Electronic Colloquium on Computational Complexity, Report No. 166 (2015)

# A better-than-3$n$ lower bound for the circuit complexity of an explicit function

Magnus Gausdal Find[1], Alexander Golovnev[2], Edward A. Hirsch[3], and Alexander S. Kulikov[3]

[1]National Institute of Standards and Technology
[2]New York University
[3]St. Petersburg Department of Steklov Institute of Mathematics of the Russian Academy of Sciences

October 17, 2015

**Abstract**

We consider Boolean circuits over the full binary basis. We prove a $(3+\frac{1}{86})n - o(n)$ lower bound on the size of such a circuit for an explicitly defined predicate, namely an affine disperser for sublinear dimension. This improves the $3n - o(n)$ bound of Norbert Blum (1984). The proof is based on the gate elimination technique extended with the following three ideas. We generalize the computational model by allowing circuits to contain cycles, this in turn allows us to perform affine substitutions. We use a carefully chosen circuit complexity measure to track the progress of the gate elimination process. Finally, we use quadratic substitutions that may be viewed as delayed affine substitutions.



ISSN 1433-8092

# Contents



# 1 Introduction

In this paper we consider Boolean circuits over the full binary basis, that is, directed acyclic graphs where each internal node computes a binary Boolean operation $\{0,1\}\times\{0,1\}\to\{0,1\}$, inputs are fed into nodes of indegree zero, and one node (or the negation of a node) is designated as the output. The *size* of a circuit is the number of its internal nodes. A simple counting argument [Sha49] shows that most Boolean functions require circuits of exponential size. However, showing superpolynomial lower bounds for explicitly defined functions (for example, for functions from **NP**) remains a hopelessly difficult task. (In particular, such lower bounds would imply $\mathbf{P}\neq\mathbf{NP}$.) Moreover, even superlinear bounds are unknown for functions in $\mathbf{E}^{\mathbf{NP}}$. The smallest uniform complexity class for which superpolynomial bounds are known is **MAEXP** (exponential-time Merlin-Arthur games) [BFT98], and the smallest such class with arbitrary polynomial lower bounds is $\mathbf{O_2}$ (the oblivious symmetric second level of the polynomial hierarchy) [CR06].

People started to tackle the problem in the 60s. Kloss and Malyshev [KM65] proved a $2n-O(1)$ lower bound for the function $\bigoplus_{1\le i<j\le n} x_ix_j$. Schnorr [Sch74] proved a $2n-O(1)$ lower bound for functions that for any pair of variables $x,y$, have at least three different subfunctions among the four functions obtained after substituting constants to $x$ and $y$. Stockmeyer [Sto77] proved a $2.5n-O(1)$ bound for certain symmetric functions. Paul [Pau77] proved a $2n-o(n)$ lower bound for the storage access function and a $2.5n-o(n)$ lower bound for a function combining several storage access functions using simple operations. Eventually, Blum [Blu84] extended Paul's argument and proved a $3n-o(n)$ bound.

Mysteriously, Blum's bound remained unbeaten for more than thirty years. Recently, Demenkov and Kulikov [DK11] showed a similar bound for affine dispersers. Their proof is

much simpler (if one assumes the existence of explicitly defined dispersers for granted), but that does not improve the bound (actually, the $o(n)$ term in [DK11] is worse than that in [Blu84]). In this paper we eventually improve the bound from [DK11] to $(3 + \frac{1}{86})n - o(n)$, which is stronger than Blum's bound.

**Other models.** The exact complexity of computational problems is different in different models of computation: for example, switching from multitape to single-tape Turing machines squares the time complexity; random access machines are even more efficient. For example, the quadratic lower bound for recognizing palindromes by a single-tape Turing machine [HU69] is worthless for stronger computational models. Boolean circuits over the full binary basis make a very robust computational model. Using a different constant-arity basis only changes the constants in the complexity. A fixed set of gates of arbitrary arity (for example, ANDs, ORs and XORs) still preserves the complexity in terms of the number of wires. After all, finding a function hard for Boolean circuits can be viewed as a combinatorial problem, in a contrast to lower bounds for uniform models. Therefore breaking the linear barrier for Boolean circuits can be viewed as an important milestone on the way to stronger complexity lower bounds.

In this paper we consider single-output circuits (that is, circuits computing predicates). It would be natural if allowing many outputs would lead us to non-linear bounds. However, the only tool we have to transfer bounds from one output to several outputs is Lamagna and Savage [LS73] argument showing that in order to compute simultaneously $m$ different functions requiring $c$ gates each, one needs at least $m + c - 1$ gates. That is, we do not have superlinear bounds for multioutput functions either.

For the basis $U_2$ consisting of all binary Boolean functions except for parity (xor) and its complement, Schnorr [Sch76] proved that the circuit complexity of the parity function is $3n - 3$. Zwick [Zwi91] gave a $4n - O(1)$ lower bound for certain symmetric functions, Lachish and Raz [LR01] showed a $4.5n - o(n)$ lower bound for a strongly two-dependent function (a function that has exactly four subfunctions with respect to any two variables and remains so after sufficiently many substitutions). Iwama and Morizumi [IM02] improved this bound to $5n - o(n)$. Demenkov et al. [DKMM15] gave a simpler proof of a $5n - o(n)$ lower bound for a function with $o(n)$ outputs as well as presented a $7n - o(n)$ lower bound for a function with $n$ outputs.

While we do not have nonlinear bounds for constant-arity Boolean circuits, exponential bounds are known for weaker models: one thread was initiated by Razborov [Raz85] for monotone circuits, another one was started by Yao and Håstad for constant-depth circuits without XORs [Yao85, Hås86]. Shoup and Smolensky [SS91] proved a superlinear lower bound $\Omega(n \log n / \log \log n)$ for linear circuits of polylogarithmic depth over infinite fields. Also, superlinear bounds for formulas are known for half a century. For de Morgan formulas (i.e., formulas over AND, OR, NOT) Subbotovskaya [Sub61] proved an $\Omega(n^{1.5})$ lower bound for the parity function using the random restrictions method. Khrapchenko [Khr71] showed an $\Omega(n^2)$ lower bound for parity. Applying Subbotovskaya's random restrictions method to the universal function by Nechiporuk [Nec66], Andreev [And87] proved an $\Omega(n^{2.5-o(1)})$ lower bound. By analyzing how de Morgan formulas shrink under random restrictions, Andreev's lower

bound was improved to $\Omega(n^{2.55-o(1)})$ by Impagliazzo and Nisan [IN93], then to $\Omega(n^{2.63-o(1)})$ by Paterson and Zwick [PZ93], and eventually to $\Omega(n^{3-o(1)})$ by Håstad [Hås98] and Tal [Tal14]. For formulas over the full binary basis, Nechiporuk [Nec66] proved an $\Omega(n^{2-o(1)})$ lower bound for the universal function and for the element distinctness function. These bounds, however, do not translate to superlinear lower bounds for general constant-arity Boolean circuits.

**Connections to CircuitSAT algorithms.** A recent promising direction initiated by Williams [Wil13] connects the complexity of circuits to the complexity of algorithms for CircuitSAT (this is the problem of checking whether a given circuit has a satisfying assignment, that is, a substitution of inputs by constants that forces the circuit to output one). Namely, the existence of better-than-$2^n$ algorithms for CircuitSAT for a particular circuit model implies exponential lower bounds for these circuits for functions in large classes like **NEXP**. This way unconditional exponential lower bounds have been proved for $\mathrm{ACC}_0$ circuits (constant-depth circuits with unbounded-arity OR, AND, NOT, and arbitrary modular gates) [Wil14]. Ben-Sasson and Viola [BV14] have demonstrated that in order to prove a specific linear lower bound for a function in $\mathbf{E}^{\mathbf{NP}}$ it suffices to lower the base of the exponent in the 3-SAT complexity down to an appropriate constant.

It should be noted, however, that currently available algorithms for the satisfiability problem for general circuit classes are not sufficient for proving new lower bounds. Current techniques require upper bounds of the form $O(2^n/n^a)$ for circuits with $n$ inputs and size $n^k$, while for most classes only $c^g$-time algorithms are available, where $g$ is the number of the gates and $c > 1$ is a constant.

On the other hand, the techniques used in the $c^g$-time algorithms for CircuitSAT are somewhat similar to the techniques used for proving linear lower bounds for (general) Boolean circuits over the full binary basis. In particular, an $O(2^{0.4058g})$-time algorithm by Nurk [Nur09] (and subsequently an $O(2^{0.389667g})$-time algorithm by Savinov [Sav14]) use reconstruction of the linear part of a circuit similar to the one suggested by Paul [Pau77]. These algorithms and proofs use similar tricks in order to simplify circuits; however, at present no rigorous statement is known that would connect these two complexities. The only cases where certain types of algorithms for general complexity classes yield linear lower bounds for them are average-case results for formulas [San10, ST13, KRT13, CKK+15] and circuits [CK15], which are somewhat weaker than the current worst-case bounds.

**Our methods.** Almost all previous lower bounds have been proved using a simple gate elimination technique: one gradually simplifies the function (for example, by substituting variables one by one) showing that every simplification step eliminates a certain number of gates. A crucial idea [Sch74] is to keep the function in the same class. Following [DK11], we prove lower bounds for affine dispersers, that is, functions that are non-constant on affine subspaces of certain dimensions: Ben-Sasson and Kopparty [BK12] gave an explicit construction of affine dispersers for sublinear dimensions.

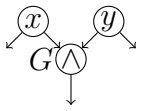
Feeding an appropriate constant to a non-linear gate (for example, AND) makes this gate constant and therefore eliminates subsequent gates, which helps to eliminate more gates than

in the case of a linear gate (for example, XOR). On the other hand, linear gates, when stacked together, sometimes allow to reorganize the circuit. Then affine substitutions can kill such gates while keeping the properties of an affine disperser. Such linear reconstructions were used for proving circuit lower bounds by Paul [Pau77], Stockmeyer [Sto77], and Blum [Blu84]. Seto and Tamaki [ST13] used it to prove upper bounds for satisfiability of formulas over the full binary basis. Demenkov and Kulikov [DK11] used affine substitutions to prove a circuit lower bound for affine dispersers.

Thus, it is natural to consider a circuit as composed of linear circuits connected by non-linear gates. In our case analysis it is important that we make affine substitutions but not restrictions. That is, instead of just saying that $x_1 \oplus x_2 \oplus x_3 \oplus x_9 = 0$ and removing all gates that become constant, we make sure to replace all occurrences of $x_1$ by $x_2 \oplus x_3 \oplus x_9$. Since a gate computing such a sum might be unavailable and we do not want to increase the number of gates, we "rewire" some parts of the circuit, which, however, may potentially introduce cycles. This leads us to the first ingredient of our proof: *cyclic circuits*. That is, the linear components of our "circuits" may now have directed cycles; however, we require that the values computed in the gates are still uniquely determined (which is actually a requirement on the rank of the corresponding linear system). Cyclic circuits were studied, e.g., by Rivest [Riv77], Dymond and Cook [DC89], Nickelsen, Tantau, and Weizsäcker [NTW04], Riedel and Bruck [RB12] (the last reference also contains an overview of previous work on cyclic circuits).

Thus we are able to make affine substitutions. We try to make such a substitution in order to make the topmost (i.e., closest to the inputs) non-linear gate constant. This, however, does not seem to be enough. The second ingredient in our proof is a *complexity measure* that manages difficult situations (bottlenecks) by allowing to perform an amortized analysis: we count not just the number of gates, we compute a linear combination of the number of gates and the number of bottlenecks. Such measures were previously considered by several authors. For example, Zwick [Zwi91] counted the number of (internal) gates minus the number of inputs of outdegree 1. The same measure was later used by Lachish and Raz [LR01] and by Iwama and Morizumi [IM02]. Kojevnikov and Kulikov [KK10] used a measure assigning different weights to linear and non-linear gates to show that Schnorr's $2n - O(1)$ lower bound [Sch76] can be strengthened to $7n/3 - O(1)$. Carefully chosen complexity measures are also used to estimate the progress of splitting algorithms for **NP**-hard problems [Kul99, FGK09].

Our main bottleneck (called "troubled gate") is as follows:




Here all gates have outdegrees exactly as shown on the picture, i.e., two inputs of degree 2 feed an and-type gate of outdegree 1.

Sometimes in order to fight a troubled gate, we have to make a *quadratic substitution*, which is the third ingredient of our proof. This happens if the gate below $G$ is a linear gate fed by a variable $z$; in the simplest case a substitution $z = xy$ kills $G$, the linear gate, and

the gate below (actually, we show it kills much more). However, quadratic substitutions may make affine dispersers constant, so we consider a special type of quadratic substitutions. Namely, we consider quadratic substitutions as a form of delayed affine substitutions (in the example above, if we promise to substitute later a constant either to $x$ or $y$, the substitution can be considered affine). In order to maintain this, instead of affine subspaces (where affine dispersers are non-constant by definition) we consider so-called read-once depth-2 quadratic sources (essentially, this means that all variables in the right-hand sides of the quadratic substitutions that we make are pairwise distinct free variables). We show that an affine disperser for a sublinear dimension remains non-constant for read-once depth-2 quadratic sources of a sublinear dimension.

**Open questions.** An affine disperser for dimension $d$ may be viewed as a function that is not constant on any affine subspace of *size* at least $2^d$. A natural extension is to allow similarly sized varieties defined by quadratic polynomials. As shown by Golovnev and Kulikov [GK15], such dispersers with appropriate parameters must have circuit size at least $3.1n$. However, explicit constructions of such dispersers are currently unknown. There are known constructions of dispersers for algebraic varieties for large finite fields [Dvi12], and known constructions of such dispersers for $\mathbb{F}_2$ [CT15, Sha11] but with weaker parameters than needed for the lower bound to work.

# 2 Definitions

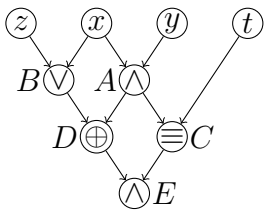
**Gates and notation** A circuit is an acyclic directed graph in which incoming edges are numbered for every node. The nodes are called *gates*. A gate may have either indegree zero (in which case it is called an *input* gate, or a *variable*) or indegree two (in which case it is called an *internal* gate). Every internal gate is labelled by a Boolean function $g\colon \{0,1\} \times \{0,1\} \to \{0,1\}$, and the set of all the sixteen such functions is denoted by $B_2$. We call these binary functions *operations* in order to distinguish them from functions of $n$ variables computed in the gates. The size of a circuit is the number of internal gates.

We say that an operation is of *and-type* if it computes $g(x,y) = (c_1 \oplus x)(c_2 \oplus y) \oplus c_3$ for some constants $c_1, c_2, c_3 \in \{0,1\}$, and of *xor-type* if it computes $g(x,y) = x \oplus y \oplus c_1$ for some constant $c_1 \in \{0,1\}$. Similarly, we call gates and-type and xor-type. If a gate computes an operation depending on precisely one of its inputs, we call it *passing*.

If an (internal) gate computes a constant operation, we call it *trivial* (note that it still has two incoming edges). If a substitution forces some gate $G$ to compute a constant, we say that it *trivializes* $G$. (For example, for a gate computing the operation $g(x,y) = x \wedge y$, the substitution $x = 0$ trivializes it.)

We denote by $out(G)$ the outdegree of the gate $G$. If $out(G) = k$, we call $G$ a $k$-gate. If $out(G) \geq k$, we call it a $k^+$-gate. We adopt the same terminology for variables (so we have 0-variables, 1-variables, $2^+$-variables, etc.).

One gate of outdegree zero is designated as the output.




$B = (z \vee x)$
$A = (x \wedge y)$
$D = (B \oplus A)$
$C = (A \equiv t)$
$E = (D \wedge C)$

Figure 1: An example of a circuit and the program it computes.

A toy example of a circuit is shown in Figure 1. For input gates, the corresponding variables are shown inside. For an internal gate, we show its operation inside and its label near the gate. As figure shows, a circuit corresponds to a simple program for computing a Boolean function: each instruction of the program is a binary Boolean operation whose inputs are input variables or the results of the previous instructions.

**Affine dispersers** An *affine disperser* for dimension $d(n)$ is a family of functions $f_n \colon \mathbb{F}_2^n \to \mathbb{F}_2$ such that for all sufficiently large $n$, $f_n$ is non-constant on any affine subspace of dimension at least $d(n)$. Explicit constructions of affine dispersers have drawn a lot of attention recently. First, polynomial-time computable affine dispersers for any linear dimension were constructed [BKS$^+$10, Bou07], and then it was shown that there are polynomial-time computable affine dispersers for sublinear dimensions $d(n) = o(n)$ [BK12, Yeh11, Li11, Sha11].

## 2.1 Generalizations of circuits

**Cyclic circuits** In this paper we use generalizations of circuits that simplify circuit transformations. These generalized circuits may contain cycles; however, the underlying graphs are still not arbitrary labelled directed graphs.

A *cyclic circuit* is a directed (not necessarily acyclic) graph where all vertices have indegree either 0 or 2. We adopt the same terminology for its nodes (input and internal gates) and its size as for ordinary circuits. We restrict our attention to *cyclic xor-circuits*, where all gates compute affine operations. While the most interesting internal gates compute either $\oplus$ or $\equiv$, for technical reasons we also allow passing gates and trivial gates. We will be interested in multioutput cyclic circuits, so, in contrast to our definition of ordinary circuits, several gates may be designated as outputs, and they may have nonzero outdegree.

A circuit, and even a cyclic circuit, naturally corresponds to a system of equations over $\mathbb{F}_2$, where internal gates correspond to variables of the system and variables of the (cyclic) circuit are counted in the constants of the system (that is, we formally have a separate system for every assignment to the input gates, but all these systems share the same matrix). For a gate $G$ fed by gates $F$ and $H$ and computing some operation $\odot$, we write the equation $G \oplus (F \odot H) = 0$. A more specific clarifying example would be a gate $G$ computing $F \oplus x \oplus 1$, where $x$ is an input gate; then the line in the system would be $G \oplus F = x \oplus 1$, where $G$ and $F$ contribute two 1's to the matrix, and $x \oplus 1$ contributes to the constant vector.

For a cyclic xor-circuit, this is a linear system with a square matrix. We call a cyclic xor-circuit *fair* if this matrix has full rank. It follows that for every assignment of the inputs, there exist *unique* values for the gates such that these values are consistent with the circuit (that is, for each gate its value is correctly computed from the values in its inputs). Thus, similarly to an ordinary circuit, every gate in a fair circuit computes a function of the values fed into its input gates (clearly, it is an affine function). A simple example of a fair cyclic xor-circuit is shown in Figure 2. Note that if we additionally impose the requirement that the graph is acyclic, we arrive at ordinary linear circuits (that is, circuits consisting of xor-type gates, passing gates, and constant gates).

**Relationship between cyclic and acyclic xor-circuits.** It is not difficult to show that for multiple outputs, fair cyclic xor-circuits form a stronger model than acyclic xor-circuits. For example, the 9 functions computed simultaneously by the cyclic xor-circuit shown in Figure 2 cannot be computed by an acyclic xor-circuit with 9 gates. To see this, assume for the sake of contradiction, that an acyclic xor-circuit with 9 gates computes the same functions. Since the circuit has 9 gates all gates must compute outputs. Consider a topologically minimal gate $G$. Such a gate exists since the circuit is acyclic. Since $G$ is topologically minimal it computes the sum of two input gates, therefore it cannot compute an output.

On the other hand, a minimal xor-circuit of $k$ variables computing a single output has exactly $k-1$ internal gates and is acyclic.

**Semicircuits.** We introduce the following notion, called *semicircuits*, a generalization of both Boolean circuits and cyclic xor-circuits.

A semicircuit is a composition of a cyclic xor-circuit and an (ordinary) circuit. Namely, its nodes are split into two sets, $X$ and $C$. The nodes in the set $X$ form a cyclic xor-circuit. The nodes in the set $C$ form an ordinary circuit (if wires going from $X$ to $C$ are replaced by variables). There are no wires going back from $C$ to $X$. A semicircuit is called fair if $X$ is fair. ***In what follows we abuse the notation by using the word "circuit" to mean a fair semicircuit.***

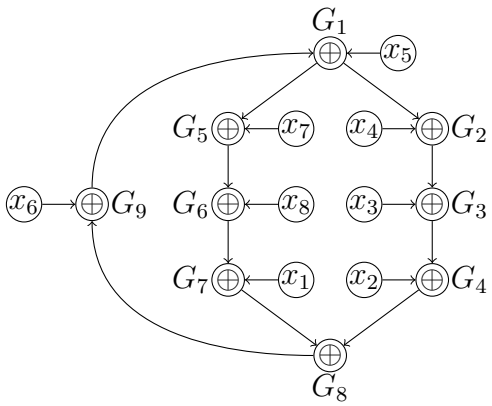
# 3 Lower bound

## 3.1 Overview

In this section we prove the main theorem.

The proof goes by induction. We start with an affine disperser and a circuit computing it on $\{0,1\}^n$. Then we gradually shrink the space where it is computed by building smaller and smaller rdq-sources. While we add more equations to the source, we simplify the circuit by reducing the number of gates (and other parameters counted by the complexity measure) and eliminating the variable we have just substituted.

In Subsection 3.2 we show how to make substitutions in fair semicircuits, and how to simplify them afterwards. We also introduce "troubled" gates, special "unwanted" fragments




$$
\begin{aligned}
G_1 &= x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8 \\
G_2 &= x_1 \oplus x_2 \oplus x_3 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8 \\
G_3 &= x_1 \oplus x_2 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8 \\
G_4 &= x_1 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8 \\
G_5 &= x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_8 \\
G_6 &= x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6 \\
G_7 &= x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6 \\
G_8 &= x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_7 \oplus x_8 \\
G_9 &= x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_6 \oplus x_7 \oplus x_8
\end{aligned}
$$

$$
\begin{aligned}
G_1 &= G_9 \oplus x_5 \\
G_2 &= G_1 \oplus x_4 \\
G_3 &= G_2 \oplus x_3 \\
G_4 &= G_3 \oplus x_2 \\
G_5 &= G_1 \oplus x_7 \\
G_6 &= G_5 \oplus x_8 \\
G_7 &= G_6 \oplus x_1 \\
G_8 &= G_4 \oplus G_7 \\
G_9 &= G_8 \oplus x_6
\end{aligned}
\qquad
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1
\end{bmatrix}
\times
\begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \\ G_5 \\ G_6 \\ G_7 \\ G_8 \\ G_9 \end{bmatrix}
=
\begin{bmatrix} x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_7 \\ x_8 \\ x_1 \\ 0 \\ x_6 \end{bmatrix}
$$

Figure 2: A simple example of a cyclic xor-circuit. In this case all the gates are labeled with $\oplus$. The affine functions computed by the gates are shown to the right of the circuit. The bottom row shows the program computed by the circuit as well as the corresponding linear system.

of circuits that we count in the complexity measure. We check that the simplification methods we use do not increase the number of these gates too much.

In order to eliminate troubled gates, sometimes we use quadratic substitutions. In Subsection 3.3 we describe formally subsets of points of $\{0,1\}^n$ resulting from all kinds of the substitutions we make and show that affine dispersers are non-constant on such subsets. In Subsection 3.4 we define the circuit complexity measure and formulate the main result: we can always reduce the measure by an appropriate amount by extending our rdq-source; the lower bound follows.

Finally, Subsection 3.5 proves the main result by going through a number of cases.

## 3.2 Cyclic circuit transformations

### 3.2.1 Basic substitutions

In this section we consider several types of substitutions. It is straightforward how to substitute a constant to an input:

**Proposition 1.** *Let $C$ be a circuit with input gates $x_1, \dots, x_n$, and let $c \in \{0,1\}$ be a constant. For every gate $G$ fed by $x_1$ replace the operation $g(x_1, t)$ computed by $G$ with the operation $g'(x_1, t) = g(c, t)$ (thus the result becomes independent of $x_1$). This transforms $C$ into another circuit $C'$ (in particular, it is still a fair semicircuit) such that it has the same number of gates, the same topology, and for every gate $H$ that computes a function $h(x_1, \dots, x_n)$ in $C$, the corresponding gate in the new circuit $C'$ computes the function $h(c, x_2, \dots, x_n)$.*

We call this transformation a *substitution by a constant*.

A more complicated type of a substitution is when we replace an input $x$ with a function computed in a different gate $G$. In this case in each gate fed by $x$, we replace wires going from $x$ by wires going from $G$.

We call this transformation a *substitution by a function*.

**Proposition 2.** *Let $C$ be a circuit with input gates $x_1, \dots, x_n$, and let $g(x_2, \dots, x_n)$ be a function computed in a gate $G$. Consider the construction $C'$ obtained by substituting a function $g$ to $x_1$ (it has the same number of gates as $C$). Then if $G$ is not reachable from $x_1$ by a directed path in $C$, then $C'$ is a fair semicircuit, and for every gate $H$ that computes a function $h(x_1, \dots, x_n)$ in $C$, except for $x_1$, the corresponding gate in the new circuit $C'$ computes the function $h(g(x_2, \dots, x_n), x_2, \dots, x_n)$.*

*Proof.* Note that we require that $G$ is not reachable from $x_1$ (thus we do not introduce new cycles), and also that $g$ does not depend on $x_1$. Functions computed in the gates are the solution of the system corresponding to the circuit (see Section 2.1). The transformation simply replaces every equation of the form $H = F \odot x_1$ with the equation $H = F \odot G$ (and equation of the form $H' = x_1 \odot x_1$ with the equation $H' = G \odot G$). Consider specific values for $x_2, \dots, x_n$. Assume that the solution for the old system does not satisfy the new equation. Then take $x_1 = g(x_2, \dots, x_n)$, it violates the corresponding equation in the old system, a contradiction. Vice versa, consider a different solution for the new system. It must satisfy the old system (where $x_1 = g(x_2, \dots, x_n)$), but the old system has a unique solution. □

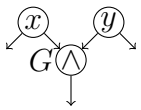
In what follows, however, we will also use substitutions that do not satisfy the hypothesis of this proposition: substitutions that create cycles. We defer this construction to Section 3.2.3.

### 3.2.2 Normalization and troubled gates

In order to work with a circuit, we are going to assume that it is "normalized", that is, it does not contain obvious inefficiencies (such as trivial gates, etc.), in particular, those created by substitutions. We describe certain normalization rules below; however, while normalizing we need to make sure the circuit remains within certain limits: in particular, it must remain fair and compute the same function. We need to check also that we do not "spoil" a circuit by introducing "bottleneck" cases. Namely, we are going to prove an upper bound on the number of newly introduced unwanted fragments called "troubled" gates.

We say that an internal gate $G$ is *troubled* if it satisfies the following three criteria:

- $G$ is an and-type gate of fanout 1,
- the gates feeding $G$ are input gates,
- both input gates feeding $G$ have fanout 2.

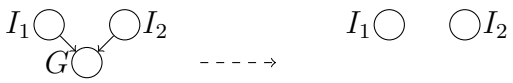



*(From now on, we denote all and-type gates by $\wedge$, and all xor-type gates by $\oplus$.)*

We always make substitutions consciously and thus can count the number of troubled gates that can possible emerge. However, what if a gate is killed because of simplifications? We limit the process of removing gates to normalization rules, and make sure that we never get more than four new troubled gates per killed gate.

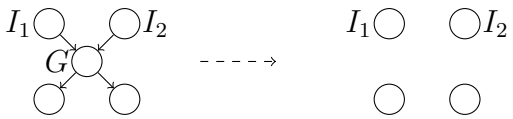
We say that a circuit is *normalized* if none of the following rules is applicable to it. Each rule eliminates a gate $G$ whose inputs are gates $I_1$ and $I_2$. (Note that $I_1$ and $I_2$ can be inputs or internal nodes, and, in rare cases, they can coincide with $G$ itself.)

**Rule 1:** If $G$ has no outgoing edges and is not marked as an output, then remove it.




Note also that it could not happen that the only outgoing edge of $G$ feeds itself, because this would make a trivial equation and violate the circuit fairness.

**Rule 2:** If $G$ is trivial, i.e., it computes a constant operation $c$, remove $G$ and "embed" this constant to the next gates. That is, for every gate $H$ fed by $G$, replace the operation $h(g,t)$ computed in this gate (where $g$ is the input from $G$ and $t$ is the other input) by the operation $h'(g,t) = h(c,t)$. (Clearly, $h'$ depends on at most one argument, which is not optimal, and in this case after removing $G$ one typically applies Rule 3 or Rule 2 to its successors.)




**Rule 3:** If $G$ is passing, i.e., it computes an operation depending only on one of its inputs, remove $G$ by reattaching its outgoing wires to that input. This may also require changing the operations computed at its successors (the corresponding input may be negated; note that an and-gate (xor-gate) remains an and-gate (xor-gate)).

If $G$ feeds itself and depends on another input, then the self-loop wire (which would now go nowhere) is dropped. (Note that if $G$ feeds itself it cannot depend on the self-loop input.)

If $G$ has no outgoing edges it must be an output gate (otherwise it would be removed by Rule 0). In this special case, we remove $G$ and mark the corresponding input of $G$ (or its negation) as the output gate.


**Rule 4:** If $G$ is a 1-gate that feeds a single gate $Q$, $Q$ is distinct from $G$ itself, and $Q$ is also fed by one of $G$'s inputs, then replace in $Q$ the incoming wire going from $G$ by a wire going from the other input of $G$ (this might also require changing the operation at $Q$); then remove $G$. We call such a gate $G$ *useless*.


**Rule 5:** If the inputs of $G$ coincide ($I_1$ and $I_2$ refer to the same node) then we replace the binary operation $g(x, y)$ computed in $G$ with the operation $g'(x, y) = g(x, x)$. Then perform the same operation on $G$ as described in Rule 3 or 2.

**Proposition 3.** *Each of the Rules 1–5 removes one internal gate, introduces at most four new troubled gates. An input gate that was not connected by a directed path to the output gate cannot be connected by a new directed path*[1]*. None of the rules change the functions of n input variables computed in the gates that are not removed. A fair semicircuit remains a fair semicircuit.*

*Proof. Fairness.* The circuit remains fair since no rule changes the set of solutions of the system.

*New troubled gates.* For all the rules, the only gates that may become troubled are $I_1$, $I_2$ (if they are and-type gates), and the gates they feed after the transformation (if $I_1$ or $I_2$ is a variable). Each of $I_1$, $I_2$ may create at most two new troubled gates. Hence each rule, when applied, introduces at most four new troubled gates. □

[1]This trivial observation will be formally needed when we later count the number of such gates.

### 3.2.3 Affine substitutions

In this subsection, we show how to make substitutions that do create cycles. This will be needed in order to make affine substitutions. Namely, we take a gate computing an affine function $x_1 \oplus \bigoplus_{i \in I} x_i \oplus c$ (where $c \in \{0,1\}$ is a constant) and "rewire" a circuit so that this gate is replaced by a trivial gate computing a constant $b \in \{0,1\}$, while $x_1$ is replaced by an internal gate. The resulting circuit over $x_2, \ldots, x_n$ may be viewed as the initial circuit under the substitution $x_1 \leftarrow \bigoplus_{i \in I} x_i \oplus c \oplus b$. The "rewiring" is formally explained below; however, before that we need to prove a structural lemma (which is trivial for acyclic circuits) that guarantees its success.

For an xor-circuit, we say that a gate $G$ depends on a variable $x$ if $G$ computes an affine function in which $x$ is a term. Note that in a circuit without cycles this means that precisely one of the inputs of $G$ depends on $x$, and one could trace this dependency all the way to $x$, therefore there always exists a path from $x$ to $G$. In the following lemma we show that it is always possible to find such a path in a fair cyclic circuit too. However, it may be possible that some nodes on this path do not depend on $x$. Note that dependencies in cyclic circuits are sometimes counterintuitive. For example, in Figure 2, gate $G_4$ is fed by $x_2$ but does not depend on it.

**Lemma 1.** *Let $C$ be a fair cyclic xor-circuit, and let the gate $G$ depend on the variable $x$. Then there is a path from $x$ to $G$.*

*Proof.* Let us substitute all variables in $C$ except for $x$ to 0. Since $G$ depends on $x$, it can only compute $x$ or its negation.

Let $\mathcal{R}$ be the set of internal gates that are reachable from $x$, and $\mathcal{U}$ be the set of internal gates that are not reachable from $x$. Let us enumerate the gates in such a way that gates from $\mathcal{U}$ have smaller indices than gates from $\mathcal{R}$. Then the circuit $C$ corresponds to the system

$$\begin{bmatrix} U & 0 \\ R_1 & R_2 \end{bmatrix} \times \mathcal{G} = \begin{bmatrix} L_U \\ L_R \end{bmatrix},$$

where $\mathcal{G} = (g_1, \ldots, g_{|C|})^T$ is a vector of unknowns (the gates' values), $U$ is the principal submatrix corresponding to $\mathcal{U}$ (a square submatrix whose rows and columns correspond to the gates from $\mathcal{U}$). Note that

- the upper right part of the matrix is 0, because there are no wires going from $\mathcal{R}$ to $\mathcal{U}$, and thus unknowns corresponding to gates from $\mathcal{R}$ do not appear in the equations corresponding to gates from $\mathcal{U}$,
- $L_U$ is a vector of constants, it cannot contain $x$ since $\mathcal{U}$ is not reachable from $x$,
- $L_R$ is a vector of affine functions of $x$, since all other inputs are substituted by zeros.

If $U$ is singular, then the whole matrix is singular, which contradicts the fairness of $C$. Therefore, $U$ is nonsingular, i.e., the values $\mathcal{G}' = (g_1, \ldots, g_{|\mathcal{U}|})^T$ are uniquely determined by

$U \times \mathcal{G}' = L_U$, and they are constant (independent of $x$). This means that $G$ cannot belong to $\mathcal{U}$.

□

We now come to rewiring.

**Lemma 2.** *Let $C$ be a fair semicircuit with input gates $x_1, \ldots, x_n$ and internal gates $G_1, \ldots, G_m$. Let $G$ be a gate not reachable by a directed path from any and-gate. Assume that $G$ computes the function $x_1 \oplus \bigoplus_{i \in I} x_i \oplus c$, where $I \subseteq \{2, \ldots, n\}$. Let $b \in \{0, 1\}$ be a constant. Then one can transform $C$ into a new circuit $C'$ with the following properties:*

1. *graph-theoretically, $C'$ has the same gates as $C$, plus a new internal gate $Z$; some edges are changed, in particular, $x_1$ is disconnected from the circuit;*
2. *the operation in $G$ is replaced by the constant operation $b$;*
3. $\mathrm{in}_{C'}(Z) = 2$, $\mathrm{out}_{C'}(G) = \mathrm{out}_C(G) + 1$, $\mathrm{out}_{C'}(x_1) = 0$. $\mathrm{out}_{C'}(Z) = \mathrm{out}_C(x_1) - 1$.
4. *The indegrees and outdegrees of all other gates are the same in $C$ and $C'$.*
5. *$C'$ is fair.*
6. *all gates common for $C'$ and $C$ compute the same functions on the affine subspace defined by $x_1 \oplus \bigoplus_{i \in I} x_i \oplus c \oplus b = 0$, that is, if $f(x_1, \ldots, x_n)$ is the function computed by an internal gate in $C$ and $f'(x_2, \ldots, x_n)$ is the function computed by its counterpart in $C'$, then $f(\bigoplus_{i \in I} x_i \oplus c \oplus b, x_2, \ldots, x_n) = f'(x_2, \ldots, x_n)$. The gate $Z$ computes the function $\bigoplus_{i \in I} x_i \oplus c \oplus b$ (which on the affine subspace equals $x_1$).*

*Proof.* Consider a path from $x_1$ to $G$ that is guaranteed to exist by Lemma 1. Denote the internal gates on this path by $G_1, \ldots, G_k = G$. Denote by $T_1, \ldots, T_k$ the other inputs of these gates. Note that we assume that $G_1, \ldots, G_k$ are pairwise different gates while some of the gates $T_1, \ldots, T_k$ may coincide with each other and with some of $G_1, \ldots, G_k$ (it might even be the case that $T_i = G_i$).

The transformation is as shown in Figure 3. The gates $A_0, \ldots, A_k$ are shown on the picture just for convenience: any of $x_1, Z, G_1, \ldots, G_k$ may feed any number of gates, not just one $A_i$.

To show the fairness of $C'$, assume the contrary, that is, the sum of a subset of rows of the new matrix is zero. The row corresponding to $G_k = b$ must belong to the sum (otherwise we would have only rows of the matrix for $C$, plus an extra column). However, this would mean that if we sum up the corresponding lines of the system (not just the matrix) for $C$, we get $G_k = \mathrm{const} \oplus \bigoplus_{j \in J} x_j$ where $J \not\ni 1$ (note that $x_1$ was replaced by $Z$ in the new system, and cancelled out by our assumption). This contradicts the assumption of the Lemma that $G_k$ computes the function $x_1 \oplus \bigoplus_{i \in I} x_i \oplus c$. Therefore, the matrix for $C'$ has full rank.

The programs shown next to the circuits explain that for $x_1 = \bigoplus_{i \in I} x_i \oplus c \oplus b$, the gates $G_1, \ldots, G_k$ compute the same values in $C'$ and $C$; the value of $Z$ is also clearly correct. □

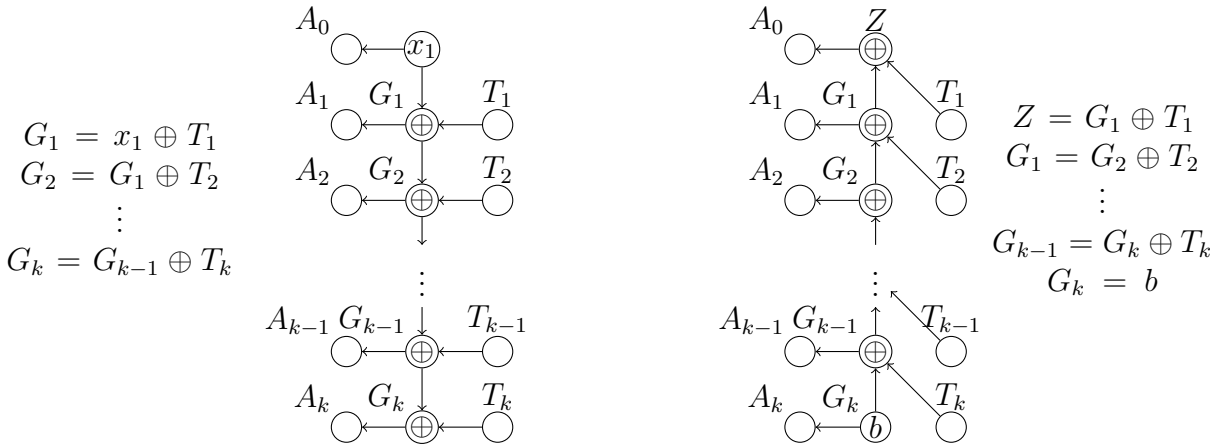



Figure 3: This figure illustrates the transformation from Lemma 2. We use $\oplus$ as a generic label for xor-type gates. That is, in the picture, gates labelled $\oplus$ may compute the function $\equiv$.

**Corollary 1.** *This transformation does not introduce new troubled gates.*

*Proof.* Indeed, the gates being fed by $G_1, \ldots, G_{k-1}, G_k, Z$ are not fed by variables; these gates themselves are not and-type gates; other gates do not change their degrees or types of input gates. □

After we apply the transformation, we apply Rule 2 to $G$. Since the only troubled gates introduced by this rule are the inputs of the removed gate, no troubled gates are introduced (and one gate, $G$ itself, is eliminated, thus the combination of Lemma 2 and Rule 2 does not increase the number of gates).

## 3.3 Read-once depth-2 quadratic sources

We generalize affine sources as follows.

**Definition 1.** *Let the set of variables $\{x_1, \ldots, x_n\}$ be partitioned into three disjoint sets $F, L, Q \subseteq \{1, \ldots, n\}$ (for* free*,* linear*, and* quadratic*). Consider a system of equalities that contains*

- *for each variable $x_j$ with $j \in Q$, a quadratic equality of the form*

$$x_j = (x_i \oplus c_i)(x_k \oplus c_k) \oplus c_j \,,$$

  *where $i, k \in F$; the variables from the right-hand side of all the quadratic substitutions are pairwise disjoint;*

- *for each variable $x_j$ with $j \in L$, an affine equality of the form*

$$x_j = \bigoplus_{i \in F_j \subseteq F} x_i \oplus \bigoplus_{i \in Q_j \subseteq Q} x_i \oplus c_j \,.$$

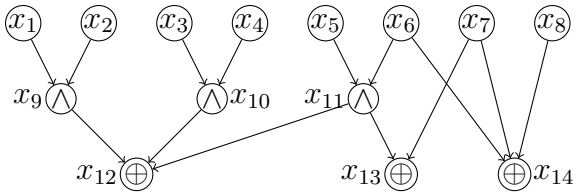



Figure 4: An example of an rdq-source. Note that a variable can be read just once by an and-type gate while it can be read many times by xor-gates.

*A subset $R$ of $\{(x_1, x_2, \ldots, x_n) \in \mathbb{F}_2^n\}$ that satisfies these equalities is called a* read-once depth-2 quadratic source *(or* rdq-source*) of dimension $d = |F|$.*

An example of such a system is presented at Figure 4.

The variables from the right-hand side of quadratic substitutions are called *protected*. Other free variables are called *unprotected*.

For this, we will gradually build a straight-line program (that is, a sequence of lines of the form $x = f(\ldots)$, where $f$ is a function depending on the inputs (free variables) and the values computed in the previous lines) that produces an rdq-source. We build it bottom-up. Namely, we will take an unprotected free variable and extend our current program with either a quadratic substitution

$$x_j = (x_i \oplus c_i)(x_k \oplus c_k) \oplus c_j\,,$$

depending on free unprotected variables $x_i, x_k$ or a linear substitution

$$x_j = \bigoplus_{i \in J} x_i \oplus c_j$$

depending on any variables. It is clear that such a program can be rewritten into a system satisfying Definition 1. In general, we cannot use protected free variables without breaking the rdq-property. However, there are two special cases where this is possible: (1) we can substitute a constant to a protected variable (and update the quadratic line accordingly: for example, $z = xy$ and $x = 1$ yield $z = y$ and $x = 1$); (2) we can substite one protected variable for another variable (or its negation) from the same quadratic equation (for example, $z = xy$ and $x = y$ yield $z = x$ and $x = y$).

In what follows we abuse the notation by denoting by the same letter $R$ the source, the straight-line program defining it, and a mapping $R\colon \mathbb{F}_2^d \to \mathbb{F}_2^n$ computed by this program that takes the $d$ free variables and evaluates all other variables.

**Definition 2.** *Let $R \subseteq \mathbb{F}_2^n$ be an rdq-source of dimension $d$, let the free variables be $x_1, x_2, \ldots, x_d$, and let $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ be a function. Then $f$ restricted to $R$, denoted $f|_R$, is a function $f|_R\colon \mathbb{F}_2^d \to \mathbb{F}_2$, defined by $f|_R(x_1, \ldots, x_d) = f(R(x_1, \ldots, x_d))$.*

Note that affine sources are precisely rdq-sources with $Q = \emptyset$. We define dispersers for rdq-sources similarly to dispersers for affine sources.

**Definition 3.** *An* rdq-disperser *for dimension $d(n)$ is a family of functions $f_n\colon \mathbb{F}_2^n \to \mathbb{F}_2$ such that for all sufficiently large $n$, for every rdq-source $R$ of dimension at least $d(n)$, $f_n|_R$ is non-constant.*

The following proposition shows that affine dispersers are also rdq-dispersers for related parameters.

**Proposition 4.** *Let $R$ be an rdq-source of $\mathbb{F}_2^n$ of dimension $d$. Then $R$ contains an affine subspace of dimension at least $d/2$.*

*Proof.* For each quadratic substitution $x_j = (x_i \oplus c_i)(x_k \oplus c_k) \oplus c_j$, further restrict $R$ by setting $x_i = 0$. This replaces a quadratic substitution by two affine substitutions $x_i = 0$ and $x_j = c_i(x_k \oplus c_k) \oplus c_j$; the number of free variables is decreased by one. Also, since the free variables do not occur on the left-hand side, the newly introduced affine substitution is consistent with the previous affine substitutions.

Since the variables occurring on the right-hand side of our quadratic substitutions are disjoint we have initially that $2|Q| \leq |F| = d$, so the number of newly introduced affine substitutions is at most $d/2$. □

Note that it is important in the proof that protected variables do not appear on the left-hand sides. The proposition above is obviously false for quadratic varieties: no Boolean function can be non-constant on all sets of common roots of $n - o(n)$ quadratic polynomials. For example, the system of $n/2$ quadratic equations $x_1x_2 = x_3x_4 = \ldots = x_{n-1}x_n = 1$ defines a single point, so any function is constant on this set.

**Corollary 2.** *An affine disperser for sublinear dimension is also an rdq-disperser for sublinear dimension.*

## 3.4 Circuit complexity measure

For a circuit $C$ and a straight-line program $R$ defining an rdq-source, we define the following circuit complexity measure:

$$\mu(C, R) = g + \alpha_Q \cdot q + \alpha_T \cdot t + \alpha_I \cdot i\,,$$

where $g$ is the number of internal gates in $C$, $q$ is the number of quadratic substitutions in $R$, $t$ is the number of troubled gates in $C$, and $i$ is the number of *influential* input gates in $C$. We say that an input is influential if it feeds at least one gate or is protected (recall that a variable is protected if it occurs in the right-hand side of a quadratic substitution in $R$). The constants $\alpha_Q, \alpha_T, \alpha_I > 0$ will be chosen later.

Proposition 3 implies that when a gate is removed from a circuit by applying a normalization rule the measure $\mu$ is reduced by at least $\beta = 1 - 4\alpha_T$. The constant $\alpha_T$ will be chosen to be very close to 0 (certainly less than $1/4$), so $\beta > 0$.

In order to estimate the initial value of our measure, we need the following lemma.

**Lemma 3.** *Let $C$ be a circuit computing an affine disperser $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ for dimension $d$, then the number of troubled gates in $C$ is less than $\frac{n}{2} + \frac{5d}{2}$.*

*Proof.* Let $V$ be the set of the inputs, $|V| = n$. In what follows we let $\sqcup$ denote the disjoint set union. Let us call two inputs $x$ and $y$ neighbors if they feed the same troubled gate. Assume to the contrary that $t \geq \frac{n}{2} + \frac{5d}{2}$. Let $v_i$ be the number of variables feeding exactly $i$ troubled gates. Since a variable feeding a troubled gate must have outdegree 2, $v_i = 0$ for $i > 2$. By double counting the number of wires from inputs to troubled gates, $2t = v_1 + 2v_2$. Since $v_1 + v_2 \leq n$,

$$n + 5d \leq 2t = v_1 + 2v_2 \leq n + v_2.$$

Let $T$ be the set of inputs that feed two troubled gates, $|T| = v_2 \geq 5d$. We now construct two disjoint subsets $X \subset T$ and $Y \subset V$ such that

- $|X| = d$,
- there are $|Y|$ consistent linear equations that make the circuit $C$ independent of variables from $X \sqcup Y$.

When the sets $X$ and $Y$ are constructed the theorem statement follows immediately. Indeed, we first take $|Y|$ equations that make $C$ independent of $X \sqcup Y$, then we set all the remaining variables $V \setminus (X \sqcup Y)$ to arbitrary constants. After this, the circuit $C$ evaluates a constant (since it does not depend on variables from $X \sqcup Y$ and all other variables are set to constants). We have $|Y| + |V \setminus (X \sqcup Y)| = |V \setminus X| = n - d$ linear equations which contradicts the assumption that $f$ is an affine disperser for dimension $d$.

Now we turn to constructing $X$ and $Y$. For this we will repeat the following routine $d$ times. First we pick any variable $x \in T$, it feeds two troubled gates, let $y_1$ and $y_2$ be neighbors of $x$ ($y_1$ may coincide with $y_2$). We add $x$ to $X$, also we add $y_1, y_2$ to $Y$. Note that it is possible to assign constants to $y_1$ and $y_2$ to make $C$ independent of $x$. (See the figure below. If $y_1$ differs from $y_2$, then we substitute constants to them so that they eliminate troubled gates fed by $x$ and leave $C$ independent of $x$. If $y_1$ coincides with $y_2$, then either $x = c$, or $y_1 = c$, or $y_1 = x \oplus c$ eliminates both troubled gates for some constant $c$; if we make an $x = c$ substitution, then formally we have to interchange $x$ and $y$, that is, add $y$ rather than $x$ to $X$.) Each of $y_1$, $y_2$ has at most one neighbor different from $x$. We remove $x$, $y_1$, $y_2$, neighbors of $y_1$ and $y_2$ (at most five vertices total) from the set $T$, if they belong to it. Since at each step we remove at most five vertices from $T$, we can repeat this routine $d$ times. Since we remove the neighbors of $y_1$ and $y_2$ from $T$, we guarantee that in all future steps when we pick an input, its neighbors do not belong to $Y$, so we can make arbitrary substitutions to them and leave the system consistent.


□

We are now ready to formulate our main result.

**Theorem 1.** *Let $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ be an rdq-disperser for dimension $d$ and $C$ be a fair semicircuit computing $f$. Let $\alpha_Q, \alpha_T, \alpha_I \geq 0$ be some constants, and $\alpha_T \leq 1/4$. Then $\mu(C, \emptyset) \geq \delta(n-d-2)$ where*

$$\delta := \alpha_I + \min\left\{\frac{\alpha_I}{2}, 4\beta, 3+\alpha_T, 2\beta+\alpha_Q, 5\beta-\alpha_Q, 2.5\beta+\frac{\alpha_Q}{2}\right\}, \tag{1}$$

*and $\beta = 1 - 4\alpha_T$.*

We defer the proof of this theorem to the next section. This theorem, together with Corollary 2, implies a lower bound on the circuit complexity of affine dispersers.

**Corollary 3.** *Let $\delta, \beta, \alpha_Q, \alpha_T, \alpha_I$ be constants as above, then the circuit size of an affine disperser for sublinear dimension is at least*

$$\left(\delta - \frac{\alpha_T}{2} - \alpha_I\right) n - o(n)\,.$$

*Proof.* Note that $q = 0$, $i \leq n$, $t < \frac{n}{2} + \frac{5d}{2}$ (see Lemma 3). Thus, the circuit size is

$$g = \mu - \alpha_Q \cdot q - \alpha_T \cdot t - \alpha_I \cdot i > \delta(n-d-2) - \alpha_T \cdot \left(\frac{n}{2} + \frac{5d}{2}\right) - \alpha_I \cdot n$$
$$= \left(\delta - \frac{\alpha_T}{2} - \alpha_I\right) n - \left(\delta + \frac{5\alpha_T}{2}\right) d - 2\delta = \left(\delta - \frac{\alpha_T}{2} - \alpha_I\right) n - o(n)\,.$$

□

The maximal value of $\delta - \frac{\alpha_T}{2} - \alpha_I$ satisfying the condition from Corollary 3 is given by the following linear program: maximize $\delta - \frac{\alpha_T}{2} - \alpha_I$ subject to

$$\begin{aligned} \beta + 4\alpha_T &= 1 \\ \alpha_T, \alpha_Q, \alpha_I, \beta &\geq 0 \\ \delta &\leq \alpha_I + \min\left\{\frac{\alpha_I}{2}, 4\beta, 3+\alpha_T, 2\beta+\alpha_Q, 5\beta-\alpha_Q, 2.5\beta+\frac{\alpha_Q}{2}\right\}. \end{aligned}$$

The optimal values for this linear program are

$$\begin{aligned} \alpha_T &= \frac{1}{43}, \\ \alpha_Q &= 1 + 22\alpha_T = \frac{65}{43}, \\ \alpha_I &= 6 + 2\alpha_T = 6 + \frac{2}{43}, \\ \beta &= 1 - 4\alpha_T = \frac{39}{43}, \\ \delta &= 9 + 3\alpha_T = 9 + \frac{3}{43}. \end{aligned}$$

This gives a $(3 + \frac{1}{86})n - o(n)$ lower bound for an affine disperser for sublinear dimension.

## 3.5 Gate elimination

In order to prove Theorem 1 we first show that it is always possible to make a substitution and decrease the measure by $\delta$.

**Theorem 2.** *Let $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ be an rdq-disperser for dimension $d$, let $R$ be an rdq-source of dimension $s \geq d+2$, and let $C$ be an optimal (i.e., $C$ with the smallest $\mu(C,R)$) fair semicircuit computing the function $f|_R$. Then there exist an rdq-source $R'$ of dimension $s' < s$ and a fair semicircuit $C'$ computing the function $f|_{R'}$ such that*

$$\mu(C', R') \leq \mu(C, R) - \delta(s - s')\,.$$

Before we proceed to the proof, we show how to infer the main theorem from this claim:

*Proof of Theorem 1.* We prove that for optimal $C$ computing $f|_R$, $\mu(C,R) \geq \delta(s-d-2)$. We do it by induction on $s$, the dimension of $R$. Note that the statement is vacuously true for $s \leq d+2$, since $\mu$ is nonnegative. Now suppose the statement is true for all rdq-sources of dimension strictly less than $s$ for some $s > d+2$, and let $R$ be an rdq-source of dimension $s$. Let $C$ be a fair semicircuit computing $f|_R$. Let $R'$ be the rdq-source of dimension $s'$ guaranteed to exist by Theorem 2, and let $C'$ be a fair semicircuit computing $f|_{R'}$. We have that

$$\mu(C,R) \geq \mu(C',R') + \delta(s-s') \geq \delta(s-d-2),$$

where the second inequality comes from the induction hypothesis. □

### 3.5.1 Proof sketch

The proof of Theorem 2 is based on a careful consideration of a number of cases. Before considering all of them formally, we show a high-level picture of the case analysis.

We fix the values of constants $\alpha_T, \alpha_Q, \alpha_I, \beta, \delta$ to the optimal values: $\alpha_T = \frac{1}{43}, \alpha_Q = \frac{65}{43}, \alpha_I = 6\frac{2}{43}, \beta = \frac{39}{43}, \delta = 9\frac{3}{43}$. Now it suffices to show that we can always make one substitution and decrease the measure by at least $\delta = 9\frac{3}{43}$. First we normalize the circuit. By Proposition 3, during normalization if we eliminate a gate then we introduce at most four new troubled gates, this means that we decrease the measure by at least $1 - 4\alpha_T = \frac{39}{43}$. Therefore, normalization never increases the measure.

We always make constant, linear or simple quadratic *substitution* to a variable. Then we remove the substituted variable from the circuit, so that for each assignment to the remaining variables the function is defined. It is easy to make a constant substitution $x = c$ for $c \in \{0,1\}$. We propagate the value $c$ to the inputs fed by $x$ and remove $x$ from the circuit, since it does not feed any other gates. An affine substitution $x = \bigoplus_{i\in I} x_i \oplus c$ is harder to make, because a straightforward way to eliminate $x$ would be to compute $\left(\bigoplus_{i\in I} x_i \oplus c\right)$ elsewhere. Fortunately, Lemma 2 shows how to compute it on the affine subspace defined by the substitution without using $x$ and without increasing the number of gates (after an extra gate introduced by this lemma is removed by normalization).

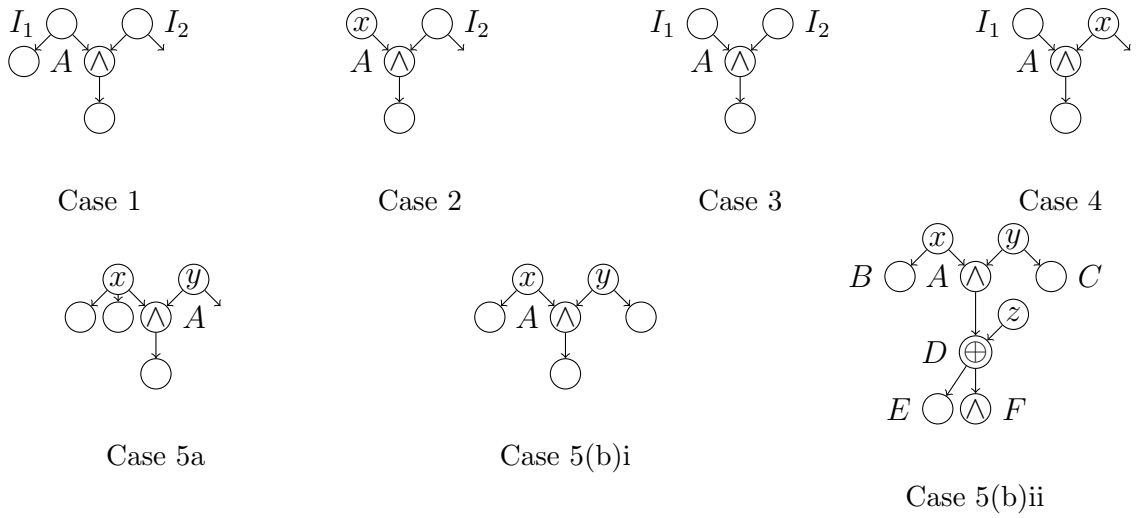



Figure 5: Gate elimination process in Theorem 2.

Thus, in this sketch we will be making arbitrary affine substitutions for sums that are computed in gates without saying that we need to run the reconstruction procedure first. Also, we will make a simple quadratic substitution $z = (x \oplus c_1)(y \oplus c_2) \oplus c_3$ only if the gates fed by $z$ are canceled out after the substitution, so that we do not need to propagate this quadratic value to other gates. We want to stay in the class of rdq-sources, therefore we cannot make an affine substitution to a variable $x$ if it already has been used in the right-hand side of some quadratic restriction $z = (x \oplus c_1)(y \oplus c_2) \oplus c_3$, also we cannot make quadratic substitutions that overlap in the variables. In this proof sketch we ignore these two issues, but they are addressed in the full proof in the next subsection.

Let $A$ be a topologically minimal and-type gate (i.e., an and-gate that is not reachable from any and-gate), let $I_1$ and $I_2$ be the inputs of $A$ ($I_1$ and $I_2$ can be variables or internal gates). Now we consider the following cases (see Figure 5).

1. At least one of $I_1$, $I_2$ (say, $I_1$) is an internal gate of outdegree greater than one. There is a constant $c$ such that if we assign $I_1 = c$, then $A$ becomes constant. (For example, if $A$ is an and, then $c = 0$, if $A$ is an or, then $c = 1$ etc.) When $A$ becomes constant it eliminates all the gates it feeds. Therefore, if we assign the appropriate constant to $I_1$, we eliminate $I_1$, two of the gates it feeds (including $A$), and also a successor of $A$, four gates total, and we decrease the measure by at least $\alpha_I + 4\beta = 9\frac{29}{43} > \delta$.

2. At least one of $I_1$, $I_2$ (say, $I_1$) is a variable of outdegree one. We assign the appropriate constant to $I_2$. This eliminates $I_2$, $A$, a successor of $A$, and $I_1$. This assignment eliminates at least two gates and two variables, so the measure decrease is at least $2\alpha_I + 2\beta = 13\frac{39}{43} > \delta$.

3. $I_1$ and $I_2$ are internal gates of outdegree one. Then if we assign the appropriate constant to $I_1$, we eliminate $I_1$, $A$, a successor of $A$, and $I_2$ (since $I_2$ does not feed any gates). We decrease measure by at least $\alpha_I + 4\beta > \delta$.

4. $I_1$ is an internal gate of outdegree one, $I_2$ is a variable of outdegree greater than one. Then we assign the appropriate constant to $I_2$. This assignment eliminates $I_2$, at least two of its successors (including $A$), a successor of $A$, and $I_1$ (since it does not feed any gates). Again, we decrease the measure by at least $\alpha_I + 4\beta > \delta$.

5. $I_1$ and $I_2$ are variables of outdegree greater than one.

   (a) $I_1$ or $I_2$ (say, $I_1$) has outdegree at least three. By assigning the appropriate constant to $I_1$ we eliminate at least three of the gates it feeds and a successor of $A$, four gates total.

   (b) $I_1$ and $I_2$ are variables of degree two. If $A$ is a $2^+$-gate we eliminate at least four gates by assigning $I_1$ so in what follows we assume that $A$ is a 1-gate. In this case $A$ is a troubled gate. We want to make the appropriate substitution and eliminate $I_1$ (or $I_2$), its successor, $A$, and $A$'s successor.

      i. If this substitution does not introduce new troubled gates, then we eliminate a variable, three gates and decrease the number of troubled gates by one. Thus, we decrease the measure by $\alpha_I + 3 + \alpha_T = 9\frac{3}{43} = \delta$.

      ii. If the substitution introduces troubled gates, then we consider which normalization rule introduces troubled gates. The full case analysis is presented in the next subsection, here we demonstrate just one case of the analysis. Let us consider the case when a new troubled gate is introduced when we eliminate the gate fed by $A$ (see Figure 5, the variable $z$ will feed a new troubled gate after assignments $x = 0$ or $y = 0$). In such a case we make a different substitution: $z = (x \oplus c_1)(y \oplus c_2) \oplus c_3$. This substitution eliminates gates $A, D, E, F$ and a gate fed by $F$. Thus, we eliminate one variable, five gates, but we introduce a new quadratic substitution, and decrease the measure by at least $\alpha_I + 5\beta - \alpha_Q = 9\frac{3}{43} = \delta$.

It is conceivable that when we count several eliminated gates, some of them coincide, so that we actually eliminate fewer gates. Usually in such cases we can prove that some other gates become trivial. This and other degenerate cases are handled in the full proof in the next subsection.

### 3.5.2 Full proof

*Proof of Theorem 2.* Since normalization does not increase the measure and does not change $R$, we may assume that $C$ is normalized.

In what follows we will further restrict $R$ by decreasing the number of free variables either by one or by two, then we will implement these substitutions in $C$ and normalize $C$ afterwards. Formally, we do it as follows:

- We add an equation or two to $R$.

- Since we now compute the disperser on a smaller set, we simplify $C$ (in particular, we disconnect the substituted variables from the rest of the circuit). For this, we
  - change the operations in the gates fed by the substituted variables or restructure the xor part of the circuit according to Lemma 2,
  - apply some normalization rules to remove some gates (and disconnect substituted variables).
- We count the decrease of $\mu$.
- We further normalize the circuit (without increase of $\mu$) to bring it to the normalized state required for the next induction step.

Since $s \geq d + 2$, even if we add two more lines to $R$, the disperser will not become a constant. This, in particular, implies that if a gate becomes constant then it is not an output gate and hence feeds at least one other gate. By going through the possible cases we will show that it is always possible to perform one or two consecutive substitutions matching at least one of the following types (by $\Delta\mu$ we denote the decrease of the measure after subsequent normalization).

1. Perform two consecutive affine substitutions to reduce the number of influential inputs by at least three. Per one substitution, this gives $\Delta\mu \geq 1.5\alpha_I$.
2. Perform one affine substitution to reduce the number of influential inputs by at least 2: $\Delta\mu \geq 2\alpha_I$ (numerically, this case is subsumed by the previous one).
3. Perform one affine substitution to kill four internal gates: $\Delta\mu \geq 4\beta + \alpha_I$.
4. Perform one constant substitution to eliminate three internal gates including at least one troubled gate so that no new troubled gate is introduced: $\Delta\mu \geq \alpha_I + 3 + \alpha_T$.
5. Perform one *quadratic* substitution to kill five internal gates: $\Delta\mu \geq 5\beta - \alpha_Q + \alpha_I$.
6. Perform two affine substitutions to kill at least five internal gates and replace a quadratic substitution by an affine one, reducing the measure by at least $5\beta + \alpha_Q + 2\alpha_I$. Per substitution this is $\Delta\mu \geq 2.5\beta + \frac{\alpha_Q}{2} + \alpha_I$.
7. Perform one affine substitution to kill two internal gates and replace one quadratic substitution by an affine one: $\Delta\mu \geq 2\beta + \alpha_Q + \alpha_I$.

All substitutions that we perform are of the form such that adding them to an rdq-source results in a new rdq-source.

We check all possible cases of $(C, R)$. In every case we assume that the conditions of the previous cases are not satisfied. We also rely on the specified order of applications of the normalization rules where applicable.

Note that the measure can accidentally drop less than we expect if new troubled gates emerge. We take care of this when counting the number of internal gates that disappear, recall Proposition 3 that guarantees the decrease of $\beta$ per one eliminated gate. If some additional gate accidentally disappears, it may introduces new troubled gates but does not increase the measure, because $\beta \geq 0$.

**Cases:**

1. The circuit contains a protected variable $q$ that either feeds an and-type gate or feeds at least two internal gates. Then there is a type 7 substitution of $q$ by a constant.

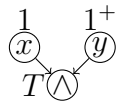
2. The circuit contains a protected 0-variable $q$ occurring in the right-hand side of a quadratic substitution together with some variable $q'$. We substitute a constant to $q'$. After this neither $q$ nor $q'$ are influential, so we have a type 2 substitution.

   *Note that after this case all protected variables are 1-variables feeding xor gates.*

3. The circuit contains a variable $x$ feeding an and-type gate $T$, and $out(x) + out(T) \geq 4$. Then if $x$ gets the value that trivializes $T$, we remove four gates: $T$ by Rule 2, and descendants of $x$ and $T$ by Rule 3. If some of these descendants coincide, this gate becomes trivial (instead of passing) and is removed by Rule 2 (instead of Rule 3), and an additional gate (a descendant of this descendant) is removed by Rule 3. This makes a type 3 substitution.

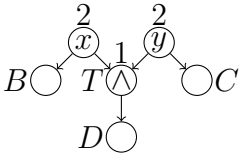
   *Note that after this case all variables feeding and-gates have outdegree one or two.*

4. There is an and-type gate $T$ fed by two input gates $x$ and $y$, one of which (say, $x$) has outdegree 1. Adopt the notation from the following picture. In this and all the subsequent pictures we show the outdegrees near the gates that are important for the case analysis.




   We substitute $y$ by a constant trivializing $T$. This removes the dependence on $x$ and $y$ (which are both influential and unprotected), a type 2 substitution.

5. There is an and-type gate $T$ fed by two input gates $x$ and $y$, and at this point (due to the cases 3 and 4) we inevitably have $out(T) = 1$ and $out(x) = out(y) = 2$, that is, $T$ is "troubled". Adopt calling conventions from the following picture:

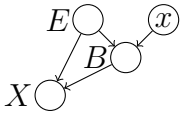


Since the circuit is normalized, $B \neq D$ and $C \neq D$ (Rule 4). One can now remove three gates by substituting a constant to $x$ that trivializes $T$. If in addition to the three gates one more gate can be killed, we are done (substitution of type 3). Otherwise, we have just three gates, but the troubled gate $T$ is removed. If this does not introduce a new troubled gate, it makes a substitution of type 4. Likewise, if this is the case for a substitution to $y$, we are done.

*So in the remaining subcases of Case 5 we will be fighting the situation where only three gates are eliminated while one or more troubled gates are introduced.*

How can it happen that a new troubled gate is introduced? This means that something has happened around some and-type gate $E$. Whatever has happened, it is due to two gates, $B$ and $D$, that became passing (if some of them became trivial, then one more gate would be removed). The options are:

- *$E$ gets as input a variable* instead of an internal gate (because some gate in between became passing).
- A variable *increases its outdegree* from 1 to 2 (because a gate of degree at least two became passing), and this variable starts to feed $E$ (note that it could not feed it before, because after the increase it would feed it twice).
- A variable *decreases its outdegree* to 2. This variable could not feed $E$ before this happens, because this would be Case 3. It takes at least one passing gate, $X$, to pass a new variable to $E$, thus the decrease of the outdegree has happened because of a single passing gate $Y$. In order to decrease the outdegree of the variable this gate must have outdegree 1, thus it would be removed by Rule 4 as useless.
- *$E$ decreases its outdegree* to 1.
  - This could happen if two gates, $B$ and $D$, became passing, and they fed a single gate. However, in this case $E$ should already have 2-variables as its inputs, Case 3.
  - This could also happen if $E$ feeds $B$ and some gate $X$, and $B$ becomes passing to $X$. However, in this case $B$ is useless (Rule 4). (Note that $out(B) = 1$, because otherwise $E$ would not decrease its outdegree to 1.)



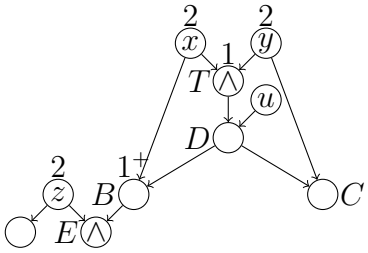

  - Similarly, if $E$ feeds $D$ and some gate $X$, and $D$ becomes passing to $X$.

Summarizing, only the two first possibilities could happen, and both pass some variable to $E$ through either $B$ or $D$ (or both).

The plan for the following cases it to exploit the local topology, that is, possible connections between $B$, $D$, and $C$. First we consider "degenerate" cases where these gates are locally connected beyond what is shown in the figure in case 5. After this, we continue to the more general case.

6. If $B = C$, then one can trivialize both $T$ and $B$ either by substituting a constant to $x$ or $y$ or by one affine substitution $y = x \oplus c$ (using Proposition 2) for the appropriate constant $c$ (this can be easily seen by examining the possible operations in the two gates). Since $x$ and $y$ are unprotected, the number of influential variables is decreased by 2, making a substitution of type 2.

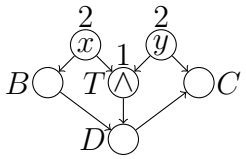
7. Assume that $D$ feeds both $B$ and $C$. In this case, a new troubled gate may emerge only because $D$ is fed by a variable $u$, and it is passed to some and-type gate $E$. Note that $out(D) \leq 2$, because otherwise $u$ would become a 3-variable and $E$ would not become troubled. Therefore, $u$ cannot be passed by $D$ to $E$ directly, it is passed via $B$.




If $out(B) \geq 2$, then even if $out(u) = 1$, it must be that $C = E$ or that $B$ feeds $C$, because otherwise $u$ would become a 3-variable after substituting $x$. Neither are possible: $C = E$ would imply $B = D$ and $y = z$, contradicting the assumption that $D \neq B$ (from 5); if $B$ feeds $C$, that means that $B = D$, which is impossible. Therefore, we conclude that $out(B) = 1$. So we can substitute constants for $z$, to make $B$ a 0-gate, and for $y$, to trivialize $T$. This way $x$ ceases to be influential, and we have $\Delta\mu \geq 3\alpha_I$ for two substitutions (type 1).

*Note that after this case we can assume that $D$ does not feed $B$. If it does, we switch the roles of the variables $x$ and $y$.*

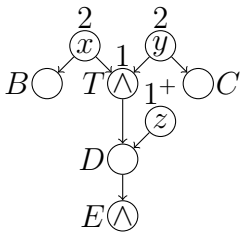
8. Assume now that $B$ feeds $D$, and $D$ feeds $C$. (Or, symmetrically, $C$ feeds $D$, and $D$ feeds $B$.) Then substituting $y$ to trivialize $T$ removes $T$, $D$, and $C$, and introduces *no new troubled gates*, because $C$ and $D$ are passing the internal gate $B$. We assumed it is not the case.



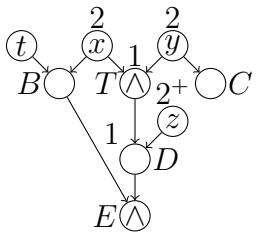

9. We can now assume that $B$ and $D$ are not connected (in any direction).

   Indeed, if $B$ feeds $D$, we can switch the roles of $x$ and $y$ unless $C$ feeds $D$ (impossible, because then $D$ has three inputs: $T$, $B$, and $C$) or unless we switched $x$ and $y$ before (that is, $D$ feeds $C$, Case 8).

(a) Assume that $D$ feeds a new troubled gate under the substitution of $x$. The troubled gate $E$ gets some variable $z$ from $D$ (directly, as $D$ and $B$ are not connected).



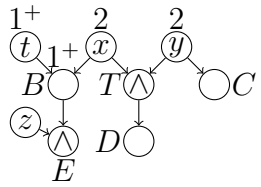

- If $out(z) \geq 2$, then $out(D) = 1$ and $E$ is fed by another variable $t$ either directly or via $B$. In the former case, we can substitute $t$ to trivialize $E$, this kills $E$ and the gate it feeds, and also makes $D$ and then $T$ 0-gates; a type 3 substitution. In the latter case:




  - if $out(B) \geq 2$, then $B$ is a xor-type gate (see Case 5a), and by substituting $x = t \oplus c$ for the appropriate constant $c$, we can make $B$ a constant trivializing $E$ and remove two more descendants of $B$ and $E$, a type 3 substitution;
  - if $out(B) = 1$, then we can set $z$ and $y$ to constants trivializing $T$ and $E$, respectively. Then $B$ becomes a 0-gate and is eliminated, which means that $x$ becomes a 0-variable. We then get a substitution of type 1.

  *We can now assume that $out(z) = 1$ and thus $out(D) \geq 2$, because $z$ must get outdegree two in order to feed the new troubled gate.*
- If $D$ is an and-type gate, substituting $z$ by the appropriate constant trivializes $D$ and kills both gates that it feeds; also $T$ becomes a 0-gate, a type 3 substitution.
- If $z$ is protected, we set $x$ and $z$ to constants trivializing $T$, $D$, and $E$. This additionally removes $B$ and the gates that $E$ feeds, at least five gates in total. Since we also kill a quadratic substitution, this makes a type 6 substitution.
- Since we can now assume that $z$ is unprotected and $D$ is an xor-type gate, we can make a substitution $z = (x \oplus c_1)(y \oplus c_2) \oplus c_3$ for appropriate constants $c_1$, $c_2$, $c_3$ to assign $D$ a value that trivializes $E$. This makes $T$ a 0-gate and removes also $D$, $E$, another gate that $D$ feeds, and the gate(s) that $E$ feeds. As usual, if some passing gates coincide, another gate is removed. Taking into account the penalty for introducing a quadratic substitution, we get a substitution of type 5.

(b) Since $D$ does not feed a new troubled gate, $B$ does, and $B$ is fed directly by a variable $t$ (since $B$ and $D$ are not connected). The new troubled gate $E$ must be also fed directly by a variable $z$ (because $D$ does not feed it).



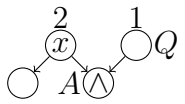

- If $out(B) \geq 2$ (which means $B$ is a xor-type gate, see Case 5a), then by substituting $x = t \oplus c$ (using Proposition 2) for the appropriate constant $c$, we can make $B$ a constant trivializing $E$ and remove two more descendants of $B$ and $E$, a type 3 substitution.
- If $out(B) = 1$, then we can set $z$ and $y$ to constants trivializing $T$ and $E$, respectively. Then $B$ becomes a 0-gate and is eliminated, which means that $x$ becomes a 0-variable. We then get a substitution of type 1.

---

*Starting from the next case we will consider a topologically minimal and-type gate and call it $A$ for the remaining part of the proof. Here $A$ is topologically minimal if it cannot be reached from another and-type gate via a directed path.*

Note that the circuit $C$ must contain at least one and-type gate (otherwise it computes an affine function, and a single affine substitution makes it constant). The minimality implies that both inputs of $A$ are computed by fair cyclic xor-circuits (note that a subcircuit of a fair circuit is fair, because it corresponds to a submatrix of a full-rank matrix); in particular, they can be input gates.

10. One input of $A$ is an input gate $x$ of outdegree 2 while the other one is an internal gate $Q$ of outdegree 1.




Recall that $x$ is unprotected due to Case 1, and $x$ cannot feed $Q$ because of Rule 4. Substituting $x$ by the constant trivializing $A$ eliminates the two successors of $x$, all the successors of $A$, and makes $Q$ a 0-gate which is then eliminated by Rule 1. A type 3 substitution. (As usual, if the only successor of $A$ coincides with the other successor of $x$ then this gate becomes constant so its successors are also eliminated. That is, in any case at least four gates are eliminated.)

11. One input to $A$ is an internal gate $Q$. Denote the other input by $P$. If $P$ is also an internal gate and has outdegree larger than $Q$ we switch the roles of $P$ and $Q$.

    In this case we will try to substitute a value to $Q$ in order to trivialize $A$. $Q$ is a gate computed by a fair xor-circuit, so it computes an affine function $c \oplus \bigoplus_{i \in I} x_i$. For this, we use the xor-reconstruction procedure described in Lemma 2. In order to perform it, we need at least one unprotected variable $x_i$ with $i \in I$.

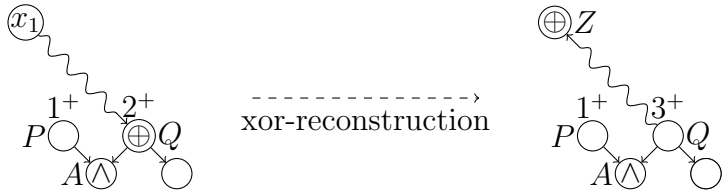
    (a) Such a variable $x_1$ exists.

    We then add the substitution $x_1 = b \oplus c \oplus \bigoplus_{i \in I \setminus \{1\}} x_i$ to the rdq-source $R$ for the appropriate constant $b$ (so that $Q$ on the updated $R$ computes the constant trivializing $A$). We could now simply replace the operation in $Q$ by this constant (since the just updated circuit computes correctly the disperser on the just updated $R$). However, we need to eliminate the just substituted variable $x_1$ from the circuit. To do this, we perform the reconstruction described in Lemma 2. Note that it only changes the in- and outdegrees of $x_1$ (replacing it by a new internal gate $Z$) and $Q$. No new troubled gates are introduced, and the subsequent application of Rule 2 to $Q$ removes $Q$ without introducing new troubled gates as well.

    Moreover, normalizations remove all descendants of $Q$, all descendants of $A$, and, in the case $out(P) = 1$, Rule 1 removes $P$ if it is an internal gate, or $P$ becomes a 0-variable, if it was a variable. It remains to count the decrease of the measure.

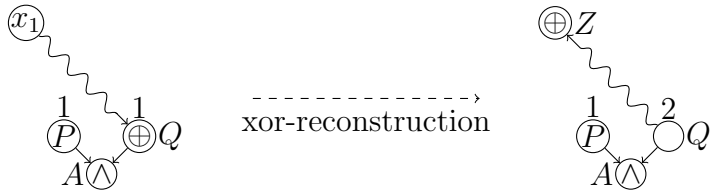
    Below we go through several subcases depending on the type of the gate $P$.

    i. $Q$ is a $2^+$-gate. We recall the general picture of xor-reconstruction.

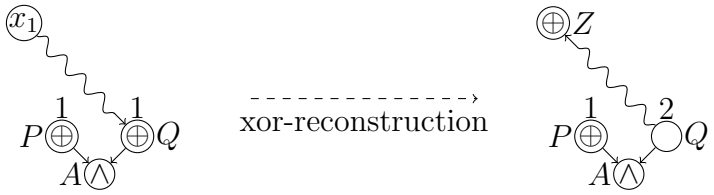
    After the reconstruction, there are at least three descendants of $Q$ and at least one descendant of $A$, a type 3 substitution.

    ii. $Q$ is an internal 1-gate and $P$ is an input gate. Then $P$ has outdegree 1 and is unprotected (see Cases 10, 1).

    Note that $P \neq x_1$ since the only outgoing edge of $P$ goes to an and-type gate. This means that $P$ is left untouched by the xor-reconstruction. After trivializing $A$ the circuit becomes independent of both $x_1$ and $P$ giving a type 2 substitution.

iii. $Q$ is an internal 1-gate and $P$ is an internal gate. Then $P$ is a 1-gate (if the outdegree of $P$ were larger we would switch the roles of $P$ and $Q$).




Again, $P$ is left untouched by the xor-reconstruction since it only has one successor and it is of and-type while the xor-reconstruction is performed in the linear part of the circuit. After the substitution, we remove two successors of $Q$, at least one successor of $A$, and make $P$ a 0-gate. A type 3 substitution. Note that $P$ cannot be a successor of $Q$ because of Rule 4.

(b) All variables in the affine function computed by $Q$ are protected.

i. Both inputs to $Q$, say $x_j$ and $x_k$, are variables, and they occur in the same quadratic substitution $w = (x_j \oplus c)(x_k \oplus c') \oplus c''$. Then perform a substitution $x_j = x_k \oplus c'''$ (using Proposition 2) in order to trivialize the gate $A$. It kills the quadratic substitution (and does not harm other quadratic substitutions, because $x_j$ and $x_k$ could not occur in them), $Q$, $A$, its descendant (and more, but we do not need it), which makes $\Delta\mu \geq 3\beta + \alpha_Q + \alpha_I$, a type 7 substitution.

ii. $Q$ is a $2^+$-gate. Take any $j \in I$. Assume that $x_j$ occurs in a quadratic substitution $x_p = (x_j \oplus a)(x_k \oplus b) \oplus c$. Recall that at this point all protected variables are 1-variables feeding xor-gates (see Cases 1 and 2). We substitute $x_k$ by a constant $d$ and normalize the circuit. This eliminates the successor of $x_k$, kills the quadratic substitution, and makes $x_j$ unprotected. If at least two gates are removed during normalization then we get $\Delta\mu \geq 2\beta + \alpha_Q + \alpha_I$, a type 7 substitution. In what follows we assume that the only gate removed during normalization after the substitution $x_k \leftarrow d$ is the successor of $x_k$.
If the gate $Q$ is not fed by $x_k$ then it has outdegree at least 2 after the substitution $x_k \leftarrow d$ and normalizing the descendants of $x_k$. If the gate $Q$ is fed by $x_k$ then its second input must be an internal xor-gate $Q'$ (if it were an input gate it would be a variable $x_j$ but then we would fall into Case 11(b)i). Then after substituting $x_k \leftarrow d$ and normalizing $Q$ the gate $Q'$ feeds $A$ and has outdegree at least 2. We denote $Q'$ by $Q$ in this case.
Hence in any case, in the circuit normalized after the substitution $x_k \leftarrow d$, the gate $A$ is fed by the $2^+$-gate $Q$ that computes an affine function of variables containing an unprotected variable $x_j$. We then make $Q$ constant trivializing $A$ by the appropriate affine substitution to $x_j$. This kills four gates. Together with the substitution $x_k \leftarrow d$, it gives $\Delta\mu \geq 5\beta + \alpha_Q + 2\alpha_I$, a type 6 substitution.

*Hence in what follows we assume that* $out(Q) = 1$. *Therefore* $P$ *is either a variable or an internal xor-type* 1*-gate.*

iii. $P$ is an input gate. Then it has the following properties as in Case 11(a)ii. Take any $j \in I$ and assume that $x_j$ appears with $x_k$ in a quadratic substitution. We first substitute $x_k \leftarrow d$ and normalize the circuit. After this the second input of $A$ still computes a linear function that depends on $x_j$ which is now unprotected. We make an affine substitution to $x_j$ trivializing $A$. This makes $P$ a 0-variable, a type 1 substitution.

iv. $P$ is an internal xor-type 1-gate. If $P$ computes an affine function of variables at least one of which is unprotected, we are in Case 11(a)iii with $P$ and $Q$ exchanged. So, in what follows we assume that both $P$ and $Q$ compute affine functions of protected variables.

   A. Both inputs to $P$ or $Q$ (say, $P$) are variables $x_p$ and $x_q$. Let $x_j$ be a variable from the affine function computed at $Q$ and let $x_k$ be its couple. Note that $x_j \neq x_p, x_q$ while it might be the case that $x_k = x_p$ or $x_k = x_q$. We substitute $x_k$ by a constant to make $x_j$ unprotected. We then trivialize $A$ by an affine substitution to $x_j$. This way, we kill the dependence on three variables by two substitutions. A type 1 substitution.
   *Thus in what follows we can assume that both $P$ and $Q$ have at least one internal xor-gate as an input.*

   B. One of $P$ and $Q$ (say, $Q$) computes an affine function of variables one of which (call it $x_j$) have a couple $x_k$ that does not feed $P$. We substitute $x_k$ by a constant and normalize the descendant of $x_k$. It only kills one xor-gate fed by $x_k$ and makes $x_j$ unprotected. Note that at this point $P$ is still a 1-xor. We then trivialize $A$ by substituting $x_j$ by an affine function. Similarly to Case 11(a)iii, this kills four gates and gives, for two substitutions, $\Delta\mu \geq 5\beta + \alpha_Q + 2\alpha_I$. A type 6 substitution.

   C. The only case when the condition of the previous case does not apply is the following: $P$ computes an affine function on a single variable $x_i$, $Q$ computes an affine function on a single variable $x_j$, the variables $x_i$ and $x_j$ appear together in a quadratic substitution, and moreover $x_i$ feeds $Q$ while $x_j$ feeds $P$. But this is just impossible. Indeed, since $x_i$ is a protected variable it only feeds $Q$. As $Q$ computes an affine function on $x_i$, Lemma 1 guarantees that there is a path from $x_i$ to $Q$. But this path must go through $P$ and $A$ leading to a cycle that goes through an and-type gate $A$.

□

# Acknowledgements

Research is partially supported by NSF (grant 1319051) and the Government of the Russian Federation (grant 14.Z50.31.0030). We also would like to thank Dmitry Itsykson and Alexander Knop who survived a six-hour seminar on the proof and made valueable comments.

# References


[And87] Alexander E. Andreev. On a method for obtaining more than quadratic effective lower bounds for the complexity of $\pi$-schemes. *Moscow Univ. Math. Bull.*, 42(1):63–66, 1987.

[BFT98] Harry Buhrman, Lance Fortnow, and Thomas Thierauf. Nonrelativizing separations. In *CCC-98*, 1998.

[BK12] Eli Ben-Sasson and Swastik Kopparty. Affine dispersers from subspace polynomials. *SIAM J. Comput.*, 41(4):880–914, 2012.

[BKS+10] Boaz Barak, Guy Kindler, Ronen Shaltiel, Benny Sudakov, and Avi Wigderson. Simulating independence: New constructions of condensers, Ramsey graphs, dispersers, and extractors. *J. ACM*, 57(4), 2010.

[Blu84] Norbert Blum. A Boolean function requiring $3n$ network size. *Theor. Comput. Sci.*, 28:337–345, 1984.

[Bou07] Jean Bourgain. On the construction of affine extractors. *GAFA Geometric And Functional Analysis*, 17(1):33–57, 2007.

[BV14] Eli Ben-Sasson and Emanuele Viola. Short PCPs with projection queries. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 163–173. Springer, 2014.

[CK15] Ruiwen Chen and Valentine Kabanets. Correlation bounds and #sat algorithms for small linear-size circuits. In Dachuan Xu, Donglei Du, and Dingzhu Du, editors, *Computing and Combinatorics - 21st International Conference, COCOON 2015, Beijing, China, August 4-6, 2015, Proceedings*, volume 9198 of *Lecture Notes in Computer Science*, pages 211–222. Springer, 2015.

[CKK+15] Ruiwen Chen, Valentine Kabanets, Antonina Kolokolova, Ronen Shaltiel, and David Zuckerman. Mining circuit lower bound proofs for meta-algorithms. *Computational Complexity*, 24(2):333–392, 2015.

[CR06] Venkatesan T. Chakaravarthy and Sambuddha Roy. Oblivious symmetric alternation. In Bruno Durand and Wolfgang Thomas, editors, *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23-25, 2006, Proceedings*, volume 3884 of *Lecture Notes in Computer Science*, pages 230–241. Springer, 2006.

[CT15] Gil Cohen and Avishay Tal. Two structural results for low degree polynomials and applications. In Naveen Garg, Klaus Jansen, Anup Rao, and José D. P.

Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2015, August 24-26, 2015, Princeton, NJ, USA*, volume 40 of *LIPIcs*, pages 680–709. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[DC89] Patrick W. Dymond and Stephen A. Cook. Complexity theory of parallel time and hardware. *Inf. Comput.*, 80(3):205–226, 1989.

[DK11] Evgeny Demenkov and Alexander S. Kulikov. An elementary proof of a $3n - o(n)$ lower bound on the circuit complexity of affine dispersers. In Filip Murlak and Piotr Sankowski, editors, *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 256–265. Springer, 2011.

[DKMM15] Evgeny Demenkov, Alexander S. Kulikov, Olga Melanich, and Ivan Mihajlin. New lower bounds on circuit size of multi-output functions. *Theory Comput. Syst.*, 56(4):630–642, 2015.

[Dvi12] Zeev Dvir. Extractors for varieties. *Computational Complexity*, 21(4):515–572, 2012.

[FGK09] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5), 2009.

[GK15] Alexander Golovnev and Alexander S. Kulikov. Weighted gate elimination: Boolean dispersers for quadratic varieties imply improved circuit lower bounds. Submitted, 2015.

[Hås86] Johan Håstad. Almost optimal lower bounds for small depth circuits. In Juris Hartmanis, editor, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 6–20. ACM, 1986.

[Hås98] Johan Håstad. The shrinkage exponent of de Morgan formulas is 2. *SIAM J. Comput.*, 27(1):48–64, 1998.

[HU69] John E. Hopcroft and Jeffrey D. Ullman. *Formal Languages and Their Relation to Automata*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1969.

[IM02] Kazuo Iwama and Hiroki Morizumi. An explicit lower bound of $5n - o(n)$ for Boolean circuits. In Krzysztof Diks and Wojciech Rytter, editors, *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26-30, 2002, Proceedings*, volume 2420 of *Lecture Notes in Computer Science*, pages 353–364. Springer, 2002.

[IN93] Russell Impagliazzo and Noam Nisan. The effect of random restrictions on formula size. *Random Struct. Algorithms*, 4(2):121–134, 1993.

[Khr71] Valeriy M. Khrapchenko. A method of determining lower bounds for the complexity of $\pi$-schemes. *Math. Notes of the Acad. of Sci. of the USSR*, 10(1):474–479, 1971.

[KK10] Arist Kojevnikov and Alexander S. Kulikov. Circuit complexity and multiplicative complexity of Boolean functions. In Fernando Ferreira, Benedikt Löwe, Elvira Mayordomo, and Luís Mendes Gomes, editors, *Programs, Proofs, Processes, 6th Conference on Computability in Europe, CiE 2010*, volume 6158 of *Lecture Notes in Computer Science*, pages 239–245. Springer, 2010.

[KM65] Boris M. Kloss and Vadim A. Malyshev. Estimates of the complexity of certain classes of functions. *Vestn.Moskov.Univ.Ser.1*, 4:44–51, 1965. In Russian.

[KRT13] Ilan Komargodski, Ran Raz, and Avishay Tal. Improved average-case lower bounds for demorgan formula size. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 588–597. IEEE Computer Society, 2013.

[Kul99] Oliver Kullmann. New methods for 3-SAT decision and worst-case analysis. *Theor. Comput. Sci.*, 223(1-2):1–72, 1999.

[Li11] Xin Li. A new approach to affine extractors and dispersers. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity, CCC 2011, San Jose, California, June 8-10, 2011*, pages 137–147. IEEE Computer Society, 2011.

[LR01] Oded Lachish and Ran Raz. Explicit lower bound of $4.5n - o(n)$ for Boolean circuits. In Jeffrey Scott Vitter, Paul G. Spirakis, and Mihalis Yannakakis, editors, *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*, pages 399–408. ACM, 2001.

[LS73] Edward A. Lamagna and John E. Savage. On the logical complexity of symmetric switching functions in monotone and complete bases. Technical report, Brown University, 1973.

[Nec66] Edward I. Nechiporuk. On a Boolean function. *Doklady Akademii Nauk. SSSR*, 169(4):765–766, 1966.

[NTW04] Arfst Nickelsen, Till Tantau, and Lorenz Weizsäcker. Aggregates with component size one characterize polynomial space. *Electronic Colloquium on Computational Complexity (ECCC)*, (028), 2004.

[Nur09] Sergey Nurk. An $2^{0.4058m}$ upper bound for Circuit SAT. Technical Report 10, Steklov Institute of Mathematics at St.Petersburg, 2009. PDMI Preprint.

[Pau77] Wolfgang J. Paul. A $2.5n$-lower bound on the combinational complexity of Boolean functions. *SIAM J. Comput.*, 6(3):427–443, 1977.

[PZ93] Mike Paterson and Uri Zwick. Shrinkage of de Morgan formulae under restriction. *Random Struct. Algorithms*, 4(2):135–150, 1993.

[Raz85] Alexander A. Razborov. Lower bound on monotone complexity of some Boolean functions. *Doklady Akademii Nauk. SSSR*, 281(4):798–801, 1985.

[RB12] Marc D. Riedel and Jehoshua Bruck. Cyclic boolean circuits. *Discrete Applied Mathematics*, 160(13-14):1877–1900, 2012.

[Riv77] Ronald L. Rivest. The necessity of feedback in minimal monotone combinational circuits. *IEEE Trans. Computers*, 26(6):606–607, 1977.

[San10] Rahul Santhanam. Fighting perebor: New and improved algorithms for formula and QBF satisfiability. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 183–192. IEEE Computer Society, 2010.

[Sav14] Sergey Savinov. Upper bounds for the Boolean circuit satisfiability problem. Master Thesis defended at St.Peterburg Academic University of Russian Academy of Sciences, 2014. In Russian.

[Sch74] Claus-Peter Schnorr. Zwei lineare untere Schranken für die Komplexität Boolescher Funktionen. *Computing*, 13(2):155–171, 1974.

[Sch76] Claus-Peter Schnorr. The combinational complexity of equivalence. *Theor. Comput. Sci.*, 1(4):289–295, 1976.

[Sha49] Claude E. Shannon. The synthesis of two-terminal switching circuits. *Bell Systems Technical Journal*, 28:59–98, 1949.

[Sha11] Ronen Shaltiel. Dispersers for affine sources with sub-polynomial entropy. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 247–256. IEEE Computer Society, 2011.

[SS91] Victor Shoup and Roman Smolensky. Lower bounds for polynomial evaluation and interpolation problems. In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 378–383. IEEE Computer Society, 1991.

[ST13] Kazuhisa Seto and Suguru Tamaki. A satisfiability algorithm and average-case hardness for formulas over the full binary basis. *Computational Complexity*, 22(2):245–274, 2013.

[Sto77] Larry J. Stockmeyer. On the combinational complexity of certain symmetric Boolean functions. *Mathematical Systems Theory*, 10:323–336, 1977.

[Sub61] Bella A. Subbotovskaya. Realizations of linear functions by formulas using $+, \cdot, -$. *Doklady Akademii Nauk. SSSR*, 136(3):553–555, 1961.

[Tal14] Avishay Tal. Shrinkage of de Morgan formulae by spectral techniques. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 551–560. IEEE, 2014.

[Wil13] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM J. Comput.*, 42(3):1218–1244, 2013. Extended abstract appeared in Proc. STOC-2010.

[Wil14] Ryan Williams. Nonuniform ACC circuit lower bounds. *JACM*, 61(1), 2014. Extended abstract appears in Proc. CCC-2011.

[Yao85] Andrew C. Yao. Separating the polynomial-time hierarchy by oracles (preliminary version). In *FOCS*, pages 1–10. IEEE Computer Society, 1985.

[Yeh11] Amir Yehudayoff. Affine extractors over prime fields. *Combinatorica*, 31(2):245–256, 2011.

[Zwi91] Uri Zwick. A $4n$ lower bound on the combinational complexity of certain symmetric Boolean functions over the basis of unate dyadic Boolean functions. *SIAM J. Comput.*, 20(3):499–505, 1991.



ECCC ISSN 1433-8092

http://eccc.hpi-web.de