Lecture #23: Cook’s Proof

NP-Completeness of Satisfiability

Below we prove that the satisfiability problem for boolean expressions (SAT) is NP-complete by showing that for every NTM $M$, we can prove that $L(M) \leq_P SAT$.

Satisfiability

Let $U = \{u_1, u_2, ..., u_m\}$ be a set of boolean variables. A truth assignment for $U$ is a function $t: U \rightarrow \{T, F\}$. If $u_i$ is a boolean variable in $U$, then $u_i$ and $\neg u_i$ ("not $u_i$") are called literals over $U$. The literal $u_i$ is true if $t(u_i) = T$. $\neg u_i$ is true iff $u_i$ is false.

A clause over $U$ is a set of literals over $U$. It represents the disjunction (the "or") of those literals and is satisfied by a truth assignment iff at least one of its members is true.

A collection $C$ of clauses over $U$ is satisfiable iff there exists some truth assignment for $U$ that simultaneously satisfies all the clauses in $C$ (ie, the "and" of the clauses is true).

Example: Let $c_i$ be the clause $\{u_1, u_2, \neg u_3\}$. $C$ is satisfied if $u_1$ is true or $u_2$ is true or $u_3$ is false. We could rewrite $C$ as $c_i = (u_1 \lor u_2 \lor \neg u_3)$.

Satisfiability (SAT): Given a set $U$ of variables and a collection $C$ of clauses over $U$, is there a satisfying truth assignment for $C$?

Theorem (Cook, 1971). Satisfiability is NP-complete.


1. Show that SAT is in NP. This part is easy. We guess a truth assignment and check to see if it satisfies all the clauses.

2. Show that any language in NP may be reduced to SAT in polynomial time. That is, for every $A$ in NP we need to show that $A \leq_P SAT$.

We will show how, for any NTM $M$, we can prove that $L(M) \leq_P SAT$. This will allow us to speak of generic NTMs instead of specific languages in NP.
Let $M = (Q, \Sigma, \Gamma, \delta, s, t, r)$ be a NTM. Let $p(n) \geq n$ be a bound on the time required by $M$. We build a transformation $\sigma$ which maps from strings in $\Sigma^*$ to instances of SAT such that for all $x \in \Sigma^*$, $x \in L(M)$ iff $\sigma(x)$ is satisfiable.

Constructing $\sigma$: Let $x$ be accepted by $M$. Then there is an accepting computation for $M$ on $x$ such that the number of steps in the checking and guessing stages are bounded by $p(n)$ where $n = |x|$. Thus every tape square involved is numbered from 0 to $p(n)$. We can describe the configuration of our TM at some point in the computation by specifying:

1. the current state $q$
2. the contents of all those tape squares
3. the position of the read/write head

This configuration contains all the information known at that point in the computation.

Let $Q = \{q_0, \ldots, q_r\}$ where $q_0 = s$, $q_1 = t$, and $q_2 = r$. Let $\Gamma = \{s_0, \ldots, s_v\}$ where $s_0 = \uparrow$ and $s_1 = \_$. Our boolean variables are as follows:

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

A computation of $M$ induces a truth assignment on these variables. That is, we let the literal $Q[i, k]$ be true and $\neg Q[i, k]$ be false if at time $i$, $M$ is in state $q_k$. The only hitch is if the program halts at time $m < p(n)$ in which case we need to define our variables for $i > m$. We do this by saying that the computation remains static. That is, if the machine halts at time $m$, then for $i \geq m$ we let $Q[i, k] = Q[m, k]$.

The tape contents at time 0 has the input $x$ in squares 1 through $|x|_1$ on one track and the "guess" on another track.
Notice that we could make a truth assignment that would not correspond to any computation. For instance, for $k_1 \neq k_2$, we could have $Q[i, k_1]$ and $Q[i, k_2]$ both be true, indicating that at time $i$, $M$ was in two different states at the same time.

We now need to devise clauses such that they are satisfiable iff the truth assignment corresponds to a valid accepting computation.

We need to ensure the following:

1. At each time $i$, $M$ is in exactly 1 state.
2. At each time $i$, the read/write head is scanning exactly one tape square.
3. At each time $i$, each tape square contains exactly one symbol.
4. At time 0, the computation is in the initial configuration.
5. By time $p(n)$, $M$ has entered state $t$.
6. For each time $i$, $0 \leq i < p(n)$, the configuration of $M$ at time $i+1$ follows by a single application of the transition function $\delta$ from the configuration at time $i$.

Conditions 1 - 5 are insured by the following clauses:

<table>
<thead>
<tr>
<th>Clause group</th>
<th>Clauses in group</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>{ Q[i,0], Q[i,1], ..., Q[i,r] }, $0 \leq i \leq p(n)$ { ~Q[i, j], ~Q[i, j'] }, $0 \leq j &lt; j' \leq r$</td>
</tr>
<tr>
<td>G2</td>
<td>{ H[i, 0], H[i, 1], ..., H[i, p(n)] }, $0 \leq i \leq p(n)$ { ~H[i, j], ~H[i, j'] }, $0 \leq i \leq p(n), 0 \leq j &lt; j' \leq p(n)$</td>
</tr>
<tr>
<td>G3</td>
<td>{ S[i, j, 0], S[i, j, 1], ..., S[i, j, v] }, $0 \leq i, j \leq p(n)$ { ~S[i, j, k], ~S[i, j, k'] }, $0 \leq i, j \leq p(n), 0 \leq k &lt; k' \leq v$</td>
</tr>
<tr>
<td>G4</td>
<td>{ Q[0, 0] }, { H[0, 0] }, { S[0, 0, 0] }, { S[0, 1, k_1] }, { S[0, 2, k_2] }, ..., { S[0, n, k_n] }, { S[0, n+1, 1] }, { S[0, n+2, 1] }, ..., { S[0, p(n), 1] }, where $x = s_{k_1} s_{k_2} ... s_{k_n}$</td>
</tr>
<tr>
<td>G5</td>
<td>{ Q[p(n), 1] }</td>
</tr>
</tbody>
</table>
The last condition guarantees that the changes from one configuration to the next are in accordance with the transition function $\delta$ for $M$. For each quadruple $(i, j, k, l)$, $0 \leq i, j \leq p(n)$, $0 \leq k \leq r$, and $0 \leq l \leq v$, If $\delta(q_k, s_l) = (q'_k, s'_l, \Delta)$, then the clause group $G_6$ needs to ensure the implications

$$Q[i, k] \land H[i, j] \land S[i, j, l] \Rightarrow Q[i+1, k']$$

$$Q[i, k] \land H[i, j] \land S[i, j, l] \Rightarrow H[i+1, j+\Delta] \quad \text{(where $\Delta \in \{-1, 1\}$)}$$

$$Q[i, k] \land H[i, j] \land S[i, j, l] \Rightarrow S[i+1, j, l']$$

This is done with the following three clauses for each $(i, j, k, l)$:

$G_6$

\[
\begin{align*}
&\{ \neg Q[i, k], \neg H[i, j], \neg S[i, j, l], Q[i+1, k'] \} \\
&\{ \neg Q[i, k], \neg H[i, j], \neg S[i, j, l], H[i+1, j+\Delta] \} \\
&\{ \neg Q[i, k], \neg H[i, j], \neg S[i, j, l], S[i+1, j, l'] \}
\end{align*}
\]

for $0 \leq i, j \leq p(n)$, $0 \leq k \leq r$, and $0 \leq l \leq v$

These six clause groups ensure the conditions that make the boolean expression represent a valid accepting computation. If $x \in L(M)$, then there is an accepting computation of $M$ on $x$ of length $p(n)$ or less, and this computation imposes a truth assignment that satisfies all the clauses in $C = G_1 \cup G_2 \cup G_3 \cup G_4 \cup G_5 \cup G_6$.

Conversely, the construction of boolean formula $C$ is such that any satisfying truth assignment for $C$ must correspond to an accepting computation of $M$ on $x$. It follows that $\sigma(x)$ has a satisfying truth assignment \textit{iff} $x \in L(M)$.

We conclude that for every $A \in \text{NP}$, there is a (polynomial-time) transformation from $A$ to SAT. It follows that SAT is NP-complete.