Computer Science Track Core Course, IST Austria **Spring 2016**

Instructors: Krishnendu Chatterjee, Vladimir Kolmogorov, Krzysztof Pietrzak

Teaching Assistants: Bernhard Kragl, Josef Tkadlec, Michal Rolinek

Lecture 1: February 29th 2016

Lecturer: Krishnendu Chatterjee *Scribe notes by: Amir Goharshady*

Notes:

- *Course webpage is located at https://courses.app.ist.ac.at/index.php?id=133,*
- *Lecture notes are hosted on GitHub at https://github.com/bkragl/CS-601_S16,*
- *LaTeX template courtesy of UC Berkeley EECS department and my laziness in creating something from scratch.*

1.1 Review of Complexity Classes

Definition 1.1 (Alphabet, Strings, Languages, Complexity Class) *A finite alphabet is a finite non-empty set Σ. A finite string over Σ is a finite sequence of elements in Σ and the set of all finite strings over Σ is denoted Σ^*. Any set of finite strings over a finite alphabet is called a language, i.e. each subset L of Σ^* is a language. Any set of languages is called a complexity class.*

An example of a language is the set $L = \{(G, s, t) | G \text{ is a graph and } s \text{ and } t \text{ are vertices of } G \text{ connected by a path}\}$. The algorithms we look into focus on decision problems. These are "yes, no" problems defined as follows:

Definition 1.2 (Decision Problem) *Given a language L, the algorithmic decision problem for L is to find an algorithm A that gets strings as its input and decides whether the given string is in L, more specifically we are looking for an algorithm A such that for each string x:*

$$A(x) = \begin{cases} 1 & x \in L \\ 0 & x \notin L \end{cases}.$$

When there is no fear of confusion, we do not distinguish between a language and its decision problem.

We are specifically interested in algorithms that run in polynomial time.

Definition 1.3 (Worst-Case Runtime of an Algorithm) *Given an algorithm A, its worst-case runtime, T_A, is a function of the input length, defined to be the maximum runtime of A over all strings of that length, more formally:*

$$T_A(n) = \max_{x \in \Sigma^n} T(A, x),$$

where $T(A, x)$ is the execution time of A when given x as its input.

Definition 1.4 (Polytime Algorithm) *An algorithm A is a polytime algorithm if its worst-case runtime T_A is bounded by a polynomial.*

We now review some important complexity classes.

Definition 1.5 (P) *A language L is in the complexity class P if there exists a polytime algorithm A that solves its decision problem.*

Definition 1.6 (NP and coNP) *A language L is in the complexity class NP if there exist a polynomial p and a polytime algorithm A such that:*

$$\forall x \in L \ \ \exists y \in \{0,1\}^{p(|x|)} \ \ A(x,y) = 1,$$

and

$$\forall x \notin L \ \ \forall y \in \{0,1\}^{p(|x|)} \ \ A(x,y) = 0.$$

If $A(x,y) = 1$, then y is said to be a witness for x. A language L is in coNP if and only if its complement, $\Sigma^* \setminus L$, is in NP.

Proposition 1.7 *P is a subset of NP.*

Proof: If $L \in P$, then there exists a polytime algorithm A that solves its decision problem. The same algorithm can be used as in 1.6 with any arbitrary witness to show that L is in NP. ■

1.2 Probabilistic Complexity Classes

In the probabilistic computation model the algorithms get as their input a string x and a random input $r \in \{0,1\}^{p(|x|)}$ for some polynomial p, i.e. length of the random input is bounded by a polynomial in terms of x's length. Moreover r is assumed to be chosen uniformly. The algorithm then has to compute an output, or a decision, based on x and r.

Definition 1.8 (Probabilistic Polytime) *A is a probabilistic polytime algorithm if:*

- *Length of the random part, r, is bounded by a polynomial in string x's length, and*
- *Worst-case runtime of A is bounded by a polynomial.*

We now define several useful probabilistic complexity classes.

Definition 1.9 (RP – Randomized Polynomial) *A decision problem L is in RP if there exist a polynomial p and a probabilistic polytime algorithm A such that:*

$$\forall x \in L \quad Pr[A(x,r) = 1] > \frac{1}{2},$$

and

$$\forall x \notin L \quad Pr[A(x,r) = 1] = 0,$$

where the probabilities are calculated over all $r \in \{0,1\}^{p(|x|)}$ uniformly.

This intuitively means that the algorithm does rejection correctly, i.e. every $x \notin L$ is always rejected and accepts correctly with probability more than half.

Definition 1.10 (coRP) *A decision problem L is in coRP if there exist a polynomial p and a probabilistic polytime algorithm A such that:*

$$\forall x \in L \quad Pr[A(x,r)=1]=1,$$

and

$$\forall x \notin L \quad Pr[A(x,r)=1]<\frac{1}{2},$$

where the probabilities are calculated over all $r \in \{0,1\}^{p(|x|)}$ uniformly.

This intuitively means that the algorithm does acceptance correctly, i.e. every $x \in L$ is always accepted and rejects correctly with probability more than half.

Both RP and coRP account for one-sided error, now we define a complexity class that allows two-sided errors, i.e. in both acceptance and rejection.

Definition 1.11 (BPP – Bounded Probabilistic Polynomial) *A decision problem L is in BPP if there exist a polynomial p and a probabilistic polytime algorithm A such that:*

$$\forall x \in L \quad Pr[A(x,r)=1] \geq \frac{2}{3},$$

and

$$\forall x \notin L \quad Pr[A(x,r)=1] \leq \frac{1}{3},$$

where the probabilities are calculated over all $r \in \{0,1\}^{p(|x|)}$ uniformly.

Note: The $\frac{1}{2}$'s in definitions of RP and coRP are arbitrary numbers and one can get same complexity classes using other constant numbers or even constants raised to the power of a polynomial by simply repeating the algorithms. On the other hand in the definition of BPP, the first constant must be bigger than a half and the second one must be less than a half. To see why, consider a coin-tossing algorithm.

Definition 1.12 (Extended Decision Algorithms) *An extended decision algorithm A is an algorithm that can return one of the three values 0, 1 and ?, signifying rejection, acceptance and doubt or failure respectively.*

Definition 1.13 (ZPP – Zero-error Probabilistic Polynomial) *A decision problem L is in ZPP if there is a polynomial p and a polytime extended algorithm A such that:*

$$\forall x \quad Pr[A(x,r)=?] \leq \frac{1}{2},$$

and

$$\forall x \quad \forall r \in \{0,1\}^{p(|x|)} \quad A(x,r) \neq ? \Rightarrow A(x,r) = \begin{cases} 1 & x \in L \\ 0 & x \notin L \end{cases},$$

where the probabilities are calculated over all $r \in \{0,1\}^{p(|x|)}$ uniformly.

Intuitively, this means that the algorithm fails (is unsure) with probability less than half, and when it does not fail it will always produce the correct answer.

Proposition 1.14 *$P \subseteq RP$.*

Proof: Use the algorithm A from 1.5 in 1.9 and ignore r. ■

One can similarly and easily show that $P \subseteq coRP$ and $P \subseteq ZPP$.

Proposition 1.15 *$RP \subseteq NP$.*

Proof: If $L \in RP$, then for each $x \in L$, $Pr[A(x,r) = 1] > \frac{1}{2}$. Since this probability is positive, there exists an r such that $A(x,r) = 1$. This r can be used as a witness for x. Similarly, if $x \notin L$, no witness can be found since $Pr[A(x,r) = 1] = 0$. ■

Note: We do not know whether $P = RP$ or whether $RP = NP$.

Proposition 1.16 *$RP \subseteq BPP$.*

Proof: Let $L \in RP$. Take an algorithm A for it as in 1.9 and construct the algorithm $A^{(k)}$ which takes a string x as input and works as follows:

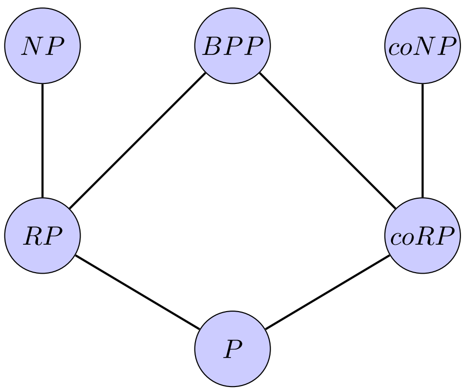
Choose random strings $r_1, r_2, \dots, r_k$
if $\exists i \;\; A(x, r_i) = 1$ **then**
  return 1
else
  return 0
endif

If $x \in L$, then $A(x,r)$ returns 1 with probability more than $\frac{1}{2}$, so $A^{(k)}$ returns 1 with probability more than $1 - \frac{1}{2^k}$. It is sufficient to choose k such that this value gets bigger than $\frac{2}{3}$. On the other hand, if $x \notin L$, then $A(x,r)$ always returns 0 and so does $A^{(k)}$.

■

Note: The relationship between BPP and NP is an open problem.

Homework: Prove that if $NP \subseteq BPP$, then $NP = RP$.



The figure above shows relations between various complexity classes. An edge between two classes means that the upper class is a superset of the lower one.

Definition 1.17 (Probabilistic Average Runtime) *Given a probabilistic algorithm A, its average runtime is a function of the length, n, of input x defined as:*

$$\max_{x \in \{0,1\}^n} E[T(A, x, r)],$$

where $T(A, x, r)$ is the runtime of A with inputs x and r and the expectation is defined uniformly over all possible r. Intuitively, for each input x, we take the average time the algorithm requires to terminate on x among all random r's and then we take the maximum over all strings x of the fixed length n.

Definition 1.18 (ACP – Average Case Polynomial) *A decision problem L is in ACP if there is an algorithm A with polynomial average runtime that always produces the right answer for L.*

Proposition 1.19 $ZPP = ACP$.

Proof: We first prove that $ZPP \subseteq ACP$. Let $L \in ZPP$ and A be an algorithm as in 1.13. We provide the following algorithm A':

```
1: Choose a random string r
if A(x, r) ≠? then
    return A(x, r)
else
    goto 1
endif
```

This algorithm, will always return the correct answer upon termination. Assuming that $A(x, r)$ terminates in time at most t, A', when run on x, has an average runtime of at most

$$t + \frac{1}{2}t + \frac{1}{4}t + \ldots = t\sum_{i=0}^{\infty} \frac{1}{2^i} = 2t.$$

Now we prove that $ACP \subseteq ZPP$. Let A be an algorithm as in 1.18. We create the following algorithm A':

```
Run the first 2t(x) steps of A(x), where t(x) is the average runtime of A(x).
if an answer was returned (a decision was made) then
    return the same answer (decision)
else
    return ?
endif
```

When the returned value is not ?, A' returns only correct answers because A has the same property. We should only show that given a string x, the probability that A' returns ? is at most $\frac{1}{2}$. Suppose otherwise, then $A(x)$ terminates in $2t(x)$ steps with probability less than $\frac{1}{2}$ which is a contradiction.

■