第十章

10.9** A ***Boolean formula is a Boolean circuit wherein every gate has only one output wire. The same input variable may appear in multiple places of a Boolean formula. Prove that a language has a polynomial size family of formulas iff it is in NC^1. Ignore uniformity considerations.

First we show that every language in $\mathbf{NC}^1$ can be computed by a polynomial size family of formulas. A simple induction on d shows that every Boolean circuit of depth d is equivalent to a formula of size at most $O(2^d)$. For the induction step, note that the output gate of the circuit has fan-in at most 2, and we can apply the induction hypothesis to each input gate.

The n-th circuit C_n in an $\mathbf{NC}^1$ circuit family has depth $O(\log n)$, so the corresponding formula has size $2^{O(\log n)} = n^{O(1)}$ (polynomial size).

For the converse, we need to prove the Hint. Let T be a binary rooted tree with $m \geq 2$ leaves. Starting at the root of T, and walk toward the leaves, always choosing the subtree with at least half of the number of leaves of the current subtree. Stop when reaching a subtree T' with at most $2m/3$ leaves. Then T' has at least $m/3$ leaves since the previous subtree has more than $2m/3$ leaves. Thus T' is the desired subtree.

Now we show that a polynomial size family of formulas can be converted to an equivalent circuit family of depth $O(\log n)$. It suffices to show how to convert a formula φ with s leaves to an equivalent circuit of depth $O(\log s)$. (We may assume that all nodes have fan-in 2, by pushing not gates to the leaves, using DeMorgan's laws.)

Now we prove by induction on $s \geq 4$, that a formula φ with s leaves is equivalent to a formula φ' with depth at most $C \log_2 s$, where the constant C will be determined. If $s \leq 4$, let $\varphi' = \varphi$. Otherwise apply the Hint to the tree structure of φ to obtain a subformula ψ with between $s/3$ and $2s/3$ leaves. Let $\hat{\varphi}(y)$ be φ with the subformula ψ replaced by a new variable y. Thus φ is the same as $\hat{\varphi}(\psi)$, and φ is equivalent to φ_1, which is given by

$$\varphi_1 = (\psi \wedge \hat{\varphi}(1)) \vee (\neg\psi \wedge \hat{\varphi}(0))$$

Note that $\hat{\varphi}(1)$ and $\hat{\varphi}(0)$ each have at most $2s/3$ leaves which are variables, and in general we may replace a formula involving the constants 0 and 1 and variables with an equivalent formula of no greater depth without the constants.

Finally let φ' be φ_1 with the equivalents of subformulas ψ, $\varphi\hat{(}1)$, and $\varphi\hat{(}0)$ replaced by equivalent small depth formulas given by the induction hypothesis. Thus the depth of φ' is at most $C \log_2((2/3)s) + 3$. This is at most $C \log_2 s$, provided $C \geq 3/\log_2(3/2)$.

10.12 Show that if P = NP, then P = PH.

10.12

- $\mathbf{P} = \mathbf{NP} \implies \mathbf{NP} = \mathbf{coNP} = \mathbf{P}$: invert answer of **NP**
- So true for $i = 1$. Prove by induction: $\sum_i^p, \prod_i^p \subseteq \mathbf{P}$
- Assume it is true for $i - 1$. Then assume $\mathbf{P} = \sum_{i-1}^p = \prod_{i-1}^p$
- Let $L \in \sum_i^p$. Then by definition,
  - $x \in L \iff \exists u_1 \; \forall u_2 \; \ldots \; Q_i u_i \;\; M(x, u_1, \ldots, u_i) = 1$
  - Define another L' by making u_1 part of the input : $x \circ u_1 \in L' \iff \forall u_2 \ldots Q_i u_i \;\; M(x, u_1, u_2, \ldots, u_i) = 1$
  - $L' \in \prod_{i-1}^p$, and by inductive hypothesis, $L' \in \mathbf{P}$ $\implies \exists$ **TM** M' such that $M'(x, u_1) = 1$ iff $x \circ u_1 \in L'$
  - But now, $x \in L \iff \exists u_1 \; M'(x, u_1) = 1$
  - But this is exactly the definition of **NP**
  - By assumption, $\mathbf{NP} = \mathbf{P}$, hence $L \in \mathbf{P}$

10.19

Question: Show that if NP $\subseteq$ BPP, then NP = RP. HINT: Use the downward self-reducibility of SAT to eliminate error on NO instances.

Solution: Assume NP $\subseteq$ BPP. Note that RP $\subseteq$ NP unconditionally, hence we just need to show that NP $\subseteq$ RP. Since SAT is NP-complete and RP is closed under polynomial-time m-reductions, it is enough to show that SAT is in RP.

By assumption SAT is in BPP. Let M be a probabilistic Turing machine running in polynomial time and accepting SAT with error at most $1/3$. Using error amplification, we can define a probabilistic Turing machine M' running in polynomial time and accepting SAT with error at most $2^{-\Omega(n)}$, where n is the input length. We will define a probabilistic Turing machine N running in polynomial time, which on input ϕ, accepts with probability at least $1/2$ if ϕ is satisfiable, and accepts with probability 0 if ϕ is unsatisfiable.

The basic idea is to use downward self-reducibility to construct a satisfying assignment with high probability for YES instances, and to accept only if a satisfying assignment has been constructed. More precisely, N does the following on an input ϕ. Assume wlog that the variables in ϕ are $x_1, x_2 \ldots x_m$. N first runs M on ϕ. If M rejects, N rejects. Otherwise N sets x_1 to "false" in ϕ and runs M on the corresponding formula ϕ_0. If M accepts, N continues to build a satisfying assignment by setting x_2 to "false" in ϕ_0 and running M on the corresponding formula ϕ_{00}. If M rejects, N runs M on formula ϕ_1 obtained by setting x_1 to "true" in ϕ, continuing to build an assignment if M accepts on ϕ_1 and rejecting otherwise. This process continues until either N rejects or all variables are set. If the latter, N checks whether the corresponding assignment satisfies ϕ. If yes, it accepts, otherwise it rejects.

N runs in polynomial time since it makes at most a linear number of calls to the polynomial-time probabilistic TM M, and each of these calls is on an input whose length is at most the length of ϕ (setting variables can only decrease the length of a formula). N never accepts on an unsatisfiable formula, hence all we need to show is that N accepts with probability at least $1/2$ on a satisfiable formula. Note that if M always gives correct answers on calls to M, then when ϕ is satisfiable, N constructs a satisfying assignment to ϕ and hence accepts. The probability that this happens is at least $1 - n2^{-\Omega(n)}$, which is at most $1/2$ for large enough n, since the probability that M gives at least one wrong answer is at most $n2^{-\Omega(n)}$ by the union bound.

10.21 Let $EQ_{\mathrm{BP}} = \{\langle B_1, B_2\rangle |\ B_1 \text{ and } B_2 \text{ are equivalent branching programs}\}$. Show that EQ_{BP} is coNP-complete.

This last question requires a somewhat different approach: prove that **BPL** $\subseteq$ **P**.

Solution Sketch. Let L be a **BPL** language and M the machine required by the definition of **BPL**. On input x of length n, let C be the number of configurations of $M(\cdot, x)$. Construct a $C \times C$ matrix P such that $P[c_1, c_2] = 1/2$ if c_2 is reachable from c_1 in one step, and $P[c_1, c_2] = 0$ otherwise. For every t, $P^t[c_1, c_2]$ is the probability of reaching configuration c_2 from configuration c_1 in t steps, where P^t is the matrix obtained by multiplying P with itself t times. By computing all powers of P up to the running time of $M(\cdot, x)$ we can compute the accepting probability of $M(r, x)$, and decide if $x \in L$. This time the calculations can be exact: each probability is an integer multiple of $1/2^{p(n)}$, and so it can be represented using a polynomial number of digits.

10.22

(a) Prove that $\mathbf{RL} \subseteq \mathbf{BPL}$.

Solution Sketch. We only need to reduce the error probability from $1/2$ to below $1/3$, for example by doing two independent repetitions of the algorithm and accepting if and only if both repetitions accept.

(b) Prove that $\mathbf{BPL} \subseteq \mathbf{SPACE}(O((\log n)^2)$.

Solution Sketch. This is similar to Savitch's algorithm. Let L be a **BPL** language and M the machine required by the definition of **BPL**. Let $p(n)$ be a polynomial upper bound on the running time of M on inputs of length n. We now describe a deterministic $O(\log^2 n)$-space algorithm for L. On input x of length n, we consider the *configuration graph* of $M(\cdot, x)$. A vertex in the graph is a description of a configuration of M on input x: state of the machine, content of the work tape, position of the head on the input tape, position of the head on the work tape. Each vertex has two outgoing edges, corresponding to the two possible next configurations of the machine depending on the bit read from the random tape. The accepting and rejecting configurations have no outgoing edges.

We define a recursive procedure $prob(c_1, c_2, k)$ that given two configurations c_1 and c_2 computes the probability that the machine reaches configuration c_2 starting from configuration c_1 in exactly k steps. The base case when $k = 1$ are treated separately. When $k \geq 2$, then we compute

$$\sum_c prob(c_1, c, \lceil k/2 \rceil) \cdot prob(c, c_2, \lfloor k/2 \rfloor)$$

where the summation is taken over all configurations c. It is enough to represent the probabilities as truncated decimals using $O(\log p(n))$ digits. The truncation introduces errors, but the error in $proc(\cdot, \cdot, p(n))$ is small provided that we use $O(\log p(n))$ digits. The analysis of the recursion is as in Savitch's algorithm.

(c) This last question requires a somewhat different approach: prove that $\mathbf{BPL} \subseteq \mathbf{P}$.

Solution Sketch. Let L be a **BPL** language and M the machine required by the definition of **BPL**. On input x of length n, let C be the number of configurations of $M(\cdot, x)$. Construct a $C \times C$ matrix P such that $P[c_1, c_2] = 1/2$ if c_2 is reachable from c_1 in one step, and $P[c_1, c_2] = 0$ otherwise. For every t, $P^t[c_1, c_2]$ is the probability of reaching configuration c_2 from configuration c_1 in t steps, where P^t is the matrix obtained by multiplying P with itself t times. By computing all powers of P up to the running time of $M(\cdot, x)$ we can compute the accepting probability of $M(r, x)$, and decide if $x \in L$. This time the calculations can be exact: each probability is an integer multiple of $1/2^{p(n)}$, and so it can be represented using a polynomial number of digits.

10.20

Answer:

RP $\cap$ *co-RP* $\subseteq$ *ZPP*. Let $L \in$ RP $\cap$ co-RP. Then let A be an RP algorithm for L and let B be a co-RP algorithm for L.

Let w be the input. One step of our ZPP algorithm will be the following: run A on w, and then run B on w. If A accepts, we accept, and if B rejects, we reject. Note that this step takes polynomial time since each A and B take polynomial time.

If A rejects and B accepts, we repeat.

First, we show correctness. Note that if $w \notin L$, then A always rejects, so if A accepts, $w \in L$. Similarly, if B rejects, $w \notin L$. Thus, when we output an answer, we are always correct (which is better than probability $\frac{2}{3}$). Note that our algorithm never outputs ? [1] .

Then, we prove that the running time is polynomial in expectation. Note that since each step has polynomial running time, it suffices to show the number of steps is polynomial in expectation (in fact, we will show it is constant in expectation). Note that if $w \in L$, then A will accept with probability at least $\frac{1}{2}$ (by definition of RP). Similarly for $w \notin L$ and B rejecting. Thus, each step succeeds with probability at least $\frac{1}{2}$.

Let X be a random variable denoting how many steps we take. Then let $p_k = \Pr[X = k]$ denote the probability we take exactly k steps ($k \geq 1$). Note that p_k is the probability that we fail on the first $k-1$ steps and then succeed, so $p_k \leq \frac{1}{2^{k-1}}$. Then

$$\mathbb{E}[X] = \sum^{\infty} \frac{k}{2^{k-1}} = 4$$

and we are done[2] .

ZPP $\subseteq$ *RP* (we can prove that ZPP $\subseteq$ co-RP analogously) Let L be in ZPP, and let A be a ZPP algorithm for L, and let $p(n)$ be the expected running time of A on inputs of length n. Then we build the following RP algorithm: on input w of length n, we run A on w for time $6p(n)$. If A accepts or rejects, we output the answer A gives us. Otherwise (if A does not terminate, or outputs ?), we reject.

Note that since A has expected polynomial running time, $p(n)$ is a polynomial, so $6p(n)$ is also polynomial. It suffices to prove correctness. First, if $w \notin L$, then A will either reject, print ?, or not terminate. In all of these cases, we reject, so we are correct.

On the other hand, let $w \in L$. Again, three things can happen: A accepts, A prints ?, or A does not terminate in time. We are correct in the first two cases, so it suffices to show the probability of the latter two are at most $\frac{1}{2}$. The probability that A outputs ? is at most $\frac{1}{3}$. The probability that A does not terminate in $6p(n)$ time is at most $\frac{1}{6}$ (by Markov's inequality). Thus, if $w \in L$, the probability that A is wrong is at most $\frac{1}{2}$, thus this algorithm is an RP algorithm for L.