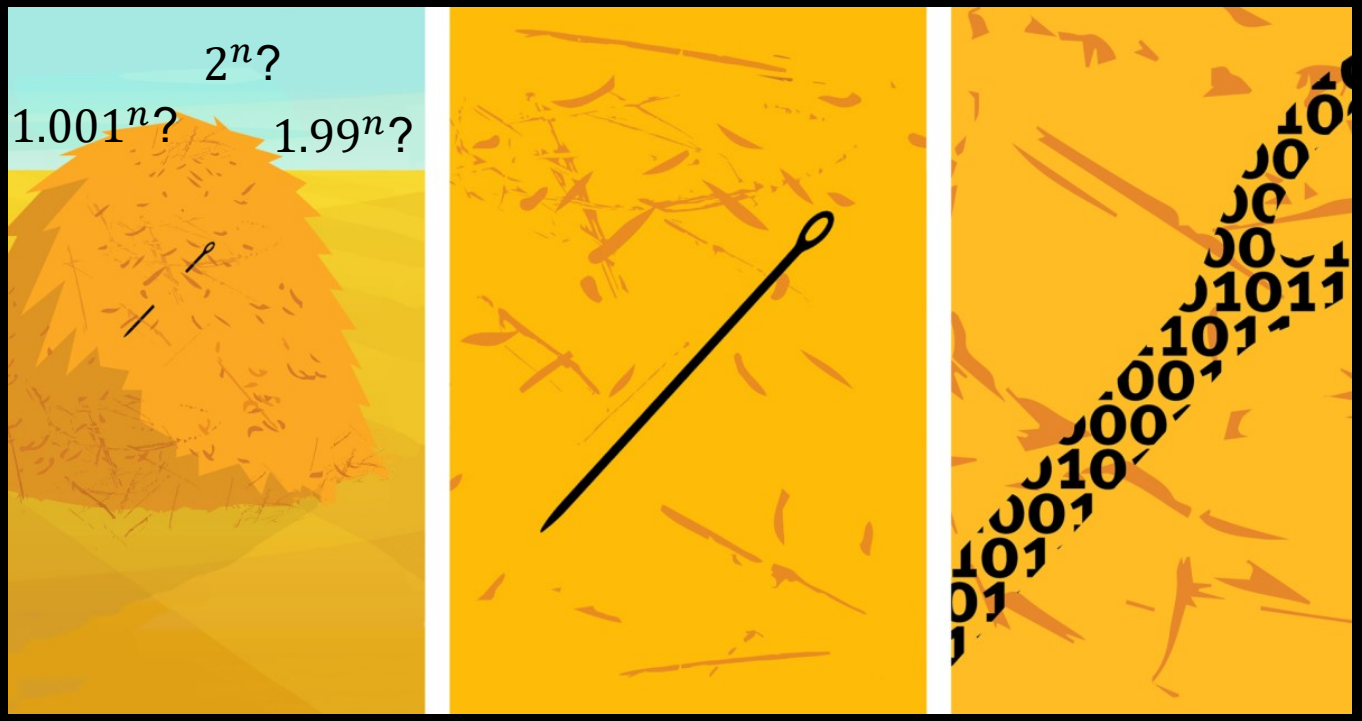
Around the Fine-Grained Complexity of SAT



(picture courtesy of the Simons Institute, UC Berkeley)

Ryan Williams, MIT

My Goal

To tell you what we know so far about the time complexity of SAT, and all it connects to

A theory of *"fine-grained complexity"*, developed over the last 20 years, uses hypotheses about SAT (and other problems) to reason about the complexity of problems in a way that NP-completeness never could before

1. I personally believe SAT can probably be solved much faster, and researchers are currently in various "local optima"
2. It's important to know what can be ***mathematically proved*** about how well (or poorly) SAT solvers could possibly run
3. The time complexity of SAT is related to ***much*** more than **NP**
4. Solving Circuit SAT in the worst-case, even slightly faster than brute-force, implies circuit complexity lower bounds

Flavors of SAT

Circuit-SAT = { satisfiable Boolean circuits }

Formula-SAT = { satisfiable propositional formulas }

CNF-SAT = { satisfiable boolean formulas in CNF }

k-SAT = { SAT where all clauses have at most k literals}

Measures of SAT instances:

- n = **number of variables**
- m = **number of clauses/connectives/gates**

All are solvable in $2^n \cdot poly(m)$ time.

All are **NP-complete**, so are ***equivalent*** from the point of view of **P** versus **NP**. ***But their actual time complexities are apparently quite different!***

The best-known algorithms for k-SAT run slower and slower, as k increases

Q: 3-SAT, 4-SAT, CNF-SAT are all NP-complete… so why don't they have the *same* running time?

A: Reducing from 4-SAT to 3-SAT increases the variables!

We introduce a new variable for every clause.

Example: a 3-SAT algorithm using c^n **time** would only imply a $\approx c^{n+m}$ **time** algorithm for 4-SAT, using the typical polynomial-time reduction

Generally speaking, $m >> n$ …

Moral: The representation of the formula matters.

Time Complexity of k-SAT (with respect to n)

[MS 85] **1.62^n for 3-SAT, 1.84^n for 4-SAT** **(DPLL-like branching)**

In general: $2^{n\left(1-\frac{c}{2^k}\right)}$ for k-SAT for a constant $c > 1$

[PPZ 97] **$2^{n(1-1/k)}$ for k-SAT** **(randomized branching w/ restarts**

(e.g. 1.59^n for 3-SAT, 1.69^n for 4-SAT) **+ unit clause rule)**

[PPSZ 98] **1.308^n** for Unique 3-SAT **(randomized w/ bounded resolution)**

$2^{n\left(1-\frac{c}{k}\right)}$ for $c \approx \pi^2/6 > 1$ **[notoriously difficult to analyze!]**

[Schoening 99] **$2^{n(1-c''/k)}$** for k-SAT, **1.34^n** for **3-SAT** **(local search)**

[Hertli 11] **1.308^n** for 3-SAT, **1.47^n** for **4-SAT** **(better PPSZ analysis)**

[Chan-W 18] **$2^{n\left(1-c'''/k\right)}$ for #k-SAT** **(polynomial method)**

[HKZZ 19] $2^{n\left(1-\frac{c''''}{k}\right)}$ time, **1.307^n** for Unique 3-SAT **(PPSZ + ``bias'')**

[Scheder 21] The PPSZ algorithm achieves **1.307^n** for 3-SAT already ☺

All known algorithms take > $2^{n(1 - c/k)}$ time to solve k-SAT

All running times converge to 2^n as $k \rightarrow \infty$

CNF-SAT: best known algs take $O(2^{n - n/(\log(m/n))})$ time [Schuler'05,CIP'06]

Refinements of P ≠ NP

There is a dramatic difference in how well various NP-complete problems can be solved in practice.

A formal understanding of 'what is possible' requires more 'fine-grained' assumptions than P ≠ NP.

We need lower bound hypotheses that guide us to a better understanding of the frontier: what we can and cannot expect to solve much faster than brute force.

Ultimately, we want to find better algorithms, even if they take super-polynomial time. (And if we can't, we want to know what the consequences would be!)

Two Major Conjectures About SAT

- ***Is 3-SAT in $2^{\epsilon n}$ time, for every constant $\epsilon > 0$?***

  Exponential Time Hypothesis [IPZ'01]: **Conjectures "no"**

  ETH: "3-SAT is not in $(1+\epsilon)^n$ time, for some $\epsilon > 0$"

3-SAT can't be solved in $1.0000\cdots01^n$ time (for *some* number of 0's)

- ***Is k-SAT in $2^{\delta n}\,poly(m)$ time for some universal $\delta < 1$?***

  Strong ETH [IP'99,CIP'09]: **Conjectures "no"**

  SETH: "For all $\delta < 1$, there is a k such that k-SAT is not in $(2-\delta)^n$ poly(m) time"

CNF-SAT can't be solved in $1.9999\cdots9^n$ time (for *all* numbers of 9's)

Theorem: SETH implies ETH (not obvious!)

Useful: ETH and SETH imply *many* interesting predictions in TCS

Evidence for SETH and ETH?

All known algorithms are consistent with SETH and ETH

[Haken] **General Resolution needs $2^{\Omega(n)}$ size**

[Beame Pitassi '96] **k-CNFs need resolution proofs of size > $2^{\Omega\left(\frac{n}{k}\right)}$**

[Pudlak Impagliazzo '00] [Beck Impagliazzo '13]

There are unsatisfiable k-CNFs which *require* "regular" resolution proofs of size $2^{n-\frac{n}{k^{0.25}}}$ ("Strong ETH holds for DPLL")

There are unsatisfiable k-CNFs which require general resolution proofs of size $(1.5)^{n-\frac{n}{k^{0.25}}}$ ("ETH holds for CDCL")

[Pudlak et al '17] [Scheder Talebanfard '20]

There are satisfiable k-CNFs such that the "standard" version of PPSZ requires $2^{n\left(1-\frac{O(1)}{k}\right)}$ steps to solve, in expectation.

So, if SETH/ETH are false, we may need radically new algs!

Even improving over an exponent of $n\left(1-\frac{c}{k}\right)$ seems hard…

Tight lower bounds under ETH

Assuming ETH: the problems

Independent Set, Clique, Vertex Cover, Dominating Set, Graph Coloring, Max Cut, Set Splitting, Hitting Set, Min Bisection, Feedback Vertex Set, Hamiltonian Path, Max Leaf Spanning Tree, Subset Sum, Knapsack, Cluster Editing, 3-Dimensional Matching, Treewidth ***and many others*** do not have $2^{\varepsilon n}$ time algorithms, for various $\varepsilon > 0$.

Predictions Beyond NP!

For **many fundamental *polynomial-time* problems**, improving the best known algorithms even slightly, implies **¬SETH** or **¬ETH**

That is:

SETH and ETH predict the optimality of many known *polynomial-time* algorithms!

"Hardness Within P"

Fine-Grained Complexity

Hardness Within P

Edit Distance: Given two n-bit strings x and y, find the min number of symbol insertions/deletions needed to transform x into y

Well-known to be in $O(n^2)$ time

[BI'14] [AHVW'15] **SETH** $\Longrightarrow$ NOT solvable in $O(n^{2-\epsilon})$ time

d-SUM: Given n numbers, are there d that sum to zero?

In $O(n^{\lceil d/2 \rceil})$ time. [PW'10] **ETH** $\Longrightarrow$ d-SUM requires $n^{\Omega(d)}$ time

Partial Match Queries: Given database D of n strings d-bits long, and given n queries with wildcards, is there a query that matches a string in D?

In $O(n^2 d)$ time. [W'05] **SETH** $\Longrightarrow$ NOT in $n^{2-\varepsilon} \cdot 2^{o(d)}$ time for all $\varepsilon > 0$

CNF Batch Evaluation: Given a CNF on m clauses and m^{100} assignments to its variables, report the value of the CNF on all assignments.

In about m^{101} time. [GKW'??] **SETH** $\Longrightarrow$ NOT in $m^{101-\varepsilon}$ time for all $\varepsilon > 0$

A Natural Next Hypothesis?

The best known k-SAT algorithms run in time

$2^{n\left(1-\frac{c}{k}\right)}$ for various universal $c > 1$.

This bound is achieved by at least four different algorithms (two of which resemble practical algorithms)

1. DPLL-Like Branching [PPZ,PPSZ]
2. Local Search [Schoening]
3. Random Restrictions/Switching Lemma [IMP'12]
4. Algebraic/Polynomial Evaluation [W,Chen-W'18]

Each one navigates the search space in different ways. Yet they're all stuck at the same bound…

A Natural Next Hypothesis?

The best known k-SAT algorithms run in time

$2^{n\left(1-\frac{c}{k}\right)}$ for various universal $c > 1$.

Is this *particular* dependence on k necessary?

Could it be improved *at all*?

Super-SETH [myself, 2015]

There is NO unbounded function $f(k)$ such that

for infinitely many k, k-SAT is in $2^{n\left(1-\frac{f(k)}{k}\right)}$ time.

SSETH $\Rightarrow$ No $2^{n\left(1-\frac{\log(k)}{k}\right)}$ time algorithm for k-SAT!

Completely consistent with our knowledge!

What We Know About Super SETH

1. The "standard" version of the PPSZ algorithm (for k-SAT) **cannot break** Super SETH. [Scheder-Talebanfard'20]

2. Super SETH is *false* for Random k-SAT

For example, on the "planted k-SAT distribution" where we pick a random assignment on n variables, then choose $O(n)$ random clauses consistent with the assignment:

k-SAT is in $2^{n\left(1-\frac{\log^2 k}{k}\right)}$ time [Vyas-W'21, Lincoln-Yedida'21]

3. If there's an algorithm running in $2^{n\left(1-\frac{f(k)}{k}\right)}$ time on formulas with *exactly one SAT assignment*, Super SETH is false [VW'21]

4. The best known algorithms for Partial Match Queries (based on fast polynomial evaluation) cannot be easily extended to refute Super SETH. [unpublished]

My Personal Opinion

I believe Super-SETH and SETH are false.

(But ETH, I don't know…)

My belief in ¬SETH is the minority opinion.

(But the chances I'll be proved wrong in my lifetime are nil ?!)

Even if SETH is true, my belief in ¬SETH led me to many ideas I'd have never found otherwise.

That's another talk…

New Complexity Theory Through SAT Solving

Or: Why Did I Only Talk About Faster Algorithms for CNF-SAT?

Faster Circuit-SAT algorithms?

This is open!

We know k-SAT can be solved in << 2^n time

3-SAT: $O(1.307^n)$ time

k-SAT: $O(2^{n - cn/k})$

CNF SAT: $O(2^{n - n/\log n})$

Q: Why can't the above results be used to solve Circuit-SAT faster than 2^n?

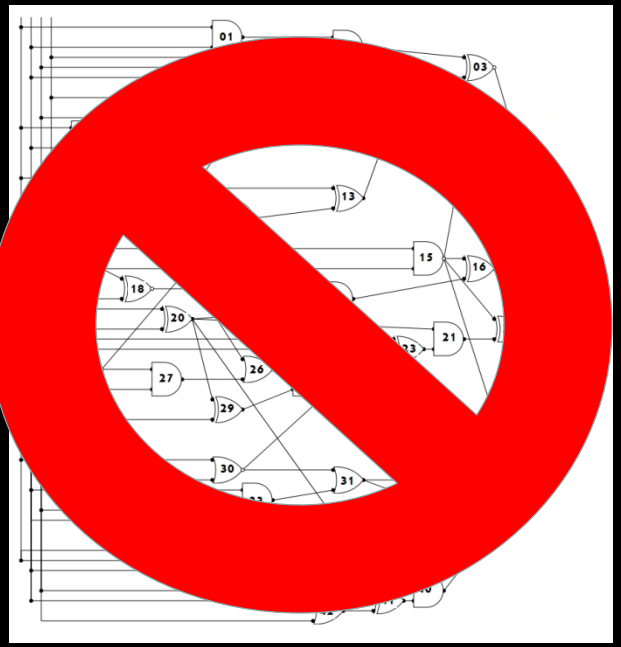
A: The transformation from Circuit Satisfiability to CNF blows up the number of variables!

Introduces a new variable for each *gate* of the circuit

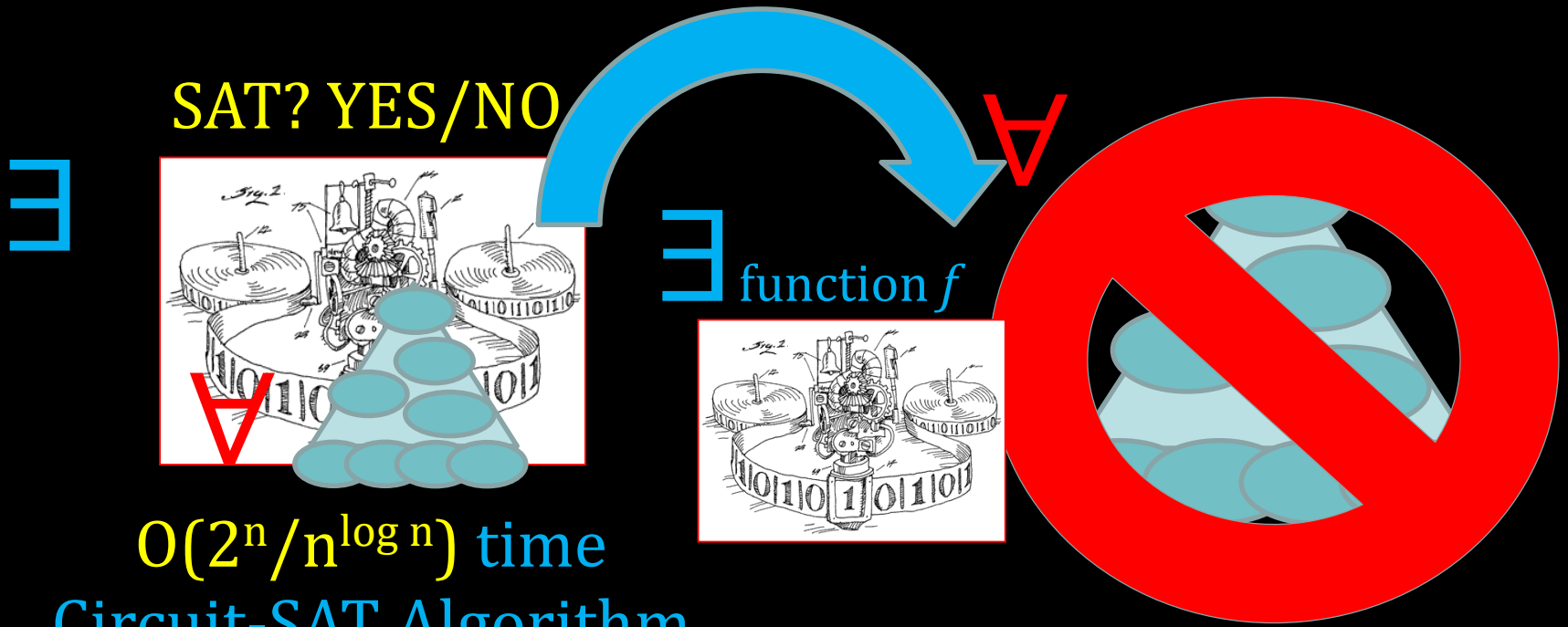
Circuit-SAT Algs Imply Lower Bounds!

If **Circuit-SAT** is in $\frac{2^n}{n^{\log n}}$ time for all poly(n)-size circuits…

Then: Impossibility results for computing certain functions with **small non-uniform circuits**

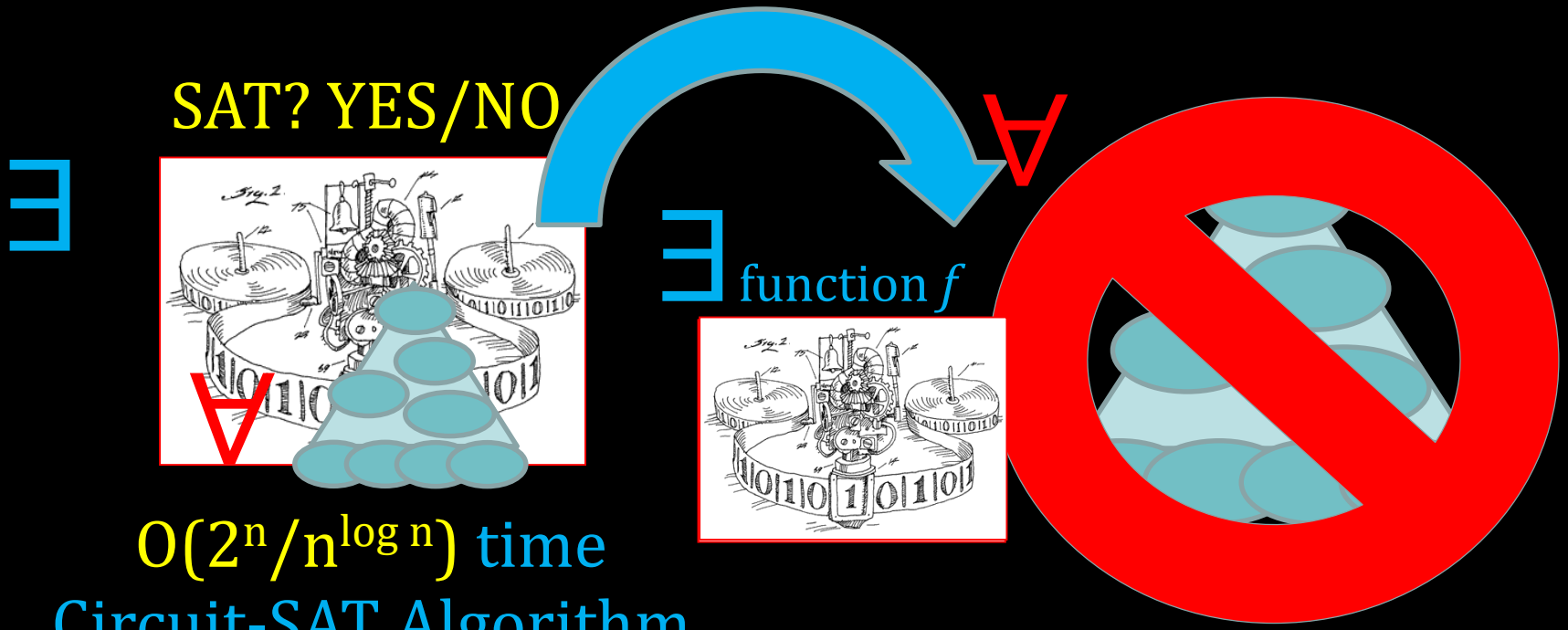


Circuit-SAT Algs Imply Lower Bounds!



Analogous results hold for Formulas and other representations!

SAT Solving Can Help Complexity Theory!

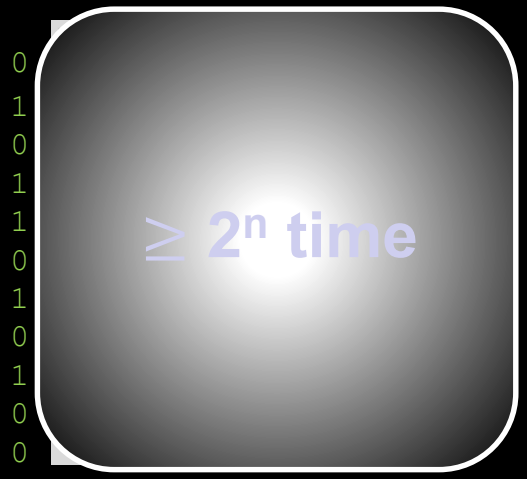


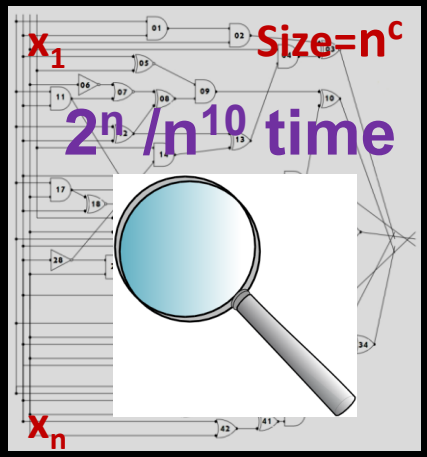
Intuition: Why are such implications true?

Cryptographic Intuition:

Faster Circuit-SAT algorithms reveal a ***weakness*** of small circuits

Small circuits cannot "obfuscate" the all-zeroes function as well as a black-box can!





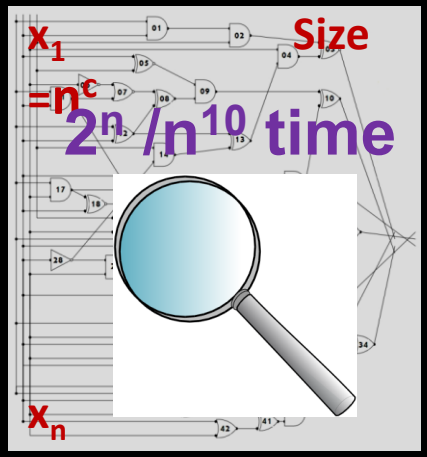
Proposition: **For every algorithm *A* computing SAT on black-boxes, there is a box *B* such that *A* must call *B* for 2^n times!**

Therefore: a faster Circuit SAT algorithm demonstrates a concrete difference between a "white-box" circuit problem and a "black-box" problem

Intuition: Why are such implications true?

Algorithmic Intuition: Faster Circuit-SAT algorithms show a ***strength*** of "*faster than* 2^n" algorithms!

A "quicker" algorithm can tell when a given circuit computes the all-zeroes function!



Therefore: Faster-Than-2^n time Algorithms are **"strong"** and Small Circuits are **"weak"**... *allowing us to construct an "algorithmically-defined function" which doesn't have small circuits*

Open Problems

- Give more evidence that **SETH** is true?
  - **Prove that ETH is equivalent to SETH?**

    Consider the Circuit-SAT problem on ϵn variables and 2^n size. Clearly solvable in about $2^{n+\epsilon n}$ time.

    Theorem: If there is any $\varepsilon > 0$ such that this Circuit-SAT problem is in $2^{n+o(n)}$ time, then **ETH is false**!
  - **Prove that ¬SETH implies an *unlikely* result in theoretical computer science?** We know it implies circuit complexity lower bounds, but we *expect* those!
- Evidence against **SETH**?

Can general resolution refute all unsatisfiable k-CNF formulas with proofs of size at most 1.99^n?

- Give evidence that **Super-SETH** is false? ☺

Thank you!
Grazie!