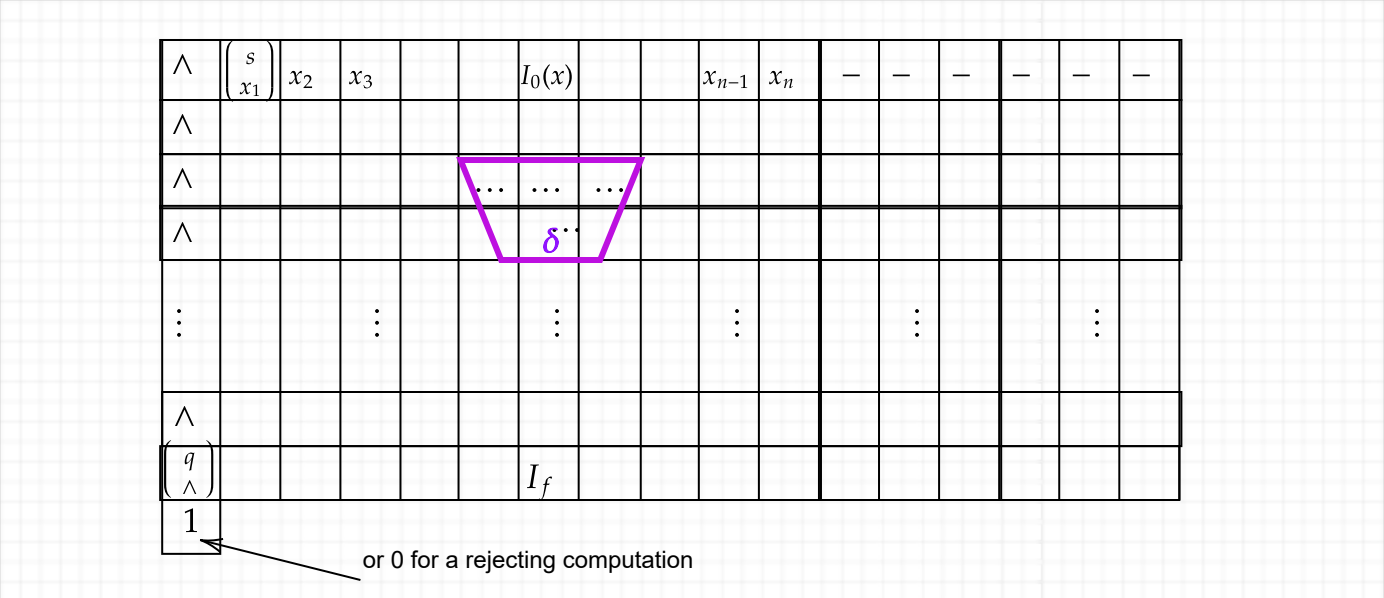
CSE491/596 Lecture Thu. 10/27/22: Cook-Levin Theorem

Before we state and prove the theorem, let us see one more application of the idea of tracing a sequence of IDs $I_0(x),\ I_1,\ I_2,\ \ldots,\ I_t$ that represent a valid t-step computation by a TM M, in this case a DTM. Whereas the Kleene T-predicate pictures them side-by-side, now we will stack them up into $t+1$ columns in a grid. For visual convenience we will suppose M is a 1-tape TM whose tape has a left end and is infinite only to the right, but this is not essential and we could add another grid to handle a second tape, with wires between the grids as well as within them. But for polynomial time, the simple one-plane grid is enough. Initially it has $n+1$ columns to hold the $\wedge$ left-endmarker and the input x. Over t steps, M cannot possibly visit more than t more cells, so we can lay the whole thing out on a $(t+1) \times s$ grid with $s \leq t+1$.



Every cell contains either a character in the work alphabet Γ of M or a pair in $Q \times \Gamma$ of a state and a char. We can use a binary encoding (a-la ASCII) of both. Then we can program a fixed finite function in Boolean logic, depending only on the instructions δ of M, that determines the contents of a cell in any row $i \geq 1$ depending only on the contents of it and its neighbor cell(s) in row $i-1$ for the previous timestep. The top row is initialized to $I_0(x)$ plus blanks to fill out the remaining columns up to t.

Because NAND is a universal gate, we can program the entire grid into a Boolean circuit C_x entirely of NAND gates, with an output wire w_0 at the bottom giving the final results, 1 or 0. Because the formula for δ over every cell is the same, the circuit C_x has such a regular structure (pun quasi-intended) that it is easily computed in $O(t^2)$ time given x. [Added afterward] The "x" is used only once and the values of its bits do not affect the layout, so we can give it via n **input gates** to what is otherwise a circuit C_n that depends only on the length n of x. We could suppose $\Sigma = \{0,1\}$ so x is already in binary, but we could also regard the Boolean encoding of $\Gamma \cup (Q \times \Gamma)$ that the circuit is already using as implicit at the inputs, so there are really $n' = O(n)$ binary input gates. The theorem we have proved has its own significance:

Theorem (often attributed to John E. Savage): For any language A in P and all n we can compute in $n^{O(1)}$ time a circuit C_n of NAND gates such that for all $x \in \Sigma^n$, $x \in A \iff C_n(x) = 1$. ⊠

The meaning of this theorem is that "software can be burned into hardware." The fact that $f(x) = \langle C_n, x \rangle$ is polynomial-time computable goes into saying that the sequence $[C_n]_{n=1}^{\infty}$ of circuits is **P-uniform**. The only reason f is not a "regular reduction" just like the reduction to A_{TM} is that C_n needs counting up to $n = |x|$ and more, and FSTs like DFAs cannot do unbounded counting. But it is close-to-regular in other senses of the above "Scholium" that in fact we get the stronger notion of being **DLOGTIME-uniform**.

Similar diagram from the ALR notes, ch. 27, section 3, showing how each cell depends on its 3 neighbors in the previous row:

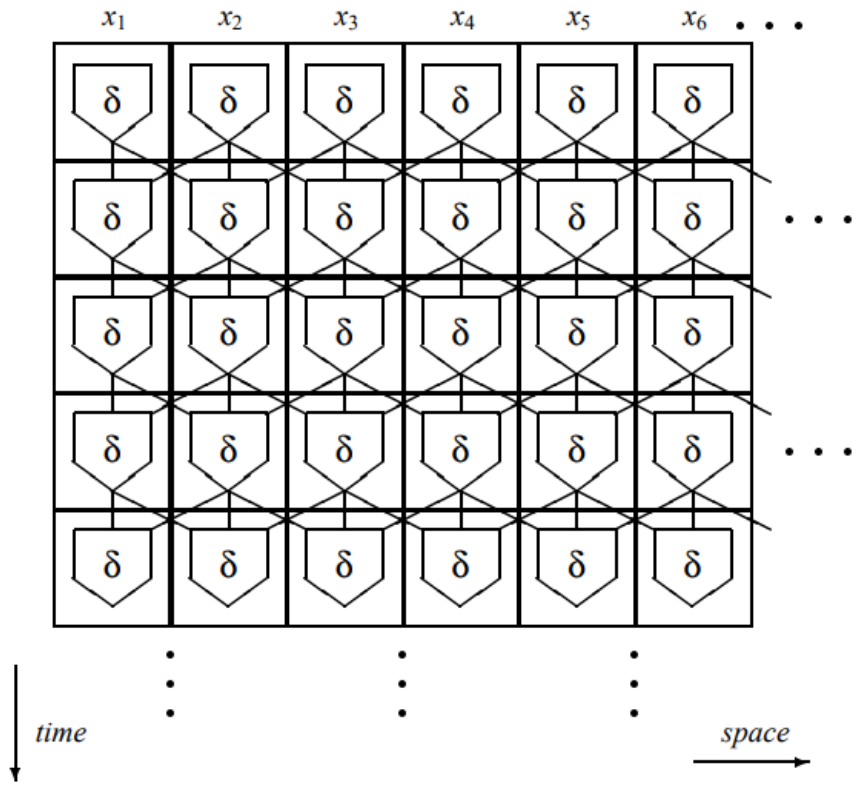


Figure 1: Conversion from Turing machine to Boolean circuits

The Cook-Levin Reduction Function and Proof

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 $\overline{x}_i$. 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

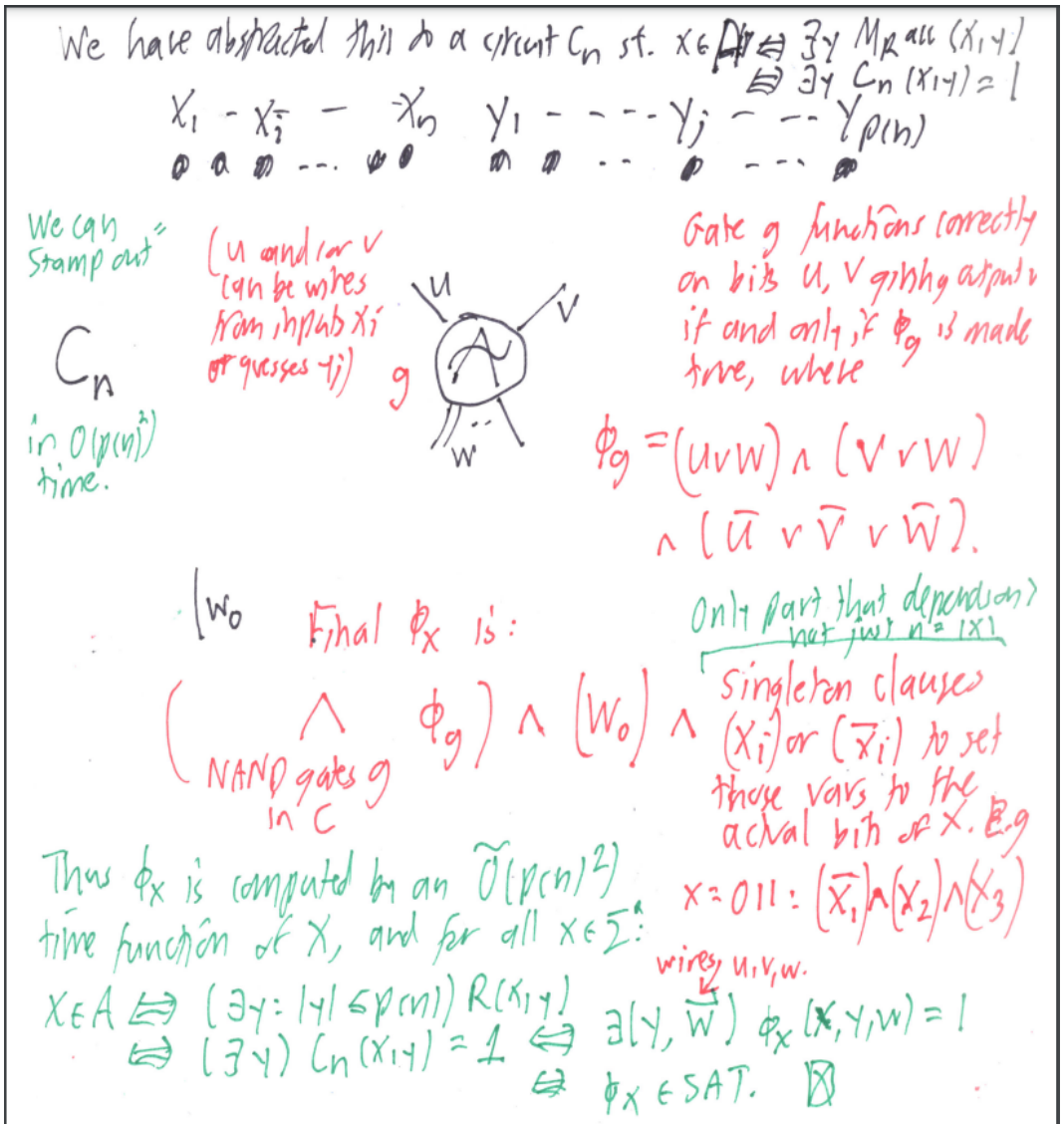
$$f(x) = \phi_x = \phi_n \wedge (w_0) \wedge \beta_1 \wedge \cdots \wedge \beta_n .$$

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:

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

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 the 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.