More on Polynomial Time and Space

20.1 The Original NP-Completeness Proof

A configuration/snapshot of a machine is a representation of its current state (what info would be needed to restart computation at that point).

Cook and Levin both proved the following:

**Thm.** SAT is NP-complete.

**Proof Sketch.** We need to show that any language in NP reduces to SAT. Let $A$ be the language. Say it is decided by nondeterministic TM $N$. We need to create a transducer that:

- on input a string $w$, outputs a boolean formula $\phi_w$ such that $\phi_w$ is satisfiable if and only if $w \in A$.

We use the fact that $w \in A$ if and only if there is a computation of $N$ starting with $w$ ending in an accept state.

We know the TM $N$ runs in polynomial time, say $p(n)$. So there is a bound on how much of the tape that $N$ can ever access. Further, we can tweak a TM so that it never goes off the left end of the input. This means we can align the configurations per row, so that a column represents the same cell of the tape. We call the result a tableau. Note that the first row is the start configuration, where $w$ is on the tape, and the last row must be an accepting configuration. By definition, a tableau exists only when $N$ accepts $w$. That is,

- $w \in A$ if and only if it is possible to fill in the blanks on a $p(n) \times p(n)$ grid to make a tableau for $N$ on $w$.

Note that a tableau is polynomial-size. The heart of the proof is to translate the “fill-in-the-blanks” problem into a satisfiability problem. This gets a bit hairy at times! But hopefully it is clear that the question of the existence of the tableau is a question of the form can we do the fill-in to satisfy all the constraints, which has the same flavor as a satisfiability problem. Note too that efficiency is not relevant; as long as we can do the translation in polynomial-time, we are done.

We will get to boolean variables in a moment. But consider if you had the task of writing a computer program that would take in an alleged tableau and check whether it was valid or not. Your code would probably: first check the input had the right dimensions, that the first line was the appropriate one, and the final line corresponded to an accepting state. Then the daunting part is to check the rest of the input. But actually, it is not that hard. First note that we only need to:
check that each line is a legal successor of the previous line.

Furthermore, most of the line will be identical to the previous line: indeed at most one cell can have changed, which must be where the head was in the previous line, and the state updated correctly.

Now, to translate all this into a boolean formula: we need to keep track of where the head is. For this we add a marker in the actual cell. We also need to keep track of the state. One idea is to add that to the cell where the head is, or like we did with computation string, artificially add a new cell just before the head position. But it’s okay to have a separate variable for each row recording the state.

Define a window-record as the contents of 3 consecutive columns from 2 consecutive rows along with the states from these rows. The key point to allow to conversion to a formula is:

to check that row $i+1$ follows from row $i$, it is sufficient to check that each window-record is correct.

Specifically, consider the three window-records that contain the old head position. Then the chars in row $i$ and the current state determine the chars in row $i+1$ and the new state. For all window-records not containing the original head, the middle column must be the same in both rows.

That is, we can pre-compute a (finite) list $W$ of $2 \times 3$ arrays such that all window-records in all tableaux are on the list $W$, and if all window-records in the input are in $W$, then the input must be a valid computation sequence.

Finally, we need to convert this to a boolean formula. To represent the contents in row $i$ column $j$, we simply introduce variables $x_{i,j,s}$ for each char $s$ in the alphabet. We also have $h_{i,j}$ to indicate whether the head is present or not. And variables $S_{i,m}$ for whether the state in row $i$ is state $m$.

Then the boolean formula $\phi_w$ has four parts:

- ensure that exactly one char per cell: add a formula for each $i,j$ ensuring that exactly one of the $x_{i,j,s}$ is true;
- ensure that exactly one state per configuration
- ensure that the start configuration is correct: add a formula checking that the first row is given by $w$ followed by blanks, that the initial state is the start state, and the head is in the start positions;
- ensure that the final configuration is accepting: add a formula so that the final state is an accept state
- ensure that every $2 \times 3$ window-record is permissible.

The end result is that we have automatically constructed a $\phi_w$ such that it is satisfiable if and only if $w \in A$. That is, we have reduced $A$ to SAT.
20.2 Nondeterministic Space Complexity

Recall that the space complexity of a TM is the maximum number of cells scanned as a function of the input length $n$. \( \text{SPACE}(f(n)) \) is the set of all languages decidable in $O(f(n))$ space. \( \text{NSPACE} \) is defined similarly.

Savitch’s theorem shows that in dealing with space, to get rid of nondeterminism it is sufficient to square the original space.

**Thm.** Every nondeterministic $f(n)$-space TM has an equivalent $O(f^2(n))$-space deterministic TM provided $f(n) \geq n$.

**Proof Sketch.** Let $N$ be the machine. The running time of the machine $N$ is at most $B = 2^{O(f(n))}$, since if it were to run for longer than this it would repeat a configuration and therefore be in an infinite loop.

Consider the start configuration $c_s$ and the end configuration $c_e$. The start configuration is determined by the input, say $w$. To make life easier one can reprogram any machine to erase the tape when it reaches an accept state, and then go to a final accept state, so that the end configuration is unique. Here is a recursive algorithm to check if there is an accepting computation for $N$ on input $w$.

*For all possible half-way points $c_m$, check to see whether there is a computation leading from $c_s$ to $c_m$ in at most $B/2$ steps, and then check whether there is a computation from $c_m$ to $c_e$ in at most $B/2$ steps.*

Each level of the recursion has iteration over all configurations and for each configuration, two calls to function on half the size. (Though note that only one call is active at any stage.) Depth of the recursion is $\log B$; space needed at each level (on the machine stack) is $O(f(n))$. ♦

**Example:** Consider an NFA with $q$ states. One can simulate an NFA in $O(q)$ space, by keeping track of the set of states the automaton can be in at each point. The language $\overline{\text{ALL}_{NFA}}$ is the set of NFAs whose language is not everything. The shortest string not accepted by an NFA, if it exists, has length at most $2^q$ (why?). So we can decide this language by nondeterministically guessing the non-accepted string one char at a time: we use a counter that goes up to $2^q$. 

---

3
20.3 \( \mathsf{PSPACE}\)-Completeness

Recall that \( \mathsf{PSPACE} \) is the set of all languages decidable in polynomial space. That is: \( \bigcup_k \mathsf{SPACE}(n^k) \). \( \mathsf{NPSPACE} \) is the nondeterministic equivalent. But by Savitch above, since the square of a polynomial is a polynomial, \( \mathsf{PSPACE} = \mathsf{NPSPACE} \). Using earlier results we get the chain:

\[
P \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} = \mathsf{NPSPACE} \subseteq \mathsf{EXPTIME}
\]

Here is the expected definition: A language \( B \) is \( \mathsf{PSPACE}\)-complete if

(a) \( B \in \mathsf{PSPACE} \);

(b) for all \( A \in \mathsf{PSPACE} \) we have that \( A \leq_P B \).

\( \mathsf{TQBF} \) is the set of all true quantified boolean formula. For example, \( \forall x \exists y (x \lor y) \land (\bar{x} \lor \bar{y}) \) is true: for each \( x \) choose \( y \) to be the opposite.

**Thm.** \( \mathsf{TQBF} \) is decidable in \( O(n) \) space.

**Proof Sketch.** Calculate the value of the QBF recursively: strip off one variable at a time, substitute the two values for the var, and call the procedure recursively on the two resultant QBFs. The depth of the recursion is the number of variables. ♦

**Thm.** \( \mathsf{TQBF} \) is \( \mathsf{PSPACE}\)-complete.

**Proof Sketch.** Use a tableau again. Say the machine \( N \) for language \( A \in \mathsf{PSPACE} \) runs in space \( n^k \). Then its running time is at most \( B = c^{n^k} \), where \( c \) is the size of the tape alphabet.

Now the tableau is too big! So we use the halving idea from Savitch’s theorem:

There is a valid tableau if and only if there exists a middle configuration such that both halves are valid tableaux.

And repeat. But to save space in the formula, we have to “fold” the tableau in half such that we re-use the formula for the two halves.

Specifically, design formula \( \phi_{c_1,c_2,t} \) to be true if it is possible to get from configuration \( c_1 \) to configuration \( c_2 \) in time \( t \). Then

\[
\phi_{c_1,c_2,t} = \exists m \forall (c_3,c_4) \in \{(c_1,m_1),(m_1,c_2)\} \left[ \phi_{c_3,c_4,\frac{t}{2}} \right]
\]

Need to go \( \log B \) levels. ♦

Other \( \mathsf{PSPACE}\)-complete problems include whether the first player has a win on an arbitrary version of a game such as generalized geography.

It is known that \( \mathsf{SPACE}(n) \) is strictly contained in \( \mathsf{SPACE}(n^2) \). More strange is that it is known that \( \mathsf{SPACE}(n) \) is not equal to \( \mathsf{P} \), but it is not known whether one contains the other.
Exercises

20.1. Show that if $P = NP$, then we can factor in polynomial time.

20.2. True or false or unknown:
   - $TQBF$ is $NP$-complete
   - $PATH$ is not $PSPACE$-complete

20.3. Show that $PSPACE$ is closed under union and complementation.

20.4. Show that the reduction in the Cook–Levin theorem fails if we use only $2 \times 2$ windows.

20.5. Show that one can determine in polynomial space whether $R$ and $S$ are equivalent regular expressions.

20.6. Show that the acceptance problem for LBA (linearly bounded automata) is $PSPACE$-complete.