The Cook-Levin Thm

Exposition by William Gasarch—U of MD
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 $C_1 \land \cdots \land C_m$ where each $C_i$ is an $\lor$ of literals.

3. $k$-SAT is the set of all boolean formulas in SAT of the form $C_1 \land \cdots \land C_m$ where each $C_i$ is an $\lor$ of exactly $k$ literals.

4. DNFSAT is the set of all boolean formulas in SAT of the form $C_1 \lor \cdots \lor C_m$ where each $C_i$ is an $\land$ of literals.

5. $k$-DNFSAT is the set of all boolean formulas in SAT of the form $C_1 \lor \cdots \lor C_m$ where each $C_i$ is an $\land$ 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 \(h\).
- \(\Sigma\) is a finite alphabet. It contains the symbol \#.
- \(\delta : (Q \setminus \{h\}) \times \Sigma \rightarrow Q \times \Sigma \cup \{R, L\}\)
- \(s \in Q\) 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

   \[ abba###\ldots \]

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:

$$abba\#ab\text{c}\text{ab}\#a\#\#\#\#\cdots$$

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.
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 $\delta(q, b) = (p, a)$

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>a</th>
<th>b</th>
<th>b</th>
<th>(b, s)</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>(b, q)</td>
<td>b</td>
<td>#</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>(a, p)</td>
<td>b</td>
<td>#</td>
</tr>
</tbody>
</table>

- The left endpoint is the end of the tape.
- The unseen symbols on the right are all #
How to Represent an NP Computation

Let $X \in \text{NP}$. 
Let $X \in \text{NP}$. 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 $X \in \text{NP}$. 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 $X \in \text{NP}$. 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 $\leq q(|x| + |y|) = q(|x| + p(|x|)).$
Let $t(n) = q(n + p(n))$, a poly.
How to Represent an NP Computation

Let $X \in \text{NP}$. 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 $\leq q(|x| + |y|) = q(|x| + p(|x|))$. Let $t(n) = q(n + p(n))$, a poly. Here is ALL that matters:

- Numb of steps $M(x, y)$ takes is $\leq t(|x|)$. Hence $\leq t(|x|)$ 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 #.

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.
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) # ··· #
```

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

$$X = \{ x : (\exists y)[|y| = q(|x|) \land M(x, y) = Y] \}$$
Summary of What’s Important

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

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

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

SAT is NP-complete.

We need to prove two things:

1. SAT ∈ NP.

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

Formally

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

The satisfying assignment is the witness.

2. For all \( X \in NP \), \( X \leq SAT \). This is the bulk of the proof.
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 $\delta$.
- $C_t$ 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?
KEY 1: We have variables for every possible entry in every possible configuration. The variables are

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

If there is an accepting sequence of configurations then $z_{i,j,\sigma} = T$ iff the $j$th symbol in the $i$th configuration is $\sigma$. 
Making the $z_{i,j,\sigma}$ Make Sense

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

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

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

$$z_{i,j,\sigma} \rightarrow \bigwedge_{\tau \in \Sigma \cup (\Sigma \times Q) \setminus \{\sigma\}} \neg z_{i,j,\tau}$$
$C_1$ 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:

$$\bigwedge_{j=q(n)+n+3}^{t(n)} Z_{1,j,\#}$$
$C_1$ 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

$\leq 2 \times 7 = 14$ steps.

Formula saying $C_1$ codes $x$ as input is

$$Z_{1,1,a} \land Z_{1,2,(b,s)} \land Z_{1,3,\#} \land$$

$$(Z_{1,4,a} \lor Z_{1,4,b}) \land (Z_{1,5,a} \lor Z_{1,5,b}) \land (Z_{1,6,a} \lor Z_{1,6,b}) \land (Z_{1,7,a} \lor Z_{1,7,b}) \land$$

$$Z_{1,8,\#} \land \cdots \land Z_{1,23,\#}$$
$C_t$ 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:

<table>
<thead>
<tr>
<th>$\sigma_1$</th>
<th>$(a, q)$</th>
<th>$\sigma_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_1$</td>
<td>$(b, p)$</td>
<td>$\sigma_2$</td>
</tr>
</tbody>
</table>

Formula is a $\land$ over relevant $i, j, \sigma_1, \sigma_2$ of:

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

$\left( z_{(i+1)j}\sigma_1 \land z_{(i+1)(j+1),(b,p)} \land z_{(i+1),(j+2)\sigma_2} \right)$
Thought Experiment: What if $\delta(q, a) = (p, L)$. Then:

\[
\begin{array}{ccc}
\sigma_1 & (a, q) & \sigma_2 \\
(\sigma_1, p) & a & \sigma_2 \\
\end{array}
\]

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 $\phi$ 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 $\sigma$ 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 $\text{SAT} \in \text{P}$ then every set in $\text{NP}$ is in $\text{P}$, so we would have $\text{P} = \text{NP}$.
- We will soon have more $\text{NP}$-complete problems.
- If any $\text{NP}$-complete problem is in $\text{P}$ then $\text{P} = \text{NP}$.
- 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 $\text{P}$ vs $\text{NP}$ was one of them.
Variants of SAT: Which ones are Hard? I

1. SAT is the set of all boolean formulas that are satisfiable.
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 $C_1 \land \cdots \land C_m$ where each $C_i$ is an $\lor$ 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 $C_1 \land \cdots \land C_m$ where each $C_i$ is an $\lor$ of exactly $k$ literals. 3-SAT is NP-complete, 2-SAT is in Poly Time.
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.
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 \( C_1 \land \cdots \land C_m \) where each \( C_i \) is an \( \lor \) 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 $C_1 \land \cdots \land C_m$ where each $C_i$ is an $\lor$ 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 $C_1 \land \cdots \land C_m$ where each $C_i$ is an $\lor$ of literals. NP-complete. The proof of Cook-Levin yields a CNF formula.
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 $C_1 \land \cdots \land C_m$ where each $C_i$ is an $\lor$ 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 $C_1 \land \cdots \land C_m$ where each $C_i$ is an $\lor$ 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 $C_1 \land \cdots \land C_m$ where each $C_i$ is an $\lor$ 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 $C_1 \land \cdots \land C_m$ where each $C_i$ is an $\lor$ of exactly $k$ literals. 3-SAT is NP-complete, 2-SAT is in Poly Time.
1. DNFSAT is the set of all boolean formulas in SAT of the form $C_1 \lor \cdots \lor C_m$ where each $C_i$ is an $\land$ of literals.
1. DNFSAT is the set of all boolean formulas in SAT of the form $C_1 \lor \cdots \lor C_m$ where each $C_i$ is an $\land$ 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 $C_1 \lor \cdots \lor C_m$ where each $C_i$ is an $\land$ of exactly $k$ literals.
1. DNFSAT is the set of all boolean formulas in SAT of the form $C_1 \lor \cdots \lor C_m$ where each $C_i$ is an $\land$ 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 $C_1 \lor \cdots \lor C_m$ where each $C_i$ is an $\land$ of exactly $k$ literals. Poly Time since DNFSAT is Poly Time.
CNFSAT Hard; DNFSAT Easy.

CNFSAT $\rightarrow$ DNFSAT. Collect $1,000,000$

**Idea** Given $\phi$ in CNF form, convert to DNF form, solve DNF-SAT problem in Poly time, and now know if $\phi$ is in SAT.
CNFSAT Hard; DNFSAT Easy.  
CNFSAT $\rightarrow$ DNFSAT. Collect $1,000,000$

**Idea** Given $\phi$ in CNF form, convert to DNF form, solve DNF-SAT problem in Poly time, and now know if $\phi$ 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 $\phi$ in CNF form, convert to DNF form, solve DNF-SAT problem in Poly time, and now know if $\phi$ 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 $\phi$ in CNF form, convert to DNF form, solve DNF-SAT problem in Poly time, and now know if $\phi$ 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 $\phi$ in CNF form, convert to DNF form, solve DNF-SAT problem in Poly time, and now know if $\phi$ 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 whether the following statement is TRUE or FALSE:

There is a proof that CNFSAT ≤ DNFSAT is NOT true. That is, there is NO poly time algorithm that will transform $\phi$ in CNF form to $\psi$ in DNF form such that $\phi \in \text{SAT}$ iff $\psi \in \text{SAT}$. 

TRUE, we DO have a proof! Hard to believe.
Vote on CNF vs DNF

Vote on whether the following statement is TRUE or FALSE:

*There is a proof that CNFSAT $\leq$ DNFSAT is NOT true. That is, there is NO poly time algorithm that will transform $\phi$ in CNF form to $\psi$ in DNF form such that $\phi \in \text{SAT}$ iff $\psi \in \text{SAT}$. TRUE, we Do have a proof!. Hard to believe."
Convert the following into CNF form

1. \((x_1 \lor y_1)\)
2. \((x_1 \lor y_1) \land (x_2 \lor y_2)\)
3. \((x_1 \lor y_1) \land (x_2 \lor y_2) \land (x_3 \lor y_3)\)
4. \((x_1 \lor y_1) \land (x_2 \lor y_2) \land (x_3 \lor y_3) \land (x_4 \land y_4)\)
CNF vs DNF

Convert the following into DNF form

1. \((x_1 \lor y_1)\)
Convert the following into DNF form

1. \((x_1 \lor y_1)\)
   \[x_1 \lor y_1\]

2. \((x_1 \lor y_1) \land (x_2 \lor y_2)\)
CNF vs DNF

Convert the following into DNF form

1. \((x_1 \lor y_1)\)
   \[x_1 \lor y_1\]

2. \((x_1 \lor y_1) \land (x_2 \lor y_2)\)
   \[(x_1 \land x_2) \lor (x_1 \land y_2) \lor (y_1 \land x_2) \lor (y_1 \land y_2).\]

3. \((x_1 \lor y_1) \land (x_2 \lor y_2) \land (x_3 \lor y_3)\)
CNF vs DNF

Convert the following into DNF form

1. \((x_1 \lor y_1)\)
   \[x_1 \lor y_1\]

2. \((x_1 \lor y_1) \land (x_2 \lor y_2)\)
   \[ (x_1 \land x_2) \lor (x_1 \land y_2) \lor (y_1 \land x_2) \lor (y_1 \lor y_2). \]

3. \((x_1 \lor y_1) \land (x_2 \lor y_2) \land (x_3 \lor y_3)\)
   \[ (x_1 \land x_2 \land x_3) \land (x_1 \land x_2 \land y_3) \land (x_1 \land y_2 \land x_3) \land (x_1 \land y_2 \land y_3) \land \]
   \[ (y_1 \land x_2 \land x_3) \land (y_1 \land x_2 \land y_3) \land (y_1 \land y_2 \land x_3) \land (y_1 \land y_2 \land y_3) \]

4. \((x_1 \lor y_1) \land (x_2 \lor y_2) \land (x_3 \lor y_3) \land (x_4 \lor y_4)\)
CNF vs DNF

Convert the following into DNF form

1. \((x_1 \lor y_1)\)
   \[x_1 \lor y_1\]

2. \((x_1 \lor y_1) \land (x_2 \lor y_2)\)
   \[(x_1 \land x_2) \lor (x_1 \land y_2) \lor (y_1 \land x_2) \lor (y_1 \land y_2).\]

3. \((x_1 \lor y_1) \land (x_2 \lor y_2) \land (x_3 \lor y_3)\)
   \[(x_1 \land x_2 \land x_3) \land (x_1 \land x_2 \land y_3) \land (x_1 \land y_2 \land x_3) \land (x_1 \land y_2 \land y_3) \land \]
   \[(y_1 \land x_2 \land x_3) \land (y_1 \land x_2 \land y_3) \land (y_1 \land y_2 \land x_3) \land (y_1 \land y_2 \land y_3) \land \]
   \[\]

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