$\mathcal{P}$ and $\mathcal{NP}$

We say a deterministic TM has time-complexity T(n) if for every input w with length |w| = n the TM halts (whether or not it accepts w) after T(n) steps. The class $\mathscr{P}$ is { L | L is a language accepted by some TM with polynomial time complexity}

We say that a non-deterministic TM has time-complexity T(n) if for every input w with length n the TM can halt after T(n) steps, in an Accept state if the TM accepts w. The class $\mathcal{NP}$ is { L | L is a language accepted by some non-deterministic TM with polynomial time complexity}

While you can ask if any language is in $\mathscr{P}$ or $\mathscr{NP}$ we are often interested in algorithmic questions such as "Find the shortest path from node q_1 to node q_2 in this weighted graph." That translates to a $\mathscr{P}$ or $\mathscr{NP}$ question by looking at the language $\{g1110^n \mid g$ is an encoding of a weighted graph and the graph has a path of length n or less from node q_1 to node $q_2\}$ A TM might determine if $g1110^n$ for a particular graph g and a particular n is in this language by finding a path from q_1 to q_2 with length n.

Note that a non-deterministic TM can solve this by guessing the sequence of nodes on the shortest path from q_1 to q_2 and then verifying in polynomial time that these nodes do form a path from q_1 to q_2 and that the sum of the lengths of the edges on this path is no more than n.

Many people describe $\mathscr{P}$ as the set of problems that can be *solved* in polynomial time while $\mathscr{NP}$ is the set of problems for which a solution can be *verified* in polynomial time.

It is obvious that $\mathscr{P}$ is a subset of $\mathscr{NP}$. Perhaps the most important unsolved question in CS is: Is $\mathscr{P} = \mathscr{NP}$? This question arises from Cook's (or Cook-Levin) Theorem, which says that if one specific language L is in $\mathscr{P}$ then $\mathscr{P}=\mathscr{NP}$.

Let L be a language in $\mathcal{NP}$. We say L is NP-complete if for every language A in $\mathcal{NP}$ there is a polynomial time reduction of A to L in the sense that we can covert any string w in polynomial time to a string w' so that w is in A if and only if w' is in L. A polynomial-time decider for L then gives us a polynomial-time decider for every language A in $\mathcal{NP}$.

In other words, if L is NP-complete and L is in $\mathscr{P}$, then every problem that can be verified in polynomial-time could actually be solved in polynomial-time. That would have enormous ramifications.

We say a language L is NP-hard if every language A in $\mathcal{NP}$ reduces to L. So to be NP-complete a language must be

a) In $\mathcal{NP}$
b) NP-hard

Boolean expressions.

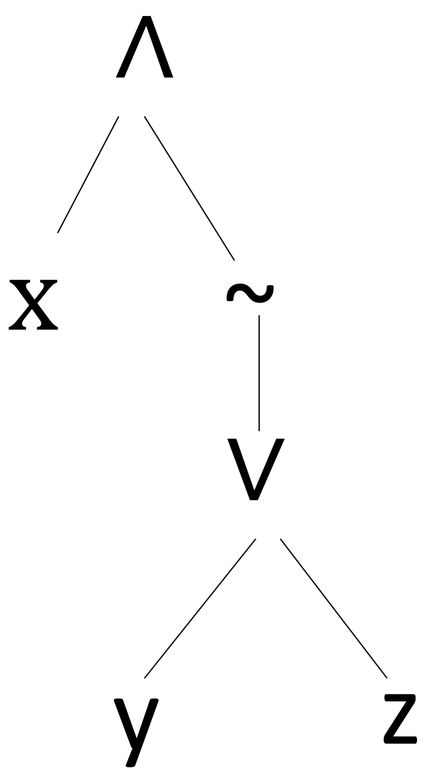
We will use $\wedge$, $\vee$, and ~ to represent the Boolean operators *and*, *or*, and *not*.

Definition: A Boolean expression is

a) A variable that can have value T or F
b) $e \wedge f$, $e \vee f$, $\sim e$, or (e) where e and f are Boolean expressions

For example, $x \wedge \sim(y \vee z)$ is a Boolean expression

Given values of the variables we can find the value of this expression: build a parse tree for it (linear time) and pass the Boolean values up the tree from the leaves to the root:



Given a Boolean expression we can find if there is a set of assignments to its variables for which the expression evaluates to T. We say such an expression is *satisfiable*. For example, we could build a truth table for it:

x	**y**	**z**	$\mathbf{x \wedge {\sim}(y \vee z)}$
T	T	T	F
T	T	F	F
T	F	T	F
T	F	F	T
F	T	T	F
F	T	F	F
F	F	T	F
F	F	F	F

Unfortunately, a truth table with k variables has 2^k lines so it can't be completed in polynomial time.

SAT is the language of satisfiable Boolean expressions.

Ex: $x \wedge \sim(y \vee z)$ is in SAT: take x=T, y=F, z=F

Ex: $x \wedge \sim y \wedge (y \vee \sim x)$ is not in SAT

Cook's Theorem (Stephen Cook, U. Toronto, 1971): SAT is NP-complete.

It is easy to see that SAT is in $\mathcal{NP}$: Guess the right values of the variables and verify them by evaluating a parse tree for the expression. This takes linear time.

To prove Cook's Theorem we need to show that every $\mathcal{NP}$ problem reduces in polynomial time to SAT.

Let L be any language in NP. This means there is a non-deterministic TM M that accepts L and M halts on any input w in time p(|w|) for some polynomial p.

To prove Cook's Theorem we will produce from M and w a Boolean expression that is satisfiable if and only if M accepts w.

Suppose w is any string with |w| = n and M is any TM. If M accepts w there is a sequence of configurations $\alpha_0\ \alpha_1 \ldots \alpha_{p(n)}$ so that

a) α_0 is the initial configuration for the computation of M on w
b) Each $\alpha_i \Rightarrow \alpha_{i+1}$
c) $\alpha_{p(n)}$ is a configuration in an accept state.

We will create a Boolean expression B that is satisfiable if and only if such a sequence of configurations is possible. So if SAT is in $\mathscr{P}$ we can show L is in $\mathscr{P}$:

a) Start with a nondeterministic TM that accepts L
b) For any string w construct B in polynomial time
c) determine if B is in SAT in polynomial time
d) B is in SAT if and only if w is in L

Note that we need to construct B in polynomial time, so it is important that |B| be a polynomial function of |w|.

In k steps we can write at most |w|+k symbols on the tape so we'll assume the non-blank portion of the tape is no longer than p(n).

Also, we 'll assume the TM runs exactly p(n) steps for any input w with |w|= n

Here is some notation we'll use:

X_{ij} is the j^th symbol of the i^th configuration. If the 4^th configuration is $11q_2 00$ then X_{30} =1, X_{31}=1, X_{32}=q_2, X_{33}=0, and X_{34}=0

For any tape symbol or state A, Y_{ijA} is a Boolean variable whose intuitive meaning is "X_{ij}==A"

We will assume the start state of any TM is q_1.

The Boolean expression we will construct is $B=S \land N \land F$ where

- S says the first configuration is $q_1 w$ (where q_1 is the start state of the TM)
- N says each configuration is derived from the previous one.
- F says that in the $p(n)^{th}$ configuration the TM is in a final state

S and F are easy; N takes some work.

Step 1: If input w is $a_1a_2 \ldots a_n$ then

$$S = Y_{00q1} \wedge Y_{01a1} \wedge Y_{02a2} \ldots \wedge Y_{0nan}$$

Step 2: Let $q_{f1}..q_{fk}$ be all of the final states of M.

Let F_{ji} be $Y_{p(n)jqfi}$ This says the j^{th} symbol of the last configuration is q_{fi}

Let F_j be $F_{j1} \vee F_{j2} \vee .. \vee F_{jk}$ This says the jth symbol of the last configuration is a final state.

Finally, F is $F_0 \vee F_1 \vee \ldots \vee F_{p(n)}$ this says the TM accepts w.

Note that |Fj| is independent of w, so |S| and |F| are both O(p(n))

Step 3: We only need N, which says that each configuration is derived from the previous one. In fact, we'll make

$$N = N_0 \wedge N_1 \wedge \ldots \wedge N_{p(n)-1}$$

where N_i says that configuration i+1 is derived from configuration i.

To make Ni we need two kinds of subexpressions:

A_{ij} will say that the state symbol of the ith configuration is at position j and also that the j-1st, jth, and j+1st symbols of the i+1st configuration are correct for the corresponding transition of M.

B_{ij} will say that either the state symbol of the ith configuration is at position j-1 or j+1 (and so symbol j is covered by A_{ij}) or else position j has a tape symbol that is copied correctly from configuration i to configuration i+1.

Given these, $N_i=(A_{i0} \vee B_{i0}) \wedge (A_{i1} \vee B_{i1}) \wedge \ldots \wedge (A_{ip(n)} \vee B_{ip(n)})$

Let's pause for an example. Suppose the i^{th} configuration is $010q_110$ and M has transition $\delta(q_1,1)=(q_2,1,R)$. We want the $i+1^{st}$ configuration to be $0101q_20$.

B_{i0} will say the initial 0 is copied correctly
B_{i1} will say the 1 is copied correctly
B_{i2} will say T
B_{i3} will say F
A_{i3} will say $0q_11$ is changed to $01q_2$
B_{i4} will say T
B_{i5} will say the final 0 is copied correctly

To make Bij, let $t_1 \ldots t_k$ be all of the tape symbols and $q_1 .. q_m$ all of the states.

$B_{ij} = (Y_{i(j-1)q1} \vee Y_{i(j-1)q2} \vee \ldots \vee Y_{i(j-1)qm}) \vee (Y_{i(j+1)q1} \vee Y_{i(j+1)q2} \vee \ldots \vee Y_{i(j+1)qm})$
$\vee [(Y_{ijt1} \wedge Y_{(i+1)jt1}) \vee (Y_{ijt2} \wedge Y_{(i+1)jt2}) \vee \ldots \vee (Y_{ijtk} \wedge Y_{(i+1)jtk})]$

Note that |Bij| has nothing to do with the input w.

A_{ij} describes the legal transitions..

Suppose we have a move to the right: $\delta(q_s,a)=(q_t,b,R)$

If the i^{th} configuration is $\alpha c q_s a \beta$ with q_s at position j, we want the $i+1^{st}$ configuration to be $\alpha c b q_t \beta$

The phrase of Aij for this is

$$p = Y_{ijqs} \wedge Y_{i(j+1)a} \wedge Y_{(i+1)jb} \wedge Y_{(i+1)(j+1)qt}$$
$$\wedge[(Y_{i(j-1)t1} \wedge Y_{(i+1)(j-1)t1}) \vee \ldots \vee(Y_{i(j-1)tk} \wedge Y_{(i+1)(j-1)tk})]$$

On the other hand suppose we have a move left: $\delta(q_s,a)=(q_t,b,L)$

If the i^{th} configuration is $\alpha c q_s a \beta$ with q_s at position j, we want the i+1st configuration to be $\alpha q_t c b \beta$. The phrase of Aij for this is

$$p = Y_{ijqs} \wedge Y_{(i+1)(j-1)qt} \wedge Y_{i(j+1)a} \wedge Y_{(i+1)(j+1)b}$$
$$\wedge[(Y_{i(j-1)t1} \wedge Y_{(i+1)jt1}) \vee \ldots \vee (Y_{i(j-1)tk} \wedge Y_{(i+1)jtk})]$$

If M has L transitions and p_{ijt} is the corresponding A_{ij} phrase for transition t then

$$A_{ij} = p_{ij1} \vee p_{ij2} \vee \ldots \vee p_{ijL}$$

This completes the construction. Note that this seamlessly incorporates the nondeterminism of the TM: SAT's question about whether *some* assignment of variables satisfies B corresponds to the nondeterministic question of whether there is *some* valid sequence of configurations that gets to a terminal state.

Now, how big is B? $B = S \wedge N \wedge F$

$|S| = O(n)$

$|F| = O(\ p(n)\ )$

$|N| = O(\ p^2(n)\ )$

This completes the proof that SAT is NP-complete.