TIME COMPLEXITY AND POLYNOMIAL TIME;
NON DETERMINISTIC TURING MACHINES AND NP

THURSDAY Mar 20
COMPLEXITY THEORY

Studieds what can and can’t be computed under limited resources such as time, space, etc.

Today: Time complexity
Definition:

Suppose $M$ is a TM that halts on all inputs. The **running time** or **time-complexity** of $M$ is the function $f : N \rightarrow N$, where $f(n)$ is the maximum number of steps that $M$ uses on any input of length $n$. 
MEASURING TIME COMPLEXITY

We measure time complexity by counting the elementary steps required for a machine to halt.

Consider the language $A = \{ 0^k1^k \mid k \geq 0 \}$

On input of length $n$:

1. Scan across the tape and reject if the string is not of the form $0^i1^j$

2. Repeat the following if both 0s and 1s remain on the tape:
   - Scan across the tape, crossing off a single 0 and a single 1

3. If 0s remain after all 1s have been crossed off, or vice-versa, reject. Otherwise accept.
ASYMPTOTIC ANALYSIS

\[ 5n^3 + 2n^2 + 22n + 6 = O(n^3) \]
Let $f$ and $g$ be two functions $f, g : \mathbb{N} \to \mathbb{R}^+$. We say that $f(n) = O(g(n))$ if there exist positive integers $c$ and $n_0$ so that for every integer $n \geq n_0$

$$f(n) \leq cg(n)$$

When $f(n) = O(g(n))$, we say that $g(n)$ is an asymptotic upper bound for $f(n)$

$f$ asymptotically NO MORE THAN $g$

$$5n^3 + 2n^2 + 22n + 6 = O(n^3)$$

If $c = 6$ and $n_0 = 10$, then $5n^3 + 2n^2 + 22n + 6 \leq cn^3$
\[2n^{4.1} + 200283n^4 + 2 = O(n^{4.1})\]

\[3n \log_2 n + 5n \log_2 \log_2 n = O(n \log_2 n)\]

\[n \log_{10} n^{78} = O(n \log_{10} n)\]

\[\log_{10} n = \frac{\log_2 n}{\log_2 10}\]

\[O(n \log_{10} n) = O(n \log_2 n) = O(n \log n)\]
Definition: \( \text{TIME}(t(n)) = \{ L \mid L \text{ is a language decided by a } O(t(n)) \text{ time Turing Machine} \} \)

\[ A = \{ 0^k1^k \mid k \geq 0 \} \in \text{TIME}(n^2) \]
Big-oh necessary

- Moral: big-oh notation **necessary** given our model of computation
  - Recall: $f(n) = O(g(n))$ if there exists $c$ such that $f(n) \leq c \cdot g(n)$ for all sufficiently large $n$.
  - TM model incapable of making distinctions between time and space usage that differs by a constant.
Theorem: Suppose TM M decides language L in time $f(n)$. Then for any $\varepsilon > 0$, there exists TM M’ that decides L in time $
olinebreak \varepsilon f(n) + n + 2$.

Proof:
- simple idea: increase “word length”
- M’ will have
  - one more tape than M
  - m-tuples of symbols of M
    $$\Sigma_{\text{new}} = \Sigma_{\text{old}} \cup \Sigma_{\text{old}}^m$$
  - many more states
Linear Speedup

- part 1: compress input onto fresh tape

```
a b a b b a a a
aba bba aa_
```

...
Linear Speedup

- part 2: simulate M, m steps at a time

\[
\begin{array}{cccccccc}
  & b & b & a & a & b & a & b & a & a & a & b & \ldots \\
\end{array}
\]

\[
\begin{array}{cccccccc}
  \text{m} &  &  &  &  &  &  &  \\
\end{array}
\]

\[
\begin{array}{cccccccc}
  & \text{abb} & \text{aab} & \text{aba} & \text{aab} & \text{aba} & \ldots \\
\end{array}
\]

- 4 (L,R,R,L) steps to read relevant symbols, “remember” in state
- 2 (L,R or R,L) to make M’s changes
Linear Speedup

• accounting:
  – part 1 (copying): n + 2 steps
  – part 2 (simulation): 6 (f(n)/m)
  – set m = 6/\varepsilon
  – total: \varepsilon f(n) + n + 2

**Theorem**: Suppose TM M decides language L in space f(n). Then for any \varepsilon > 0, there exists TM M’ that decides L in space \varepsilon f(n) + 2.

• Proof: same.
A = \{ 0^k1^k \mid k \geq 0 \} \in \text{TIME}(n\log n)

Cross off every other 0 and every other 1. If the number of 0s and 1s left on the tape is odd, reject

```
00000000000011111111111111
x0x0x0x0x0x0x0xx1x1x1x1x1x1x
xxx0xxx0xxx0xxxx1xxx1xxx1xxx1x
xxxxxxx0xxxxxx0xxxxxxx1xxxx1xxxxx
xxxxxxxxxxxx0xxxxxxxxxxxx1xxxxxxx
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```
We can prove that a one-tape TM cannot decide A in less time than $O(n \log n)$.

*7.49 Extra Credit. Let $f(n) = o(n \log n)$. Then Time$(f(n))$ contains only regular languages.

where $f(n) = o(g(n))$ iff $\lim_{n \to \infty} f(n)/g(n) = 0$

ie, for all $c > 0$, $\exists n_0$ such that $f(n) < cg(n)$ for all $n \geq n_0$

f asymptotically LESS THAN g
Can $A = \{ 0^k1^k \mid k \geq 0 \}$ be decided in time $O(n)$ with a two-tape TM?

Scan all 0s and copy them to the second tape. Scan all 1s, crossing off a 0 from the second tape for each 1.
Different models of computation yield different running times for the same language!
Theorem: Let $t(n)$ be a function such that $t(n) \geq n$. Then every $t(n)$-time multi-tape TM has an equivalent $O(t(n)^2)$ single tape TM.

Claim: Simulating each step in the multi-tape machine uses at most $O(t(n))$ steps on a single-tape machine. Hence total time of simulation is $O(t(n)^2)$. 
MULTITAPE TURING MACHINES

FINITE STATE CONTROL

\[ \delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R\}^k \]
Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine

\[ \delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R\}^k \]
Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine.
Theorem: Every Multitape Turing Machine can be transformed into a single tape Turing Machine.
Analysis: (Note, $k$, the # of tapes, is fixed.)

Let $S$ be simulator

- Put $S$’s tape in proper format: $O(n)$ steps
- **Two scans** to simulate one step,
  1. to obtain info for next move $O(t(n))$ steps, why?
  2. to simulate it (may need to shift everything over to right possibly $k$ times): $O(t(n))$ steps, why?
\[ P = \bigcup_{k \in \mathbb{N}} \text{TIME}(n^k) \]
NON-DETERMINISTIC TURING MACHINES AND NP
Definition: A Non-Deterministic TM is a 7-tuple $T = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where:

- $Q$ is a finite set of states
- $\Sigma$ is the input alphabet, where $\square \notin \Sigma$
- $\Gamma$ is the tape alphabet, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \rightarrow 2^{(Q \times \Gamma \times \{L,R\})}$
- $q_0 \in Q$ is the start state
- $q_{accept} \in Q$ is the accept state
- $q_{reject} \in Q$ is the reject state, and $q_{reject} \neq q_{accept}$
NON-DETERMINISTIC TMs

...are just like standard TMs, except:

1. The machine may proceed according to several possibilities

2. The machine accepts a string if there exists a path from start configuration to an accepting configuration
Deterministic Computation

Non-Deterministic Computation

accept or reject

accept

reject

accept
Definition: Let $M$ be a NTM that is a decider (ie all branches halt on all inputs).
The running time or time-complexity of $M$ is the function $f : N \rightarrow N$, where $f(n)$ is the maximum number of steps that $M$ uses on any branch of its computation on any input of length $n$. 
Theorem: Let $t(n)$ be a function such that $t(n) \geq n$. Then every $t(n)$-time nondeterministic single-tape TM has an equivalent $2^{O(t(n))}$ deterministic single tape TM.
Definition: \( \text{NTIME}(t(n)) = \{ L \mid L \text{ is decided by a } O(t(n))-\text{time non-deterministic Turing machine} \} \)

\[ \text{TIME}(t(n)) \subseteq \text{NTIME}(t(n)) \]
A satisfying assignment is a setting of the variables that makes the formula true.

\[ \phi = (\neg x \land y) \lor z \]

\(x = 1, y = 1, z = 1\) is a satisfying assignment for \(\phi\).
A Boolean formula is **satisfiable** if there exists a satisfying assignment for it.

**YES** \( a \land b \land c \land \neg d \)

**NO** \( \neg(x \lor y) \land x \)

\[
\text{SAT} = \{ \phi \mid \phi \text{ is a satisfiable Boolean formula} \} 
\]
A 3cnf-formula is of the form:

\[(x_1 \lor \neg x_2 \lor x_3) \land (x_4 \lor x_2 \lor x_5) \land (x_3 \lor \neg x_2 \lor \neg x_1)\]

\[\text{literals}\]

\[\text{clauses}\]

\[
\begin{align*}
\text{YES} & \quad (x_1 \lor \neg x_2 \lor x_1) \\
\text{NO} & \quad (x_3 \lor x_1) \land (x_3 \lor \neg x_2 \lor \neg x_1) \\
\text{NO} & \quad (x_1 \lor x_2 \lor x_3) \land (\neg x_4 \lor x_2 \lor x_1) \lor (x_3 \lor x_1 \lor \neg x_1) \\
\text{NO} & \quad (x_1 \lor \neg x_2 \lor x_3) \land (x_3 \land \neg x_2 \land \neg x_1)
\end{align*}
\]

\[3\text{SAT} = \{ \phi \mid \phi \text{ is a satisfiable 3cnf-formula } \} \]
3SAT = \{ \phi \ | \ \phi \text{ is a satisfiable 3cnf-formula} \} 

**Theorem:** 3SAT \in \text{NTIME}(n^2)

**On input** \( \phi \):

1. Check if the formula is in 3cnf
2. For each variable, non-deterministically substitute it with 0 or 1
3. Test if the assignment satisfies \( \phi \)

\[
(x \lor \neg y \lor x) \\
(0 \lor \neg y \lor 0) \\
(0 \lor \neg 0 \lor 0) \\
(0 \lor \neg 1 \lor 0)
\]
NP = \bigcup_{k \in \mathbb{N}} \text{NTIME}(n^k)
Theorem: $L \in \text{NP} \iff$ if there exists a poly-time Turing machine $V(\text{erifier})$ with

$L = \{ x \mid \exists y (\text{witness}) |y| = \text{poly}(|x|) \text{ and } V(x,y) \text{ accepts} \}$

Proof:

(1) If $L = \{ x \mid \exists y |y| = \text{poly}(|x|) \text{ and } V(x,y) \text{ accepts} \}$ then $L \in \text{NP}$

Because we can guess $y$ and then run $V$

(2) If $L \in \text{NP}$ then

$L = \{ x \mid \exists y |y| = \text{poly}(|x|) \text{ and } V(x,y) \text{ accepts} \}$

Let $N$ be a non-deterministic poly-time TM that decides $L$ and define $V(x,y)$ to accept if $y$ is an accepting computation history of $N$ on $x$
3SAT = \{ \phi \mid \exists y \text{ such that } y \text{ is a satisfying assignment to } \phi \text{ and } \phi \text{ is in 3cnf} \} \\
SAT = \{ \phi \mid \exists y \text{ such that } y \text{ is a satisfying assignment to } \phi \}
A language is in NP if and only if there exist **polynomial-length certificates** for membership to the language.

SAT is in NP because a satisfying assignment is a polynomial-length certificate that a formula is satisfiable.

* that can be verified in poly-time
HAMiltonian paths
HAMPATH = \{ (G,s,t) \mid G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t \} \\

**Theorem:** HAMPATH \( \in \) NP

The Hamilton path itself is a certificate
K-CLIQUES
CLIQUE = \{ (G,k) \mid G \text{ is an undirected graph with a } k\text{-clique} \} \\

**Theorem:** CLIQUE \in NP \\

The k-clique itself is a certificate
NP = all the problems for which once you have the answer it is easy (i.e. efficient) to verify
POLY-TIME REDUCIBILITY

$f : \Sigma^* \rightarrow \Sigma^*$ is a polynomial time computable function if some poly-time Turing machine $M$, on every input $w$, halts with just $f(w)$ on its tape.

Language $A$ is polynomial time reducible to language $B$, written $A \leq_p B$, if there is a poly-time computable function $f : \Sigma^* \rightarrow \Sigma^*$ such that:

$$w \in A \iff f(w) \in B$$

$f$ is called a polynomial time reduction of $A$ to $B$. 
Theorem: If $A \leq_p B$ and $B \in P$, then $A \in P$

Proof: Let $M_B$ be a poly-time (deterministic) TM that decides $B$ and let $f$ be a poly-time reduction from $A$ to $B$

We build a machine $M_A$ that decides $A$ as follows:

On input $w$:

1. Compute $f(w)$
2. Run $M_B$ on $f(w)$
**Definition:** A language B is NP-complete if:

1. $B \in \text{NP}$
2. Every A in NP is poly-time reducible to B (i.e. B is NP-hard)
Suppose B is NP-Complete

So, if B is NP-Complete and $B \in P$ then $NP = P$. Why?
Theorem (Cook-Levin): SAT is NP-complete

Corollary: SAT ∈ P if and only if P = NP
NP-COMPLETENESS:
THE COOK-LEVIN THEOREM
Theorem (Cook-Levin.’71): SAT is NP-complete

Corollary: SAT ∈ P if and only if P = NP
Theorem (Cook-Levin): SAT is NP-complete

Proof:

(1) SAT ∈ NP

(2) Every language A in NP is polynomial time reducible to SAT

We build a poly-time reduction from A to SAT

The reduction turns a string w into a 3-cnf formula φ such that w ∈ A iff φ ∈ 3-SAT.

φ will simulate the NP machine N for A on w.

Let N be a non-deterministic TM that decides A in time n^k. How do we know N exists?
So proof will also show:
3-SAT is NP-Complete
The reduction $f$ turns a string $w$ into a 3-cnf formula $\phi$ such that: $w \in A \iff \phi \in 3\text{SAT}$. $\phi$ will “simulate” the NP machine $N$ for $A$ on $w$. 
Deterministic Computation

accept or reject

Non-Deterministic Computation

accept

\[ \text{reject} \]

\[ \exp(n^k) \]
Suppose $A \in \text{NTIME}(n^k)$ and let $N$ be an NP machine for $A$. A tableau for $N$ on $w$ is an $n^k \times n^k$ table whose rows are the configurations of some possible computation of $N$ on input $w$. 

\[
\begin{array}{cccccccc}
\# & q_0 & w_1 & w_2 & \ldots & w_n & \square & \ldots & \square & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\# & \# & \# & \# & \# & \# & \# & \# & \# & \# \\
\end{array}
\]
A tableau is **accepting** if any row of the tableau is an accepting configuration.

Determining whether $N$ accepts $w$ is equivalent to determining whether there is an accepting tableau for $N$ on $w$.

Given $w$, our 3cnf-formula $\phi$ will describe a *generic* tableau for $N$ on $w$ (in fact, essentially *generic* for $N$ on any string $w$ of length $n$).

The 3cnf formula $\phi$ will be satisfiable *if and only if* there is an accepting tableau for $N$ on $w$. 
VARIABLES of $\phi$

Let $C = Q \cup \Gamma \cup \{ # \}$

Each of the $(n^k)^2$ entries of a tableau is a cell

cell[i,j] = the cell at row $i$ and column $j$

For each $i$ and $j$ ($1 \leq i, j \leq n^k$) and for each $s \in C$ we have a variable $x_{i,j,s}$

# variables = $|C|n^{2k}$, ie $O(n^{2k})$, since $|C|$ only depends on $N$

These are the variables of $\phi$ and represent the contents of the cells

We will have: $x_{i,j,s} = 1 \iff \text{cell}[i,j] = s$
\[ x_{i,j,s} = 1 \]

means

\[ \text{cell}[i,j] = s \]
We now design $\phi$ so that a satisfying assignment to the variables $x_{i,j,s}$ corresponds to an accepting tableau for $N$ on $w$.

The formula $\phi$ will be the AND of four parts:

$$\phi = \phi_{\text{cell}} \land \phi_{\text{start}} \land \phi_{\text{accept}} \land \phi_{\text{move}}$$

$\phi_{\text{cell}}$ ensures that for each $i,j$, exactly one $x_{i,j,s} = 1$.

$\phi_{\text{start}}$ ensures that the first row of the table is the starting (initial) configuration of $N$ on $w$.

$\phi_{\text{accept}}$ ensures* that an accepting configuration occurs somewhere in the table.

$\phi_{\text{move}}$ ensures* that every row is a configuration that legally follows from the previous config.

*if the other components of $\phi$ hold
\( \phi_{\text{cell}} \) ensures that for each \( i,j \), exactly one \( x_{i,j,s} = 1 \)

\[
\phi_{\text{cell}} = \bigwedge_{1 \leq i, j \leq n^k} \left( \bigvee_{s \in C} x_{i,j,s} \right) \land \left( \bigwedge_{s,t \in C} (\neg x_{i,j,s} \lor \neg x_{i,j,t}) \right)
\]

- at least one variable is turned on
- at most one variable is turned on
$$\phi_{\text{start}} = x_{1,1,\#} \land x_{1,2,q_0} \land$$

$$x_{1,3,w_1} \land x_{1,4,w_2} \land \ldots \land x_{1,n+2,w_n} \land$$

$$x_{1,n+3,\square} \land \ldots \land x_{1,n^k-1,\square} \land x_{1,n^k,\#}$$
\( \phi_{\text{accept}} \) ensures that an accepting configuration occurs somewhere in the table

\[
\phi_{\text{accept}} = \bigvee_{1 \leq i, j \leq n^k} x_{i,j,q_{\text{accept}}}
\]
\( \phi_{\text{move}} \) ensures that every row is a configuration that legally follows from the previous.

It works by ensuring that each \( 2 \times 3 \) "window" of cells is **legal** (does not violate N’s rules).
If $\delta(q_1,a) = \{(q_1,b,R)\}$ and $\delta(q_1,b) = \{(q_2,c,L), (q_2,a,R)\}$

Which of the following windows are legal:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>q_1</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>q_2</td>
<td>a</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>q_1</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>q_1</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>a</td>
<td>q_1</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>#</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>q_1</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>q_2</td>
<td>b</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>a</td>
<td>q_1</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>
If $\delta(q_1, a) = \{(q_1, b, R)\}$ and $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$

Which of the following windows are legal:
CLAIM:
If
• the top row of the tableau is the start configuration, and
• and every window is legal,
Then
each row of the tableau is a configuration that legally follows the preceding one.
CLAIM:
If
• the top row of the tableau is the start configuration, and
• and every window is legal,

Then
each row of the tableau is a configuration that legally follows the preceding one.

Proof:
In upper configuration, every cell that doesn’t contain the boundary symbol #, is the center top cell of a window.
CLAIM:
If
• the top row of the tableau is the start configuration, and
• and every window is legal,

Then
each row of the tableau is a configuration that legally follows the preceding one.

Proof:
In upper configuration, every cell that doesn’t contain the boundary symbol #, is the center top cell of a window.

Case 1. center cell of window is a non-state symbol and not adjacent to a state symbol
CLAIM:
If
• the top row of the tableau is the start configuration, and
• and every window is legal,
Then
each row of the tableau is a configuration that legally follows the preceding one.

Proof:
In upper configuration, every cell that doesn’t contain the boundary symbol #, is the center top cell of a window.

Case 1. center cell of window is a non-state symbol and not adjacent to a state symbol
Case 2. center cell of window is a state symbol
<table>
<thead>
<tr>
<th>#</th>
<th>q₀</th>
<th>w₁</th>
<th>w₂</th>
<th>w₃</th>
<th>w₄</th>
<th>...</th>
<th>wₙ</th>
<th>□</th>
<th>...</th>
<th>□</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>ok</td>
<td>ok</td>
<td>w₂</td>
<td>w₃</td>
<td>w₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#</td>
</tr>
<tr>
<td></td>
<td>$q_0$</td>
<td>$w_1$</td>
<td>$w_2$</td>
<td>$w_3$</td>
<td>$w_4$</td>
<td>...</td>
<td>$w_n$</td>
<td>□</td>
<td>...</td>
<td>□</td>
<td>#</td>
</tr>
<tr>
<td>---</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-----</td>
<td>-------</td>
<td>---</td>
<td>-----</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>#</td>
<td>ok</td>
<td>ok</td>
<td>$w_2$</td>
<td>$w_3$</td>
<td>$w_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#</td>
</tr>
<tr>
<td>#</td>
<td>$q_0$</td>
<td>$w_1$</td>
<td>$w_2$</td>
<td>$w_3$</td>
<td>$w_4$</td>
<td>...</td>
<td>$w_n$</td>
<td>#</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#</td>
<td>ok</td>
<td>ok</td>
<td>$w_2$</td>
<td>$w_3$</td>
<td>$w_4$</td>
<td>...</td>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>#</td>
<td>a₁</td>
<td>q</td>
<td>a₂</td>
<td>a₃</td>
<td>a₄</td>
<td>a₅</td>
<td>...</td>
<td>aₙ</td>
<td>□</td>
<td>...</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>----</td>
<td>---</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>-----</td>
<td>----</td>
<td>---</td>
<td>-----</td>
</tr>
<tr>
<td></td>
<td>#</td>
<td>ok</td>
<td>ok</td>
<td>ok</td>
<td>a₃</td>
<td>a₄</td>
<td>a₅</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The table shows a sequence of elements and their corresponding values. The highlighted area indicates the part of the table that is focused on.
<table>
<thead>
<tr>
<th>#</th>
<th>$a_1$</th>
<th>q</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
<th>$a_5$</th>
<th>…</th>
<th>$a_n$</th>
<th>□</th>
<th>…</th>
<th>□</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>ok</td>
<td>ok</td>
<td>ok</td>
<td>$a_3$</td>
<td>$a_4$</td>
<td>$a_5$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
So the lower configuration follows from the upper!!!
The \((i,j)\) Window

<table>
<thead>
<tr>
<th>row (i)</th>
<th>col. (j-1)</th>
<th>col. (j)</th>
<th>col. (j+1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>((i,j-1))</td>
<td>((i,j))</td>
<td>((i,j+1))</td>
<td></td>
</tr>
<tr>
<td>(a_1)</td>
<td>(a_2)</td>
<td>(a_3)</td>
<td></td>
</tr>
<tr>
<td>((i+1,j-1))</td>
<td>((i+1,j))</td>
<td>((i+1,j+1))</td>
<td></td>
</tr>
<tr>
<td>(a_4)</td>
<td>(a_5)</td>
<td>(a_6)</td>
<td></td>
</tr>
</tbody>
</table>
\[ \phi_{\text{move}} = \bigwedge ( \text{the } (i, j) \text{ window is legal} ) \]
\[ 1 \leq i, j \leq n^k \]

the \((i, j)\) window is legal = \[
\bigvee_{a_1, \ldots, a_6} ( x_{i,j-1,a_1} \land x_{i,j,a_2} \land x_{i,j+1,a} \land x_{i+1,j-1,a} \land x_{i+1,j,a} \land x_{i+1,j+1,a} )
\]

is a legal window

This is a disjunct over all \((\leq |C|^6)\) legal sequences \((a_1, \ldots, a_6)\).
\[ \phi_{\text{move}} = \bigwedge ( \text{the (i, j) window is legal} ) \]

\[ 1 \leq i, j \leq n^k \]

the (i, j) window is legal =

\[ \bigvee_{a_1, \ldots, a_6} ( x_{i,j-1,a_1} \land x_{i,j,a_2} \land x_{i,j+1,a} \land x_{i+1,j-1,a} \land x_{i+1,j,a} \land x_{i+1,j+1,a} ) \]

is a legal window

This is a disjunct over all \((\leq |C|^6)\) legal sequences \((a_1, \ldots, a_6)\).

This disjunct is satisfiable

\[ \iff \]

There is *some* assignment to the cells (ie variables) in the window \((i,j)\) that makes the window legal
\[ \phi_{\text{move}} = \bigwedge \ ( \text{the (i, j) window is legal} ) \]

\[ 1 \leq i, j \leq n^k \]

the (i, j) window is legal =

\[ \bigvee_{a_1, \ldots, a_6} \left( x_{i,j-1,a_1} \land x_{i,j,a_2} \land x_{i,j+1,a} \land x_{i+1,j-1,a} \land x_{i+1,j,a} \land x_{i+1,j+1,a} \right) \]

is a legal window

This is a disjunct over all (\( \leq |C|^6 \)) legal sequences (\( a_1, \ldots, a_6 \)).

So \( \phi_{\text{move}} \) is satisfiable

\[ \Leftrightarrow \]

There is some assignment to each of the variables that makes every window legal.
\[ \phi_{\text{move}} = \bigwedge \left( \text{the (i, j) window is legal } \right) \\
1 \leq i, j \leq n^k \]

the (i, j) window is legal =
\[
\bigvee_{a_1, \ldots, a_6}
\left( x_{i,j-1,a_1} \land x_{i,j,a_2} \land x_{i,j+1,a} \land x_{i+1,j-1,a} \land x_{i+1,j,a} \land x_{i+1,j+1,a} \right) 
\]

is a legal window

This is a disjunct over all (\( \leq |C|^6 \)) legal sequences \((a_1, \ldots, a_6)\).

Can re-write as equivalent conjunct:
\[
\equiv \bigwedge \left( \bigvee_{i,j-1,a_1} \lor \bigvee_{i,j,a_2} \lor \bigvee_{i,j+1,a} \lor \bigvee_{i+1,j-1,a} \lor \bigvee_{i+1,j,a} \lor \bigvee_{i+1,j+1,a} \right) a_1, \ldots, a_6
\]

ISN’T a legal window
\[ \phi = \phi_{\text{cell}} \land \phi_{\text{start}} \land \phi_{\text{accept}} \land \phi_{\text{move}} \]

\( \phi \) is satisfiable (i.e., there is some assignment to each of the variables s.t. \( \phi \) evaluates to 1)

\[ \iff \text{there is some assignment to each of the variables s.t.} \quad \phi_{\text{cell}} \land \phi_{\text{start}} \land \phi_{\text{accept}} \land \phi_{\text{move}} \text{ each evaluates to 1} \]

\[ \iff \text{There is some assignment of symbols to cells in the tableau such that:} \]

- The first row of the tableau is a start configuration and
- Every row of the tableau is a configuration that follows from the preceding by the rules of \( N \) and
- One row is an accepting configuration

\[ \iff \text{There is some accepting computation for } N \text{ with input } w \]
3-SAT?

How do we convert the whole thing into a 3-cnf formula?

Everything was an AND of ORs
We just need to make those ORs with 3 literals
3-SAT?

How do we convert the whole thing into a 3-cnf formula?

Everything was an AND of ORs
We just need to make those ORs with 3 literals

If a clause has less than three variables:

\[ a \equiv (a \lor a \lor a), \quad (a \lor b) \equiv (a \lor b \lor b) \]
3-SAT?

How do we convert the whole thing into a 3-cnf formula?

Everything was an AND of ORs
We just need to make those ORs with 3 literals

If a clause has less than three variables:

\[ a \equiv (a \lor a \lor a), \quad (a \lor b) \equiv (a \lor b \lor b) \]

If a clause has more than three variables:

\[ (a \lor b \lor c \lor d) \equiv (a \lor b \lor z) \land (\neg z \lor c \lor d) \]

\[ (a_1 \lor a_2 \lor \ldots \lor a_t) \equiv \]

\[ (a_1 \lor a_2 \lor z_1) \land (\neg z_1 \lor a_3 \lor z_2) \land (\neg z_2 \lor a_4 \lor z_3) \ldots \]
WHAT’S THE LENGTH OF $\phi$?
\[
\phi_{\text{cell}} = \bigwedge_{1 \leq i, j \leq n^k} \left( \bigvee_{s \in C} x_{i,j,s} \right) \land \left( \bigwedge_{s, t \in C \atop s \neq t} (\neg x_{i,j,s} \lor \neg x_{i,j,t}) \right)
\]

If a clause has less than three variables:

\[(a \lor b) = (a \lor b \lor b)\]
\[ \phi_{\text{cell}} = \bigwedge_{1 \leq i, j \leq n^k} \left[ \left( \bigvee_{s \in C} x_{i,j,s} \right)^{\wedge} \left( \bigwedge_{s,t \in C \atop s \neq t} (\neg x_{i,j,s} \lor \neg x_{i,j,t}) \right) \right] \]

\[ \text{O}(n^{2k}) \text{ clauses} \]

\[ \text{Length}(\phi_{\text{cell}}) = \text{O}(n^{2k}) \text{ O}(\log (n)) = \text{O}(n^{2k} \log n) \]

Length(indices)
\[ \phi_{\text{start}} = X_{1,1},# \land X_{1,2},q_0 \land \]
\[ X_{1,3},w_1 \land X_{1,4},w_2 \land \ldots \land X_{1,n+2},w_n \land \]
\[ X_{1,n+3},\square \land \ldots \land X_{1,n^{k-1}},\square \land X_{1,n^{k}},# \]
\[ = (X_{1,1},# \lor X_{1,1},# \lor X_{1,1},#) \land \]
\[ (X_{1,2},q_0 \lor X_{1,2},q_0 \lor X_{1,2},q_0) \land \ldots \land \]
\[ (X_{1,n^{k}},# \lor X_{1,n^{k}},# \lor X_{1,n^{k}},#) \]
\[ \phi_{\text{start}} = X_{1,1,#} \land X_{1,2,q} \land \\
X_{1,3,w_1} \land X_{1,4,w_2} \land \ldots \land X_{1,n+2,w_n} \land \\
X_{1,n+3,\square} \land \ldots \land X_{1,n^k-1,\square} \land X_{1,n^k,#} \]

\[ O(n^k) \]
$\phi_{\text{accept}} = \bigvee_{1 \leq i, j \leq n^k} x_{i,j,q_{\text{accept}}}$

$(a_1 \lor a_2 \lor \ldots \lor a_t) = (a_1 \lor a_2 \lor z_1) \land (\neg z_1 \lor a_3 \lor z_2) \land (\neg z_2 \lor a_4 \lor z_3) \ldots$
\[ \phi_{\text{accept}} = \bigvee_{1 \leq i, j \leq n^k} x_{i,j,q_{\text{accept}}} \]

\[ O(n^{2k}) \]
\[ \phi_{\text{move}} = \bigwedge ( \text{the } (i, j) \text{ window is legal} ) \]
\[
1 \leq i, j \leq n^k
\]

the \((i, j)\) window is legal =

\[ \bigwedge \left( \neg x_{i,j-1,a_1} \vee \neg x_{i,j,a_2} \vee \neg x_{i,j+1,a_3} \vee \neg x_{i+1,j-1,a_4} \vee \neg x_{i+1,j,a_5} \vee \neg x_{i+1,j+1,a_6} \right) \]

ISN’T a legal window

This is a conjunct over all \((\leq |C|^6)\) illegal sequences \((a_1, \ldots, a_6)\).

\[ O(n^{2k}) \]
Theorem (Cook-Levin): 3-SAT is NP-complete

Corollary: 3-SAT ∈ P if and only if P = NP