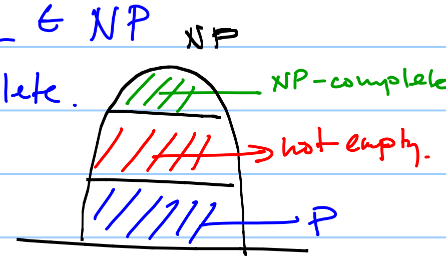
Note Title 4/16/2013

We will discuss two more theorems about the structure of NP before going on to non-deterministic space classes.
The first is Ladner's theorem which we will just state and not prove:

Ladner's Theorem: If $P \neq NP$ then $\exists\, L \in NP$ s.t. $L \notin P$ and L is not NP-complete.



The proof of Ladner's theorem which is omitted here uses a lazy diagonalization argument similar to the proof of the Non-deterministic time hierarchy theorem.

The next theorem is about unary languages (subsets of Σ_1^* in which Σ_1 has only one character).

One way of viewing NPC languages is that they are computationally hard. Another way to see them is that they are expressive. (e.g. Boolean satisfiability can express any language in NP).

A sparse language contains at most poly(n) strings of length $\leq n$ $\forall\, n \geq 1$.

It can be shown that if $P \neq NP$, then there is no sparse NPC language.

We will prove a weaker version of this theorem.
A language is unary if it is a subset of 1^* ← $\leq n$ strings of length n.

Theorem (Berman '78)

If a unary language is NP-complete, then P = NP.

Proof: Let U be a unary language and assume that SAT $\propto$ U. by reduction R.

We will show a polynomial time algorithm for SAT.

$\phi(x_1, \ldots, x_n)$ instance of SAT.

Consider a partial assignment to the first i variables $t_1 \cdots t_i$.
This gives a new boolean formula on n-i variables:

$\phi(t_1, \ldots, t_i, x_{i+1}, \ldots, x_n)$

if a literal becomes true, remove any clause in which it participates.

if a literal becomes false, remove it from any clause in which it participates

Eventually, we will get an empty clause (ϕ not satisfiable) or we will remove all clauses (ϕ satisfiable).

Note that $\phi(t_1, \ldots, t_i, x_{i+1}, \ldots, x_n)$ is a boolean formula to which R can be applied.

Consider an exhaustive search to determine if ϕ is satisfiable.

$\phi(x_1, \ldots, x_n)$

$\phi(1, x_2 \ldots x_n)$ $\quad$ $\phi(0, x_2 \cdots x_n)$

$\phi(1, \ldots, 1)$ $\quad$ $\phi(0 \ldots 0)$

At each leaf in the tree, the truth assignment is determined $\Rightarrow$ can tell if $\phi(t_1 \dots t_n) = T$ or F.

Instead of exhaustively searching the entire tree, we will use a hashing scheme to prune the tree. Will use R as the hash function.

Note that if $R(\phi) = R(\phi')$ then ϕ is satisfiable iff ϕ' is satisfiable.

We will keep a list:

[$R(\phi(t))$, Y or N] [$R(\phi(t'))$, Y or N], ...

t is a generic partial assignment

tells whether $\phi(t)$ is satisfiable.

Eval [$\phi(t)$]

{ Check if $R(\phi(t))$ is in the table. If so, return the correct answer.

If not:

Eval [$\phi(t0)$]

Eval [$\phi(t1)$]

If either is satisfiable insert [$R(\phi(t))$, Y] into the tree + return YES.

If neither is satisfiable, insert [$R(\phi(t))$, N] into the tree + return NO

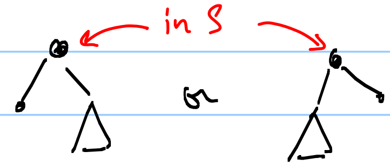
}

Each $R(\phi(t))$ has length $\le$ poly$(|\phi|)$.
Since $R(\phi(t)) \subseteq 1^*$ there can be at most a polynomial distinct values.
(if R produces a string w/ any 0's, we know it's not in U, we can replace it with a fixed string '0' instead).

The size of the hash table is never bigger than poly$(|\phi|)$.

How many nodes in the tree are visited?
Call this number M.
The overall running time is $O(M \cdot \text{poly}|\phi|)$.

We will prune the tree to contain only the visited nodes. Let S be the set of parents of leaves in the tree:



We know that $4|S| \ge M$.

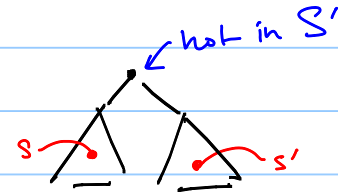
Can show by induction that in a binary tree #interior nodes $\le$ # leaves -1.
$2|S| \ge$ # leaves. $M \le$ int + leaves $\le$ 2 leaves $\le 4|S|$.

Remove from S any node which is the ancestor of another node in S to give a new set S'.

$n|S'| \ge |S|$.

The length of a path from a node to the root $\le n$.
Each node left in S' can cause the removal of $\le n$ nodes from S.

So now we have this picture



if $s + s' \in S'$
then their lowest common ancestor is not in S'.

Claim: for $s, s' \in S'$, let s correspond to $\phi(t)$
s' correspond to $\phi(t')$
then $R(\phi(t)) \neq R(\phi(t'))$.

Suppose s is visited first. $R(\phi(t))$ is stored in the hash table before s' is visited. The recursive (call $\phi(t)$ completes before s' is reached).

If $R(\phi(t')) = R(\phi(t))$ then there would have been no recursive calls to the children of s'.
This contradicts the fact that $s' \in S'$.

The number of recursive calls $\leq 4n \, \mathrm{poly}(|\phi|)$
n = # variables in ϕ
which is $\leq |\phi|$.

Recap on complexity classes:

NTIME classes: NP, co-NP, NEXP.
NP $\neq$ NEXP (Ntime hierarchy).

Major open questions: $P \stackrel{?}{=} NP$, $NP \stackrel{?}{=}$ co-NP.

NP has intermediate problems (unless $P = NP$)
Sparse/unary languages not NP-C unless $P = NP$.

Circuit SAT   NP - Complete
UNSAT   Co-NP - complete.
Succinct Ckt SAT is NEXP - complete.