Chapter 2: The Theory of NP-completeness

- Problems vs. Languages
- Encoding schemes
- P, NP
Hey, I just proved the MNR problem NP-complete! Now I can get my Ph.D!
Encoding Schemes, cont.

Problems aren’t NP-complete, you idiot, languages are!
Yea, well, technically you’re right but I encoded MNR as the language L[MNR,e] and then proved that NP-complete.
Uhh...well, while it may be true that the language $L[MNR,e]$ is NP-complete, that doesn't mean the MNR problem is.
Actually, yes it does. That's what encoding schemes are all about, you fool!
The fact that $L[MNR,e]$ is NP-complete, allows me to *informally* say the same thing about the MNR *problem*, and there is no loss of generality.
Knowing you, I bet you picked the encoding scheme $e$ so that it makes the MNR problem appear more difficult than it actually is. You will not be able to fool your committee this time!
Wrong again!
My encoding scheme is perfectly reasonable!
Fine, whatever! But one NP-completeness result isn’t going to get you a Ph.D. anyway.
As computer scientists, we are typically concerned with solving problems, not recognizing languages (unless your last name is Stansifer).

However, ultimately we want to make precise claims about problems:

- $\Pi \in P$, $\Pi \in NP$ or $\Pi \in NP$-complete
- 2-SAT can be solved by a deterministic polynomial time algorithm
- TSP can be solved by a non-deterministic polynomial time algorithm
Such claims raise several questions:
- What is an algorithm?
- What do deterministic and non-deterministic mean?
- What does it mean to “solve” a problem?”
- What are P, NP, NP-complete?
- How is time measured, and what is input length?

In order to answer such questions precisely, we need to precisely define our terms, i.e., we will use a formal model of computation, i.e., the Turing Machine.
Symbol – An atomic unit, such as a digit, character, lower-case letter, etc.

Alphabet – A finite set of symbols, usually denoted by $\Sigma$.

\[ \Sigma = \{0, 1\} \quad \Sigma = \{0, a, 4\} \quad \Sigma = \{a, b, c, d\} \]

String – A finite length sequence of symbols, presumably from some alphabet.

\[ w = 0110 \quad y = 0aa \quad x = aabcaa \quad z = 111 \]

special string: $\varepsilon$
String operations:

\[ w = 0110 \quad y = 0aa \quad x = aabcaa \quad z = 111 \]

concatenation: \[ wz = 0110111 \]

length: \[ |w| = 4 \quad |\epsilon| = 0 \quad |x| = 6 \]

reversal: \[ y^R = aa0 \]
Some special sets of strings:

- \( \Sigma^* \): All strings of symbols from \( \Sigma \) (Kleene closure)
- \( \Sigma^+ \): \( \Sigma^* - \{\varepsilon\} \) (positive closure)

Example:

- \( \Sigma = \{0, 1\} \)
- \( \Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001,\ldots\} \)
- \( \Sigma^+ = \{0, 1, 00, 01, 10, 11, 000, 001,\ldots\} \)
A (formal) language is:

1) A (finite or infinite) set of strings from some alphabet.
2) Any subset $L$ of $\Sigma^*$
Language Examples:

\[ \Sigma = \{0, 1\} \]
\[ L = \{x \mid x \in \Sigma^* \text{ and } x \text{ contains an even number of } 0\text{'s}\} \]

\[ \Sigma = \{0, 1, 2, \ldots, 9, .\} \]
\[ L = \{x \mid x \in \Sigma^* \text{ and } x \text{ forms a finite length real number}\} = \{0, 1.5, 9.326, \ldots\} \]

\[ \Sigma = \{a, b, c, \ldots, z, A, B, \ldots, Z\} \]
\[ L = \{x \mid x \in \Sigma^* \text{ and } x \text{ is a Java reserved word}\} = \{\text{while, for, if,} \ldots\} \]
Language Examples:

\[ \Sigma = \{ \text{Java reserved words} \} \cup \{ (, ), ., :, ;, \ldots \} \cup \{ \text{Legal Java identifiers} \} \]
\[ L = \{ x \mid x \in \Sigma^* \text{ and } x \text{ is a syntactically correct Java program} \} \]

\[ \Sigma = \{ \text{English words} \} \]
\[ L = \{ x \mid x \in \Sigma^* \text{ and } x \text{ is a syntactically correct English sentence} \} \]

Some special languages:

\{\} The empty set/language, containing no strings
\{\epsilon\} A language containing one string, the empty string.
\Sigma^* and \Sigma+
An analogy between languages and problems:
- $L$ partitions $\Sigma^*$ into two sets: $L$ and $\Sigma^* - L$
- $Y_\Pi$ partitions $D_\Pi$ into two sets: $Y_\Pi$ and $D_\Pi - Y_\Pi$

Additionally:
- DTM & NDTM
- Deterministic & nondeterministic algorithms

How we use these analogies:
- The formal language level will be used to define various terms and concepts, i.e., algorithm, deterministic, non-deterministic, solve, problem, $P$, $NP$, $NP$-complete, etc.
- Encoding schemes will form the “bridge” between the formal and informal levels.
- This will allow us to use these terms in the context of problems.
What is an encoding scheme (informally)?

- Suppose you wrote a program that inputs a graph \( G = (V, E) \)
- You need to come up with an input (or file) format
- That format is (basically) an encoding scheme
Suppose we are dealing with a problem in which each instance is a graph:

\[ G = (V,E) \]
\[ V = \{V_1, V_2, V_3, V_4\} \]
\[ E = \{\{V_1, V_2\}, \{V_2, V_3\}\} \]

Encoding schemes:

- **vertex list & edge list**
  \[ V[1], V[2], V[3], V[4], (V[1], V[2]), (V[2], V[3]) \]
  length: 36

- **neighbor list**
  \[ (V[2])(V[1], V[3])(V[2])() \]
  length: 24

- **adjacency matrix**
  \[ 0100/1010/0010/0000 \]
  length: 19
Notes:
- A problem and an associated encoding scheme define a language
- A particular problem has many different encoding schemes
- Length is now precise - the number of input symbols

<table>
<thead>
<tr>
<th>Encoding Scheme</th>
<th>Lower Bound</th>
<th>Upper Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex list, Edge list</td>
<td>$4v + 10e$</td>
<td>$4v + 10e + (v + 2e) \cdot \lceil \log_{10} v \rceil$</td>
</tr>
<tr>
<td>Neighbor list</td>
<td>$2v + 8e$</td>
<td>$2v + 8e + 2e \cdot \lceil \log_{10} v \rceil$</td>
</tr>
<tr>
<td>Adjacency matrix</td>
<td>$v^2 + v - 1$</td>
<td>$v^2 + v - 1$</td>
</tr>
</tbody>
</table>

Figure 1.5 General bounds on input lengths for the three encoding schemes of Figure 1.4 for graphs $G = (V, E)$ with $|V| = v$, $|E| = e$. Since $e < v^2$, these show that the input lengths differ at most polynomially from each other. ($\lceil x \rceil$ denotes the least integer not less than $x$.)
Let $\Pi$ be a problem, $e$ an encoding scheme that uses symbols from $\Sigma$. Then $\Pi$ partitions $\Sigma^*$ into 3 subsets:

- $\{ x \mid x \in \Sigma^* \text{ and } x \notin D_\Pi \}$
- $\{ x \mid x \in \Sigma^*, x \in D_\Pi \text{ and } x \in Y_\Pi \}$
- $\{ x \mid x \in \Sigma^*, x \in D_\Pi \text{ and } x \notin Y_\Pi \} (x \in D_\Pi - Y_\Pi)$

The second of the above is the language associated with $\Pi$ and $e$:

$$L[\Pi, e] = \{ x \in \Sigma^* \mid \Sigma \text{ is the alphabet used by } e, \text{ and } x \text{ is the encoding under } e \text{ of an instance } I \in Y_\Pi \}$$

(page 20)
How does this allow application of formal terms, concepts and definitions at the problem level? (See page 20, second paragraph)

- Terms such as “NP-complete” are defined formally in terms of formal languages
- Encoding schemes allow us to apply these terms immediately to problems
- Saying that a problem \( \Pi \) is NP-complete is a somewhat informal way of saying that the language \( L[\Pi, e] \) is NP-complete, for some encoding scheme \( e \).
The previous page suggests our results for a problem are encoding dependent

But is this true?

- Could a property hold for a problem under one encoding scheme, but not another?
- Yes!
Example #1:

- Input a graph $G = (V, E)$ and for each pair of vertices output an indication of whether or not the two vertices are adjacent.
  - Adjacency matrix: $O(n)$ (n = input length)
  - Vertex-edge list: $O(n^2)$ (no edges)

Note that the encoding scheme does make a difference
Encoding Schemes, cont.

- Example #2:
  - Input a positive integer \( n \) and output all binary numbers between 0 and \( n-1 \)
  - Input \( n \) in binary: \( O(m2^m) \) (\( m = \) input length)
  - Input \( n \) in unary: \( O(\log(m)2^{\log(m)}) = O(m\log m) \)

- Note that the encoding scheme makes a dramatic difference in running time, i.e., polynomial vs. exponential

- The second encoding scheme is unnecessarily padding the input, and is therefore said to be *unreasonable*
Reasonable encoding schemes are:
- Concise (relative to how a computer stores things)
- Decodable (in polynomial time, pages 10, 21)

We do not want the fact that a problem is NP-complete or solvable in polynomial time to depend on a specific encoding scheme
- Hence, we restrict ourselves to reasonable encoding schemes
- Comes naturally, unreasonable encoding schemes will look suspicious
Hey, I just proved the MNR problem NP-complete! Now I can get my Ph.D!
Encoding Schemes, cont.

Problems aren't NP-complete, you idiot, languages are!
Yea, well, technically you’re right but I encoded MNR as the language $L_{[MNR,e]}$ and then proved that NP-complete.
Uhh...well, while it may be true that the language $L_{[MNR,e]}$ is NP-complete, that doesn't mean the MNR problem is.
Actually, yes it does. That's what encoding schemes are all about, you fool!
The fact that $L[MNR,e]$ is NP-complete, allows me to *informally* say the same thing about the MNR *problem*, and there is no loss of generality.
Knowing you, I bet you picked the encoding scheme $e$ so that it makes the MNR problem appear more difficult than it actually is. You will not be able to fool your committee this time!
Wrong again! My encoding scheme is perfectly reasonable!
Fine, whatever!
But one NP-completeness
Result isn't going to
Get you a Ph.D. anyway.
One last concept needs to be defined before moving on to Turing Machines…

The running time of an algorithm, or the computational complexity of a problem is typically expressed as a function of the length of a given instance.

- $O(n^2)$
- $\Omega (n^3)$

In order to talk about the computational complexity of a problem or the running time of an algorithm, we need to associate a length with each instance.

More specifically, for each NP-completeness proof, we must prove that it can be performed in time polynomial in the input length.
Every problem will have an associated length function:

\[
\text{LENGTH: } D_\Pi \rightarrow \mathbb{Z}^+
\]

which must be encoding-scheme independent, i.e., “polynomial related” to the input lengths we would obtain from a reasonable encoding scheme (see page 19 for a definition).

This ensures that the length function is reasonable and that complexity results for a problem are consistent with those for an associated encoded language.
Recall the CLIQUE problem:

**CLIQUE**

INSTANCE: A Graph $G = (V, E)$ and a positive integer $J \leq |V|$.  
QUESTION: Does $G$ contain a clique of size $J$ or more?

What would be a reasonable length function?

- # of vertices
- # of edges
- # of vertices + # of edges

Which would be polynomial related to a reasonable encoding scheme?

- Note that the first and last differ quite substantially, although not exponentially.
Deterministic, One-Tape TM

- Two-way, infinite tape, broken into cells, each containing one symbol.
- Two-way, read/write tape head.
- Finite control, i.e., a program, containing the position of the read head, current symbol being scanned, and the current state.
- An input string is placed on the tape, padded to the left and right infinitely with blanks, read/write head is positioned at the left end of input string.
- In one move, depending on the current state and the current symbol being scanned, the TM 1) changes state, 2) prints a symbol over the cell being scanned, and 3) moves its' tape head one cell left or right.
Deterministic, One-Tape TM

A DTM is a seven-tuple: $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

- $Q$: A finite set of states
- $\Gamma$: A finite tape alphabet
- $B$: A distinguished blank symbol, which is in $\Gamma$
- $\Sigma$: A finite input alphabet, which is a subset of $\Gamma - \{B\}$
- $q_0$: The initial/starting state, $q_0$ is in $Q$
- $q_n, q_y$: Halt states, which are in $Q$
- $\delta$: A next-move function, which is a mapping (i.e., may be undefined) from $(Q - \{q_n, q_y\}) \times \Gamma \rightarrow Q \times \Gamma \times \{-1,+1\}$

Intuitively, $\delta(q,s)$ specifies the next state, symbol to be written, and the direction of tape head movement by $M$ after reading symbol $s$ while in state $q$. 
Let M be a DTM.

M accepts $x \in \Sigma^*$ IFF M halts in $q_y$ when given $x$ as input.

The language recognized by M is:

$$L_M = \{ x \mid x \in \Sigma^* \text{ and M accepts } x \}$$

M is an algorithm IFF it always halts.
The time used by \( M \) on \( x \in \Sigma^* \) is the number of steps that occur up until a halt state is entered.

- In what sense does this represent “time?”

Suppose \( M \) always halts. Then the time complexity function \( T_M : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+ \) is denoted by:

\[
T_M(n) = \max \{ m : \text{there is an } x \in \Sigma^*, \text{ with } |x| = n, \text{ such that the computation of } M \text{ on input } x \text{ takes time } m \}
\]

\( M \) is a polynomial time DTM program if there exists a polynomial \( p \) such that:

\[
T_M(n) \leq p(n)
\]

for all \( n \in \mathbb{Z}^+ \)
Definition of $\text{P}$

- **Definition:**

  $$ \text{P} = \{ L : \text{There exists a polynomial-time DTM } M \text{ such that } L = L_M \} $$

- Finally, tying it all back into decision problems..

- $M$ solves a decision problem $\Pi$ under encoding scheme $e$ if $M$ *always halts* and $L_M = L[\Pi, e]$.

- If $L[\Pi, e] \in \text{P}$ then we say that $\Pi$ belongs to $\text{P}$ under $e$.
  - In other words, the problem $\Pi$ can be solved in polynomial time.
A NDTM is a seven-tuple: $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

- $Q$: A finite set of states
- $\Gamma$: A finite tape alphabet
- $B$: A distinguished blank symbol, which is in $\Gamma$
- $\Sigma$: A finite input alphabet, which is a subset of $\Gamma - \{B\}$
- $q_0$: The initial/starting state, $q_0$ is in $Q$
- $q_f$: Halt state, which is in $Q$
- $\delta$: A next-move function, which is a mapping (i.e., may be undefined) from $(Q - \{q_f\}) \times \Gamma \rightarrow$ finite subsets of $Q \times \Gamma \times \{-1,+1\}$

Intuitively, $\delta(q,s)$ specifies zero or more options, each of which includes the next state, symbol to be written, and the direction of tape head movement by $M$ after reading symbol $s$ while in state $q$. 
Let M be a NDTM.

M accepts \( x \in \Sigma^* \) IFF there exists some sequence of transitions that leads M to (halt in) \( q_y \).

The language recognized by M is:

\[
L_M = \{ x \mid x \in \Sigma^* \text{ and } M \text{ accepts } x \}
\]
A DTM is both a language defining device and an intuitive model of computation.
- A computer program can solve a problem in polynomial time IFF a DTM can.

A NDTM is not so much a good intuitive model of computation:
- How a NDTM “computes” is not even clear, at least in a real-world sense.

For our purposes, it is not as important to think of a NDTM as a “practical” computing device, so much as a language defining device.
The time used by M to accept $x \in L_M$ is:

$$\min \{ i \mid \text{M accepts } x \text{ in } i \text{ transitions} \}$$

Then the time complexity function $T_M : \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ is denoted by:

$$T_M(n) = \max \{ \{1\} \cup \{ m : \text{there is an } x \in \Sigma^*, \text{ with } |x| = n, \text{ such that the time to accept } x \text{ by M is } m \} \}$$

M is a polynomial time NDTM program if there exists a polynomial $p$ such that:

$$T_M(n) \leq p(n)$$

for all $n \geq 1$. 
NDTM Definitions

- Note that $T_M(n)$ only depends on accepting computations.
  - Some non-accepted strings may have very long (exponential time) computations.
  - In fact, if no inputs of a particular size $n$ are accepted, the complexity function is 1.
  - A polynomial-time NDTM could even have infinite loops on an input of size $n$, and this has no impact on the time complexity function.

- In contrast, $T_M(n)$ for a DTM depends on all computations.
  - Recall, we only consider DTMs that always halt.

- Why the difference?
  - Basically, for simplicity and convenience.
Finally, with the authors' model of a NDTM and time complexity, it is easy to:

- Describe their version of a NDTM
- Define the class NP of problems in a very inclusive, somewhat intuitive way
- Show problems are in NP
- Show that $P \subseteq NP$

In short, the author’s approach turns out to be a decent way to do things…
Definition of NP

Definition:

\[ NP = \{ L : \text{There exists a polynomial-time NDTM program } M \text{ such that } L = L_M \} \]

Finally, tying it all back into problems...

- M solves a decision problem \( \Pi \) under encoding scheme e if \( L_M = L[\Pi, e] \).

- If \( L[\Pi, e] \in NP \) then we say that \( \Pi \) belongs to NP under e.

What these definitions suggest is that to show a problem is in NP you must give a polynomial time NDTM for it (wow!)
Lemma: Every polynomial-time DTM is a polynomial-time NDTM.

Proof:
- Observe that every DTM is a NDTM (see the definitions).
- If a DTM runs in $T_M(n)$ deterministic time, then it runs in $T_{\text{ND}}(n)$ non-deterministic time, by definition (you can verify that the latter is no more than the former, i.e., $T_{\text{ND}}(n) \leq T_{\text{d}}(n)$).
- Therefore, if a DTM runs in deterministic polynomial time, then it runs in non-deterministic polynomial time.

Theorem: $P \subseteq NP$

Proof:
- Suppose $L \in P$
- By definition, there exists a polynomial-time DTM $M$ such that $L = L_M$
- By the lemma, $M$ is also a polynomial-time NDTM such that $L = L_M$
- Hence, $L \in NP$
Theorem: If \( L \in NP \) then there exists a polynomial \( p \) and a DTM \( M \) such that \( L = L_M \) and \( T_M(n) \leq O(2^{p(n)}) \).

Proof:

- Let \( M \) be a NDTM where \( L = L_M \)
- Let \( q(n) \) be a polynomial-time bound on \( M \)
- Let \( r \) be the maximum number of options provided by \( \delta \), i.e:

\[
 r = \max\{ |S| : q \in Q, s \in \Gamma, \text{ and } \delta(q,s) = S \}
\]

If \( x \in L_M \) and \( |x| = n \), then

- There is a "road map" of length \( \leq q(n) \) for getting to \( q_y \)
- There are \( r^{q(n)} \) road maps of length \( \leq q(n) \)
- A (3-tape) DTM \( M' \) generates and tries each road map
- \( M' \) operates in (approximately) \( O(q(n)r^{q(n)}) \) time, which is \( O(2^{p(n)}) \) for some \( p(n) \).
DTMs vs. NDTMs

Question:
- Could this DTM simulation of a NDTM be improved to run in polynomial time?
- Only if P = NP

Observations:
- This proof differs slightly from that given in the book, due to the different NDTM model.
- Note the “exhaustive search” nature of the simulation.
- For many problems in NP, exhaustive search is the only non-deterministic algorithm.
- Non-deterministic algorithms are starting to look like they might be useful for realistic problems, i.e., those using deterministic exponential time algorithms.
A NDTM is a seven-tuple:

\[ M = (Q, \Sigma, \Gamma, \delta, q_0, B, F) \]

- \( Q \) is a finite set of states
- \( \Gamma \) is a finite tape alphabet
- \( B \) is a distinguished blank symbol, which is in \( \Gamma \)
- \( \Sigma \) is a finite input alphabet, which is a subset of \( \Gamma \) – \{B\}
- \( q_i \) is the initial/starting state, \( q_0 \) is in \( Q \)
- \( q_f \) are halt states, which are in \( Q \)
- \( \delta \) is a next-move function, which is a mapping (i.e., may be undefined) from \((Q - \{q_f\}) \times \Gamma \) to finite subsets of \( Q \times \Gamma \times \{-1,+1\}\)

Exactly as before, however,…
Version #2 of a NDTM has two modules:

- Guessing module (non-deterministic)
- Verification module (deterministic)

Initial configuration:

- Input string x is placed in squares 1 through |x|, blanks everywhere else
- Write-only head is scanning square -1, read-write head is scanning square 1
Computation on $x$ takes place in two separate stages:

- Guessing stage - at each step non-deterministically decides whether to a) write a symbol, or b) stop.
  
  a) writes a symbol from $\Gamma$, selected non-deterministically, and moves one square to the left
  
  b) transfers control to the finite control

- Checking stage - the finite control starts in the start state $q_0$ and computes deterministically using the same rules as a DTM; the finite control either halts and accepts, halts and rejects, or loops infinitely
Notes:

- The guessing module can guess any string from $\Gamma^*$
  - Hmmmm…so what exactly does a guessing module look like?

- The guessing module allows for an infinite # of computations
  - This explains the need for the time complexity function to only consider accepting computations.

- The guessed string is usually used in the second stage
M accepts $x \in \Sigma^*$ if there exists a guess $y \in \Gamma^*$ that brings $M$ to $q_y$

- Note that this is reminiscent of our definition of string acceptance with NDTM Version #1

All definitions & theorems from the first version of NDTMs still apply:

- $L_M$, $T_M(n)$, NP
- $P \subseteq NP$ (verify this one)
- If $L \in NP$ then there exists a polynomial $p$ and a DTM such that $L = L_M$ and $T_M(n) \leq O(2^{p(n)})$
Are the two different versions of NDTMs equivalent?

In terms of the languages accepted, yes; there exists a NDTM version #1 that accepts a language L IFF there exist a NDTM version #2 that accepts L

Prove this as an exercise

But are the models equivalent with respect to time complexity?

Yes!

Do the two models define the same class NP?
Why do Garey and Johnson use Version #2?
- Convenience
- At the problem level, it is easier to show a problem (or language) is in NP

How does one show a problem (or language) is in P?

How does one show a problem (or language) is in NP?
To show $\Pi \in \text{NP}$ we need to give a NDTM, or rather, a non-deterministic algorithm for $\Pi$, that operates in polynomial time.

For most decision problems, Garey and Johnson’s NDTM model makes this easy.
Another way to look at model #2:

\[ \Pi \in \text{NP IFF there exists a deterministic polynomial time algorithm } A \text{ such that:} \]

- If \( I \in \Pi \), then there exists a “structure” \( S \) such that \((I, S)\) is accepted by the checking module.
- If \( I \notin \Pi \), then for any “structure” \( S \), \((I, S)\) is rejected by the checking module.
Recall TSP:

**TRAVELING SALESMAN**

INSTANCE: Set $C$ of $m$ cities, distance $d(c_i, c_j) \in \mathbb{Z}^+$ for each pair of cities $c_i, c_j \in C$ positive integer $B$.

QUESTION: Is there a tour of $C$ having length $B$ or less, i.e., a permutation $<c_{\pi(1)}, c_{\pi(2)}, \ldots, c_{\pi(m)}>$ of $C$ such that:

$$\sum_{i=1}^{m-1} d(c\pi(i), c\pi(i+1)) + d(c\pi(m), c\pi(1)) \leq B?$$

**A Non-Deterministic Polynomial Time Algorithm for TSP:**

- Guess (non-deterministically, in polynomial-time) a permutation of the cities.
- Verify (deterministically, in polynomial-time) that the permutation is a valid tour of the cities having total length less than $B$ (output “yes,” output “no” otherwise).

From this it follows that $TSP \in NP$
Recall Clique:

**CLIQUE**
INSTANCE: A Graph $G = (V, E)$ and a positive integer $J \leq |V|$.
QUESTION: Does $G$ contain a clique of size $J$ or more?

**A Non-Deterministic Polynomial Time Algorithm for Clique:**
- Guess (non-deterministically, in polynomial-time) a set of vertices from $G$.
- Verify (deterministically, in polynomial-time) that the set is a clique of size $J$ or more.

From this it follows that Clique $\in$ NP
Example #3: Graph K-Colorability

- Recall Graph K-Colorability:

  **GRAPH K-COLORABILITY**
  INSTANCE: A Graph $G = (V, E)$ and a positive integer $K \leq |V|$. 
  QUESTION: Is the graph $G$ $K$-colorable? 

- A Non-Deterministic Algorithm Polynomial Time for Graph K-colorability:
  - Guess (non-deterministically, in polynomial-time) a coloring of $G$.
  - Verify (deterministically, in polynomial-time) that the coloring is a valid $K$-coloring of $G$.

- From this it follows that Graph K-colorability $\in$ NP
Recall Satisfiability:

Satisfiability

INSTANCE: Set $U$ of variables and a collection $C$ of clauses over $U$.
QUESTION: Is there a satisfying truth assignment for $C$?

A Non-Deterministic Polynomial Time Algorithm for Satisfiability:
- Guess (non-deterministically, in polynomial-time) a truth assignment for the variables in $U$.
- Verify (deterministically, in polynomial-time) that the truth assignment satisfies $C$.

From this it follows that Satisfiability $\in$ NP
Example #5: Non-Tautology

**NON-TAUTOLOGY**

**INSTANCE:** Boolean expression $E (\land, \lor, \neg, \rightarrow)$.

**QUESTION:** Is $E$ not a tautology?

- A Non-Deterministic Polynomial Time Algorithm for Non-Tautology:
  - Guess (non-deterministically, in polynomial-time) a truth assignment for $E$.
  - Verify (deterministically, in polynomial-time) that the truth assignment does not satisfy $E$.

- From this it follows that Non-Tautology $\in$ NP
Example #6: Tautology

TAUTOLOGY

INSTANCE: Boolean expression $E (\wedge, \vee, \neg, \rightarrow)$.

QUESTION: Is $E$ a tautology?

- A Non-Deterministic Polynomial Time Algorithm for Tautology:
  - Guess?
  - Verify

- Is Tautology $\in$ NP?
  - We do not know!

- Exercise(s): Look at problems in the back of the book & verify they are in NP.
Stepping back for a moment…

- We know that $P \subseteq NP$

- If $P = NP$ then everything in NP can be solved in deterministic polynomial time.

- If $P \neq NP$ then:
  - $P \subseteq NP$
  - Everything in P can be solved in deterministic polynomial time
  - Everything in NP - P is intractable

- Unfortunately, we do not know if $P = NP$ or $P \neq NP$
Tentative View of the World of NP

- Given a problem $\Pi$ we would like to know if $\Pi \in P$ or $\Pi \in NP - P$

- For some problems $\Pi$ we can show that $\Pi \in P$

- For other problems $\Pi$ we would like to show that $\Pi \in NP - P$, but technically we can’t, at least not without resolving the $P =? NP$ question.

- What we can show, however, is something a bit weaker:

  "if $P \neq NP$ then $\Pi \in NP - P$"
More formally, we will show that many problems are NP-complete.

To show a problem \( \Pi \) is NP-complete, we will have to use a polynomial (time) transformation.

A polynomial transformation from a language \( L_1 \subseteq \Sigma_1^* \) to a language \( L_2 \subseteq \Sigma_2^* \) is a function \( f: \Sigma_1^* \rightarrow \Sigma_2^* \) that satisfies the following conditions:

- There is a polynomial time DTM program that computes \( f \)
- For all \( x \in \Sigma_1^* \), \( x \in L_1 \) if and only if \( f(x) \in L_2 \)

If there is a polynomial transformation from \( L_1 \) to \( L_2 \) then we write \( L_1 \propto L_2 \), read “\( L_1 \) transforms to \( L_2 \)”
The significance of polynomial transformations follows from Lemma 2.1

Lemma 2.1: If \( L_1 \propto L_2 \) and \( L_2 \in P \) then \( L_1 \in P \)

Proof: Let \( M \) be a polynomial time DTM that recognizes \( L_2 \), and let \( M' \) be a polynomial time DTM that computes \( f \). Then \( M' \) and \( M \) can be composed to give a polynomial time DTM that recognizes \( L_1 \).

Equivalently, if \( L_1 \propto L_2 \) and \( L_1 \notin P \) then \( L_2 \notin P \)
Informally, if \( L_1 \propto L_2 \) then it is frequently said that:

- \( L_1 \) is no harder than \( L_2 \)
- \( L_2 \) is no easier than \( L_1 \)
- \( L_2 \) is at least as hard as \( L_1 \)
- \( L_1 \) is as easy as \( L_2 \)

Note that “hard” and “easy” here are only with respect to polynomial v.s. exponential time:

- We could have \( L_1 \) recognized in \( 2^n \) time, \( L_2 \) recognized in \( 2^{\sqrt{n}} \) time, but where \( L_1 \) is not.
Another helpful factoid concerning transformations:

**Lemma 2.2:** If $L_1 \propto L_2$ and $L_2 \propto L_3$ then $L_1 \propto L_3$

In other words, polynomial transformations are transitive.
At the problem level a *polynomial transformation* from one decision problem $\Pi_1$ to another $\Pi_2$ is a function $f : D_{\Pi_1} \rightarrow D_{\Pi_2}$ that satisfies the following conditions:

- $f$ is computable by a polynomial time algorithm
- For all $I \in D_{\Pi_1}$, $I \in Y_{\Pi_1}$ if and only if $f(I) \in Y_{\Pi_2}$

We will deal with transformations only at the problem level.

Examples:
- SAT $\preceq$ 3SAT
- VC $\preceq$ TSP
- TSP $\preceq$ HC
A decision problem \( \Pi \) is NP-complete if:

- \( \Pi \in \text{NP} \), and
- For all other decision problems \( \Pi' \in \text{NP} \), \( \Pi' \preceq \Pi \)

For many problems, proving the first requirement is easy.

On the surface, the second requirement would appear very difficult to prove, especially given that there are an infinite number of problems in \( \text{NP} \).

As will be shown, the second requirement will be established, in part, by using the following lemma:

**Lemma 2.3:** If \( L_1 \) and \( L_2 \) belong to \( \text{NP} \), \( L_1 \) is NP-complete, and \( L_1 \preceq L_2 \) then \( L_2 \) is NP-complete.
Definition of NP-Complete, cont.

- Let \( \Pi \) be an NP-complete problem

- Observation:
  - \( \Pi \in P \) if and only if \( P = NP \)

(only if)

- If \( \Pi \) can be solved in polynomial time, then every problem in NP can be solved in polynomial time, by definition.
  - If \( \Pi \in P \) then \( P = NP \)

(if)

- If \( P=NP \), then every problem in NP can be solved in polynomial time. Since \( \Pi \in NP \) it follows that \( \Pi \) can be solved in polynomial time.
  - if \( P = NP \) then \( \Pi \in P \)
Could there be an NP-complete problem that is solvable in polynomial time, but where there is another problem in NP that requires exponential time? No!

For this reason, NP-complete problems are referred to as the “hardest” problems in NP.
Cooks Theorem - Background

- Our first NP-completeness proof will not use Lemma 2.3, but all others will.

SATISFIABILITY (SAT)
INSTANCE: Set $U$ of variables and a collection $C$ of clauses over $U$.
QUESTION: Is there a satisfying truth assignment for $C$?

- Example #1:
  \[ U = \{ u_1, u_2 \} \]
  \[ C = \{ \{ u_1, \overline{u}_2 \}, \{ \overline{u}_1, u_2 \} \} \]
  Answer is “yes” - satisfiable by make both variables $T$

- Example #2:
  \[ U = \{ u_1, u_2 \} \]
  \[ C = \{ \{ u_1, u_2 \}, \{ u_1, \overline{u}_2 \}, \{ \overline{u}_1 \} \} \]
  Answer is “no”
Statement of Cook’s Theorem

- SAT is shown NP-complete by Cook’s theorem.

- Question: Is Cook’s theorem a polynomial-time transformation?

- Answer: Sort of – it is a generic polynomial-time transformation.

- All other NP-completeness proofs perform a transformation from a specific problem known to be NP-complete.

- Cook’s theorem is generic in that it doesn’t transform a specific problem to SAT, but rather, given any specific problem in NP, that problem combined with Cook’s theorem will give a specific polynomial-time transformation.
Recall that a polynomial-time transformation is a function that can be computed by a polynomial-time DTM (i.e., algorithm) $A$.

Questions:
- What is the function computed by $A$?
- What is the input to $A$?
- What is the output of $A$?
Cook’s Theorem

Cook’s theorem outlines a \textit{generic} transformation from one unspecified language in NP language to SAT:

\[ \text{...} \rightarrow A \rightarrow \text{SAT} \]
Cook’s theorem outlines a generic transformation from one unspecified language in NP language to SAT:

- Given an actual polynomial-time NDTM M that accepts an actual language L, the generic transformation along with M will give an actual transformation from L to SAT.

- Thus, with one generic transformation, Cook’s theorem simultaneously gives a transformation from every problem in NP to SAT.
Cook’s Theorem

For a particular polynomial-time NDTM M:

- What is the input for the transformation?
  - Answer: Any string $x$ in $\Sigma^*$

- What is the output?
  - Answer: An instance of SAT

- What is the function computed by $A$?
  - $f : \Sigma^* \rightarrow \{y \mid y \text{ is an instance of SAT such that } x \text{ is in } L_M \text{ iff } y \text{ is satisfiable} \}$
Cook’s Theorem

- How does this show SAT is NP-complete?
  - For a \( L \in NP \), by definition, there exists a polynomial-time NDTM \( M \) such that \( L = L_M \).
  - Given \( M \), Cook’s theorem translates from a generic polynomial-time transformation to a specific polynomial-time transformation \( f_L \).

- The transformation \( f_L \) will have the following properties:
  - There exists a polynomial-time DTM that computes \( f_L \)
  - For all \( x \) in \( \Sigma^* \), \( x \) is in \( L \) iff \( f_L(x) \) is satisfiable.
**Theorem**: SAT is NP-complete (Cook).

**Proof**:
1. SAT $\in$ NP
2. We must show that for all $\Pi \in$ NP, $\Pi \approx$ SAT
Cook’s Theorem

- Suppose \( L \in \text{NP} \).

- By definition there exists a NDTM \( M = (Q, \Sigma, \Gamma, \delta, q_0, q_y, q_n, B, F) \) such that \( M \) accepts \( L \) and \( T_M(n) \leq p(n) \).

- An accepting computation by \( M \) on any \( x \) in \( \Sigma^* \) can be described by a sequence of “snapshots” or instantaneous descriptions.

- More specifically, if \( x \in L_M \) and \( |x| = n \) then \( M \)'s accepting computation on \( x \) can be described by a sequence of \( p(n)+1 \) snapshots.

- Each such snapshot consists of a finite amount of information:
  - Current state
  - Tape contents from tape square \(-p(n)\) up to \( p(n)+1\)
  - Current position of the read/write head
Cook’s Theorem

The information in each snapshot, and hence the entire computation, can be described by assigning truth values to a collection of Boolean variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Range</th>
<th>Intended meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q[i,k]$</td>
<td>$0 \leq i \leq p(n)$</td>
<td>At time $i$, $M$ is in state $q_k$</td>
</tr>
<tr>
<td></td>
<td>$0 \leq k \leq r$</td>
<td></td>
</tr>
<tr>
<td>$H[i,j]$</td>
<td>$0 \leq i \leq p(n)$</td>
<td>At time $i$, the read-write head is scanning tape square $j$</td>
</tr>
<tr>
<td></td>
<td>$-p(n) \leq j \leq p(n)+1$</td>
<td></td>
</tr>
<tr>
<td>$S[i,j,k]$</td>
<td>$0 \leq i \leq p(n)$</td>
<td>At time $i$, the contents of tape square $j$ is symbol $s_k$</td>
</tr>
<tr>
<td></td>
<td>$-p(n) \leq j \leq p(n)+1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$0 \leq k \leq v$</td>
<td></td>
</tr>
</tbody>
</table>

- A computation of $M$ induces a truth assignment on these variables.

- An (arbitrary) truth assignment to the variables, however, does not necessarily correspond to a valid computation.
Recall the following TM: \( \{ w \mid w \text{ is in } \{0,1\}^* \text{ and } w \text{ ends with a } 0 \} \)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0)</td>
<td>((q_0, 0, R))</td>
<td>((q_0, 1, R))</td>
<td>((q_1, B, L))</td>
</tr>
<tr>
<td>(q_1)</td>
<td>((q_2, 0, R))</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(q_2)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Sample Computation: (on 010)

\(q_0010\)

state “lightbulbs”

\(\bullet\)  \(\bigcirc\)  \(\bigcirc\)
Recall the following TM: \( \{ w \mid w \text{ is in } \{0,1\}^* \text{ and } w \text{ ends with a 0} \} \)

<table>
<thead>
<tr>
<th>State</th>
<th>0</th>
<th>1</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>((q_0, 0, R))</td>
<td>((q_0, 1, R))</td>
<td>((q_1, B, L))</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>((q_2, 0, R))</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Sample Computation: (on 010)

\[ q_0010 \rightarrow 0q_010 \]

state “lightbulbs”

\[ \text{state “lightbulbs”} \]

\[ \text{state “lightbulbs”} \]

\[ \text{state “lightbulbs”} \]
Recall the following TM: \( \{w \mid w \text{ is in } \{0,1\}^* \text{ and } w \text{ ends with a } 0\} \)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0)</td>
<td>((q_0, 0, R))</td>
<td>((q_0, 1, R))</td>
<td>((q_1, B, L))</td>
</tr>
<tr>
<td>(q_1)</td>
<td>((q_2, 0, R))</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(q_2)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Sample Computation: (on 010)

\[
\begin{align*}
q_0010 & \rightarrow 0q_010 \\
\quad & \rightarrow 01q_00
\end{align*}
\]

state “lightbulbs”

[Diagram with state “lightbulbs” indicated by symbols]
Recall the following TM: \( \{w \mid w \text{ is in } \{0,1\}^* \text{ and } w \text{ ends with a } 0\} \)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0)</td>
<td>((q_0, 0, R))</td>
<td>((q_0, 1, R))</td>
<td>((q_1, B, L))</td>
</tr>
<tr>
<td>(q_1)</td>
<td>((q_2, 0, R))</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(q_2)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Sample Computation: (on 010)

\[
\begin{array}{l}
q_0010 \rightarrow q_010 \\
\rightarrow 0q_010 \\
\rightarrow 01q_00 \\
\rightarrow 010q_0
\end{array}
\]

state “lightbulbs”

\[\bullet \quad \bigcirc \quad \bigcirc\]
Recall the following TM: \( \{w \mid w \text{ is in } \{0,1\}^* \text{ and } w \text{ ends with a } 0\} \)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0)</td>
<td>((q_0), 0, R)</td>
<td>((q_0), 1, R)</td>
<td>((q_1), B, L)</td>
</tr>
<tr>
<td>(q_1)</td>
<td>((q_2), 0, R)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(q_2)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Sample Computation: (on 010)

\[
\begin{align*}
q_0 & \rightarrow 0q_010 \\
& \rightarrow 01q_00 \\
& \rightarrow 010q_0 \\
& \rightarrow 01q_10 \\
\end{align*}
\]

state “lightbulbs”

○ ○ ○
Recall the following TM: \( \{w \mid w \text{ is in } \{0,1\}^* \text{ and } w \text{ ends with a } 0\} \)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0)</td>
<td>(q_0, 0, R)</td>
<td>(q_0, 1, R)</td>
<td>(q_1, B, L)</td>
</tr>
<tr>
<td>(q_1)</td>
<td>(q_2, 0, R)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(q_2)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Sample Computation: (on 010)

- \(q_0010 \rightarrow 0q_010\)
- \(0q_010 \rightarrow 01q_00\)
- \(01q_00 \rightarrow 010q_0\)
- \(010q_0 \rightarrow 01q_10\)
- \(01q_10 \rightarrow 010q_2\)

Similarly, there could be tape head position lightbulbs or tape square contents lightbulbs.
Cook’s Theorem

- The information in each snapshot, and hence the entire computation, can be described by assigning truth values to a collection of Boolean variables:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Range</th>
<th>Intended meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q[i,k]$</td>
<td>$0 \leq i \leq p(n)$</td>
<td>At time $i$, $M$ is in state $q_k$</td>
</tr>
<tr>
<td></td>
<td>$0 \leq k \leq r$</td>
<td></td>
</tr>
<tr>
<td>$H[i,j]$</td>
<td>$0 \leq i \leq p(n)$</td>
<td>At time $i$, the read-write head is scanning tape square $j$</td>
</tr>
<tr>
<td></td>
<td>$-p(n) \leq j \leq p(n)+1$</td>
<td></td>
</tr>
<tr>
<td>$S[i,j,k]$</td>
<td>$0 \leq i \leq p(n)$</td>
<td>At time $i$, the contents of tape square $j$ is symbol $s_k$</td>
</tr>
<tr>
<td></td>
<td>$-p(n) \leq j \leq p(n)+1$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$0 \leq k \leq v$</td>
<td></td>
</tr>
</tbody>
</table>

- A computation of $M$ induces a truth assignment on these variables.

- An (arbitrary) truth assignment to the variables, however, does not necessarily correspond to a valid computation.
Cook’s Theorem

For this computation the state variables are:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Range</th>
<th>Intended meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q[i,k]$</td>
<td>$0 \leq i \leq p(n)$</td>
<td>At time $i$, $M$ is in state $q_k$</td>
</tr>
<tr>
<td></td>
<td>$0 \leq k \leq r$</td>
<td></td>
</tr>
</tbody>
</table>

Time

State

<table>
<thead>
<tr>
<th>State</th>
<th>$Q[0,0]$</th>
<th>$Q[1,0]$</th>
<th>$Q[2,0]$</th>
<th>$Q[3,0]$</th>
<th>$Q[4,0]$</th>
<th>$Q[5,0]$</th>
</tr>
</thead>
</table>
Cook’s Theorem

- And their values are:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Range</th>
<th>Intended meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q[i,k]$</td>
<td>$0 \leq i \leq p(n)$</td>
<td>At time $i$, $M$ is in state $q_k$</td>
</tr>
<tr>
<td></td>
<td>$0 \leq k \leq r$</td>
<td></td>
</tr>
</tbody>
</table>

- Notice that each column contains exactly one “T”
The **tape head position** variables are:

<table>
<thead>
<tr>
<th>Variable $H[i,j]$</th>
<th>Range $0 \leq i \leq p(n)$, $-p(n) \leq j \leq p(n)+1$</th>
<th>Intended meaning</th>
<th>At time $i$, the read-write head is scanning tape square $j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H[0,0]$</td>
<td>$H[1,0]$ $H[2,0]$ $H[3,0]$ $H[4,0]$ $H[5,0]$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cook’s Theorem

And their values are:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Range</th>
<th>Intended meaning</th>
</tr>
</thead>
</table>
| $H_{[i,j]}$ | $0 \leq i \leq p(n)$  
- $-p(n) \leq j \leq p(n)+1$ | At time $i$, the read-write head is scanning tape square $j$ |

<table>
<thead>
<tr>
<th>$H_{[0,-5]}$</th>
<th>$F$</th>
<th>$H_{[1,-5]}$</th>
<th>$F$</th>
<th>$H_{[2,-5]}$</th>
<th>$F$</th>
<th>$H_{[3,-5]}$</th>
<th>$F$</th>
<th>$H_{[4,-5]}$</th>
<th>$F$</th>
<th>$H_{[5,-5]}$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$H_{[0,-4]}$</td>
<td>$F$</td>
<td>$H_{[1,-4]}$</td>
<td>$F$</td>
<td>$H_{[2,-4]}$</td>
<td>$F$</td>
<td>$H_{[3,-4]}$</td>
<td>$F$</td>
<td>$H_{[4,-4]}$</td>
<td>$F$</td>
<td>$H_{[5,-4]}$</td>
<td>$F$</td>
</tr>
<tr>
<td>$H_{[0,-3]}$</td>
<td>$F$</td>
<td>$H_{[1,-3]}$</td>
<td>$F$</td>
<td>$H_{[2,-3]}$</td>
<td>$F$</td>
<td>$H_{[3,-3]}$</td>
<td>$F$</td>
<td>$H_{[4,-3]}$</td>
<td>$F$</td>
<td>$H_{[5,-3]}$</td>
<td>$F$</td>
</tr>
<tr>
<td>$H_{[0,-2]}$</td>
<td>$F$</td>
<td>$H_{[1,-2]}$</td>
<td>$F$</td>
<td>$H_{[2,-2]}$</td>
<td>$F$</td>
<td>$H_{[3,-2]}$</td>
<td>$F$</td>
<td>$H_{[4,-2]}$</td>
<td>$F$</td>
<td>$H_{[5,-2]}$</td>
<td>$F$</td>
</tr>
<tr>
<td>$H_{[0,-1]}$</td>
<td>$F$</td>
<td>$H_{[1,-1]}$</td>
<td>$F$</td>
<td>$H_{[2,-1]}$</td>
<td>$F$</td>
<td>$H_{[3,-1]}$</td>
<td>$F$</td>
<td>$H_{[4,-1]}$</td>
<td>$F$</td>
<td>$H_{[5,-1]}$</td>
<td>$F$</td>
</tr>
<tr>
<td>$H_{[0,0]}$</td>
<td>$F$</td>
<td>$H_{[1,0]}$</td>
<td>$F$</td>
<td>$H_{[2,0]}$</td>
<td>$F$</td>
<td>$H_{[3,0]}$</td>
<td>$F$</td>
<td>$H_{[4,0]}$</td>
<td>$F$</td>
<td>$H_{[5,0]}$</td>
<td>$F$</td>
</tr>
<tr>
<td>$H_{[0,1]}$</td>
<td>$T$</td>
<td>$H_{[1,1]}$</td>
<td>$F$</td>
<td>$H_{[2,1]}$</td>
<td>$F$</td>
<td>$H_{[3,1]}$</td>
<td>$F$</td>
<td>$H_{[4,1]}$</td>
<td>$F$</td>
<td>$H_{[5,1]}$</td>
<td>$F$</td>
</tr>
<tr>
<td>$H_{[0,2]}$</td>
<td>$F$</td>
<td>$H_{[1,2]}$</td>
<td>$F$</td>
<td>$H_{[2,2]}$</td>
<td>$F$</td>
<td>$H_{[3,2]}$</td>
<td>$F$</td>
<td>$H_{[4,2]}$</td>
<td>$F$</td>
<td>$H_{[5,2]}$</td>
<td>$T$</td>
</tr>
<tr>
<td>$H_{[0,3]}$</td>
<td>$F$</td>
<td>$H_{[1,3]}$</td>
<td>$T$</td>
<td>$H_{[2,3]}$</td>
<td>$F$</td>
<td>$H_{[3,3]}$</td>
<td>$F$</td>
<td>$H_{[4,3]}$</td>
<td>$T$</td>
<td>$H_{[5,3]}$</td>
<td>$F$</td>
</tr>
<tr>
<td>$H_{[0,4]}$</td>
<td>$F$</td>
<td>$H_{[1,4]}$</td>
<td>$T$</td>
<td>$H_{[2,4]}$</td>
<td>$T$</td>
<td>$H_{[3,4]}$</td>
<td>$F$</td>
<td>$H_{[4,4]}$</td>
<td>$F$</td>
<td>$H_{[5,4]}$</td>
<td>$F$</td>
</tr>
<tr>
<td>$H_{[0,5]}$</td>
<td>$F$</td>
<td>$H_{[1,5]}$</td>
<td>$F$</td>
<td>$H_{[2,5]}$</td>
<td>$F$</td>
<td>$H_{[3,5]}$</td>
<td>$F$</td>
<td>$H_{[4,5]}$</td>
<td>$F$</td>
<td>$H_{[5,5]}$</td>
<td>$F$</td>
</tr>
<tr>
<td>$H_{[0,6]}$</td>
<td>$F$</td>
<td>$H_{[1,6]}$</td>
<td>$F$</td>
<td>$H_{[2,6]}$</td>
<td>$F$</td>
<td>$H_{[3,6]}$</td>
<td>$F$</td>
<td>$H_{[4,6]}$</td>
<td>$F$</td>
<td>$H_{[5,6]}$</td>
<td>$F$</td>
</tr>
</tbody>
</table>

Notice once again that each column contains exactly one “T”
### Cook’s Theorem

The **tape content variables** are:

<table>
<thead>
<tr>
<th>Variable $S[i,j,k]$</th>
<th>Range $0 \leq i \leq p(n)$</th>
<th>Range $-p(n) \leq j \leq p(n)+1$</th>
<th>Intended meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$0 \leq k \leq v$</td>
<td></td>
<td>At time $i$, the contents of tape</td>
</tr>
</tbody>
</table>

The intended meaning is that **square $j$ is symbol $s_k$**.

<table>
<thead>
<tr>
<th>$S[0,0,0]$</th>
<th>$S[0,0,0]$</th>
<th>$S[1,0,0]$</th>
<th>$S[2,0,0]$</th>
<th>$S[2,1,0]$</th>
<th>$S[3,1,0]$</th>
<th>$S[4,1,0]$</th>
<th>$S[5,1,0]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S[0,0,1]$</td>
<td>$S[0,0,1]$</td>
<td>$S[1,0,1]$</td>
<td>$S[2,0,1]$</td>
<td>$S[2,1,1]$</td>
<td>$S[3,1,1]$</td>
<td>$S[4,1,1]$</td>
<td>$S[5,1,1]$</td>
</tr>
<tr>
<td>$S[0,0,2]$</td>
<td>$S[0,0,2]$</td>
<td>$S[1,0,2]$</td>
<td>$S[2,0,2]$</td>
<td>$S[2,1,2]$</td>
<td>$S[3,1,2]$</td>
<td>$S[4,1,2]$</td>
<td>$S[5,1,2]$</td>
</tr>
<tr>
<td>$S[0,0,3]$</td>
<td>$S[0,0,3]$</td>
<td>$S[1,0,3]$</td>
<td>$S[2,0,3]$</td>
<td>$S[2,1,3]$</td>
<td>$S[3,1,3]$</td>
<td>$S[4,1,3]$</td>
<td>$S[5,1,3]$</td>
</tr>
<tr>
<td>$S[0,0,5]$</td>
<td>$S[0,0,5]$</td>
<td>$S[1,0,5]$</td>
<td>$S[2,0,5]$</td>
<td>$S[2,1,5]$</td>
<td>$S[3,1,5]$</td>
<td>$S[4,1,5]$</td>
<td>$S[5,1,5]$</td>
</tr>
</tbody>
</table>

**Cook’s Theorem** provides a framework for understanding the computational complexity of problems, particularly in the context of **NP-completeness**. The theorem is crucial in computational theory and has implications for various fields, including computer science, mathematics, and cognitive science. This foundational concept helps in identifying problems that are computationally intractable, guiding the development of efficient algorithms and the design of computational models.
Cook’s Theorem

- And their values for symbol 0 (i.e., B) are:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Range</th>
<th>Intended meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>S[i,j,k]</td>
<td>0 ≤ i ≤ p(n) -p(n) ≤ j ≤ p(n)+1 0 ≤ k ≤ v</td>
<td>At time i, the contents of tape square j is symbol s_k</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>S[0,-5,0]</th>
<th>S[1,-5,0]</th>
<th>S[2,-5,0]</th>
<th>S[3,-5,0]</th>
<th>S[4,-5,0]</th>
<th>S[5,-5,0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>S[0,-4,0]</td>
<td>S[1,-4,0]</td>
<td>S[2,-4,0]</td>
<td>S[3,-4,0]</td>
<td>S[4,-4,0]</td>
<td>S[5,-4,0]</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>S[0,-3,0]</td>
<td>S[1,-3,0]</td>
<td>S[2,-3,0]</td>
<td>S[3,-3,0]</td>
<td>S[4,-3,0]</td>
<td>S[5,-3,0]</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>S[0,-2,0]</td>
<td>S[1,-2,0]</td>
<td>S[2,-2,0]</td>
<td>S[3,-2,0]</td>
<td>S[4,-2,0]</td>
<td>S[5,-2,0]</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>S[0,-1,0]</td>
<td>S[1,-1,0]</td>
<td>S[2,-1,0]</td>
<td>S[3,-1,0]</td>
<td>S[4,-1,0]</td>
<td>S[5,-1,0]</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>S[0,0,0]</td>
<td>S[1,0,0]</td>
<td>S[2,0,0]</td>
<td>S[3,0,0]</td>
<td>S[4,0,0]</td>
<td>S[5,0,0]</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>S[0,1,0]</td>
<td>S[1,1,0]</td>
<td>S[2,1,0]</td>
<td>S[3,1,0]</td>
<td>S[4,1,0]</td>
<td>S[5,1,0]</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>S[0,2,0]</td>
<td>S[1,2,0]</td>
<td>S[2,2,0]</td>
<td>S[3,2,0]</td>
<td>S[4,2,0]</td>
<td>S[5,2,0]</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>S[0,3,0]</td>
<td>S[1,3,0]</td>
<td>S[2,3,0]</td>
<td>S[3,3,0]</td>
<td>S[4,3,0]</td>
<td>S[5,3,0]</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>S[0,4,0]</td>
<td>S[1,4,0]</td>
<td>S[2,4,0]</td>
<td>S[3,4,0]</td>
<td>S[4,4,0]</td>
<td>S[5,4,0]</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>S[0,5,0]</td>
<td>S[1,5,0]</td>
<td>S[2,5,0]</td>
<td>S[3,5,0]</td>
<td>S[4,5,0]</td>
<td>S[5,5,0]</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>S[0,6,0]</td>
<td>S[1,6,0]</td>
<td>S[2,6,0]</td>
<td>S[3,6,0]</td>
<td>S[4,6,0]</td>
<td>S[5,6,0]</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

- Notice that every column is identical.
### Cook’s Theorem

- And their values for symbol 1 (i.e., 0) are:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Range</th>
<th>Intended meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S[i,j,k]$</td>
<td>$0 \leq i \leq p(n)$, $-p(n) \leq j \leq p(n)+1$, $0 \leq k \leq v$</td>
<td>At time $i$, the contents of tape square $j$ is symbol $s_k$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S[0,-1,1]</th>
<th>S[1,-1,1]</th>
<th>S[2,-1,1]</th>
<th>S[3,-1,1]</th>
<th>S[4,-1,1]</th>
<th>S[5,-1,1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S[0,0,1]</th>
<th>S[1,0,1]</th>
<th>S[2,0,1]</th>
<th>S[3,0,1]</th>
<th>S[4,0,1]</th>
<th>S[5,0,1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S[0,1,1]</th>
<th>S[1,1,1]</th>
<th>S[2,1,1]</th>
<th>S[3,1,1]</th>
<th>S[4,1,1]</th>
<th>S[5,1,1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S[0,6,1]</th>
<th>S[1,6,1]</th>
<th>S[2,6,1]</th>
<th>S[3,6,1]</th>
<th>S[4,6,1]</th>
<th>S[5,6,1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

- Notice that every column is identical.
Cook’s Theorem

- And their values for symbol 2 (i.e., 1) are:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Range</th>
<th>Intended meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S[i,j,k]$</td>
<td>$0 \leq i \leq p(n)$</td>
<td>At time $i$, the contents of tape square $j$ is symbol $s_k$</td>
</tr>
<tr>
<td>$-p(n) \leq j \leq p(n)+1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$0 \leq k \leq v$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| $F$ | $F$ | $F$ | $F$ | $F$ | $F$ |
| $F$ | $F$ | $F$ | $F$ | $F$ | $F$ |
| $F$ | $F$ | $F$ | $F$ | $F$ | $F$ |
| $T$ | $T$ | $T$ | $T$ | $T$ | $T$ |
| $S[0,1,2]$ | $S[1,1,2]$ | $S[2,1,2]$ | $S[3,1,2]$ | $S[4,1,2]$ | $S[5,1,2]$ |
| $F$ | $F$ | $F$ | $F$ | $F$ | $F$ |
| $S[0,0,2]$ | $S[1,0,2]$ | $S[2,0,2]$ | $S[3,0,2]$ | $S[4,0,2]$ | $S[5,0,2]$ |
| $F$ | $F$ | $F$ | $F$ | $F$ | $F$ |
| $S[0,1,2]$ | $S[1,1,2]$ | $S[2,1,2]$ | $S[3,1,2]$ | $S[4,1,2]$ | $S[5,1,2]$ |
| $F$ | $F$ | $F$ | $F$ | $F$ | $F$ |
| $T$ | $T$ | $T$ | $T$ | $T$ | $T$ |
| $F$ | $F$ | $F$ | $F$ | $F$ | $F$ |
| $F$ | $F$ | $F$ | $F$ | $F$ | $F$ |
| $F$ | $F$ | $F$ | $F$ | $F$ | $F$ |
| $S[0,6,2]$ | $S[1,6,2]$ | $S[2,6,2]$ | $S[3,6,2]$ | $S[4,6,2]$ | $S[5,6,2]$ |
| $F$ | $F$ | $F$ | $F$ | $F$ | $F$ |

- Notice that every column is identical.
The assignment reflects the entire computation on a given string $x$.
- The state it is in at each step
- The contents of each tape square at each step
- The location of the tape head at each step

A different computation, on a different input string, gives a different assignment to the variables.

An arbitrary truth assignment doesn’t necessarily give a valid computation, accepting or otherwise.
- What if all of the variables are assigned “T”?
- What if all are assigned “F”?
- How about a random assignment?
Cook’s Theorem

- In addition, in general, a NDTM has many computation on a given string $x$, and each corresponds to many truth assignments.

- A DTM, in contrast, has only one computation on a given string, which corresponds to one truth assignment.
Now for the big part…

The transformation constructs clauses in such a way that there is a satisfying truth assignment iff that assignment corresponds to an accepting computation.

There are 6 groups of clauses, each imposing a different constraint on a satisfying assignment, and the corresponding computation.

G1 : At each time $i$, $M$ is in exactly one state.

G2 : At each time $i$, the read-write head is scanning exactly one tape square.

G3 : At each time $i$, each tape square contains exactly one symbol from $\Gamma$.

G4 : At time 0, the computation is in the initial configuration of its checking stage for input $x$.

G5 : By time $p(n)$, $M$ has entered state $q_y$ and hence has accepted $x$.

G6 : For each time $i$, $0 \leq i \leq p(n)$, the configuration of $M$ at time $i+1$ follows by a single application of the transition function $\delta$ from the configuration at time $i$. 
Cook’s Theorem

- G1: At each time $i$, $M$ is in exactly one state.

$\{Q[i,0], Q[i,1], \ldots, Q[i,r]\}, \quad 0 \leq i \leq p(n)$

At each time $i$, $M$ must be in at least one state.

$\{\neg Q[i,j], \neg Q[i,j']\}, \quad 0 \leq i \leq p(n), \quad 0 \leq j < j' \leq r$

At each time $i$, $M$ cannot be in more than one state.
Cook’s Theorem

- **G2**: At each time $i$, the read/write head is scanning exactly one tape square.

\[
\{H[i,-p(n)], H[i,-p(n)+1], \ldots, H[i,p(n)+1]\} \quad 0 \leq i \leq p(n)
\]

At each time $i$, $M$ must be scanning at least one tape square.

\[
\{\sim H[i,j], \sim H[i,j']\} \quad 0 \leq i \leq p(n)
\]

- $-p(n) \leq j < j' \leq p(n)+1$

At each time $i$, $M$ cannot be scanning more than one tape square.
Cook’s Theorem

G3: At each time $i$, each tape square contains exactly one symbol from $\Gamma$.

\[
\{S[i,j,0], S[i,j,1], \ldots, S[i,j,v]\}
\]

$0 \leq i \leq p(n)$

$-p(n) \leq j \leq p(n)+1$

At each time $i$, each tape square contains at least one symbol from $\Gamma$.

\[
\{\sim S[i,j,k], \sim S[i,j,k']\}
\]

$0 \leq i \leq p(n)$

$-p(n) \leq j \leq j' \leq p(n)+1$

$0 \leq k \leq k' \leq v$

At each time $i$, $M$ cannot be scanning more than one tape square.
G4 : At time 0, the computation is in the initial configuration of its checking stage for input $x$.

\[
\begin{align*}
\mathcal{Q}[0,0] & \quad \text{At time 0, } M \text{ must be in state } q_0. \\
\mathcal{H}[0,1] & \quad \text{At time 0, } M \text{ must be scanning tape square 1.} \\
\mathcal{S}[0,0,0] & \quad \text{At time 0, } M \text{ must have a blank on tape square 0.} \\
\mathcal{S}[0,1,k_1], \mathcal{S}[0,2,k_2], \ldots, \mathcal{S}[0,n,k_n] & \quad \text{At each time 0, the string } x = s_{k_1} s_{k_2} \ldots s_{k_n} \text{ is in positions 1..n on the tape.} \\
\mathcal{S}[0,n+1,0], \mathcal{S}[0,n+2,0], \ldots, \mathcal{S}[0,p(n)+1,0] & \quad \text{At each time 0, all tap squares to the right of the input contain blanks (where } x = s_{k_1} s_{k_2} \ldots s_{k_n}).
\end{align*}
\]
Cook’s Theorem

G5: By time $p(n)$, $M$ has entered state $q_y$ and hence has accepted $x$.

$\{Q[p(n),1]\}$
- G6: For each time \( i, 0 \leq i \leq p(n) \), the configuration of \( M \) at time \( i+1 \) follows by a single application of the transition function \( \delta \) from the configuration at time \( i \).
  - Part 1 – if a tape square isn’t being scanned by the tape head, then it doesn’t change.
  - Part 2 – any changes that occur to either a tape square, head position, or current state, do so according to \( \delta \).

- Suppose that \( q_k \in Q \setminus \{q_Y, q_N\} \) and that \( \delta(q_k, s_l) = (q_k', s_l', \Delta) \).

- In the case where \( q_k \in \{q_Y, q_N\} \) then \( \Delta = 0, k' = k, \) and \( l' = l. \)
Cook’s Theorem

- Part 1: if a tape square isn’t being scanned by the tape head, then it doesn’t change.

\[ \{\neg S[i,j,l], H[i,j], S[i+1,j,l]\} \quad 0 \leq i \leq p(n) \]
\[-p(n) \leq j \leq p(n)+1 \]
\[0 \leq l \leq v\]

- A more understandable way to write the clause:

if \((S[i,j,l] \text{ and } \neg H[i,j])\) then \(S[i+1,j,l]\)
\[0 \leq i \leq p(n)\]
\[-p(n) \leq j \leq p(n)+1\]
\[0 \leq l \leq v\]

At time \(i\), if tape square \(j\) contains symbol \(l\), but \(M\) is not scanning tape square \(j\), then at time \(i+1\) the tape square still contains symbol \(l\).
Part 2: any changes that occur to either a tape square, head position, or current state, do so according to $\delta$.

- State change
- A symbol is written
- Tape head moves 1 square
Cook’s Theorem

- State change:

\[
\{ \neg H[i,j], \neg Q[i,k], \neg S[i,j,l], Q[i+1,k'], } \]

\[
0 \leq i \leq p(n) \]
\[
-p(n) \leq j \leq p(n)+1 \]
\[
0 \leq k,k' \leq r \]
\[
0 \leq l \leq v \]

- A more understandable way to write the clause:

if \((H[i,j] \text{ and } Q[i,k] \text{ and } S[i,j,l])\) then \(Q[i+1,k']\)

\[
0 \leq i \leq p(n) \]
\[
-p(n) \leq j \leq p(n)+1 \]
\[
0 \leq k,k' \leq r \]
\[
0 \leq l \leq v \]

At time \(i\), if the tape head is scanning tape square \(j\), and \(M\) is in state \(k\), and if tape square \(j\) contains symbol \(l\), then at time \(i+1\) the state has changed to \(k'\).
Cook’s Theorem

A symbol is written:

\[
\{ \sim H[i,j], \sim Q[i,k], \sim S[i,j,l], S[i+1,l'], \}
\]

\[
0 \leq i \leq p(n)
\]

\[
-p(n) \leq j \leq p(n)+1
\]

\[
0 \leq k \leq r
\]

\[
0 \leq l, l' \leq v
\]

A more understandable way to write the clause:

if \((H[i,j] \text{ and } Q[i,k] \text{ and } S[i,j,l])\) then \(S[i+1,l']\)

At time \(i\), if the tape head is scanning tape square \(j\), and \(M\) is in state \(k\), and if tape square \(j\) contains symbol \(l\), then at time \(i+1\) the tape head has written symbol \(l'\) to position \(j\).
Cook’s Theorem

- Tape head moves 1 square:

\[
\{ \sim H[i,j], \sim Q[i,k], \sim S[i,j,l], H[i+1,j+\Delta], \} \\
0 \leq i \leq p(n) \\
-p(n) \leq j \leq p(n)+1 \\
0 \leq k \leq r \\
0 \leq l \leq v
\]

- A more understandable way to write the clause:

if \( (H[i,j] \text{ and } Q[i,k] \text{ and } S[i,j,l]) \) then \( H[i+1,j+\Delta] \)

\[
0 \leq i \leq p(n) \\
-p(n) \leq j \leq p(n)+1 \\
0 \leq k \leq r \\
0 \leq l \leq v
\]

At time \( i \), if the tape head is scanning tape square \( j \), and \( M \) is in state \( k \), and if tape square \( j \) contains symbol \( l \), then at time \( i+1 \) the tape head has moved to tape square \( j+\Delta \).
Let M be a fixed, polynomial-time NDTM M where \( L = L_M \).

The NDTM M, along with Cook's theorem, gives an algorithm \( A_M \) that transforms \( L \) to SAT.

Given \( x \in \Sigma^* \), \( A_M \) constructs a set of clauses forming a Boolean formula such that:

\[
f_L(x) \text{ is satisfiable iff } x \in L
\]
Cook’s Theorem

Diagrammatically:

- $x \in \Sigma^*$
- $A \rightarrow f_L(x)$
- Runs in deterministic polynomial-time

L “instance”

SAT instance
Cook’s Theorem

- If there is an accepting computation of M on x, then are the clauses satisfiable?
- If the clauses are satisfiable, then is there an accepting computation of M on x?
Can the transformation be performed in deterministic polynomial-time?

Let \( L \in \text{NP} \) and \( M \) be a polynomial-time NDTM such that \( L = L_M \)

It can be verified that:
- \( |U| = O(p(n)^2) \)
- \( |C| = O(p(n)^2) \)

Therefore, \( O(p(n)^4) \) total time.
One final question…

Where is the guess in all of the clauses, or how is it represented?
  - Note that the S[0,j,k] variables, where j<=0, don’t appear in any clauses!
  - The assignments to the variables in the third group of clauses at time 0 on tape squares -p(n) to -1 tells us what the guess is.

Suppose \(SAT\) is satisfiable. Can the non-deterministic guess be determined from the satisfying truth assignment?
  - Yes, each satisfying truth assignment defines a guess and a computation
  - Note that the NDTM used in the construction could be based on either model #1 or model #2