CS 221: Computational Complexity **Prof. Salil Vadhan**

Lecture Notes 12

March 8, 2010 **Scribe: David Wu**

1 Recap

Recall our definition of the basic complexity classes involving randomized computation:

Definition 1
$L \in \mathbf{BPP}, \mathbf{RP}, \mathbf{co\text{-}RP}$ iff there exists a PPT ("probabilistic polynomial time m algorithm") M such that

	BPP	**RP**	**co-RP**
$x \in L \rightarrow \Pr[M(x) = 1]$	$\geq 2/3$	$\geq 1/2$	$= 1$
$x \notin L \rightarrow \Pr[M(x) = 1]$	$\leq 1/3$	$= 0$	$\leq 1/2$

Recall that the choice of constants above is arbitrary - any constants suffice (so long as there is an appropriate gap), and in fact, we can obtain any desired $2^{-\text{poly}(n)}$ by iteration.

2 Randomized Logspace

Analogous to the time-bounded classes $\mathbf{BPP}, \mathbf{RP}, \mathbf{co\text{-}RP}, \mathbf{ZPP}$, we can define space-bounded classes $\mathbf{BPL}, \mathbf{RL}, \mathbf{co\text{-}RL}, \mathbf{ZPL}$. We will just give a brief survey of what's known about these classes.

Similarly to the time-bounded case, there is strong evidence that of these classes equal $\mathbf{L}$; indeed, it is known that if SAT requires circuits of size $2^{\Omega(n)}$ then $\mathbf{BPL} = \mathbf{L}$ (so randomness saves at most a constant factor in space).[1]

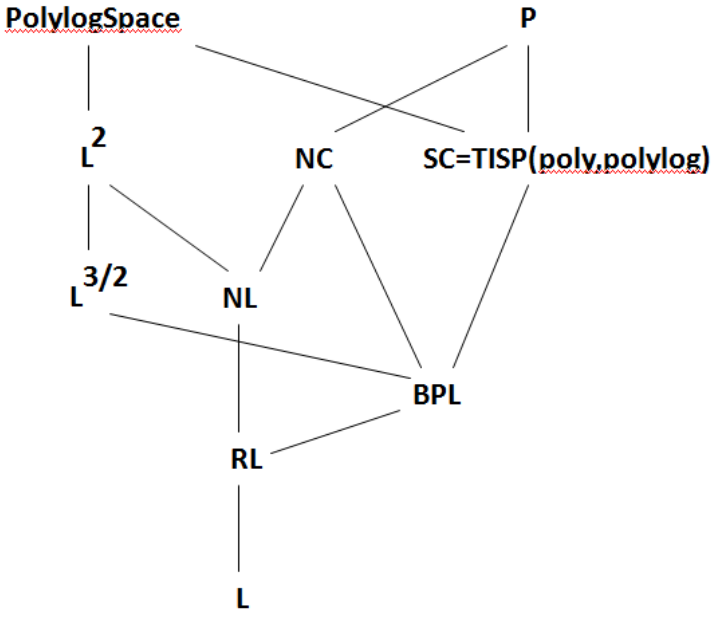
Unlike the time-bounded case, however, there is real hope for proving $\mathbf{BPL} = \mathbf{L}$ unconditionally. In contrast, it is known that derandomization of time-bounded classes implies superpolynomial circuit lower bounds for $\mathbf{NEXP}$ (Impagliazzo–Kabanets–Wigderson, Kabanets–Impagliazzo; covered in Arora–Barak Ch.20). No such results are known for $\mathbf{BPL}$. Indeed, there has been substantial progress in giving unconditional derandomizations of $\mathbf{BPL}$:

- For a long time, the best example of problem where randomization seemed to save on space was the UPATH path problem — deciding connectivity of two vertices s and t in an undirected graph. The randomized logspace algorithm for this problem (from 1979) is simply to do a random walk of $O(n^3)$ steps from s and accept if t is ever visited. In 2004, Reingold derandomized this algorithm, showing that UPATH $\in \mathbf{L}$.

[1] More generally, $\mathbf{BPL} = \mathbf{L}$ if $\mathbf{SPACE}(O(n))$ requires branching programs of size $2^{\Omega(n)}$. This variant of the Impagliazzo–Wigderson Theorem is due to Klivans and van Melkebeek.

- By Savitch's Theorem, it follows that $\mathbf{RL} \subseteq \mathbf{L}^2$, and this can be extended to show $\mathbf{BPL} \subseteq \mathbf{L}^2$. However, using derandomization techniques, better bounds on the deterministic space complexity of **BPL** are known. Indeed, it is known that $\mathbf{BPL} \subseteq \mathbf{L}^{3/2}$. In addition, it is known that $\mathbf{BPL} \subseteq \mathbf{TISP}(\text{poly}(n), \log^2 n) \subseteq \mathbf{SC}$, something that is not known for **NL**.

A summary of the known relations for randomized logspace is in the figure below:

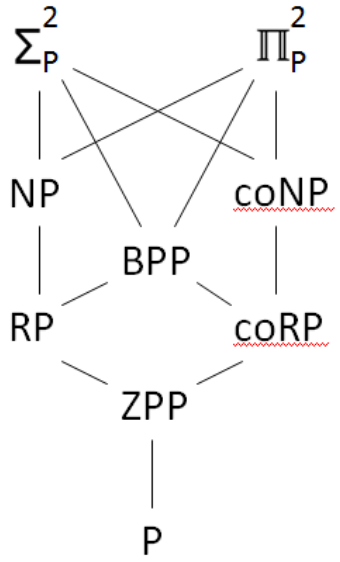


3 Relations of Randomized Poly-Time to Other Classes

Now we prove two results relating randomized polynomial time to other complexity classes. First we show that **BPP** is in the polynomial hierarchy:

Theorem 2 $\mathbf{BPP} \subseteq \mathbf{\Sigma_2^p} \cap \mathbf{\Pi_2^p}$

Thus, we have the following diagram:



Proof: It suffices to show that $\mathbf{BPP} \in \mathbf{\Sigma_2^p}$ since **BPP** is closed under complement.

Let $L \in BPP$.

Then, let M be a PPT algorithm for L with error probability $\leq 2^{-n}$ on inputs of length n. We view M as a deterministic polynomial-time algorithm of the input x and the coin tosses $r \in \{0,1\}^m$, where $m = poly(n)$ is the bound on runtime:

$$x \in L \rightarrow Pr[M(x) = 1] \geq 1 - 2^{-n}$$

$$x \notin L \rightarrow Pr[M(x) = 1] \leq 2^{-n}$$

We let S_x be the set of all sequences of coin tosses that cause M to accept:

$$S_x = \{r \in \{0,1\}^m : M(x,r) = 1\}$$

Intuitively, if $x \in L$, then S_x will cover most of the space $\{0,1\}^m$, and if $x \notin L$ then S_x will be at most a tiny portion of the space. The idea is that by taking m "shifts" of S_x, we can cover the entire space in the former case, and will not be able to in the latter.

We define a "shift" of S_x by $u \in \{0,1\}^m$ by:

$$S_x \oplus u = \{r \oplus u : r \in S_x\},$$

where $\oplus$ denotes bitwise xor. (Other operations, such as sum modulo 2^m could be used instead.

Claim 3

1. *If $x \in L$ then $\exists u_1, ..., u_m \in \{0,1\}^m$ such that $\forall r \in \{0,1\}^m$: $r \in \bigcup_i (S_x \oplus u_i)$.*
2. *If $x \notin L$ then $\forall u_1, ..., u_m \in \{0,1\}^m$ there $\exists r \in \{0,1\}^m$: $r \notin \bigcup_i (S_x \oplus u_i)$.*

Proof of claim: Part 2: For any $u_1, ..., u_m$ we consider the size of the set:

$$\left|\bigcup_i (S_x \oplus u_i)\right| \leq \sum_i |S_x \oplus u_i| = \sum_i |S_x| \leq m 2^{-n} 2^m << 2^m,$$

for sufficiently large n. Therefore, there exists some r not in $\bigcup_i (S_x \oplus u_i)$.

Part 1: Proof by the Probabilistic Method. Choose $u_1, ..., u_m$ uniformly at random. Consider any fixed r. Then, by independence of the u_i's,

$$\begin{aligned} \Pr_{u_1,...,u_m}[r \notin \bigcup_i S_x \oplus u_i] &= \Pr_u[r \notin S_x \oplus u]^m \\ &= \Pr_u[r \oplus u \notin S_x]^m \\ &= 2^{-nm}, \end{aligned}$$

where the last equality follows from the fact that if u is uniformly random, then so is $r \oplus u$. This is the probability that any single r is not covered by the m shifts of S_x. Then, the probability that any of the r's fails to be covered by the m shifts of S_x is at most 2^m times that:

$$\Pr_{u_1,...,u_m}[\exists r \quad r \notin \bigcup_i S_x \oplus u_i] \leq 2^m \cdot 2^{-nm} \ll 1$$

for sufficiently large n. Therefore, there must exist some $u_1, ..., u_n$ such that every r is covered. That is,

$$\exists u_1, ..., u_m \in \{0,1\}^m \; \forall r \in \{0,1\}^m \; r \in \bigcup_i (S_x \oplus u_i)$$

□

Now we return to the proof of Theorem 2. By Claim 3, we an easily give a Σ_2 algorithm. We existentally choose $u_1, ..., u_m$, and then universally choose r, and check if r is contained in any of these shifts of S_x, which we do by running $M(x; r \oplus u_i)$ for each i. This gives a $\mathbf{\Sigma_2^p}$ decider for L.

Therefore, $\mathbf{BPP} \in \mathbf{\Sigma_2^p} \cap \mathbf{\Pi_2^p}$

■

4 Randomness vs. Nonuniformity

The following theorem shows that "nonuniformity is more powerful than randomness".

Theorem 4 $\mathbf{BPP} \subseteq \mathbf{P/poly}$.

Proof:

Let $L \in \mathbf{BPP}$, let M be a poly-time PPT for L with error prob at most 2^{-2n}. We show that for each input length n, there exists a sequence of coin tosses that work on all inputs of that length.

There are 2^n inputs of that length, and the probability that each one might fail is at most 2^{-2n}, so the probability that some input might fail is:

$$\Pr_{r \in \{0,1\}^m}[\exists x \in \{0,1\}^n \; M(x,r) \text{ is incorrect}] \leq 2^n 2^{-2n} \leq 2^{-n}$$

Since this probability is less than 1, there exists a sequence of coin tosses for each input length n that work on all inputs of that length. Then, we simply provide this sequence as our polynomial-sized advice. Therefore, $L \in \mathbf{P/poly}$.

Therefore, $\mathbf{BPP} \subseteq \mathbf{P/poly}$

■

5 Promise Problems

In this section, we introduce the notion of a *promise problem*, which are decision problems where the input is "promised" to be in some set. Many computational problems of interest are most naturally formulated as promise problems. For example:

Example 5 Given a planar graph, decide if it is Hamiltonian.

The most natural interpretation of the above is that the input is "promised" to be planar; i.e. we don't care how the algorithm behaves if the input is not planar. In this case (and some others), the promise can be checked efficiently, so we tend to brush it under the rug, e.g. we might consider the language $\{G : G \text{ is a planar, Hamiltonian graph}\}$, which requires testing the conjunction of Hamiltonicity and planarity. Since planarity is in $\mathbf{P}$, this has the same complexity as testing Hamiltonicity of graphs that are promised to be planar. But in some cases, the complexity of the promise is not clear (e.g. testing planarity of graphs that are promised to be Hamiltonian), and it is thus useful to be able to deal with it explicitly.

A more complexity-theoretic motivation for promise problems, related to our current topic, is that no hierarchy theorems or complete problems are known for **BPTIME** classes or **BPP**, because we don't know how to enumerate $\mathbf{BPTIME}(n^k)$ algorithms (not clear how to discard algorithms that accept with probabilities in the gap between 1/3 and 2/3). But using promise problems, we can effectively eliminate these difficulties by "promising" that the input satisfies the gap.

Formally, we have:

Definition 6 *A* promise problem $\Pi = (\Pi_Y, \Pi_N)$ *consists of two disjoint sets of strings* Π_Y, Π_N, *expressing the computational problem "Given x promised to be in one set or the other, decide which is the case."*

Note that languages are a special case of promise problems, namely where $\Pi_Y \cup \Pi_N = \{0,1\}^*$.

All usual complexity classes generalize to promise problems in the obvious way, e.g.

Definition 7 $\Pi \in \mathbf{prBPP}$ *iff there exists a PPT M such that*

$$x \in \Pi_Y \rightarrow \Pr[M(x) = 1] \geq 2/3$$

$$x \in \Pi_N \rightarrow \Pr[M(x) = 1] \leq 1/3$$

Most results we prove about language classes also extend to promise classes via the same proof (e.g. **prBPP** $\subseteq$ **prP/poly**). However, as you'll see on Problem Set 4, there are cases where promise classes behave differently.

Theorem 8 *For any constant $\varepsilon \leq 1/6$, the problem $\pm\varepsilon$-*Approx Circuit Acceptance Probability*, abbreviated* $\mathrm{CAP}^{\varepsilon}$*, is* **prBPP***-complete.*

$$\mathrm{CAP}^{\varepsilon}_Y = \{(C,p) : \Pr_r[C(r)=1] \geq p+\varepsilon\}$$

$$\mathrm{CAP}^{\varepsilon}_N = \{(C,p) : \Pr_r[C(r)=1] \leq p\}$$

Proof: $\underline{\mathrm{CAP}^{\varepsilon} \in \mathbf{prBPP}}$: On input (C,p), choose $r_1,...,r_t$ uniformly at random. If $C(r_i)=1$ for at least $(p+\varepsilon/2)t$ values of i, accept, else reject. By the Chernoff bound, if $t = c/\varepsilon^2$, the error probability on $\mathrm{CAP}^{\varepsilon}_Y \cup \mathrm{CAP}^{\varepsilon}_N$ is at most $1/3$.

$\underline{\mathrm{CAP}^{\varepsilon} \text{ is } \mathbf{prBPP}\text{-hard:}}$ We take any $\Pi \in \mathbf{prBPP}$ and construct a poly-time reduction f:

$$x \in \Pi_Y \rightarrow f(x) \in \mathrm{CAP}^{\varepsilon}_Y$$

$$x \in \Pi_N \rightarrow f(x) \in \mathrm{CAP}^{\varepsilon}_N$$

Let M be a PPT for Π, considered as a deterministic machine of the input and its coin tosses, $M(x,r)$. Define f by:

$$f(x) = (C_x(\cdot) = M(x,\cdot), 1/2)$$

That is, we hardcode x into M and the take the random coins the as input, and then transform M into a circuit C_x.

If $x \in \Pi_Y$, then the probability that C_x accepts on a random input is at least $2/3$, which is more than ε from $1/2$. If $x \in \Pi_N$ then the then the probability that C_x accepts on a random input is at most $1/3$, which is less than $1/2$. Therefore, this is a reduction, as desired.

So we conclude that $\mathrm{CAP}^{\varepsilon}$ is **prBPP**-hard. ■

As we see from the example of CAP, promise problems are good for capturing decisional analogues of approximation problems. Other examples of approximation problems that are known to be in **prBPP** but are not known to be in **prP** are:

Example 9 $\times(1+\varepsilon)$-Approx #DNF: approximating the number of satisfying assignments of a DNF formula to within a *multiplicative factor* of $(1+\varepsilon)$. As a promise problem, we are given a DNF formula φ and a number N, and should say Yes if $|\varphi^{-1}(1)| \geq (1+\varepsilon)N$ and No if $|\varphi^{-1}(1)| \leq N$.

Example 10 $\times(1+\varepsilon)$-Approx Permanent: given an $n \times n$ nonnegative matrix M, approximate its permanent $\mathrm{perm}(M) = \sum_{\sigma \in S_n} \prod_i M_{i,\sigma(i)}$ to within a multiplicative factor of $(1+\varepsilon)$. As a promise problem, we are given M and a threshold t and should say Yes if $\mathrm{perm}(M) \geq (1+\varepsilon)t$ and No if $\mathrm{perm}(M) \leq t$.

There are many other approximation problems known to be in **prBPP** but not known to be in **prP**; thus **prBPP** is perhaps a better example of the power of randomization than the language class **BPP**. On the other hand, there is the same evidence that **prBPP** = **prP** that there is for **BPP** = **P**; in particular both hold under the assumption that SAT requires exponential-sized circuits.