Can computers replace mathematicians?

\[ L = \{ (x,k) \mid \text{statement } x \text{ has a proof of length } \leq k \} \]

A yes answer to this question is among the many profound implications that would result if \( P = \text{NP} \).

The class \( \text{NP} \) is an abbreviation of non-deterministic Polynomial time.

The class \( \text{NP} \) will initially be defined in terms of non-deterministic models of computation, but we shall see that it is equivalent to the witness version that we have seen earlier.

This lecture will cover:

- Non-deterministic models of computation
- Non-deterministic time classes
- \( \text{NP} \), \( \text{NP} \)-completeness
- \( \text{co-NP} \)
- \( \text{NTIME} \) Hierarchy
- Ladner's theorem

Recall the definition of non-deterministic Turing Machines:

\[ \delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{ L, R, \}_L, R, \}

In a non-deterministic TM we have \( \Delta \) instead of \( \delta \):

\[ \Delta \subseteq (Q \times \Sigma) \times (Q \times \Sigma \times \{ L, R, \}_L, R, \} \]
Given the current state and symbol, there may be more than one (or no) choices for what to do next.

In deterministic computation: given a config of the TM, there is a unique next configuration.
In a non-deterministic computation: there may be several possibilities for the next configuration.

For a given TM \( M \) on input \( x \), we can build a configuration graph (nodes = TM configurations, \((q_1, c_1), (q_2, c_2) \in E \) if \( c_2 \) is reachable from \( c_1 \) in one step).

Deterministic TM

\[ q_0 \xrightarrow{x_1} q_1 \xrightarrow{x_2} \ldots \xrightarrow{x_n} q_f \]

Non-deterministic TM

\[ q_0 \xrightarrow{x_1 \ldots x_n} \]

\[ q_0 \xrightarrow{x_1} q_1 \xrightarrow{x_2} \ldots \]

In order for an NTM to accept or reject \( x \):

- All computation paths must terminate.
- Running time: length of the longest root to leaf path.
- Space: max # tape cells used on any path from the root to a leaf.

Accept if any leaf is an Accept.
Reject if all leaves are Reject.
\[ \text{NTIME}(f(n)) : \text{languages decidable by a multi-tape NTM that runs in at most } f(n) \text{ steps along any computation path where } n \text{ is the length of the input.} \]

\[ \text{NSPACE}(f(n)) : \text{languages decidable by a multi-tape NTM that touches } f(n) \text{ cells of work tape along any computation path.} \]

Time classes:
\[ \text{NP} = \bigcup_{k \geq 1} \text{NTIME}(n^k) \]
\[ \text{NEXP} = \bigcup_{k \geq 1} \text{NTIME}(2^{n^k}) \]

Useful alternative view of NP:

Is there a way to prove that \( x \in L \) with a poly-sized proof \( y \) that can be verified by a poly-time verifier?

\[ L \in \text{NP} \iff \exists \text{ polytime verifier } R \text{ and constant } k \text{ s.t. } \]
\[ x \in L \iff \exists y \mid y \mid \leq |x|^k \text{ and } R \text{ accepts } (x, y) \]

\[ L = \exists x \mid \exists y \mid |y| \leq |x|^k (x, y) \in L(R) \]

Examples:
\[ 3\text{SAT} = \exists \phi \mid \phi \text{ is a 3-CNF formula for which } \exists \text{ assignment } \alpha \]
\[ (\phi, \alpha) \in L(R) \]
\[ L(R) = \exists (\phi, \alpha) \mid \alpha \text{ is a satisfying assignment for } \phi \]
A is a witness for $\phi \in 3\text{SAT}$

$R$ is a poly-time TM.

Other examples: Hamiltonian Path, etc.

Why are these two definitions the same?

**Theorem:** $L \in \text{NP}$ iff it is expressible as:

$L = \exists x \exists y \ y \leq x \ (x, y) \in R$

for some poly-time TM $R$.

**Proof:** $\Leftarrow$ Show NTM that decides $L$.

1. Compute $|x|^k$ by marking off $|x|^k$ symbols on a tape.
2. "Guess" a string $y$ of length $|x|^k$ and write it on a tape.
3. Run $R$ on $(x, y)$ and accept if $R$ accepts.
   (Reject if $R$ rejects)

What does 2 look like?

\[ \Delta(\#, \#_{\text{guess}}) = \frac{1}{2} (0, \#_{\text{guess}}, R), (1, \#_{\text{guess}}, R) \]
\[ \text{\( \exists \) NTM } M \text{ that decides } L \text{ in time } \mathcal{O} \]

Consider a string \( y \) consisting of the non-deterministic choices at each step.

\[ y = \left[ \left( q, a, L \right), \left( q', b, R \right), \left( \bar{q}, a, - \right), \ldots \right] \]

- \( \text{ triples} \)

\[ R(x, y) : \text{ simulates } M \text{ on } x \]

- \( \text{ checks if } \left[ \left( q, a \right) x (q', b, L) \right] \in \Delta M \)

- \( \text{ If so, execute step and continue.} \)

- \( \text{ If not, reject.} \)

If \( M \) halts then \( R \) halts (\text{acc/rej depending on } M) \)

An accepting path of computation of \( M \) will correspond to some \( y \) which causes \( R \) to accept \((x, y)\)

If all paths in \( M \) reject, there will be no \( y \) that can cause \( R \) to accept.

\[ \text{Why NP? There are a huge number of problems that are complete for NP.} \]

\[ \text{Why not EXP? Too strong important problems are not complete for EXP.} \]
Central question in computer science: \[ P =?= NP \]

Finding a solution vs. Verifying a solution.

**NP-completeness:**

**Circuit SAT:** Given a boolean circuit with variables, is there an assignment to the variables that makes it output 1?

**Theorem:** Circuit-SAT is NP-complete.

**Circuit SAT \in NP:** guess assignment for the variables and check if circuit is satisfied.

If \( L \in NP \), \( L \times Circuit SAT \)

\[ \exists k \in \text{Poly-time} R: \]

\[ L = \exists x \ | \ \exists y \ | \ \| x \| \leq k \ | \ R(x, y) \text{ accept} \]

Recall the tableau from last lecture corresponding to the computation of \( R \):

```
1
|x^k
1
```

First row of tableau:

\[ x_1 \ldots x_n y_1 \ldots y_m \]

\( m = |x^k| \)

0/1 inputs, variable inputs to circuit

As discussed before, this can be made into a circuit.
Reduction: given \( x \) output \( E_x \).

Only difference is \( P \)-completeness proof are the input variables for \( y_1, \ldots, y_m \). //

Witness version for \( \text{NEXP} \): (or "proof system" version)

\[ L \in \text{NEXP} \text{ if and only if } \exists k \in \text{polytime} \ R \text{ s.t.} \]

\[ \exists x \mid \exists y, |y| = 2^{|x|^k} \Rightarrow R \text{ accepts } (x, y) \]

\[ R \text{ is poly time in } |x|, |y| \]

Since \(|y|\) is already exp long in \(|x|\).

\[ \text{Pf of theorem similar to the proof for } \text{NP}. \]

\[ \text{SUCCINCT Ckt SAT} \]

\begin{itemize}
  \item Succinctly encoded Boolean circuit with \( n \) non-variable inputs and \( m \) variable inputs
  \item Is there a setting of the variables that causes the circuit to evaluate to 1?
\end{itemize}

\[ \text{Theorem} \quad \text{SUCCINCT Ckt SAT is } \text{NEXP-complete}. \]

\[ \text{Pf uses same ideas as the proof for } \text{SUCCINCT-Ckt-EVAL} \text{ is } \text{EXP-complete}. \]

\[ \text{The tableau is a record of the verifier's (R's) computation on input } x = x_1 \ldots x_n \text{ with variables } y_1, \ldots, y_m \text{. In this case } m = 2^k \text{ and the size of the circuit is poly } (n, m). \]
Complement Classes

If $C$ is a complexity class then $\text{co-} C$ is the class containing complements of languages in $C$.

\[
L \in C \implies \text{co-} L \in \text{co-} C \\
\text{co-} L \in C \implies L \in C
\]

Note that $\text{co-} L$ is not quite $\Sigma^* - L$ because invalid encodings are always excluded from the language.

\[
L = \text{graphs w/ a Hamiltonian cycle} \\
\text{co-} L = \text{graphs w/ no Hamiltonian cycle}
\]

(both languages exclude strings which are not valid encoodings of graphs).

Some classes are closed under complements:

$\text{co-} P = P$.

What about $\text{co-} NP$? We can't just exchange $\text{ace}$ and $\text{gre}$.

$\text{NP: } x \in L \\
\triangleleft \text{ace} \\
x \notin L \\
\triangleleft \text{gre}$

$\text{NP-} \text{ languages with short proofs}$

$\text{co-} \text{NP} \text{ unlikely to have short proofs}$

(all $\phi$ that are unsatisfiable).

$P \equiv NP$ implies $NP = \text{co-} NP$

Believed that

$NP \neq \text{co-} NP$. 
Non-deterministic Time Hierarchy Theorem:

If \( f + g \) are time constructible (proper) and \( f(n) \) is \( o(g(n)) \) then \( \text{NTIME}(f(n)) \neq \text{NTIME}(g(n)) \)

We will only show: \( \text{NTIME}(t(n)) \neq \text{NTIME}(t(n)^{11}) \)

We will assume the existence of an NTM called NSIM. On input \( <M, x> \) (\( M \) is non-deterministic) NSIM simulates \( M \) on \( x \). If \( M \) runs in time \( t(n) \) then NSIM runs in time \( t(n)^{11} \)

We can't just naively diagonalize. It's not clear how to flip the output of an NTM (because of the asymmetry in the acceptance criteria).

We will use a technique called "lazy diagonalization".

We'll use a technique called "lazy diagonalization."

Note that when we build this tableau:

\[
\begin{array}{cccc}
 & \rightarrow \ & \rightarrow \ & s. t. \ \text{flip the diagonal} \\
M & \downarrow & \downarrow & \text{we just need to find a language} \\
\hline
\end{array}
\]

Each row in the table corresponds to a string. Some strings will not encode a TM at all. Other strings will encode a TM that does not halt in time \( t(n) \). Some will encode \( t(n) \) time NTM's.

We will construct a language \( L \) computed by \( \text{NTM} \ D \) in time \( f(n) \), s.t. for every \( \text{NTM} \ M \) that runs in time \( t(n) \), \( \exists \ z \) s.t.

\[
D(z) \neq M(z) \quad (z's \ don't \ have \ to \ be \ consecutive).\]
\[ M_i = \text{ith string in lexicographic order of all strings, interpreted as a TM.} \]

\[ \text{Define } f(i) = 2 \]
\[ f(i+1) = \left\lfloor \frac{f(i)+1}{2} \right\rfloor + 1 \]

\[ \begin{array}{cccc}
1 & 1 & 1 & 1 \\
M_2 & M_1 & f(i+2) & f(i+3) \\
M & j^{f(i)} & f(i) & f(i+1) \\
\hline
1 & 1 & 1 & 1 \\
\end{array} \]

If \( M_i \) is an NTM which runs in time \( t(n) \), then the language \( L \) (i.e., \( D(L) \)) will differ from \( L(M_i) \) somewhere in the range \( 1^{f(i)+1}, 1^{f(i)+2}, \ldots, 1^{f(i+1)} \).

Here's what \( D \) does on input \( x \):

- If \( x \neq 1^n \) then reject.
- If \( x = 1^n \), compute \( i \) s.t. \( f(i) < n \leq f(i+1) \).
  - If \( i \) is not a valid encoding of a TM = reject.
  - Otherwise output whatever \( M_i \) does. Takes time \( t(n+1) \).

If \( f(i) < n < f(i+1) \), use NSIM to simulate \( M_i \) on \( 1^{n+1} \) for \( t(n+1) \) steps. If \( M_i \) does not finish = accept. Otherwise output whatever \( M_i \) does. Takes time \( t(n+1) \).

If \( M_i \) runs in time \( t(i) \) then \( 1^n \in L(D) \iff 1^{n+1} \in L(M_i) \).

If \( n = f(i+1) \) (brute force)

D deterministically simulates \( M_i \) on input \( 1^{f(i)+1} \).

\( D \) accepts on input \( 1^{f(i)+1} \) if \( M_i \) rejects \( 1^{f(i)+1} \).
This takes $2^{t(f(i)+1)}$ steps of $N_i$. With simulation overhead $\left\lceil \frac{t(f(i)+1)}{2} \right\rceil \leq 2^{t(f(i)+1)} \leq f(i+1) = n$ line-time!

There must be some $1^m$ in this range on which they disagree.

//