CAS CS 535: Complexity Theory

Lecturer: Mark Bun — Fall 2023

Lecture Notes 7:

Savitch's Theorem, $\mathbf{PSPACE}$, $\mathbf{PSPACE}$-Completeness

Reading.

- Arora-Barak § 4.2

Last time: Relativization barrier, space complexity

Space Complexity Classes

$$\mathbf{SPACE}(S(n)) = \{L \subseteq \{0,1\}^* \mid L \text{ is decidable by a deterministic TM in space } O(S(n))\}.$$
$$\mathbf{NSPACE}(S(n)) = \{L \subseteq \{0,1\}^* \mid L \text{ is decidable by an NTM in space } O(S(n))\}.$$

Given a (N)TM M and input x, define the configuration graph $G_{M,x}$ as follows. The graph has a vertex for every triple of the form

$$C = (\text{state}, \text{head locations}, \text{work tape contents}),$$

and an edge $C \to C'$ if there exists a transition that takes C to C'. For example, if [pretending we're working with a one-tape TM for simplicity] the transition functions of an NTM specify $\delta_0(q_7, 0) = (q_4, 0, L)$ and $\delta_1(q_7, 0) = (q_3, 1, R)$, then vertex $C = (q_7, 3, 10010)$ has outgoing edges to $C'_0 = (q_4, 2, 10010)$ and $C'_1 = (q_3, 4, 10110)$.

Without loss of generality, we can assume that M erases its worktapes and restores its heads to the left after halting, so there are unique accept and reject vertices, C_{acc} and C_{rej}, in the configuration graph.

The configuration graph has the following useful property.

$$M(x) \text{ accepts} \iff \text{there exists a path from } C_{\text{start}} \text{ to } C_{\text{acc}} \text{ in } G_{M,x}.$$

Last time, we stated and proved half of the following result relating time and space complexity classes:

Theorem 1. *For space-constructible $S(n)$, we have*

$$\begin{aligned}\mathbf{DTIME}(S(n)) \subseteq \mathbf{NTIME}(S(n)) &\subseteq \mathbf{SPACE}(S(n)) \\ &\subseteq \mathbf{NSPACE}(S(n)) \subseteq \mathbf{DTIME}(2^{O(S(n))}) =: \bigcup_{c=1}^{\infty} \mathbf{DTIME}(2^{cS(n)}).\end{aligned}$$

Proof of $\mathbf{NSPACE}(S(n)) \subseteq \mathbf{DTIME}(2^{O(S(n))})$. Let N be an NTM running in space $S(n)$. Consider the following deterministic TM:

On input x:

1. Construct the configuration graph $G_{N,x}$
2. Run BFS to determine if there is a path from C_{start} to C_{acc} in $G_{N,x}$. Accept iff this is the case.

We can reason about the time needed to construct $G_{N,x}$ as follows. First, the size of each configuration is $O(S(|x|))$. This means that the number of possible configurations, i.e., the size of the graph, is $2^{O(S(n))}$. Therefore, one can materialize the whole graph and run BFS on it in time $2^{O(S(n))}$. □

1 Savitch's Theorem

Theorem 2. *For all space-constructible $S(n)$, we have* $\mathbf{NSPACE}(S(n)) \subseteq \mathbf{SPACE}(S(n)^2)$.

Proof. Let L be decidable by an NTM N running in space $S(n)$. Then

$$\begin{aligned} x \in L &\iff \exists \text{ a path from } C_{\text{start}} \text{ to } C_{\text{acc}} \text{ in } G_{N,x} \\ &\iff \exists \text{ a path from } C_{\text{start}} \text{ to } C_{\text{acc}} \text{ in } G_{N,x} \text{ of length } \le M, \end{aligned}$$

where $M = 2^{O(S(n))}$ is the number of vertices in $G_{N,x}$. (This holds because we can remove loops from any longer path.)

The main idea is to determine whether such a path exists by a divide-and-conquer algorithm. Specifically, we'll design a recursive algorithm called Reach with the following property:

$$\mathsf{Reach}(u, v, i) = \begin{cases} \text{YES} & \text{if } \exists \text{ a path from } u \text{ to } v \text{ in } G_{N,x} \text{ of length } \le 2^i \\ \text{NO} & \text{otherwise.} \end{cases}$$

That is, Reach determines whether there is a path from vertex u to v in the configuration graph with length at most 2^i. Note that $x \in L \iff \mathsf{Reach}(C_{\text{start}}, C_{\text{accept}}, \log M) = \text{YES}$.

So now let's describe the algorithm $\mathsf{Reach}(u, v, i)$:

<u>Base case:</u> Suppose $i = 0$. If $u = v$ or $u \to v$, return YES. Otherwise, return NO.

<u>Recursive case:</u> For each vertex z in $G_{N,x}$:

- Compute $\mathsf{Reach}(u, z, i-1)$
- Compute $\mathsf{Reach}(z, v, i-1)$ (using the same space)
- Output YES iff both runs say YES

Output NO. □

This works because of the following observation: For vertices u, v, there exists a path of length 2^i from u to v iff there exists a "midpoint" z such that there are paths from u to z and from z to v both of length at most 2^{i-1}.

To calculate the space usage, let $S_{N,i}$ denote the space consumption of Reach when the underlying NTM is N and the length parameter is i. Then

$$S_{N,i} = \underbrace{S_{N,i-1}}_{\text{recursive call}} + \underbrace{O(\log M)}_{\text{counter over } z}.$$

Unrolling the recursion gives us $S_{N,\log M} = O(\log^2 M) = O(S(n)^2)$.

2 Some Space Classes

$$\mathbf{PSPACE} = \bigcup_{c=1}^{\infty} \mathbf{SPACE}(n^c)$$
$$\mathbf{NPSPACE} = \bigcup_{c=1}^{\infty} \mathbf{NSPACE}(n^c) = \mathbf{PSPACE}$$
$$\mathbf{L} = \mathbf{SPACE}(\log n)$$
$$\mathbf{NL} = \mathbf{NSPACE}(\log n).$$

The class we'll start understanding today is $\mathbf{PSPACE}$, which is big. Pretty much any reasonable-looking algorithm solves a problem in $\mathbf{PSPACE}$.

Example 3. $\mathsf{SAT} \in \mathbf{SPACE}(n) \subseteq \mathbf{PSPACE}$.

This is because we can very efficiently recycle space to try all possible satisfying assignments:
On input φ:
For each candidate assignment u:
  Evaluate $\varphi(u)$, accept if $= 1$. Erase work tape.
Reject.

3 $\mathbf{PSPACE}$-Completeness

The open question of the day is $\mathbf{P} \stackrel{?}{=} \mathbf{PSPACE}$. The answer seems to be no; $\mathbf{P} = \mathbf{PSPACE}$ would, in particular, imply $\mathbf{P} = \mathbf{NP}$. But we are still far from ruling this out.

As with $\mathbf{NP}$, a useful starting point for studying this and other questions is the notion of $\mathbf{PSPACE}$-completeness.

Definition 4. A language L is $\mathbf{PSPACE}$-hard if $A \leq_p L$ for every $A \in \mathbf{PSPACE}$. (A is poly-time reducible to L.)

L is $\mathbf{PSPACE}$-complete if $L \in \mathbf{PSPACE}$ and L is $\mathbf{PSPACE}$-hard.

3.1 A $\mathbf{PSPACE}$-Complete Problem

The canonical $\mathbf{PSPACE}$-complete problem is a generalization of SAT defined in terms of "quantified Boolean formulas." To build up to these, recall what it means for a formula to be satisfiable:

$$(x \vee y) \wedge z \in \mathsf{SAT} \iff \Psi := \exists x \exists y \exists z (x \vee y) \wedge z \text{ is "true".}$$

The formula Ψ is a "quantified Boolean formula", i.e., a formula where every variable is bound by a quantifier. In general, a (fully) quantified Boolean formula might mix existential and universal quantifiers.

Example 5. Let $\Psi = \exists x \forall y \exists z (x \vee y) \wedge z$. The QBF Ψ evaluates to "true." To see why this is the case, set x to 1. Then for either choice of $y \in \{0, 1\}$, setting $z = 1$ makes the formula true.

In general, a QBF looks like

$$\Psi = Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \varphi(x_1, \ldots, x_n)$$

where each quantifier $Q_i = \exists$ or $\forall$.

Game view: Since every variable in a QBF is bound, it has a definite truth value (either true or false). A helpful way of thinking about determining this value is through a two player game. Suppose for simplicity that we have a QBF of the form

$$\Psi = \exists x_1 \forall y_1 \exists x_2 \forall y_2 \dots \exists x_n \forall y_n \varphi(x_1, \dots, x_n, y_1, \dots, y_n).$$

Player 1, the "Existential" player, has the goal of setting the x_i's to make the formula φ evaluate to "true".

Player 2, the "Universal player" (or adversary), has the goal of setting the y_i's to make the formula evaluate to "false."

The QBF Ψ overall is "true" iff Player 1 has a winning strategy: No matter what Player 2 does in setting the y variables, Player 1 can come up with a way to set the x variables to make the underlying formula true.

Now you try: What is the truth value of the QBF $\exists x_1 \forall y_1 \exists x_2 \forall y_2 (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$?

Now for our $\mathbf{PSPACE}$-complete problem:

Definition 6.

$$\mathsf{TQBF} = \{\Psi \mid \Psi \text{ is a true quantified Boolean formula}\}.$$

Theorem 7. TQBF *is* $\mathbf{PSPACE}$*-complete.*

Proof. As usual, there are two things to prove: First, that $\mathsf{TQBF} \in \mathbf{PSPACE}$, and second, that it is $\mathbf{PSPACE}$-hard.

$\mathsf{TQBF} \in \mathbf{PSPACE}$. We design a (space-recycling) recursive algorithm A as follows. Consider an input of the form $\Psi = Q_1 x_1 Q_2 x_2 \dots Q_n x_n \varphi(x_1, \dots, x_n)$.
Base case: If $n = 0$, then φ is a constant, so just output it.
Recursive case:
If $Q_1 = \exists$:

- Run $A(\Psi|_{x_1=0})$
- Run $A(\Psi|_{x_1=1})$
- Accept if either run accepts.

If $Q_1 = \forall$:

- Run $A(\Psi|_{x_1=0})$
- Run $A(\Psi|_{x_1=1})$
- Accept if both runs accept.

Correctness holds by induction on the number of quantifiers of the formula.

To analyze the space usage, let $S_{n,m}$ denote the space consumption when n is the number of variables and $m = |\varphi|$. Then we have the recurrence

$$S_{0,m} = O(m), \qquad S_{n,m} = S_{n-1,m} + O(m)$$

and so $S_{n,m} = O(mn)$.

TQBF **is PSPACE-hard.** We need to show that for every language $L \in \mathbf{PSPACE}$, we have $L \leq_p \mathsf{TQBF}$. Let L be such a language and let M decide L in (polynomial) space $S(n)$. Our goal is to, in poly-time, convert an instance x into a QBF Ψ such that $M(x) = 1 \iff \Psi \in \mathsf{TQBF}$.

Our first idea will be to define a two-player game such that Player 1 has a winning strategy in this game iff $M(x) = 1$. Then we'll formalize this game into a QBF.

Recall from our discussion of configuration graphs that

$$M(x) = 1 \iff \text{there exists a path from } C_{\text{start}} \text{ to } C_{\text{acc}} \text{ in } G_{M,x}$$

Consider the following (informal) game:
Player 1: The goal is to show that there exists a path (of length $2^{O(S(n))}$) from C_{start} to C_{acc}.
Player 2: The goal is to show that there is *no* such path.
When it's Player 1's turn to move, they'll pick a vertex v that's on the alleged path from C_{start} to C_{acc}.

When it's Player 2's turn, they'll issue a "challenge" to recurse either to the left or right of v, i.e., to force Player 1 in the next round to either exhibit a vertex on the path from C_{start} to v or from v to C_{acc} And so on and so forth...

The point of this game is that Player 1 has a winning strategy iff there indeed exists a path from C_{start} to C_{acc} in $G_{M,x}$.

Now let's turn this intuitive description of a game into a QBF. Let $m = O(S(n))$ be the number of bits needed to encode one configuration (vertex of the configuration graph). The idea will be to recursively construct formulas of the form $\Psi_i(C, C')$, for $C, C' \in \{0,1\}^m$, such that

$$\Psi_i(C, C') \in \mathsf{TQBF} \iff \exists \text{ a path of length } \leq 2^i \text{ from } C \text{ to } C'.$$

The final formula we want will be $\Psi_m(C_{\text{start}}, C_{\text{acc}})$.

Base case: If $i = 0$, we use the proof of the Cook-Levin Theorem to encode the question of whether there is a transition from C to C' as an unquantified formula $\Psi_0(C, C')$.

Recursive case: As a first attempt, we might want to define $\Psi_i(C, C') = \exists v \Psi_{i-1}(C, v) \wedge \psi_{i-1}(v, C')$. The problem with this is that the size of the formula doubles with each call, so we'd end up with an exponentially long formula.

A better idea is introduce auxiliary variables to capture an equivalent condition without blowing up the formula size. One way to do this is to define

$$\Psi_i(C, C') = \exists v \forall x, y \qquad (x = C \wedge y = v) \vee (x = v \wedge y = C') \implies \Psi_{i-1}(x, y).$$

Note that one can unpack the $\implies$ connective using ORs and negations, and "push all quantifiers" in Ψ_{i-1} to the left of the whole expression.

The time it takes to generate each formula is polynomial in the size, which by induction, is at most $|\Psi_i| \leq O(m^2)$. $\square$