The Cook-Levin Theorem: SAT is NP-Complete

Omri Weinstein

1 Reduction

Theorem 1 (Cook-Levin 1971)**.** *SAT is NP-Complete.*

Proof. By definition of the NP-complete class, one must show that SAT is in NP as well as that it is NP-Hard. The former part is immediate. The latter one is a little complicated technically, but as we shall see, the concept of the proof is actually quite simple.

Claim 1.1. *$SAT \in NP$*

Proof: Our witness would be a satisfying assignment $x_1, x_2, ..., x_n$ s.t $x_i \in T/F$. A deterministic TM can easily verify that the assignment satisfies all clauses, in polynomial (even linear) time in n, m. The statement follows by equivalence of definitions to the NP class. □

Lemma 1.2. *$SAT \in$ NP-Hard*

Proof Idea: We must show that any language A in NP is efficiently reducible to SAT. The reduction function, given the NP-machine M_A for A, receives a string w and produces a Boolean formula $\phi_{M_A,w}$ that simulates the run of M_A on the input w (Note that since A is in NP, there exists such a TM). An assignment to $\phi_{M_A,w}$ may represent a computational path of the NP machine, so that $\phi_{M_A,w}$ will be satisfiable iff the machine M_A accepts w.

Note that AND, OR and NOT form a universal system, i.e, a basis for Boolean logical operators, thus it is intuitive that we can construct a proper CNF formula that simulates the transitions of a Turing Machine. The concept of the reduction is simple, it is only technically complicated because it involves many details.

Proof: Let M be a non-deterministic TM that decides A in time cn^k. Note that we use M in our reduction although we do not describe it. That's OK, we can refer to M as given, since we know it must exist for every language in NP.

We define a **tableau** for M to be a $(cn^k) \times (cn^k)$ table whose rows are the consecutive configurations of a computational path of M on input w:

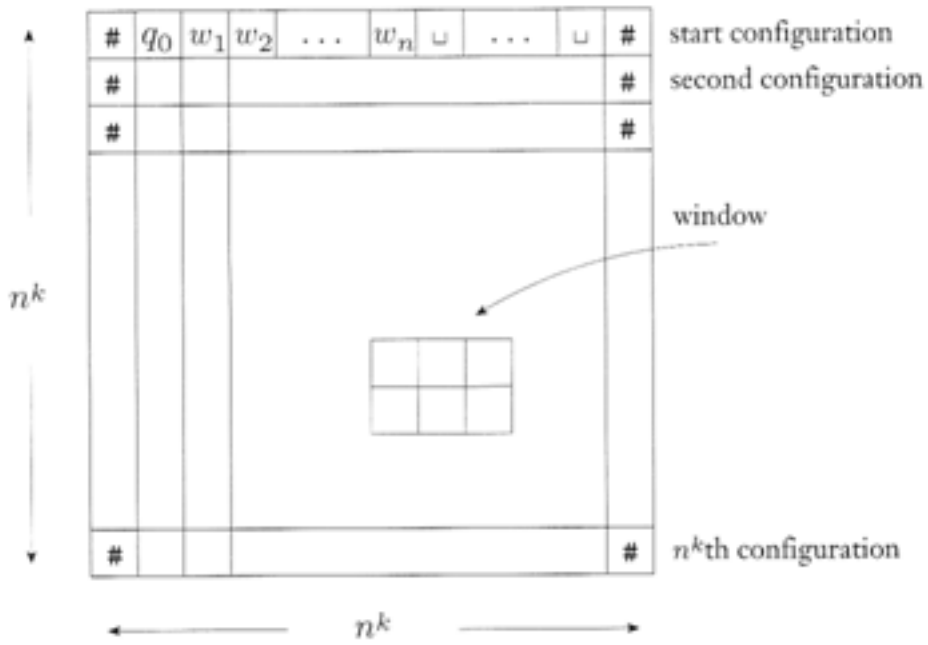

Figure 1: A tableau

The first row of the tableau is the starting configuration of M, and each row should follow the previous by M's transition function δ . A tableau is **accepting** if any of its rows is in an accepting state of M. There is a $1-1$ correspondence between an accepting tableau and an accepting computation, so the problem of determining whether M accepts w is equivalent to the problem of determining whether there exists an accepting tableau for M on w. Note that since M is non-deterministic, there might be many accepting computations, just as there might be many satisfying assignments to a CNF formula. As a convention, each line in the table will start and end with a $\sharp$ symbol, and the state, in which the machine M is in, is described by a symbol on the column immediately to the left of the cell the head is at.

Let's start by describing $\phi = \phi_{M,w}$'s variables: if Q and Γ are M's state-set and alphabet, let $G = Q \cup \Gamma \cup \{\sharp\}$, and for each i and j between 1 and cn^k [1] and $s \in G$ we will have one Boolean variable $x_{i,j,s}$ of ϕ (note that $|G|$ is constant). Each of the (cn^{2k}) entries $[i,j]$ of the tableau is called a **cell**, and its content is represented via ϕ's variables: $x_{i,j,s} = 1$ iff cell $[i,j]$ has the value s. A valid accepting tableau must satisfy the following conditions:

1. Is each cell in the tableau legal (i.e., contains only one value type:

   a state, a letter or a-$\sharp$)?

[1] The number of columns is in fact $cn^k + 3$ to accommodate for the two $\sharp$ symbols and the state in which M is at. Let us therefore assume M halts with $cn^k - 3$ steps, which would make the numbers OK.

2. Is the starting configuration legal (with respect to w)?

3. Are the transitions between consecutive rows (configurations) legal?

4. Is there an accepting row (configuration) ?

We require all of the 4 parts to be true, so ϕ has the following structure:

$$\phi_{cell} \wedge \phi_{start} \wedge \phi_{accept} \wedge \phi_{move}$$

where each Boolean formula guarantees the corresponding condition. We describe each one in turn:
The first part in showing correspondence between an assignment and a tableau is to ensure that the assignment turns on exactly one value from G in each cell (i.e, that each cell represents either a letter, a state or start/end of row, but only one single type). This is ensured by the following Boolean formula:

$$\phi_{cell} = \bigwedge_{i \leq i,j \leq cn^k} \left[\left(\bigvee_{s \in G} x_{i,j,s} \right) \wedge \left(\bigwedge_{s,t \in G} \overline{x_{i,j,s}} \vee \overline{x_{i,j,t}} \right) \right]$$

The outer AND runs on all cells, and the inner fragments ensure that at least 1 variable is turned on, and that no more than 1 is on (out of each pair, at least 1 variable is turned off). Thus, any satisfying assignment to ϕ specifies exactly 1 symbol in each cell.

Formula ϕ_{start} asserts that the initial configuration is legal (with respect to w):

$$\phi_{start} = x_{1,1,\sharp} \wedge x_{1,2,q_0} \wedge x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,cn^k-1,\sqcup} \wedge x_{1,n+3,\sqcup} \wedge x_{1,cn^k,\sharp}$$

Formula ϕ_{accept} simply asserts that an accepting configuration occurs in the tableau:

$$\phi_{accept} = \bigvee_{i \leq i,j \leq cn^k} x_{i,j,q_{accept}}$$

Formula ϕ_{move} makes sure each configuration results from the previous one legally, according to M's transition function δ. It does so by ensuring that each **window** of cells is legal. A window $\Delta \subseteq G^6$ is a 2×3 cell block of the tableau. Note that since M is allowed to move at most one step in each transition, then validating each cell is local (no need to look further than 1 step to the right and to the left). We say a window is **legal** if it follows the rules dictated by δ. For example, let's assume δ contains the following transitions upon reading letters a, b (Remember: M is non-deterministic):

$$\delta(q_1, a) = \{(q_1, b, R)\}, \quad \delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$$

Then the following windows are legal:

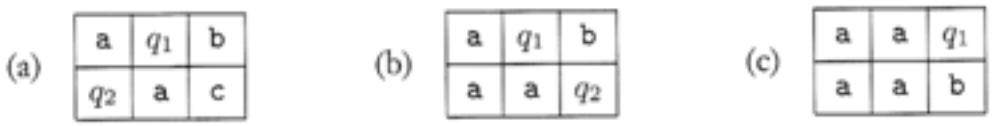

Figure 2: The state q_i appears to the left of the cell to which it currently points.

For example, windows (a) and (b) above are legal because δ allows M to move and write as shown above. Note that we only disqualify windows that violate M's rules. In window (c), for example, we don't know what symbol the head is pointing to; We cannot declare it as illegal since it might be consistent with M's rules (if the head is pointing to the symbol a, then window c complies with δ).

Proposition 1.3. *If the top row of the table (start configuration) is legal (with respect to w) and every window is legal as well, then each row in the table is a configuration of M that legally follows from the previous one.*

Proof: Assume by way of contradiction that the start configuration and every window are legal, and let r_i be the first row (configuration) that doesn't follow from the previous one. In particular, r_{i-1} is a legal row. Let j be the column index of the state symbol in row r_{i-1} (it must appear exactly once since this row is legal). Without loss of generality, we may assume that cells 1 to $j-2$ and $j+2$ to cn^k in rows r_{i-1} and r_i contain the same exact symbols respectively, since otherwise this will imply an illegal window contradicting our assumption (As the state symbol will not appear in the window, M cannot change any of these symbols in a single step and we will discover an illegal window). Thus, the error must be in cells $j-1$ to $j+1$ in row r_i. But if these 3 positions are updated incorrectly, then the window that contains cell $[i,j]$ in the top middle spot will be illegal, since the windows bottom row will not follow M's rules, again contradicting our assumption that all windows are legal. □

Now we turn to formally describe ϕ_{move}. Each window contains 6 cells that can be set only in a **constant** number of ways (since the largest set of non-deterministic transitions in δ is bounded). ϕ_{move} says that for each window in the table, the setting of each of these cells must be legal:

$$\bigwedge_{1<i\leq cn^k, 1<j<cn^k} (\text{window } [i,j] \text{ is legal})$$

where the predicate "window $[i,j]$ is legal" is replaced with the following Boolean formula (the contents of 6 cells that comprise a window are written as $\Delta = a_1, ..., a_6$):

$$\bigvee_{\Delta \text{ is a legal window}} (x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6})$$

where the OR operator goes over all legal combinations $a_1, ..., a_6$ of a legal window (constant number!). We could give a formal description of these contents in terms of the transition function, the states set and the alphabet, but this would be extremely technical deviate us from the main idea of the proof.

2 complexity

We turn to analyze the complexity of the reduction, showing it indeed operates in polynomial time. Clearly, the running time is proportional to the size of ϕ. First of all, the total number of variables ϕ uses is equal to the number of combinations $x_{i,j,s}$ can have. There are $(cn^k)^2$ cells (i and j both range from 1 to cn^k), and s has $|G|$ elements in it. As we mentioned before $|G|$ is constant (since $|Q|$ and $|\Gamma|$ are constants). Thus, there are $O(n^{2k})$ variables, a polynomial number in n. We analyze the size of each of the 4 sub-formulas in ϕ. ϕ_{cell} contains $|G|$ variables per cell, so its size is $O(n^{2k})$. ϕ_{start} contains $|G|$ variables but only for the first row, so its size is $O(n^k)$. As we already mentioned, ϕ_{move} uses a constant number of variables per window, and the number of windows is at most the number of cells in the tableau, since each cell can appear in at most 6 different windows. Thus, it also has size of $O(n^{2k})$. Finally, ϕ_{start} has size $O(n^{2k})$ since it has one variable per cell in the tableau.

Remark: To be precise, if we are interested in the reduction's output size (i.e., count the number of symbols in the output), then writing the variables‘ names in the formula adds a factor of $O(logn)$, since the variables have indices that can range from 1 to cn^k. If we're just interested in the size of the Boolean formula, then we don't consider the indexes since it is measured only by the number of variables and clauses. In any case, this doesn't change the polynomiality of the result.

3 3-Sat is NP-Complete

The above reduction already produces a Boolean formula that is almost in Conjunctive Normal Form (CNF). All but ϕ_{move} are in CNF (though the number of variables in each clause varies). The distributive laws (see Sipser, chapter 0) allow us to replace an OR of AND's with an AND of OR's. This replacement may significantly increase the size of each subformula (describing the contents of a window), but the important thing is that this number is constant, since each subformula depends only on M. The resulting manipulation is a proper CNF formula. In your previous course in Computational models you've probably

seen how to reduce it to a 3-SAT formula, namely, one that contains exactly 3 literals per clause. For those of you who haven't, a short explanation can be found in Sipsers book, at the bottom of p.286.