Alin Tomescu, *6.840 Theory of Computation (Fall 2013)*, taught by Prof. Michael Sipser

Space and time complexity theorems

"Space is more powerful than time"

"Because time cannot be reused, but space can."

If you have a $s(n)$ **space** TM, then its running time is bounded by $c^{s(n)}$, where c is a TM-specific constant (governed by the number of states, size of the alphabet, etc.). This bound results from the maximum number of configurations a TM can be in, given that it only consumes up to $s(n)$ space and halts (for each tape cell you can place up to c symbols on it $\Rightarrow c^{s(n)}$ time).

If you have a $t(n)$ **time** TM, then it cannot run in more than $t(n)$ space. How could it write more than $s(n)$ cells if it's running in $s(n)$ of time?

The Cook-Levin theorem (SAT is NP-complete)

This theorem tells us that the satisfiability problem (SAT) is NP-complete. This means that given a SAT solver, you can reduce any problem Q in NP to an instance of SAT, solve that instance using your solver, then convert the SAT solution into a Q solution by reversing your reduction. Hence, you would have solved your initial Q problem.

Theorem: SAT is NP-complete

Proof idea: To oversimplify, since computers (and hence TMs) are pretty much just a big Boolean formula it's not surprising that we can encode a TM as a Boolean formula. The idea is to take a TM M for a language A in NP and encode whether it accepts w or not in a Boolean formula. We use *tableaus* to figure out what needs to be encoded in terms of transitions, accept and start states, etc. See page 306 of Sipser 3rd Ed. for more details.

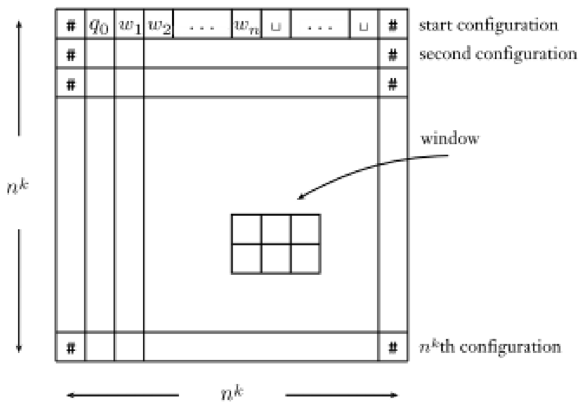

Figure 1: A tableau for a TM that runs in time n^k.

Savitch's theorem ($NSPACE\big(s(n)\big) \subseteq SPACE\big(s^2(n)\big)$, NPSPACE = PSPACE)

This theorem tells us that (surprisingly) we can simulate non-deterministic space machines using deterministic space machines with just an order of magnitude overhead.

Theorem: $NSPACE\big(s(n)\big) \subseteq SPACE\big(s^2(n)\big), \forall s(n) \geq \log n$

Proof idea: Since we are not limited by time in $NSPACE\big(f(n)\big)$ machines, the idea is to search over all the configurations of the $NSPACE$ machine using a binary search and see if there is a path from the start configuration to an accepting configuration. We can simply generate all intermediate configurations (exponential number of such things) and try and connect the start configuration with the accepting one via an intermediate configurations. We apply this search recursively.

Imagine a *configuration table* of $M \in NSPACE\big(s(n)\big)$ on w, with rows representing the *configuration* (tape contents, previous and next state, head position) of M at a particular point.

- The first row is the start configuration.
- The last row is the accept configuration.
- Consecutive rows are adjacent configurations.
- A row will be $s(n)$ long (because the tape cannot have more than $s(n)$ data on it, since $M \in NSPACE(s^n)$), and there are at most $d^{s(n)}$ such rows for an accepting computation branch (because $SPACE\big(s(n)\big) \subseteq TIME\big(2^{O(s(n))}\big) \subseteq \cup_d TIME\big(d^{s(n)}\big)$).

The deterministic TM N will simulate the nondetermistic TM M by searching the above tableau and trying to figure out whether it accepts or rejects w. N will do a binary search, reusing space, to see if there is a path from the first configuration C_{start} to the accepting configurations C_{acc} in in less than $d^{s(n)}$ steps. N can verify that such a path exists by looking at the description of M on its tape. Note that N has no time limitation, only space. The algorithm is recursive, and the base case occurs when N has to check two adjacent configurations $C_i \rightarrow^{t=1} C_j$.

Space hierarchy theorem (NL ≠ PSPACE ≠ EXPSPACE)

Theorem: For *almost any** function $f: \mathbb{N} \rightarrow \mathbb{N}$, a language A exists that is decidable in $O\big(f(n)\big)$ space but not in $o\big(f(n)\big)$ space.

* by *any function* we mean a *space constructible function*, which means ∃TM that, given input 1^n, always halts with the binary representation of $f(n)$ on its tape using $O\big(f(n)\big)$ space. It seams $f(n)$ has to be at least $O(\log n)$.

This implies that $NL \neq PSPACE$, because Savitch's theorem tells us $NL = NSPACE(\log n) \subseteq SPACE((\log n)^2)$ and the space hierarchy theorem tells us that there exists an A that runs in $O\big(poly(n)\big)$ space and not in anything less than that such as $(\log n)^2$ space. Thus, there's a problem in $PSPACE$ that is not in NL.

Time-hierarchy theorem (P ≠ EXPTIME)

Theorem: For *almost any** function $f: \mathbb{N} \rightarrow \mathbb{N}$, a language A exists such that A is decidable in time $O\big(f(n)\big)$ but not decidable in time $o(f(n)/\log f(n))$.

* by *any function* we mean a *time constructible function*, which means ∃TM that, given input 1^n, always halts with the binary representation of $f(n)$ on its tape in time $O\big(f(n)\big)$. It seems that $f(n)$ has to be at least $O(n \log n)$.

"What this means is that for almost all $f(n)$, some languages can be decided in $f(n)$-time but no faster. This helps prove some time classes are different."

The time-hierarchy theorem implies that $P \neq EXPTIME$, if you let $f(n) = 2^n$.

P and NP closure properties

P is closed under:

- **Complement**, because I can take the TM M for any language $A \in P$ and just flip its answer to get a TM M' for $\bar{A}$, which means $\bar{A} \in P$.'
- **Union,** because given two TMs N and M for $A, B \in P$ I can build a third TM D which first simulates N and then simulates M. If either of them accept, D accepts, otherwise it rejects. D will thus decide if $w \in A$ or $w \in B$, and will run in poly-time (since simulation overhead is polynomial) thus $A \cup B \in P$.

- **Concatenation,** because given two TMs N and M for $A, B \in P$ I can build a third TM D which tries all possible ways of splitting the input w as $w = ab$ and tests if $a \in A$ and $b \in B$. Note that there are $n + 1$ such splits and checking each one involves simulating N and M which run in $poly(n)$.
- **Kleene star,** because we proved it in the HW with a dynamic programming technique similar to the one used to decide A_{CFG} in poly-time.

NP is closed under:

- **Union**, just like P
- **Concatenation**, just like P

NP is *not known* to be closed under **complement:** no one knows if $NP = coNP$. Note that you can't just flip the answer of an NTM for a language $A \in NP$ and get a decider for $\bar{A}$. This is a flaw in thinking, because you assume you would be flipping the final *yes* or *no* answer of the NTM, but in fact you are flipping the answer for a single branch in the NTM computation. This is because, very informally speaking, you do NOT have access to the final yes/no answer of the NTM, as you "modify your NTM to flip the answer". You can only modify your NTM to flip a branch's answer, and this will not help you, because you will end up flipping, say, the $(branch_1, acc), (branch_2, rej), (branch_3, acc)$ accepting NTM computation into $(branch_1, rej), (branch_2, acc), (branch_3, rej)$ which is still an accepting NTM computation (since you still have an accepting branch).

Surprise, NL = coNL!

$PATH \in NL$, $PATH$ is NL-complete (note this implies $\overline{PATH}$ is coNL complete, if you reverse the reduction).

The difficult part is to show that $UNREACHABILITY = \overline{PATH} \in NL$. Once that's done (see textbook), the following comes easy:

$$\overline{PATH} \in NL \Rightarrow PATH \in coNL \Rightarrow (\text{since } PATH \text{ is NL} - \text{complete}) \Rightarrow \forall A \in NL, A \in coNL$$

$$PATH \in NL \Rightarrow \overline{PATH} \in coNL \Rightarrow (\text{we know that } \overline{PATH} \text{ is coNL} - \text{complete}) \Rightarrow \forall A \in coNL, A \in NL$$

Thus, $NL = coNL$.