The complexity of β-reduction in low orders

Aleksy Schubert*
Institute of Informatics
Warsaw University
alx@mimuw.edu.pl

9 November, 2000

Abstract

We study the complexity of β-reduction for redexes of order $2, 3$ and 4. The obtained results are: evaluation of boolean expressions can be reduced to β-reduction of order 2 and β-reduction of order 2 is in $O(n \log n)$, β-reduction of order 3 is complete for PTIME, and β-reduction of order 4 is complete for PSPACE.

1 Introduction

The mechanism of evaluation in functional languages is based on β-reduction. Thus, it is interesting issue 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 very high order we chose to make research primarily for low orders. This paper concerns reductions in 2nd, 3rd and 4th orders of simly typed lambda calculus.

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

Related research There is similar problem of β-equivalence. This was studied in [Sta79] and non-elementary bound on the complexity of the problem was found. This problem was studied also in [Mai92] where an alternative proof of the result is described. Another similar problem of finding the

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

length of a β-reduction sequence for a term is 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 is discussed but for a slightly different problem where restricted syntax is considered and some δ-rules are allowed.

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

2 Basic notions

We deal with simply typed λ-calculus denoted by $\lambda_{\rightarrow}$ as in [Bar92]. The results we obtain here have their versions in 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 use also the notion of context which is usually denoted by $C[\cdot]$ and 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. A context where its hole is filled in with the term M is denoted by $C[M]$.

The notion of order is defined as:

Definition 1 `(order)`
The order of a type is defined as

$$\begin{array}{lcl} \mathrm{ord}(\alpha) & = & 1 \text{ for } \alpha \text{ atomic;} \\ \mathrm{ord}(\sigma_1 \rightarrow \sigma_2) & = & \max(\mathrm{ord}(\sigma_1)+1, \mathrm{ord}(\sigma_2)) \end{array}$$

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.

If 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 most general formulation of the problem we deal with is presented hereafter

Problem 2 `(reducibility in order` n`)`
Input: *A* $\lambda_\to$ *term* M_1 *with redexes of order at most* n *and a normal form* $\lambda_\to$ *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 $\lambda_\to$ and has redexes of suitable order. We do not make any checks that the input values are correct in presented algorithms. Such checks require at least essentially polynomial time algorithm which majorises bounds on the resources needed in some of constructions presented in the paper. In fact, all the presented reductions and algorithms may be performed for both Curry and Church terms.

3 The order 2

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

The second-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 suitable texts about graph reduction include: [Lam90] or [AL93]. We use the presentation included in the latter paper. For the sake of clarity of presentation we use the version of graph reduction where fan-nodes have more than 2 auxiliary ports. This approach can easily be translated to the one with 2-port fan-nodes without affecting the complexity.

Definition 3 `(algorithm for 2nd order` β`-reduction)`
Let M_1 and M_2 be the input for the algorithm. The algorithm `reduce_2nd` 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 so go to the point (5) else pop the value of i from S, 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; 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 3.1 `(the algorithm reduce_2nd is in` $O(n \log n)$`)`
Let M_1 have redexes of order at most 2 and M_2 be in normal form. The algorithm `reduce_2nd` *results in success on these terms iff $M_1 \to_\beta^* M_2$.*

Moreover, `reduce_2nd` *needs only $O(n)$ 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_2nd`.

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 latter operation takes $O(\log n)$ time because of the length of the counter and the pointer to the node.

This altogether gives $O(n)$ time. □

3.2 Boolean expressions reduce to second-order β-reduction

Boolean expression is an expression that is build from the connectives $\wedge, \vee$ and values `true` and `false`. An example is (`true` $\wedge$ `false`) $\vee$ `true`. We can associate with each such an expression 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 4 `(evaluation of boolean expressions)`
Input: A boolean expression E.
Question: Is `true` the value of E.

The problem is in ALOGTIME (see [Bus87]). We present a first-order reduction of the problem to the second-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. A helpful definition of logical values is

Definition 5 `(boolean values)`

$$\begin{array}{rcl} \text{TRUE} & = & \lambda x_1 x_2 . x_1 \\ \text{FALSE} & = & \lambda x_1 x_2 . x_2 \end{array}$$

The translation from boolean expressions is:

Definition 6 `(translation from boolean expressions to` $\lambda_{\rightarrow}$`)`
The translation from boolean expressions to $\lambda_{\rightarrow}$ has as an input a boolean expression E and as a result two terms M_1 and M_2. We put $M_1 = \texttt{E2L}(E)$ and $M_2 = \text{TRUE}$. The function `E2L` is defined by induction on the form of the boolean expression:

- $\texttt{E2L}((E_1 \wedge E_2)) = \lambda xy.(\texttt{E2L}(E_1))((\texttt{E2L}(E_2))xy)y$;
- $\texttt{E2L}((E_1 \vee E_2)) = \lambda xy.(\texttt{E2L}(E_1))x((\texttt{E2L}(E_2))xy)$;
- $\texttt{E2L}(\texttt{true}) = \text{TRUE}$;
- $\texttt{E2L}(\texttt{false}) = \text{FALSE}$.

Theorem 3.2 `(boolean expression and` $\lambda_{\rightarrow}$`)`
Let E be a boolean expression. E has the result `true` *iff the term* $\texttt{E2L}(E)$ *reduces to* TRUE*.*

Moreover, the term $\texttt{E2L}(E)$ *has redexes of order at most 2.*

Proof:
The main claim follows 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 $\texttt{E2L}(E)$ is of the type $\alpha \rightarrow \alpha \rightarrow \alpha$ so these redexes are of order 2. □

Theorem 3.3 `(the 'running time' for E2L)`
The term $\texttt{E2L}(E)$ *may be represented by a first-order formula over the signature of boolean expressions.*

Proof:
The formula that constituates 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 just mentioned encoding.

Details are left for the reader. □

4 The order 3

4.1 Third order β-reduction is in PTIME

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

Let us see how does the graph reduction in this case look like. The starting point of such a reduction is shown on Figure 1(a). The figure pictures a β-redex somewhere in some term (the omission of a part of the context of the redex is denoted by the dotted line). The star denoted by G_0 symbolises the body of the λ-abstraction that takes part in β-reduction. The circle denoted by G_1 symbolises the body of the argument that takes part in β-reduction. For the sake of clarity of presentation we represent a set of fan-nodes by a single fan with many entry ports.

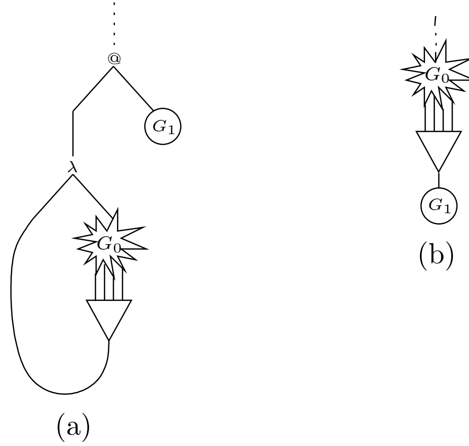

Figure 1: (a) The starting point for 3rd-order β-reduction. (b) The result of the first phase of β-reduction

The result of the first β-reduction step is shown on Figure 1(b). As we see the argument G_1 goes into several places of the subterm G_0. Since we use fan-node the argument is not copied. This kind of reduction is performed during the first phase of our algorithm. Note that performing some β-redexes may introduce other ones. There are two ways such a redex may occur: as in the term $(\lambda x_1.(\lambda x_2.M))N_1N_2$ or as in the term $(\lambda x_1.C[x_1M])(\lambda x_2.N)$. We conduct our reduction so that redexes of the first kind are contracted in this phase whereas the redexes in 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 7). Note that this makes us not to reduce some redexes but these redexes are certainly of order 2. 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.

Whereas there are no explicit redexes (except for the marked ones) we have some redexes hidden behind fan-nodes. We can extract these redexes as on Figure 2(a) and then contract them so that the resulting term is as on 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).

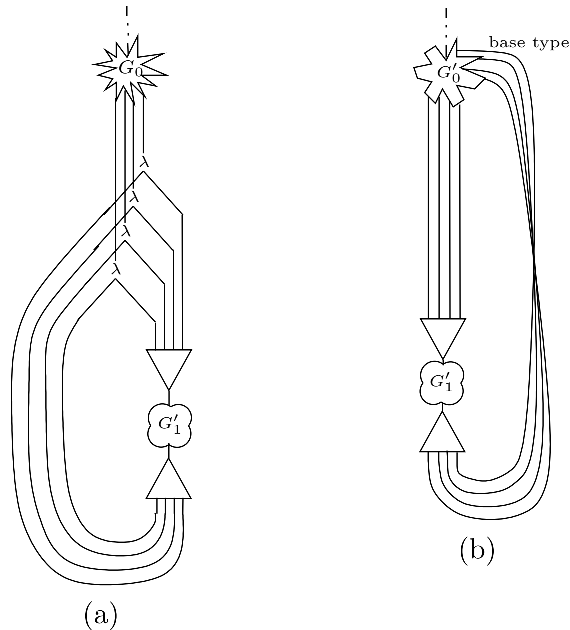

Figure 2: (a) The λ-nodes are extracted from the fan-nodes. (b) After the second-order reduction

Definition 7 `(the algorithm for 3rd order)`
Let M_1 and M_2 be the input data for the algorithm. The algorithm `reduce_3rd` proceeds performing the following steps:

1. Translate the term M_1 into the corresponding graph.
2. Perform one by one all the 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 an edge going out of a λ-node).
3. Push all the λ-nodes through fans.
4. Perform one by one all the existing β-reductions and if necessary go to the 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 else fail.

For the sake of clarity, we omit reductions for brackets and croissants in the description assuming that they are performed implicitly so that no such nodes occur at principal ports of fan-nodes, λ-nodes and @-nodes.

Theorem 4.1 `(partial correctness of reduce_3rd)`
If `reduce_3rd` *stops with success then* $\mathrm{NF}(M_1) \equiv_\alpha M_2$*. If* `reduce_3rd` *fails then* $\mathrm{NF}(M_1) \not\equiv_\alpha M_2$*.*
Proof:
The correctness of `reduce_3rd` is implied by the correctness of the graph reduction. The only thing that remains to be proved is that before entering the point (6) in Definition 7, 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 latter claim in two steps: First, we prove that after performing the step (2) there will be no 3rd-order redexes. Second, we prove that after performing the steps (3-4) there will be no 2nd-order redexes.

For the proof of the first step, we proceed by contradiction. Assume that we have a 3rd-order redex that may be performed at some reduction path starting from the result H_0 of the step (2). The residual of the λ-node that takes part in the reduction is present in H_0. The λ-node may not be reduced in H_0 either because it does not interact with @-node or it interacts with such a node via marked edge. The latter case is not possible since this would mean that we performed a 4th-order reduction step earlier. So λ-node interacts with: (1) another λ-node, (2) a @-node via non-principal port, (3) fan-node, (4) bracket, (5) croissant.

1. In this case, we have again the same cases as in the main proof. Thus, we may inductively continue in this case until some non-λ-node is reached. Note that the type of the next λ-node includes the type of the previous one.

2. This case is impossible since this would mean that at some point of the reduction, there occurs a redex at the principal port of @-node, but this redex would be of order greater than 3.

3. The λ node may interact with a fan-node at principal or at non-principal port. The non-principal port is impossible at this stage of reduction since we forbid in our algorithm fan-reductions at this stage. The λ-node at principal port of a fan-node means that the λ-node was an argument of some β-redex, but such a redex must have had the order 4.

4. Interaction with bracket is impossible since such a node is implicitly pushed either behind λ-node or before the preceding nodes.

5. The reasoning is similar to the one for bracket.

As no case is possible, we obtained contradiction.

For the proof of the claim that in the graph H_1 obtained after the point (4) there are no possible reductions with 2nd-order redexes, we proceed by contradiction too. Suppose that at some reduction path of H_1 we have a 2nd-order redex. The residual of the λ-node that takes part in the reduction is present in H_1. The λ-node may not be reduced in H_1 because it does not interact with @-node at principal port.

1. If λ-node interacts with another λ-node we may inductively apply our reasoning to the next λ-node.

2. If λ-node interacts with an @-node at non-principal port then the @ node should be reduced in the reduction so there is another λ-node that will be reduced we may inductively apply our reasoning to the node.

3. If λ-node interacts with a fan-node at its principal port then the point (3) may be applied so we are not after the (4) point. If λ-node interacts with a fan-node at its non-principal port then the type of the λ-node must be a base type (see Figure 2(b)), but this is impossible.

4. If λ-node interacts with a bracket- or croissant-node then these nodes may be pushed behind the λ-node or before the preceding nodes.

Again, no case is possible so we obtained contradiction. This ends the whole proof as 2-nd order redexes are the redexes of lowest possible order.

The algorithm fails only in points (6) and (7). $\mathrm{NF}(M_1) \not\equiv_\alpha M_2$ is obvious in these cases. □

In order to perform an analysis of the complexity of the algorithm `reduce_3rd`, we have to introduce the notion of *mixed bracket property.* This notion formalises and generalises the property of old-fashioned arithmetical notation where 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 A may be closed only if each parenthesis of any other kind B that is opened after the parenthesis of the kind A is closed. For example, if we open [and then afterwards (then in order to put] we have to put) first.

Definition 8 `(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 afterwards 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 4.2 `(mixed parenthesis property in third-order reduction)` *During the reductions performed in* `reduce_3rd` *all the paths have mixed parenthesis property.*

Proof:
The proof is by the number of steps of reduction during the algorithm `reduce_3rd`. Before the point (2) of Definition 7, our property is retained as there is no path that exits a fan-node.

Assume that our property is retained until n-th step. We show that it is retained in the next step too.

If the step is inside point (2) then the property is retained as there is no path that exits a fan-node.

If the step is inside point (3) then the property is retained as its violation would require bounding outside of the scope of λ.

If the step is inside point (4) then the move corresponds to concatenating well formed sequences of parenthesis. This does not violate our property.

If the step is inside point (5) then we only erase only matching inmost parenthesis which does not violate our property.

Points (6–7) do not change the graph so they retain our property.

Details are left for the reader. □

Theorem 4.3 `(reduce_3rd runs in polynomial time)`
The procedure `reduce_3rd` *runs in* $O(n^2)$ *time.*

Proof:
Let $n = |M_1| + |M_2|$. Performing the translation in the point (1) takes $O(n)$ time since the translation is local. Performing the point (2) takes also $O(n)$. This is so since there is a constant k that bounds the number of relinkings and implicit reductions that should be made in order to go from the situation on Figure 1(a) to the situation on Figure 1(b). Moreover, these operations need to be performed at most as many times as the number of 3rd order variables in the original term. This is so because graph reduction implements the optimal reduction. The points (3–4) need $O(n)$ time since we have to produce as many new λ-nodes as the number of occurrences of second-order variables and as many new fan-nodes as the number of third-order variables. Such a creation together with linking these new nodes needs a constant time so the time is linear. The point (5) needs also $O(n)$ time since the number of fan-nodes linearly depends on the size of the input. The point (6) needs $O(n^2)$ steps since the mixed parenthesis property (Fact 4.2) ensures that there are no two fan-nodes that meet with principal ports. If it were matching fan-nodes then they would be reduced in point (5), if it were non-matching fan-nodes then they would 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 the 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 Third order β-reduction is PTIME-hard

The problem of evaluation of boolean circuits is reduced to the problem of β-reduction in third order in this section. The reduction is in LOGSPACE. This implies that third-order β-reduction is PTIME-hard (see. [Pap95])

Definition 9 `(boolean circuit)`
A boolean circuit is a directed acyclic graph such that:

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

The result of a boolean circuit is defined recursively as follows

Definition 10 `(the result of a circuit)`
The *result of a boolean circuit* is defined as the value of its node labelled with `result`. The value of a node is defined recursively as follows

- the value of `true` is `true`;
- the value of `false` is `false`;
- the value of $\vee$ 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 $\wedge$ is $v_1 \wedge v_2$ 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 $\neg$ is $\neg v$ where v is the value of the node at the end of the outgoing edge;
- the value of `result` is v where v is the value of the node at the end of the outgoing edge.

Definition 11 `(the problem of evaluation of a boolean circuit)`
The problem of evaluation of a boolean circuit is the following:
Input: A boolean circuit $\mathcal{C}$
Question: Does the circuit have the result `true`?

We define *level* of a node in boolean circuit. This notion helps in defining our reduction.

Definition 12 `(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 define also logical values and quantifiers which are needed in forthcoming parts of the paper.

Definition 13 `(connectives for translations)`

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

Definition 14 `(reduction from boolean circuits)`
This reduction is recursively defined on the level of nodes. We introduce variables $\{x_i^j \mid i \text{ is a node on the level } j\}$.

- The term LEVEL_{-1} is defined as $x^0_{\texttt{result}}$.
- The term LEVEL_{n+1} is defined based on the term LEVEL_n as

  $$(\lambda x_1^{n+1} \ldots x_k^{n+1} . \mathrm{LEVEL}_n) B_1 \ldots B_k$$

  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;

- $B_i = \text{NOT} x^l_k$ 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;
- $B_i = \text{TRUE}$ if the i-th node on the level $n+1$ is `true`;
- $B_i = \text{FALSE}$ if the i-th node on the level $n+1$ is `false`.

Theorem 4.4 (`boolean circuits and` $\lambda_{\rightarrow}$)
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 3.*

Proof:
The proof is by induction on the maximal level of the graph G.

If the level is 0 then G consists only of two vertices: `result` and one of `true` or `false`. So, LEVEL_0 has either the form $(\lambda x^0_{\texttt{result}}.x^0_{\texttt{result}})\text{TRUE}$ or $(\lambda x^0_{\texttt{result}}.x^0_{\texttt{result}})\text{FALSE}$ respectively. These terms reduce to TRUE and FALSE respectively. This proves our claim in this case.

If the level is $n > 0$ then the term LEVEL_n has the form

$$(\lambda x^{n+1}_1 \dots x^{n+1}_k.\text{LEVEL}_n)B_1 \dots B_k$$

where B_i for $i = 1, \dots, k$ are either TRUE or FALSE. We reduce this term to

$$\text{LEVEL}_n[x^{n+1}_1 := B_1, \dots, x^{n+1}_k := B_k]$$

then we replace subterm of the form

- AND MTRUE with M;
- AND MFALSE with FALSE;
- OR MTRUE with TRUE;
- OR MFALSE with M;
- NOT FALSE with TRUE;
- NOT TRUE with FALSE

(we have omitted the cases symmetric wrt. the position of M). The modifications performed to the term LEVEL_n correspond directly to performing evaluation steps for one level of the graph G. The modified term evaluates to TRUE iff the original one evaluates to TRUE. This is true as there is only one normal form and our modifications may be performed as usual β-reduction. The modified graph gives the result `true` iff the original one does. This is true as a simple verification according Definition 10 reveals. The partially evaluated graph evaluates to `true` iff the modified term reduces to TRUE by induction hypothesis. This completes the proof. Details are left for the reader.

The redexes in LEVEL_n occur in subterms of the form: $\text{OPER}M_1M_2$ where OPER is AND or OR, $\text{NOT}M$, and $(\lambda\vec{x}.\text{LEVEL}_l)B_1\dots B_k$. The arguments for all those terms have the type $\alpha \to \alpha \to \alpha$ so these terms are of order 3. Thus the redexes in terms LEVEL_n are of order at most 3. □

Theorem 4.5 `(boolean circuits and` $\lambda_\to$`)`
The term LEVEL_n *may be generated with use of additional* $O(\log|G|)$ *space.*

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 AND, OR, NOT, 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 4

5.1 Fourth order β-reduction is in PSPACE

The fourth order reduction can be performed in polynomial space. Our algorithm, similarly to the third-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 on Figure 3(a). The figure pictures a β-redex somewhere in some term. The star denoted by G_0 symbolises the body of the λ-abstraction that takes part in β-reduction. The circle denoted by G_1 symbolises the body of the argument that takes part in β-reduction. The dotted lines represent parts of the term that are missing on the picture.

The result of the first β-reduction step is shown on Figure 3(b). As we see, the argument G_1 goes into several places of the subterm G_0 similarly to the 3rd order case. This kind of reduction is performed during the first phase of our algorithm. Again, performing some β-redexes may introduce other ones. Again, we perform only some of the new redexes similarly to the 3rd-order case. We repeat this kind of reduction until there are no redexes. The result of the process is a term that has no 4th-order redexes.

Whereas there are no explicit redexes (except for the marked ones) we have some redexes hidden behind fan-nodes. We can extract these redexes as on Figure 4(a) and then contract them with @-nodes that come from G_0 as depicted on 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 on Figure 5(a). We have two fan-nodes surrounding G_1' — the upper one because the term occurs in

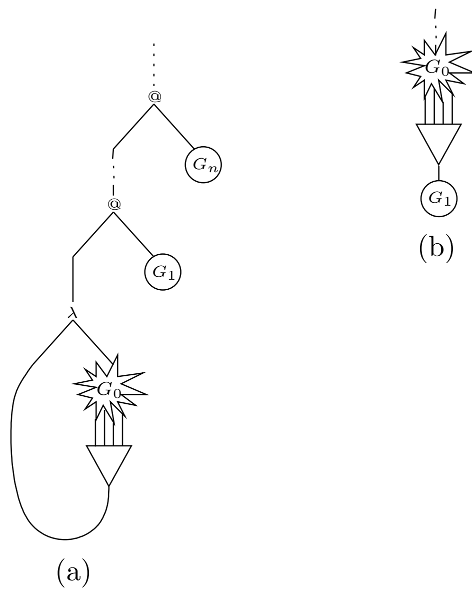

Figure 3: (a) The starting point for 4th-order β-reduction. (b) The result of the first phase of β-reduction

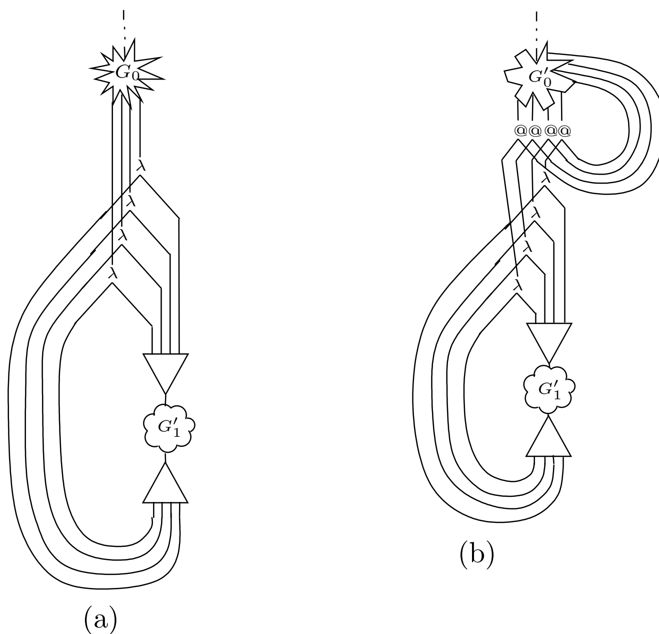

Figure 4: (a) The λ-nodes are extracted from fan-nodes. (b) The λ-nodes meet suitable @-nodes

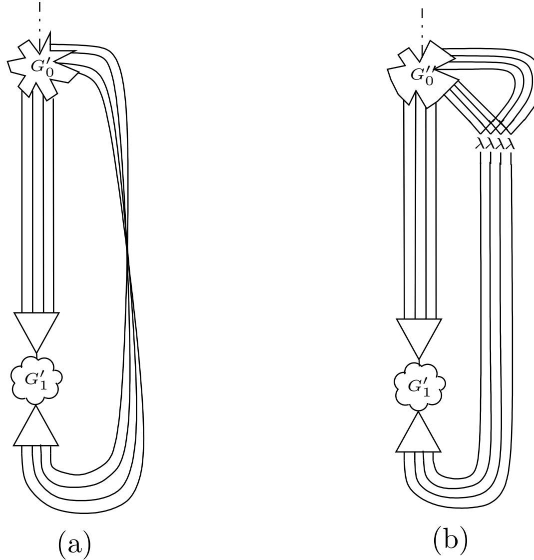

Figure 5: (a) The result of the second phase of reduction. (b) Second-order λ-nodes begin to reduce

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 3rd-order redexes).

The last phase of the reduction begins — the reduction of 2nd-order redexes. These redexes occur as on Figure 5(b) and begin to interact with the graph G'_1. As the 2nd-order variable that took part in the 3rd-order reduction (the lambdas of which were multiplied on Figure 4(a)) can occur in several places inside G'_1, several @-nodes will take part in the reduction of 2nd-order redexes. We can see these @-nodes on Figure 6(a). As this multiplication concerns only one variable, we have a fan-node that performs this operation — also visible on Figure 6(a).

The fan-nodes that meet begin to interact. The result of the interaction is depicted on 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 on Figure 7. The next step to perform is to push @-nodes through fan-nodes. The result of performing such a step is partially shown on 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 on 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 15 `(the algorithm for 4th order)`

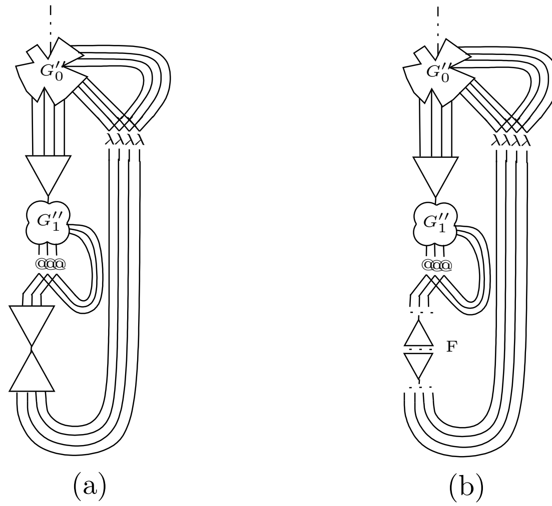

Figure 6: (a) Multiple occurrences of 2nd-order variables with surrounding @-nodes. (b) Fan-nodes interact

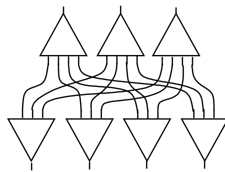

Figure 7: The interaction of fan-nodes

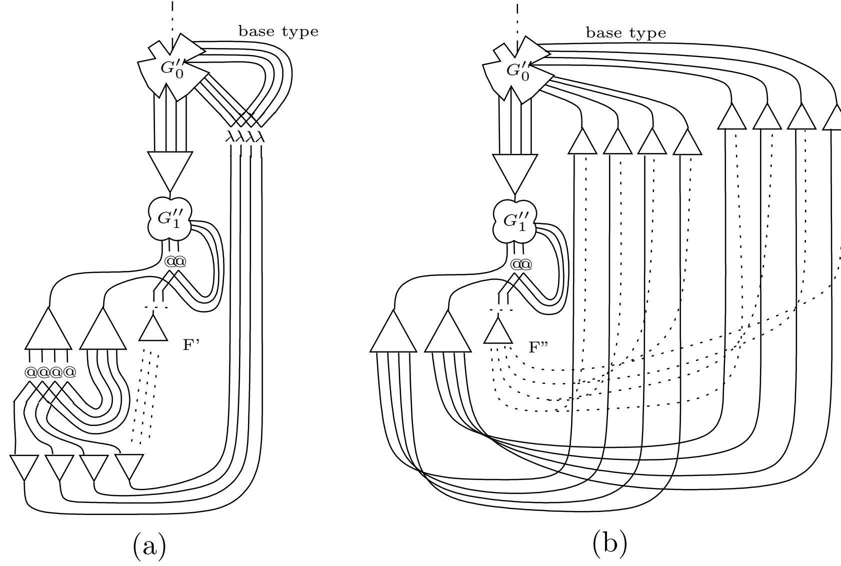

Figure 8: (a) One application goes between fan-nodes. (b) After pushing λ-nodes through fans, applications are reduced

Let M_1 and M_2 be the input data for the algorithm. The algorithm `reduce_4th` proceeds performing the following steps:

1. Translate the term M_1 into corresponding graph.
2. Perform one by one all the 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 an edge going out of a λ-node).
3. Clear all the markings.
4. Push all the fans through λ-nodes.
5. Perform one by one all the existing β-reductions (again with marking).
6. Perform all the interactions between fans and afterwards push all the fans through λ- and @-nodes.
7. Perform one by one all the 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 else fail.

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

Definition 16 `(level of a redex)`
Let us define a special kind of reduction where

$$(\lambda x.M)N \rightarrow_{\beta'} M[x := N^*]$$

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

$$(\lambda x.M)^* N \rightarrow_{\beta} M[x := N^*].$$

Of course, all the 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. Also the notion of level of a redex straightforwardly translates to graph reduction. The algorithm for 4th order reduction needs redexes of order at most 2. The redexes of level 0 are reduced in the step (2) of the algorithm, the redexes of level 1 are reduced in the step (5) of the algorithm and at last redexes of level 2 are reduced in the step (7) of the algorithm.

Theorem 5.1 `(the algorithm reduce_4th is in PSPACE)`
Let M_1 have redexes of order at most 4 and M_2 be in normal form. The algorithm `reduce_4th` *results in success on these terms iff $M_1 \rightarrow_{\beta}^{*} M_2$.*

Moreover, `reduce_4th` *needs only $O(n^3)$ space 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_4th`.

The point (1) is the matter of usual syntax analysis and may be performed in $O(n)$ time and thus in $O(n)$ space.

The point (2) does not add any new nodes to the reduced graph and requires a simple walk through the graph so may be performed in $O(n)$ time and thus in $O(n)$ space (see Figure 3).

The point (3) is again linear since it requires a walk through the graph in hand while the size of the graph is linear.

The point (4) requires multiplication of λ-nodes and fan-nodes. This multiplication is performed as on 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 point (5) 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 $O(n^3)$.

The point (6) (see Figure 5(b), Figure 6, and Figure 8) requires a copying of fan-nodes, then a copying of @-nodes, λ-nodes and fan-nodes. As the result of the first operation of copying, we obtain at most $k_3 \cdot k_4$ fan-nodes where k_3 is the number of redexes of level 0 and k_4 is the number of places where a second-order symbol occurs. Both these numbers may be bounded by n so the final number of nodes is $O(n^2)$. The @- and λ-nodes occur at most the same number of times so again the number is majorised by $O(n^2)$. At the same time the fan-nodes duplicate so we obtain $O(n^2)$ nodes.

The 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 $O(n^3)$.

The point (8) gives the size $O(n^3)$ as it cannot produce an output which is greater than n and it must walk through the graph in hand.

The point (9) needs $O(n)$ space.

There are of course also bracket- and croissant-nodes. The number of them is closely connected to the number of fan-nodes so we do not mention analysis of complexity for them.

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

5.2 Fourth order β-reduction is PSPACE-hard

We present a PTIME reduction of the QBF problem to the 4th order reduction problem.

The translation is defined as follows

Definition 17 `(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 $M_1 = \texttt{Q2L}(\phi)$ and $M_2 = \text{TRUE}$. The function `Q2L` is defined by induction on the form of the QBF formula:

- $\texttt{Q2L}(\texttt{true}) = \text{TRUE}$;

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

Theorem 5.2 (QBF and $\lambda_\rightarrow$)
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 4.

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 $v : A \to \{\mathtt{true}, \mathtt{false}\}$ 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 $\mathrm{AND}, \mathrm{OR}, \mathrm{NOT}, \mathbf{\forall}, \mathbf{\exists}$. The type for $\mathrm{AND}, \mathrm{OR}$ and NOT is of order 3. These terms take as arguments values of the type $\alpha \to \alpha \to \alpha$ (which is the type of booleans TRUE and FALSE). The type for $\mathbf{\forall}$ and $\mathbf{\exists}$ is more complicated and is of order 4. 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. □

Theorem 5.3 (the running time for Q2L)
The translation from QBF to $\lambda_\rightarrow$ can be performed in $O(n \log n)$.

Proof:
Parsing of QBF formulas using usual syntax analysis methods can be performed in $O(n)$ time. The result of the parsing is a pointer structure representing the formula. This pointer structure may be traversed recursively in order to obtain the term defined by Q2L. The recursive traversal needs to visit each node of the pointer structure once and we need $\log n$ storage for recursive step. Thus the traversal may be performed in $O(n)$. At last we add the TRUE term in constant time, so the whole procedure runs in $O(n \log n)$ time. □

6 Acknowledgements

I would like to thank Damian Niwiński for his hints concerning good problems to reduce from.

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.