Notes on Complexity Theory Last updated: October, 2011

Lecture 13

Jonathan Katz

1 Randomized Time Complexity

1.1 How Large is $\mathcal{BPP}$?

We know that

$$\mathcal{P} \subseteq \mathcal{ZPP} = \mathcal{RP} \cap \mathsf{co}\mathcal{RP} \subseteq \mathcal{RP} \cup \mathsf{co}\mathcal{RP} \subseteq \mathcal{BPP} \subseteq \mathsf{PSPACE}.$$

We currently do not have a very good unconditional bound on the power of $\mathcal{BPP}$ — in particular, it could be that $\mathcal{BPP} = \mathsf{NEXP}$. Perhaps surprisingly, especially in light of the many randomized algorithms known, the current conjecture is that $\mathcal{BPP}$ is *not* more powerful than $\mathcal{P}$. We will return to this point later in the semester when we talk about derandomization.

What (unconditional) upper bounds *can* we place on $\mathcal{BPP}$? Interestingly, we know that it is not more powerful than polynomial-size circuits; actually, the following theorem is also a good illustration of the power of non-uniformity.

Theorem 1 $\mathcal{BPP} \subset \mathcal{P}/\mathsf{poly}$.

Proof Let $L \in \mathcal{BPP}$. Using amplification, we know that there exists a polynomial-time Turing machine M such that $\Pr[M(x) \neq \chi_L(x)] < 2^{-|x|^2}$. Say M uses (at most) $p(|x|)$ random coins for some polynomial p. (Note that p is upper-bounded by the running time of M.) An equivalent way of stating this is that for each n, and each $x \in \{0,1\}^n$, the set of "bad" coins for x (i.e., coins for which $M(x)$ outputs the wrong answer) has size at most $2^{p(n)} \cdot 2^{-n^2}$. Taking the union of these "bad" sets over all $x \in \{0,1\}^n$, we find that the total number of random coins which are "bad" for *some* x is at most $2^{p(n)} \cdot 2^{-n} < 2^{p(n)}$. In particular, there exists at least one set of random coins $r_n^* \in \{0,1\}^{p(n)}$ that is "good" for *every* $x \in \{0,1\}^n$ (in fact, there are many such random coins). If we let the sequence of "advice strings" be exactly $\{r_n^*\}$ (using the alternate definition of $\mathcal{P}_{/\mathsf{poly}}$), we obtain the result of the theorem. ■

We can also place $\mathcal{BPP}$ in the polynomial hierarchy:

Theorem 2 $\mathcal{BPP} \subseteq \Sigma_2 \cap \Pi_2$.

Proof We show that $\mathcal{BPP} \subseteq \Sigma_2$; since $\mathcal{BPP}$ is closed under complement, this proves the theorem.

We begin by proving some probabilistic lemmas. Say $S \subseteq \{0,1\}^m$ is *large* if $|S| \geq (1 - \frac{1}{m})2^m$, and is *small* if $|S| < \frac{2^m}{m}$. For a string $z \in \{0,1\}^m$ define $S \oplus z \stackrel{\text{def}}{=} \{s \oplus z \mid s \in S\}$.

Claim 3 *If S is small, then for all $z_1, \ldots, z_m \in \{0,1\}^m$ we have $\bigcup_i (S \oplus z_i) \neq \{0,1\}^m$.*

This follows easily since

$$\left|\bigcup_i (S \oplus z_i)\right| \leq \sum_i |S \oplus z_i| = m \cdot |S| < 2^m.$$

Claim 4 *If S is large, then there exist $z_1, \ldots, z_m \in \{0,1\}^m$ such that $\bigcup_i (S \oplus z_i) = \{0,1\}^m$.*

In fact, we show that choosing at random works with high probability; i.e.,

$$\Pr_{z_1,\ldots,z_m \in \{0,1\}^m} \left[\bigcup_i (S \oplus z_i) = \{0,1\}^m \right] \geq 1 - \left(\frac{2}{m}\right)^m.$$

To see this, consider the probability that some fixed y is not in $\bigcup_i (S \oplus z_i)$. This is given by:

$$\begin{aligned} \Pr_{z_1,\ldots,z_m \in \{0,1\}^m} [y \notin \textstyle\bigcup_i (S \oplus z_i)] &= \textstyle\prod_i \Pr_{z \in \{0,1\}^m} [y \notin (S \oplus z)] \\ &\leq \left(\frac{1}{m}\right)^m. \end{aligned}$$

Applying a union bound by summing over all $y \in \{0,1\}^m$, we see that the probability that there *exists* a $y \in \{0,1\}^m$ which is not in $\bigcup_i (S \oplus z_i)$ is at most $\frac{2^m}{m^m}$.

We now prove the theorem. Given $L \in \mathcal{BPP}$, there exist a polynomial m and an algorithm M such that M uses $m(|x|)$ random coins and errs with probability less than $1/m$. For any input x, let $S_x \subseteq \{0,1\}^{m(|x|)}$ denote the set of random coins for which $M(x;r)$ outputs 1. Thus, if $x \in L$ (letting $m = m(|x|)$) we have $|S_x| > (1 - \frac{1}{m}) \cdot 2^m$ while if $x \notin L$ then $|S_x| < \frac{2^m}{m}$. This leads to the following Σ_2 characterization of L:

$$x \in L \Leftrightarrow \exists z_1, \ldots, z_m \in \{0,1\}^m \; \forall y \in \{0,1\}^m : y \in \textstyle\bigcup_i (S_x \oplus z_i).$$

(Note the desired condition can be efficiently verified by checking if $M(x; y \oplus z_i) \stackrel{?}{=} 1$ for some i.) ■

1.2 Complete Problems for $\mathcal{BPP}$?

As usual, we might like to study a class by focusing on the "hardest" problems in that class. With this in mind, we can ask whether $\mathcal{BPP}$ has any complete problems. The obvious thought is to consider the following language:

$$\left\{ (M, x, 1^p) \mid \begin{array}{c} M \text{ is a probabilistic machine that accepts } x \\ \text{with probability at least } 2/3 \text{ within } p \text{ steps} \end{array} \right\}.$$

While this language is $\mathcal{BPP}$-hard, it is not known to be in $\mathcal{BPP}$! (Consider the case when $\Pr[M(x) = 1] = 2/3 - 2^{-|x|}$.)

We can address this issue using the notion of *promise problems*, which gives an alternative to languages as a way to define complexity classes. A promise problem consists of two disjoint sets of strings $\Pi_Y, \Pi_N \subseteq \{0,1\}^*$ with $\Pi_Y \cap \Pi_N = \emptyset$. The "promise" is that all inputs will be from $\Pi_Y \cup \Pi_N$, and we only need to "solve" the problem on such inputs; in particular, we do not care what happens if we get an input that is not in $\Pi_Y \cup \Pi_N$. Thus, using this formulation, promise-$\mathcal{P}$ would be defined as the class of promise problems (Π_Y, Π_N) for which there exists a polynomial-time machine M such that

$$\begin{aligned} x \in \Pi_Y &\Rightarrow M(x) = 1] \\ x \in \Pi_N &\Rightarrow M(x) = 0. \end{aligned}$$

Promise problems generalize languages, since we may view a language L equivalently as the promise problem $(L,\ \{0,1\}^* \setminus L)$.

We can define the class promise-$\mathcal{BPP}$ as the class of promise problems (Π_Y, Π_N) for which there exists a probabilistic polynomial-time machine M such that

$$\begin{aligned} x \in \Pi_Y &\Rightarrow \Pr[M(x) = 1] \geq 2/3 \\ x \in \Pi_N &\Rightarrow \Pr[M(x) = 1] \leq 1/3. \end{aligned}$$

We don't care about the behavior of M on inputs not in (Π_Y, Π_N) — it might always accept, or accept some inputs but not others, or accept with arbitrary probability.

Arguably, promise problems are more natural than languages. (For example, we might speak of the input as representing an undirected graph and then need to ensure that every string encodes *some* undirected graph; it would be more natural to simply restrict our attention to strings that canonically represent undirected graphs.) And, indeed, promise-$\mathcal{BPP}$ does have a complete language:

$$\begin{aligned} \Pi_Y &= \left\{ (M, x, 1^p) \mid \begin{array}{c} M \text{ is a probabilistic machine that accepts } x \\ \text{with probability at least } 2/3 \text{ within } p \text{ steps} \end{array} \right\} \\ \Pi_N &= \left\{ (M, x, 1^p) \mid \begin{array}{c} M \text{ is a probabilistic machine that rejects } x \\ \text{with probability at least } 2/3 \text{ within } p \text{ steps} \end{array} \right\}. \end{aligned}$$

2 Randomized Space Complexity

When defining randomized space-complexity classes there are two subtleties to be aware of. The first subtlety arises if we model probabilistic computation by a Turing machine having a read-only random tape. (The issue does not come up if we instead view a probabilistic machine as a non-deterministic machine where the transition function is chosen at random.) Here, as in the case of the certificate-based definition of non-deterministic space complexity, we need to restrict the machine to having "read-once" access to its tape.

A second issue (regardless of which formulation of probabilistic computation we use) is that we must also impose a time bound on the machine.[1] E.g., for the case of one-sided error:

Definition 1 *A language L is in* $\text{RSPACE}(s(n))$ *if there exists a randomized Turing machine M using $s(n)$ space* **and** $2^{O(s(n))}$ **time** *such that*

$$x \in L \Rightarrow \Pr[M(x) = 1] \geq 1/2 \quad \textit{and} \quad x \notin L \Rightarrow \Pr[M(x) = 0] = 1.$$

If we do not impose this restriction, then probabilistic space classes become too powerful:

Proposition 5 *Define* RSPACE' *as above, but without the time restriction. Then for any space-constructible $s(n) \geq \log n$ we have* $\text{RSPACE}'(s(n)) = \text{NSPACE}(s(n))$.

[1] An equivalent condition is to require that the machine halts for every possible set of random choices. This is different from requiring the machine to halt with probability 1; see the proof of Proposition 5 for an illustration of this phenomenon.

Proof (Sketch) Showing that $\text{RSPACE}'(s(n)) \subseteq \text{NSPACE}(s(n))$ is easy. We turn to the other direction. The basic idea is that, given a language $L \in \text{NSPACE}(s(n))$, we construct a machine which on input x guesses valid witnesses for x (where a witness here is an accepting computation of the non-deterministic machine on input x). Since there may only be a single witness, we guess a doubly-exponential number of times. This is where the absence of a time bound makes a difference.

In more detail, given L as above we know that any $x \in L \cap \{0,1\}^n$ has a witness (i.e., an accepting computation) of length at most $\ell(n) = 2^{O(s(n))}$. If we happen to have such a witness written on the random tape, we can verify its correctness using space $O(s(n))$. So what we do is the following: alternately (1) read the next $\ell(n)$ bits of the random tape and check whether it encodes a witness, (2) read the next $\ell(n)$ bits of the random tape and halt if they are all 0. If $x \notin L$ this machine never accepts; if $x \in L$ then it accepts with probability $1/2$. Note that M may run for an unbounded amount of time, however it halts on all inputs with probability 1. ■

The most interesting probabilistic space classes are those where logarithmic space is used:

Definition 2 *$L \in \mathcal{RL}$ if there is a machine M using logarithmic space and running in polynomial time such that:*

$$x \in L \Rightarrow \Pr[M(x) = 1] \geq 1/2 \quad \textit{and} \quad x \notin L \Rightarrow \Pr[M(x) = 0] = 1.$$

$L \in \mathcal{BPL}$ if there is a machine M as above such that:

$$x \in L \Rightarrow \Pr[M(x) = 1] \geq 2/3 \quad \textit{and} \quad x \notin L \Rightarrow \Pr[M(x) = 1] \leq 1/3.$$

Here, too, the exact constants are arbitrary as error reduction still works. (We need only to maintain a counter of the fraction of accepting executions.) It is immediate that $\mathcal{RL} \subseteq \mathsf{NL}$. One can also show that $\mathcal{BPL} \subseteq \text{SPACE}(\log^2(n))$ and $\mathcal{BPL} \subseteq \mathcal{P}$.

Bibliographic Notes

For an in-depth discussion of promise problems, and arguments in favor of taking that approach, see the survey by Goldreich [1].

References

[1] O. Goldreich. On promise problems. Manuscript available on-line at `http://www.wisdom.weizmann.ac.il/~oded/prpr.html`