第九章

9.1 证明 $TIME(2^n)=TIME(2^{n+1})$.

证明：$2^n=O(2^{n+1}) \Rightarrow TIME(2^n) \subseteq TIME(2^{n+1})$.

$2^{n+1}=O(2^n) \Rightarrow TIME(2^{n+1}) \subseteq TIME(2^n)$.

所以 $TIME(2^n)=TIME(2^{n+1})$.

9.2 证明 $TIME(2^n) \subset TIME(2^{2n})$。注：这里“$\subset$”是严格包含。

证明：令 $f(n)=2^{2n}$,则 $f(n)/\log f(n)=2^{2n}/2n$，由时间层次定理有

$$TIME(o(2^{2n}/2n)) \subset TIME(2^{2n}).$$

又由于 $2^n=o(2^{2n}/2n)$，$TIME(2^n) \subseteq TIME(o(2^{2n}/2n))$，所以

$$TIME(2^n) \subset TIME(2^{2n}).$$

9.3 证明 $NTIME(n) \subset PSPACE$.

证明：$NTIME(n) \subseteq NSPACE(n) \subseteq SPACE(n^2) \subset SPACE(n^3) \subset PSPACE$.

9.6 证明若 $A \in P$，则 $P^A = P$。

证明：首先 $P \subseteq P^A$。这是因为不带谕示即可。下面证明 $P^A \subseteq P$。

任取 $A \in P$，则存在多项式图灵机 T 判定 A。

设 $B \in P^A$，则存在带语言 A 的谕示的多项式时间图灵机 M^A 判定 B。

如下构造不带谕示的图灵机 D：

D=“对于输入串 w:

1) 在 w 上运行 M^A。
2) 每当 M^A 要在谕示带上写下某个字符串 x，则在 x 上运行 T，若 T 接受，则代替谕示回答 x 属于 A，否则代替谕示回答 x 不属于 A。

3) 若 M^A 接受，则接受；否则，拒绝。”

设 M^A 的运行时间是 n^a，T 的运行时间是 n^b。谕示带上写下的字符串的长度不会超过 n^a，询问谕示带的次数也不会超过 n^a。D 的运行时间是 $n^a (n^a)^b=n^{a+ab}$，所以 $A\in P$。

9.7 给出带指数的正则表达式，产生如下在字母表{0,1}上的语言：

a. 所有长为 500 的字符串. $(0\cup 1)^{500}$。

b. 所有长度不超过 500 的字符串.$(0\cup 1\cup \varepsilon)^{500}$.

c. 所有不少于 500 的字符串. $(0\cup 1)^{500}(0\cup 1)^*$.

d. 所有长度不等于 500 的字符串. $(0\cup 1\cup \varepsilon)^{499}\cup(0\cup 1)^{501}(0\cup 1)^*$.

e. 所有恰好包含 500 个 1 的字符串. $0^*(10^*)^{500}$.

f. 所有包含至少 500 个 1 的字符串. $(0\cup 1)^*(1(0\cup 1)^*)^{500}$.

g. 包含至多 500 个 1 的字符串. $0^*((\varepsilon\cup 1)0^*)^{500}$.

h. 所有长度不少于 500 并且在第 500 个位置上是 0 的字符串. $(0\cup 1)^{499}0(0\cup 1)^*$.

i. 所有包含两个 0 并且其间至少相隔 500 个符号的字符串。$(0\cup 1)^*0(0\cup 1)^{500}(0\cup 1)^*0(0\cup 1)^*$.

9.8 若 R 是正则表达式，令 $R^{\{m,n\}}$代表表达式 $R^m\cup R^{m+1}\cup\ldots\cup R^n$，说明怎样用通常的指数算子实现算子 $R^{\{m,n\}}$，但不许用“...”。

答：设 m<n，则可用 $R^m(R\cup\varepsilon)^{n-m}$.

9.9

For the second part we need to show that $coNP \subseteq P^{SAT}$, or in other words:

$$\overline{L} \in NP \Rightarrow L \in P^{SAT}.$$

This can be achieved via the construction of M^{SAT}, where we just invert the answer of the oracle. By this we can be sure that the new M^{SAT} decides $\overline{L}$ (in polynomial time).

For the third part we assume that $NP = P^{SAT}$, where we need to show that $NP = coNP$.
At first we show that $coNP \subseteq NP$, where we can conclude directly from our assumption $NP = P^{SAT}$, that $P^{SAT} \subseteq NP$. We showed in the previous part that $coNP \subseteq P^{SAT}$, which results in: $coNP \subseteq NP$.

We still need to show that $NP \subseteq coNP$.
The complexity class P is closed under the operation of complement, so is P^{SAT}, because we can just reinterpretate (swap) the accept and reject states of the decider for a language L. We can conclude:

$$L \in P^{SAT} \Rightarrow \overline{L} \in P^{SAT}.$$

Now we can manage to show our statement by using the precondition that $NP = P^{SAT}$, because if this would be the case any language in P^{SAT} would be in NP and vice versa. For that NP also has to be closed under the operation of complement:

$$L \in NP \Rightarrow \overline{L} \in NP,$$

which is just the same as $L \in NP \Rightarrow L \in coNP$ by the defintion of $coNP$, or in other words: $NP \subseteq coNP$.

Finally we showed the two directions: $NP \subseteq coNP$ and $coNP \subseteq NP$, under the precondition that $NP = P^{SAT}$. We can conclude that $NP = coNP$, which is exactly what we wanted to show.

Problem 9.12

Answer: SAT being in TIME(n^k) does not mean that NP $\subset$ TIME(n^k), because reductions are allowed to take more than $O(n^k)$ time.

9.13

(a) Prove that if $A \in \mathbf{TIME}(n^6)$, then $pad(A, n^2) \in \mathbf{TIME}(n^3)$.
(*Note:* This part will not be graded as we proved this in section. You need not submit the solution to this, but you can attempt this part to understand the definition.)

(b) (Sipser 9.14) Define $\mathbf{EXPTIME} = \mathbf{TIME}(2^{n^{O(1)}})$ and $\mathbf{NEXPTIME} = \mathbf{NTIME}(2^{n^{O(1)}})$. Use the function *pad* to prove that

$$\mathbf{NEXPTIME} \neq \mathbf{EXPTIME} \Rightarrow \mathbf{P} \neq \mathbf{NP}$$

[15 points]

SOLUTION:

(a) Let M be the machine that decides A in time n^6. Now, consider the machine M' for $pad(A, n^2)$ that on input x, check if x is of the format $pad(w, |w|^2)$ for some string $w \in \Sigma^*$. If not, reject. Otherwise, simulate M on w. The running time of M' is $O(|x|^3) + O(|w|^6) = O(|x|^3)$.

(b) We shall prove the contrapositive. Suppose that $\mathbf{P} = \mathbf{NP}$. Then, consider any language $L \in \mathbf{NEXPTIME}$, and let c be a positive integer such that $L \in \mathbf{NTIME}(2^{n^c})$. Then, it is easy to see that $pad(L, 2^{n^c}) \in \mathbf{NP}$. By assumption, $\mathbf{P} = \mathbf{NP}$, so $pad(L, 2^{n^c}) \in \mathbf{P}$ and therefore $L \in \mathbf{TIME}(2^{O(n^c)}) \subseteq \mathbf{EXPTIME}$. It follows that $\mathbf{EXPTIME} = \mathbf{NEXPTIME}$.

9.13 Consider the function $pad: \Sigma^* \times \mathcal{N} \longrightarrow \Sigma^*\#^*$ that is defined as follows. Let $pad(s, l) = s\#^j$, where $j = \max(0, l - m)$ and m is the length of s. Thus, $pad(s, l)$ simply adds enough copies of the new symbol # to the end of s so that the length of the result is at least l. For any language A and function $f: \mathcal{N} \longrightarrow \mathcal{N}$, define the language $pad(A, f)$ as

$$pad(A, f) = \{pad(s, f(m)) | \text{ where } s \in A \text{ and } m \text{ is the length of } s\}.$$

Prove that if $A \in \mathrm{TIME}(n^6)$, then $pad(A, n^2) \in \mathrm{TIME}(n^3)$.

Let M be the machine that decides A in time n^6. Now, consider the machine M' for $pad(A, n^2)$ that on input x, check if x is of the format $pad(w, |w|^2)$ for some string $w \in \Sigma^*$. If not, reject. Otherwise, simulate M on w. The running time of M' is $O(|x|^3) + O(|w|^6) = O(|x|^3)$.

(Sipser 9.14) Define $\mathbf{EXPTIME} = \mathbf{TIME}(2^{n^{O(1)}})$ and $\mathbf{NEXPTIME} = \mathbf{NTIME}(2^{n^{O(1)}})$. Use the function *pad* to prove that

$$\mathbf{NEXPTIME} \neq \mathbf{EXPTIME} \Rightarrow \mathbf{P} \neq \mathbf{NP}$$

We shall prove the contrapositive. Suppose that $\mathbf{P} = \mathbf{NP}$. Then, consider any language $L \in \mathbf{NEXPTIME}$, and let c be a positive integer such that $L \in \mathbf{NTIME}(2^{n^c})$. Then, it is easy to see that $pad(L, 2^{n^c}) \in \mathbf{NP}$. By assumption, $\mathbf{P} = \mathbf{NP}$, so $pad(L, 2^{n^c}) \in \mathbf{P}$ and therefore $L \in \mathbf{TIME}(2^{O(n^c)}) \subseteq \mathbf{EXPTIME}$. It follows that $\mathbf{EXPTIME} = \mathbf{NEXPTIME}$.

9.14

THEOREM 1
If $\mathbf{EXPTIME} \neq \mathbf{NEXPTIME}$ *then* $\mathbf{P} \neq \mathbf{NP}$.

PROOF: We prove the contrapositive: assuming $\mathbf{P} = \mathbf{NP}$ we show $\mathbf{EXPTIME} = \mathbf{NEXPTIME}$. Suppose $L \in \mathbf{NTIME}(2^{n^c})$. Then the following language

$$L_{\text{pad}} = \left\{ < x, 1^{2^{|x|^c}} >: \ x \in L \right\} \tag{1}$$

is in $\mathbf{NP}$ (in fact in $\mathbf{NTIME}(n)$). (Aside: this technique of adding a string of symbols to each string in the language is called *padding*.) Hence if $\mathbf{P} = \mathbf{NP}$ then L_{pad} is in $\mathbf{P}$. But if L_{pad} is in $\mathbf{P}$ then L is in $\mathbf{EXPTIME}$: to determine whether an input x is in L, we just pad the input and decide whether it is in L_{pad} using the polynomial-time machine for L_{pad}. □

9.16

4. Show that $TQBF \notin \text{DSPACE}(n^{1/4})$. You may refer to the proof of Theorem 8.9 in the text, and assume the fact that the reduction presented there can be carried out in log space.

   Solution:

   Assume to the contrary that $TQBF \in \text{DSPACE}(n^{1/4})$. We use the proof of Theorem 8.9 to show that $\text{DSPACE}(n) \subseteq \text{DSPACE}(n^{1/2})$, which contradicts the space hierarchy theorem.

   Suppose $A \in \text{DSPACE}(n)$, and let M be a TM which accepts A in space $O(n)$. Given $x \in \Sigma^*$ with $|x| = n$, the proof of Sipser's Theorem 8.9 shows how to construct a quantified Boolean formula φ_x of length $O(n^2)$ such that M accepts x iff φ_x is true. In fact the construction can be carried out in space $O(\log n)$. Let M_{tr} be a log space transducer which converts x to φ_x.

According to our assumption, there is a TM M_{TQBF} which solves $TQBF$ in space $O(n^{1/4})$. We can use M_{TQBF} together with the transducer M_{tr} to construct a TM M_A which accepts A in space $O(n^{1/2})$ as follows. M_A on input x simulates M_{TQBF} on input φ_x, without ever completely writing φ_x. Each time M_{TQBF} needs to read a symbol of φ_x, M_A simulates M_{tr} on input x (the same as the input of M_A) until M_{tr} prints the desired symbol, and then M_A feeds this to M_{TQBF}. Finally M_A accepts x iff M_{TQBF} decides that φ_x is true.

The space required by M_A is the space required by M_{tr} (which is just $O(\log n)$), together with the space required by M_{TQBF}, which by assumption is $O(m^{1/4})$, where now $m = |\varphi_x| = O(n^2)$ (see the end of the proof of Theorem 8.9). Hence the total space required by M_A is $O(n^{1/2})$, which contradicts the space hierarchy theorem, as explained above.

2

I would like some hints on how to approach this problem, I know for instance that $TQBF$ is *PSPACE-Complete*, so it can solved in poly space and any other *PSPACE-Complete* problems can be log spaced reduced to $TQBF$. I believe that I need to employ the space hierarchy theorem in some way but I am not sure how, this is a homework question so I just want a hints. Thank you!

complexity-theory space-complexity

share improve this question

asked Nov 27 at 22:03
InsigMath
28 ● 5

I cannot give you any more hints than the facts you've already mentioned. – Yuval Filmus Nov 28 at 2:12

2 By the way, TQBF being PSPACE-complete means that TQBF is in PSPACE, and *any* problem in PSPACE is polytime-reducible to it. (Perhaps even logspace-reducible.) – Yuval Filmus Nov 28 at 2:13

Thanks we were told that it is logspace reducible. I understand that you cannot give anymore hints without basically spelling out the solution. I will think more deeply about this question. Have a good evening @YuvalFilmus – InsigMath Nov 28 at 5:29

This is actually not quite as easy as I had assumed. – Yuval Filmus Dec 11 at 8:45

1 Answer

active oldest votes

3

The argument is actually surprisingly delicate. This sketch follows Tompa's *Introduction to computational complexity*, Chapter 10. The space hierarchy theorem shows that there is some problem $L \in \mathrm{SPACE}(n^{1+\epsilon}) \setminus \mathrm{SPACE}(n)$ for every $\epsilon > 0$. Since TQBF is PSPACE-complete, there is some logspace reduction from L to TQBF. But we actually know more - using Savitch's theorem, we can come up with some reduction which blows up instances of L of size n to instances of TQBF of size $O(n^{2(1+\epsilon)} \log n)$ (at least according to Ryan Williams, page 1 at the bottom). If TQBF were solvable in space $O(n^{\delta})$ then this would give an algorithm for L in space $O(n^{2\delta(1+\epsilon)})$. Taking the limit $\epsilon \to 0$, this shows that $\delta \geq 1/2$.

9.19

Theorem 1 (Valiant-Vazirani) *USAT* $\in$ *P implies NP* = *RP.*

We prove this by means of the following lemma, which effectively states that there is a randomized reduction from SAT to USAT:

Lemma 2 *In polynomial time we can probabilistically reduce a formula* $\phi \in SAT$ *to another formula* ψ *such that* $\phi \notin SAT \Rightarrow \psi \notin SAT$*, and* $\phi \in SAT \Rightarrow$ *with probability* $1/\text{poly}(n), \psi$ *is uniquely satisfiable.*

(Here n in the number of variables in *phi*). Lemma 2 implies Theorem 1 because, if USAT $\in$ P, then our RP-machine for solving SAT could just perform this reduction polynomially many times (as many as is needed to amplify the probability of finding at least one correct reduction to a constant) and accept iff our USAT solver accepts any of these reductions[1]. We proceed, then, to prove the Lemma, as outlined in the previous lecture.

1.1 Proving SAT reduces to USAT

Proof Strategy: Let $\psi(x) = \phi(x) \wedge h(x) = 0$ for a suitably chosen h.

First of all, we want to pick h from a pairwise independent, nice (as defined in the last lecture) family of functions: so we will pick our h from the family $\{h_{A,b} : x \mapsto Ax + b\}$ where all operations are done in $\mathbb{Z}_2$, A is an $m \times n$ matrix, b is an m-element vector, and all elements of A and b are chosen uniformly from $\{0, 1\}$. (Note that these functions take elements of $\mathbb{Z}_2^n$ to elements of $\mathbb{Z}_2^m$, so the phrase "$h(x) = 0$" is a vector equality.) Note that if ϕ is unsatisfiable, it is immediate that ϕ is unsatisfiable also: so henceforth, we will concern ourselves only with the case that *phi* has one or more satisfying assignments.

The question remains, what should m be? Denote the set of all satisfying assignments of ϕ by S, and let $M = |S|$. If we know the value of M, it turns out (as we will see shortly) that taking m such that $2^{m-2} \leq M \leq 2^{m-1}$ is a good choice. However, there is no cheap way to even approximate M, so what we do is the following:

Choice of m: Choose m randomly (and uniformly) from $\{2, 3, \ldots, n+1\}$.

Since ϕ has between 0 and 2^n satisfying assignments (its n boolean variables can only take on 2^n distinct values), we have a $1/n$ chance of picking the correct m.

9.20

Solution1:

The point of the proof is to show that NP^C != coNP^C for some oracle C. In order to prove this, we need to show that some language Lc exists that is not in NP^C, but IS in coNP^C (or vice versa). To do this we construct an oracle C and a language Lc that have this property. The algorithm works step by step. At each step it adds a finite number of strings to C. At the algorithm's completion C will be fully built.

First we describe the language Lc. If you note, it is based on C. So in effect, by constructing C, we are also constructing Lc.

Lc = { w in {0,1}* | for all x in C, |w| != |x| }

To summarize. A string w is in Lc if NO string with the same length is in C. w is not in Lc if there IS at least one string with identical length in C.

The strategy is simple. We must construct C in such a way that Lc is different from any language in NP^C. This would imply Lc !in NP^C. The way we do this is to make sure that for every language L in NP^C, there exists at least one string that Lc and L differ on. We arbitrarily choose this string to be 0^n, where the exact n differs for each language in NP^C. So if 0^n is in L, we construct C such that 0^n is NOT in Lc. If 0^n is not in L, we make sure 0^n IS in Lc.

The algorithm works its way, one by one, through a list of all nondeterministic polynomial time Turing machines N1,N2,...,Ni,... that recognize a language in NP^C. For each Ni, it first simulates the machine on the input 0^n and records its output - accept or reject. If it accepts, we construct C such that 0^n is not in Lc. If it rejects, we construct C so that 0^n is in Lc, thus making L(Ni) and Lc differ.

We now go into some of the details about how we do this. At each step of the algorithm (one step per NTM) we add a finite number of strings to C. We denote by Ci the current set of strings that are officially in C at step i. Now lets take the case where after simulation, Ni accepts 0^n. Therefore our job is to make sure 0^n is not in Lc. Recall, that for a string w to be out of Lc, C must contain a string of equal size. So in this case all we have to do is add a string of length n to Ci. Now what happens if Ni rejects? If Ni rejects 0^n, we have to make sure 0^n IS in Lc. However, we can't make this true by adding strings to Ci. Recall, that a string is in Lc if no string of equal length is in C. What we have to do is make sure that we never add any string of length n to C. To do this, we create a 'forbidden list', Fi. The forbidden list contains a list of all strings that we are never allowed to add to C. So to ensure that 0^n IS in Lc, we add all strings of length n to the forbidden list, and hence assurre that Lc and L(Ni) again differ.

Solution2:
Okay, I've been putting this off long enough. I was able to get a copy of the Baker, Gill, and Solovay paper, and I have to confess that I read their proof before trying to work it out on my own. Just couldn't help myself.

First, here is (roughly) the proof as Sipser gives it, that there exists a language A such that P^A != NP^A:

Define L_A = {w : |w| = |x| for some x in A}. So in other words, if some string x is in A, then every string with length |x| is in L_A. Then L_A is in NP^A for any language A. Our construction will aim to create a language A such that A is not in P^A. To do this, we diagonalize over all polynomial time oracle TMs Mi using oracle language A, as follows.

Let M1, M2, ..., Mi, ... be an enumeration of all polynomial time oracle TMs such that Mi runs in time n^i (where n is the length of the input). We construct A in stages, starting with A empty.
At stage i, choose an n such that n is greater than the size of all strings whose membership in A has been determined so far, and also so

that 2^n > n^i. Now simulate Mi on input 1^n. If Mi queries about a string which has been previously added to A, answer YES; otherwise answer NO. (In particular, if Mi queries about a string w with |w| = n, the answer will be NO.) After Mi halts, we do the following: If Mi accepts, we declare all strings of length n to be out of A. If Mi rejects, we find a string of length n which hasn't been queried yet, and put it in A. We are guaranteed that we can do this because since 2^n > n^i, Mi can't query all strings of length n. Either way, Mi responds inconsistently with L_A on input 1^n. To finish the current stage, all strings with length <= n that haven't already been determined get placed out of A.

Since no polynomial time oracle TM with oracle A correctly decides L_A, L_A is not in P^A.

Now, how can we adapt this construction to give an oracle C such that NP^C != coNP^C? We start the same way, defining L_C = {w : |w| = |x| for some x in C}, so that L_C is in NP^C. What would it mean for L_C to be in coNP^C? It would mean that the complement of L_C is in NP^C. So, we construct C so that no NTM Ni decides L_C-complement. Let N1, N2, ..., Ni, ... be an enumeration of all polynomial time oracle NTMs such that Ni runs in time n^i (where n is the length of the input). We construct C in stages, starting with C empty.

At stage i, choose an n such that n is greater than the size of all strings whose membership in C has been determined so far, and also so that 2^n > n^i. Now simulate Ni on input 1^n. If Ni queries about a string which has been previously added to C, answer YES; otherwise answer NO. (In particular, if Ni queries about a string w with |w| = n, the answer will be NO.) After Ni halts, we do the following: If Ni rejects, we declare all strings of length n to be out of C. If Ni accepts, we pick some accepting branch of Ni's computation, find a string of length n which wasn't queried in that branch, and put it in C. We are guaranteed that we can do this because since 2^n > n^i, Ni can't query all strings of length n on any particular branch. (This might change the outcome of other computational branches, but we only need one accepting branch for Ni to accept.) Either way, Ni responds consistently with L_C on input 1^n, and hence inconsistently with L_C-complement. To finish the current stage, all strings with length <= n that haven't already been determined get placed out of C.

Since no polynomial time oracle NTM with oracle C correctly decides L_C-complement, L_C-complement is not in NP^C. In other words, L_C is not in coNP^C.