# An Elementary Proof of a $3n - o(n)$ Lower Bound on Circuit Complexity of Affine Dispersers

E. Demenkov and A. Kulikov

Steklov Institute of Mathematics at St. Petersburg

Estonian Theory Days
08 October 2011

# Boolean Circuits

Inputs:

$x_1, x_2, \ldots, x_n, 0, 1$

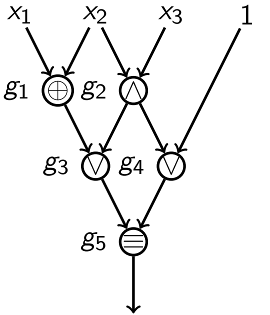
Gates:

binary functions

Fan-out:

unbounded

$$
\begin{aligned}
g_1 &= x_1 \oplus x_2 \\
g_2 &= x_2 \wedge x_3 \\
g_3 &= g_1 \vee g_2 \\
g_4 &= g_2 \vee 1 \\
g_5 &= g_3 \equiv g4
\end{aligned}
$$



# Random Functions are Complex

- Shannon counting argument: count how many different Boolean functions in $n$ variables can be computed by circuits with $t$ gates and compare this number with the total number $2^{2^n}$ of all Boolean functions.

# Random Functions are Complex

- Shannon counting argument: count how many different Boolean functions in $n$ variables can be computed by circuits with $t$ gates and compare this number with the total number $2^{2^n}$ of all Boolean functions.
- The number $F(n, t)$ of circuits of size $\leq t$ with $n$ input variables does not exceed

$$\left(16(t+n+2)^2\right)^t \ .$$

  Each of $t$ gates is assigned one of 16 possible binary Boolean functions that acts on two previous nodes, and each previous node can be either a previous gate ($\leq t$ choices) or a variables or a constant ($\leq n+2$ choices).

# Random Functions are Complex

- Shannon counting argument: count how many different Boolean functions in $n$ variables can be computed by circuits with $t$ gates and compare this number with the total number $2^{2^n}$ of all Boolean functions.
- The number $F(n,t)$ of circuits of size $\leq t$ with $n$ input variables does not exceed
$$\left(16(t+n+2)^2\right)^t .$$
Each of $t$ gates is assigned one of 16 possible binary Boolean functions that acts on two previous nodes, and each previous node can be either a previous gate ($\leq t$ choices) or a variables or a constant ($\leq n+2$ choices).
- For $t = 2^n/(10n)$, $F(n,t)$ is approximately $2^{2^n/5}$, which is $\ll 2^{2^n}$.

# Random Functions are Complex

- Shannon counting argument: count how many different Boolean functions in $n$ variables can be computed by circuits with $t$ gates and compare this number with the total number $2^{2^n}$ of all Boolean functions.
- The number $F(n,t)$ of circuits of size $\leq t$ with $n$ input variables does not exceed
$$\left(16(t+n+2)^2\right)^t .$$
Each of $t$ gates is assigned one of 16 possible binary Boolean functions that acts on two previous nodes, and each previous node can be either a previous gate ($\leq t$ choices) or a variables or a constant ($\leq n+2$ choices).
- For $t = 2^n/(10n)$, $F(n,t)$ is approximately $2^{2^n/5}$, which is $\ll 2^{2^n}$.
- Thus, the circuit complexity of almost all Boolean functions on $n$ variables is exponential in $n$. Still, we do not know any explicit function with super-linear circuit complexity.

# Known Lower Bounds

| | circuit size | formula size |
|---|---|---|
| full binary basis $B_2$ | $3n - o(n)$ [Blum] | $n^{2-o(1)}$ [Nechiporuk] |
| basis $U_2 = B_2 \setminus \{\oplus, \equiv\}$ | $5n - o(n)$ [Iwama et al.] | $n^{3-o(1)}$ [Hastad] |
| monotone basis $M_2 = \{\vee, \wedge\}$ | exponential [Razborov; Alon, Boppana; Andreev; Karchmer, Wigderson] | |

# Known Lower Bounds for Circuits over $B_2$

## Known Lower Bounds

| | |
|---|---|
| $2n - c$ | [Kloss and Malyshev, 65] |
| $2n - c$ | [Schnorr, 74] |
| $2.5n - o(n)$ | [Paul, 77] |
| $2.5n - c$ | [Stockmeyer, 77] |
| $3n - o(n)$ | [Blum, 84] |

# Known Lower Bounds for Circuits over $B_2$

## Known Lower Bounds

| | |
|---|---|
| $2n - c$ | [Kloss and Malyshev, 65] |
| $2n - c$ | [Schnorr, 74] |
| $2.5n - o(n)$ | [Paul, 77] |
| $2.5n - c$ | [Stockmeyer, 77] |
| $3n - o(n)$ | [Blum, 84] |

## This Talk

In this talk, we will present a new proof of a $3n - o(n)$ lower. The proof is much simpler than Blum's proof, however the function used is much more complicated.

# Gate Elimination Method

## Gate Elimination

All the proofs are based on the so-called gate elimination method. This is essentially the only known method for proving lower bounds on circuit complexity.

# Gate Elimination Method

### Gate Elimination

All the proofs are based on the so-called gate elimination method. This is essentially the only known method for proving lower bounds on circuit complexity.

### The main idea

- Take an optimal circuit for the function in question.

# Gate Elimination Method

## Gate Elimination

All the proofs are based on the so-called gate elimination method. This is essentially the only known method for proving lower bounds on circuit complexity.

## The main idea

- Take an optimal circuit for the function in question.
- Setting some variables to constants obtain a subfunction of the same type (in order to proceed by induction) and eliminate several gates.

# Gate Elimination Method

## Gate Elimination

All the proofs are based on the so-called gate elimination method. This is essentially the only known method for proving lower bounds on circuit complexity.

## The main idea

- Take an optimal circuit for the function in question.
- Setting some variables to constants obtain a subfunction of the same type (in order to proceed by induction) and eliminate several gates.
- A gate is eliminated if it computes a constant or a variable.

# Gate Elimination Method

## Gate Elimination

All the proofs are based on the so-called gate elimination method. This is essentially the only known method for proving lower bounds on circuit complexity.

## The main idea

- Take an optimal circuit for the function in question.
- Setting some variables to constants obtain a subfunction of the same type (in order to proceed by induction) and eliminate several gates.
- A gate is eliminated if it computes a constant or a variable.
- By repeatedly applying this process, conclude that the original circuit must have had many gates.

# Gate Elimination Method

## Gate Elimination

All the proofs are based on the so-called gate elimination method. This is essentially the only known method for proving lower bounds on circuit complexity.

## The main idea

- Take an optimal circuit for the function in question.
- Setting some variables to constants obtain a subfunction of the same type (in order to proceed by induction) and eliminate several gates.
- A gate is eliminated if it computes a constant or a variable.
- By repeatedly applying this process, conclude that the original circuit must have had many gates.

# Gate Elimination Method

## Gate Elimination

All the proofs are based on the so-called gate elimination method. This is essentially the only known method for proving lower bounds on circuit complexity.

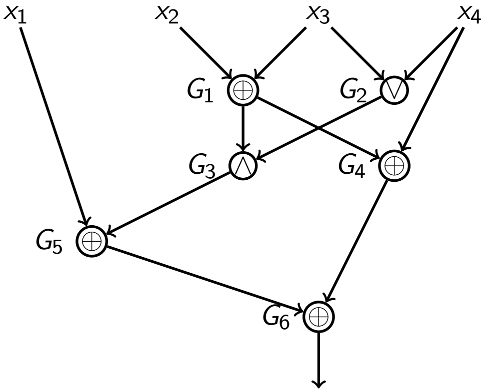
## The main idea

- Take an optimal circuit for the function in question.
- Setting some variables to constants obtain a subfunction of the same type (in order to proceed by induction) and eliminate several gates.
- A gate is eliminated if it computes a constant or a variable.
- By repeatedly applying this process, conclude that the original circuit must have had many gates.

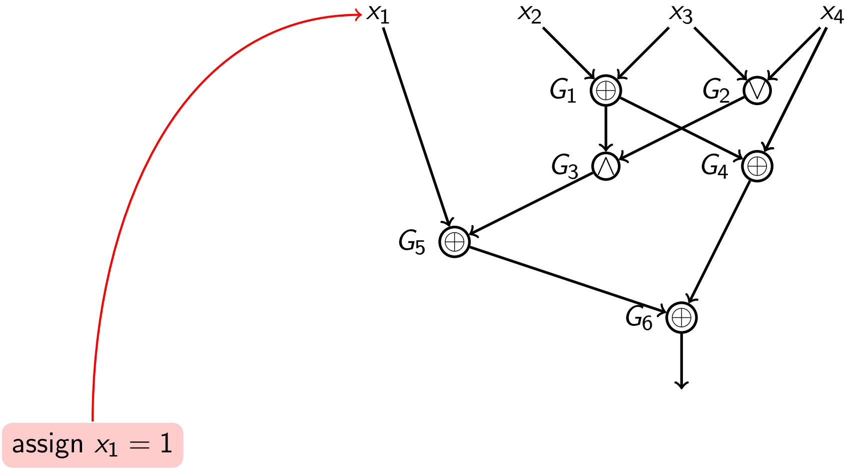
## Remark

This method is very unlikely to produce non-linear lower bounds.

# Example

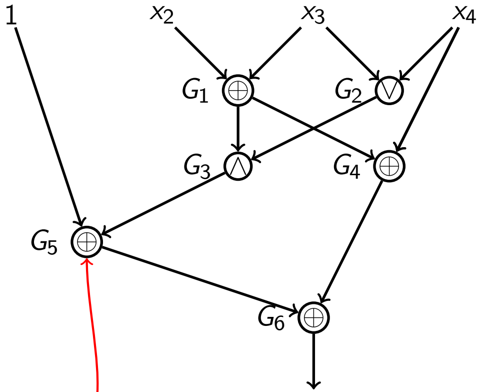


# Example



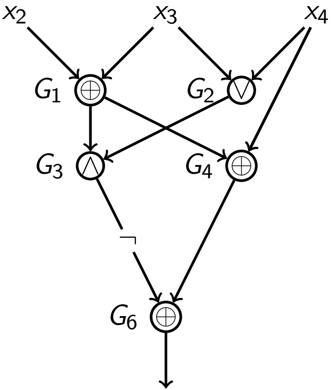
$x_1$
$x_2$
$x_3$
$x_4$
$G_1$ $\oplus$
$G_2$ $\vee$
$G_3$ $\wedge$
$G_4$ $\oplus$
$G_5$ $\oplus$
$G_6$ $\oplus$
assign $x_1 = 1$

# Example

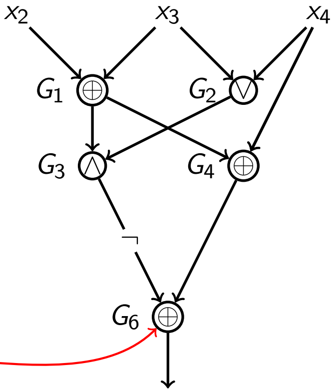



$G_5$ now computes $G_3 \oplus 1 = \neg G_3$

# Example

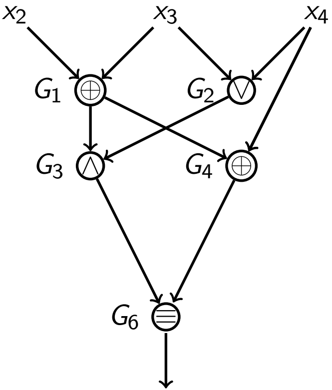


# Example

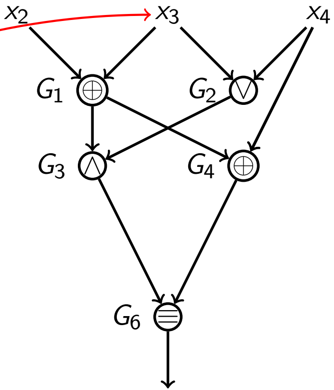



now we can change the binary function assigned to $G_6$

# Example

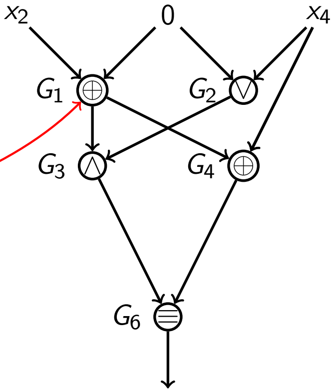


# Example



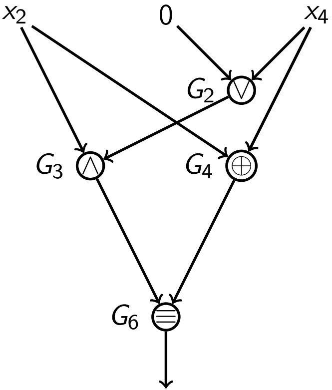

now assign $x_3 = 0$

# Example

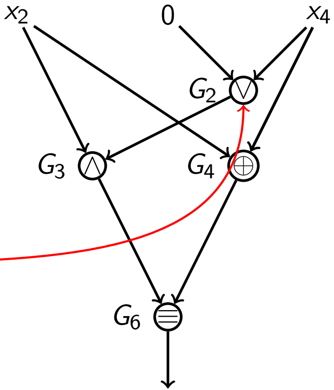



$G_1$ then is equal to $x_2$

# Example

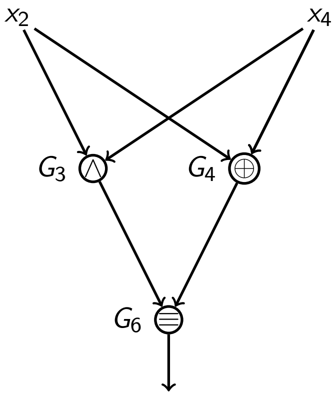


# Example

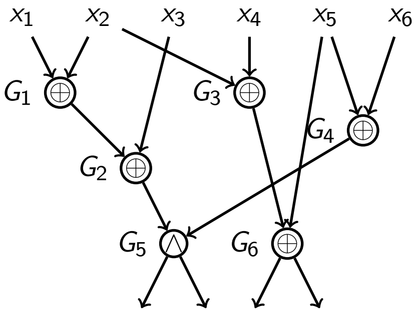



$G_2 = x_4$

# Example

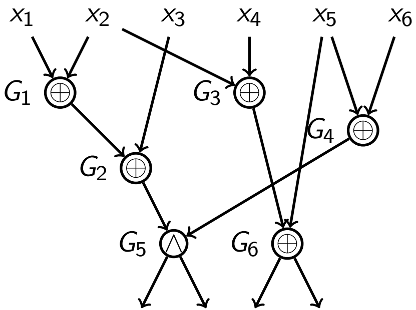


# A Typical Bottleneck



# A Typical Bottleneck

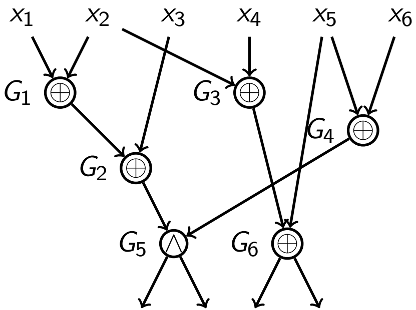
this is how a typical bottleneck case looks like



# A Typical Bottleneck

this is how a typical bottleneck case looks like

by assigning a variable we cannot kill more than 2 gates

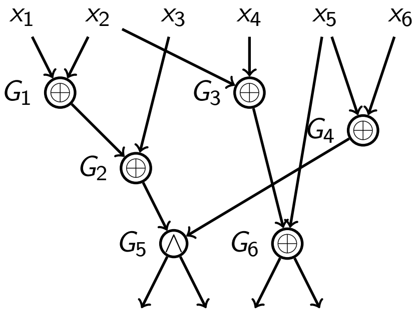


# A Typical Bottleneck

this is how a typical bottleneck case looks like

by assigning a variable we cannot kill more than 2 gates

at the same time we cannot exclude that a top of a circuit looks like this



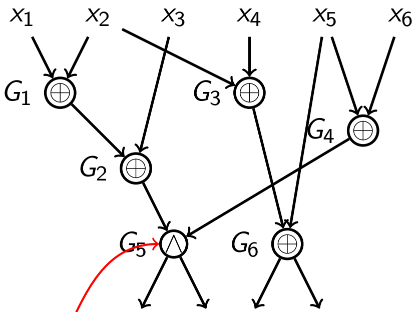
# A Typical Bottleneck

this is how a typical bottleneck case looks like

by assigning a variable we cannot kill more than 2 gates

at the same time we cannot exclude that a top of a circuit looks like this

consider a substitution $x_1 \oplus x_2 \oplus x_3 = 0$: under it $G_5$ trivializes



# Affine Dispersers

- OK, linear substitutions do help in gate elimination, but where is a function that survives under such substitutions?

# Affine Dispersers

- OK, linear substitutions do help in gate elimination, but where is a function that survives under such substitutions?
- Constructing a function that does not become a constant after any $n - o(n)$ linear substitutions is non-trivial. E.g., any symmetric function may be turned into a constant after $n/2$ linear substitutions: $x_1 \oplus x_2 = 1, x_3 \oplus x_4 = 1, \dots$.

# Affine Dispersers

- OK, linear substitutions do help in gate elimination, but where is a function that survives under such substitutions?
- Constructing a function that does not become a constant after any $n - o(n)$ linear substitutions is non-trivial. E.g., any symmetric function may be turned into a constant after $n/2$ linear substitutions: $x_1 \oplus x_2 = 1, x_3 \oplus x_4 = 1, \ldots$.
- An object that we are looking for is called an affine disperser.

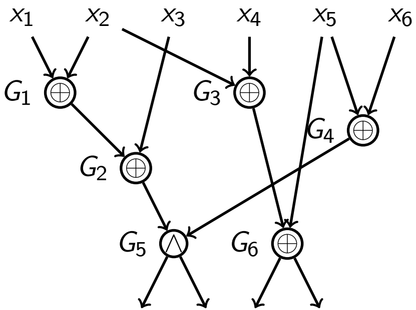
# Affine Dispersers

- OK, linear substitutions do help in gate elimination, but where is a function that survives under such substitutions?
- Constructing a function that does not become a constant after any $n - o(n)$ linear substitutions is non-trivial. E.g., any symmetric function may be turned into a constant after $n/2$ linear substitutions: $x_1 \oplus x_2 = 1, x_3 \oplus x_4 = 1, \ldots$.
- An object that we are looking for is called an affine disperser.
- Formally, an affine disperser for dimension $d$ is a function $f\colon \{0,1\}^n \to \{0,1\}$ that is not constant on any affine subspace of $\{0,1\}^n$ of dimension at least $d$.

# Affine Dispersers

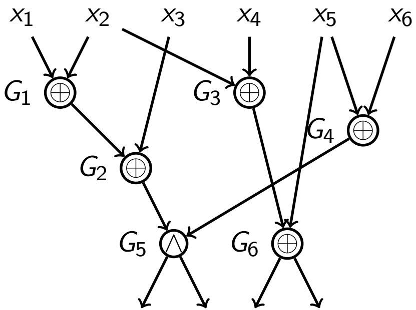
- OK, linear substitutions do help in gate elimination, but where is a function that survives under such substitutions?
- Constructing a function that does not become a constant after any $n - o(n)$ linear substitutions is non-trivial. E.g., any symmetric function may be turned into a constant after $n/2$ linear substitutions: $x_1 \oplus x_2 = 1, x_3 \oplus x_4 = 1, \ldots$.
- An object that we are looking for is called an affine disperser.
- Formally, an affine disperser for dimension $d$ is a function $f\colon \{0,1\}^n \to \{0,1\}$ that is not constant on any affine subspace of $\{0,1\}^n$ of dimension at least $d$.
- Only recently, an explicit affine disperser for $d = o(n)$ was constructed [Ben-Sasson and Kopparty, 09].

# Proof Idea



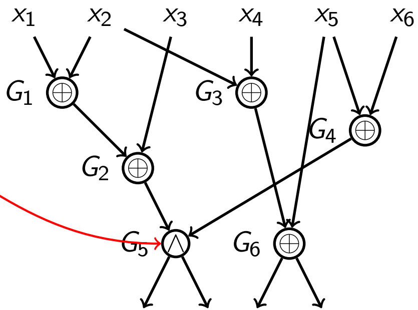
# Proof Idea

take the first gate which is not a XOR of out-degree 1



# Proof Idea

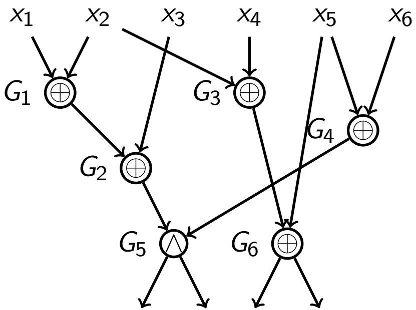
take the first gate which is not a XOR of out-degree 1



# Proof Idea

take the first gate which is not a XOR of out-degree 1

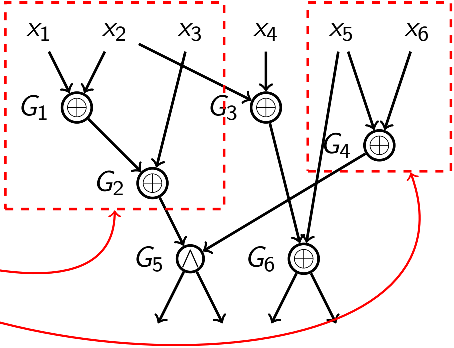
both its inputs compute linear functions



# Proof Idea

take the first gate which is not a XOR of out-degree 1

both its inputs compute linear functions

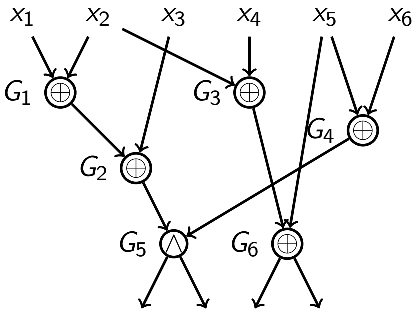


# Proof Idea

take the first gate which is not a XOR of out-degree 1

both its inputs compute linear functions

make a substitution
$x_1 \oplus x_2 \oplus x_3 \oplus x_5 \oplus x_6 = 1$



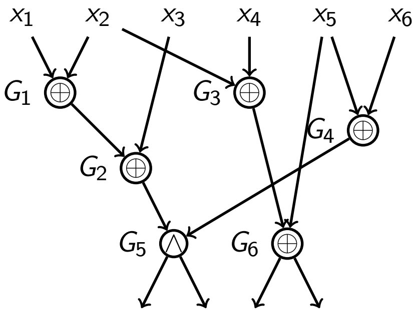
# Proof Idea

take the first gate which is not a XOR of out-degree 1

both its inputs compute linear functions

make a substitution
$x_1 \oplus x_2 \oplus x_3 \oplus x_5 \oplus x_6 = 1$

this kills the considered gate and all its successors



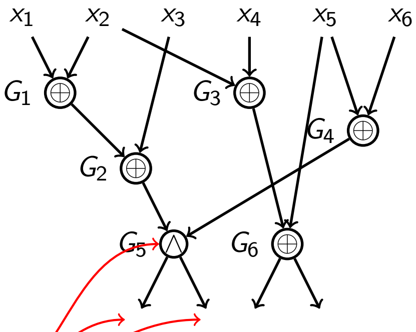
# Proof Idea

take the first gate which is not a XOR of out-degree 1

both its inputs compute linear functions

make a substitution
$x_1 \oplus x_2 \oplus x_3 \oplus x_5 \oplus x_6 = 1$

this kills the considered gate and all its successors



# Proof Idea

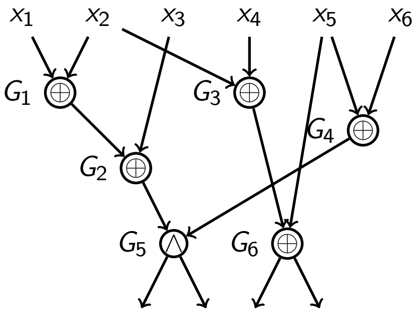
take the first gate which is not a XOR of out-degree 1

both its inputs compute linear functions

make a substitution $x_1 \oplus x_2 \oplus x_3 \oplus x_5 \oplus x_6 = 1$

this kills the considered gate and all its successors

moreover, all its predecessors are not needed any more



# Proof Idea

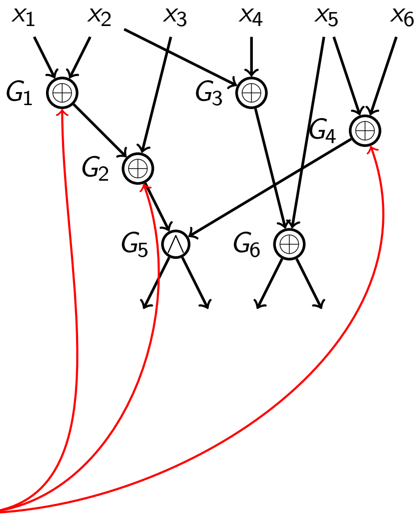
take the first gate which is not a XOR of out-degree 1

both its inputs compute linear functions

make a substitution
$x_1 \oplus x_2 \oplus x_3 \oplus x_5 \oplus x_6 = 1$

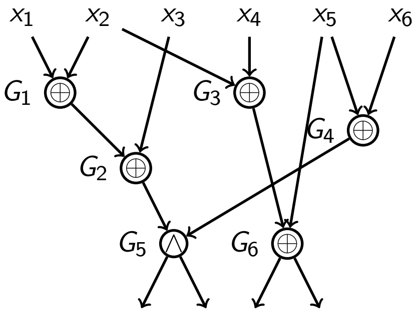
this kills the considered gate and all its successors

moreover, all its predecessors are not needed any more



# Proof Idea

by a short case analysis it is possible to show that this way one can always eliminate 3 gates; since we can make $n - o(n)$ such substitutions a lower bound $3n - o(n)$ follows



# Thank you for your attention!