The classes **NP** and **coNP**.

Examples of Problems in **NP**.

Verifiability.

**Poly-Time Reductions**

**NP completeness**

**SAT** is **NP Complete**

Sipser, Chapter 7, Sections 7.3, 7.4, 7.5
Non-Deterministic Time (reminder)

Let $N$ be a non-deterministic TM, and let

$$f : \mathcal{N} \rightarrow \mathcal{N}$$

We say that $N$ runs in time $f(n)$ if

- For every input $x$ of length $n$,
- the maximum number of steps that $N$ uses,
- on any branch of its computation tree on $x$,
- is at most $f(n)$.
Let \( f : \mathbb{N} \rightarrow \mathbb{N} \) be a function.

**Definition:**

\[
\text{NTIME}(f(n)) = \{ L \mid L \text{ is a language, decided by an } O(f(n))-\text{time NTM} \}
\]
The Class NP

**Definition:** NP is the set of languages decidable in polynomial time on non-deterministic TMs.

\[ NP = \bigcup_{c \geq 0} \text{NTIME}(n^c) \]

- The class NP is
  - Invariant for all TMs with any number of tapes.
  - Insensitive to choice of reasonable non-deterministic computational model.
  - Roughly corresponds to problems whose positive solutions cannot be efficiently generated (⇒ intractable), but can be efficiently checked.
The Class NP

NP is important because it includes many problems of practical interest, e.g.

- Hamiltonian path
- Travelling salesman (salesperson, that is)
- Scheduling (operations research)
- Placement and routing (VLSI design)
- Composites (factoring/cryptography)

...
A Hamiltonian path in a directed $G$ visits each node exactly once.
Hamiltonian Path

\[
\text{HAMPATH} = \{ \langle G, s, t \rangle | G \text{ has Hamiltonian path from } s \text{ to } t \}\]

**Question:** How hard is it to decide this language?
Hamiltonian Path

\[ \text{HAMPATH} = \{ \langle G, s, t \rangle | G \text{ has Hamiltonian path from } s \text{ to } t \} \]

Easy to obtain exponential time algorithm:

- generate each potential path
- check whether it is Hamiltonian
The Class NP

Here is an NTM that decides HAMPATH in poly time.

On input \( \langle G, s, t \rangle \),

1. Guess and write down a list of numbers \( p_1, \ldots, p_m \), where \( m \) is number of nodes in \( G \), and \( 1 \leq p_i \leq m \).

2. Check for repetitions in list. If any found, reject.

3. Check whether \( p_1 = s \) and \( p_m = t \). If either does not hold, reject.

4. For \( i, 1 \leq i \leq m - 1 \), check whether \( (p_i, p_{i+1}) \) is an edge in \( G \). If any is not, reject. Otherwise accept.
On input \( \langle G, s, t \rangle \),

1. **Guess** and write down a list of numbers \( p_1, \ldots, p_m \) . . .
2. Check for repetitions . . .
3. Check whether \( p_1 = s \) and \( p_m = t \) . . .
4. Check whether \( (p_i, p_{i+1}) \) is an edge in \( G \) . . .

- Stage 1 polynomial time
- Stages 2 and 3 simple checks.
- Stage 4 simple poly-time too.
Hamiltonian Path

This problem has one very interesting feature: polynomial verifiability.

- we don’t know a fast way to find a Hamiltonian path
- but we can check whether a given path is Hamiltonian in polynomial time.

In other words,

- verifying correctness of a path is much easier
- than determining whether one exists
A natural number is composite if it is the product of two integers greater than one.

\[
\text{COMPOSITES} = \{ x \mid x = pq \text{ for integers } p, q > 1 \}
\]

- The best deterministic algorithms to factor \( x \) run in time this is in \( 2^{O(\sqrt[3]{n\log n})} \) (for \( n \)-bits inputs). Even 1000 bit numbers are way out of reach.

- We don’t know a deterministic polynomial-time algorithm for deciding this problem.

- But we can easily verify that a number is composite (how?)
Composite Numbers

A natural number is composite if it is the product of two integers greater than one.

\[ \text{COMPOSITES} = \{ x \mid x = pq \text{ for integers } p, q > 1 \} \]

Actually, in summer 2002, two Indian undergrads and their advisor found a deterministic polynomial time algorithm, so we now know that \( \text{COMPOSITES} \in \mathcal{P} \).

Interestingly, this later algorithm finds certificates for compositeness that are not the factors. Factoring is still a computationally hard problem.

However, let us pretend we’re still in 1/1/2002…
Verifiability

A verifier for a language $\mathcal{A}$ is an algorithm $\mathcal{V}$ where

$$\mathcal{A} = \{ w \mid \mathcal{V} \text{ accepts } \langle w, c \rangle \text{ for some string } c \}$$

- The verifier uses the additional information $c$ to verify $w \in \mathcal{A}$.
- We measure verifier run time by length of $w$.
- The string $c$ is called a certificate (or proof) for $w$ if $\mathcal{V}$ accepts $\langle w, c \rangle$.
- A polynomial verifier runs in polynomial time in $|w|$ (so $|c| \leq |w|^{O(1)}$).
- A language $\mathcal{A}$ is polynomially verifiable if it has a polynomial verifier.
Short Certificates for HAMPATH

For HAMPATH, a certificate for

\[ \langle G, s, t \rangle \in \text{HAMPATH} \]

is simply the Hamiltonian path from \( s \) to \( t \).

Can verify in time polynomial in \(|\langle G \rangle|\) whether given path is Hamiltonian.
Short Certificates for Compositeness

For COMPOSITES, a certificate for

\[ x \in \text{COMPOSITES} \]

is simply one of its divisors.

Can verify in time polynomial in \(|x|\) if given divisor indeed divides \(x\).
Verifiability

Not all problems are polynomially verifiable.

There is no known way to verify HAMPATH in polynomial time.

In fact, we will see many examples where $L$ is polynomially verifiable, but its complement, $\overline{L}$, is not known to be polynomially verifiable.
NP and Verifiability

Theorem: A language is in NP if and only if it has a polynomial time verifier.

Proof – Intuition:

- NTM simulates verifier by guessing the certificate.
- Verifier simulates NTM by using accepting branch as certificate.
Claim: If $A$ has a poly-time verifier, then it is decided by some polynomial-time NTM.

Let $V$ be poly-time verifier for $A$.

- single-tape TM
- runs in time $n^k$

$N$: on input $w$ of length $n$
- Nondeterministically select string $c$ of length $n^k$.
- Run $V$ on $\langle w, c \rangle$
- If $V$ accepts, accept; otherwise reject.
Claim: If $A$ is decided by a polynomial-time NTM $N$, running in time $n^k$, then $A$ has a poly-time verifier.

Construct polynomial-time verifier $V$ as follows.

$V$: on input $w$ of length $n$, and on a string $c$ of length $n^k$

- Simulate $N$ on input $w$, treating each symbol of $c$ as a description of the non-deterministic choice in each step of $N$.
- If this branch accepts, accept, otherwise reject.
A clique in a graph is a subgraph where every two nodes are connected by an edge.

A $k$-clique is a clique of size $k$.

What is the largest $k$-clique in the figure?
Examples: Clique

Define the language

\[
\text{CLIQUE} = \{ \langle G, k \rangle | G \text{ is an undirected graph with a } k\text{-clique} \}\]
Examples: Clique

Theorem:

\[ \text{CLIQUE} \in NP \]

The clique is the certificate.
Here is a verifier \( \mathcal{V} \): on input \( (\langle G, k \rangle, c) \)

- if \( c \) is not a \( k \)-clique, reject
- if \( G \) does not contain all vertices of \( c \), reject
- accept
Examples: SUBSET-SUM

An instance of the problem

- A collection of numbers $x_1, \ldots, x_k$
- Target number $t$
- Question: does some subcollection add up to $t$?

\[
\text{SUBSET-SUM} = \{ \langle S, t \rangle \mid S = \{x_1, \ldots, x_k\} \}
\]

\[\exists \ \{y_1, \ldots, y_\ell\} \subseteq \{x_1, \ldots, x_k\}, \sum_{y_j} = t\}

Collections are sets: repetitions not allowed.
Examples: SUBSET-SUM

We have

\[(\{4, 11, 16, 21, 27\}, 25) \in \text{SUBSET-SUM}\]

because \(4 + 21 = 25\).

\[(\{4, 11, 16, 21, 27\}, 26) \notin \text{SUBSET-SUM}\]
(why?)
Examples: SUBSET-SUM

Theorem:

SUBSET-SUM $\in NP$

The subset is the certificate.

Here is a verifier:

$\mathcal{V}$: on input $(\langle S, t \rangle, c)$

- test whether $c$ is a collection of numbers summing to $t$.
- test whether $c$ is a subset of $S$
- if either fail, reject, otherwise accept.
Complementary Problems

**CLIQUE** and **SUBSET-SUM** seem not to be members of NP. It is harder to efficiently verify that something does not exist than to efficiently verify that something does exist.

**Definition:** The class **coNP**: $L \in \text{coNP}$ if $\overline{L} \in \text{NP}$.

So far, no one knows if **coNP** is distinct from **NP** (recall first slide in lecture 10).
The question \( P = NP? \) is one of the great unsolved mysteries in contemporary mathematics.

- Most computer scientists believe the two classes are not equal.
- Most bogus proofs show them equal (why?)
Observations

If $\mathcal{P}$ differs from $\mathcal{NP}$, then the distinction between $\mathcal{P}$ and $\mathcal{NP} - \mathcal{P}$ is meaningful and important.

- languages in $\mathcal{P}$ tractable
- languages in $\mathcal{NP} - \mathcal{P}$ intractable

Until we can prove that $\mathcal{P} \neq \mathcal{NP}$, there is no hope of proving that a specific language lies in $\mathcal{NP} - \mathcal{P}$.

Nevertheless, we can prove statements of the form “If $\mathcal{P} \neq \mathcal{NP}$ then $A \in \mathcal{NP} - \mathcal{P}$.”
The class of **NP-complete** languages are

- “hardest” languages in $\mathcal{NP}$
- “least likely” to be in $\mathcal{P}$
- If any NP-complete $A \in \mathcal{P}$, then $\mathcal{NP} = \mathcal{P}$. 
Theorem: There is a language $S \in NP$ such that $S \in P$ if and only if $P = NP$.

This theorem establishes the class of NP-complete languages.

Such language, like Frodo Baggins, “carries on its back” the burden of all of $NP$. 
**Poly-Time Computable Functions**

**Definition:** A function

\[ f : \Sigma^* \rightarrow \Sigma^* \]

is polynomial-time computable if there is a poly-time deterministic TM that

- starts with input \( w \), and
- halts with \( f(w) \) on tape.
Poly-Time Reducibility

**Definition:** We say that a language $A$ is **polynomial time mapping reducible** to $B$, written

$$A \leq_p B,$$

if there is a poly-time computable function

$$f : \Sigma^* \rightarrow \Sigma^*$$

such that, for every $w$,

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

The function $f$ is called a **polynomial-time reduction** from $A$ to $B$. 
Poly-Time Reductions

Convert questions about membership in $A$ to membership in $B$, and do it **efficiently**.
Poly-Time Reductions

Theorem: If $A \leq_P B$ and $B \in P$ then $A \in P$.

Proof: Let

- $f$ the reduction from $A$ to $B$, computed by TM $M_f$.
- On input $x$ of length $n$, $M_f$ takes at most $c_1 n^{a_1}$ steps.
- $M$ be the poly-time decider for $B$.
- On input $y$ of length $m$, $M$ takes at most $c_2 m^{a_2}$ steps.
Poly-Time Reductions

Define $\mathcal{N}$: on input $x$

1. compute $f(x)$
2. run $\mathcal{M}$ on input $f(x)$ and output whatever $\mathcal{M}$ outputs.

Analysis:

- On input $x$ of length $n$, computing $y = f(x)$ takes at most $c_1 n^{a_1}$ steps.
- On input $y$ of length $m = c_1 n^{a_1}$, $\mathcal{M}$ takes at most $c_2 m^{a_2} = c_2 (c_1 n^{a_1})^{a_2} = (c_2 c_1^{a_2}) n^{a_1 a_2}$ steps.
- Summing both stages, we got a polynomial in $n$.
- Correctness is clear, so $A \in P$.  

Slides modified by Benny Chor, based on original slides by Maurice Herlihy, Brown University.
Satisfiability

A boolean variable assumes values

- true (written 1), and false (written 0).

Boolean operations:

- and: $\wedge$
- or: $\vee$
- not: $\neg$

Examples:

\[
0 \wedge 1 = 0 \\
0 \vee 1 = 1 \\
\overline{0} = 1
\]
Satisfiability

A boolean formula is an expression involving boolean variables and operations.

\[ \phi = (\overline{x} \land y) \lor (x \land \overline{z}) \]

**Definition:** A formula is *satisfiable* if some assignment of 0s and 1s to the variables makes the formula evaluate to 1.
Satisfiability

\[ \phi = (\overline{x} \land y) \lor (x \land \overline{z}) \]

is satisfiable by

\[ \begin{align*}
    x &= 0 \\
    y &= 1 \\
    z &= 0
\end{align*} \]

This assignment satisfies \( \phi \).
Define

\[
\text{SAT} = \{ \langle \phi \rangle \mid \phi \text{ is satisfiable Boolean formula} \}
\]
Satisfiability

It is useful to consider a special version:

- A **literal** is a variable or negated variable: \( x \) or \( \overline{x} \).
- A **clause** is several literals joined by \( \lor \)s: \( (x_1 \lor \overline{x}_2 \lor \overline{x}_3) \)
- A Boolean formula is in **conjunctive normal form** (CNF) if it consists of **clauses**, connected with \( \land \)s.
- For example

\[
(x_1 \lor \overline{x}_2 \lor \overline{x}_3 \lor x_4) \land (x_3 \lor \overline{x}_5 \lor x_6) \land (x_3 \lor \overline{x}_6)
\]
Satisfiability

**Definition:** A Boolean formula is in 3CNF form if it is a CNF formula, and all clauses have three literals.

\[(x_1 \lor \overline{x_2} \lor \overline{x_3}) \land (x_3 \lor \overline{x_5} \lor x_6) \land (x_3 \lor \overline{x_6} \lor x_4)\]

Define

\[3SAT = \{\langle \phi \rangle \mid \phi \text{ is satisfiable 3CNF formula}\}\]

Clearly, if \(\phi\) is a satisfiable 3CNF formula, then for any satisfying assignment of \(\phi\), every clause must contain at least one literal assigned 1.
Claim: There is a poly time reduction from 3SAT to CLIQUE. In other words,

$$\text{3SAT} \leq_P \text{CLIQUE}.$$ 

We’ll construct a poly time reduction $f$ that maps 3CNF formulae $\phi$ to graphs and numbers, $\langle G, k \rangle$.

The function $f$ will have the property that $\phi$ is satisfiable if and only if $G$ has a clique of size $k$. 
Reminder: A **clique** in a graph is a subgraph where every two nodes are connected by an edge. A **$k$-clique** is a clique of size $k$. For example, the graph above has a 5-clique.
3SAT \leq_P CLIQUE

Let \( \phi \) be a 3CNF formula with \( k \) clauses.

\[
(x_1 \lor \overline{x_2} \lor \overline{x_3}) \land (x_3 \lor \overline{x_5} \lor x_6) \land (x_3 \lor \overline{x_6} \lor x_4)
\]

We define a graph \( G \) as follows:
3SAT $\leq_P$ CLIQUE

We define a graph $G$ as follows:

- nodes in $G$ are organized into triples $t_1, \ldots, t_k$.
- each triple corresponds to a clause of $\phi$.
- each node in a triple corresponds to a literal.
$3\text{SAT} \leq_P \text{CLIQUE}$

$\left( x_1 \lor \overline{x_2} \lor x_3 \right) \land \left( \overline{x_3} \lor x_5 \lor x_6 \right) \land \left( \overline{x_3} \lor \overline{x_4} \lor x_6 \right)$
3SAT vs. CLIQUE

\[(x_1 \lor \overline{x_2} \lor \overline{x_3}) \land (\overline{x_3} \lor \overline{x_5} \lor x_6) \land (x_3 \lor x_4 \lor \overline{x_6})\]

Add edges between all vertex pairs, except
- within same triple
- between contradictory literals
Claim: If $\phi$ is satisfiable, $G$ has a $k$-clique.

Suppose $\phi$ is satisfiable.

- at least one literal is true in every clause
- in every tuple, select one true literal
- they can be joined by edges
- yielding a $k$-clique
Claim: If $\phi$ is satisfiable, $G$ has a $k$-clique.

\[ (x_1 \lor x_2 \lor x_3) \land (x_3 \lor x_5 \lor x_6) \land (x_3 \lor x_4 \lor x_6) \]
3SAT $\leq_P$ CLIQUE

Claim: If $G$ has a $k$-clique, $\phi$ is satisfiable.

- No two of the cliques nodes are in the same triple.
- Have $k$ vertexes and $k$ clauses, so each triple has exactly one clique node.
- Assign 1 to each node in clique
- no contradictions.
3SAT $\leq_P$ CLIQUE

- We’ve constructed a poly time computable function $f$.
- We saw that the function $f$ has the property that $\phi \in 3\text{SAT}$ if and only if $f(\phi) \in \text{CLIQUE}$.
- Therefore $f$ is a reduction from 3SAT to CLIQUE, so $3\text{SAT} \leq_P \text{CLIQUE}$. 

♣️
An independent in a graph is a set of vertexes, no two of which are linked by an edge.

The independent set problem asks whether there exists an independent set of size $k$. 
Independent Set

Define

\[ \text{INDEPENDENT-SET} = \{ \langle G, k \rangle | G \text{ contains an independent set of size } k \} \]

Claim: \text{INDEPENDENT-SET} is polynomial time reducible to \text{CLIQUE},

\[ \text{INDEPENDENT-SET} \leq_P \text{CLIQUE} \]

and vice-versa,

\[ \text{CLIQUE} \leq_P \text{INDEPENDENT-SET} \]
**Independent Set**

**Definition:** The complement of a graph $G = (V, E)$ is a graph $G^c = (V, E^c)$, where

$E^c = \{(v_1, v_2) | v_1, v_2 \in V \text{ and } (v_1, v_2) \notin E\}$.

**Claim:** If $V$ is an independent set in $G$, then $V$ is a clique in $G^c$.

’nuff said.
Independent Set
A Hamiltonian path in a directed $G$ visits each note once.
Hamiltonian Path

\[ \text{HAMPATH} = \{ \langle G, s, t \rangle | G \text{ has Hamiltonian path from } s \text{ to } t \} \]
Hamiltonian Circuit

visits each note once.
ends up where it started
Hamiltonian Circuit

\[ \text{HAMCIRCUIT} = \{ \langle G \rangle \mid G \text{ has Hamiltonian circuit} \} \]

**Theorem:** HAMPATH is polynomial-time reducible to HAMCIRCUIT,

\[ \text{HAMPATH} \leq_P \text{HAMCIRCUIT} \]
Theorem: HAMPATH is polynomial-time reducible to HAMCIRCUIT.
Reduction

**Theorem:** HAMCIRCUIT is polynomial-time reducible to HAMPATH.

**Proof:** Left as an easy (recommended) exercise.
Definition

A language $B$ is **NP-complete** if it satisfies

- $B \in NP$, and
- Every $A$ in NP is polynomial time reducible to $B$
A language $B$ is **RE-complete** if it satisfies

- $B \in \text{RE}$, and
- Every $A$ in RE is mapping reducible to $B$
Theorem

Theorem: If $B$ is NP-complete and $B \in P$, then $P = NP$.

To show $P = NP$ (and make an instant fortune, see [www.claymath.org/millennium/P_vs_NP/](http://www.claymath.org/millennium/P_vs_NP/)), suffices to find a polynomial-time algorithm for some NP-complete problem.
Theorem

**Theorem:** If $B$ is NP-complete, $C \in NP$, and $B \leq_P C$, then $C$ is NP-complete.

- We know that $C \in NP$,
- must show that every $A$ in NP is poly-time reducible to $C$.
- Because $B$ is NP-complete,
- every language in NP is poly-time reducible to $B$.
- $B$ is poly-time reducible to $C$
- Can compose poly-time reductions (why?), so
- $A$ is poly-time reducible to $C$.  ♣
Once we have one “structured” NP-complete problem, we can generate more by poly-time reduction.

Getting the first one requires some work.

This is what Steve Cook (then in Berkeley, now in Toronto) and Leonid Levin (then in Moscow, now in Boston) did in the early seventies.
Traveling Salesman

Parameters:
- set of cities $C$
- set of inter-city distances $D$
- goal $k$

Slides modified by Benny Chor, based on original slides by Maurice Herlihy, Brown University.
Define \( \text{TRAVELING-SALESMAN} = \{ \langle C, D, k \rangle \mid (C, D) \text{ has a TS tour of total distance } \leq k \} \)

Remark: Can consider two versions – undirected and directed.

Recall
\( \text{HAMCIRCUIT} = \{ \langle G \rangle \mid G \text{ has Hamiltonian circuit} \} \)

Theorem: \( \text{HAMCIRCUIT} \) is polynomial-time reducible to \( \text{TRAVELING-SALESMAN} \),
\[ \text{HAMCIRCUIT} \leq_P \text{TRAVELING-SALESMAN} \]
The reduction: Given a directed graph $G = (V, E)$ we construct a directed traveling salesman instance.

- The cities are identical to the nodes of the original graph, $C = V$.
- The distance of going from $v_1$ to $v_2$ is 1 if $(v_1, v_2) \in E$, and 2 otherwise.
- The bound on the total distance of a tour is $k = |V|$.
HAMCIRCUIT $\leq_P$ TSP

Validity of Reduction

$\Rightarrow$ Suppose $G$ has a Hamiltonian circuit. The distance assigned by the reduction to all edges in this circuit is 1. Thus in $(C, D)$ there is a traveling salesman tour of total distance $|V| = k$, namely $(C, D, k) \in$ TRAVELING-SALESMAN.

$\Leftarrow$ Suppose $(C, D)$ has a traveling salesman tour of total distance $|V| = k$. Tour cannot contain any edge of distance 2. Therefore it gives a Hamiltonian circuit in $G$.

Efficiency: Reduction in quadratic time (filling up distances for all edges of the complete graph). ♣
\textbf{3SAT (reminder)}

\textbf{Definition:} A Boolean formula is in \textit{3CNF form} if it is a \textit{CNF} formula, and all terms have \textbf{three literals}.

\[(x_1 \lor \overline{x_2} \lor \overline{x_3}) \land (x_3 \lor \overline{x_5} \lor x_6) \land (x_3 \lor \overline{x_6} \lor x_4)\]

Define

\[3\text{SAT} = \{\langle \phi \rangle \mid \phi \text{ is satisfiable 3CNF formula}\}\]

Clearly, if \(\phi\) is a satisfiable 3CNF formula, then for any satisfying assignment of \(\phi\), every clause must contain at least one literal assigned 1.
The Language SAT

**Definition:** A Boolean formula is in conjunctive normal form (CNF) if it consists of terms, connected with $\land$s.

For example $(x_1 \lor \overline{x}_2 \lor \overline{x}_3 \lor x_4) \land (x_3 \lor \overline{x}_5 \lor x_6) \land (x_3 \lor \overline{x}_6)$

**Definition:** \( \text{SAT} = \{ \langle \phi \rangle \mid \phi \text{ is satisfiable CNF formula} \} \)
Strategy

- Once we have one **structured** NP-complete problem, we can generate more by **poly-time reductions**.
- Getting the first one requires some work.
Cook-Levin (early 70s)

**Theorem:** SAT is NP complete.

- Must show that every NP problem reduces to SAT in poly-time.
- **Proof Idea:** Suppose $\mathcal{L} \in \mathcal{NP}$, and $M$ is an NTM that accepts $\mathcal{L}$.
- On input $w$ of length $n$, $M$ runs in time $t(n) = n^c$.
- We consider the $n^c$-by-$n^c$ tableau that describes the computation of $M$ on input $w$. 
The Tableau

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>...</th>
<th>t(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₀</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

*cell[1,1] *

*cell[1,t(n)]*
The Tableau

- Row 1 in tableau represents **initial configuration** of $M$ on input $w$.

- Row $i$ in tableau represents **$i$-th configuration** in a computation of $M$ on input $w$. 
A Formula Simulating the Tableau

- We construct a Boolean CNF formula $\phi_w$ that “mimics” the tableau.
- Given the string $w$, it takes $O(n^{2c})$ steps to construct $\phi_w$.
- The following property holds:
  $$\phi_w \in SAT \text{ iff } M \text{ accepts } w.$$  
- So the mapping $w \mapsto \phi_w$ is a poly time reduction from $\mathcal{L}$ to $SAT$, establishing $\mathcal{L} \leq_p SAT$.

- We still got a few small details to take care of...
Details of Formula (Partial List)

- We construct a Boolean CNF formula $\phi_w$ that “mimics” the tableau:
  - $\phi_w$ uses Boolean variables of three types.
    - $b_{i,j,\sigma}$ is true iff the $j$-th cell in $i$-th configuration contains the letter $\sigma \in \Gamma$.
    - $s_{i,q}$ is true iff in $i$-th configuration, $M$ is in state $q \in Q$.
    - $h_{i,j}$ is true iff in $i$-th configuration $M$, has is head in cell $j$ on tape.
- The formula $\phi_w$ consists of four parts:
  \[ \phi_w = \phi_{\text{unique}(M)} \land \phi_{\text{start}(w)} \land \phi_{\text{accept}(M)} \land \phi_{\text{compute}(M)} \]
Details of Formula (cont.)

- $\phi_{\text{unique}}(M)$ guarantees that the variables encode legal configurations. For example, at most one of $b_{i,j,0}$ and $b_{i,j,1}$ is true.

- $\phi_{\text{start}}(w)$ guarantees that the variables corresponding to the first row ($i = 1$) encode the initial configuration of $M$ on $w$.

- $\phi_{\text{accept}}(M)$ guarantees that $M$ reached an accepting configuration.

- $\phi_{\text{compute}}(M)$ guarantees that the configuration described by the $i+1$-st row is a legal succession of the configuration described by the $i$-th row.
Details of Formula (cont.)

- $\phi_{\text{compute}}(M)$ is the “heart” of $\phi_w$. To construct it, employ locality of computations.

- To determine contents of tableau entry $(i, j)$ (cell $j$ in configuration $i$), only the contents of three tableau entries (from configuration $i - 1$), $(i - 1, j - 1)$, $(i - 1, j)$, $(i - 1, j + 1)$, and $M$’s table, are needed.

- If head not in area, nothing changes. And and if it is, changes are local and determined using $M$. 
The Tableau in Perspective

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>...</th>
<th>t(n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₀</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>⋯</td>
</tr>
</tbody>
</table>

- cell[1,1]
- cell[1,t(n)]
Correctness of Reduction

- All four components of $\phi_w$ can be put in CNF, so $\phi_w$ itself ($\land$ of the four) is also in CNF.
- The transformation $w \mapsto \phi_w$ is computable in time $O(n^{2c})$.
- An assignment satisfying $\phi_{\text{unique}}(M) \land \phi_{\text{start}}(w) \land \phi_{\text{compute}}(M)$ corresponds to a valid computation of $M$ on $w$.
- An assignment satisfying, in addition $\phi_{\text{accept}}(M)$, corresponds to an accepting computation of $M$ on $w$.
- Therefore $M$ accepts $w$ iff $\phi_w \in SAT$.

For complete details, consult Sipser or take the Complexity course.
Strategy

- We have seen that SAT is NP-complete.
- We now reduce SAT to 3SAT.
- And then will reduce 3SAT to a bunch of other problems in NP.
- In class and recitation will give in detail just a few examples.
- Full list contains hundreds or thousands of known NP-complete problems (from combinatorics, operation research, VLSI design, computational geometry, bioinformatics, ...).
- NP-completeness of new and of old problems is still established these days.
SAT and 3SAT

Recall

\[ \text{SAT} = \{ \langle \phi \rangle \mid \phi \text{ is a satisfiable CNF formula} \} \]

\[ \text{3SAT} = \{ \langle \phi \rangle \mid \phi \text{ is satisfiable 3CNF formula} \} \]

The reduction maps CNF formulae to 3CNF ones “clause by clause”. A clause with \( \ell \) literals is mapped to \( \ell \) clauses, built on the original literals together with \( \ell - 1 \) new ones.

For example:

\[
(x_1 \lor \overline{x}_2 \lor \overline{x}_3 \lor x_4 \lor x_8) \rightarrow
(x_1 \lor \overline{y}_1) \land (\overline{y}_1 \lor \overline{x}_2 \lor y_2) \land (\overline{y}_2 \lor \overline{x}_3 \lor y_3) \land
(\overline{y}_3 \lor x_4 \lor y_4) \land (\overline{y}_4 \lor x_8)
\]
Consider mapping $\phi \mapsto \phi_3$, e.g. $(x_1 \lor \overline{x_2} \lor \overline{x_3} \lor x_4 \lor x_8) \mapsto (x_1 \lor y_1) \land (\overline{y_1} \lor \overline{x_2} \lor y_2) \land (\overline{y_2} \lor \overline{x_3} \lor y_3) \land (\overline{y_3} \lor x_4 \lor y_4) \land (\overline{y_4} \lor x_8)$

Claim: $\phi$ has a satisfying assignment iff $\phi_3$ does.

Proof sketch: $\iff$ An assignment satisfying $\phi_3$ cannot “rely” on new literals alone – at least one original literal must be satisfied.

$\iff$ An assignment satisfying $\phi$ makes at least one literal per clause happy. In the “$\phi_3$ clause” of this literal the new variable is under no constraints. This enables propagation to a satisfying assignment that “relies” on new vars alone in rest of $\phi_3$ clauses.

This establishes validity of the reduction. Since it is in polynomial time (why?), we get $\text{SAT} \leq_P \text{3SAT}$. ♣.
3SAT – Cousins and Cambrians

We now know that \( \text{SAT} \leq_P \text{3SAT} \). Since \( \text{SAT} \) is NP-complete and \( \text{3SAT} \in \text{NP} \), this proves that \( \text{3SAT} \) is itself NP-complete.

What about the \( \text{3SAT} \leq_P \text{SAT} \) direction?

We now want to examine what happens if we further reduce the number of literals per clause in CNF formulae.

**Definition:** A Boolean formula is in 2CNF if it is a CNF formula, and all terms have at most two literals. For example

\[
(\overline{x_1} \lor x_2) \land (\overline{x_5} \lor x_6) \land (\overline{x_6} \lor \overline{x_4})
\]
3SAT – Cousins and Cambrians

Definition:

\[ 2\text{SAT} = \{\langle \phi \rangle \mid \phi \text{ is satisfiable 2CNF formula}\} \]

Betting time: Is 2SAT NP-complete? Is it in P? Or maybe we do not know? …

Well, turns out 2SAT is in P. For details, though, you’ll have to refer to the algorithms, ahhhm, efficiency of computations, course.
Chains of Reductions: NPC Problems

- SAT
  - IntegerProg
  - 3SAT
    - Clique
    - 3Color
    - HamPath
      - IndepSet
      - Scheduling
        - VertexCover
          - SetCover
            - 3ExactCover
              - Knapsack
            - TRAVELING-SALESMAN
        - HamCircuit