CS 2401 - Introduction to Complexity Theory

Lecture #5: Fall, 2015

Lecturer: Toniann Pitassi

Scribe Notes by: Ken Hoover

1 Review - Space Bounded Languages

- Previously covered L, NL, PSPACE, $\mathsf{NPSPACE}$
  - A language $\mathcal{L}$ is in $\mathsf{SPACE}(s(n))$, where $s : \mathbb{N} \to \mathbb{N}$, iff there is a deterministic TM M deciding $\mathcal{L}$ such that, on any input of length n, M's tape head visits at most $c \cdot s(n)$ cells over all non-input tapes, where c is a constant
  - Similarly, $\mathcal{L} \in \mathsf{NSPACE}(s(n))$ iff there is a non-deterministic TM M deciding $\mathcal{L}$ such that, on any input of length n and for any possible sequence of configurations, M's tape head visits at most $c \cdot s(n)$ cells over all non-input tapes
  - Saw that $\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{PSPACE} \subseteq \mathsf{NPSPACE}$
- Introduced TQBF, the problem of determining the truth of quantified boolean formulas, i.e. boolean formulas of the form

$$Q_1 x_1 Q_2 x_2 \dots Q_n x_n \phi(x_1, x_2, \dots, x_n),$$

  where $Q_i \in \{\forall, \exists\}$
  - Showed that TQBF is complete for PH, the polynomial time hierarchy, and that for any level of the hierarchy, a version of TQBF where the number of quantifiers is restricted is complete
  - Showed that TQBF $\in \mathsf{PSPACE}$

2 TQBF is PSPACE-complete

We aready showed that TQBF $\in \mathsf{PSPACE}$ last class, by observing that we can express the validity of a TQBF using a recursive algorithm, where the maximum depth of the recursion is n, the number of variables quantified over, and the space used at each level is $O(m)$, where m is the size of the formula.

Continuing from this, we show that we can create a QBF from a TM M deciding a language $\mathcal{L}$ in space $s(n)$ and an input x, using a polynomial amount of space, which is true iff the accept configuration of M is reachable from the initial configuration of M on x.

Theorem 1 *TQBF is PSPACE-hard.*

Proof Suppose $\mathcal{L} \in \mathsf{PSPACE}$. We shall prove that there is a QBF $\phi_{\mathcal{L}}$ such that $x \in \mathcal{L} \Leftrightarrow \langle \phi_{\mathcal{L}}(x)$ is true$\rangle$. Suppose M is an $s(n)$-space TM that decides $\mathcal{L}$, where $s(n)$ is a polynomial. We can define the configuration graph $G_{M,n}$ of M on inputs of length n; this graph has size $2^{c \cdot s(n)}$, as shown last class. We next define $\phi^{M,n}(\mathbf{y}, \mathbf{y}')$, which is true iff $\mathbf{y}$ and $\mathbf{y}'$ encode configurations of M and running M for a single step from configuration $\mathbf{y}$ can yield configuraton $\mathbf{y}'$. Note that $\phi^{M,n}$ is poly-sized in n, due to locality of computation.

To determine whether or not we can reach an accept configuration from the initial configuration on $G_{M,n}$, we use the path-splitting algorithm for STCON. We first try to inductive define $\phi_i^{M,n}(\mathbf{y}, \mathbf{y}')$, which we want to be true iff there is a path of length at most 2^i from configuration $\mathbf{y}$ to configuration $\mathbf{y}'$, as follows:

$$\phi_0^{M,n}(\mathbf{y}, \mathbf{y}') = \left((\mathbf{y} = \mathbf{y}') \vee \phi^{M,n}(\mathbf{y}, \mathbf{y}')\right) \tag{1}$$
$$\phi_i^{M,n}(\mathbf{y}, \mathbf{y}') = \exists \mathbf{y}'' \left(\phi_{i-1}^{M,n}(\mathbf{y}, \mathbf{y}'') \wedge \phi_{i-1}^{M,n}(\mathbf{y}'', \mathbf{y}')\right). \tag{2}$$

The intuition behind this definition is, if there is a path of length at most 2^i from s to t in a graph, then there must be a vertex v on the path (possibly one of the end-points) such that there is a path from s to v and a path from v to t, where each of those paths has length at most 2^{i-1}.

However, this definition has a problem. Namely, each level of the induction doubles the size of the formula. So, in particular, if there are $s(n)$ levels, then after unwinding the induction, the formula will have length $O(2^{O(s(n))})$. The problem here is, we aren't fully using the power of quantification. If instead of encoding $\phi_{i-1}^{M,n}$ twice, we instead used quantification to reuse a single encoding, then the final formula would have length polynomial in n. Namely, if we define $\phi_i^{M,n}$ instead as

$$\phi_i^{M,n}(\mathbf{y}, \mathbf{y}') = \exists \mathbf{y}'' \forall \mathbf{z}_1 \forall \mathbf{z}_2 \left(\left((\mathbf{z}_1 = \mathbf{y} \wedge \mathbf{z}_2 = \mathbf{y}'') \vee (\mathbf{z}_1 = \mathbf{y}'' \wedge \mathbf{z}_2 = \mathbf{y}')\right) \Rightarrow \phi_{i-1}^{M,n}(\mathbf{z}_1, \mathbf{z}_2)\right), \tag{3}$$

then since each configuration can be written as a binary vector using $O(s(n))$ bits, we have

$$\left|\phi_i^{M,n}\right| = O(s(n)) + \left|\phi_{i-1}^{M,n}\right|. \tag{4}$$

And since $\left|\phi_0^{M,n}\right|$ is poly-sized in n, say $O(f(n))$, we have that $\left|\phi_i^{M,n}\right|$ is in $O(f(n) + i \cdot s(n))$, which remains polynomial in n so long as i is bounded above by a polynomial. In particular, $\left|\phi_{s(n)}^{M,n}\right|$ is poly-sized in n. Also note that, since the configuration graph has at most $2^{s(n)}$ vertices, any path connecting two vertices, without loops, must have length at most $2^{s(n)}$. So, if the new definition is correct, then since M only accepts an input x iff there is a path from the initial configuration of M on x to an accepting configuration, we'll have that M accepts x iff $\phi_{s(n)}^{M,n}$ is true on the initial configuration and some accepting configuration.

As for why this new definition remains correct, suppose it is true on inputs $\mathbf{y}$ and $\mathbf{y}'$. Then, in particular, there is some configuration $\mathbf{y}''$ such that $\phi_{i-1}^{M,n}(\mathbf{y}, \mathbf{y}'')$ and $\phi_{i-1}^{M,n}(\mathbf{y}'', \mathbf{y}')$ are true, following from the cases of $\mathbf{z}_1 = \mathbf{y} \wedge \mathbf{z}_2 = \mathbf{y}''$ and $\mathbf{z}_1 = \mathbf{y}'' \wedge \mathbf{z}_2 = \mathbf{y}'$. But then the old definition is true as well. Similarly, suppose the old definition is true on $\mathbf{y}$ and $\mathbf{y}'$. Then we know there is some $\mathbf{y}''$ such that $\phi_{i-1}^{M,n}(\mathbf{y}, \mathbf{y}'')$ and $\phi_{i-1}^{M,n}(\mathbf{y}'', \mathbf{y}')$ are true, and so in the second definition, when $\mathbf{z}_1 = \mathbf{y} \wedge \mathbf{z}_2 = \mathbf{y}''$ or $\mathbf{z}_1 = \mathbf{y}'' \wedge \mathbf{z}_2 = \mathbf{y}'$, we have that $\phi_{i-1}^{M,n}(\mathbf{z}_1, \mathbf{z}_2)$ is true. Thus the new definition is also true.

If we let $\varphi(\mathbf{y})$ be a boolean formula which is true iff configuration $\mathbf{y}$ is an accepting configuration – which exists and is poly-sized in $|\mathbf{y}|$ – and if we let $\mathbf{c}_0(x)$ be the initial configuration of M on input x, computable in polyspace, then we finally have that

$$x \in \mathcal{L} \Leftrightarrow \exists \mathbf{y} \left(\varphi(\mathbf{y}) \wedge \phi^{M,n}_{s(n)}(\mathbf{c}_0(x), \mathbf{y}) \right), \tag{5}$$

where the right hand side is a TQBF constructible in space and of size polynomial in n. Thus any language in PSPACE is reducible to an instance of TQBF.

Something to note about this proof; nowhere did we actually need the assumption that M was deterministic, i.e. that each vertex in the graph had out-degree 1. Thus, the proof also works to show that TQBF is $\mathsf{NPSPACE}$-hard, and thus that $\mathsf{PSPACE} = \mathsf{NPSPACE}$.

3 Results in Space Complexity

As it turns out, the STCON problem comes up very often in the analysis of space complexity. This makes sense; any language can be reduced to an STCON problem on the configuration graph of some TM deciding it. Savich's Theorem, relating non-deterministic space complexity to deterministic space complexity, is an example of using this fact.

Theorem 2 (Savich′s Theorem)

$$\mathit{NSPACE}(s(n)) \subseteq \mathit{SPACE}(s(n)^2)$$

Proof Sketch The rough idea is to show that STCON on an N node graph is solvable using $c \cdot \log^2 N$ space, in a deterministic way. Thus, since the configuration graph of any TM M using $c \cdot s(n)$ space has $2^{c \cdot s(n)}$ vertices, showing STCON on that graph is computable in $\log^2(2^{c \cdot s(n)}) = c^2 \cdot s(n)^2$ space puts the language in $\mathsf{SPACE}(s(n)^2)$.

Alternatively, this can be stated as the problem of finding the (s,t) entry of $(A+I)^N$, the adjacency matrix of the configuration graph plus the identity matrix, using deterministic $\log^2 N$ space; $A + I$ itself is the adjacency matrix of the configuration graph with an added self-edge for every vertex. Note that any entry (k, j) in $(A + I)^N$ is non-zero iff there is a path from k to j in the graph of length at most N; this follows from $(A + I)^N = A^N + A^{N-1} + \ldots + A + 1$, where for each entry (x, y) in each A^i, $A^i_{x,y}$ is non-negative, and zero iff there is no path of length exactly i from x to y.

Another result, this one rather surprising, is that $\mathsf{NL} = \mathsf{coNL}$. This is due indepenedently to Immerman and Szelepcsényi, and generalises to $\mathsf{NSPACE}(s(n)) = \mathsf{coNSPACE}(s(n))$ for $s(n) \geq \log n$ via padding.

Theorem 3 (Immerman – Szelepcsényi Theorem)

$$\mathit{NL} = \mathit{coNL}$$

Proof Sketch Very briefly, one can prove this theorem by first showing that STCON is NL-complete, and then proving that coSTCON, the problem of deciding if s and t are not connected, on the configuration graphs of languages in NL is computable in NL as well. This gives $\mathsf{coNL} \subseteq \mathsf{NL}$. For equality, we note that if $\mathcal{L} \in \mathsf{NL}$, then $\overline{\mathcal{L}} \in \mathsf{NL}$ as well. So there is some logspace NTM M which accepts input x iff $x \notin \mathcal{L}$. But then solving coSTCON on the configuration graph of M gives a coNL algorithm for deciding $\mathcal{L}$, and so $\mathsf{NL} \subseteq \mathsf{coNL}$.

The next result, this one due to Reingold, shows that STCON in undirected graphs, or USTCON, is in L. This is a very non-trivial result, combining techniques from graph theory and computational complexity.

Theorem 4

$$\text{USTCON} \in \mathsf{L}$$

4 Randomized Complexity

We next study the use of randomness in computation. Two reasons motivate this: first, we do not know how to efficiently derandomize algorithms, in general. Second, it is very hard to actually generate random bits; current methods involve taking a small number of high-entropy bits, and using them to deterministicly generate a sequence of bits "sufficiently" indistinguishable from a truly random sequence. So algorithms which require less randomness are, in some sense, more efficient than those which require more.

The definition of a probabilistic language is similar to the certificate definition of a nondeterministic language.

Definition A probabilistic Turing Machine (PTM) is a deterministic Turing Machine M with two inputs, x and $\mathbf{r}$. M is a polytime PTM if it runs in time polynomial in $|x|$.

We can define three different probabilistic polytime classes, corresponding to the two-sided, one-sided, and zero-sided error of a PTM M deciding a language $\mathcal{L}$.

Definition A language $\mathcal{L}$ is in BPTIME, also called BPP, if there is a polytime PTM M such that

$$\forall x \in \mathcal{L} \left[\Pr_{\mathbf{r},|\mathbf{r}| \in p(|x|)}[M(x,\mathbf{r}) = 1] \geq \frac{2}{3} \right],$$

where $\Pr_{\mathbf{r},|\mathbf{r}| \in p(x)}[M(x,\mathbf{r}) = 1]$ is the probability over the uniform distribution of strings $\mathbf{r}$ of length polynomial in $|x|$ that M accepts $(x,\mathbf{r})$, and

$$\forall x \notin \mathcal{L} \left[\Pr_{\mathbf{r},|\mathbf{r}| \in p(|x|)}[M(x,\mathbf{r}) = 1] \leq \frac{1}{3} \right].$$

The particular values $\frac{2}{3}$ and $\frac{1}{3}$ are actually arbitrary; they can be as close to a half as $\frac{1}{2} \pm \frac{1}{m}$, where $m \in p(|x|)$, just so long as the probability of M accepting when x is in $\mathcal{L}$ and the probability of it rejecting when x is not in $\mathcal{L}$ are both greater than a half.

Definition A language $\mathcal{L}$ is in RPTIME, also called RP, if there is a polytime PTM M such that

$$\forall x \in \mathcal{L} \left[\Pr_{\mathbf{r},|\mathbf{r}| \in p(|x|)}[M(x,\mathbf{r}) = 1] \geq \frac{2}{3} \right],$$

and

$$\forall x \notin \mathcal{L} \left[\Pr_{\mathbf{r},|\mathbf{r}| \in p(|x|)}[M(x,\mathbf{r}) = 1] = 0 \right].$$

Note that $\mathsf{RP} \subseteq \mathsf{BPP}$.

Definition A language $\mathcal{L}$ is in ZPTIME, also called ZPP, if there is a PTM M such that M accepts $(x, \mathbf{r})$ iff $x \in \mathcal{L}$, and for every x runs in expected polytime, with the expectation being over the uniform distribution of strings $\mathbf{r}$ with length polynomial in $|x|$.

It is widely conjectured that $\mathsf{BPP} = \mathsf{P}$.

4.1 Two Famous Problems in BPP

Two interesting problems in computer science happen to lie in BPP, with one of the two having been proved to lie in P. They are

- the PRIMES problem: this was recently shown to be a deterministic polytime problem, but the randomized algorithm still runs much faster (and is simpler). It essentially guesses at factors of the candidate, and outputs whether or not none of the guesses actually factored the candidate. We do not know how to derandomise this simple algorithm.
- the polynomial identity testing (PIT) problem: this problem asks, given either an algebraic formula, or an algebraic circuit, with coefficients drawn from either $\mathbb{Z}$ or some finite field, test if the forumla is identically 1 over $\mathbb{Z}$ or the field. A similar algorithm to the PRIMES algorithm is used. Namely, evaluate the formula/circuit at random points, and output whether or not it was 1 at all sampled points.

5 Next class

- Where do randomized classes fit in, with respect to time and space bounded classes
- Why we can shrink the error thresholds for BPP and RP