# of Computing: From circuit lower bounds to compression and SAT algorithms, and back

Valentine Kabanets (SFU)

(joint work with Antonina Kolokolova (MUN))

Are lower bounds useful ?

" Obviously, P ≠ NP ! Why spend so much time trying to prove it ??? "

- We want to **understand** efficient computation.
- The proof will give us important **new insights**.
- The proof will likely lead to **new algorithms**.

Applications of lower bounds

- Cryptography :

  hard problems $\Rightarrow$ security of crypto protocols [BM, Yao, …]

- Derandomization :
  - hard problems $\Rightarrow$ BPP = P [NW, IW, …]
  - hard problems $\Rightarrow$ derandomization of Noether's Normalization Lemma [Mulmuley]

Looking inside the lower-bound proof

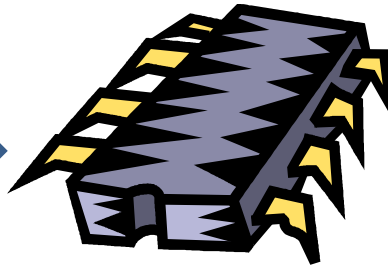
"A **natural proof** contains an efficient algorithm to distinguish an easy Boolean function from a random function, when given the truth table of the function as input."

Corollary: A natural proof of circuit lower bounds against a class C $\Rightarrow$ C cannot compute strong Pseudo-Random Generators.

Distinguishing easy functions from random functions: A dream version

Easy function

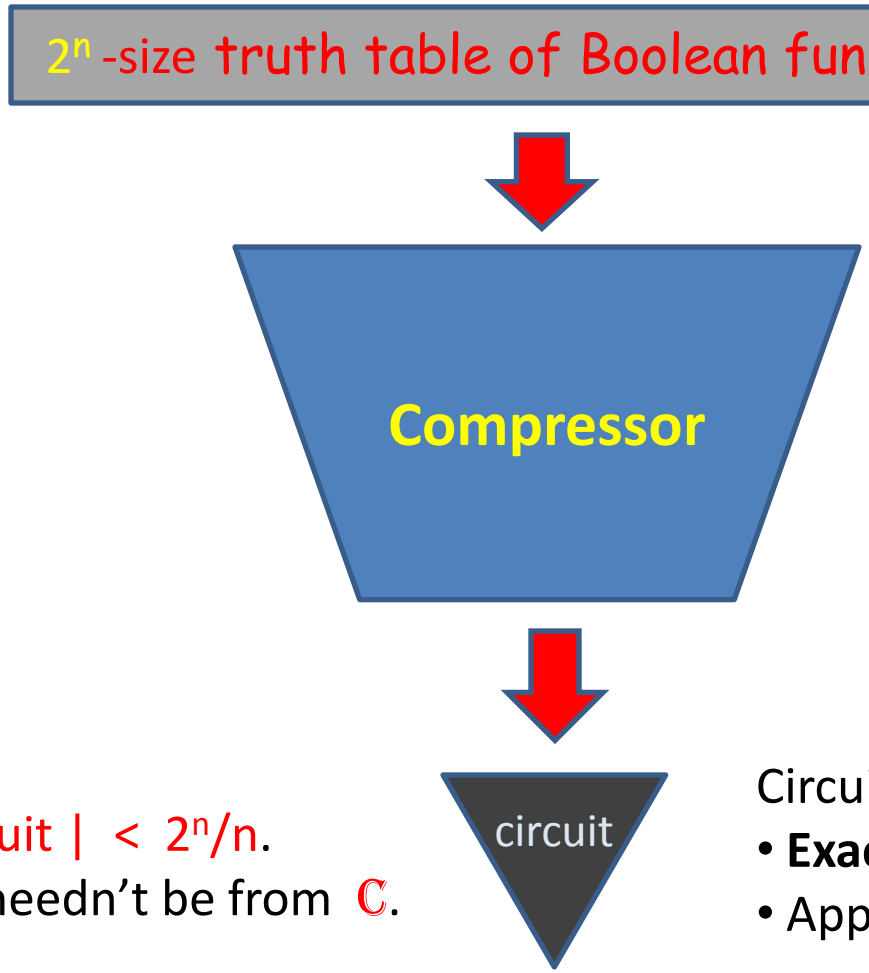
Efficient Compressor



Small circuit

Dream: A function is easy iff it compressible by our efficient compression algorithm.

Boolean-function Compression



From some class $\mathbb{C}$ of easy functions (e.g., $AC^0[poly]$)

Runtime: $poly(2^n)$

Want $|circuit| < 2^n/n$.
The circuit needn't be from $\mathbb{C}$.

Circuit computes f
- **Exactly** (**lossless** compression), or
- Approximately (lossy compression)

Locally decodable compression

Compression

- Data Compression :

  mp3, JPEG, zip, …

- Computational Learning :

  PAC learning , Exact learning

- Circuit Minimization :

  min-DNF, min-AC^0, …

Complexity-theoretic motivation

- Understanding easy functions:
  - how to extract a small circuit from the truth table of an easy function,

- Understanding hard functions:
  - circuit lower bounds from compression.

Circuit Minimization vs. Boolean-function Compression

- C-Minimization asks to find a min-size circuit of the type C that computes a given n-variate Boolean function.

- C-Compression asks for a "small" circuit, not necessarily of the type C.

(C-compression is proper if it produces a circuit of the type C.)

Learning implies Compression

L.G. Valiant '84 : PAC Learning

PAC learning algorithm (with membership queries) for uniform distribution implies a randomized lossy-compression algorithm.

D. Angluin '87: Exact Learning

Exact learning algorithm (with membership and equivalence queries) implies a lossless-compression algorithm.

Learning vs. Compression

- Learning: Small runtime of the learning algo $\Rightarrow$ small size of the hypothesis.

- Compression: Want circuits of size $< 2^n/n$, but allow time $\text{poly}(2^n)$.

Our result

"Circuit lower bound proofs based on random restrictions $\Rightarrow$ Compression."

Can compress functions computable by

- AC^0 circuits of size $< 2^{n^{1/(d-1)}}$,
- de Morgan formulas of size $< n^{2.5}$,
- general formulas & branching programs of size $< n^2$,
- read-once branching programs of size $< 2^{n/3}$

Related early work

O.B. Lupanov '58 : Every n-variate Boolean function has a circuit of size $2^n/n\ (1 + o(1))$.

Can be constructed efficiently, using (k,s)- representation of the function.

S.V. Yablonski '59 : Every n-variate Boolean function "with few distinct subfunctions" has a circuit of size $\sigma\ 2^n/n$, for some $\sigma < 1$.

Uses Lupanov's (k,s)- representation.

Restriction-based lower bounds $\Rightarrow$ structure of easy functions

If $f: \{0,1\}^n \to \{0,1\}$ is computable by a small circuit, then

$$f = g_1 \vee g_2 \vee \ldots \vee g_t,$$

where

- $t << 2^n$, and
- almost all g_i's are simple (have $O(n)$ description size).

Warm-up: DNF- Compression

Given truth table of n-variate Boolean function f, can efficiently compute DNF for f of size $O(n \cdot OPT)$, where OPT is the min-DNF size.

Using greedy heuristic for SetCover
[Johnson'74, Lovasz'75, Chvatal'79]

Greedy heuristic for SetCover

Let $S_1, \ldots, S_t$ be subsets of U. Suppose U can be covered by at most m of the subsets.

Greedy Algorithm:

Repeat until all of U is covered: Find S_i that covers the most of not-yet-covered points in U, and add S_i to the set cover.

This algorithm finds a set cover of size $O(m \log |U|)$.

Finding small DNF via Set Cover

- Given $f : \{0,1\}^n \to \{0,1\}$, find all conjunctions ϕ_i of n literals such that $\phi^{-1}_i(1) \subseteq f^{-1}(1)$.

- Run the Set Cover heuristic on $U = f^{-1}(1)$ and the sets $S_i = \phi^{-1}_i(1)$.

Runtime: $\text{poly}(2^n)$

(# conjunctions on n literals $< \text{poly}(2^n)$)

DNF- Compression vs DNF- Minimization

- There is a deterministic polynomial-time proper DNF- Compression algorithm.

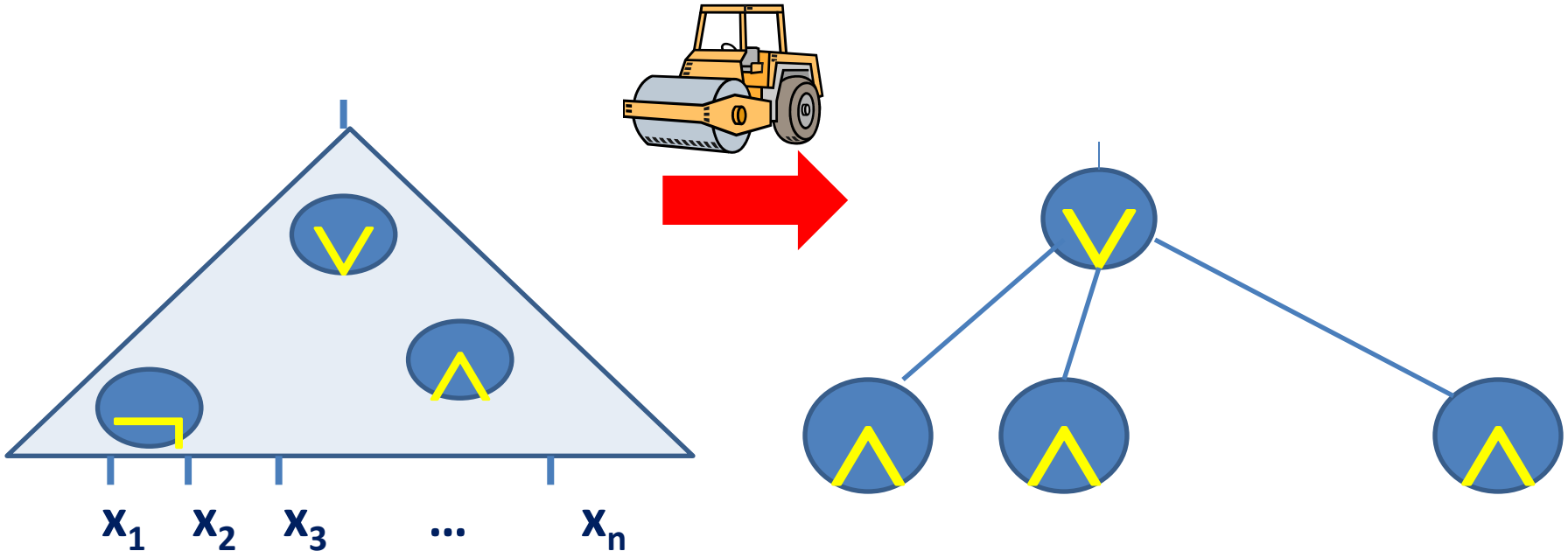
- **Theorem** [Masek'79; Feldman'09; Allender et al.'08]:
  DNF- Minimization is NP-hard (even for n^{γ} - approximation, for some constant $1 > \gamma > 0$).

AC^0 - Compression

Theorem: Can compress n-variate f computable by a depth d circuit of size s into a DNF of size at most $\text{poly}(s) \cdot 2^{n(1-\mu)}$, where $\mu \approx 1/(\log s)^{d-1}$.

Nontrivial compression for $s \leq 2^{n^{1/(d-1)}}$.

AC^0 to DNF via Switching Lemma [IMP'12]



AC^0 circuit:
size cn, depth d

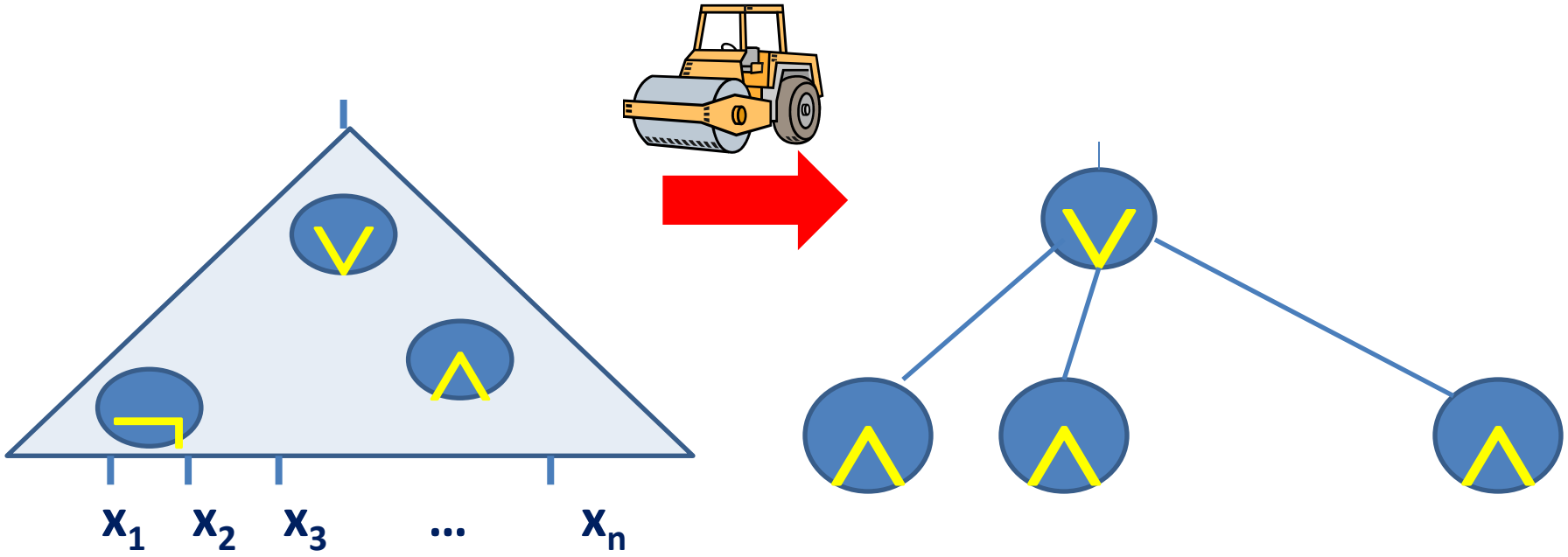
DNF : $\leq 2^{n(1-\mu)}$ ANDs, with
$\mu = 1/(\log c + d \log d)^{d-1}$

Switching Lemma [Hastad'86, Razborov'93, IMP'12] : Almost all restricted sub-circuits have shrunk size < # variables.
So, usually, no need to query all n vars !

Formula Compression

Theorem: Can compresses n-variate f of de Morgan formula size $c \cdot n$ into a DNF of size at most $n \cdot 2^{n(1-\mu)}$, where $\mu < 1$ is a constant dependent on c.

Formula to DNF via Shrinkage [San'10]



De Morgan formula: size cn

DNF : $\leq 2^{n(1-\mu)}$ ANDs, with $\mu = \mu(c)$

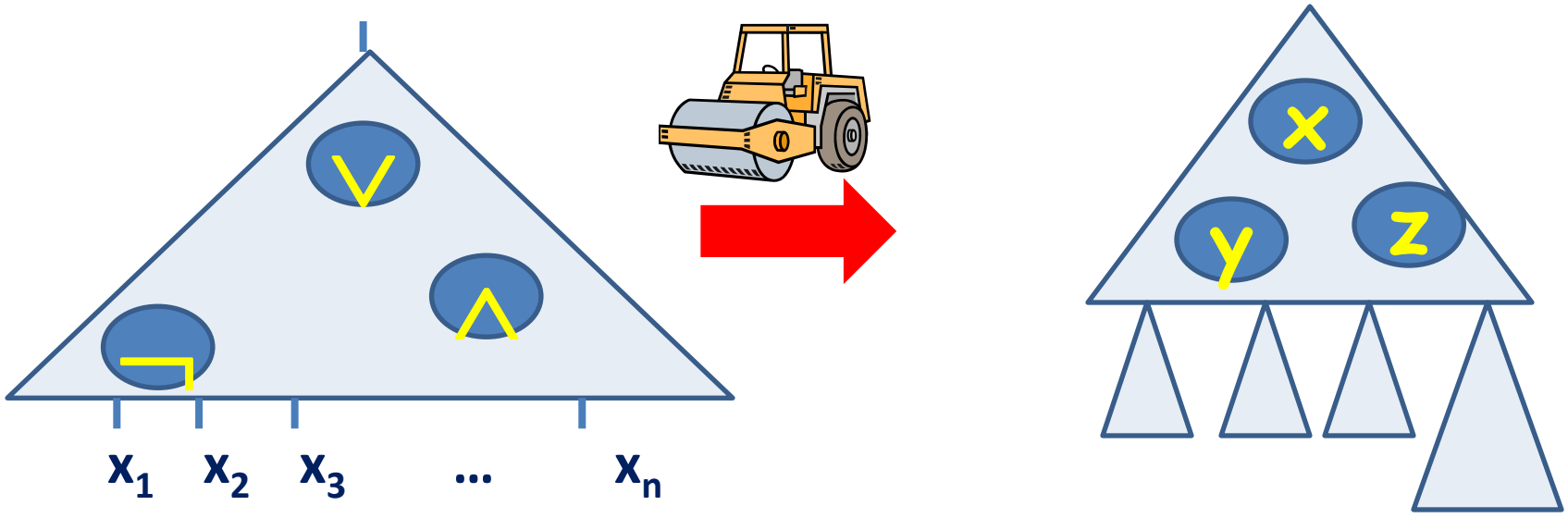
High-probability shrinkage [Sub'61, San'10]: Almost all restricted subformulas have shrunk size < # variables. So, usually, no need to query all n vars !

Compression of superlinear-size (general) formulas & BPs

Theorem: Can compress n-variate f of formula-size n^d into a formula of size at most $2^{n-n^{\varepsilon}}$, where $\varepsilon < 1$ is a constant dependent on d, and

- $d < 2.5$ for de Morgan formulas,
- $d < 2$ for general formulas & branching programs.

De Morgan Formulas Shrink [San'10,KR'12]

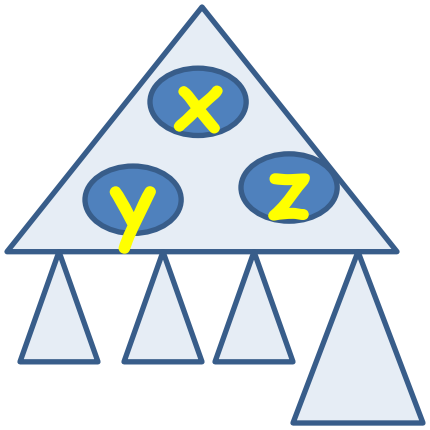


De Morgan formula:
size s on n variables,
where $s < n^{2.48}$

Decision tree : depth $n-k$, all but 2^{-k} fraction of leaves are formulas of size $< n^{0.99}$, on k vars, where $k = n^{o(1)}$.

Note: a typical leaf-formula is small, but **not** smaller than # vars . (n^2-size de Morgan computes PARITY.)

Restriction Decision Trees



Decision tree : depth $n-k$, all but 2^{-k} fraction of leaves are formulas of size $< n^{0.99}$, where $k = n^{o(1)}$.

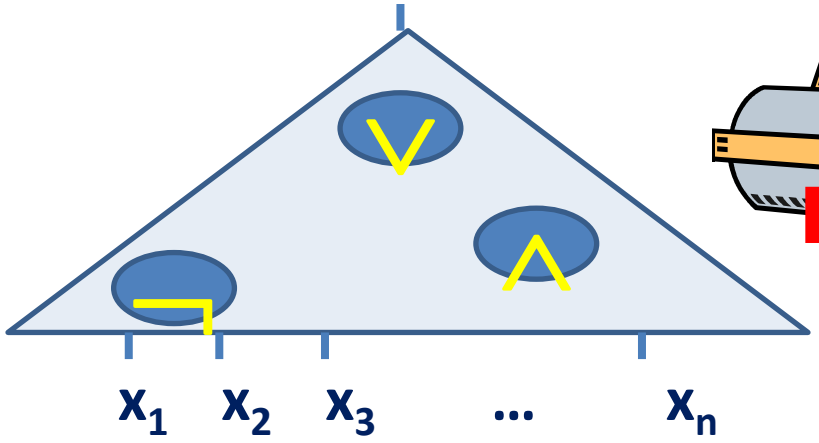
This decision tree = Disjunction of 2^{n-k} formulas, almost all of description size $O(n)$, while the rest are responsible for very few inputs.

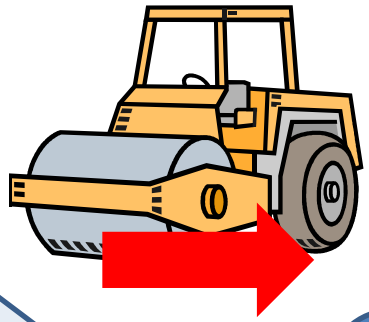
Generalized Set Cover heuristic:
Find $O(n\,2^{n-k})$ linear-size formulas plus one "not too large" DNF, whose disjunction computes the original function. The overall size of the circuit is $O(n^2\,2^{n-k})$.

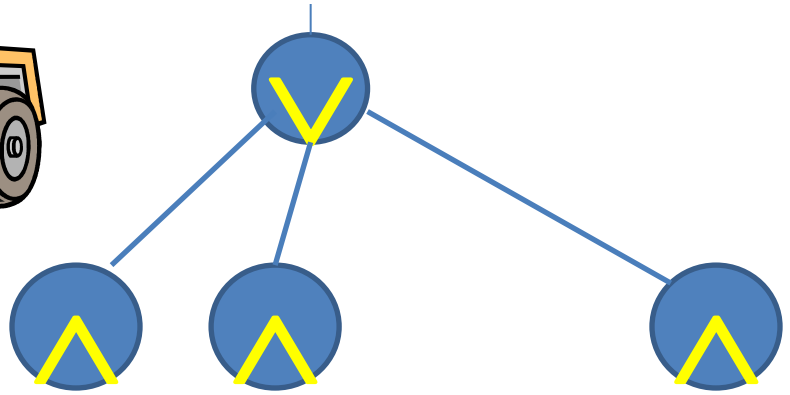
#SAT Algorithms from circuit lower bounds

#SAT Algorithms from restriction-based circuit lower bounds

#SAT Algorithms [IMP'12, San'10]







AC^0 circuit, or linear-size de Morgan formula

DNF : $\leq 2^{n(1-\mu)}$ ANDs, with $\mu < 1$.

ANDs have disjoint sets of satisfying assignments

Algo: (1) Convert to DNF. (2) Sum # sat assignments over all ANDs

Run Time $< 2^{n(1-\mu)}$. **Better than the naive runtime** 2^n poly(n).

Our #SAT-Algorithm

Theorem: Can solve #SAT for n-variate formula of size n^d in time $2^{n-n^{\varepsilon}}$, where

- $d < 2.5$ for de Morgan formulas,
- $d < 2$ for full-basis formulas, and general branching programs.

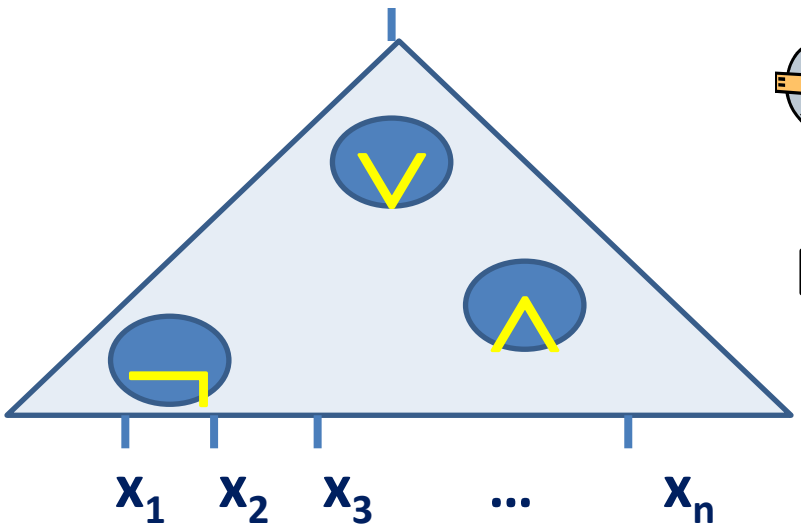
Related work

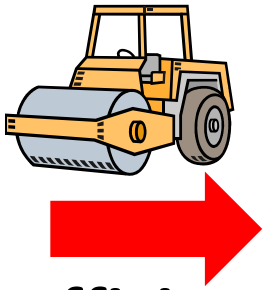
- [Santhanam'10; Seto,Tamaki'12]: non-trivial #SAT algorithms for linear-size de Morgan and general formulas.

- [IMZ'12]:

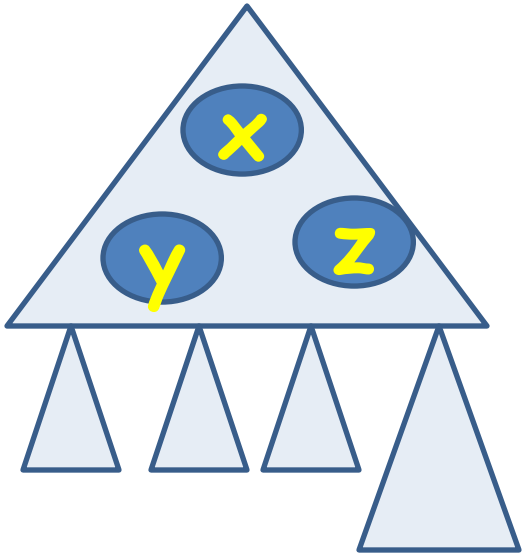
  "pseudorandomness from shrinkage" for n^3 -size de Morgan and n^2-size general formulas/branching programs.

W.h.p., Formulas Shrink [San'10,KR'12]





Efficient !

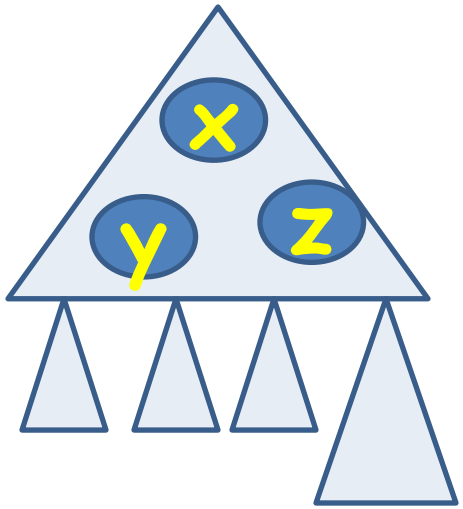


Full-basis formula:
size s on n variables,
where $s < n^{1.98}$

Decision tree : depth $n-k$, all but 2^{-k} fraction of leaves are formulas of size $O(s\,(k/n)) < n^{0.99}$, where $k = n^{o(1)}$.

DT construction is the **greedy heuristic**: branch on the most frequent variable $(n-k)$ times.

#SAT Algorithm: Main Idea

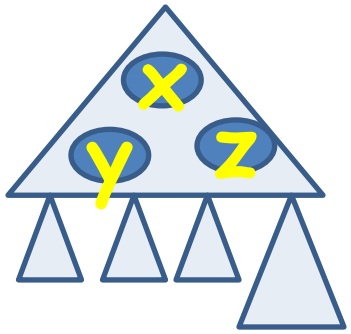


Decision tree : depth $n-k$, all but 2^{-k} fraction of leaves are formulas of size $O(s (k/n)) < n^{0.99}$, where $k = n^{o(1)}$.

Observation: # $n^{0.99}$ -size formulas << 2^{n-k} = # leaves in the decision tree. So many leaf formulas are the same.

- Precompute #SAT for all $n^{0.99}$ -size formulas !

#SAT Algorithm: Details



Decision tree : depth $n-k$, all but 2^{-k} fraction of leaves are formulas of size $O(s(k/n)) < n^{0.99}$, where $k = n^{o(1)}$.

1. Construct the decision tree of depth $n - k = n - n^{\varepsilon}$.
2. Solve & store #SAT values for all formulas of size $n^{0.99}$.
3. For each leaf of the decision tree,

   If the corresponding leaf formula has size $< n^{0.99}$, then look up the precomputed answer,

   else compute the answer by "brute force" in time 2^k.

   Add the answer to the running sum.

Overall running time: $2^{n-k} + 2^{n^{0.991}} + 2^{n-k} + 2^{n-k}\,2^{-k}\,2^{k} < 2^{n-k}$

Circuit lower bounds from Compression

C-Compression $\Rightarrow$ NEXP not in C

Theorem: Let $C \subseteq P/poly$ be any circuit class. Suppose that for every c, there is a deterministic polytime compression algorithm mapping $f \in C[n^c]$ to a circuit of size $< 2^n/n$. Then NEXP not in C.

Informally: Slightly nontrivial compression for $\mathscr{C}$-PolySize $\Rightarrow$ NEXP not in $\mathscr{C}$-PolySize.

$\mathcal{C}$-Circuit Satisfiability

Theorem [Williams '10]: There is $k>0$ such that :

If $\mathcal{C}$_SAT for n^c-size n-input circuits is in time $O(2^n/n^k)$ for every c,

then $\mathrm{NTime}(2^n)$ is not in $\mathcal{C}$-PolySize.

Informally: Slightly nontrivial $\mathcal{C}$-SAT algorithm $\Rightarrow$ NEXP not in $\mathcal{C}$_PolySize.

Compression of ACC^0 ?

Open: Nontrivial compression for ACC^0 ?

Would give a different proof of NEXP not in ACC^0.

Compression $\Rightarrow$ Circuit lower bounds

Theorem: Let $C \subseteq P/poly$. Suppose that for every c, there is a deterministic polytime compression algorithm mapping $f \in C[n^c]$ to a circuit of size $< 2^n/n$. Then NEXP not in C.

Generalizes Theorem [IKW'02]: Natural property useful against P/poly $\Rightarrow$ NEXP not in P/poly.

Proof Sketch

Claim ([IKW'02]): If $NEXP \subseteq C \subseteq P/poly$, then $\exists\ c > 0$ s.t. every NE language has witnesses in $C[n^c]$.

- Define NE language L': On x, $|x|=n$, nondeterministically guess 2^n-bit string, and accept if it is **not** compressible.

- L' doesn't have witnesses in $C[n^c]$, contrary to the Claim.

QED

Monotone functions

Theorem: If can compress m -variate Boolean functions of monotone circuit complexity $poly(m)$ to (not necessarily monotone) circuit size $< 2^m / m^{1.51}$, then get a natural property useful against $P/poly$.

Proof Idea: Monotone and non-monotone circuit complexities are same for slice functions.

Use optimal embedding of an arbitrary Boolean function into a slice function.

Summary

- Restriction-based circuit lower bound techniques $\Rightarrow$ Meta-Algorithms : Compression & SAT

- Nontrivial Compression or SAT algorithm for a circuit class C implies NEXP not in C [poly]

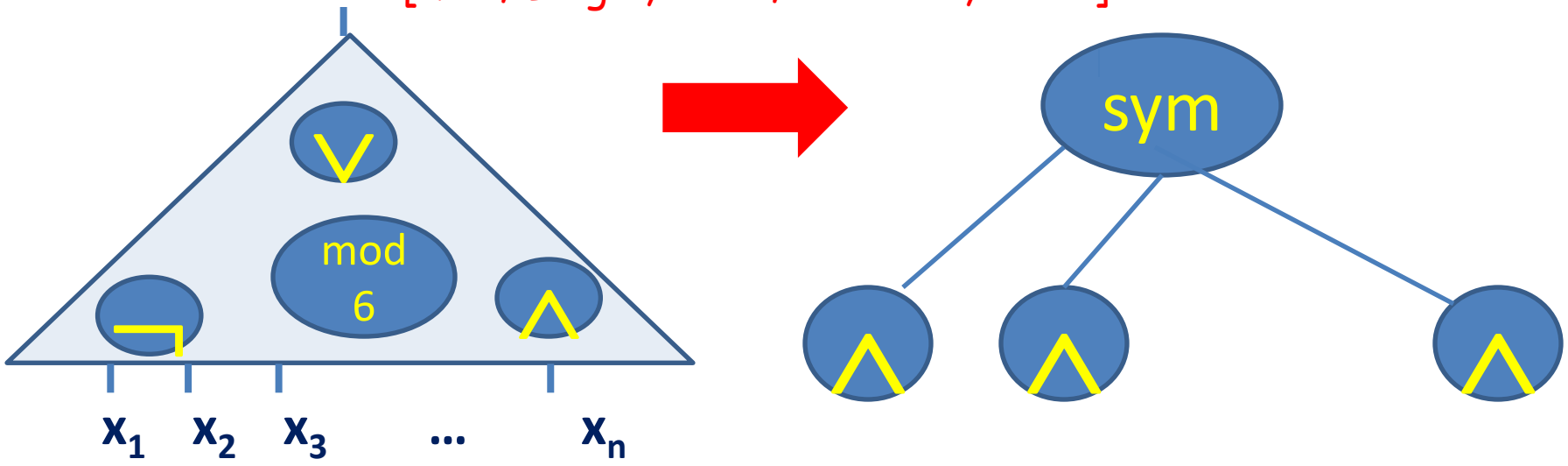
Questions

- Better compression for AC^0 ? (polysize ?)
- Compression for $AC^0[\oplus]$?
- Compression for ACC^0 ?
- Every C-circuit lower bound proof we know yields Compression & SAT algorithms for the circuit class C ?

Thank you !

ACC^0 – #SAT Algorithm [Williams]

[Yao; Beigel, Tarui; Allender, Gore]



ACC^0 circuit:
size s, depth d

SYM-AND: $s^{poly(\log s)}$ of ANDs

#SAT for SYM-AND : Time $(2^n + poly(S))\, poly(n)$ via Dynamic Programming (zeta-transform)