td>29</td>
<td>8292</td>
</tr>
</tbody>
</table>

Figure 13. The function \( g \)

It is also possible to find examples of exponential explosion with respect to a linear growth of the number of family reductions. An interesting case is provided by the \( \lambda \)-term \( h = \lambda n. (n \ two' two' I c) \). For this function, the number of family reductions \( f(n) \) grows linearly in its input \( n \) (in particular, \( f(n) = 12 + 3 * n \). The number of fan-interactions \( c(n) \) is given by the formula

\[
c(n) = 12 * n - 2 + 4 * 2^n
\]

Note that, again, \( c(n) \leq 2^{f(n)} \). In Figure 14, you will find the experimental results in BOHM. In this case, we also make a comparison with two standard (respectively, strict and lazy) implementations such as Caml-Light and Yale Haskell. The results of the test should give a gist of the power of the optimal graph reduction technique.

4 Discussion

We conjecture that, under our assumption, the complexity of Lamping’s abstract algorithm is at most exponential with respect to the number of family reductions required for normalizing the term. Unfortunately we still do not have a proof of this fact. However, in our opinion, the real issue is of a different nature. In particular, the number of family

\(^5\) BOHM is available by anonymous ftp at ftp.cs.unibo.it, in the directory /pub/aspetri. Get the file BOHM.tar.Z (compressed tar format).

UBLCS-95-12 11
Figure 14. The function h

reductions does not seem to provide a reasonable lower bound to the “intrinsic” complexity of a \( \lambda \)-term. Intuitively, Lamping’s abstract algorithm does not seem to perform any useless operation. Our claim is that the total number of these rules, instead of the number of family reductions, would provide a more reasonable and interesting measure of the “intrinsic” complexity of a \( \lambda \)-term. More precisely, we propose to count the total number of annihilation rules between fans (plus the number of family reductions). Note that, under our assumptions, the two complexity measures above turn out to be equivalent (if the term reduces to a constant, all fan, application and \( \lambda \) nodes have to be annihilated, soon or later).

There are several motivations to support our claim. First of all, as it was remarked in [GAL92a], \( \lambda \) and application nodes can be assimilated to fans, and the \( \beta \)-reduction rule can be seen as an annihilation rule between a pair of fans. From this respect, there is no clear reason for giving a special status to \( \beta \)-redexes.

The second point is subtler. Using context semantics, it is possible to prove that the annihilation rules between fans are in bijective correspondence with the number of discriminants for different \( \odot \)-cycles in the \( \lambda \)-term (see [AL93, ADLR94]). Roughly, a \( \odot \)-cycle is a particular kind of looping path inside the argument of an application. Now, every time we have a discriminant for such a cycle (i.e. the cycle is \textit{shared}), we also have an extra and unavoidable operation that amounts to choose the proper discriminant when coming back from the loop. Following [DR95], this extra operation (that essentially amount to save a suitable return information in presence of a possible looping situation), can be easily recognised in other typical implementation techniques, such as environment machines.

Our complexity measure has been confirmed so far by all the tests we made on many available implementations of functional languages.

5 Conclusions

There are a lot of interesting open problems related to optimal reductions. First of all, it looks important to provide a definite upper bound to the complexity of Lamping’s “abstract” algorithm in terms of family reductions. Secondly, we should understand the complexity of what we called the “oracle”. The complexity of this part of the algorithm is actually very different in all the reduction techniques proposed so far (see [As95] for a discussion), and it is still a subject of research. As you can see by the few examples in BOHM (that is now quite sophisticated from this respect), the complexity of the oracle is the crucial issue of Lamping’s technique. Although, in BOHM, it is clearly not linear w.r.t. the number of applications of abstract rules, we have found no evidence so far of an exponential explosion of its complexity.
References


