CIS 511: Theory of Computation **Feb 28, 2017**

Lecture 14: More Space Complexity

Professor Sampath Kannan *Zach Schutzman*

NB: *These notes are from CIS511 at Penn. The course followed Michael Sipser's Introduction to the Theory of Computation (3ed) text.*

$DSPACE$ v $NSPACE$

Recall, we showed last time that $SAT \in DSPACE(n)$.

We were thinking about the language $NA_{NFA} = \{\langle M\rangle | M \text{ is an } NFA,\ L(M) \neq \Sigma^*\}$. We showed in homework that the shortest string not accepted by an NFA can be exponential in the number of states of the machine. Recall also that for the DFA problem, the upper bound is simply the number of states in the machine. We can transform a k-state NFA into a DFA with at most 2^k states, so we can upper bound the length of the shortest non-accepted string by $2^k - 1$.

Our input is an NFA with description length n. The number of states is approximately equal to n (maybe n^2, we don't really care). A naive deterministic approach involves checking every string. Let's design a non-deterministic space algorithm to do the search. We non-deterministically guess one symbol at a time. The depth of the tree is $2^n - 1$, which is still exponential. But we don't really need to remember the whole path, just the current possible states and the counter of the depth. This count is length order n and the set of current states is also order n. We therefore have an $NSPACE(n)$ algorithm to solve this, hence $NA_{NFA} \in NSPACE(n)$.

Claim 14.1 *Savitch's Theorem: $NSPACE(f(n)) \subseteq DSPACE(f(n)^2)$. That is, we only get a quadratic increase by moving to a deterministic model.*

Proof:

(For space $f(n) = \Omega(n)$)

Given an $O(f(n))$-space NDTM M, we will design an $O(f(n)^2)$-space DTM D which recognizes the same language.

The input x determines the initial configuration of M, C_{init}. We are interested in whether M can get to some accepting configuration with the q_a state in the first cell and the rest of the tape blank (canonically). We have an upper bound of $2^{O(f(n))}$ steps needed to reach such a configuration.

Write $C_i \vDash_t^M C_j$ to say that there is some valid computation in M that moves from configuration C_i to configuration C_j on some branch in at most t steps. We want to know if $C_{init} \vDash_{2^{O(f(n))}}^M C_{accept}$. We can approach this with divide-and-conquer by asking whether $C_{init} \vDash_{\frac{2^{O(f(n))}}{2}}^M C_m \vDash_{\frac{2^{O(f(n))}}{2}}^M C_{accept}$ for some intermediate configuration C_m.

At each step, we need to check whether the start configuration can reach the end configuration. Since we can reuse space, we need to remember $\log 2^{O(f(n))} = O(f(n))$ intermediary configurations, each of length $O(f(n))$. This requires total space of $O(f(n)^2)$

■

Definition 14.2 *PSPACE is defined as* $\bigcup_k DSPACE(n^k)$*. NPSPACE is* $\bigcup_k NSPACE(n^k)$*.*

Corollary 14.3 *By Savitch's Theorem,* $PSPACE = NPSPACE$*. Nobody talks about NPSPACE because it is a redundant name for PSPACE.*

Space vs Time Complexity

We have $DTIME(f(n)) \subseteq DSPACE(f(n))$*, as if you can only take* $f(n)$ *steps, you are only using* $f(n)$ *cells (at most). The same holds for the non-deterministic case.*

We also have that $DSPACE(f(n)) \subseteq DTIME(2^{O(f(n))})$*, because we can enumerate all of the* $2^{O(f(n))}$ *configurations of the machine in* $f(n)$ *space, one at a time. To see that we only see each configuration at most once, observe that even in the non-deterministic setting, if two branches have the same configuration, the continuations of those branches must be identical, so we don't need to check it more than once. Therefore, without loss of generality, there is some accepting path that does not contain repeated configurations. We can keep a 'timer' that terminates a branch when it has exceeded* $2^{O(f(n))}$ *steps, thus ensuring that every branch terminates in at most* $2^{O(f(n))}$ *steps. The same holds for the non-deterministic case.*

We now have $P \subseteq NP \subseteq PSPACE$*. We don't know if* $P = PSPACE$*, but this should be easier to prove than* $P = NP$*.*

The Class *PSPACE-Complete*

Definition 14.4 *A language L is PSPACE-Complete if L is in PSPACE and for all* $A \in PSPACE$*, there exists a mapping reduction from A to L in polynomial time.*

Claim 14.5 *The language of True Quantified Boolean Formulas (TQBF) is PSPACE-Complete.*