Chapter 5 : Randomized computation : Solutions of the exercises

---

1. Clearly, $\mathrm{PP} \subseteq \mathrm{PP}'$. For proving the converse inclusion take any $L \in \mathrm{PP}'$ and let N' be a PP' machine for L. Further let N' run in time $\leqslant f(n)$ for all n. I now design a PP machine N for L. N first makes $f(n)+1$ coin tosses. If all tosses give 'tail', N rejects and halts, else it simulates N' on the input α.

If $\alpha \in L$, then the probability that N accepts α is $\Pr[N' \text{ accepts } \alpha] \times \left(1 - 2^{-(f(n)+1)}\right)$. But N' works in $\leqslant f(n)$ steps, i.e., each branch of computation of N' has a probability of 2^{-k} for some $k \leqslant f(n)$. In particular, for $\alpha \in L$ we have $\Pr[N \text{ accepts } \alpha] \geqslant \left(\frac{1}{2} + 2^{-k}\right)\left(1 - 2^{-(f(n)+1)}\right) = \frac{1}{2} + 2^{-k} - 2^{-(f(n)+2)} - 2^{-(f(n)+k+1)} > \frac{1}{2}$, where the last inequality follows from that $k \leqslant f(n)$ and $k \geqslant 1$.

On the other hand, for $\alpha \notin L$ we have $\Pr[N \text{ rejects } \alpha] = \Pr[N' \text{ rejects } \alpha] \times \left(1 - 2^{-(f(n)+1)}\right) + 2^{-(f(n)+1)} \geqslant \frac{1}{2}\left(1 - 2^{-(f(n)+1)}\right) + 2^{-(f(n)+1)} = \frac{1}{2} - 2^{-(f(n)+2)} + 2^{-(f(n)+1)} > \frac{1}{2}$.

2. Let L be any language over Σ. Design a PPT N'' to accept L as follows. N'' first tosses a coin. If the outcome is 'head', it accepts. If the outcome is 'tail', it rejects. Clearly, for any α we have $\Pr[N'' \text{ accepts } \alpha] = \Pr[N'' \text{ rejects } \alpha] = \frac{1}{2}$, i.e., N'' is a PP'' machine for L. (It follows that PP' of the previous exercise is the weakest possible definition of a meaningful probabilistic complexity class.)

3. In view of Exercise 1 it suffices to show that $\mathrm{PP}_k = \mathrm{PP}'$ for $k > 1$.

$\left[\mathrm{PP}_k \subseteq \mathrm{PP}'\right]$ Let N_k be a PP_k machine for a language L. I now design a PP' machine N' for L. N' starts by making $k+1$ coin tosses and treats the outcomes of the tosses as an integer r in binary representation. If $0 \leqslant r \leqslant 2^k - 2$, N' accepts immediately. If $r = 2^k - 1$, N' rejects immediately. Finally, if $2^k \leqslant r \leqslant 2^{k+1} - 1$, N' simulates N_k on the input α.

If $\alpha \in L$, the probability that N' accepts α is $\left(\frac{1}{2} - \frac{1}{2^{k+1}}\right) + \frac{1}{2} \times \Pr[N_k \text{ accepts } \alpha] > \left(\frac{1}{2} - \frac{1}{2^{k+1}}\right) + \frac{1}{2} \times \frac{1}{2^k} = \frac{1}{2}$. On the other hand, if $\alpha \notin L$, N' rejects α with probability $\frac{1}{2^{k+1}} + \frac{1}{2} \times \Pr[N_k \text{ rejects } \alpha] \geqslant \frac{1}{2^{k+1}} + \frac{1}{2} \times \left(1 - \frac{1}{2^k}\right) = \frac{1}{2}$.

$\left[\mathrm{PP}' \subseteq \mathrm{PP}_k\right]$ Let N' be a PP' machine for a language L. I now design a PP_k machine N_k for L. N_k starts by making $k-1$ coin tosses. If the outcomes are *not* all 'heads', N_k rejects its input α, else it simulates N' on α.

N_k accepts $\alpha \in L$ with probability $\frac{1}{2^{k-1}} \times \Pr[N' \text{ accepts } \alpha] > \frac{1}{2^{k-1}} \times \frac{1}{2} = \frac{1}{2^k}$, whereas N_k rejects $\alpha \notin L$ with a probability $\left(1 - \frac{1}{2^{k-1}}\right) + \frac{1}{2^{k-1}} \times \Pr[N' \text{ rejects } \alpha] \geqslant \left(1 - \frac{1}{2^{k-1}}\right) + \frac{1}{2^{k-1}} \times \frac{1}{2} = 1 - \frac{1}{2^k}$.

4. Clearly, $\mathrm{BPP} \subseteq \mathrm{BPP}'$. For the converse let N' be a BPP' machine for L. I want to construct a BPP machine N for L. Take $t = k\,p^2(n)$ with k large enough so that $e^{-k/3} \leqslant 1/3$. N iterates N' t times and takes the decision by majority. By Chernoff's bounds the error probability for N is then $\leqslant 1/3$. Moreover, N simulates the poly-time algorithm N' only for a polynomial number (t) of times, i.e., N is also a poly-time algorithm.

5. Let $L \in \mathrm{PP}$ and N a PP (or PP') algorithm for L. We would like to produce a poly-space deterministic simulation of N. The idea is to run N for all possible toss outcomes. Let $p(n)$ be a (polynomial) bound on the running time of N. Thus N can make at most $p(n)$ coin tosses. The following simulation works:

1. Initialize counters $c_0 = c_1 = 0$.
2. for each outcome of $p(n)$ coin tosses do:
3.   Run N under the current sequence of coin tosses.
4.   If N accepts, then increment c_1, else increment c_0.
5. if $(c_1 > c_0)$, *accept*, else *reject*.

The simulation reuses space for different runs of N. Since N runs in $p(n)$ time and hence in $p(n)$ space too, all the simulations in Stage 3 requires only polynomial space. Moreover, the counters c_0 and c_1 store values at most as big as $2^{p(n)}$. Binary encoding of these values requires $\mathrm{O}(p(n))$ space. Thus the above simulation runs in polynomial space.

6. **(a)** We know that if $\mathrm{P} = \mathrm{NP}$, then the polynomial hierarchy collapses to its zeroth level, namely to P. In particular, $\mathrm{BPP} \subseteq \Sigma_2\mathrm{P} = \mathrm{P}$. The reverse inclusion ($\mathrm{P} \subseteq \mathrm{BPP}$) is obvious.

(b) From $\mathrm{RP} \subseteq \mathrm{NP} \subseteq \mathrm{coRP} \subseteq \mathrm{coNP}$, it follows that $\mathrm{NP} = \mathrm{coNP} = \mathrm{RP} = \mathrm{coRP} = \mathrm{RP} \cap \mathrm{coRP} = \mathrm{ZPP}$.

(c) Since SAT is NP-complete, it suffices to show that if SAT has a BPP algorithm N_B, it also has an RP algorithm N_R. In view of the Chernoff bound, we may assume that the error of N_B is $\leqslant \frac{3}{\pi^2(n+1)^2}$. N_R simulates N_B several times in order to obtain a probably satisfying assignment for the input formula ϕ, as explained below:

1. Let m be the number of variables in the input formula ϕ. Call the variables $x_1, \ldots, x_m$.
2. Call $\phi_0(x_1, \ldots, x_m) := \phi(x_1, \ldots, x_m)$.
3. for $i = 1, \ldots, m$ do:
4. Simulate N_B on the input ϕ_i with x_i set to 0.
5. If N_B accepts, set $a_i := 0$, else set $a_i := 1$.
6. Set $\phi_i(x_{i+1}, \ldots, x_m) := \phi(a_1, \ldots, a_i, x_{i+1}, \ldots, x_m)$.
7. If $\phi(a_1, \ldots, a_m) = 0$, *reject*, else *accept*.

Clearly, N_R runs in poly-time. Moreover, if $\langle\phi\rangle \notin \mathrm{SAT}$, Stage 7 rejects $\langle\phi\rangle$, thereby making the error one-sided. If $\langle\phi\rangle \in \mathrm{SAT}$, N_R may still reject, that is, it ends up in having a non-satisfying assignment of the variables. This happens, only if (at least) one of the calls of N_B gives erroneous result. Let n_i be the size of ϕ_i. We have $n_i \geqslant m - i$. Therefore, the probability of error is $\leqslant \frac{3}{\pi^2}\sum_{i=0}^{m}\frac{1}{(n_i+1)^2} \leqslant \frac{3}{\pi^2}\sum_{i=0}^{m}\frac{1}{(m-i+1)^2} < \frac{3}{\pi^2}\sum_{i=0}^{\infty}\frac{1}{(i+1)^2} = \frac{3}{\pi^2} \times \frac{\pi^2}{6} = \frac{1}{2}$.

7. Let $\mathcal{C}$ be one of the classes RP and BPP. Let N_1 and N_2 be two class $\mathcal{C}$ machines deciding the languages L_1 and L_2 respectively. Let ϵ_1 and ϵ_2 be the two errors, as discussed in the text. For RP, $\epsilon_1 = 0, \epsilon_2 \leqslant 1/2$, whereas for BPP we take $\epsilon_1, \epsilon_2 \leqslant 1/4$.

[$\mathcal{C}$ is closed under intersection]

Consider the following algorithm $N_\cap$, that accepts, if and only if both N_1 and N_2 accept:

1. Simulate N_1 on input α.
2. If N_1 rejects, *reject* (and halt).
3. Simulate N_2 on α.
4. If N_2 accepts, *accept*, else *reject*.

First consider $\mathcal{C} = \mathrm{RP}$. $N_\cap$ accepts $\alpha \in L_1 \cap L_2$ with probability $\geqslant 1/4$ and rejects $\alpha \notin L_1 \cap L_2$ with probability 1 (since in the second case either $\alpha \notin L_1$ or $\alpha \notin L_2$ and so $N_\cap$ cannot have any accepting branches).

Next consider $\mathcal{C} = \mathrm{BPP}$. $N_\cap$ accepts $\alpha \in L_1 \cap L_2$ with probability $\geqslant \frac{3}{4} \times \frac{3}{4} = \frac{9}{16}$. Finally for $\alpha \notin L_1 \cap L_2$ we consider two cases. If $\alpha \notin L_1$, then $\Pr[N_\cap \text{ rejects } \alpha] = \Pr[N_1 \text{ or } N_2 \text{ rejects } \alpha] = \Pr[N_1 \text{ rejects } \alpha] + \Pr[N_2 \text{ rejects } \alpha] - \Pr[N_1 \text{ and } N_2 \text{ rejects } \alpha] \geqslant \Pr[N_1 \text{ rejects } \alpha] \geqslant \frac{3}{4} > \frac{9}{16}$. If $\alpha \in L_1 \setminus L_2$, the probability that N_2 is simulated is at least $\frac{3}{4}$. But then N_2 rejects α with probability $\geqslant \frac{3}{4}$. Thus $\Pr[N_\cap \text{ rejects } \alpha] \geqslant \frac{3}{4} \times \frac{3}{4} = \frac{9}{16}$. Since $\frac{9}{16}$ is a constant bounded away from $\frac{1}{2}$, $N_\cap$ is a BPP algorithm.

[RP is closed under union]

Consider the following algorithm $N_\cup$:

1. Make a coin toss.
2. If the outcome is 'head', simulate N_1 on the input α, else simulate N_2 on α.
3. Echo the decision of the simulated machine.

First assume $\alpha \in L_1 \cup L_2$. If $\mathcal{C} = \mathrm{RP}$, $N_\cup$ accepts α with probability $\frac{1}{2} \times \Pr[N_1 \text{ accepts } \alpha] + \frac{1}{2} \times \Pr[N_2 \text{ accepts } \alpha] \geqslant \frac{1}{2} \times \frac{1}{2} + \frac{1}{2} \times \frac{1}{2} = \frac{1}{2}$. Now let $\alpha \notin L_1 \cup L_2$, i.e., α is in neither of L_1 or L_2. We have *no* accepting branches and $N_\cup$ rejects with certainty.

[BPP is closed under union]

BPP is closed under intersection and complement. Now use the fact that $L_1 \cup L_2 = \overline{\overline{L_1} \cap \overline{L_2}}$.

8. **(a)** Since every deterministic algorithm can be thought of as a probabilistic one, $\mathrm{SPACE}(f(n)) \subseteq \mathrm{RSPACE}(f(n))$. Also an $\mathrm{RSPACE}(f(n))$ algorithm can be run as a nondeterministic machine (with nondeterministic choices replacing the coin tosses), implying that $\mathrm{RSPACE}(f(n)) \subseteq \mathrm{NSPACE}(f(n))$.

(b) As in Part (a) $\mathrm{RSPACE}'(f(n)) \subseteq \mathrm{NSPACE}(f(n))$. For proving the converse let N decide L in nondeterministic space $f(n)$. Since each branch of computation of N terminates, no configuration can repeat in a branch. Moreover, each branch uses $\mathrm{O}(f(n))$ space, and so N runs in time $2^{\mathrm{O}(f(n))}$. We may assume that at every step N has at most two nondeterministic choices. The above time bound shows that N has at most $2^{2^{\mathrm{O}(f(n))}}$ branches of computation on an input α of size n. If $\alpha \notin L$, all these branches are rejecting, whereas if $\alpha \in L$, then there exists at least one accepting branch. Therefore, $\mathrm{O}(2^{2^{\mathrm{O}(f(n))}})$ random selections of branches reveal the accepting branch with probability $\geqslant 1/2$. This idea leads to the next algorithm. Unfortunately, however, the number of attempts is doubly exponential in $f(n)$ and keeping track of counters of that big size requires exponential space. We avoid this problem by using randomized counters.

1. Repeat the following stages:
2. Make $2^{\mathrm{O}(f(n))}$ coin tosses. (Don't remember outcomes.)
3. If all tosses give 'head', *reject*.
4. Simulate N on input α with nondeterministic choices dictated by outcomes of (new) coin tosses.
5. If N accepts, *accept*.

Stage 2 makes an exponential number of coin tosses. Storing the outcomes requires exponential space and is not needed (Just maintain an 'all heads' flag!). However, a counter that can store integers as big as $2^{\mathrm{O}(f(n))}$ is required to ensure that the correct number of times coins are tossed. Binary representation of integers $\leqslant 2^{\mathrm{O}(f(n))}$ requires only $\mathrm{O}(f(n))$ space. The condition in Stage 3 has probability $2^{-2^{\mathrm{O}(f(n))}}$, i.e., after $\mathrm{O}(2^{2^{\mathrm{O}(f(n))}})$ iterations the outermost loop is *expected* to terminate. This gives us good opportunity to expect with high probability that one accepting configuration shows up in Stage 5 (provided $\alpha \in L$). On the other hand, if α is not a member of L, no branches accept, so the algorithm terminates at Stage 3, when an 'all heads' outcome is encountered. Each simulation of N in Stage 4 requires $\mathrm{O}(f(n))$ space and this space can be reused during every simulation. Thus we have designed an $\mathrm{RSPACE}'(f(n))$ algorithm for L.

Note that this algorithm *need not terminate*, since the conditions in Stages 3 and 5 may never be met. (We can rewrite it in the form of a proper algorithm that always terminates. But it may then be difficult to manage with $\mathrm{O}(f(n))$ space!) Its expected running time is, however, $\mathrm{O}(2^{2^{\mathrm{O}(f(n))}})$ which is doubly exponential in $f(n)$. This motivates us to put a cap on the running time and be happier with RSPACE rather than with RSPACE'.

Wait! I never mentioned that there cannot exist better (more time-efficient) randomized simulations of NSPACE algorithms. The possibility that $\mathrm{RSPACE}(f(n)) = \mathrm{RSPACE}'(f(n))$, though unlikely, is not ruled out altogether.

9. The simple RL algorithm for UPATH is as follows:

1. Let $m := |V(G)|$ and $v_0 := s$.
2. for $i = 1, 2, \ldots, 8m^3$ do:
3. Choose a neighbor v_i uniformly from the set of neighbors of v_{i-1}.
4. If $v_i = t$, *accept*.
5. No s, t-walk has been discovered in $8m^3$ steps, so *reject*.

If G does not possess an s, t-path, no walk (random or otherwise) from s can reach t and the above algorithm rejects at Stage 5. On the other hand, if G contains an s, t-path, a random walk from s visits t in $\leqslant 8m^3$ steps with probability $\geqslant 1/2$ (by the given result). In that case, the algorithm accepts with probability $\geqslant 1/2$. Thus we have a randomized algorithm for UPATH.

One needs to store only the current vertex v_i in the walk and a counter (in binary) that can count up to $8m^3$, i.e., the above algorithm requires only logarithmic space. Moreover, its running time is polynomial in the input size ($8m^3$ choices of neighbors). So $\mathrm{UPATH} \in \mathrm{RL}$.