On the complexity of beta-reduction

Andrea Asperti
Dipartimento di Matematica
P.zza di Porta S.Donato 5, Bologna, Italy
asperti@cs.unibo.it

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 Introduction

Twenty years ago, Lévy [Le78] introduced the notion of *redex family* to formalize the intuitive idea of optimal sharing in the λ-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 λ-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 lenght 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 lenght of the family reduction is not a reasonable lower bound to the "intrinsic complexity" of λ-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 β-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 interaction rule between a pair of 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 λ-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 λ-term $M = (two\ \delta)$, where $two = \lambda x.\lambda y.(x(xy))$ and $\delta = \lambda x.(x\,x)$.
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 λ towards its variables. So, each node in the graph (@, λ and fan) has exactly three distinguished sites (ports) where it will be connected with other ports.

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 $(two\,\delta)$. 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 the graph rewriting system is an Interaction Net in Lafont's sense [Laf90], and it satisfies a one-step diamond property (that implies not only confluence but also that, if a term has a normal form, all normalizing derivations 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 β-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 β-rule, this reduction can be expressed by the completely local graph rewriting rule in Figure 2.
By firing the outermost β-redex in $(two\,\delta)$, we get the graph in Figure 1.(2). Since the next redex involves a shared λ-expression, we must eventually proceede to the duplication of δ. In ordinary graph reduction, this du-

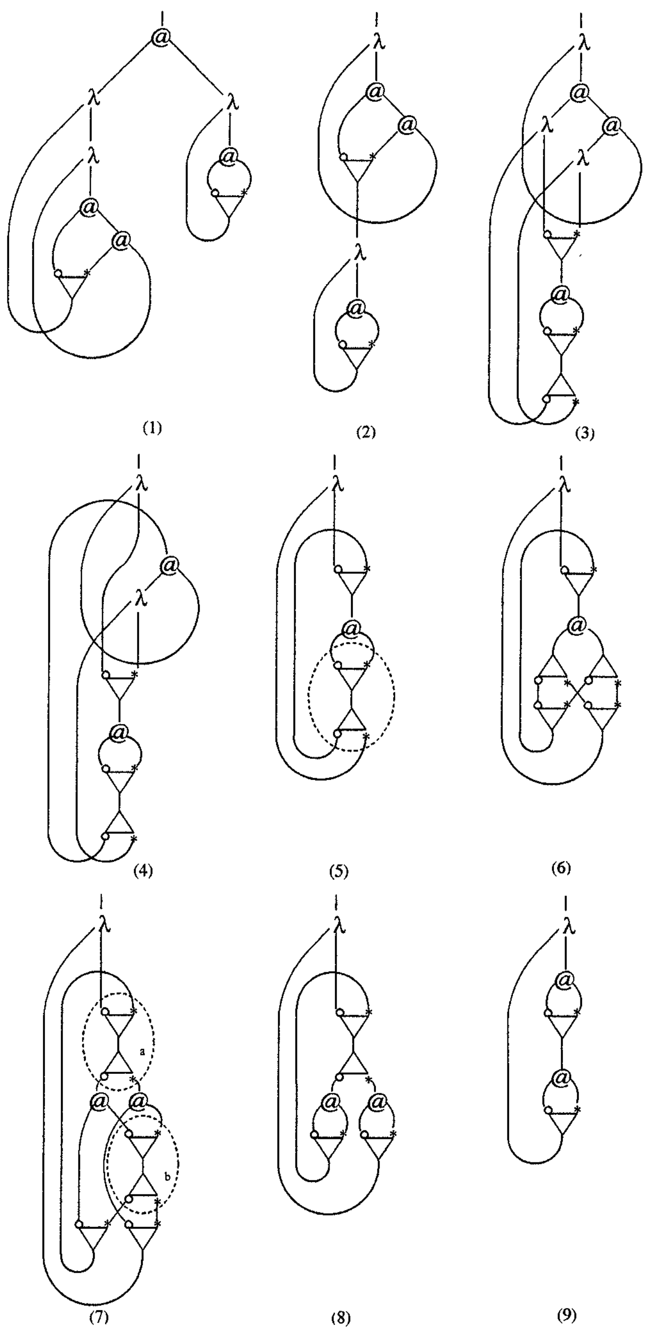

Figure 1: Graph reduction of $(two\ \delta)$

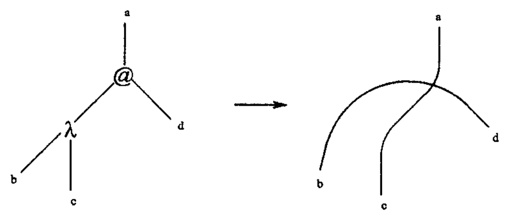

Figure 2: the β-rule

plication would be performed as a unique, global step on the shared piece of graph. On the contrary, the optimal graph reduction technique proceedes in a more lazy way, duplicating the external λ 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[1]. 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 λ, as described in Figure 3. Let us now proceede in the analysis of our example.

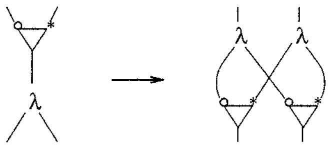

Figure 3: Fan-λ interaction

By applying the fan-λ interaction rule, we get the graph in Figure 1.(3). Now, two β-redexes have been created, and by their firing we are lead to the graph in Figure 1.(5). We have no more β-redexes in the graph, and no fan-λ interactions, so we must proceede in the duplication process, but we must be very carefully here. In particular, the following graph rewriting rule is strictly

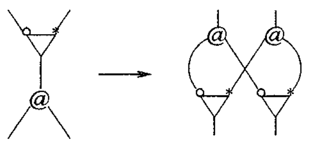

Figure 4: not optimal duplication of the application

forbidden, in the optimal imlementation technique (although semantically correct). 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 δ, while the fan-out is a residual of the shared variable of *two*, in the process of duplicating δ. 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

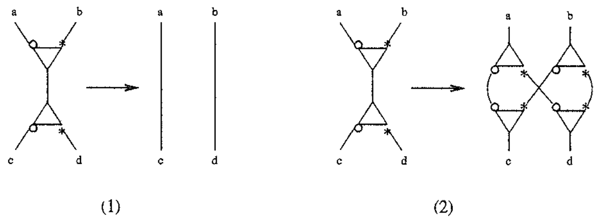

Figure 5: fan-annihilation rule

in front of the function-port of the application. In this case, we can apply the following rule: Intuitively, this

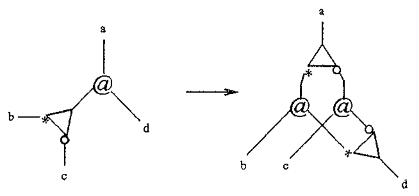

Figure 6: fan-@ interaction

rule is correct from 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 assumptions. First of all, a typical feature

[1]Although there is no operational distinction between a fan-in and a fan-out, their intutive 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". Obviously, in order to give a precise definition of the abstract algorithm we should provide the formal definition of the oracle, that is very complex and would just obscure the intuitive nature of the abstract rules. In particular, the obvious and naive idea of labeling fans does not work (see [Lam89]).

of optimal techniques is that of anticipating work that could become usefull only later on in the computation. A reasonable way to take into account this "extra" work is that of restricing 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 β-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[2].

Our second assumption will be to consider only terms of the λ-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 a λ-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 proceede by considering a few auxiliary terms.

Let us start with a simple case. In Figure 7 is a possible representation of the Church Integer *two* in sharing graphs, obtained by reducing the λ-term $two' = \lambda x.\lambda y.(\lambda z.(z(z\ y))\ \lambda w.(x\ w))$. Let us now consider the application of 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 representaion of) the church integer *four*. The reduction is shown in Figure 8. The two first reduction steps are β-redex. After these reductions we are left with the term in Figure 8.(3), where the subterm $\lambda y.(x(x\ y))$ 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 λ 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 β-redex has been put in evidence. Firing this redex, we obtain the final configuration in Figure 8.(5). Summing up, we executed 3

[2]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.

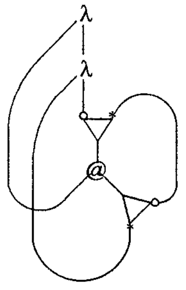

Figure 7: two': a representation of Church's integer two

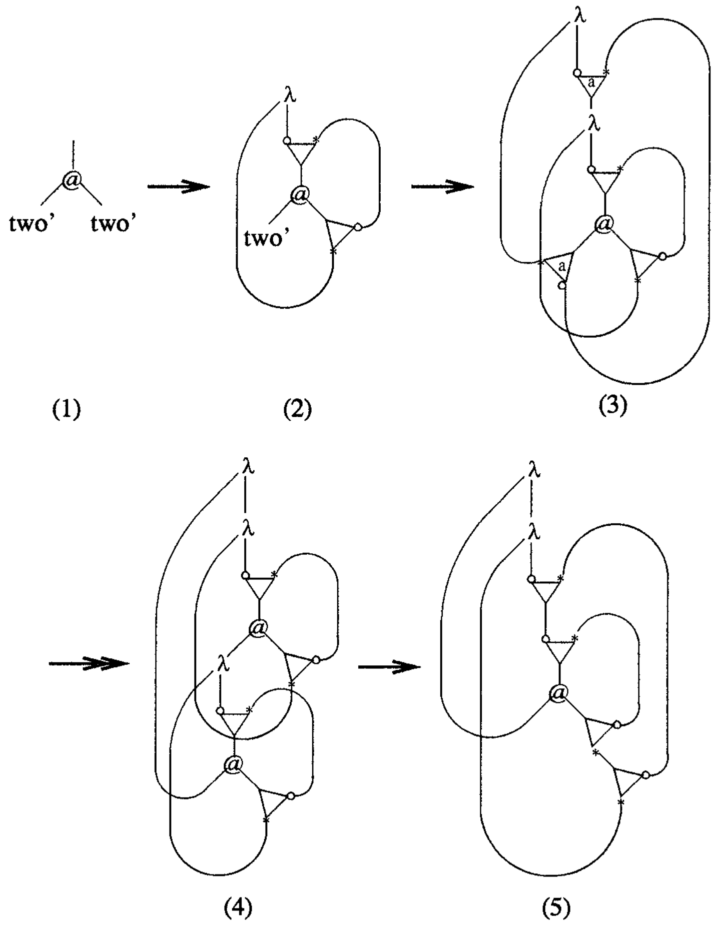

Figure 8: the reduction of (two' two')

β-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

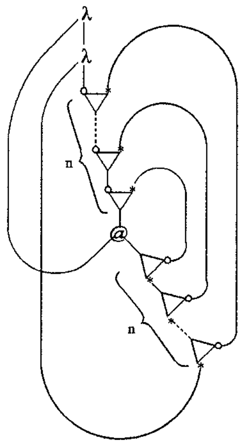

Figure 9: a representation of 2^n

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 β-redexes we get the term in Figure 10. As

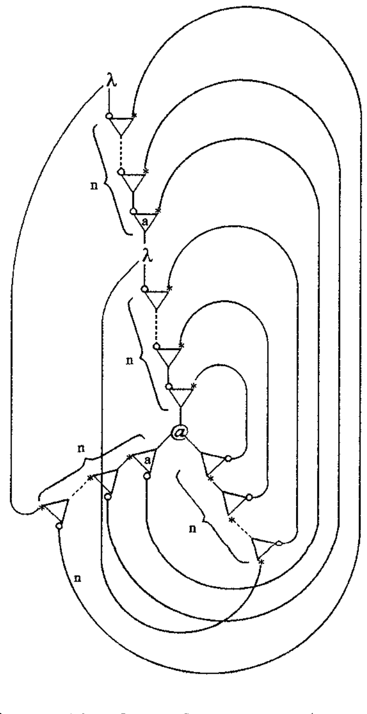

Figure 10: the reduction of (2^n 2^n)

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 λ 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 β-redex has been created. By firing this

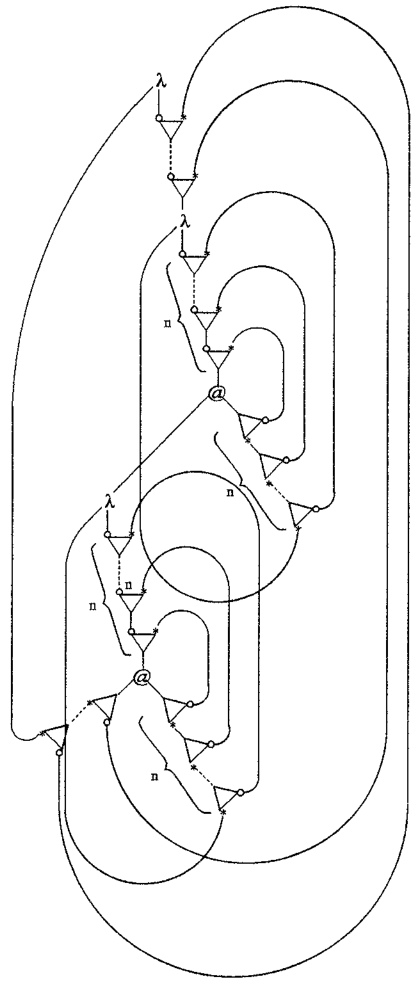

Figure 11: the reduction of (2^n 2^n)

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 β-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 doubled 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

$$((4+3n)+(4+3*(2n))+(4+3*(4n))+\ldots+(4+3*(2^{n-1}n)))$$

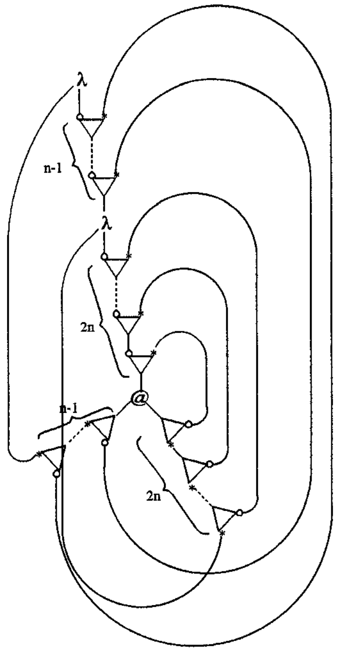

Figure 12: the reduction of $(2^n\ 2^n)$

$$= 4n + 3n * \sum_{i=0}^{n-1} 2^i = 4n + 3n * (2^n - 1) = n * (3 * 2^n + 1)$$

In contrast, we have executed just n β-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 λ-I-calculus, whose normal form is an atomic constant).

The term we consider is: $g = \lambda n.(n\ \delta\ two'\ I\ q)$, where $\delta = \lambda x.(xx)$, $two' = \lambda x.\lambda y.(\lambda z.(z(z\ y))\ \lambda w.(x\ w))$, I is the identity and q is some constant. If n is a church integer, $(g\ n)$ obviously reduces to q. The term $(n\ \delta\ two')$ 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 $a_0 = 2; a_{i+1} = a_i^{a_i}$. 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 two', 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 $b_0 = 1; b_{i+1} = b_i * 2^{b_i}$.

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

$$f(n) = 9 + 3 * n + \sum_{i=0}^{n-1} b_i$$

Finally, let us consider the number of fan-interactions. For each application of $(\delta\ a_i)$, we have $1 + 4 + 3 * b_i$ interactions for duplicating a_i, plus $b_i + 3 * b_{i+1}$ operations in the reduction of $(a_i\ a_i)$ (recall that $b_{i+1} = b_i * 2^{b_i}$). Moreover, we have one single operation internal to two', $5 * (n-1)$ operations for creating all copies of δ, and b_n final operations of fan-effacement when we apply the external parameters.

Summing up, the number $c(n)$ of fan-interactions (for $n > 0$) is given by the formula

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

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

Note now that $2^{\sum_{i=0}^{n-1} b_i} = b_n$. So,

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

For $n \geq 3$, it is easy to show that $2^{9+3n} < 2^8 * b_n$. Hence:

$$2^{f(n)} < 2^8 * b_n^2 < 2^4 * c(n)^2.$$

and finally (for any $n \geq 3$)

$$c(n) > \frac{1}{4} * 2^{\frac{f(n)}{2}}$$

We can also easily prove that, for any n, $c(n) \leq 2^{f(n)}$. One one side we have $2^{f(n)} > 2^9 * b_n$. On the other side, $\sum_{i=0}^{n-1} b_i \leq b_n$ and obviously $n \leq b_n$, so

$$c(n) < 21 * b_n < 2^9 * b_n < 2^{f(n)}$$

3.1 The Bologna Optimal Higher-order Machine

The previous formulas have been also experimentally confirmed by our prototype implementation of (a variant of) Lamping's algorithm: the Bologna Optimal Higher-Order Machine (BOHM)[3]. BOHM [AGN95] is a simple interpreter written in C. Its source language is a sugared λ-calculus enriched with booleans, integers, lists and basic operations on these data types. The extension of Lamping's technique to this language is essentially based on Asperti and Laneve's work on Interaction

[3]BOHM is available by anonymous ftp at ftp.cs.unibo.it, in the directory /pub/asperti. Get the file BOHM1.0.tar.Z (compressed tar format).

Systems [AL93b]. In particular, all syntactical operators are represented as nodes in a graph. These nodes are divided into *constructors* and *destructors*, and reduction is expressed as a local interaction (graph rewriting) between constructor-destructor pairs.
BOHM is *lazy* (in the sense that it always reduces the *family* of the leftmost outermost redex) and *weak* (that is, it stops computing as soon as the top node in the graph is a constructor). As a consequence, BOHM supports lazy data structures, such as streams.
Due to its prototyping nature, BOHM has been especially designed to provide a large number of experimental data relative to each computation (user and system time, total number of different kinds of interactions, storage allocation, garbage collection, and so on). The results of the computation of the function g of the previous section are shown in figure 13. The four columns

Input	user	tot. inter.	fam.	fan-inter.
(g zero)	0.00 s.	38	9	2
(g one)	0.00 s.	64	13	18
(g two)	0.00 s.	200	18	66
(g three)	15.90 s.	2642966	29	8292

Figure 13: The function g

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.
It is also possible to find examples of exponential explosion whith respect to a *linear* grow of the number of family reductions. An interesting case is provided by the λ-term $h = \lambda n.(n\ two'\ two'\ I\ q)$. Using the same technique of the previous section, it is easy to prove that, for this function, the number of family reductions $f(n)$ grows linearly in its input n (in particular, $f(n) = 9 + 3 * n$), while the number of fan-interactions $c(n)$ is given by the formula

$$c(n) = 12 * n - 2 + 4 * 2^n$$

In particular, for any n,

$$2^{\frac{f(n)-7}{3}} \leq c(n) \leq 2^{f(n)}$$

In the table of Figure14, 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.
CamlLight [LM92] is a bytecoded, portable implementation of a dialect of the ML language (about 100K for the runtime system, and another 100K of bytecode for the compiler, versions for the Macintosh and IBM PC are also available) developed at the INRIA-Rocquencourt (France). In spite of its limited dimensions, the performance of CamlLight is quite good for a bytecoded implementation: about five times slower than SML-NJ. We used CamlLight v. 0.5.
Yale Haskell [Ya94] is a complete implementation of the Haskell language developed at Yale University. The Haskell compiler is written in a small Lisp dialect similar to Scheme which runs on top of Common Lisp. Haskell programs are translated into Lisp, and then compiled by the underlying Lisp Compiler. We used Yale Haskell Y2.3b-v2 built on CMU Common Lisp 17f.
The results of the test should give a gist of the power of the optimal graph reduction technique.
In general, the relative amount of fan-interaction rules with respect to the number of family reductions in a computation looks related to the amount of sharing in the term. So, the cases where Lamping's algorithm is exponential in the number of family reductions are also the cases where the performance of BOHM is so amazingly better with respect to convential implementations. The worse cases for BOHM are when Lamping's abstract algorithm is *linear* in the number of families. However, also in these cases, its performance is not as bad as one could expect: BOHM is always five to ten times better than the Yale Haskell *interpreter*, and it is often comparable with the Yale Haskell *compiler*. As an example, in Figure 15 we give the experimental results about the computation of the Fibonacci function, defined in the obvious way. In this case, the number of family reductions is the total number of constructor-destructor interactions. For Haskell, the fibonacci function has been compiled.

4 Discussion

We proved that the complexity of Lamping's algorithm can be exponential in the number of family reductions. We conjecture that, under our assumptions (terms of the λI-calculus reducing to a constant), this value should also also provide an upper bound to the complexity of what we called the *abstract* algorithm. More precisely, if f is the number of family reductions required for normalizing the term, we conjecture that the total number of reductions rules required by Lamping's abstract algorithm is alwasy less than 2^f. This result looks difficult to prove, since *nothing* is known about the structure of graphs along a reduction (so, it is not clear how to use induction).
However, in our opinion, the real issue is of a different nature. In particular, the number of family reduc-

Input	BOHM				Caml-Light	Haskell
	user	tot. inter.	fam.	fan-inter.	user	user
(h one)	0.00 s.	67	12	18	0.00 s.	0.00 s.
(h two)	0.00 s.	119	15	38	0.00 s.	0.02 s.
(h three)	0.00 s.	204	18	66	0.00 s.	0.18 s.
(h four)	0.00 s.	414	21	110	1.02 s.	51.04 s.
(h five)	0.00 s.	1054	24	186	??	??
(h six)	0.02 s.	3274	27	326		
(h seven)	0.07 s.	11534	30	594		
(h eight)	0.26 s.	43394	33	1118		
(h nine)	1.01 s.	168534	36	2154		
(h ten)	4.04 s.	664554	39	4214		

Figure 14: The function h

Input	BOHM				Caml-Light	Haskell
	user	tot. inter.	fam.	fan-inter.	user	user
(fib 4)	0.00 s.	265	75	190	0.00 s.	0.00 s.
(fib 8)	0.02 s.	1991	506	1485	0.00 s.	0.01 s.
(fib 12)	0.13 s.	13822	3462	10360	0.01 s.	0.06 s.
(fib 16)	0.80 s.	94913	23723	71190	0.03 s.	0.38 s.
(fib 20)	4.78 s.	650719	162594	488125	0.23 s.	3.63 s.

Figure 15: Fibonacci

tions does not seem to provide a reasonable lower bound to the "intrinsic" complexity of a λ-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 λ-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 λ nodes have to be annihilated, soon or later). More precisely, consider a normalizing computation for a term M of the λI-calculus that reduces to a constant. Let f be the number of family reductions in the derivation, d be the number of duplication rules, e be the number of fan-annihilation rules, and $|M|$ be the total number of application, abstraction and fan nodes in M. Then, obviously,

$$|M| + 2*d - 2*e - 2*f = 0$$

So, $e + f = d + \frac{|M|}{2}$.
There are several motivations to support our claim. First of all, as it was remarked in [GAL92a], λ and application nodes can be assimilated to fans, and the β-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 β-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 @-cycles in the λ-term[4]. [AL93, ADLR94]. Rouglhy, a @-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 *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 en-

[4]The proof of the bijective correspondence between fan-annihilations and discriminants for different @-cycles will be the subject of a forthcoming paper in collaboration with Cosimo Laneve. Note that, in this way, we relate a dynamic notion (an interaction) to a static one (a path). This result looks particularly relevant in view of complexity issues: computing the number of different paths of this kind is not easy, but it still looks simpler than directly computing the number of fan-annihilations.

vironment 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 one of the crucial issues 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.
Finally, from the implentative point of view, the big problem is to understand if and how Lamping's algorithm could be compiled.

Acknowledgements

We would like to thank Cosimo Laneve for many interesting discussions on the subject of this paper.

References

[As94] A. Asperti. *Linear Logic, Comonads, and Optimal Reductions.* Fundamenta Informaticae, Special Issue devoted to Categories in Computer Science, Vol. 22, n.1, pp.3-22. 1995.

[As95] A. Asperti. *$\delta o!\epsilon = 1$: Optimizing Optimal λ-calculus implementations.* Proc. of the Sixth Conf. on Rewriting Techniques and Applications, (RTA'95), Kaiserlautern, Germany. 1995.

[AGN95] A. Asperti, C. Giovannetti, A. Naletto. *The Bologna Optimal Higher-order Machine.* Technical Report UBLCS-95-9, Laboratory for Computer Science, University of Bologna. To appear in the Journal of Functional Programming.

[AL93] A. Asperti, C. Laneve. *Paths, Computations and Labels in the λ-calculus.* To appear in Theoretical Computer Science, Special issue devoted to RTA'93, Montreal. June 1993.

[AL93b] A. Asperti, C. Laneve. *Interaction Systems II: the practice of optimal reductions.* Technical Report UBLCS-93-12, Laboratory for Computer Science, University of Bologna. To appear in Theoretical Computer Science.

[ADLR94] A. Asperti, V. Danos, C. Laneve, L. Regnier. *Paths in the λ-calculus: three years of communications without understandings.* Proceedings of LICS'94. Paris. 1994.

[DR95] V. Danos, L. Regnier. *Reversible and Irreversible Computations: GOI and λ-machines.* Draft. 1995.

[GAL92a] G. Gonthier, M. Abadi, J.J. Lévy. *The geometry of optimal lambda reduction.* Proc. of the 19th Symposium on Principles of Programming Languages (POPL 92). 1992.

[GAL92b] G. Gonthier, M. Abadi, J.J. Lévy. *Linear Logic without boxes.* Proc. of the 7th Annual Symposium on Logic in Computer Science (LICS'92). 1992.

[Laf90] Y. Lafont. *Interaction Nets.* Proc. of the 17th Symposium on Principles of Programming Languages (POPL 90). San Francisco. 1990.

[Lam89] J. Lamping. *An algorithm for optimal lambda calculus reductions.* Xerox PARC Internal Report. 1989.

[Lam90] J. Lamping. *An algorithm for optimal lambda calculus reductions.* Proc. of the 17th Symposium on Principles of Programming Languages (POPL 90). San Francisco. 1990.

[Le78] J.J.Levy. *Réductions correctes et optimales dans le lambda-calcul.* Thèse de doctorat d'état, Université de Paris VII. 1978.

[Le80] J. J. Lévy. *Optimal reductions in the lambda-calculus.* In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry, Essays on Combinatory Logic, Lambda Calculus and Formalism,* pages 159 - 191. Academic Press. 1980.

[LM92] X. Leroy, M.Mauny. *The Caml Light system, release 0.5. Documentation and user's manual.* INRIA Technical Report. September 1992.

[Ya94] The Yale Haskell Group. *The Yale Haskell Users Manual.* Yale University. October 1994.