The Cook-Levin Thm

Exposition by William Gasarch—U of MD

BILL, RECORD LECTURE!!!!

BILL RECORD LECTURE!!!

Variants of SAT

1. SAT is the set of all boolean formulas that are satisfiable. That is, $\phi(\vec{x}) \in SAT$ if there exists a vector $\vec{b}$ such that $\phi(\vec{b}) = TRUE$.
2. CNFSAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of literals.
3. k-SAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of exactly k literals.
4. DNFSAT is the set of all boolean formulas in SAT of the form $C_1 \vee \cdots \vee C_m$ where each C_i is an $\wedge$ of literals.
5. k-DNFSAT is the set of all boolean formulas in SAT of the form $C_1 \vee \cdots \vee C_m$ where each C_i is an $\wedge$ of exactly k literals.

Turing Machines Def

Def A *Turing Machine* is a tuple $(Q, \Sigma, \delta, s, h)$ where

- Q is a finite set of states. It has the state h.
- Σ is a finite alphabet. It contains the symbol $\#$.
- $\delta : (Q - \{h\}) \times \Sigma \rightarrow Q \times \Sigma \cup \{R, L\}$
- $s \in Q$ is the start state, h is the halt state.

Note There are many variants of Turing Machines- more tapes, more heads. All equivalent.

Conventions for our Turing Machines

1. Tape has a left endpoint; however, the tape goes off to infinity to the right.
2. The alphabet has symbols $\{a, b, \#, \$, Y, N\}$.
3. $\#$ is the blank symbol.
4. \$ is a separator symbol.
5. Y and N are only used when the machine goes into a halt state. They are YES and NO.
6. The input is written on the left. So the input *abba* would be on the tape as

$$abba\#\#\#\cdots$$

7. The head is initially on the rightmost symbol of the input. So it he above it would be on the *a* just before the $\#$ symbol.

How to Represent any Computation

Let M be a Turing Machine and $x \in \Sigma^*$. We represent the computation $M(x)$ as follows:

Example The tape has:

$$abba\#abcab\#a\#\#\#\cdots$$

If the machine is in state q and the head is looking at the c then we represent this by:

$$abba\#ab(c,q)ab\#a\#\#\#\cdots$$

Convention—extend alphabet and allow symbols $\Sigma \times Q$. The symbol (c,q) means the symbol is c, the state is q, and that square is where the head of the machine is.

Configurations

We need a term for strings like:

$$abba\#ab(c,q)a$$

Def Strings in $\Sigma^*(\Sigma \times Q)\Sigma^*$ are **configuration**.

The Computation $M(x)$ is represented by a sequence of configs.

Key A config is finite since what we don't see is $\#$.

Example

If $\delta(s, b) = (q, L)$ and $\delta(q, b) = (p, a)$

a	a	b	b	(b, s)	$\#$
a	a	b	(b, q)	b	$\#$
a	a	b	(a, p)	b	$\#$

- The left endpoint is the end of the tape.
- The unseen symbols on the right are all $\#$

How to Represent an NP Computation

Let $X \in \mathrm{NP}$.

How to Represent an NP Computation

Let $X \in \mathrm{NP}$.
Then there exists a poly p and a TM that runs in time poly q such that

$$X = \{x \mid (\exists y)[|y| = p(|x|) \text{ AND } M(x, y) = Y]\}$$

How to Represent an NP Computation

Let $X \in \mathrm{NP}$.
Then there exists a poly p and a TM that runs in time poly q such that

$$X = \{x \mid (\exists y)[|y| = p(|x|) \text{ AND } M(x,y) = Y]\}$$

$M(x,y)$ runs in time $\leq q(|x| + |y|) = q(|x| + p(|x|))$.

How to Represent an NP Computation

Let $X \in \mathrm{NP}$.
Then there exists a poly p and a TM that runs in time poly q such that

$$X = \{x \mid (\exists y)[|y| = p(|x|) \text{ AND } M(x,y) = Y]\}$$

$M(x,y)$ runs in time $\le q(|x| + |y|) = q(|x| + p(|x|))$.
Let $t(n) = q(n + p(n))$, a poly.

How to Represent an NP Computation

Let $X \in \mathrm{NP}$.
Then there exists a poly p and a TM that runs in time poly q such that

$$X = \{x \mid (\exists y)[|y| = p(|x|) \text{ AND } M(x,y) = Y]\}$$

$M(x,y)$ runs in time $\le q(|x| + |y|) = q(|x| + p(|x|))$.
Let $t(n) = q(n + p(n))$, a poly.
Here is ALL that matters:

- Numb of steps $M(x,y)$ takes is $\le t(|x|)$. Hence $\le t(|x|)$ configs.
- Computation can only look at the first $t(|x|)$ tapes squares on any config.

New Convention

Old Convention

$\#$	a	a	b	b	(s, b)	$\#$

means that off to the right there are an infinite number of $\#$.

New Convention

Old Convention

$\#$	a	a	b	b	(s, b)	$\#$

means that off to the right there are an infinite number of $\#$.

New Convention

$\#$	a	a	b	b	(s, b)	$\#$	$\cdots$	$\#$

Tape is $t(|x|)$ long so **know** when stops. Can include entire tape.

Key Config is finite since what we don't see is never used.

Summary of What's Important

Let $X \in \mathrm{NP}$ via poly q and TM M, so

$$X = \{x : (\exists y)[|y| = q(|x|) \wedge M(x, y) = Y]$$

Summary of What's Important

Let $X \in \mathrm{NP}$ via poly q and TM M, so

$$X = \{x : (\exists y)[|y| = q(|x|) \wedge M(x, y) = Y]$$

$x \in X$ implies $(\exists y)[|y| = q(|x|) \wedge M(x, y) = Y]$ implies $(\exists y, C_1, \ldots, C_t)[C_1, \ldots, C_t$ is an accepting comp of $M(x, y)]$

Cook-Levin Thm

Theorem

SAT is NP*-complete.*

We need to prove two things:

1. $\mathrm{SAT} \in NP$.

$$\mathrm{SAT} = \{\phi : (\exists \vec{y})[\phi(\vec{y}) = T]\}$$

   Formally

$$B = \{(\phi, \vec{y}) : \phi(\vec{y}) = T\}$$

   The satisfying assignment is the witness.

2. For all $X \in \mathrm{NP}$, $X \leq \mathrm{SAT}$. This is the bulk of the proof.

$x \in X \to \ldots$

If $x \in X$ then there is a y of length $p(|x|)$ such that $M(x, y) = Y$.
If $x \in X$ then there is a y and a sequence of configurations $C_1, C_2, \ldots, C_t$ such that

- C_1 is the configuration that says 'input is $x\$y$, and I am in the starting state.'
- For all i, C_{i+1} follows from C_i (note that M is deterministic) using δ.
- C_t is the configuration that is in state h and the output is Y.
- $t = q(|x| + p(|x|))$.

How to make all of this into a formula?

How to Represent Sequence of Configs as Fml

KEY 1: We have variables for every possible entry in every possible configuration. The variables are

$$\{z_{i,j,\sigma} : 1 \le i, j \le t, \sigma \in \Sigma \cup (Q \times \Sigma)\}$$

If there is an accepting sequence of configurations then $z_{i,j,\sigma} = T$ iff the jth symbol in the ith configuration is σ.

Making the $z_{i,j,\sigma}$ Make Sense

Need that for all $1 \leq i, j \leq t$ there exists exactly one σ such that $z_{ij\sigma}$ is TRUE.

$$\bigvee_{\sigma \in \Sigma \cup (\Sigma \times Q)} z_{i,j,\sigma}$$

for each $\sigma \in \Sigma \cup (\Sigma \times Q)$

$$z_{i,j,\sigma} \rightarrow \bigwedge_{\tau \in \Sigma \cup (\Sigma \times Q) - \{\sigma\}} \neg z_{i,j,\tau}$$

C_1 is Start Config

C_1 is the $\bigwedge$ of the following:
C_1 starts with x. Let $x = x_1 \cdots x_n$.

$$z_{1,1,x_1} \wedge \cdots \wedge z_{1,n-1,x_{n-1}}, z_{1,n,(x_n,s)} \wedge z_{1,n+1,\$}$$

C_1 then has $q(|x|)$ symbols from $\{a,b\}$, so NOT the funny symbols.

$$\bigwedge_{j=n+2}^{n+q(|x|)+1} \bigvee_{\sigma\in\{a,b\}} z_{1,j,\sigma}$$

C_1 then has all blanks:

$$\wedge \bigwedge_{j=q(n)+n+3}^{t(n)} z_{1,j,\#}$$

C_1 is Start Config: Example

$x = ab$, $p(n) = n^2$, and $q(n) = 2n$
$|y| = 4$. Input to M is of length $2 + 4 + 1 = 7$, so $M(x, y)$ runs $\le 2 \times 7 = 14$ steps.
Formula saying C_1 codes x as input is

$$z_{1,1,a} \wedge z_{1,2,(b,s)} \wedge z_{1,3,\$} \wedge$$

$$(z_{1,4,a} \vee z_{1,4,b}) \wedge (z_{1,5,a} \vee z_{1,5,b}) \wedge (z_{1,6,a} \vee z_{1,6,b}) \wedge (z_{1,7,a} \vee z_{1,7,b}) \wedge$$

$$z_{1,8,\#} \wedge \cdots \wedge z_{1,23,\#}$$

C_t is an Accept Config

Convention $M(x,y)$ accepts means $M(x,y)$ leaves a Y on the left most square and the head is on the left most square.
The state in C_t is h, the halt state,

$$z_{t,1,(Y,h)}$$

C_i leads to C_{i+1}

Thought Experiment: What if $\delta(q,a) = (p,b)$. Then:

σ_1	(a,q)	σ_2
σ_1	(b,p)	σ_2

Formula is a $\bigwedge$ over relevant i, j, σ_1, σ_2 of:

$$(z_{ij\sigma_1} \wedge z_{i(j+1),(a,q)} \wedge z_{i,(j+2)\sigma_2}) \rightarrow$$

$$(z_{(i+1)j\sigma_1} \wedge z_{(i+1)(j+1),(b,p)} \wedge z_{(i+1),(j+2)\sigma_2})$$

C_i leads to C_{i+1}

Thought Experiment: What if $\delta(q,a) = (p,L)$. Then:

σ_1	(a,q)	σ_2
(σ_1, p)	a	σ_2

One can make a formula out of this as well. (Leave for HW.)

C_i leads to C_{i+1}

Note that only the symbols at or near the head get changed.

Also need a formula saying that if the (i,j) spot is NOT near the head and $z_{i,j,\sigma}$ then $z_{i+1,j,\sigma}$.

Putting it All Together

On input x you output a formula ϕ constructed as follows

1. $t(|x|) = q(|x| + p(|x|))$. We call this t.
2. Variables $\{z_{i,j,\tau} : 1 \leq i, j \leq t, \tau \in \Sigma \cup (\Sigma \times Q)\}$.
3. Formula saying:
   3.1 For all $1 \leq i, j \leq t$, exists ONE σ with $z_{i,j,\sigma} = T$.
   3.2 C_1 is the start config with x.
   3.3 C_t is the accept config.
   3.4 For each instruction of the TM have a formula saying C_i goes to C_{i+1} if that instruction is relevant.
   3.5 If head is not within 2 square of (i, j) and $z_{ij\sigma}$ then $z_{(i+1)j\sigma}$.

Important Upshot

- If $\mathrm{SAT} \in \mathrm{P}$ then **every set in NP** is in P, so we would have $\mathrm{P} = \mathrm{NP}$.
- We will soon have more NP-complete problems.
- If **any** NP-complete problem is in P then $\mathrm{P} = \mathrm{NP}$.
- In the year 2000 the Clay Math Institute posted seven math problems and offered $1,000,000 for the solution to any of them. Resolving P vs NP was one of them.

Variants of SAT: Which ones are Hard? I

1. SAT is the set of all boolean formulas that are satisfiable.

Variants of SAT: Which ones are Hard? I

1. SAT is the set of all boolean formulas that are satisfiable. That is, $\phi(\vec{x}) \in SAT$ if there exists a vector $\vec{b}$ such that $\phi(\vec{b}) = TRUE$.

Variants of SAT: Which ones are Hard? I

1. SAT is the set of all boolean formulas that are satisfiable. That is, $\phi(\vec{x}) \in SAT$ if there exists a vector $\vec{b}$ such that $\phi(\vec{b}) = TRUE$. NP-Complete.

Variants of SAT: Which ones are Hard? I

1. SAT is the set of all boolean formulas that are satisfiable. That is, $\phi(\vec{x}) \in SAT$ if there exists a vector $\vec{b}$ such that $\phi(\vec{b}) = TRUE$. NP-Complete.
2. CNFSAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of literals.

Variants of SAT: Which ones are Hard? I

1. SAT is the set of all boolean formulas that are satisfiable. That is, $\phi(\vec{x}) \in SAT$ if there exists a vector $\vec{b}$ such that $\phi(\vec{b}) = TRUE$. NP-Complete.
2. CNFSAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of literals. NP-complete.

Variants of SAT: Which ones are Hard? I

1. SAT is the set of all boolean formulas that are satisfiable. That is, $\phi(\vec{x}) \in SAT$ if there exists a vector $\vec{b}$ such that $\phi(\vec{b}) = TRUE$. NP-Complete.
2. CNFSAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of literals. NP-complete. The proof of Cook-Levin yields a CNF formula.

Variants of SAT: Which ones are Hard? I

1. SAT is the set of all boolean formulas that are satisfiable. That is, $\phi(\vec{x}) \in SAT$ if there exists a vector $\vec{b}$ such that $\phi(\vec{b}) = TRUE$. NP-Complete.
2. CNFSAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of literals. NP-complete. The proof of Cook-Levin yields a CNF formula.
3. k-SAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of exactly k literals.

Variants of SAT: Which ones are Hard? I

1. SAT is the set of all boolean formulas that are satisfiable. That is, $\phi(\vec{x}) \in SAT$ if there exists a vector $\vec{b}$ such that $\phi(\vec{b}) = TRUE$. NP-Complete.
2. CNFSAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of literals. NP-complete. The proof of Cook-Levin yields a CNF formula.
3. k-SAT is the set of all boolean formulas in SAT of the form $C_1 \wedge \cdots \wedge C_m$ where each C_i is an $\vee$ of exactly k literals. 3-SAT is NP-complete, 2-SAT is in Poly Time.

Variants of SAT: Which ones are Hard? II

1. DNFSAT is the set of all boolean formulas in SAT of the form $C_1 \vee \cdots \vee C_m$ where each C_i is an $\wedge$ of literals.

Variants of SAT: Which ones are Hard? II

1. DNFSAT is the set of all boolean formulas in SAT of the form $C_1 \vee \cdots \vee C_m$ where each C_i is an $\wedge$ of literals. Poly Time. If some C_i does not have (say) both x and $\neg x$ then satisfiable, else not.
2. k-DNFSAT is the set of all boolean formulas in SAT of the form $C_1 \vee \cdots \vee C_m$ where each C_i is an $\wedge$ of exactly k literals.

Variants of SAT: Which ones are Hard? II

1. DNFSAT is the set of all boolean formulas in SAT of the form $C_1 \vee \cdots \vee C_m$ where each C_i is an $\wedge$ of literals. Poly Time. If some C_i does not have (say) both x and $\neg x$ then satisfiable, else not.
2. k-DNFSAT is the set of all boolean formulas in SAT of the form $C_1 \vee \cdots \vee C_m$ where each C_i is an $\wedge$ of exactly k literals. Poly Time since DNFSAT is Poly Time.

CNFSAT Hard;DNFSAT Easy.
CNFSAT $\rightarrow$ DNFSAT. Collect \$1,000,000

Idea Given ϕ in CNF form, convert to DNF form, solve DNF-SAT problem in Poly time, and now know if ϕ is in SAT.

CNFSAT Hard;DNFSAT Easy.
CNFSAT $\rightarrow$ DNFSAT. Collect \$1,000,000

Idea Given ϕ in CNF form, convert to DNF form, solve DNF-SAT problem in Poly time, and now know if ϕ is in SAT.

Show me the Money! \$1,000,000 is mine!

CNFSAT Hard;DNFSAT Easy.
CNFSAT $\rightarrow$ DNFSAT. Collect \$1,000,000

Idea Given ϕ in CNF form, convert to DNF form, solve DNF-SAT problem in Poly time, and now know if ϕ is in SAT.

Show me the Money! \$1,000,000 is mine!

Bad News This does not work.

CNFSAT Hard;DNFSAT Easy. CNFSAT $\rightarrow$ DNFSAT. Collect \$1,000,000

Idea Given ϕ in CNF form, convert to DNF form, solve DNF-SAT problem in Poly time, and now know if ϕ is in SAT.

Show me the Money! \$1,000,000 is mine!

Bad News This does not work.

Good News The reason it does not work is interesting.

CNFSAT Hard;DNFSAT Easy.
CNFSAT $\rightarrow$ DNFSAT. Collect \$1,000,000

Idea Given ϕ in CNF form, convert to DNF form, solve DNF-SAT problem in Poly time, and now know if ϕ is in SAT.

Show me the Money! \$1,000,000 is mine!

Bad News This does not work.

Good News The reason it does not work is interesting.

Bad News I'd rather have the \$1,000,000 than be enlightened.

Vote on CNF vs DNF

Vote on whether the following statement is TRUE or FALSE:
There is a **proof** *that* $\mathrm{CNFSAT} \le \mathrm{DNFSAT}$ *is NOT true. That is, there is NO poly time algorithm that will transform* ϕ *in CNF form to* ψ *in DNF form such that* $\phi \in \mathrm{SAT}$ *iff* $\psi \in \mathrm{SAT}$*.*

Vote on CNF vs DNF

Vote on whether the following statement is TRUE or FALSE:
There is a **proof** *that* $\mathrm{CNFSAT} \le \mathrm{DNFSAT}$ *is NOT true. That is, there is NO poly time algorithm that will transform* ϕ *in CNF form to* ψ *in DNF form such that* $\phi \in \mathrm{SAT}$ *iff* $\psi \in \mathrm{SAT}$.
TRUE, we Do have a proof!. Hard to believe.

Work with Neighbor

Convert the following into CNF form

1. $(x_1 \vee y_1)$
2. $(x_1 \vee y_1) \wedge (x_2 \vee y_2)$
3. $(x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge (x_3 \vee y_3)$
4. $(x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge (x_3 \vee y_3) \wedge (x_4 \wedge y_4)$

CNF vs DNF

Convert the following into DNF form

1. $(x_1 \vee y_1)$

CNF vs DNF

Convert the following into DNF form

1. $(x_1 \vee y_1)$
   $x_1 \vee y_1$
2. $(x_1 \vee y_1) \wedge (x_2 \vee y_2)$

CNF vs DNF

Convert the following into DNF form

1. $(x_1 \vee y_1)$
   $x_1 \vee y_1$
2. $(x_1 \vee y_1) \wedge (x_2 \vee y_2)$
   $(x_1 \wedge x_2) \vee (x_1 \wedge y_2) \vee (y_1 \wedge x_2) \vee (y_1 \vee y_2)$.
3. $(x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge (x_3 \vee y_3)$

CNF vs DNF

Convert the following into DNF form

1. $(x_1 \vee y_1)$
   $x_1 \vee y_1$
2. $(x_1 \vee y_1) \wedge (x_2 \vee y_2)$
   $(x_1 \wedge x_2) \vee (x_1 \wedge y_2) \vee (y_1 \wedge x_2) \vee (y_1 \vee y_2)$.
3. $(x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge (x_3 \vee y_3)$

   $(x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge y_3) \vee (x_1 \wedge y_2 \wedge x_3) \vee (x_1 \wedge y_2 \wedge y_3) \vee$

   $(y_1 \wedge x_2 \wedge x_3) \vee (y_1 \wedge x_2 \wedge y_3) \vee (y_1 \wedge y_2 \wedge x_3) \vee (y_1 \wedge y_2 \wedge y_3)$
4. $(x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge (x_3 \vee y_3) \wedge (x_4 \wedge y_4)$

CNF vs DNF

Convert the following into DNF form

1. $(x_1 \vee y_1)$
   $x_1 \vee y_1$
2. $(x_1 \vee y_1) \wedge (x_2 \vee y_2)$
   $(x_1 \wedge x_2) \vee (x_1 \wedge y_2) \vee (y_1 \wedge x_2) \vee (y_1 \vee y_2)$.
3. $(x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge (x_3 \vee y_3)$

   $(x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge y_3) \vee (x_1 \wedge y_2 \wedge x_3) \vee (x_1 \wedge y_2 \wedge y_3) \vee$

   $(y_1 \wedge x_2 \wedge x_3) \vee (y_1 \wedge x_2 \wedge y_3) \vee (y_1 \wedge y_2 \wedge x_3) \vee (y_1 \wedge y_2 \wedge y_3)$
4. $(x_1 \vee y_1) \wedge (x_2 \vee y_2) \wedge (x_3 \vee y_3) \wedge (x_4 \wedge y_4)$
   Not going to do it but it would take 16 clauses.