Electronic Colloquium on Computational Complexity, Report No. 110 (2007)

# The Optimal Read-Once Branching Program Complexity for the Direct Storage Access Function

Beate Bollig

FB Informatik, LS2, Univ. Dortmund, 44221 Dortmund, Germany
beate.bollig@uni-dortmund.de

**Abstract.** Branching programs are computation models measuring the space of (Turing machine) computations. Read-once branching programs (BP1s) are the most general model where each graph-theoretical path is the computation path for some input. Exponential lower bounds on the size of read-once branching programs are known since a long time. Nevertheless, there are only few functions where the BP1 size is asymptotically known exactly. In this paper, the exact BP1 size of a fundamental function, the direct storage access function, is determined.

**Keywords:** Computational complexity, lower bounds, read-once branching programs

## 1 Introduction and Result

Branching programs are a model of computation measuring the space of (Turing machine) computations. Since the proof of non-polynomial lower bounds on the branching program complexity of so-called explicitly defined Boolean functions is out of scope of known lower bound methods, several restricted models have been studied. Read-once branching programs are the most general model where each graph-theoretical path is the computation path for some input. Besides the complexity theoretical viewpoint restricted (read-once) branching programs are used as data structure for boolean functions in several applications (see e.g. Wegener (2000)), where the complexity of fundamental functions is of interest.

**Definition 1.** *A branching program* (BP) *on* $X_n := \{x_1, \ldots, x_n\}$ *is a directed acyclic graph* $G = (V, E)$ *whose sinks are labeled by Boolean constants and whose non sink* (*or inner*) *nodes are labeled by Boolean variables from* $X_n$ (*see Figure 1*)*. Each inner node has two outgoing edges one labeled by* 0 *and the other by* 1*. Each node* $v$ *represents a Boolean function* $f_v : \{0,1\}^n \to \{0,1\}$ *defined in the following way. In order to evaluate* $f_v(b)$*,* $b \in \{0,1\}^n$*, start at* $v$*. After reaching an* $x_i$*-node choose*

ISSN 1433-8092

*the outgoing edge with label $b_i$ until a sink is reached. The label of this sink defines $f_v(b)$. The branching program is called read-once if each path contains for each variable $x_i$ at most one node labeled by $x_i$. The size of a (read-once) branching program is equal to the number of its nodes. The (read-once) branching program complexity of a function $f$, denoted by* $\mathrm{BP}(f)$ ($\mathrm{BP1}(f)$)*, is the minimal size of a (read-once) branching program representing $f$.*

It is an obvious aim to determine $\mathrm{BP1}(f)$ for as many of the interesting functions $f$ as exactly as possible. This is similar to other fundamental complexity measures, among them circuit size, formula size, monotone circuit size or algebraic complexity (for such results see Wegener (1987)). Although many even exponential lower bounds on the BP1 complexity of fundamental Boolean functions are known (see e.g. Bollig and Woelfel (2001)), there are only few functions where the BP1 complexity is known exactly (see e.g. Wegener (1984)). To the best of our knowledge, we start to fill this gap by presenting the first exact bound on the BP1 complexity for a nonsymmetric function, namely the direct storage access function $\mathrm{DSA}_n$, often also called multiplexer $\mathrm{MUX}_n$.

**Definition 2.** *The direct storage access function* $\mathrm{DSA}_n$ *(or multiplexer* $\mathrm{MUX}_n$*) is defined on $n+k$ variables $a_{k-1},\ldots,a_0,x_0,\ldots,x_{n-1}$, where $n=2^k$. The a-variables are called address variables and the x-variables data variables.* $\mathrm{DSA}_n(a,x)=x_{|a|}$*, where $|a|$ is the number whose binary representation equals $(a_{k-1},\ldots,a_0)$.*

The result of the paper is the following one.

**Theorem 1.** $\mathrm{BP1}(\mathrm{DSA}_n)=2n+1$.

The upper bound is contained in Wegener (2000) (Theorem 4.3.2) and it is even shown that the direct storage access function can be represented by restricted read-once branching programs, called OBDDs, in size $2n+1$. Only recently, Bollig, Range, and Wegener (2007) have pesented the matching lower bound on the OBDD size. In Section 2, we improve their result by presenting the lower bound on the complexity of the more general BP1s. To obtain the optimal bound, we have to count the number of nodes very carefully but we do not introduce a new lower bound method. Since Bollig, Range, and Wegener (2007) have made use of the fact that in an OBDD on all paths from the source to the sinks the variables have to be tested according to a given order of the variables, we have to use new arguments to prove the lower bound for BP1s.

## 2 The BP1 Complexity of the Direct Storage Access Function

In this section, we determine a lower bound on the size of BP1s representing the direct storage access function. Figure 1 shows a restricted BP1 for the direct storage access function, where $n = 4$.

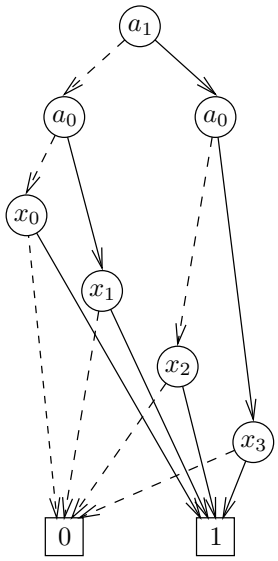


**Fig. 1.** *A BP1 for DSA$_4$ (dotted edges are edges with label 0 and solid edges are edges with label 1).*

**Lemma 1.** *The size of a* BP1 *for the representation of the direct storage access function is at least* $2n + 1$.

**Proof.**

Let $G$ be a BP1 of minimal size representing the direct storage access function. Since $\mathrm{DSA}_n$ depends essentially on all data variables, for each variable $x_i$, $0 \leq i \leq n-1$, there is at least one node labeled by $x_i$. Moreover, there have to be two sinks. In the following, we prove that there exists at least $2^k - 1$ further nodes representing non-constant subfunctions of the direct storage access function, such that the number of nodes altogether in the BP1 is at least $2 + n + 2^k - 1 = 2n + 1$.

First, we transform $G$ into a BP1 $G'$ representing $\mathrm{DSA}_n$ where on each path from the source to a sink each variable is tested exactly once

(for the transformation see e.g. Wegener (2000), Proof of Lemma 6.2.2). We only use $G'$ to define certain sets of paths and the corresponding subfunctions of $\mathrm{DSA}_n$ for which we will show that a certain number of nodes is necessary in $G$.

Now, we define sets $P_i$, $1 \leq i \leq k$, of paths in $G'$ in the following way. All paths in $P_i$ start at the source, each node labeled by a data variable is left via the 0-edge. A path in $P_i$ contains exactly $i-1$ nodes labeled by an address variable and ends at a node labeled by an address variable. Furthermore, for two paths $p$ and $q$ in $P_i$ there exists an address variable $a$ such that $p$ contains a node labeled by $a$ and left via the 0-edge and $q$ contains a node labeled by $a$ left via the 1-edge or vice versa. The paths in $P_i$, $2 \leq i \leq k$, are extensions of the paths in $P_{i-1}$. Each path in $P_{i-1}$ that reach a node $w$ labeled by an address variable is extended by leaving $w$ via the 0-edge and via the 1-edge. Our aim is to prove that there exists at least one further node for each path in $\mathcal{P} := \cup_{1 \leq i \leq k} P_i$. Obviously, a set $P_i$ contains $2^{i-1}$ paths, $1 \leq i \leq k$, and $|\mathcal{P}| = 2^k - 1$.

For two different paths in $\mathcal{P}$ there are only two possibilities:

- one path is an extension of the other path or
- there exists an address variable $a$ such that one path contains a node labeled by $a$ and left via the 0-edge and the other one contains a node labeled by $a$ left via the 1-edge.

We use the following notation. Let $p$ be a path in $P_i$, $1 \leq i \leq k$, and $a_{j_1}, \ldots, a_{j_{i-1}}$ the address variables tested on $p$, i.e., $a_{j_1}$ is the first address variable tested on $p$, $a_{j_2}$ the second one, and so on. The path $p$ ends at a node labeled by the address variable $a_{j_i}$. Furthermore, let $b_p$ be the assignment of the address variables $a_{j_1}, \ldots, a_{j_{i-1}}$ according to $p$. The group $G_p$ contains all data variables $x_j$ such that the assignment of $a_{j_1}, \ldots, a_{j_{i-1}}$ in the binary representation of $j$ equals $b_p$. $R_p$ is the set of the $x$-variables for which there does not exist a node on $p$ labeled by one of these $x$-variables.

The following two observations will be helpful.

- $G_p \cap G_q = \emptyset$ for two different paths $p$ and $q$ in $\mathcal{P}$ where $p$ is not an extension of the path $q$ and vice versa.
- A subfunction corresponding to a path $p$ essentially depends on the data variables in $G_p \cap R_p$ and does not essentially depend on all other data variables.

For each path $p$ in $\mathcal{P}$ we distinguish two cases. The first case is similar to part of the lower bound proof presented by Bollig, Range, and Wegener

(2007). The second one is more complicated and different from the OBDD lower bound proof for $DSA_n$.

**Case 1:** $G_p \cap R_p \neq \emptyset$.

Let $p$ end at a node labeled by the address variable $a$ in $G'$. We show that there also exists a node in $G$ labeled by the address variable $a$ that represents the subfunction corresponding to $p$.

Let $x_j \in G_p \cap R_p$. Obviously, the subfunction corresponding to $p$ essentially depends on $x_j$. Since $G_p \cap G_q = \emptyset$ for two different paths $p$ and $q$ where $p$ is not an extension of the path $q$ and vice versa, the subfunction corresponding to $q$ does not depend essentially on $x_j$ and has to be represented at a different node in $G$. Furthermore, the subfunction corresponding to $p$ essentially depends on the address variable $a$, since the assignment 1 to $x_j$, 0 to all other data variables, and the binary representation of $j$ to the address variables has the function value 1 but changing only the assignment of the address variable $a$ leads to the function value 0.

Altogether, we have shown that there has to be one further node labeled by the addres variable $a$ in $G$.

**Case 2:** $G_p \cap R_p = \emptyset$.

This case is more difficult. The reason is the following one. Let $p$ end at a node labeled by the address variable $a$ in $G'$. Unlike Case 1, it is possible that the subfunction corresponding to $p$ does not essentially depend on the address variable $a$. We have to inspect this case very carefully in order to guarantee that we count each node of the BP1 $G$ representing the direct storage access function only once.

Let $p'$ be the shortest shortening of $p$ such that $p'$ ends at a node labeled by an address variable $a_{j_\ell}$ and $G_{p'} \cap R_{p'} = \emptyset$. Clearly, there are $2^{\log n-\ell+1} - 1$ extensions of $p'$ (including $p'$) in $\cup_{\ell \leq i \leq k} P_i$. We prove that in $G$ there is at least one further node labeled by a data variable for each extension of $p'$. Since $G_{p'} \cap R_{p'} = \emptyset$ and $|G_{p'}| = 2^{\log n-\ell+1}$, we know that there are at least $2^{\log n-\ell+1}$ nodes on $p'$ labeled by one of the data variables in $G_{p'}$. Let $x_{i_1}, \ldots, x_{i_{2^{\log n-\ell+1}}}$ be data variables in $G_{p'}$. Now, we prove that there are at least $2 \cdot 2^{\log n-\ell+1} - 1$ nodes labeled by one of these data variables in $G$. Considering the fact that we already have counted one node for each data variable, we can conclude that there are at least $2 \cdot 2^{\log n-\ell+1} - 1 - 2^{\log n-\ell+1} = 2^{\log n-\ell+1} - 1$ further nodes in $G$.

Let $p'_{i_j}$ be the shortening of $p'$ that ends at the node labeled by $x_{i_j}$, $1 \leq j \leq 2^{\log n-\ell+1}$. Since the subfunction of $DSA_n$ corresponding to $p'_{i_j}$ essentially depends on $x_{i_j}$ there has to be a node in $G$ labeled by $x_{i_j}$, where the subfunction corresponding to $p'_{i_j}$ is represented. Next, we consider

the path $p'_{i_1}$ and extend this path by leaving the node labeled by $x_{i_1}$ via the 1-edge. The corresponding subfunction $f$ essentially depends on the variables $x_{i_j}$, $2 \leq j \leq 2^{\log n-\ell+1}$. Therefore, there has to be a node $v$ in $G$ where $f$ is represented and in the sub-BP1 $G_f$ that consists of all nodes in $G$ reachable from $v$ there has to be at least one node for each data variable $x_{i_j}$, $2 \leq j \leq 2^{\log n-\ell+1}$. It remains to prove that these nodes are different from the nodes representing the subfunctions corresponding to the paths $p'_{i_j}$, $2 \leq j \leq 2^{\log n-\ell+1}$. The subfunction corresponding to $p'_{i_j}$ essentially depends on all address variables not tested on $p'_{i_j}$. To see this, we consider the following assignment to the remaining variables (not tested on $p'_{i_j}$). The data variable $x_{i_j}$ is set to 1, the remaining data variables to 0. The address variables are set to the binary representation of $i_j$ (this is possible since $x_{i_j} \in G_{p'_{i_j}}$). Obviously, the function value for this assignment is 1 but changing only the assignment of one address variable (not tested on $p'_{i_j}$) leads to an assignment with function value 0, since all other data variables except $x_{i_j}$ are set to 0. In order to show that the $x_{i_j}$-node in $G$ representing the subfunction corresponding to $p'_{i_j}$ is different from the $x_{i_j}$-node in the sub-BP1 $G_f$, we consider the following assignment $b$ to the variables not tested on $p'_{i_j}$. The data variables are set to 0 and the remaining address variables are set to the binary representation of $i_1$ (this is possible since $x_{i_1} \in G_{p'_{i_j}}$). Obviously, the function value of the subfunction corresponding to $p'_{i_j}$ is 0. Now, there are only two possibilities. Either the subfunction represented at the considered $x_{i_j}$-node in the sub-BP1 $G_f$ does not essentially depend on one of the address variables not tested on $p'_{i_j}$, or the value of the subfunction represented at the $x_{i_j}$-node in $G_f$ for the assignment $b$ is 1. In both cases the considered $x_{i_j}$-nodes have to be different.

Since $G_p \cap G_q = \emptyset$ for two paths $p$ and $q$ in $\mathcal{P}$, where $p$ is not an extension of $q$ and vice versa, we count different nodes labeled by a data variable for $p$ and $q$.

Altogether, we have shown that there exists at least one further node for each path in $\mathcal{P}$. Therefore, there are at least $2 + n + 2^k - 1 = 2n - 1$ nodes in $G$.

□

## References


1. Bollig, B., Range, N., and Wegener, I. (2007). Exact OBDD bounds for some fundamental functions. ECCC TR07-049.

2. Bollig, B. and Woelfel, P. (2001). A read-once branching program lower bound of $\Omega(2^{n/4})$ for integer multiplication using universal hashing. Proc. of 33rd STOC, 419–424.
3. Wegener, I. (1984). Optimal decision trees and one-time-only branching-programs for symmetric Boolean functions. Information and Control 62, 129–143.
4. Wegener, I. (1987). *The Complexity of Boolean Functions*. Wiley-Teubner.
5. Wegener, I. (2000). *Branching Programs and Binary Decision Diagrams - Theory and Applications*. SIAM Monographs on Discrete Mathematics and Applications.



ECCC ISSN 1433-8092

http://eccc.hpi-web.de/