FORMAL LANGUAGES, AUTOMATA AND COMPUTATION

NP-COMPLETENESS

Carnegie Mellon University in Qatar
**SUMMARY**

- Time complexity: Big-O notation, asymptotic complexity
- Simulation of multi-tape TMs with a single tape deterministic TM can be done with a polynomial slow-down.
- Simulation of nondeterministic TMs with a deterministic TM is exponentially slower.
- The Class P: The class of languages for which membership can be *decided* quickly.
- The Class NP: The class of languages for which membership can be *verified* quickly.

We do not yet know if $P = NP$, or not.
The best method known for solving languages in NP deterministically uses exponential time, that is

\[ \text{NP} \subseteq \text{EXPTIME} = \bigcup_{k} \text{TIME}(2^{nk}) \]

It is not known whether NP is contained in a smaller deterministic time complexity class.
Cook and Levin in early 1970’s showed that certain problems in NP were such that

- If any of these problems had a deterministic polynomial-time algorithm, then
- All problems in NP had deterministic polynomial-time algorithms.

Such problems are called **NP-complete** problems.

This is important for a number of reasons:

1. If one is attempting to show that $P \neq NP$, s/he may focus on an NP-complete problem and try to show that it needs more than a polynomial amount of time.
2. If one is attempting to show that $P = NP$, s/he may focus on an NP-complete problem and try to come up with a polynomial time algorithm for it.
3. One may avoid wasting searching for a nonexistent polynomial time algorithm to solve a particular problem, if one can show it reduces to an NP-complete problem (as it is generally believed that $P \neq NP$).
**The Satisfiability Problem**

**Definition – Boolean Variables**

A boolean variable is a variable that can take on values TRUE (1) and FALSE (0).

- We have Boolean operations of AND ($x \land y$), OR ($x \lor y$) and NOT ($\neg x$ or $\overline{x}$) on boolean variables.

<table>
<thead>
<tr>
<th></th>
<th>AND</th>
<th>OR</th>
<th>NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 $\land$ 0 = 0</td>
<td>0 $\lor$ 0 = 0</td>
<td>$\overline{0}$ = 1</td>
<td></td>
</tr>
<tr>
<td>0 $\land$ 1 = 0</td>
<td>0 $\lor$ 1 = 1</td>
<td>$\overline{1}$ = 0</td>
<td></td>
</tr>
<tr>
<td>1 $\land$ 0 = 0</td>
<td>1 $\lor$ 0 = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 $\land$ 1 = 1</td>
<td>1 $\lor$ 1 = 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
THE SATISFIABILITY PROBLEM

**Definition – Boolean Formula**

A **Boolean** formula is an expression involving Boolean variables and operations.

For example: \( \phi = (\overline{x} \land y) \lor (x \land \overline{z}) \lor (y \land z) \) is a Boolean formula.

**Definition – Satisfiability**

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

We say the assignment satisfies \( \phi \).

- What possible assignments satisfy the formula above?

**Definition – The Satisfiability Problem**

The **satisfiability problem** checks if a Boolean formula is satisfiable.

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

Theorem 7.27 – The Cook-Levin Theorem

\[ \text{SAT} \in \text{P iff } \text{P = NP}. \]

Proof

Coming slowly!
**Polynomial Time Reducibility**

**Definition – Polynomial Time Computable Function**

A function $f : \Sigma^* \rightarrow \Sigma^*$ is a polynomial time computable function if some polynomial time TM $M$ exists that halts with $f(w)$ on its tape, when started on any input $w$.

**Definition – Polynomial Time Reducibility**

Language $A$ is polynomial time mapping reducible or polynomial time reducible, to language $B$, notated $A \leq_P B$, if a polynomial time computable function $f : \Sigma^* \rightarrow \Sigma^*$ exists, where for every $w$,

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

The function $f$ is called the polynomial time reduction of $A$ to $B$.

- To test whether $w \in A$ we use the reduction $f$ to map $w$ to $f(w)$ and test whether $f(w) \in B$. 

( Lecture 20)
**THEOREM 7.31**

If $A \leq_P B$ and $B \in P$, then $A \in P$. 

**PROOF**

- It takes polynomial time to reduce $A$ to $B$.
- It takes polynomial time to decide $B$. 

A literal is a Boolean variable or its negated version ($x$ or $\overline{x}$).

A clause is several literals connected with $\lor$ (OR), e.g.,

$$(x_1 \lor \overline{x}_2 \lor x_4).$$

A Boolean formula is in conjunctive normal form (or is a cnf-formula) if it consists of several clauses connected with $\land$ (AND), e.g.

$$(x_1 \lor \overline{x}_2 \lor x_4 \lor x_5) \land (x_2 \lor \overline{x}_3 \lor \overline{x}_4) \land (x_1 \lor x_2 \lor x_3 \lor \overline{x}_5)$$

A cnf-formula is a 3cnf-formula if all clauses have 3 literals, e.g.

$$(x_1 \lor \overline{x}_2 \lor x_4) \land (x_2 \lor \overline{x}_3 \lor \overline{x}_4) \land (x_1 \lor x_3 \lor \overline{x}_5)$$

$3SAT = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable 3cnf-formula} \}$.

In a satisfiable cnf-formula, each clause must contain at least one literal that is assigned 1.
**An Example Reduction: Reducing 3SAT to CLIQUE**

**Theorem 7.32**

3SAT is polynomial time reducible to CLIQUE.

**Proof Idea**

Take any 3SAT formula and polynomial-time reduce it to a graph such that if the graph has a clique then the 3cnf-formula is satisfiable.

- Some details:
  - $\phi$ is a formula with $k$ clauses each with 3 literals.
  - The $k$ clauses in $\phi$ map to $k$ groups of 3 nodes each called a **triple**.
  - Each node in the triple corresponds to one of the literals in the corresponding clause.
  - No edges between the nodes in a triple.
  - No edges between “conflicting” nodes (e.g., $x$ and $\overline{x}$)
An Example Reduction: Reducing $3SAT$ to $CLIQUE$

$$\phi = (x_1 \lor x_1 \lor x_2) \land (\overline{x_1} \lor \overline{x_2} \lor \overline{x_2}) \land (\overline{x_1} \lor x_2 \lor x_2)$$
A **N** **E** **X** **A** **M** **P** **L** **E** **R** **E** **D** **U** **C** **T** **I** **O** **N: ** **R** **E** **D** **U** **C** **I** **N** **G** **3** **S** **A** **T** **T** **O** **C** **L** **I** **Q** **U** **E**

\[ \phi = (x_1 \lor x_1 \lor x_2) \land (\overline{x_1} \lor \overline{x_2} \lor \overline{x_2}) \land (\overline{x_1} \lor x_2 \lor x_2) \]

- If \( \phi \) has a satisfying assignment, then at least one literal in each clause needs to be 1.
- We select the corresponding nodes in the corresponding triples.
- These nodes should form a \( k \)-clique.
- If \( G \) has a \( k \)-clique, then selected nodes give a satisfying assignment to variables.
**NP-Completeness**

**Definition – NP-Completeness**

A language \( B \) is **NP-complete** if it satisfies two conditions:

1. \( B \) is in \( NP \), and
2. Every \( A \) in \( NP \) is polynomial time reducible to \( B \).

**Theorem**

If \( B \) is NP-complete and \( B \in P \), then \( P = NP \). (Obvious)

**Theorem**

If \( B \) is NP-complete and \( B \leq_P C \) for \( C \) in \( NP \), then \( C \) is NP-complete.

**Proof**

All \( A \leq_P B \) and \( B \leq_P C \) thus all \( A \leq_P C \).
**Theorem**

SAT is NP-Complete.

**Proof Idea**

- Showing SAT is in NP is easy.
  - Nondeterministically guess the assignments to variables and accept if the assignments satisfy $\phi$.
- We can encode the accepting computation history of a polynomial time NTM for every problem in NP as a SAT formula $\phi$.
- Thus every language $A \in$ NP is polynomial-time reducible to SAT.
  - $N$ is a NTM that can decide $A$ in time $O(n^k)$
  - $N$ accepts $w$ if and only if $\phi$ is satisfiable.
Bird’s eye view of a polynomial time computation branch

All legal windows can be enumerated.

cell[i,j] … i’th configuration, j’th tape cell
We represent the computation of a NTM $N$ on $w$ with a $n^k \times n^k$ table, called a **tableau**.

- Rows represent configurations
- First row is the start configuration ($w +$ lots of blanks to fill the remaining of the $n^k$ cells.)
- Each row follows from the previous one using $N$’s transition function.

A tableau is **accepting** if any row of the tableau is an accepting configuration.

Every accepting tableau for $N$ on $w$ corresponds to an accepting computation branch of $N$ on $w$.

If $N$ accepts $w$, then an accepting tableau exists!
Setting up formula $\phi$

**The Variables**

- Let $C = Q \cup \Gamma \cup \{\#\}$.
- For $1 \leq i, j \leq n^k$ and for each $s \in C$, we have a variable $x_{i,j,s}$.
- $x_{i,j,s} = 1$ if the cell $[i, j]$ contains the symbol $s$.
- Note that the number of variables is polynomial function of $n$.

**The Formula $\phi$**

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

- $\phi_{\text{cell}}$ makes sure that there is only one symbol in every cell!
- $\phi_{\text{start}}$ makes sure the start configuration is correct.
- $\phi_{\text{accept}}$ makes sure the accept state occurs somewhere.
- $\phi_{\text{move}}$ makes sure configurations follow each other legally.
For all $i$ and $j$, if $cell[i, j]$ contains symbol $s$, (that is $x_{i,j,s} = 1$), it can not contain another symbol (that is, no other variable with the same $i$ and $j$, but a different symbol, is 1).

$$\phi_{cell} = \bigwedge_{1 \leq i,j \leq n^k} \left[ \left( \bigvee_{s \in C} x_{i,j,s} \right) \land \left( \bigwedge_{s,t \in C} (\overline{x_{i,j,s}} \lor \overline{x_{i,j,t}}) \right) \right]$$
For all $i$ and $j$, if $\text{cell}[i, j]$ contains symbol $s$, (that is $x_{i,j,s} = 1$), it can not contain another symbol (that is, no other variable with the same $i$ and $j$, but a different symbol, is 1).

$$\phi_{\text{cell}} = \bigwedge_{1 \leq i, j \leq n^k} \left[ \left( \bigvee_{s \in C} x_{i,j,s} \right) \land \left( \bigwedge_{s, t \in C, s \neq t} \left( \overline{x_{i,j,s}} \lor x_{i,j,t} \right) \right) \right]$$
For all $i$ and $j$, if $cell[i, j]$ contains symbol $s$, (that is $x_{i,j,s} = 1$), it can not contain another symbol (that is, no other variable with the same $i$ and $j$, but a different symbol, is 1).

\[
\phi_{cell} = \bigwedge_{1 \leq i, j \leq n^k} \left[ \bigvee_{s \in C} x_{i,j,s} \right] \wedge \left( \bigwedge_{s, t \in C, s \neq t} (\overline{x_{i,j,s}} \vee x_{i,j,t}) \right)
\]
For all \( i \) and \( j \), if \( cell[i, j] \) contains symbol \( s \), (that is \( x_{i,j,s} = 1 \)), it can not contain another symbol (that is, no other variable with the same \( i \) and \( j \), but a different symbol, is 1).

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

Note that \( \phi_{cell} \) is in a conjuctive normal form.
\( \phi_{\text{start}} \) sets up the first configuration.

\[
\phi_{\text{start}} = x_{1,1},\# \land x_{1,2},q_0 \land x_{1,3},w_1 \land x_{1,4},w_2 \land \cdots \land x_{1,n+2},w_n \land \\
x_{1,n+3,\square} \land \cdots \land x_{1,n^k-1,\square} \land x_{1,n^k},\#
\]
\( \phi_{\text{start}} \) sets up the first configuration.

\[
\phi_{\text{start}} = \begin{array}{c}
q_0 \text{ and input symbols} \\
\begin{array}{c}
x_{1,1}, \# \land x_{1,2}, q_0 \land x_{1,3}, w_1 \land x_{1,4}, w_2 \land \cdots \land x_{1,n+2}, w_n \land \\
x_{1,n+3}, \sqcup \land \cdots \land x_{1,n^k-1}, \sqcup \land x_{1,n^k}, \#
\end{array}
\end{array}
\]

all the blanks to the right
\( \phi_{accept} \) says \( q_{accept} \) occurs somewhere.

\[ \phi_{accept} = \bigvee_{1 \leq i,j \leq n^{k}} x_{i,j,q_{accept}} \]
\( \phi_{move} \) is the most interesting of the subformulas.

How many possible such windows are there?

There are \(|C|^6\) possible such windows.
**DEFINITION – LEGAL WINDOW**

A $2 \times 3$ window is **legal** if that window does not violate the actions specified by $N$’s transition function.

- Suppose $\delta$ of $N$ has the entries
  - $\delta(q_1, a) = \{(q_1, b, R)\}$
  - $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$
- The following windows are legal:

<table>
<thead>
<tr>
<th>a</th>
<th>q₁</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>q₂</td>
<td>a</td>
<td>c</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>b</td>
<td>a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>q₁</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>q₂</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>a</th>
<th>q₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>q₂</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>q₂</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>b</th>
<th>b</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>
**Definition – Legal Window**

A 2 is **legal** if that window does not violate the actions specified by $N$’s transition function.

- Suppose $\delta$ of $N$ has the entries
  - $\delta(q_1, a) = \{(q_1, b, R)\}$
  - $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$
- The following windows are NOT legal:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>$q_1$</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_1$</td>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>b</th>
<th>$q_1$</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_2$</td>
<td>b</td>
<td>$q_2$</td>
</tr>
</tbody>
</table>

**Claim**

If the top row of the table is the start configuration and every window in the tableau is legal, then every row of the table (after the first) is a configuration that follows the preceding one!
Thus

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

Where “ (the (i, j) window is legal) " is actually the following formula

\[ \bigvee_{a_1,a_2,a_3,a_4,a_5,a_6} (x_{i,j-1},a_1 \land x_{i,j},a_2 \land x_{i,j+1},a_3 \land x_{i+1,j-1},a_4 \land x_{i+1,j},a_5 \land x_{i+1,j+1},a_6) \]

is a legal window

- We have \( O(n^{2k}) \) variables (\( = |C| \times n^k \times n^k \))
- The total formula size is \( O(n^{2k}) \), so it is polynomial time reduction.
**Corollary**

3SAT is NP-complete.

- Every formula in the construction of the NP-completeness proof of SAT can actually be written as a conjunctive normal form formula with 3 literals per clause.
  - If a clause has less than 3 literals, repeat one.
  - Disjunctive normal form clauses can be transformed into conjunctive normal form clauses, e.g.,
    \[(a \land b) \lor (c \land d) = (a \lor c) \land (a \lor d) \land (b \lor c) \land (b \lor d)\]
  - Clauses longer than 3 clauses can be rewritten as clauses with 3 variable, e.g.,
    \[(a \lor b \lor c \lor d) = (a \lor b \lor z) \land (\overline{z} \lor c \lor d)\]