CSE491/596 Lecture Wed. 10/28: Cook-Levin Theorem

The reduction goes not only to SAT but to a highly restricted subcase of SAT:

Definition. A Boolean formula is in **conjunctive normal form** (CNF) if it is a conjunction of **clauses**

$$\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m,$$

where each clause C_j is a disjunction of **literals** x_i or $\bar{x}_i$. The formula is in k**-CNF** if each clause has at most k distinct literals, *strictly* so if each has exactly k.

3SAT
Instance: A Boolean formula $\phi(x_1, \ldots, x_n) = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ in 3CNF.
Question: Is there an assignment $\vec{a} = a_1 a_2 \cdots a_n \in \{0,1\}^n$ such that $\phi(a_1, \ldots, a_n) = 1$?

Theorem [Cook 1971, Levin 1971--73]: **3SAT** is **NP**-complete under $\leq_m^p$, where the reduction function also yields an efficient 1-to-1 correspondence between satisfying assignments and witnesses for the source problem.

Historical notes: Cook only stated an oracle reduction but his proof implicitly gave a mapping reduction, and the followup paper by Richard Karp in 1972 made $\leq_m^p$ the norm. The added statement about mapping the witnesses too comes from Levin and is one reason people accept that he came up with the theorem independently while working in the Soviet Union even though his paper appeared two years later. Of course **3SAT** $\leq_m^p$ **SAT** by restriction, and Cook actually showed **SAT** $\leq_m^p$ **3SAT** in general. The following proof is by Claus-Peter Schnorr from 1978.

Proof. We have already seen that **SAT** is in NP and verifying **3SAT** is even easier---see notes below. Now let any $A \in$ NP be given. This time we use the "verifier" characterization of NP. We can take a deterministic TM V_R and polynomials p, q such that for all n and x of length n,

$$x \in A \iff (\exists y : |y| = q(n))[V_R \text{ accepts } \langle x, y \rangle]$$

and such that V_R runs in time $p(r)$ where $r = n + q(n)$. Earlier we stated "$|y| \leq q(n)$" as the bound on witnesses, but now we are entitled to "play a trump card" by saying that the encoding scheme used to define $\langle x, y \rangle$ first puts things entirely in binary notation with the y parts padded out to the exact length $q(n)$. Since whatever alphabet A was originally defined over can be binary-encoded with only a constant-factor expansion of length, we can regard the length n as meaning *after* the encoding is applied. Since the reduction function f we are building is given x, its length n is a known quantity, so we can finally specify $\langle x, y \rangle$ as just being the concatenation xy of the binary strings. Then $|\langle x, y \rangle|$ really does equal $n + q(n)$. (We abbreviate $q(n)$ as just q.)

Now we apply Savage's theorem to V_R. For each n, we get a circuit C_n with $n + q$ input gates, the first n for the bits $x_1, \ldots, x_n$ of (the binary encoding of) x, and the others for $y_1, \ldots, y_q$, such that $C_n(xy) = 1 \iff V_R$ accepts $\langle x, y \rangle$. Since NAND is a universal gate, we may suppose every gate in the body of C_n is NAND. Since V_R runs in time $p(r)$, the size of C_n is order-of $p(r)^2 = p(n + q(n))^2$. Moreover, because C_n has such a regular structure, we have:

- the function $f_0(x) = \langle C_{|x|} \rangle$ is computable in $p(n + q(n))^2$ time, which is polynomial in n, and
- C_n itself depends only on $n = |x|$, not on the values of the bits of x.

Now we build a Boolean formula ϕ_n out of C_n. After the above window-dressing, this comes real quick.

We first allocate variables $x_1, \ldots x_n$ and $y_1, \ldots y_q$ to stand for the input gates, so that the positive literal x_i is carried by every *wire* out of the gate x_i, and likewise every wire out of the gate y_j carries y_j. Then we allocate variables $w_0, w_1, \ldots, w_s$ for every other wire in the circuit, where w_0 is the output wire and $s = O\big(p(n+q)^2\big)$ is also proportional to the number of NAND gates g, since every NAND gate has exactly two input wires. Then every evaluation of C_n carries a Boolean value through each wire and so gives a legal assignment to these variables---but not every assignment to the wire variables is a legal evaluation of the circuit. If it is not legal, then it must be inconsistent at some NAND gate. We write ϕ_n to enforce that all gates work correctly.

So consider any NAND gate g in the circuit, calling its input wires u and v, and consider any output wire w (there will generally be more than one of those) from g. Define

$$\phi_g = (u \vee w) \wedge (v \vee w) \wedge (\overline{u} \vee \overline{v} \vee \overline{w}) .$$

Note this is in (non-strict) 3CNF where the literals in each clause have the same sign. The point is that *ϕ_g is satisfied by, and only by, the assignments in $\{0,1\}^3$ that make $w = u$ NAND v*. We can't have u, v, w all be true, and if u or v is false, then w must be true. Thus an assignment to all the variables satisfies ϕ_g if and only if it makes the gate g work correctly for the output wire w. So:

$$\phi_n = \bigwedge_g \phi_g$$

is a (non-strict) 3CNF formula that is satisfied by exactly those assignments that are legal evaluations of C_n. We will finally get the effect of "searching for" a witness y to the particular x by fixing the x_i *variables* to the *values* given by the actual bits of x and mandating that $w_0 = 1$. This is all done by the "singleton clauses" (w_0) and for $1 \leq i \leq n$,

$$\beta_i = (x_i) \text{ if the } i\text{-th bit of } x \text{ is } 1, \text{ else } \beta_i = (\overline{x}_i).$$

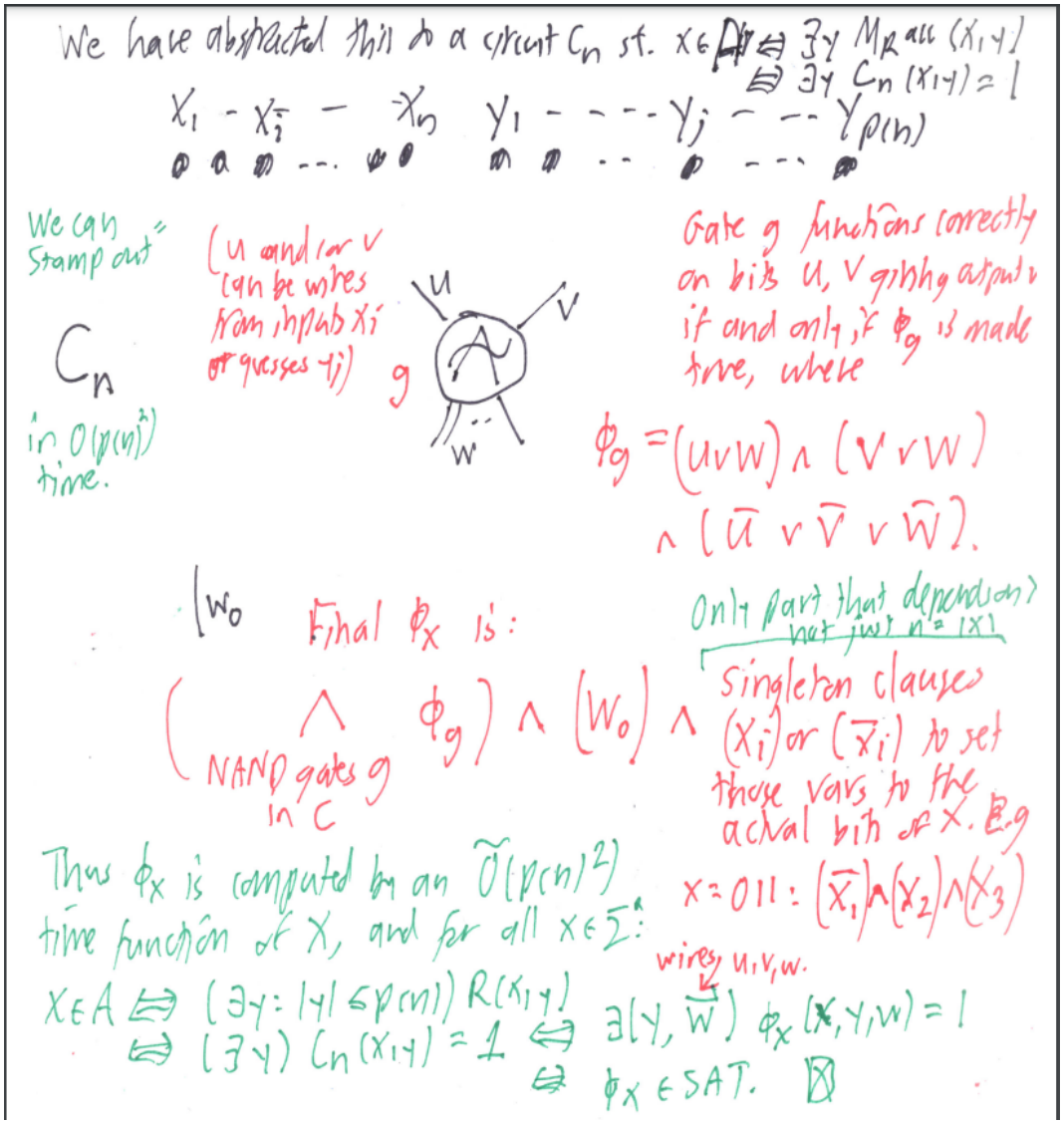
Thus we finally define the reduction function f by

$$f(x) \;=\; \phi_x \;=\; \phi_n \;\wedge\; (w_0) \;\wedge\; \beta_1 \;\wedge\; \cdots \;\wedge\; \beta_n\,.$$

Then $f(x)$ is computable by one streaming pass over the circuit C_n, and so is computable in the same polynomial $O\big(p(n+q(n))^2\big)$ time as C_n. For the mapping of the strings x, we have:

$$x \in A \iff (\exists y : |y| = q(|x|))C_n(xy) \;=\; 1 \iff \big(\exists y \in \{0,1\}^q, w \in \{0,1\}^{s+1}\big) : \text{ the assignment } (x,y,w) \text{ satisfies } \phi_n \wedge w_0 \iff \phi_x \in \mathrm{3SAT}.$$

For the witnesses, the point is that once a y is chosen, on top of x being given (and fixed by the β_i clauses), the values of the rest of the wires in C_n are determined by evaluating all the gates beginning at the top. Hence there is no choice in setting the wire variables w_k besides $w_0 \;=\; 1$. Thus the satisfying assignments are in 1-to-1 correspondence with strings y such that $V_R(\langle x,y\rangle) \;=\; 1$. (If $x \notin A$ then the correspondence is "none-to-none.") ☒



Some decision problems can be shown to be NP-hard or NP-complete by reductions that are "SAT-like." The first example uses the idea of a "mask" being a string of 0,1, and @ for "don't care". For instance, the mask string $s_0 = @01@@0@@$ forces the second bit to be 0, the third bit to be 1, and the sixth bit to be 0. A string like 00101001 "obeys" the mask, but 10011011 "violates" it in the third bit.

MASKS
Instance: A set of mask strings $s_1, \ldots, s_m$, all of the same length n.
Question: Does there exist a string $a \in \{0,1\}^n$ that violates **each** of the masks?

Then we get **3SAT** $\leq_m^p$ **MASKS** via a linear-time reduction f that converts each clause C_j to a mask s_j so that strings a that ***violate*** the mask are the same as assignments that ***satisfy*** C_j. For instance, if $C_j = (x_2 \vee \bar{x}_3 \vee x_6)$, then we get the mask $s_0 = @01@@0@@$ above. [This particular function f is invertible, so that we can readily get the clause from the mask, but it is important to keep in mind which direction the reduction is going in.]

Clearly the language of the **MASKS** problem is in NP, so it is NP-complete. We can also reduce **3TAUT** (whose instances are Boolean formulas ψ in *disjunctive normal form*, called **DNF**, having at most 3 literals per term) to the complementary problem of whether all strings x obey at least one mask. We can also make an NFA N_ψ that begins with ϵ-arcs to "lines" ℓ_j corresponding to each term T_j of ψ. Each line has n states that work to accept the strings x that obey the corresponding mask. Making N_ψ automatically accept all x of lengths other than n gives a reduction from **3TAUT** to the ALL_{NFA} problem, which finally explains why it is *hard*. (It is in fact not only co-NP hard under $\leq_m^p$ as this shows, but also NP-hard; it is in fact complete for the higher class PSPACE which we will get to next month.)

The second example uses two kinds of "recommendations":

- "Positive": choose at least one of these items or these guys;
- "Balancing": don't choose all of these items or all of these guys.

A purely-negative recommendation would be "don't choose *any* of these items or guys" but that doesn't allow any choice, so obeying **each** one doesn't add any complexity to the problem. We can get the effect of "don't choose any of u, v, w" by making the singleton "balancing" recommendations "don't choose all of $\{u\}$", "don't choose all of $\{v\}$", and ""don't choose all of $\{w\}$" anyway, since the recommendations are conjoined together in the statement of the problem:

RECS
Instance: A set U of items and sets $P_1, \ldots, P_k$ and $B_1, \ldots, B_\ell$ of positive and balancing recommendations, respectively.
Question: Is there a subset S of U that obeys **each** recommendation?

Again, the language **RECS** is in NP. To try to show **3SAT** $\leq_m^p$ **RECS** we interpret U as the set of variables (not all literals, just the positive ones) in teh given 3CNF formula ϕ and S as the subset of variables set to 1 by an assignment to ϕ.

- A clause of the form $(u \vee w)$ becomes the positive recommendation, "pick u or pick w."
- A clause of the form $(\overline{u} \vee \overline{v} \vee \overline{w})$ becomes the balancing recommendation, "don't pick all of u, v, w."
- A positive singleton x_i becomes "definitely pick x_i"; a negative singleton $\overline{x}_i$ becomes "don't pick x_i"---which as remarked above is a legal balancing recommendation.

Then an assignment satisfies each of the clauses in ϕ if and only if its "true set S obeys each of the recommendations, so ϕ is satisfiable iff $f(\phi) = \langle U, P_1, \ldots, P_k, B_1, \ldots, B_\ell \rangle$ is in the language of **RECS**. Wait---we didn't define $f(\phi)$ for clauses that have both positive and negative literals, so this isn't a reduction from **3SAT** in general. That's right---it's a reduction from the subproblem of 3SAT that arises in the Cook-Levin-Schnorr reduction. To appreciate and *use* this, we need to reflect on the proof more closely.

Scholia (more than just footnotes---some of these may be useful on HWs)

1. Without loss of generality one can take $q(n)$ to be the same polynomial as $p(n)$. This is tantamount to saying that the verifier $V_R(x, y)$ runs in time $p(|x|)$ that is polynomial in n alone. This makes the verifier incapable of having time to read any y that would be longer than $p(|x|)$. Many sources do this for simplification.
2. In place of the β_i singleton clauses, we could substitute the bit values of x for the corresponding literals into the formula $\phi_n \wedge w_0$ and simplify it. But the β_i are (IMHO) cleaner and can be written in a single streaming pass over x.
3. Many sources insist on strict 3CNF. To get this, we can play our encoding trump card once again: We code V_R so that witness strings y must have an extra final 0. Then the variable called y_q must be set to 0 in any satisfying assignment. So we can add y_q to the binary and singleton clauses to give them all size exactly 3. (If we want to disallow duplicate literals in a clause, we can enforce y ending in 00 and use $y_{q-1} = 0$ too. If we want to keep the feature that all literals in each clause have the same sign, make y end in 1100.)
4. Doing this also yields the feature that no clause can be satisfied by making all three of its literals true, since the $(\overline{u} \vee \overline{v} \vee \overline{w})$ clauses already cannot be satisfied that way owing to $(u \vee w)$ and $(v \vee w)$. This makes the Cook-Levin construction reduce to ``Not-All-Equal 3SAT," abbreviated **NAE-3SAT** (also with the restruction of the literals in each clause having the same sign).
5. We can force two wires w, w' going out of a gate to have the same value by using the clauses

$(w \vee \overline{w'}) \wedge (\overline{w} \vee w')$ rather than do a separate ϕ_g sub-formula for w'. But then we lose the equal-sign property.

6. Doing the extra ϕ_g is not a big deal because without loss of generality we can suppose that each NAND gate has *fanout* at most 2. We can put NAND gates in a tree to replicate the fanout (this may take twice as many levels as a simple binary tree). A similar point is that wlog. an NFA or NTM can be assumed to have binary nondeterminism. This enables assuming that the witness strings y are binary to begin with.
7. The size r of a 3CNF formula ϕ with n variables and m clauses is about $m \log n$ under the natural encoding. But it is generally AOK to regard it as m, and it is OK to regard it as n when we only care about polynomial time. The reason is that ϕ can have at most $8\binom{n}{3} = O(n^3)$ distinct clauses (and it must have at least $n/3$ clauses to use all variables).
8. It follows that $\widetilde{O}(m)$ and $\widetilde{O}(r)$ are the same, so that **3SAT** belongs to NTIME$\left[\widetilde{O}(r)\right]$, which is called NQL for *nondeterministic quasi-linear time*, can be shown quite simply: Guess a 0-1 truth value for every literal in every clause. One streaming pass through the clauses can verify that it makes at least one literal true in each clause. We then need to check that if, say, some positive literal x_i was given the value 0, then every other occurrence of x_i was given 0 and every occurrence of $\overline{x}_i$ was given 1. This can be done by *sorting* the (literal,value) pairs on the indices i to bring all the values for the same i together, and then doing a second single pass to see that all the values brought together are consistent. Since sorting $3m$ items needs only $O(m \log m)$ comparisons, we are done.
9. The polynomial $p(n + q(n))^2$ bounding the time of the reduction and the size of the final formula ϕ_x may seem huge, but when A belongs to NQL the hugeness goes away. First, we use the fact that the k-tapes-to-2 construction (part 2 of the first **Theorem** in the Mon. 10/26 lecture) also makes the verifier machine *oblivious* while multiplying its runtime $t(n)$ by only a $\log t(n)$ factor, which keeps it in deterministic quasilinear time (DQL). What obliviousness does is save a huge amount of unnecessary wiring in the circuits C_n. For each timestep i, we know in advance the cell j the tape head will be in at that time, regardless of the values of the bits in x. So we only need a "delta gadget" in column j and the two neighboring columns. The wires entering the other cells just continue through into the next row. This makes the number of gates in C_n be only $O(p(n))$ not $O(p(n)^2)$. Since ϕ_x is obtained in a single pass over C_n, its size also becomes $O(p(n))$. Since A being in NQL makes both $p(n)$ and $q(n)$ be $\widetilde{O}(n)$, everything is $\widetilde{O}(n)$. This proves **Schnorr's Theorem**: **3SAT** is complete for NQL under DQL reductions.
10. Thus NQL $=$ DQL if and only if **3SAT** is in DQL. IMHO, the NQL $=$ DQL? question is more fundamental than the P versus NP question. It is also "ostensibly" easier to prove NQL $\neq$ DQL than NP $\neq$ P, because you can prove it merely by showing that **3SAT** requires deterministic time $\widetilde{\Omega}(n^2)$, say. The "NQL versus quadratic time" question has begun to receive more attention only recently with evidence that some problems in P cannot be done in

less than $\widetilde{O}(n^2)$ time---if you are curious, see my joint GLL blog post https://rjlipton.wordpress.com/2015/06/01/puzzling-evidence/

11. One can further "pad" the encoding to formulas to make a version "3SAT′ " that is in nondeterministic linear time, NLIN, but you still don't get that the reduction is in deterministic linear time, DLIN. For a highly technical reason having to do with multitape Turing machines it is known that NLIN $\neq$ DLIN, but this result is not "robust"---it goes away if you allow TMs to use 2-dimensional tapes, for instance.
12. It may seem weird that NLIN has languages that are complete for all of NP, but the point is that if A is given in NTIME$[n^k]$ for $k > 1$, then the time to compute the reduction and the size of the final ϕ_x both expand to order n^k (ignoring $\log n^k = O(\log n)$ factors) even with Schnorr's improvements. My "landscape diagrams" visualize the relationships under reductions but don't work well for finer gradations of time or space complexity.

[Friday's lecture will not yet need these "scholia" points and will go into what are often called *reductions by component design*. It and next week will follow section 4 of ALR chapter 28. Debray covers this kind of reductions ultra-tersely, so ALR will be primary. But it is AOK just to read Debray section 15 for Friday (except imagine the reduction going to **IND SET** rather than to **CLIQUE**) and then dive into section 4 of ALR chapter 28 on the weekend after you've already had that taste of what goes on.]