CS 121, Fall 1987
Lecture notes for 12/11, 12/14 by Marios Mavronicolas

THEOREM: The problem of determining whether a Boolean expression is satisfiable is NP-complete. (Cook, 1971)

PROOF:

First, recall the following about Boolean expressions. A Boolean expression is an expression composed of variables, parentheses, and the operators $\wedge$ (logical AND), $\vee$ (logical OR) and $\neg$ (negation). The precedence of these operators is $\neg$ (highest), then $\wedge$, then $\vee$. Variables take on values 0 (false), or 1 (true); so do expressions. If E_1 and E_2 are Boolean expressions, then the value of $E_1 \wedge E_2$ is 1 if both E_1 and E_2 have value 1, and 0 otherwise. The value of $E_1 \vee E_2$ if either E_1 or E_2 has value 1, and 0 otherwise. The value of $\neg E_1$ is 1 if E_1 is 0 and 0 if E_1 is 1. A Boolean expression is said to be satisfiable if there exists some assignments of 0's and 1's to the variables that gives the expression the value 1. The satisfiability problem is to determine, given a Boolean expression, whether it is satisfiable or not.

To show that the satisfiability problem is NP-complete, we need to show that it is in NP and that every language L in NP is polynomially transformable to the satisfiability problem.

The easy part of the proof is to show that the satisfiability problem is in NP. To determine if a Boolean expression of length n is satisfiable, nondeterministically guess values for all the variables and then evaluate the expres-

sion to verify that it has the value 1. The evaluation can be done in time proportional to its length by a number of parsing algorithms. Thus, the satisfiability problem is in NP.

To show that every language L in NP is polynomially transformable to the satisfiability problem, we consider a nondeterministic Turing machine M of poly-time complexity that accepts L, and let w be an input to M. From M and w we can construct a Boolean expression w_0 such that w_0 is satisfiable if and only if M accepts w. The crux of the proof is in showing that for each M there is a poly-time-bounded algorithm to construct the Boolean expression w_0 from w. The polynomial depends on the machine M.

We can assume (without loss of generality) that M has a single tape. Suppose M has states $q_1, q_2, \ldots, q_R$ and tape symbols $X_1, X_2, \ldots, X_m$. Let p(n) be the time complexity of M. (q_1: start state, q_R: final state, X_1: blank symbol)

We shall construct a Boolean expression w_0 that "simulates" a sequence of ID's entered by M. Each assignment of 1 and 0 to the variables of w_0 represents at most one sequence of ID's of M, possibly not a legal one. The Boolean expression w_0 will take on the value 1 iff the assignment to the variables represents a sequence of ID's corresponding to an accepting computation.

We give next the propositional variables in w_0 with their intended meaning:

1) $C\langle i,j,t\rangle$ is 1 iff i^{th} cell on M's input tape contains x_j at time t, $1 \leq i \leq p(n)$, $1 \leq j \leq m$, $0 \leq t \leq p(n)$

2) $S\langle k,t\rangle$ is 1 iff M is in state q_k at time t, $1 \leq k \leq R$, $0 \leq t \leq p(n)$

3) $H\langle i,t\rangle$ is 1 iff the tape head is scanning i^{th} cell at time t, $1 \leq i \leq p(n)$, $0 \leq t \leq p(n)$.

We see that there are $O(p^2(n))$ propositional variables which can be represented by binary numbers with $c \log n$ bits for some constant c depending on p.

Consider an input w, $|w| = n$. We can assume that M always uses exactly $p(n)$ time and each ID of M is exactly $p(n)$ long (i.e. we can always "pad" the computation and ID's to force this). Hence, every accepting computation of M on input w can be represented by a sequence of ID's: $Q_0, Q_1, \ldots, Q_{p(n)}$. But, asserting that $Q_0, Q_1, \ldots, Q_{p(n)}$ is an accepting sequence of ID's is tantamount to asserting:

A. Head is scanning exactly one tape cell in each ID

B. Each ID has exactly one tape symbol in each cell.

C. Each ID has exactly one state.

D. At most one tape cell, the one scanned by the head, is modified from one ID to the next.

E. Change in state, head position, and tape cells between successive cells is allowed by M.

F. First ID is initial.

G. Last ID contains final state.

We will construct Boolean expressions A-G to mirror the above statements. In our construction we shall make use

of the predicate $U(x_1, x_2, \ldots, x_r)$ thas has the value 1 when exactly one of the variables $x_1, x_2, \ldots, x_r$ has value 1. U can be expressed as a Boolean expression of the form :

$$U(x_1, x_2, \ldots, x_r) = (x_1 + x_2 + \ldots + x_r) \cdot \left(\prod_{\substack{i,j \\ i \neq j}} (\neg x_i + \neg x_j) \right)$$

The first factor in U asserts that at least one of the x_i's is true. The remaining $r(r-1)/2$ factors assert that no two x_i's are true. Note that U is of length $O(r^2)$.

We now go on to our construction :

1. $A = \prod_{0 \leq t \leq p(n)} A_t$, where : $A_t = U(H\langle 1,t\rangle, \ldots, H\langle p(n),t\rangle)$

(A_t simply asserts that at time t exactly one cell is scanned. A asserts that this happens at all the times t, $0 \leq t \leq p(n)$)

Note that A has length $O(p^3(n))$.

2. $B = \prod_{i,t} B_{it}$, where : $B_{it} = U(C\langle i,1,t\rangle, \ldots, C\langle i,m,t\rangle)$

(B_{it} asserts that i^{th} cell contains exactly one symbol at time t. Note that B_{it} has constant size (independent of n, since m, the size of the tape alphabet, depends only on the TM M.)

Note that B has length $O(p^2(n))$.

3. $C = \prod_{0 \leq t \leq p(n)} U(S\langle 1,t\rangle, \ldots, S\langle R,t\rangle)$

C has length $O(p(n))$

4. $D = \prod_{i,j,t} (H\langle i,t\rangle + (C\langle i,j,t\rangle \overset{*}{\equiv} C\langle i,j,t+1\rangle))$

* $x \equiv y$ (x iff y) : it takes the value 1 iff x and y are either both 1 or both 0

D states that for all i,j,t either head is scanning cell i at time t or the j^{th} symbol is in cell i at time $t+1$ iff it was there at time t.

D has length $O(p^2(n))$, since j ranges over the alphabet size which is constant.

5. $E = \prod_{i,j,K,t} E_{ijKt}$, where:

$$E_{ijKt} = \overline{C\langle i,j,t\rangle} + \overline{H\langle i,t\rangle} + \overline{S\langle K,t\rangle} + \sum_{\ell}(C\langle i,j_\ell,t+1\rangle \cdot S\langle K_\ell,t+1\rangle \cdot H\langle i_\ell,t+1\rangle)$$

(Why?)

E has size $O(p^2(n))$ since j and K range over alphabet and state sizes, which are constants.

6. $F = S\langle 1,0\rangle H\langle 1,0\rangle \prod_{1\le i\le n} C\langle i,j_i,0\rangle \prod_{n<i\le p(n)} (i,1,0)$

$S\langle 1,0\rangle H\langle 1,0\rangle$: states that M starts in start state with head at left

$\prod_{1\le i\le n} C\langle i,j_i,0\rangle$: first n cells contain w initially

$\prod_{n<i\le p(n)} C\langle i,1,0\rangle$: remaining cells contain black symbol initially. F has length $O(p(n))$.

7. $G = S\langle R, p(n)\rangle$

The Boolean expression W_0 is the product ABCDEFG. Since formulas for A-G have at most $O(p^3(n))$ symbols, formula for W_0 has $O(p^3(n))$ symbols. Since we have been counting propositional variables as single symbols, when in fact they are represented by strings of length $O(\log n)$,

we must have a bound on $|w_0|$ of $O(p^3(n)\log n) \leq$ $\leq cnp^3(n)$, which is a polynomial function of the length of w. Thus, given w and p, w_0 can be constructed in time proportional to its length.

By our construction, given an accepting sequence of ID's we can always find an assignment of 0's and 1's to the propositional variables that will make w_0 have the value true. Conversely, given an assignment of values to these variables which make w_0 true, we can easily find an accepting sequence of ID's. Thus, w_0 is satisfiable iff M accepts w.

This completes our proof.

Def. A Boolean expression is said to be in conjunctive normal form (CNF) if it is of the form $E_1 \wedge E_2 \wedge \dots \wedge E_K$ and each E_i, called a clause, is of the form $\alpha_{i_1} \vee \alpha_{i_2} \vee \dots \vee \alpha_{i_{r_i}}$ where each α_{j_k} is a literal (either x or $\neg x$ for some variable x). We usually write $\bar{x}$ instead of $\neg x$.

For example, $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_3 \vee \bar{x}_4) \wedge \bar{x}_3$ is in CNF.

The expression is said to be in 3-CNF if each clause has exactly three distinct literals.

Corollary: The satisfiability problem for Boolean expressions in CNF is NP-complete.

Proof.

It suffices to show that each of the expressions $A, B, \dots, G$ is either already in CNF or can be manipulated

into a CNF form, (by laws of Boolean algebra), without increasing the length of the expression by more than a constant factor.

We observe that U is, by its definition, in CNF. It follows that A, B, C are in CNF form. and that F and G are trivially in CNF form, since they are products of single literals.

D is the product of expressions of the form $(x \equiv y) + z$.

But : $(x \equiv y) + z = xy + \bar{x}\bar{y} + z = (x + \bar{y} + z) \cdot (\bar{x} + y + z)$.

This proves that D can be also converted to a CNF form at most twice as long the original.

Finally, for E, since size of $E_{i,jkt}$ is constant, we can convert E_{ijkt} to CNF with length independent of n by making truth table and reading off terms.

This completes the proof of corollary.