The complexity of β-reduction in low orders

Aleksy Schubert[*]

Institute of Informatics
Warsaw University
alx@mimuw.edu.pl

Abstract. This paper presents the complexity of β-reduction for redexes of order $1, 2$ and 3. It concludes with the following results — evaluation of Boolean expressions can be reduced to β-reduction of order 1 and β-reduction of order 1 is in $O(n \log n)$, β-reduction of order 2 is complete for PTIME, and β-reduction of order 3 is complete for PSPACE.

1 Introduction

The mechanism of evaluation in functional languages is based on β-reduction. Thus, it is interesting to study the complexity of the decision problem to answer if a given value (a lambda term) is a result of some program (another lambda term). As most functional programs do not use functions of a very high order, we restrict the research to low orders. This paper concerns reductions in 1st, 2nd and 3rd orders of simply typed lambda calculus.

Another good reason to study these problems is application of the results and techniques to the study of the problem of higher-order matching. Known higher-order matching algorithms are usually based on check if a term obtained by some calculation actually reduces to particular normal form. This exactly corresponds to the situation in our problems. Additionally, the obtained proofs shed more light on the nature of β-reduction which is essential for the final solution of the higher-order matching problem.

Related research There is a similar problem of β-equivalence. It was studied in [Sta79] and non-elementary bound on the complexity of the problem was found. The problem was also discussed in [Mai92] where an alternative proof of the result was described. Another similar problem of finding the length of a β-reduction sequence for a term was studied in [Sch91]. The first attempt to analyse the complexity of β-reduction was presented in [HK96] where a whole hierarchy of orders and complexities was discussed but for a slightly different problem in which restricted syntax is considered and some δ-rules are allowed.

The content of the paper A reduction of evaluation of Boolean expressions to 1st-order β-reduction is proved in Section 3 together with a $O(n \log n)$ algorithm for the reduction, PTIME-completeness for 2nd-order β-reduction is proved in Section 4 and PSPACE-completeness for 3rd-order β-reduction is proved in Section 5.

[*] This work was partly supported by KBN grant no 8 T11C 035 14.

2 Basic notions

We deal with the simply typed λ-calculus denoted by $\lambda_{\rightarrow}$ as in [Bar92]. The results obtained match both Curry and Church-style version of the calculus. We study the β-reduction relation here. One step reduction is denoted by $\rightarrow_{\beta}$. The transitive-reflexive closure of the relation is denoted by $\rightarrow_{\beta}^{*}$. The β-normal form of a term M is denoted by $\mathrm{NF}(M)$. The relation of α-equivalence is denoted by $\equiv_{\alpha}$. We also use the notion of a context which is usually denoted by $C[\cdot]$ and it is a term with a single hole that may be filled in by a term of a suitable type. The operation of 'filling in' does not perform any variable renaming. The context in which its hole is filled in with the term M is denoted by $C[M]$.

The notion of order is defined as: $\mathrm{ord}(\alpha) = 0$ for α atomic and $\mathrm{ord}(\sigma_1 \rightarrow \sigma_2) = \max(\mathrm{ord}(\sigma_1) + 1, \mathrm{ord}(\sigma_2))$.

In the Church-style calculus, the order of the redex $(\lambda x.M)N$ in the term $P = C[(\lambda x.M)N]$ is the order of the type of $\lambda x.M$ assigned in the derivation of the type of P. In the Curry-style calculus, the order of such a redex is the minimum of orders assigned to types of $\lambda x.M$ in type derivations for P.

As far as the Curry-style definition is concerned then there occurs a question whether there is a uniform derivation of a type for P in which all redexes have minimal orders. The answer is 'there is'. The derivation for principal type of P has this property.

The general formulation of the problem we deal with will follow

Problem 1. Input: A $\lambda_{\rightarrow}$ term M_1 with redexes of order at most n and a normal form $\lambda_{\rightarrow}$ term M_2. *Question:* Does M_1 β-reduce to M_2?

We consider the problem for $n = 2, 3, 4$. Note, that we assume that the input is already a term in $\lambda_{\rightarrow}$ and has redexes of suitable order. We do not make any checks that the input values are correct in presented algorithms. These checks require at least essentially polynomial time algorithm which majorises bounds on the resources needed in some constructions presented in the paper. In fact, all presented reductions and algorithms may be performed for both Curry and Church terms.

3 The order 1

3.1 First-order β-reduction is in $O(n \log n)$

The first-order reduction can be performed in $O(n \log n)$ time. Our algorithm uses the notion of graph reduction. We assume here that the reader is familiar with this notion. The recommended readings about graph reduction include: [Lam90,AL93] and [AG98]. Due to limited room, we do not present definitions pertinent to optimal reductions algorithms. We use the presentation included in the latter paper. For the sake of clarity we use the version of graph reduction in which fan-nodes have more than 2 auxiliary ports. This approach can easily be translated into the one with 2-port fan-nodes without affecting the complexity.

Definition 1 `(algorithm for 1st order` β`-reduction)`
Let M_1 and M_2 be the input for the algorithm (we assume here w.l.o.g. that these terms are closed). The algorithm `reduce_1st` is described as follows. We need an additional stack S and a counter i. Some nodes of the graph will be marked during the reduction. We proceed as follows:

1. Translate M_1 into its graph of reduction, initiate S to the empty stack.
2. Walk through the starting λ-nodes without any change.
3. Initiate i to 0.
4. Go through @-nodes incrementing i at each one and taking their left branch until you meet a fan-node, an auxiliary port of a λ-node, a marked node or a principal port of a λ-node.
   (a) if it is a fan-node, an auxiliary port of a λ-node or a marked node then check if S is empty if it is go to the point (5) if it is not, pop the value of i from S, then pop a node A from the stack, and perform the β-redex above the node A marking the topmost node of the argument of the redex; finally, go to the point 4;
   (b) if it is a principal port of a λ-node and $i > 0$ then decrement i, push the λ-node on S, push i, step to the right branch of the last @-node and begin the whole procedure from the point (3);
   (c) if it is a principal port of a λ-node and $i = 0$ then go through the λ-node without any change and step to the point 4.
5. Perform the read-back of the graph; the resulting term is M_3.
6. Check the α-equivalence of M_3 and M_2.

Theorem 1. *Let M_1 have redexes of order at most 1 and M_2 be in normal form. The algorithm* `reduce_1st` *results in success on these terms iff $M_1 \to_\beta^* M_2$.*
Moreover, `reduce_1st` *needs only $O(n \log n)$ time to run.*

Proof. The algorithm is correct as it is only a strategy in an optimal reduction algorithm.

Let us analyse the complexity of the algorithm. Let n be the size of the input for `reduce_1st`.

The translation of the term to the graph can be performed in $O(n)$ time using usual syntax analysis methods. The rest of the algorithm visits each node at most 2 times and the number of steps performed for each node is bounded by a constant except for the time needed to store i and a node on the stack. The last operation takes $O(\log n)$ time because of the length of the counter and the pointer to the node. This altogether gives $O(n \log n)$ time.

3.2 Boolean expressions reduce to first-order β-reduction

Boolean expression is an expression that is built of the connectives $\wedge, \vee$ and values `true` and `false`. An example is (`true` $\wedge$ `false`) $\vee$ `true`. We can associate with each expression of this kind its value which is generated according to the truth tables of logical connectives $\wedge$ and $\vee$. The problem of evaluation of Boolean expressions is:

Definition 2 (`evaluation of Boolean expressions`)
Input: A Boolean expression E.
Question: Is `true` the value of E?

The problem is in ALOGTIME (see [Bus87]). In order to relate the 1st order situation to 2nd and 3rd orders we present a first-order (i.e. in first-order logic) reduction of the problem to the first-order β-reduction problem. This presentation is only for the sake of completeness with the rest of the paper where some variations of Boolean formulas are dealt with. In fact, we only relate β-reduction to the evaluation of expressions which itself has no proof of ALOGTIME-hardness. A helpful definition of logical values is

Definition 3 (`Boolean values`)
We define terms corresponding to Boolean values as TRUE $= \lambda x_1 x_2.x_1$ and FALSE $= \lambda x_1 x_2.x_2$.

The translation from Boolean expressions is:

Definition 4 (`translation from Boolean expressions to` $\lambda_{\rightarrow}$)
The translation from Boolean expressions to $\lambda_{\rightarrow}$ has as an input a Boolean expression E and results in two terms M_1 and M_2. We put $M_1 = \mathtt{E2L}(E)$ and $M_2 =$ TRUE. The function `E2L` is defined by induction on the form of the Boolean expression (we may assume w.l.o.g. that the expressions do not contain negation):

$$\begin{array}{ll} \mathtt{E2L}((E_1 \wedge E_2)) = \lambda xy.(\mathtt{E2L}(E_1))((\mathtt{E2L}(E_2))xy)y; & \mathtt{E2L}(\mathtt{true}) = \text{TRUE}; \\ \mathtt{E2L}((E_1 \vee E_2)) = \lambda xy.(\mathtt{E2L}(E_1))x((\mathtt{E2L}(E_2))xy); & \mathtt{E2L}(\mathtt{false}) = \text{FALSE}. \end{array}$$

Theorem 2. *Let E be a Boolean expression. E has the result* `true` *iff the term* $\mathtt{E2L}(E)$ *reduces to* TRUE.
Moreover, the term $\mathtt{E2L}(E)$ *has redexes of order at most 1.*

Proof. The main claim is obtained by a routine induction on the expression E.
The only redexes in the term occur during the translation in cases for $\wedge$ and $\vee$. By induction on E, we can show that $\mathtt{E2L}(E)$ is of the type $\alpha \rightarrow \alpha \rightarrow \alpha$ so these redexes are of order 1.

Theorem 3. *The term* $\mathtt{E2L}(E)$ *may be represented by a first-order formula over the signature of Boolean expressions.*

Proof. The formula that constitutes the universe has 5 variables $x_1, \ldots, x_5$. The first one is used to determine which operator is encoded the rest is used to encode the Boolean representation of nodes needed to represent a Boolean connective. The first lambda node is encoded as 0000, then x as 0001, the second λ node as 0010 and so on. The edge relation (in a λ) term is defined so that the first coordinate is constant and the other coordinates represent suitable bits as in the above-mentioned encoding. Details are left for the reader.

4 The order 2

4.1 Second-order β-reduction is in PTIME

The second-order reduction can be performed in polynomial time. Our algorithm uses again the notion of graph reduction.

Let us see what is the graph reduction look like in this case. The starting point for this reduction is shown in Figure 1(a). The figure presents a β-redex located in a term (the omission of a part of the context of the redex is denoted by the dotted line). The star marked by G_0 symbolizes the body of the λ-abstraction that takes part in β-reduction. The circle marked by G_1 symbolizes the body of the argument that takes part in β-reduction. For the sake of clarity we denote a set of fan-nodes by a single fan with many entry ports.

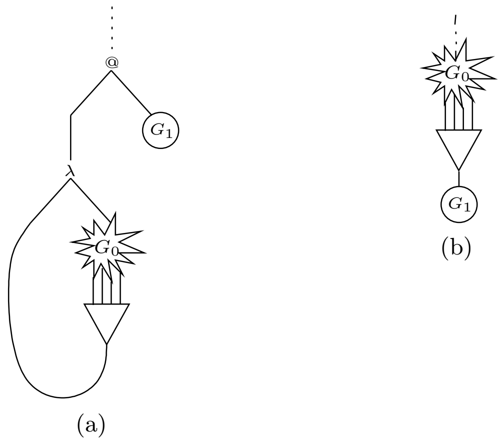

Fig. 1. (a) The starting point for 2nd-order β-reduction. (b) The result of the first phase of β-reduction

The result of the first β-reduction step is shown in Figure 1(b). As we see the argument G_1 goes into several places of the subterm G_0. Since we use a fan-node the argument is not copied. This kind of reduction is performed during the first phase of our algorithm. Note that the performing of some β-redexes may introduce other ones. There are two ways in which this redex may occur: the one as in the term $(\lambda x_1.(\lambda x_2.M))N_1N_2$ or the other as in the term $(\lambda x_1.C[x_1M])(\lambda x_2.N)$. We conduct our reduction in such a way that redexes of the first kind are contracted in this phase whereas the redexes of the second kind are not. We achieve this behaviour later in definitions by marking the edge outgoing from G_1 (see the point 2 in Definition 5). Note that this does not force us to reduce some redexes but these redexes are certainly of order 1. We repeat this kind of reduction until there are no redexes. The result of the process is a term that has no 2nd-order redexes.

Although there are no explicit redexes (except for the marked ones), we have some redexes hidden behind fan-nodes. We can extract these redexes as in Figure 2(a) and then contract them to the form presented in Figure 2(b).

This process should be repeated until there are no λ-nodes behind fan-nodes (in other words, until there are no paths which enter a fan-node and then after some number of brackets and croissants immediately enter a λ-node).

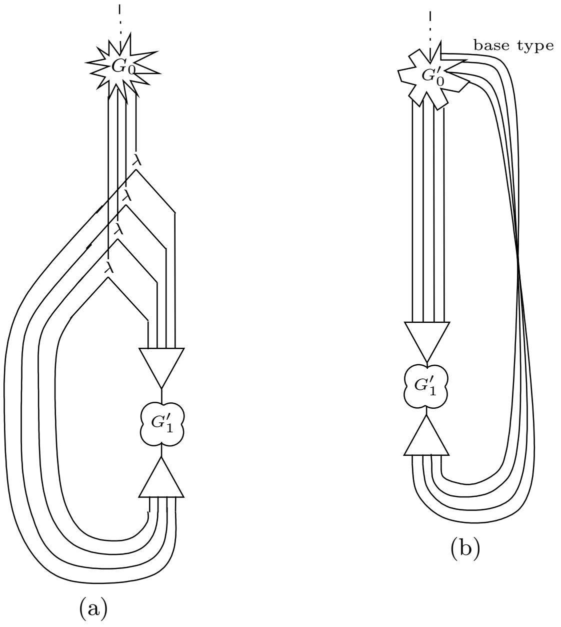

Fig. 2. (a) The λ-nodes are extracted from the fan-nodes. (b) After the first-order reduction

Definition 5 (the algorithm for 2nd order)
Let M_1 and M_2 be the input data for the algorithm. The algorithm `reduce_2nd` proceeds performing the following steps:

1. Translate the term M_1 into the corresponding graph.
2. Perform one by one all existing β-reductions (after performing a reduction step mark the edge that goes from the argument; in future reductions in this phase, omit redexes with such edges going out of λ-nodes).
3. Push all λ-nodes through fans.
4. Perform one by one all existing β-reductions and if necessary go to Point 3.
5. Reduce all matching fan-nodes so that they disappear.
6. Perform the read-back of the resulting graph; if the result is larger than the term M_2: fail. Let M_3 be the result of the read-back.
7. If $M_2 \equiv_\alpha M_3$ then success otherwise failure.

For the sake of clarity, we omit reductions for brackets and croissants in the description assuming that they are performed implicitly, resulting in the disappearance of nodes that occur at principal ports of fan-nodes, λ-nodes or @-nodes.

Theorem 4. *If* `reduce_2nd` *stops with success then* $\mathrm{NF}(M_1) \equiv_\alpha M_2$. *If the algorithm* `reduce_2nd` *fails then* $\mathrm{NF}(M_1) \not\equiv_\alpha M_2$.

Proof. The correctness of `reduce_2nd` is implied by the correctness of the graph reduction. The only thing to be proved is that before entering Point (6) in Definition 5, we obtain a graph that has no β-redexes in any reduction sequence so that the read-back gives the normal form.

We prove the last claim in two steps: First, we prove that after performing the step (2) there will be no 2nd-order redexes. Then we prove that after performing the steps (3-4) there will be no 1st-order redexes. These are proved by contradiction. Details are left for the reader.

In order to analyse the complexity of the algorithm `reduce_2nd`, we have to introduce the notion of *mixed bracket property.* This notion formalises and generalises the property of the old-fashioned arithmetical notation in which different kinds of parenthesis are used as in the expression: $[(2+3)\cdot 5+6]\cdot 11$. This property says that a parenthesis of A kind may be closed only if each parenthesis of any other kind B that is opened after the parenthesis of kind A is closed. For example, if we open [and then (then in order to put] we have to put) first.

Definition 6 `(the mixed parenthesis property)`
We say that a path ρ has the *mixed parenthesis property* iff for any fan-nodes A and B if ρ enters a fan-node A at an auxiliary port α and afterword a fan-node B at an auxiliary port β then it must exit the auxiliary port β of a node corresponding to B before it exits the auxiliary port α of a node corresponding to A.

Fact 1 *During the reductions performed in* `reduce_2nd` *all paths have mixed parenthesis property.*

Proof. The proof is by induction on the number of steps of reduction during the algorithm `reduce_2nd`. Details are left for the reader.

Theorem 5. *The procedure* `reduce_2nd` *runs in $O(n^2)$ time.*

Proof. Let $n = |M_1| + |M_2|$. Most of the points have easily verified linear time complexity. Point (6) needs $O(n^2)$ steps since the mixed parenthesis property (Fact 1) ensures that there are no two fan-nodes that meet with principal ports. If the matching fan-nodes exist then they are reduced in Point (5), if there are two non-matching fan-nodes then they break the mixed parenthesis property. As there are no fan-nodes that meet with principal ports, each path that exits a principal port of a fan-node and then after some, possibly non-zero, number of other (non-bracket and non-croissant) nodes enters a principal port of a fan-node must go through either @-node or λ-node. This ensures that such a node is visited once at least after visiting all fan-nodes. At last (7) can be performed in $O(n)$ time since M_2 is a part of input and α conversion can be performed in $O(m)$ where m is the size of terms to be checked. This altogether gives the time $O(n^2)$.

4.2 Second-order β-reduction is PTIME-hard

The problem of the evaluation of Boolean circuits is reduced to the problem of β-reduction in second-order in this section. The reduction is in LOGSPACE. This implies that second-order β-reduction is PTIME-hard.

Definition 7 (Boolean circuit)
A Boolean circuit is a directed acyclic graph such that:

- its nodes are labeled with $\vee, \wedge, \neg$, `true`, or `false` and a single node labeled with `result`;
- nodes labeled with $\vee$ and $\wedge$ have two outgoing edges;
- nodes labeled with $\neg$ and `result` have a single outgoing edge;
- nodes labeled with `true` and `false` have no outgoing edges.

The result of a Boolean circuit can be defined recursively in an obvious way e.g. the value of a $\vee$-node is $v_1 \vee v_2$ where v_1 is the value of the node at the end of the first outgoing edge and v_2 is the value of the node at the end of the second outgoing edge, the value of `result` is v where v is the value of the node at the end of the outgoing edge.

Definition 8 (the problem of evaluation of a Boolean circuit)
The problem of the evaluation of a Boolean circuit is:
Input: A Boolean circuit $\mathcal{C}$
Question: Does the circuit have the result `true`?

The above-mentioned problem is PTIME-hard (see Theorem 8.1 in [Pap95]).

We define *level* of a node in a Boolean circuit. This notion helps use define the reduction.

Definition 9 (level of a node)
In a Boolean circuit $\mathcal{C}$, the node `result` has the level 0. A node n has the level l if $l = \max\{l_1, \ldots, l_k\} + 1$ where $\{l_1, \ldots, l_k\}$ is the set of levels for nodes n' such that (n', n) is an edge in $\mathcal{C}$.

We denote by $\mathcal{C}_n$ the set of nodes of the level n.

As Boolean circuits use logical connectives $\vee, \wedge$ and $\neg$, we should define their counterparts in λ-calculus. We also define logical values and quantifiers which are needed later.

Definition 10 (connectives for translations)

$$\begin{array}{ll}
\mathrm{TRUE} = \lambda x_1 x_2 . x_1 & \forall = \lambda \phi x_1 x_2 . \mathrm{AND}(\phi \mathrm{TRUE} x_1 x_2)(\phi \mathrm{FALSE} x_1 x_2) \\
\mathrm{FALSE} = \lambda x_1 x_2 . x_2 & \exists = \lambda \phi x_1 x_2 . \mathrm{OR}(\phi \mathrm{TRUE} x_1 x_2)(\phi \mathrm{FALSE} x_1 x_2) \\
\mathrm{AND} = \lambda b_1 b_2 x_1 x_2 . b_1 (b_2 x_1 x_2) x_2 & \mathrm{NOT} = \lambda b_1 x_1 x_2 . b_1 x_2 x_1 \\
\mathrm{OR} = \lambda b_1 b_2 x_1 x_2 . b_1 x_1 (b_2 x_1 x_2) &
\end{array}$$

Definition 11 (reduction from Boolean circuits)
This reduction is recursively defined on the level of nodes. We introduce variables $\{x_i^j \mid 1 \le i \le k_j j\}$ where k_j is the number of nodes on the level j.

– The term LEVEL_{-1} is defined as $x^0_{\mathtt{result}}$.
– The term LEVEL_{n+1} is defined on the basis of the term LEVEL_n as

$$(\lambda x_1^{n+1} \ldots x_{k_{n+1}}^{n+1}.\mathrm{LEVEL}_n) B_1 \ldots B_{k_{n+1}}$$

where
- $B_i = \mathrm{AND} x_k^l x_{k'}^{l'}$ if the i-th node on the level $n+1$ is $\wedge$ and one of its outgoing edges leads to k-th node on the l-th level and the other to k'-th node on the l'-th level;
- $B_i = \mathrm{OR} x_k^l x_{k'}^{l'}$ if the i-th node on the level $n+1$ is $\vee$ and one of its outgoing edges leads to k-th node on the l-th level and the other to k'-th node on the l'-th level;
- $B_i = \mathrm{NOT} x_k^l$ if the i-th node on the level $n+1$ is $\neg$ and its outgoing edge leads to k-th node on the l-th level;
- $B_i = \mathrm{TRUE}$ if the i-th node on the level $n+1$ is `true`;
- $B_i = \mathrm{FALSE}$ if the i-th node on the level $n+1$ is `false`.
- $B_i = \mathrm{FALSE}$ if the i-th node on the level $n+1$ is `false`.

(Note that $\mathrm{LEVEL}_0 = (\lambda x^0_{\mathtt{result}}.x^0_{\mathtt{result}})A$, where A is either TRUE or FALSE.)

Theorem 6. *Let G be a Boolean circuit and n its maximum level of nodes. G has the result* `true` *iff the term* LEVEL_n *reduces to* TRUE.
Moreover, the term LEVEL_n *has redexes of order at most 2.*

Proof. The proof is by induction on the maximal level of the graph G. The induction step consists in suitable reduction of the highest level so that it disappears and the number of levels is decreased.

Theorem 7. *The term* LEVEL_n *may be generated with use of additional space of size* $O(\log |G|)$.

Proof. W.l.o.g. we may assume that Boolean circuits have assigned to each node its level. This allows us to use a counter that says on which level we are. This is enough to identify where should be placed appropriate variables and terms $\mathrm{AND}, \mathrm{OR}, \mathrm{NOT}, \mathrm{TRUE}$ and FALSE. Such a counter needs $O(\log n)$ space. Another counter is needed for names of variables, but $O(\log n)$ is sufficient here too. Details are left for the reader.

5 The order 3

5.1 Third-order β-reduction is in PSPACE

The third-order reduction can be performed in polynomial space. Our algorithm, similarly to the second-order case, uses the notion of graph reduction.

Let us see how does the process of graph reduction look like in this case. The starting point of such a reduction may look like in Figure 3(a). The figure

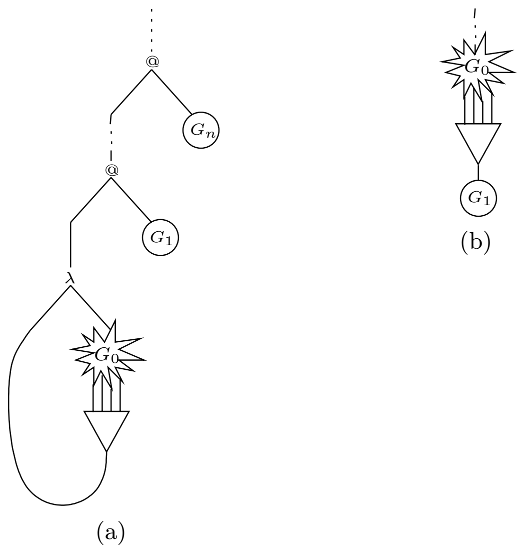

Fig. 3. (a) The starting point for 3rd-order β-reduction. (b) The result of the first phase of β-reduction

presents a β-redex in a λ-term. The star marked by G_0 denotes the body of the λ-abstraction that takes part in β-reduction. The circle marked by G_1 denotes the body of the argument that takes part in β-reduction. The dotted lines represent parts of the term that are missing in the picture.

The result of the first β-reduction step is shown in Figure 3(b). As we see, the argument G_1 goes into several places of the subterm G_0 as in the 2nd-order case. This kind of reduction is performed during the first phase of our algorithm. Again, the process of reduction may introduce other ones. Again, we perform only some of the new redexes similarly to the 2nd-order case. We repeat this kind of reduction until there are no redexes. The result of the process is a term that has no 3rd-order redexes.

Although there are no explicit redexes (except for the marked ones) we have some redexes hidden behind fan-nodes. We can extract these redexes as in Figure 4(a) and then contract them with @-nodes that come from G_0 as depicted in Figure 4(b). This process should be repeated until there are no λ-nodes behind fan-nodes (in other words, until there are no paths which enter a fan-node and then after some number of brackets and croissants immediately enter a λ-node).

The result of such reduction is depicted in Figure 5(a). We have two fan-nodes surrounding G_1' — the upper one because the term occurs in several places and the lower one because different terms are substituted for a variable depending on which place is taken into account. This ends the second phase of the reduction (the reduction of 2nd-order redexes).

The last phase of the reduction begins — the reduction of 1st-order redexes. These redexes occur as in Figure 5(b) and begin to interact with the graph G_1'. As the 1st-order variable that took part in the 2nd-order reduction (the lambdas of which were multiplied in Figure 4(a)) can occur in several places inside G_1', several @-nodes will take part in the reduction of 1st-order redexes. We can see

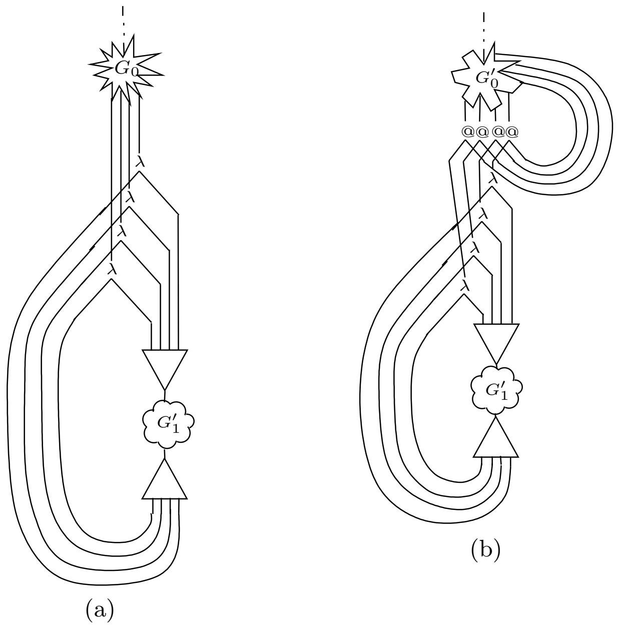

Fig. 4. (a) The λ-nodes are extracted from fan-nodes. (b) The λ-nodes meet suitable @-nodes

these @-nodes in Figure 6(a). As this multiplication concerns only one variable, we have a fan-node that performs this operation — also visible in Figure 6(a).

The fan-nodes that meet begin to interact. The result of the interaction is depicted in Figure 6(b) where it is denoted by the letter F. When we zoom the area denoted by F we will see a complicated web of links which is shown in Figure 7. The next step to perform is to push @-nodes through fan-nodes. The result of performing this step is partially shown in Figure 8(a). Each upper fan-node gets multiplied as it must go into two edges outgoing from each @-node. The next phase is to push λ-nodes through fan-nodes and perform β-redexes. The result of these operations is depicted in Figure 8(b). The left, big fan-node indicates that the body of the applied function goes into the place where application was situated previously. The right, big fan-node indicates that arguments of the application are placed in variables.

Definition 12 `(the algorithm for 3rd order)`
Let M_1 and M_2 be the input data for the algorithm. The algorithm `reduce_3rd` proceeds as follows:

1. Translate the term M_1 into the corresponding graph.
2. Perform one by one all existing β-reductions (after performing a reduction step mark the edge that goes from the argument; in future reductions within this phase, omit redexes with such an edge going out of a λ-node).
3. Clear all markings.
4. Push all fans through λ-nodes.
5. Perform one by one all existing β-reductions (again with marking).

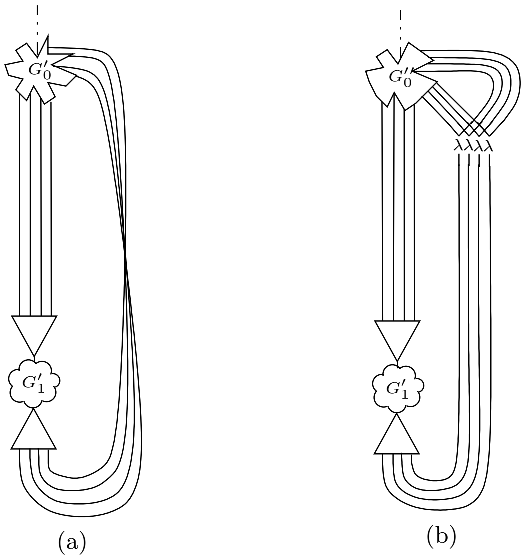

Fig. 5. (a) The result of the second phase of reduction. (b) First-order λ-nodes begin to reduce

6. Perform all interactions between fans and afterword push all fans through λ- and @-nodes.
7. Perform one by one all existing β-reductions.
8. Perform the read-back of the resulting graph; if the result is larger than the term M_2 to be equated: fail. Let M_3 be the result of the read-back.
9. If $M_2 \equiv_\alpha M_3$ then success otherwise failure.

In order to precisely describe the complexity we need a special notion called the level of a redex.

Definition 13 `(level of a redex)`
Let us define a special kind of reduction in which $(\lambda x.M)N \to_{\beta'} M[x := N^*]$ where N^* is the term N with a special marking (the marking should be understood as a new kind of language symbol similar to the application or abstraction, i.e. the marking is applied locally not throughout the whole term N and thus is not visible in redexes inside N). Note that we forbid the reduction $(\lambda x.M)^* N \to_\beta M[x := N^*]$. Of course, all reductions performed in this framework may be performed as the usual β-reduction. Thus paths of β'-reduction may be treated as paths of β-reduction. On the other hand, each path of β-reduction $M_1, \ldots, M_n$ may be presented as $M_1, \ldots, M_{i_1}, M_{i_1+1}, \ldots, M_{i_2}, \ldots, M_{i_{k-1}+1}, \ldots, M_{i_k}$ where redexes between terms $M_{i_j+1}, \ldots, M_{i_{j+1}+1}$ can be performed using β'-reduction and $M_{i_{j+1}+1}$ is a β'-normal form. The β-redexes in j-th such section are called *redexes of the level* j.

It is easily verified that each reduction of a term with redexes with order at most n has redexes of order at most $n-2$. If n is the highest order of the redex in a term then redexes of the order n are reduced during the 0-level section, the redexes of the level $n-1$ are reduced during the 1-level section and so on.

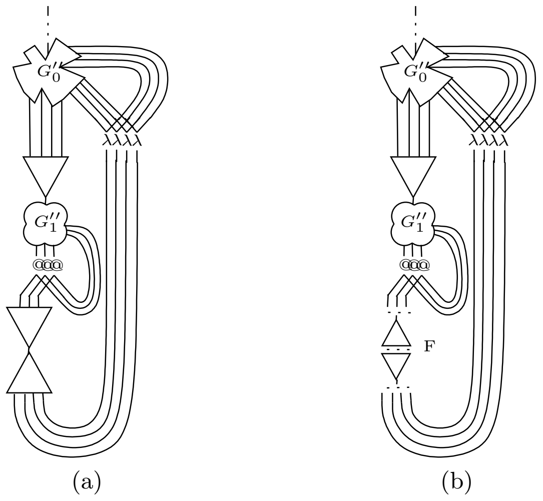

Fig. 6. (a) Multiple occurrences of 1st-order variables with surrounding @-nodes. (b) Fan-nodes interact

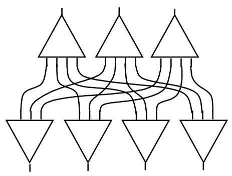

Fig. 7. The interaction of fan-nodes

Also the notion of the level of a redex straightforwardly translates to graph reduction. The algorithm for 3rd-order reduction needs redexes of order at most 1. The redexes of the level 0 are reduced in the step (2) of the algorithm, the redexes of the level 1 are reduced in the step (5) of the algorithm and at last redexes of the level 2 are reduced in the step (7) of the algorithm.

Theorem 8. *Let M_1 have redexes of order at most* 3 *and M_2 be in normal form. The algorithm* `reduce_3rd` *results in success on these terms iff $M_1 \to_\beta^* M_2$.*

Moreover, `reduce_3rd` *needs only $O(n^3)$ space to run.*

Proof. The algorithm is correct as it is only a strategy in an optimal reduction algorithm.

The analysis of the complexity of the algorithm is quite routine. Here are the most difficult cases:

Point (4) requires the multiplication of λ-nodes and fan-nodes. This multiplication is performed as in Figure 4(a) and so the number of new λ-nodes is bounded by $k_1 \cdot k_2$ where k_1 is the number of variables that take part in the step (2) of the algorithm and k_2 is the number of variables that are in the arguments of the former variables in the input. This gives the $O(n^3)$ space. The fan-nodes are replicated only $O(n)$ times as the number of variables that take part in the step (2) majorises the number of replications. The last number is of $O(n^2)$ magnitude.

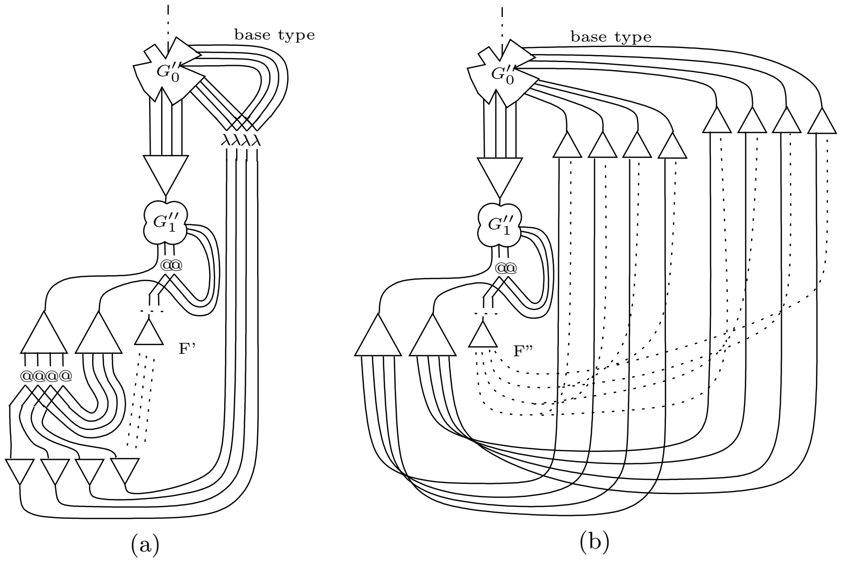

Fig. 8. (a) One application goes between fan-nodes. (b) After pushing λ-nodes through fans, applications are reduced

Point (7) is a usual walk through the graph in hand. As the size of the graph is $O(n^3)$, the time and thus the space is $O(n^3)$.

This altogether gives $O(n^3)$ space.

5.2 Third-order β-reduction is PSPACE-hard

We relay here on *quantified Boolean formulae* problem (QBF). Which consists in deciding whether a given formula with quantified Boolean variables is true. This problem in known to be PSPACE-complete. We present a PTIME reduction of the QBF problem to the 3rd-order reduction problem.

The translation is defined as follows

Definition 14 (`translation from QBF to` $\lambda_{\rightarrow}$)
The translation from QBF to $\lambda_{\rightarrow}$ has as an input a QBF sentence ϕ and as a result two terms M_1 and M_2. We put $M_1 = \mathtt{Q2L}(\phi)$ and $M_2 = \mathrm{TRUE}$. The function `Q2L` is defined by induction on the form of the QBF formula:

$\mathtt{Q2L}(\mathtt{true}) = \mathrm{TRUE}$; $\quad$ $\mathtt{Q2L}(\mathtt{false}) = \mathrm{FALSE}$;
$\mathtt{Q2L}(x) = x$ where x is a variable; $\quad$ $\mathtt{Q2L}(\neg\phi) = \mathrm{NOT}(\mathtt{Q2L}(\phi))$;
$\mathtt{Q2L}(\phi_1 \wedge \phi_2) = \mathrm{AND}(\mathtt{Q2L}(\phi_1))(\mathtt{Q2L}(\phi_2))$; $\quad$ $\mathtt{Q2L}(\phi_1 \vee \phi_2) = \mathrm{OR}(\mathtt{Q2L}(\phi_1))(\mathtt{Q2L}(\phi_2))$;
$\mathtt{Q2L}(\forall x.\phi) = \forall(\lambda x.\mathtt{Q2L}(\phi))$ $\quad$ $\mathtt{Q2L}(\exists x.\phi) = \exists(\lambda x.\mathtt{Q2L}(\phi))$

Theorem 9. *A QBF sentence ϕ is true iff the term* $\mathtt{Q2L}(\phi)$ *reduces to* TRUE. *Moreover, the term* $\mathtt{Q2L}(\phi)$ *has redexes of order at most 3.*

Proof. We need a little bit extended version of the claim:

Let ϕ be a QBF formula with free variables in $A = \{x_1, \ldots, x_n\}$. The formula ϕ is true under the valuation $v : A \rightarrow \{\mathtt{true}, \mathtt{false}\}$ iff the term $\mathtt{Q2L}(\phi)[x_1 := \mathtt{Q2L}(v(x_1)), \ldots, x_n := \mathtt{Q2L}(v(x_n))]$ reduces to TRUE.

The proof is by straightforward induction on the structure of ϕ and is left for the reader.

The redexes in the result of translation occur in subterms beginning with $\mathrm{AND}, \mathrm{OR}, \mathrm{NOT}, \forall, \exists$. The type for AND, OR and NOT is of order 2. These terms take as arguments values of the type $\alpha \to \alpha \to \alpha$ (which is the type of Boolean terms TRUE and FALSE). The type for $\forall$ and $\exists$ is more complicated and is of order 3. These terms take an argument of the type $(\alpha \to \alpha \to \alpha) \to \alpha \to \alpha \to \alpha$. No other terms occur in redex positions in translated terms.

We have also by routine analysis:

Theorem 10. *The translation from QBF to $\lambda_{\to}$ can be performed in $O(n \log n)$ time.*

6 Acknowledgments

I would like to thank Damian Niwiński for his hints concerning good problems to reduce from. Thanks also go to anonymous referees who helped to improve the paper. I also thank dear Joanna for her help in improving my English.

References

[AG98] Andrea Asperti and Stefano Guerrini, *The optimal implementation of functional programming languages*, Cambridge University Press, 1998.

[AL93] Andrea Asperti and Cosimo Laneve, *Interaction Systems II: the practice of optimal reductions*, Tech. Report UBLCS-93-12, Laboratory for Computer Science, Universitá di Bologna, 1993.

[Bar92] H. P. Barendregt, *Lambda calculi with types*, Handbook of Logic in Computer Science (S. Abramsky, D. M. Gabbay, and T. S. E. Mainbaum, eds.), vol. 2, Oxford University Press, 1992, pp. 117–309.

[Bus87] S.R. Buss, *The boolean formula value problem is in ALOGTIME*, Proceedings of the 19th Annual ACM Symposium on Theory of Computing, ACM Press, 1987, pp. 123–131.

[HK96] G. Hillebrand and P. Kanellakis, *On the expressive power of simply typed and let-polymorphic lambda calculi*, Proceedings of the 11th IEEE Conference on Logic in Computer Science, 1996, pp. 253–263.

[Lam90] John Lamping, *An algorithm for optimal lambda calculus reductions*, Proceedings of 17th ACM Symposium on Principles of Programming Languages, 1990, pp. 16–30.

[Mai92] H. Mairson, *A simple proof of a theorem of statman*, Theoretical Computer Science (1992), no. 103, 213–226.

[Pap95] Ch. H. Papadimitriou, *Computational complexity*, Addison–Wesley, 1995.

[Sch91] H. Schwichtenberg, *An upper bound for reduction sequences in the typed λ-calculus*, Archive for Mathematical Logic (1991), no. 30, 405–408.

[Sta79] R. Statman, *The typed λ-calculus is not elementary recursive*, Theoretical Computer Science (1979), no. 9, 73–81.