CS 6810: Theory of Computing **Fall 2021**

Lecture 7: Sept 16, 2021

Lecturer: Eshan Chattopadhyay *Scribe: Carlos Martínez*

1 PSPACE-complete Problems

It is widely believed that P $\neq$ PSPACE, as otherwise P = NP. Just as NP-complete problems are the "hardest" of all problems in NP, PSPACE-complete problems are the "hardest" problems in PSPACE.

Definition 1.1. *A language A is PSPACE-hard if for every $L \in$ PSPACE, $L \leq_p A$. If in addition $A \in$ PSPACE, then A is PSPACE-complete.*

Note that in the above definition, $\leq_p$ denotes a polynomial time Karp reduction. One may wonder if perhaps using polyspace reductions can be used instead. It is not hard to see that polynomial space reduction would be too powerful. In fact, it is not hard to see that every problem in PSPACE (except $\emptyset$ and $\{0,1\}^*$) would be PSPACE-complete (if we use PSPACE reductions).

We now introduce our first PSPACE-complete problem. To do this, we first need the notion of a *quantified Boolean formula*, wherein variables are *quantified* with $\exists$ (i.e., exists) and $\forall$ (i.e., for all) quantifiers. A quantified Boolean formula has the form

$$Q_1x_1Q_2x_2\dots Q_nx_n\phi(x_1,x_2,\dots,x_n),$$

where $Q_i \in \{\exists,\forall\}$ and $\phi(\cdot)$ is an unquantified Boolean formula. Note that if all variables in the formula are quantified, then such a formula is either TRUE or FALSE. As a first example, suppose we are given a quantified Boolean formula in which $Q_i = \exists$ for all i. Then, we are essentially asking whether $\phi(\cdot)$ is satisfiable (i.e., we recover the SAT problem). As a second example, suppose $Q_i = \exists$ if i is odd and $Q_i = \forall$ if i is even. Then, we can think of the quantified Boolean formula as asking whether there is a robust winning strategy for the first player making a move in a two-player game.

We define the language TQBF as the set of TRUE quantified Boolean formulas.

Theorem 1.2. *TQBF is PSPACE-complete.*

Proof. We first need to show TQBF $\in$ PSPACE. Let $\psi = Q_1x_1Q_2x_2\dots Q_nx_n\phi(x_1,x_2,\dots,x_n)$ be a quantified Boolean formula on n variables and let l denote the size of ϕ. Let A be a recursive algorithm defined as follows. If $n = 0$, the formula can be evaluated in $O(l)$ space. If $n > 0$, drop the quantifier Q_1 and make two recursive calls. In the first call, let $x_1 = 0$ everywhere in $\phi(\cdot)$. In the second call, let $x_1 = 1$ everywhere in $\phi(\cdot)$. If $Q_1 = \exists$, algorithm A returns TRUE if and only if at least one of the recursive calls is TRUE. If $Q_2 = \forall$, algorithm A returns TRUE if and only if both of the recursive calls are TRUE. Since each recursive call reuses space, A uses polynomial space. More formally, A uses space $S(n,l) = S(n-1,l) + O(l) = O(nl) = \text{poly}(n)$.

We now need to show that, for any $L \in$ PSPACE, $L \leq_p$ TQBF. Let $L \in$ PSPACE. Let M be a Turing machine that decides L in polynomial space. We use M to construct a quantified Boolean formula ψ that, given input $x \in \{0,1\}^*$, is true if and only if M accepts x. That is,

$$x \in L \iff \psi \in \text{TQBF}.$$

Let $G = G_{M,x}$ be the configuration graph of M on input x. Note that G has $2^{\text{poly}(n)} = 2^l = N$ nodes. Recall the following facts from the previous lecture:

1. $x \in L$ if and only if there exists a path from C_{start} to C_{accept} in G.
2. There exists a polynomial sized formula ϕ on $O(l)$ variables such that $\phi(C_i, C_j)$ is TRUE if and only if there exists an edge from C_i to C_j in G. Here, l is the number of bits needed to encode a state of M.

We will these facts to produce ψ. As a first approach, note that $x \in L$ if and only if there exist $C_1, \cdots, C_K$ such that

$$\phi(C_{\text{start}}, C_1) \wedge \phi(C_1, C_2) \wedge \dots \phi(C_K, C_{\text{accept}}).$$

Unfortunately, this requires $O(N)$ space. As a second approach, we recursively build a quantified Boolean formula $\psi_K(C_i, C_j)$ that is TRUE if and only if there exists a path of length $\leq 2^k$ from C_i to C_j in G. Then, the goal is to evaluate $\psi_l(C_{\text{start}}, C_{\text{accept}})$. We let $\psi_0 = \phi$ as in Fact 2. For $0 < k \leq l$, we let $\psi_k(C_i, C_j) = \exists C_q \psi_{k-1}(C_i, C_q) \wedge \psi_{k-1}(C_q, C_j)$. Note however that the formulas ψ_k are twice the size of the formulas ϕ_{k-1}. Upon unrolling all formulas, we recover our first approach using $O(N)$ space.

The crucial observation of why our second approach fails is that it does not use the full power of quantified Boolean formulas: it only uses $\exists$ quantifiers. We can fix this by letting

$$\psi_k(C_i, C_j) = \exists C_q \forall C' \forall C'' \left(\left((C' = C_i \wedge C'' = C_q) \vee \left(C' = C_q \wedge C'' = C_j\right) \right) \implies \psi_{k-1}(C', C'') \right)$$

Note that $|\psi_k| = \text{poly}(n) + |\psi_{k-1}|$, and so $|\phi_l| = l \cdot \text{poly}(n) = \text{poly}(n)$. □

Since PSPACE = NPSPACE, we also have that TQBF is NPSPACE-complete.

2 Sublinear Space Complexity

We now consider sublinear, in particular logarithmic space complexity. Define L = DSPACE($\log n$) and NL = NSPACE($\log n$).

Definition 2.1. *$f : \{0,1\}^* \to \{0,1\}^*$ is an implicitly logspace computable function if the languages $L_f = \{(x, i) : f(x)_i = 1\}$ and $L'_f = \{(x, i) : i \leq |f(x)|\}$ are in L.*

That is, f is computable in space $S(n)$ if on input (x, i), there is a space $S(n)$ algorithm that outputs $f(x)_i$, the ith bit of $f(x)$, provided $i \leq |f(x)|$.

Definition 2.2. *Language A is logspace reducible to language B, denoted $A \leq_l B$, if there is a function $f : \{0,1\}^* \to \{0,1\}^*$ that is implicitly logspace computable and, for every $x \in \{0,1\}^*$, $x \in A$ if and only if $f(x) \in B$.*

It not hard to prove that logspace reductions are transitive (see Lemma 4.15 in Arora and Barak).

Observe that L $\subseteq$ NL $\subseteq$ P. The second containment follows from the fact that we can brute force the configuration graph of a Turing machine M for a language in NL on input x in time $2^{O(\log n)} = O(n)$. It is conjectured that L = NL but NL $\subset$ P, that is there are polynomial time solvable problems that require more than logarithmic space.

We now give an NL-complete language. Let

$$\text{PATH} = \{(G, s, t) : G \text{ is a directed graph with a path from } s \text{ to } t\}.$$

Theorem 2.3. *PATH is NL-complete.*

Proof. We first claim PATH $\in$ NL. To see this, note that a nondeterministic machine can take a nondeterministic walk starting at s, maintaining a counter of how many steps it has taken, the index of the vertex it is currently at, and using nondeterminism to select a neighbor of the current vertex to go next. The machine accepts if and only if the walk reaches the end node t in $n-1$ steps, otherwise it rejects. Note that this machine only needs $O(\log n)$ space, namely to keep track of the number of steps it has taken and the identity of the current vertex.

Now, we need to show that for any $A \in$ NL, $A \leq_l$ PATH. An easy gadget is the configuration graph G of machine M for a on input x! Namely, $x \in A$ if and only if there is a path from C_{start} to C_{accept} in G. It remains to show the adjacency matrix of G can be implicitly computed by a logspace reduction. Note that we can deterministically examine any two states C, C' of G and check whether C' is one of the (at most two) configurations that can follow C according to the transition functions of M. $\square$

We now define the undirected version of the PATH problem.

$$\text{UPATH} = \{(G, s, t) : G \text{ is a undirected graph with a path from } s \text{ to } t\}.$$

A highly sophisticated deterministic logspace algorithm was developed by Reingold [1] for UPATH.

Theorem 2.4 (Reingold [1])**.** *UPATH $\in$ L.*

This provides evidence towards the L$\overset{?}{=}$NL conjecture, with hope that perhaps one design such an algorithm for PATH.

References

[1] Omer Reingold. Undirected connectivity in log-space. *Journal of the ACM (JACM)*, 55(4):1–24, 2008.