**15-855: Intensive Intro to Complexity Theory**
Spring 2009

Lecture 7: **Space vs. Time, Savitch's Theorem, TQBF, Immerman-Szelepcsényi Theorem, Intro to Circuits**

# 1 Three main theorems

We will see three basic, major theorems about space classes. The first is very easy; the second and third are quite surprising. All three statements involve nondeterministic space, although as we will see, they have consequences for deterministic (i.e., "real-world") space classes.

**Theorem 1.1.** $\mathsf{NSPACE}(s(n)) \subseteq \mathsf{DTIME}(2^{O(s(n))}) = \cup_{k \geq 1} \mathsf{DTIME}(2^{ks(n)})$.

**Theorem 1.2.** ***(Savitch's Theorem.)*** $\mathsf{NSPACE}(s(n)) \subseteq \mathsf{DSPACE}(s(n)^2)$.

**Theorem 1.3.** ***(Immerman-Szelepcsényi Theorem.)*** $\mathsf{coNSPACE}(s(n)) = \mathsf{NSPACE}(s(n))$.

The first theorem here just gives a straightforward upper-bound on nondeterministic space — hence deterministic space — by time. From it we conclude:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P}, \quad \text{and} \quad \mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \subseteq \mathsf{EXP}.$$

The second theorem, proved by Savitch in 1970, is very important: First, it shows that nondeterminism is surprisingly weak in the context of space. Second, its proof is an important paradigm that comes up again and again in complexity theory. As we'll see several times in the course, several theorems have proofs that are "basically Savitch's Theorem" or are "based on Savitch's proof". In particular, Savitch implies:

$$\mathsf{NPSPACE} = \mathsf{PSPACE}.$$

Since PSPACE is closed under complement, this immediately implies coNPSPACE = NPSPACE. Whether this holds for "lower space" as well took about 20 years to resolve; Theorem 1.3 was proved simultaneously by Immerman and Szelepcsényi in 1988. It *also* shows an unexpected result about nondeterminism and space. It hasn't been as *useful*, subsequently, as Savitch's Theorem, but it is of course essential to know.

Summarizing some of what we've learned:

$$\mathsf{L} \subseteq \mathsf{NL} = \mathsf{coNL} \subseteq \mathsf{P} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP},$$

and there's no need to ever write NPSPACE or coNPSPACE, since these equal PSPACE. Recall also from the Space Hierarchy Theorem that we know

$$\mathsf{L} \neq \mathsf{PSPACE},$$

but we can't prove any of $\mathsf{L} \neq \mathsf{NL}$, $\mathsf{NL} \neq \mathsf{P}$, or $\mathsf{P} \neq \mathsf{PSPACE}$.

## 2 Bounding space by time: configuration graphs

Let's first prove Theorem 1.1, which is quite easy. Suppose we have an NTM $M$.

**Definition 2.1.** *A* configuration *of $M$ is a tuple*

*($M$'s work tape contents, location of $M$'s TM heads, $M$'s current state).*

Recall that NTMs have some $O(1)$-size alphabet, some $O(1)$ many work tapes (hence heads), and $O(1)$ states. So if an NTM uses $s(n)$ space on all branches, the total number of configurations possible is at most

$$O(1)^{s(n)} \cdot n \cdot s(n)^{O(1)} \cdot O(1) \leq 2^{O(s(n))}.$$

(The $n$ was for the head-location on the input tape; this is at most $2^{s(n)}$ because $s(n) \geq \log n$.)

**Definition 2.2.** *Given a length-$n$ input $x$, the* configuration graph *is the directed graph $G = (V, E)$, where $V$ is the set of all configurations, and $(u, v) \in E$ if $M$ may transition from configuration $u$ to configuration $v$.*

**NB:** 1. The set $V$ does *not* depend on the input $x$, but the edges $E$ *do*. 2. A vertex $u$ may have multiple outgoing edges because $M$ is *nondeterministic*.

We can now prove Theorem 1.1.

*Proof.* Suppose $L \in \mathsf{NSPACE}(s(n))$, and let $M$ be an NTM deciding $L$. It is not hard to see that, given input $x$, a deterministic algorithm can write down the entire configuration graph for $M$ on $x$ in time $2^{O(s(n))}$. There is a unique "start" configuration $S$, and we may also assume WOLOG that there is a unique "accepting" configuration $T$ (because we can require that $M$ reset all its tapes to blank and its heads to the left before accepting). So by definition of an NTM, $x \in L$ iff there is a directed path from $S$ to $T$. This can be decided in $\text{poly}(|V||E|) = 2^{O(s(n))}$ time by any standard method; e.g., depth-first search. □

Here is a very important corollary:

**Definition 2.3.** *"STCON" is the problem: Given a directed graph and two distinguished vertices $S$ and $T$, is there a path from $S$ to $T$?*

**Corollary 2.4.** *STCON is* $\mathsf{NL}$*-complete (under log-space reductions). Hence* $\mathsf{NL} = \mathsf{L}$ *iff STCON* $\in \mathsf{L}$.

*Proof.* STCON $\in \mathsf{NL}$ is easy: Start at $S$ and use nondeterminism to guess the next vertex to visit. Repeat up to $|V|$ steps and accept if you've reached $T$. All you need to write on the work tape is the index of the current vertex, the next one you'll visit, and the number of steps you've taken, all $O(\log n)$ space.

The fact that STCON is $\mathsf{NL}$-hard follows by the proof of Theorem 1.1; we just have to check that outputting the configuration graph for an $\mathsf{NL}$-machine on $x$ can be done not just in $\text{poly}(n)$ time but actually in $O(\log n)$ space. □

Subtlety #3: Just a reminder, we mentioned that even NP-completeness can and perhaps should be defined in terms of log-space reductions. The subtlety we want to mention is that *composing* two log-space reductions in log-space is not trivial; in fact, it's a bit tricky. The point is, to output $R_2(R_1(x))$, you can't just run $R_1(x)$, write down its output on your work tape, and then run $R_2$: this output may be polynomially long. We leave it as an exercise to show how you can nevertheless compose log-space reductions.

One more comment: STCON is obviously a fundamental, important problem, so the fact that it is complete for NL means that NL is a fundamental, important class. One also defines the equally fundamental and important problem "USTCON": checking if $T$ is reachable from $S$ in an *undirected* graph. Is it any easier? Theorem 1.2, proved in 1970, tells us that STCON can be done in $O(\log^2 n)$ space. In 2004, Trifonov showed USTCON can be done in space $O(\log n \log\log n)$ and Reingold[1] simultaneously improved this:

**Theorem 2.5.** ***(Reingold's Theorem.)*** *USTCON* $\in \mathsf{L}$.

We will prove this outstanding theorem later in the course.

## 2.1 Savitch's Theorem

We now prove Theorem 1.2. Note that in particular it shows $\mathsf{NL} \subseteq \mathsf{DSPACE}(\log^2 n)$; i.e., STCON can be solved in deterministic $O(\log^2 n)$ space. Note that depth-first and breadth-first search for STCON use *linear* space (and polynomial time). We will see an algorithm using $O(\log^2 n)$ space (and $n^{O(\log n)}$ time!).

*Proof.* Suppose we have an NTM $M$ deciding $L$ using space $O(s(n))$. Write $m = O(s(n))$ for the number of bits required to write down a configuration. WOLOG $M$ has a unique accepting configuration $C_{\text{acc}}$. Given input $x$, let $C_{\text{start}}$ denote the starting configuration. We know that if $M$ accepts $x$ it does so in at most $2^m$ time. So our goal is to determine if $C_{\text{acc}}$ is reachable from $C_{\text{start}}$ in at most $2^m$ steps.

The idea is, "guess the midpoint configuration and recurse". Specifically, consider the following decision problem: $\text{REACH}(C, C', i)$, which is true if one can reach $C'$ from $C$ in the configuration graph in at most $2^i$ steps. For $i \geq 1$, we can solve it recursively by enumerating over all possible midpoint configurations $C''$ and checking if we ever have both $\text{REACH}(C, C'', i-1)$ and $\text{REACH}(C'', C', i-1)$. Writing down each configuration requires $m$ bits. The key idea is to *reuse space* when performing the two recursive calls. Note that we can solve $\text{REACH}(C, C', 0)$ in $O(m)$ space (we're just checking if $C = C'$ or $C$ is connected to $C'$), and we can solve $\text{REACH}(\cdot, \cdot, i)$ using $O(m)$ space plus the space required for $\text{REACH}(\cdot, \cdot, i-1)$. To find the final answer we just compute $\text{REACH}(C_{\text{start}}, C_{\text{acc}}, m)$. This requires $O(m^2)$ space. (There is maybe an additional $O(\log m)$ space for keeping track of the overall position in the recursion.) □

There is a "logicky" version of this proof with a very nice consequence: a natural problem for PSPACE. A "totally quantified boolean formulas" (TQBFs) is one with $\exists$ and $\forall$ quantifiers, where there are no "free" variables. We write $\text{size}(\Phi)$ for the number of bits it takes to literally write $\Phi$ down. We do not assume these TQBFs are in "prenex form", although you should check (exercise) that one can always convert a $\Phi$ to an equivalent TQBF in prenex form in polynomial time. Here is an exercise:

[1] Omer Reingold; there's another Reingold.

**Exercise 2.6.** *Let* TQBF *be the problem of deciding whether a given input TQBF is true. Then* TQBF $\in$ PSPACE*.*

**Theorem 2.7.** TQBF *is also* PSPACE*-hard, hence* PSPACE*-complete.*

*Proof.* The proof is very similar to Savitch's Theorem; we just use "logic". Suppose we have an NTM $M$ deciding $L$ using polynomial space; write $m = O(n^k)$ for the number of bits required to write down a configuration. Note that given input $x$, it's easy to write down a size-$O(m)$ (unquantified) formula $\varphi_x$ such that such that $\varphi_x(C, C')$ iff $C$ is connected to $C'$ in the configuration graph. Our goal here is to write down a TQBF $\Phi$ which is true iff $C_{\text{start}}$ can reach $C_{\text{acc}}$ in the configuration graph in at most $2^m$ steps. [[Draw tableau.]]

We do this by writing a TQBF $\Phi_i(C, C')$ which is true if $C$ can reach $C'$ in at most $2^i$ steps. We take $\Phi_0(C, C')$ to be $(\varphi_x(C, C') \vee (C = C'))$. In general, the naive approach would be to write:

$$\Phi_i(C, C') = \exists C''(\Phi_{i-1}(C, C'') \wedge \Phi_{i-1}(C'', C')).$$

But this will cause size$(\Phi_i)$ to be at least double that of size$(\Phi_{i-1})$, giving an exponential (in $m$) size TQBF for $\Phi_m$. Again, the trick is to "reuse" space, making only one "call" to $\Phi_{i-1}$. We write

$$\Phi_i(C, C') = \exists C''\forall C_1, C_2 \left[((C_1 = C' \wedge C_2 = C'') \vee (C_1 = C'' \wedge C_2 = C')) \Rightarrow \Phi_{i-1}(C_1, C_2)\right].$$

Note that we can convert this to properly use just $\neg$, $\wedge$, and $\vee$ and still have only one "call" to $\Phi_{i-1}$. Hence we get size$(\Phi_i)$ = size$(\Phi_{i-1}) + O(m)$, so finally size$(\Phi_m(C_{\text{start}}, C_{\text{acc}})) = O(m^2)$; i.e., we've produced a poly-size TQBF deciding $M(x)$. □

## 2.2 Immerman-Szelepcsényi

For this theorem, let's just do the case NL = coNL; the statement for general space bounds $s(n)$ is no harder.

Since STCON is NL-complete, we get that $\overline{\text{STCON}}$ is coNL-complete, where $\overline{\text{STCON}}$ is the problem: Given a directed graph $G$ and vertices $s$ and $t$, is $t$ *unreachable* from $s$. Our task is then to show that this problem is in NL. This is pretty weird if you think about it; we want a nondeterministic log-space algorithm for checking that $s$ *cannot* reach $t$.

**Idea 1:** Suppose you magically know the exact *number* of nodes in $G$ reachable from $s$, call it $R$. Then you can solve $\overline{\text{STCON}}$ in NL. Here is the algorithm:

- For each $v \in V$,
  - *Guess* if it's reachable from $s$.
  - If you guess also guess and check the path to $v$. (Doable in log-space.) Reject if your path guess errs.
  - If $v = t$, *reject*.
  - Else, increment `counter`.
- If `counter` equals $R$, *accept*. Else reject.

It's easy to verify that this is log-space. Now how could you possibly have a sequence of guesses leading to an accept? This could only happen if: a) `counter` indeed ends up at $R$; b) hence you correctly guessed the reachability/path for exactly $R$ nodes; c) you saw all reachable nodes; d) you did *not* see $t$.

Great. So we're done — if only we could compute $R$. If we could compute it deterministically, that would be great. But really, the only thing we have to do is compute it "in NL". I.e., we need a log-space nondeterministic computation where each branch either rejects or computes the correct $R$. This part is somewhat harder!

**Idea 2:** Let $R(i)$ be the number of vertices reachable from $s$ in at most $i$ steps. We'll try to compute $R(i)$ from $R(i-1)$ in NL. Note that once we have done this we can discard $R(i-1)$ and reuse all space.

Of course, $R(0)$ is just 1. Here's how we do $R(i+1)$:

- Initialize $R(i+1) = 0$.
- For each $v \in V$,
  - Guess if $v$ is reachable from $s$ in at most $i+1$ steps.
  - If you guessed yes, guess the path of length at most $i+1$. (Reject if you screw up.) Increment $R(i+1)$.
  - If you guessed no, "visit all $R(i)$ nodes reachable from $s$ in $\leq i$ steps" and reject if you visit $v$ or you visit a node connected to $v$.

The point here is that we can do the "visit all $R(i)$ nodes" step in a way similar to how we solve $\overline{\text{STCON}}$ assuming $R = R(n)$. The idea is that you guess for each $w \in V$ whether it can be reached in $\leq i$ steps. If you guess yes, you verify this with a guessed path; you reject if $w$ is $v$ or is connected to $v$; *and*, you increment a counter. Having done this for all $w$, you reject if the counter differs from $R(i)$.

It remains to make the easy check that this second algorithm computing $R(i+1)$ from $R(i)$ indeed uses only $O(\log n)$ space.

## 3 Circuits and Parallelism

Recall our standard definition of a circuit:

**Definition 3.1.** *A* circuit *is a dag with $n$ input wires, binary AND and OR gates, unary NOT gates, and one (or more, sometimes) output wires. It computes a function $f : \{0,1\}^n \to \{0,1\}$ in the obvious way. Its* size *is the number of AND and OR gates; its* depth *is the number of AND and OR gates along the longest input-to-output path.*

Note that NOT gates can be pushed all the way to the input wires, by De Morgan's laws, and they are typically not counted toward size/depth.

**Definition 3.2.** *A* family of circuits *is a list $C_1$, $C_2$, ..., where $C_n$ is an $n$-input circuit. We say it computes a language $L$ if $x \in L \Leftrightarrow C_{|x|}(x) = 1$. For reasons explained on Homework 1, the set of languages decided by "polynomial-size circuits" — meaning size$(C_n) \leq \text{poly}(n)$ — is denoted* $\mathsf{P}$*/poly.*

Recall that in Lecture 3 we saw:

**Theorem 3.3.** $\mathsf{P} \subseteq \mathsf{BPP} \subseteq \mathsf{P}$*/poly.*

Subtlety #4: Circuit families are a bit funny because they are "non-uniform", meaning they can "have a different algorithm for each length $n$". The proof of $\mathsf{BPP} \subseteq \mathsf{P}$/poly took full advantage of this fact. Note that this is very unrealistic: an $O(1)$-size circuit family solves the Unary Halting Problem: just let $C_n$ compute the constant 1 iff $n$ is the encoding of a TM that halts.

To get around this, we sometimes but not always restrict to families that are in some sense "based on one algorithm":

**Definition 3.4.** *A circuit family $(C_n)$ is* (log-space) uniform *if there is a log-space DTM which, on input $1^n$, outputs $C_n$.*

The following is easy; the only trick is using that CIRCUIT-VAL is $\mathsf{P}$-complete.

**Proposition 3.5.** *$L \in \mathsf{P}$ iff $L$ is decidable by a uniform family of poly-size circuits.*

As a remark, we therefore have:

$$\mathsf{NP} \neq \mathsf{P} \text{ iff } \mathsf{NP} \text{ does not have uniform poly-size circuit families.}$$

In fact, the following conjecture is widely believed:

**Conjecture 3.6.** $\mathsf{NP}$ *does not even have nonuniform poly-size circuit families.*

Although it is formally weaker than $\mathsf{NP} \neq \mathsf{P}$, the sentiment is that the conjectures are roughly the same: It's hard to imagine how nonuniformity could help so much; it's not like anyone's ever built a circuit that was surprisingly better than an algorithm at solving SAT. Indeed, if you put a gun to a complexity theorist's head and ask him/her for the best possible approach to $\mathsf{NP} \neq \mathsf{P}$, you'd probably get the suggestion, "Prove $\mathsf{NP} \not\subseteq \mathsf{P}$/poly by some kind of combinatorial means."

Not that we're so great at proving circuit lower bounds! In 1949, Shannon proved:

**Theorem 3.7.** *Almost all functions $f : \{0,1\}^n \to \{0,1\}$ require circuits of size $(1 - o(1))\frac{2^n}{n}$. Also, every function has a circuit of size $4\frac{2^n}{n}$.*

(Getting size $O(n2^n)$ is trivial using DNFs. Also, Lupanov improved the 4 to a $1 + o(1)$.)

Frustratingly, although almost every function requires exponential-size circuits, and even SAT — which is in $\mathsf{NP}$ — seems to too, here is the best we know:

**Theorem 3.8.** *(Iwama-Lachish-Morizumi-Raz). There is an explicit language in $\mathsf{P}$ which requires circuits of size $\geq 5n - o(n)$. We do not know any language even in $\mathsf{NP}$ requiring higher circuit complexity.*

Similarly depressing:

**Fact:** It is not known whether $\mathsf{NEXP} \subseteq \mathsf{P/poly}$.

### 3.1 Parallel time

Given that uniform P/poly is the same as P, is there anything to be gained by studying it as a complexity class? The answer is yes, by looking not just at size but at *depth*. At a very rough level, it is reasonable to equate:

$$\textit{circuit depth} \approx \textit{parallel time}.$$

This is somewhat reasonable, as one can imagine making each gate into a "parallel processor". It's quasi-reasonable to imagine poly($n$) processors; the goal then is to get depth which is *logarithmic*, or perhaps polylogarithmic. Here is a complexity class definition:

**Definition 3.9.** *For $k \in \mathbb{N}$, the class $\mathsf{NC}^k$ is the set of all languages decidable by a* uniform *family of circuits of polynomial size and depth $O(\log^k n)$. We also define $\mathsf{NC} = \cup_{k \in \mathbb{N}} \mathsf{NC}^k$.*[2]

The first reasonable class here is $\mathsf{NC}^1$. (However, the class $\mathsf{NC}^0$ is still frequently referred to; check that a function is in $\mathsf{NC}^0$ if its value depends on only $O(1)$ many input bits.)

Note that $\mathsf{NC} \subseteq \mathsf{P}$. Unsurprisingly we can't prove $\mathsf{NC} \neq \mathsf{P}$ although this is widely believed. It used to be contended sometimes that NC is a reasonable notion of what can be "parallelized efficiently", but people don't seem to claim this much any more. It is perhaps reasonable to say that languages is $\mathsf{NC}^1$ and $\mathsf{NC}^2$ can be parallelized efficiently. On the other hand, it *is* widely agreed that anything *not* in NC cannot be parallelized efficiently. Of course, we don't know any languages in P that are definitively not in NC, but assuming $\mathsf{P} \neq \mathsf{NC}$, any P-complete language is not in NC.

**Proposition 3.10.** *$\mathsf{NC}^k$ is closed under log-space reductions for $k \geq 2$.*

Hence if you're looking for "inherently sequential" problems, look to P-complete ones: CIRCUIT-VALUE, LINEAR-PROGRAMMING, "DEPTH-FIRST-SEARCH" (appropriately decision-versioned)... There is one mysterious intermediate problem, a la FACTORING's status for P vs. NP:

**Fact:** Computing the GCD of two numbers is not known to be in NC and is not known to be P-complete.

### 3.2 Deeper into $\mathsf{NC}^1$ and $\mathsf{NC}^2$

. $\mathsf{NC}^1 \subseteq \mathsf{L}$: was on the homework.

. Sketch proof that Matrix Multiplication (over $\mathbb{F}_2$ say) is in $\mathsf{NC}^1$. In fact, it's a formula!

**Theorem 3.11.** *$\mathsf{NC}^1$ equals the set of language decided by (uniform families of) poly-sized* formulas.[3]

. Part of this was on the homework. $\mathsf{NC}^1$ circuit to formula: recursive, starting at top. (Need to do it in log-space, technically.) Formula to $\mathsf{NC}^1$: rebalancing. Take out large subtrees.

. $\mathsf{NL} \subseteq \mathsf{NC}^2$: Need to show STCON is in it. Raise the adj. matrix to power of $n$. ($\log n$-depth tree of multiplications.)

[2] NC stand's for "Nick [Pippenger]'s Class", believe it or not.

[3] The size of a formula is the number of leaves, which are literals.