The fusion method (AKA the ultraproduct)

Ondřej Ježil

December 9, 2020

Some context

- goal: Lower bounds (ideally) on non-monotone circuits

Some context

- goal: Lower bounds (ideally) on non-monotone circuits
- so far we've seen:

Some context

- goal: Lower bounds (ideally) on non-monotone circuits
- so far we've seen:
  - Razborov's approximation method

Some context

- goal: Lower bounds (ideally) on non-monotone circuits
- so far we've seen:
  - Razborov's approximation method
  - Sipser's topological approach

Some context

- goal: Lower bounds (ideally) on non-monotone circuits
- so far we've seen:
  - Razborov's approximation method
  - Sipser's topological approach
- These approaches were unified by M. Karchmer with his "Fusion method"

Some context

- goal: Lower bounds (ideally) on non-monotone circuits
- so far we've seen:
  - Razborov's approximation method
  - Sipser's topological approach
- These approaches were unified by M. Karchmer with his "Fusion method"
- we will cover the survey article: Avi Widgerson – The Fusion Method for Lower Bounds in Circuit Complexity

General idea

Lower bound for a boolean function $\rightarrow$ Combinatorial "covering" problem

General idea

Lower bound for a boolean function $\rightarrow$ Combinatorial "covering" problem

- Ultraproduct

General idea

Lower bound for a boolean function $\rightarrow$ Combinatorial "covering" problem

- Ultraproduct
  - We have a collection $(\mathcal{A}_i, i \in I)$ of structures, $\mathcal{A}_i \models T$.

General idea

Lower bound for a boolean function $\rightarrow$ Combinatorial "covering" problem

- Ultraproduct
  - We have a collection $(\mathcal{A}_i, i \in I)$ of structures, $\mathcal{A}_i \models T$.
  - If we have an ultrafilter $\mathcal{U}$ on I. We can form a new structure

$$\prod_{i \in I} \mathcal{A}_i / \mathcal{U} \models T.$$

General idea

Lower bound for a boolean function $\rightarrow$ Combinatorial "covering" problem

- Ultraproduct
  - We have a collection $(\mathcal{A}_i, i \in I)$ of structures, $\mathcal{A}_i \models T$.
  - If we have an ultrafilter $\mathcal{U}$ on I. We can form a new structure

$$\prod_{i \in I} \mathcal{A}_i / \mathcal{U} \models T.$$

- Fusing computations

General idea

Lower bound for a boolean function $\rightarrow$ Combinatorial "covering" problem

- Ultraproduct
  - We have a collection $(\mathcal{A}_i, i \in I)$ of structures, $\mathcal{A}_i \models T$.
  - If we have an ultrafilter $\mathcal{U}$ on I. We can form a new structure

$$\prod_{i \in I} \mathcal{A}_i / \mathcal{U} \models T.$$

- Fusing computations
  - We have some program P, accepting exactly $U \subseteq \{0,1\}^n$, and for each $u \in U$, we have $P(u)$ an accepting computation of u.

General idea

Lower bound for a boolean function $\rightarrow$ Combinatorial "covering" problem

- Ultraproduct
  - We have a collection $(\mathcal{A}_i, i \in I)$ of structures, $\mathcal{A}_i \models T$.
  - If we have an ultrafilter $\mathcal{U}$ on I. We can form a new structure

$$\prod_{i \in I} \mathcal{A}_i / \mathcal{U} \models T.$$

- Fusing computations
  - We have some program P, accepting exactly $U \subseteq \{0,1\}^n$, and for each $u \in U$, we have $P(u)$ an accepting computation of u.
  - If we have some finite analogue of an ultrafilter F, we can fuse them into a new "accepting computation" of some new z, a contradiction.

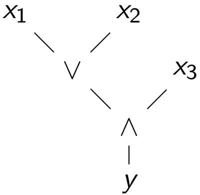
Straight-line programs, computations

Definition

Let $X = \{x_1, \ldots, x_k\}$ be a set of variables. A ***straight-line program*** *P is a tuple $(g_1, \ldots, g_t)$, such that $g_i = x_i$ for $i \in \{0, \ldots, n\}$ and $g_i = g_{i_1} \circ_i g_{i_2}$ where $i_1, i_2 < i$, and $\circ_i \in OP$ some set of binary operations.*
For $u \in \{0, 1\}^n$ we define a ***computation*** *of P on input u as $P(u) := (g_1(u), \ldots, g_t(u))$, where $g_t(u) \in \{0, 1\}$ is the output of the computation.*

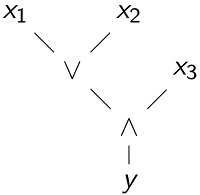
An example of a straight-line program

- Consider the circuit:



An example of a straight-line program

- Consider the circuit:

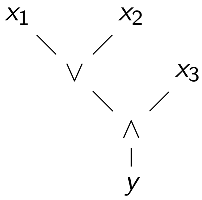


- The corresponding straight-line program is

$$P = (x_1, x_2, x_3, x_1 \vee x_2, (x_1 \vee x_2) \wedge x_3).$$

An example of a straight-line program

- Consider the circuit:



- The corresponding straight-line program is

$$P = (x_1, x_2, x_3, x_1 \vee x_2, (x_1 \vee x_2) \wedge x_3).$$

- And the following is an accepting computation of $P(1, 0, 1)$

$$P(1, 0, 1) = (1, 0, 1, 1, 1).$$

The fusion method

- Let $U \subseteq \{0,1\}^n$, we would like to find a lower bound on the length of the shortest straight-line program accepting exactly U.

The fusion method

- Let $U \subseteq \{0,1\}^n$, we would like to find a lower bound on the length of the shortest straight-line program accepting exactly U.
- This is equivalent to finding a lower bound for a straight-line program computing some boolean function f on n-letter strings by setting $U = f^{-1}[1]$.

The fusion method

- Let $U \subseteq \{0,1\}^n$, we would like to find a lower bound on the length of the shortest straight-line program accepting exactly U.
- This is equivalent to finding a lower bound for a straight-line program computing some boolean function f on n-letter strings by setting $U = f^{-1}[1]$.
- Assume for contradiction there exists some program $P = (g_1, \ldots, g_t)$ that accepts exactly U and t is too small.

The accepting computation matrix

- Consider a $|U| \times t$ matrix, where rows are indexed by U and each row is equal to the computation $P(u)$.

u					the rest of $P(u)$				
0	1	...	0	1	0	1	...	0	1
0	0	...	1	1	0	0	...	0	1
1	0	...	0	1	1	0	...	1	1
1	0	...	1	0	1	0	...	1	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	1	...	1	0	0	0	...	0	1

Producing a contradiction

u					the rest of $P(u)$				
0	1	...	0	1	0	1	...	0	1
0	0	...	1	1	0	0	...	0	1
1	0	...	0	1	1	0	...	1	1
1	0	...	1	0	1	0	...	1	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	1	...	1	0	0	0	...	0	1

- We would like to produce a contradiction using that the number of rows t is too small.

Producing a contradiction

u					the rest of $P(u)$				
0	1	...	0	1	0	1	...	0	1
0	0	...	1	1	0	0	...	0	1
1	0	...	0	1	1	0	...	1	1
1	0	...	1	0	1	0	...	1	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	1	...	1	0	0	0	...	0	1

- We would like to produce a contradiction using that the number of rows t is too small.
- We will try to construct a "new" accepting computation using the old ones. Since this table contains all accepting computations, this would be a contradiction.

Fusing the computations

- How to produce the new computation?

Fusing the computations

- How to produce the new computation?
- Let $F\colon \{0,1\}^{|U|} \to \{0,1\}$, "a functional" from some set Ω of functionals (will be specified later, e.g. $\Omega = \{\text{all functionals}\}$ works).

Fusing the computations

- How to produce the new computation?
- Let $F\colon \{0,1\}^{|U|} \to \{0,1\}$, "a functional" from some set Ω of functionals (will be specified later, e.g. $\Omega = \{\text{all functionals}\}$ works).
- F will act as our finite analogue of an ultrafilter.

Applying the functional

0	1	...	0	1	0	1	...	0	1
0	0	...	1	1	0	0	...	0	1
1	0	...	0	1	1	0	...	1	1
1	0	...	1	0	1	0	...	1	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	1	...	1	0	0	0	...	0	1

$\downarrow_F$

0	

Applying the functional

0	1	...	0	1	0	1	...	0	1
0	0	...	1	1	0	0	...	0	1
1	0	...	0	1	1	0	...	1	1
1	0	...	1	0	1	0	...	1	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	1	...	1	0	0	0	...	0	1

$\downarrow F$

0	1	

Applying the functional

...

Applying the functional

0	1	...	0	1	0	1	...	0	1
0	0	...	1	1	0	0	...	0	1
1	0	...	0	1	1	0	...	1	1
1	0	...	1	0	1	0	...	1	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	1	...	1	0	0	0	...	0	1

$\downarrow F$

0	1	...	1	1	1	0	...	1	1

Requirements on the functional

- It is obvious there is no guarantee that the resulting tuple will be an accepting computation.

Requirements on the functional

- It is obvious there is no guarantee that the resulting tuple will be an accepting computation.
- By $F(g_i)$ we mean the output of F on i-th column of the accepting computation matrix.

Requirements on the functional

- It is obvious there is no guarantee that the resulting tuple will be an accepting computation.
- By $F(g_i)$ we mean the output of F on i-th column of the accepting computation matrix.
- There are three requirements on the functional F for this to work:

Requirements on the functional

- It is obvious there is no guarantee that the resulting tuple will be an accepting computation.
- By $F(g_i)$ we mean the output of F on i-th column of the accepting computation matrix.
- There are three requirements on the functional F for this to work:
  1. F "encodes" some $z \notin U$, that is, $F(g_i) = z_i$ for $i \in \{1, \dots, n\}$ (the "u" part of the new row is z)

Requirements on the functional

- It is obvious there is no guarantee that the resulting tuple will be an accepting computation.
- By $F(g_i)$ we mean the output of F on i-th column of the accepting computation matrix.
- There are three requirements on the functional F for this to work:
  1. F "encodes" some $z \notin U$, that is, $F(g_i) = z_i$ for $i \in \{1, \ldots, n\}$ (the "u" part of the new row is z)
  2. The resulting computation is accepting, that is $F(\bar{1}) = 1$

Requirements on the functional

- It is obvious there is no guarantee that the resulting tuple will be an accepting computation.
- By $F(g_i)$ we mean the output of F on i-th column of the accepting computation matrix.
- There are three requirements on the functional F for this to work:
  1. F "encodes" some $z \notin U$, that is, $F(g_i) = z_i$ for $i \in \{1, \ldots, n\}$ (the "u" part of the new row is z)
  2. The resulting computation is accepting, that is $F(\bar{1}) = 1$
  3. F is consistent, that is $F(g_{i_1}) \circ_i F(g_{i_2}) = F(g_{i_1} \circ_i g_{i_2})$ for $n < i \leq t$

Requirements on the functional

- It is obvious there is no guarantee that the resulting tuple will be an accepting computation.
- By $F(g_i)$ we mean the output of F on i-th column of the accepting computation matrix.
- There are three requirements on the functional F for this to work:
  1. F "encodes" some $z \notin U$, that is, $F(g_i) = z_i$ for $i \in \{1, \ldots, n\}$ (the "u" part of the new row is z)
  2. The resulting computation is accepting, that is $F(\bar{1}) = 1$
  3. F is consistent, that is $F(g_{i_1}) \circ_i F(g_{i_2}) = F(g_{i_1} \circ_i g_{i_2})$ for $n < i \leq t$
- We will search for such F by considering

$$\Omega_f = \{F \in \Omega; F \text{ satisfies the first two points}\}.$$

Requirements on the functional cont.

- We will search for such F by considering

$$\Omega_f = \{F \in \Omega; F \text{ satisfies the first two points}\}.$$

Requirements on the functional cont.

- We will search for such F by considering

$$\Omega_f = \{F \in \Omega; F \text{ satisfies the first two points}\}.$$

- How do we find functional in Ω_f that satisfies the third requirement, since it depends on P?

Requirements on the functional cont.

- We will search for such F by considering

$$\Omega_f = \{F \in \Omega; F \text{ satisfies the first two points}\}.$$

- How do we find functional in Ω_f that satisfies the third requirement, since it depends on P?
- We don't! We just conclude that if such short P exists, there has to be no such functional in Ω_f.

Covering

Definition

Let OP be some set of operations. We say, that the triple $(g, h, \circ)$, $g, h \in \{0,1\}^n \to \{0,1\}, \circ \in OP$ ***covers*** *a functional F, if*

$$F(g) \circ F(h) \neq F(g \circ h).$$

For a function $f\colon \{0,1\}^n \to \{0,1\}$ *we denote* $\rho(f)$ *the smallest number of such triples that cover* Ω_f.

The lower bound

Theorem (Meta-theorem)

$\rho(f)$ is a lower bound on the shortest straight-line program computing f over OP.

The lower bound

Theorem (Meta-theorem)

$\rho(f)$ is a lower bound on the shortest straight-line program computing f over OP.

Proof.

Let $P = (g_1, \ldots, g_t)$ be a program computing f and $t < \rho(f)$. Since $\{(g_{i_1}, g_{i_2}, \circ_i); i \in \{n+1, \ldots, t\}\}$ cannot cover Ω_f, therefore there does exists $F \in \Omega_f$ that is consistent with this program. F then codes a new accepting computation of some $z \notin f^{-1}[1]$, which is a contradiction. □

The lower bound

Theorem (Meta-theorem)

$\rho(f)$ is a lower bound on the shortest straight-line program computing f over OP.

Proof.

Let $P = (g_1, \dots, g_t)$ be a program computing f and $t < \rho(f)$. Since $\{(g_{i_1}, g_{i_2}, \circ_i); i \in \{n+1, \dots, t\}\}$ cannot cover Ω_f, therefore there does exists $F \in \Omega_f$ that is consistent with this program. F then codes a new accepting computation of some $z \notin f^{-1}[1]$, which is a contradiction. □

- The lower bound is actually $n + \rho(f)$.

The lower bound

Theorem (Meta-theorem)

$\rho(f)$ is a lower bound on the shortest straight-line program computing f over OP.

Proof.

Let $P = (g_1, \ldots, g_t)$ be a program computing f and $t < \rho(f)$. Since $\{(g_{i_1}, g_{i_2}, \circ_i); i \in \{n+1, \ldots, t\}\}$ cannot cover Ω_f, therefore there does exists $F \in \Omega_f$ that is consistent with this program. F then codes a new accepting computation of some $z \notin f^{-1}[1]$, which is a contradiction. □

- The lower bound is actually $n + \rho(f)$.
- We can restrict the smallest cover to those covers for which each $(g, h, \circ)$ has g, h definable by some straight line program over OP.

Example - parity

- Let $f(x_1, x_2) = (x_1 + x_2) \bmod 2$, let $\mathrm{OP} = \{\wedge, \vee, \neg\}$.

Example - parity

- Let $f(x_1, x_2) = (x_1 + x_2) \bmod 2$, let $\mathrm{OP} = \{\wedge, \vee, \neg\}$.
- $\neg$ is not a binary operation but we can define it as $\neg(g_{i_1}, g_{i_2}) = \neg g_{i_1}$.

Example - parity

- Let $f(x_1, x_2) = (x_1 + x_2) \bmod 2$, let $\mathrm{OP} = \{\wedge, \vee, \neg\}$.
- $\neg$ is not a binary operation but we can define it as $\neg(g_{i_1}, g_{i_2}) = \neg g_{i_1}$.
- The accepting computation matrix for any program P is

$P(\mathbf{u}_1)$:	0	1	...	1
$P(\mathbf{u}_2)$:	1	0	...	1

Example - parity

- Let $f(x_1, x_2) = (x_1 + x_2) \bmod 2$, let $\mathrm{OP} = \{\wedge, \vee, \neg\}$.
- $\neg$ is not a binary operation but we can define it as $\neg(g_{i_1}, g_{i_2}) = \neg g_{i_1}$.
- The accepting computation matrix for any program P is

$P(\mathbf{u}_1)$:	0	1	...	1
$P(\mathbf{u}_2)$:	1	0	...	1

- For Ω unrestricted, what do we have in Ω_f? We have:

g:	$\mathbf{0}$	x_1	x_2	$\mathbf{1}$
$g(\mathbf{u}_1)$	0	0	1	1
$g(\mathbf{u}_2)$	0	1	0	1
$F_1(g)$	0	0	0	1
$F_2(g)$	0	1	1	1
$F_3(g)$	1	0	0	1
$F_4(g)$	1	1	1	1

Example - parity

- Let $f(x_1, x_2) = (x_1 + x_2) \bmod 2$, let $\mathrm{OP} = \{\wedge, \vee, \neg\}$.
- $\neg$ is not a binary operation but we can define it as $\neg(g_{i_1}, g_{i_2}) = \neg g_{i_1}$.
- The accepting computation matrix for any program P is

$P(\mathbf{u}_1)$:	0	1	...	1
$P(\mathbf{u}_2)$:	1	0	...	1

- For Ω unrestricted, what do we have in Ω_f? We have:

g:	**0**	x_1	x_2	**1**
$g(\mathbf{u}_1)$	0	0	1	1
$g(\mathbf{u}_2)$	0	1	0	1
$F_1(g)$	0	0	0	1
$F_2(g)$	0	1	1	1
$F_3(g)$	1	0	0	1
$F_4(g)$	1	1	1	1

- The rows of the two middle columns have to differ from the first two rows because of requirement 1.

Example - parity

- Let $f(x_1, x_2) = (x_1 + x_2) \bmod 2$, let $\mathrm{OP} = \{\wedge, \vee, \neg\}$.
- $\neg$ is not a binary operation but we can define it as $\neg(g_{i_1}, g_{i_2}) = \neg g_{i_1}$.
- The accepting computation matrix for any program P is

$P(\mathbf{u}_1)$:	0	1	...	1
$P(\mathbf{u}_2)$:	1	0	...	1

- For Ω unrestricted, what do we have in Ω_f? We have:

g:	**0**	x_1	x_2	**1**
$g(\mathbf{u}_1)$	0	0	1	1
$g(\mathbf{u}_2)$	0	1	0	1
$F_1(g)$	0	0	0	1
$F_2(g)$	0	1	1	1
$F_3(g)$	1	0	0	1
$F_4(g)$	1	1	1	1

- The rows of the two middle columns have to differ from the first two rows because of requirement 1.
- The last column contains only ones because of requirement 2.

Example - parity cont.

- We need to cover the following four functionals.

g:	$\mathbf{0}$	x_1	x_2	$\mathbf{1}$
$g(\mathbf{u}_1)$	0	0	1	1
$g(\mathbf{u}_2)$	0	1	0	1
$F_1(g)$	0	0	0	1
$F_2(g)$	0	1	1	1
$F_3(g)$	1	0	0	1
$F_4(g)$	1	1	1	1

Example - parity cont.

- We need to cover the following four functionals.

g:	$\mathbf{0}$	x_1	x_2	$\mathbf{1}$
$g(\mathbf{u}_1)$	0	0	1	1
$g(\mathbf{u}_2)$	0	1	0	1
$F_1(g)$	0	0	0	1
$F_2(g)$	0	1	1	1
$F_3(g)$	1	0	0	1
$F_4(g)$	1	1	1	1

- F_1 is covered by $(x_1, x_2, \vee)$, since $F_1(x_1) \vee F_1(x_2) = 0$, but $F_1(x_1 \vee x_2) = F_1(\mathbf{1}) = 1$

Example - parity cont.

- We need to cover the following four functionals.

g:	$\mathbf{0}$	x_1	x_2	$\mathbf{1}$
$g(\mathbf{u}_1)$	0	0	1	1
$g(\mathbf{u}_2)$	0	1	0	1
$F_1(g)$	0	0	0	1
$F_2(g)$	0	1	1	1
$F_3(g)$	1	0	0	1
$F_4(g)$	1	1	1	1

- F_1 is covered by $(x_1, x_2, \vee)$, since $F_1(x_1) \vee F_1(x_2) = 0$, but $F_1(x_1 \vee x_2) = F_1(\mathbf{1}) = 1$
- F_2 is covered by $(x_1, x_2, \wedge)$, since $F_2(x_1) \wedge F_2(x_2) = 1$, but $F_2(x_1 \wedge x_2) = F_2(\mathbf{0}) = 0$

Example - parity cont.

- We need to cover the following four functionals.

g:	$\mathbf{0}$	x_1	x_2	$\mathbf{1}$
$g(\mathbf{u}_1)$	0	0	1	1
$g(\mathbf{u}_2)$	0	1	0	1
$F_1(g)$	0	0	0	1
$F_2(g)$	0	1	1	1
$F_3(g)$	1	0	0	1
$F_4(g)$	1	1	1	1

- F_1 is covered by $(x_1, x_2, \vee)$, since $F_1(x_1) \vee F_1(x_2) = 0$, but $F_1(x_1 \vee x_2) = F_1(\mathbf{1}) = 1$
- F_2 is covered by $(x_1, x_2, \wedge)$, since $F_2(x_1) \wedge F_2(x_2) = 1$, but $F_2(x_1 \wedge x_2) = F_2(\mathbf{0}) = 0$
- F_3 is covered by $(x_1, -, \neg)$, since $\neg F_3(x_1) = 1$, but $F_3(\neg x_1) = F_3(x_2) = 0$ and so is F_4

Example - parity cont.

- We need to cover the following four functionals.

g:	$\mathbf{0}$	x_1	x_2	$\mathbf{1}$
$g(\mathbf{u}_1)$	0	0	1	1
$g(\mathbf{u}_2)$	0	1	0	1
$F_1(g)$	0	0	0	1
$F_2(g)$	0	1	1	1
$F_3(g)$	1	0	0	1
$F_4(g)$	1	1	1	1

- F_1 is covered by $(x_1, x_2, \vee)$, since $F_1(x_1) \vee F_1(x_2) = 0$, but $F_1(x_1 \vee x_2) = F_1(\mathbf{1}) = 1$
- F_2 is covered by $(x_1, x_2, \wedge)$, since $F_2(x_1) \wedge F_2(x_2) = 1$, but $F_2(x_1 \wedge x_2) = F_2(\mathbf{0}) = 0$
- F_3 is covered by $(x_1, -, \neg)$, since $\neg F_3(x_1) = 1$, but $F_3(\neg x_1) = F_3(x_2) = 0$ and so is F_4
- This is the smallest possible cover using OP, therefore the lower bound is $2 + 3 = 5$.

Quality of the lower bound

- Why should we consider Ω restricted to some type of functionals?

Quality of the lower bound

- Why should we consider Ω restricted to some type of functionals?
  - Full Ω is **huge**, $|\Omega| = 2^{2^{|U|}}$ and $|U| = \mathcal{O}(2^n)$. So covering only part of it can be much more managable.

Quality of the lower bound

- Why should we consider Ω restricted to some type of functionals?
  - Full Ω is **huge**, $|\Omega| = 2^{2^{|U|}}$ and $|U| = \mathcal{O}(2^n)$. So covering only part of it can be much more managable.
  - While considering unrestricted Ω we can obtain a larger lower bound. However in some situations for some restrictions we get the following theorem:

Meta-Converse

Theorem (Meta-Converse)

There is a program P over OP that computes f that is not much larger than $\rho(f)$.

Meta-Converse

Theorem (Meta-Converse)

There is a program P over OP that computes f that is not much larger than $\rho(f)$.

Proof.

(sketch) We have a cover $C = \{(g_1, h_1, \circ_1), \ldots, (g_t, h_t, \circ_t)\}$.

Meta-Converse

Theorem (Meta-Converse)

There is a program P over OP that computes f that is not much larger than $\rho(f)$.

Proof.

(sketch) We have a cover $C = \{(g_1, h_1, \circ_1), \ldots, (g_t, h_t, \circ_t)\}$.

Meta-Converse

Theorem (Meta-Converse)

There is a program P over OP that computes f that is not much larger than $\rho(f)$.

Proof.

(sketch) We have a cover $C = \{(g_1, h_1, \circ_1), \ldots, (g_t, h_t, \circ_t)\}$.This is not a program, and our task is to "organize" these unrelated gates into a program.

Meta-Converse

Theorem (Meta-Converse)

There is a program P over OP that computes f that is not much larger than $\rho(f)$.

Proof.

(sketch) We have a cover $C = \{(g_1, h_1, \circ_1), \ldots, (g_t, h_t, \circ_t)\}$.This is not a program, and our task is to "organize" these unrelated gates into a program.

Claim: $f(z) = 1 \Leftrightarrow \exists F \in \Omega$ that defines z and is not covered with C.

Meta-Converse

Theorem (Meta-Converse)

There is a program P over OP that computes f that is not much larger than $\rho(f)$.

Proof.

(sketch) We have a cover $C = \{(g_1, h_1, \circ_1), \ldots, (g_t, h_t, \circ_t)\}$.This is not a program, and our task is to "organize" these unrelated gates into a program.

Claim: $f(z) = 1 \Leftrightarrow \exists F \in \Omega$ that defines z and is not covered with C.
proof of the claim: For "$\Rightarrow$" pick $F_z(g) := g(z)$. This is by definition compatible with every operation.

Meta-Converse

Theorem (Meta-Converse)

There is a program P over OP that computes f that is not much larger than $\rho(f)$.

Proof.

(sketch) We have a cover $C = \{(g_1, h_1, \circ_1), \ldots, (g_t, h_t, \circ_t)\}$.This is not a program, and our task is to "organize" these unrelated gates into a program.

Claim: $f(z) = 1 \Leftrightarrow \exists F \in \Omega$ that defines z and is not covered with C.

proof of the claim: For "$\Rightarrow$" pick $F_z(g) := g(z)$. This is by definition compatible with every operation.

"$\Leftarrow$" has been already proven as a part of the Main theorem. □

Meta-Converse

Theorem (Meta-Converse)

There is a program P over OP that computes f that is not much larger than $\rho(f)$.

Proof.

(sketch) We have a cover $C = \{(g_1, h_1, \circ_1), \ldots, (g_t, h_t, \circ_t)\}$.This is not a program, and our task is to "organize" these unrelated gates into a program.

Claim: $f(z) = 1 \Leftrightarrow \exists F \in \Omega$ that defines z and is not covered with C.
proof of the claim: For "$\Rightarrow$" pick $F_z(g) := g(z)$. This is by definition compatible with every operation.
"$\Leftarrow$" has been already proven as a part of the Main theorem. □

With the claim, we just need to construct a program, that tries to find such F. We don't need the whole functional, just its values on x_i and the cover. For many choices of OP and Ω this yields program, that has either linear or polynomial length with respect to $\rho(f)$. □

The choices for Ω

- The following restrictions for Ω have been considered:

The choices for Ω

- The following restrictions for Ω have been considered:
  - $\Omega = \{F; F \text{ is a filter}\}$, where a filter F is a functional, that is monotone (flipping zeroes in the input can only make the output 1)

The choices for Ω

- The following restrictions for Ω have been considered:
  - $\Omega = \{F; F \text{ is a filter}\}$, where a filter F is a functional, that is monotone (flipping zeroes in the input can only make the output 1)
  - $\Omega = \{F; F \text{ is a filter}\}$, where a filter F is a functional, that is monotone (flipping zeroes in the input can only make the output 1)

The unification - Razborov's work

- In 1985 Razborov proved superpolynomial lower bounds on monotone circuit size for the clique and matching functions using his "approximation method"

The unification - Razborov's work

- In 1985 Razborov proved superpolynomial lower bounds on monotone circuit size for the clique and matching functions using his "approximation method"
- What about lower bounds for non-monotone circuits?

The unification - Razborov's work

- In 1985 Razborov proved superpolynomial lower bounds on monotone circuit size for the clique and matching functions using his "approximation method"
- What about lower bounds for non-monotone circuits?
- In 1989 Razborov formalized his approximation method and proved it cannot provide superlinear lower bounds for non-monotone circuits.

The unification - Razborov's work

- In 1985 Razborov proved superpolynomial lower bounds on monotone circuit size for the clique and matching functions using his "approximation method"
- What about lower bounds for non-monotone circuits?
- In 1989 Razborov formalized his approximation method and proved it cannot provide superlinear lower bounds for non-monotone circuits.
- However, he proposed a generalization of this method and proved that it actually characterizes circuit size. So it can be used to prove lower bounds for non-monotone circuits.

The unification - Razborov's work

- In 1985 Razborov proved superpolynomial lower bounds on monotone circuit size for the clique and matching functions using his "approximation method"
- What about lower bounds for non-monotone circuits?
- In 1989 Razborov formalized his approximation method and proved it cannot provide superlinear lower bounds for non-monotone circuits.
- However, he proposed a generalization of this method and proved that it actually characterizes circuit size. So it can be used to prove lower bounds for non-monotone circuits.
- What we've seen so far is actually his "generalized approximation method", in this point of view, F is seen as an approximation of a gate.

The unification - Razborov's work

- In 1985 Razborov proved superpolynomial lower bounds on monotone circuit size for the clique and matching functions using his "approximation method"
- What about lower bounds for non-monotone circuits?
- In 1989 Razborov formalized his approximation method and proved it cannot provide superlinear lower bounds for non-monotone circuits.
- However, he proposed a generalization of this method and proved that it actually characterizes circuit size. So it can be used to prove lower bounds for non-monotone circuits.
- What we've seen so far is actually his "generalized approximation method", in this point of view, F is seen as an approximation of a gate.
- 1990 Razborov proved that somewhat restricted can be associeted with non-deterministic branching programs, an proved a super-linear lower bound for the Majority function.

The unification - Sipser's work

- On the other hand, in the early 1980's Sipser proposed that we should use infinite analogue of circuits used in topology to guide our intuition.

The unification - Sipser's work

- On the other hand, in the early 1980's Sipser proposed that we should use infinite analogue of circuits used in topology to guide our intuition.
- We've seen his new proof of separation co-analytic sets from analytic sets.

The unification - Sipser's work

- On the other hand, in the early 1980's Sipser proposed that we should use infinite analogue of circuits used in topology to guide our intuition.
- We've seen his new proof of separation co-analytic sets from analytic sets.
- T the set of well founded trees is easily co-analytic, but Sipser proved that is it not analytic, by taking a sequence $t_1, t_2, \dots \in T$ that converges to $t_\infty \notin T$. Which would any analytic circuit would have to accept as well.

The unification - Sipser's work

- On the other hand, in the early 1980's Sipser proposed that we should use infinite analogue of circuits used in topology to guide our intuition.
- We've seen his new proof of separation co-analytic sets from analytic sets.
- T the set of well founded trees is easily co-analytic, but Sipser proved that is it not analytic, by taking a sequence $t_1, t_2, \dots \in T$ that converges to $t_\infty \notin T$. Which would any analytic circuit would have to accept as well.
- In his 1984 paper Sipser asks for a finite analogue of a limit that will allow us to carry out such arguments in the finite world.

The unification - Sipser's work

- On the other hand, in the early 1980's Sipser proposed that we should use infinite analogue of circuits used in topology to guide our intuition.
- We've seen his new proof of separation co-analytic sets from analytic sets.
- T the set of well founded trees is easily co-analytic, but Sipser proved that is it not analytic, by taking a sequence $t_1, t_2, \dots \in T$ that converges to $t_\infty \notin T$. Which would any analytic circuit would have to accept as well.
- In his 1984 paper Sipser asks for a finite analogue of a limit that will allow us to carry out such arguments in the finite world.
- This should remind us of Ω a finite notion of a limit, and F a notion of a converging sequence.

The unification - Karchmer's work

- Karchmeri, in his 1993 paper, was the first one to describe the fusion method in a way that was presented earlier. He observed, that it generalizes the previous efforts.

The unification - Karchmer's work

- Karchmeri, in his 1993 paper, was the first one to describe the fusion method in a way that was presented earlier. He observed, that it generalizes the previous efforts.
- He noted, that this method can be viewed as a finitary version of an ultraproduct. This idea was pushed even further by Ben-David, Karchmer and Kushilevitz who have showed that standard ultra-filter arguments can simplify Sipser's proof.

The unification - Karchmer's work cont. 1

- In his 1993 he has also proved three characterization results.

The unification - Karchmer's work cont. 1

- In his 1993 he has also proved three characterization results.
- First note that here we are considering inputs as both positive and negative literals.

The unification - Karchmer's work cont. 1

- In his 1993 he has also proved three characterization results.
- First note that here we are considering inputs as both positive and negative literals.
- Choosing $\Omega := \{F; F \text{ is a filter (a monotone functional)}\}$ results in the following characterization of P:

The unification - Karchmer's work cont. 1

- In his 1993 he has also proved three characterization results.
- First note that here we are considering inputs as both positive and negative literals.
- Choosing $\Omega := \{F; F$ is a filter (a monotone functional)$\}$ results in the following characterization of P:

Theorem (Characterization of $\mathbf{P}$)

$f \in \boldsymbol{P}$ if and only if $\rho(\Omega_f) \leq p(n)$ for some polynomial p.

The unification - Karchmer's work cont. 2

- Now consider: $\Omega' := \{F; F \text{ a self dual filter}\}$, that is a set of filters, that contain each string or its negation.

The unification - Karchmer's work cont. 2

- Now consider: $\Omega' := \{F; F \text{ a self dual filter}\}$, that is a set of filters, that contain each string or its negation.
- Note that Ω' contains more than just ultrafilters.

The unification - Karchmer's work cont. 2

- Now consider: $\Omega' := \{F; F \text{ a self dual filter}\}$, that is a set of filters, that contain each string or its negation.
- Note that Ω' contains more than just ultrafilters.
- We have the following result:

The unification - Karchmer's work cont. 2

- Now consider: $\Omega' := \{F; F \text{ a self dual filter}\}$, that is a set of filters, that contain each string or its negation.
- Note that Ω' contains more than just ultrafilters.
- We have the following result:

Theorem (Characterization of NP)

$f \in \textbf{NP}$ if and only if $\rho(\Omega'_f) \leq p(n)$ for some polynomial p.

The unification - Karchmer's work cont. 3

- Again choosing $\Omega := \{F; F \text{ is a filter (a monotone functional)}\}$, but restricting inputs to positive literals, results in the following characterization:

The unification - Karchmer's work cont. 3

- Again choosing $\Omega := \{F; F \text{ is a filter (a monotone functional)}\}$, but restricting inputs to positive literals, results in the following characterization:

Theorem (Characterization of $m\mathbf{P}$)

$f \in m\mathbf{P}$ if and only if $\rho_+(\Omega_f) \leq p(n)$ for some polynomial p.

The unification - Karchmer's work cont. 3

- Again choosing $\Omega := \{F; F \text{ is a filter (a monotone functional)}\}$, but restricting inputs to positive literals, results in the following characterization:

Theorem (Characterization of $m\mathbf{P}$)

$f \in m\mathbf{P}$ if and only if $\rho_+(\Omega_f) \leq p(n)$ for some polynomial p.

- Karchmer used this to give a new proof of Razborov's super-polynomial lower bound for the monotone clique.

Algebraic variants

- $(\{0,1\}^n, \wedge, \vee, (\neg))$ are precisely finite Boolean algebras, and a filter is a natural notion for these structures, that can give some intuition on the choice $\Omega = \{\text{filters}\}$

Algebraic variants

- $(\{0,1\}^n, \wedge, \vee, (\neg))$ are precisely finite Boolean algebras, and a filter is a natural notion for these structures, that can give some intuition on the choice $\Omega = \{\text{filters}\}$
- $(\{0,1\}^n, \wedge, \oplus)$ are precisely finite arithmetical vector spaces over GF(2), what is a "natural" choice for Ω here?

Algebraic variants

- $(\{0,1\}^n, \wedge, \vee, (\neg))$ are precisely finite Boolean algebras, and a filter is a natural notion for these structures, that can give some intuition on the choice $\Omega = \{\text{filters}\}$
- $(\{0,1\}^n, \wedge, \oplus)$ are precisely finite arithmetical vector spaces over GF(2), what is a "natural" choice for Ω here?

Algebraic variants

- $(\{0,1\}^n, \wedge, \vee, (\neg))$ are precisely finite Boolean algebras, and a filter is a natural notion for these structures, that can give some intuition on the choice $\Omega = \{\text{filters}\}$
- $(\{0,1\}^n, \wedge, \oplus)$ are precisely finite arithmetical vector spaces over GF(2), what is a "natural" choice for Ω here? $\Omega = \{\text{affine}\}$, this also results in a characterization.

Algebraic variants

- $(\{0,1\}^n, \wedge, \vee, (\neg))$ are precisely finite Boolean algebras, and a filter is a natural notion for these structures, that can give some intuition on the choice $\Omega = \{\text{filters}\}$
- $(\{0,1\}^n, \wedge, \oplus)$ are precisely finite arithmetical vector spaces over GF(2), what is a "natural" choice for Ω here? $\Omega = \{\text{affine}\}$, this also results in a characterization.
- Notice, that the whole fusion method does not depend on that the values of our functions are just $\{0,1\}$, if instead we consider functions over some ring R, this whole method works for proving lower bound on their algebraic circuit complexity.

Table of results

Inputs	Gates	Type	Mode	Ω	$\mathcal{C}_\Delta$	Upper bound
$X \cup \overline{X}$	$\{\vee, \wedge\}$	Circuit	Det.	Filters	**P**	$(\rho_\Gamma(f))^C$
$X \cup \overline{X}$	$\{\vee, \wedge\}$	BP	Det.	Filters	**NL**	$C \cdot \rho_\Gamma(f)$
$X \cup \overline{X}$	$\{\vee, \wedge\}$	Circuit	Nondet.	SDF	**NP**	$C \cdot \rho_\Gamma(f)$
X	$\{\vee, \wedge\}$	Circuit	Det.	Filters	*m***P**	$(\rho_\Gamma(f))^C$
$X \cup \overline{X}$	$\{\oplus, \wedge\}$	Circuit	Nondet.	Affine	**NP**	$C \cdot \rho_\Gamma(f)$

Table of results

Inputs	**Gates**	**Type**	**Mode**	Ω	$\mathcal{C}_\Delta$	**Upper bound**
$X \cup \overline{X}$	$\{\vee, \wedge\}$	Circuit	Det.	Filters	**P**	$(\rho_\Gamma(f))^C$
$X \cup \overline{X}$	$\{\vee, \wedge\}$	BP	Det.	Filters	**NL**	$C \cdot \rho_\Gamma(f)$
$X \cup \overline{X}$	$\{\vee, \wedge\}$	Circuit	Nondet.	SDF	**NP**	$C \cdot \rho_\Gamma(f)$
X	$\{\vee, \wedge\}$	Circuit	Det.	Filters	m**P**	$(\rho_\Gamma(f))^C$
$X \cup \overline{X}$	$\{\oplus, \wedge\}$	Circuit	Nondet.	Affine	**NP**	$C \cdot \rho_\Gamma(f)$

- A few parameters here are missing, such as restriction on the g, h in the cover triplets.

Table of results

Inputs	**Gates**	**Type**	**Mode**	Ω	$\mathcal{C}_\Delta$	**Upper bound**
$X \cup \overline{X}$	$\{\vee, \wedge\}$	Circuit	Det.	Filters	**P**	$(\rho_\Gamma(f))^C$
$X \cup \overline{X}$	$\{\vee, \wedge\}$	BP	Det.	Filters	**NL**	$C \cdot \rho_\Gamma(f)$
$X \cup \overline{X}$	$\{\vee, \wedge\}$	Circuit	Nondet.	SDF	**NP**	$C \cdot \rho_\Gamma(f)$
X	$\{\vee, \wedge\}$	Circuit	Det.	Filters	m**P**	$(\rho_\Gamma(f))^C$
$X \cup \overline{X}$	$\{\oplus, \wedge\}$	Circuit	Nondet.	Affine	**NP**	$C \cdot \rho_\Gamma(f)$

- A few parameters here are missing, such as restriction on the g, h in the cover triplets.
- $C = 4$ works for all of the upper bounds on the length of the shortest program.

Table of results

Inputs	**Gates**	**Type**	**Mode**	Ω	$\mathcal{C}_\Delta$	**Upper bound**
$X \cup \overline{X}$	$\{\vee, \wedge\}$	Circuit	Det.	Filters	**P**	$(\rho_\Gamma(f))^C$
$X \cup \overline{X}$	$\{\vee, \wedge\}$	BP	Det.	Filters	**NL**	$C \cdot \rho_\Gamma(f)$
$X \cup \overline{X}$	$\{\vee, \wedge\}$	Circuit	Nondet.	SDF	**NP**	$C \cdot \rho_\Gamma(f)$
X	$\{\vee, \wedge\}$	Circuit	Det.	Filters	*m***P**	$(\rho_\Gamma(f))^C$
$X \cup \overline{X}$	$\{\oplus, \wedge\}$	Circuit	Nondet.	Affine	**NP**	$C \cdot \rho_\Gamma(f)$

- A few parameters here are missing, such as restriction on the g, h in the cover triplets.
- $C = 4$ works for all of the upper bounds on the length of the shortest program.
- For **NP** there exists a "super-linear" lower bound: $\rho_\Gamma(f) = \Omega(\log\log\log^* n)$

Table of results

Inputs	Gates	Type	Mode	Ω	$\mathcal{C}_\Delta$	Upper bound
$X \cup \overline{X}$	$\{\vee, \wedge\}$	Circuit	Det.	Filters	**P**	$(\rho_\Gamma(f))^C$
$X \cup \overline{X}$	$\{\vee, \wedge\}$	BP	Det.	Filters	**NL**	$C \cdot \rho_\Gamma(f)$
$X \cup \overline{X}$	$\{\vee, \wedge\}$	Circuit	Nondet.	SDF	**NP**	$C \cdot \rho_\Gamma(f)$
X	$\{\vee, \wedge\}$	Circuit	Det.	Filters	*m***P**	$(\rho_\Gamma(f))^C$
$X \cup \overline{X}$	$\{\oplus, \wedge\}$	Circuit	Nondet.	Affine	**NP**	$C \cdot \rho_\Gamma(f)$

- A few parameters here are missing, such as restriction on the g, h in the cover triplets.
- $C = 4$ works for all of the upper bounds on the length of the shortest program.
- For **NP** there exists a "super-linear" lower bound: $\rho_\Gamma(f) = \Omega(\log\log\log^* n)$
- For *m***P** there exists a super-polynomial lower bound: $\rho_\Gamma(f) = \exp(\Omega(n^{1/8}))$