OUR COVERAGE OF CHAPTERS 8 AND 9 OF SIPSER

JOEL FRIEDMAN

The following is an outline of what we have covered in Sipser's textbook Chapters 8 and 9 up to Wednesday, November 13, 2014. The rough idea is (1) to learn a bit about SPACE complexity (and how it differs from TIME), and (2) to be able to prove that $\mathrm{P}^A = \mathrm{NP}^A$ for certain oracles, and (3) describe how most people are trying to prove that P is not NP. Specifically we covered the following material:

(1) Savitch's theorem (8.1).
(2) PSPACE (8.2), which equals NPSPACE by Savitch's theorem.
(3) in lieu of (8.3), we produced our own PSPACE-complete problem, which is the language of descriptions $< M, i, s >$ of: (1) a Turing machine, M, an input i (to M), and s, an integer written in *unary* (it is crucial that s be written in unary), such that M accepts i and uses at most space s to do so; the proof is straightforward.
(4) The Time Hierarchy Theorem, Theorem 9.10 and Corollaries 9.11 and 9.12. (This was covered in October).
(5) Theorem 9.20, part (2) (more specifically, if A is any PSPACE complete language (under polynomial time reductions), then $\mathrm{P}^A = \mathrm{NP}^A = \mathrm{PSPACE}$.
(6) Theorem 9.30, that a language in $\mathrm{TIME}(t(n))$ has circuit complexity $O(t^2(n))$ (see Definitions 9.27 and 9.28) in Section 9.3.

Sample Exam Problems

(1) Problems from Sipser: Chapter 8: 8.4 (we will discuss the operation "star" when we discuss regular languages), 8.6. Chapter 9: 9.1–9.6, 9.9.
(2) Argue that if L_1, L_2 are in P then so are (1) $L_1 \cup L_2$, (2) $L_1 \cap L_2$, (3) the complement of L_1. Argue that the same is true for P replaced by P^A for any oracle, A.
(3) Do Problem 2 with P replaced by NP for parts (1) and (2). Can you do this for part (3)?
(4) We have seen two definitions for NP. One was based on a non-deterministic Turing machine; another was that a language, L, over an alphabet Σ belongs

Research supported in part by an NSERC grant.

in NP iff there is a positive integer, c, and a language, L', such that for all $w \in \Sigma^*$ we have

$$w \in L \quad \text{iff} \quad \text{there exists a } v \in \Sigma^* \text{ with } |v| = c|w|^c \text{ such that } (w,v) \in L'.$$

Explain why these two definitions of NP are equivalent. [See Section 7.3 and Theorem 7.20.]

(5) Our first discussion of the Time Hierarchy Theorem was to prove that for any positive integer, a, we have that TIME(n^a) is a proper subset of TIME(n^{2a+1}). We mentioned that a faster universal Turing machine shows that for any real numbers $b > a \geq 1$, we have that TIME(n^a) is a proper subset of TIME(n^b). For any language, A, define TIME$^A(f(n))$ to be the set of languages recognized by Turing machine with oracle A running in time $O(f(n))$. Do the hierarchy theorems hold with TIME replaced by TIME^A for any oracle, A? Outline the proof of the Hierarchy theorem to justify your claim.

(6) (a) Given a language, A, consider Turing machines endowed with the oracle A. Do Axioms 1–5 (from the handout studied in September) hold for these Turing machines? Justify your answer (you should do this axiom by axiom, although Axiom 1 and 3–5 should be almost immediate).
(b) Let A = HALT be the Halting Problem (consisting of a Turing machine, p, and an input, i, to the Turing machine such that p halts on input i). If we endow Turing machines with the oracle A, then is HALT and its complement recognizable by such machines?
(c) Describe a sequence of oracles, $A_0 = \emptyset, A_1, A_2, \ldots$ such that for any positive integer i, the set of languages recognizable by Turing machines with oracle A_{i-1} is a proper subset of those recognized by Turing machines with oracle A_i.

(7) (a) Outline the proof of the Time Hierarchy Theorem.
(b) Explain how to modify this proof to show that for any positive real numbers $b > a \geq 1$ there is a language in SPACE(n^b) that is not in SPACE(n^a). [The only technical point is to show that a Turing machine that runs in space $O(n^a)$ can be simulated in space $O(n^b)$; this is not difficult, and much easier than the same claim for "space" replaced by "time."]

(8) Let $L_{\text{NP easy}}$ consist of all descriptions of a triple, $\langle M, i, t\rangle$ where M is a non-deterministic Turing machine that accepts input i, running in time t where t is expressed in unary. Write out a careful proof that shows that this language is NP-complete (with respect to polynomial time reductions).

(9) Let $L_{\text{PSPACE easy}}$ consist of all descriptions of a triple, $\langle M, i, s\rangle$ where M is a Turing machine that accepts input i, running in space s where s is expressed in unary. Write out a careful proof that shows that this language is PSPACE-complete (with respect to polynomial time reductions).

(10) For any positive integer, n, we may interpret a word of $\{0,1\}^{n(n+1)}$ as a sequence of $n+1$ non-negative integers, each of which is specified in base 2 by a sequence of exactly n binary digits. For any positive integer, n, let

$$\text{SUBSET} - \text{SUM}_n \subset \{0,1\}^{n^2+n}$$

be those words whose associated sequence of integers (as above), m_1, m_2, m_n, t, lie in SUBSET-SUM, i.e., there is an $I \subset \{1, \ldots, n\}$ such that

$$\sum_{i \in I} m_i = t.$$

Let $f(n)$ be the minimum sized circuit that computes, for a word in $\{0,1\}^{n(n+1)}$, whether or not the word lies in $\mathrm{SUBSET} - \mathrm{SUM}_n$. Show that if we can prove that for any integer, c, we have $f(n) \geq n^c$ for sufficiently large n, then $\mathrm{P} \neq \mathrm{NP}$.

(11) For any positive integer, n, we may interpret a word of $\{0,1\}^{n(n-1)/2}$ as describing a graph on n vertices, by describing which of the $n(n-1)/2$ possible edges are contained in the graph (so by a *graph* we mean undirected graphs that have no multiple edges or self-loops). For any positive integer, n, let

$$3\mathrm{COLOR}_n \subset \{0,1\}^{n(n-1)/2}$$

be those words whose associated graph (as above) is 3-colourable. Let $f(n)$ be the minimum sized circuit that computes, for a word in $\{0,1\}^{n(n-1)/2}$, whether or not the word lies in $3\mathrm{COLOR}_n$. Show that if we can prove that for any integer, c, we have $f(n) \geq n^c$ for sufficiently large n, then $\mathrm{P} \neq \mathrm{NP}$.

(12) Explain why the set of languages that are decided by a list of circuits $C = (C_0, C_1, C_2, \ldots)$ in size $n+1$ is uncountable. Is the same true with n replaced by $\sqrt{n}+1$? Is the same true with n replaced by 3?

(13) Let $3\mathrm{COLOR}_n$ be the subset of $\{0,1\}^{n(n-1)/2}$ described in Problem 11. If we can prove that $f(n) \leq n^3$ for n sufficiently large, does it necessarily follow that $\mathrm{P} = \mathrm{NP}$?

Solutions to some problems appearing two pages from now

Solutions to some problems appear starting on the next page

Solutions to Some Sample Exam Problems

(1) Problems from Sipser: Chapter 8: 8.4 (we will discuss the operation "star" when we discuss regular languages), 8.6. Chapter 9: 9.1–9.6, 9.9.

(2) Argue that if L_1, L_2 are in P then so are (1) $L_1 \cup L_2$, (2) $L_1 \cap L_2$, (3) the complement of L_1. Argue that the same is true for P replaced by P^A for any oracle, A.

Solution: Let M_1 and M_2 be Turing machines recognizing L_1 and L_2 respectively in polynomial time. For (3) we simply interchange the accepting and rejecting states, which works also when M_1 is allowed oracle calls.

For (1) and (2) we run M_1 and M_2 in parallel on the input, which amounts to forming a new Turing machine, M, as follows:

(a) M has one tape for each tape of M_1 and each tape of M_2 (so the number of tapes of M is the sum of the number of tapes of M_1 and M_2); we designate the input tape of M_1 to be the input tape of M, and the first task of M is to copy the input to the tape of M that represents the first input tape of M_2, and then we move all tape heads to the beginning of their tapes; this task takes time linear in the size of the input.

(b) We then repeatedly run one step of M_1 and one of M_2, with a state set that is the product of the state sets of M_1 and M_2 (the state set of M contains this product set, but also has states to perform the first task above.

This parallel running of M_1 and M_2 therefore terminates in polynomial time (which is the time for task one, plus the maximum of the times that M_1 and M_2 take). In case (1), $L_1 \cup L_2$, we accept an input if M_1 or M_2 is in the accepting state; in case (2), $L_1 \cap L_2$, we accept an input if both M_1 and M_2 are in the accepting state.

In cases (1) and (2), we can allow for oracles calls; in this case M is allowed only one oracle tape, so we would have to modify M a bit: one could keep in M the oracle tapes of M_1 and M_2 as "fake oracle tapes" where a word could be written but the oracle tape of M would be an additional tape. Then at any step where one or both of M_1 and M_2 make oracle requests we would write the oracle call of M_1 and/or M_2 to M's oracle tape and make an oracle call (if both M_1 and M_2 are making oracle calls at the same step, M would have to handle these oracle calls one at a time). The additional work done by M to handle these oracle calls is linear in the length of the strings in the oracle calls, which are bounded by the time bounds of M_1 and M_2, and therefore at most a polynomial in the input size.

(3) Do Problem 2 with P replaced by NP for parts (1) and (2). Can you do this for part (3)?

(4) We have seen two definitions for NP. One was based on a non-deterministic Turing machine; another was that a language, L, over an alphabet Σ belongs in NP iff there is a positive integer, c, and a language, L', such that for all $w \in \Sigma^*$ we have

$$w \in L \quad \text{iff} \quad \text{there exists a } v \in \Sigma^* \text{ with } |v| = c|w|^c \text{ such that } (w, v) \in L'.$$

Explain why these two definitions of NP are equivalent. [See Section 7.3 and Theorem 7.20.]

(5) Our first discussion of the Time Hierarchy Theorem was to prove that for any positive integer, a, we have that TIME(n^a) is a proper subset of TIME(n^{2a+1}). We mentioned that a faster universal Turing machine shows that for any real numbers $b > a \geq 1$, we have that TIME(n^a) is a proper subset of TIME(n^b). For any language, A, define TIME$^A(f(n))$ to be the set of languages recognized by Turing machine with oracle A running in time $O(f(n))$. Do the hierarchy theorems hold with TIME replaced by TIME^A for any oracle, A? Outline the proof of the Hierarchy theorem to justify your claim.

(6) (a) Given a language, A, consider Turing machines endowed with the oracle A. Do Axioms 1–5 (from the handout studied in September) hold for these Turing machines? Justify your answer (you should do this axiom by axiom, although Axiom 1 and 3–5 should be almost immediate).

Solution: Axiom 1 just says that the Result function is defined, which is clear. Axiom 2 is the universal Turing machine for oracle machines; these universal machines work just as regular universal Turing machines work, except that oracle calls from the simulated machine involve the additional step of writing the contents of the simulated oracle tape to the oracle tape of the universal machine and calling the oracle from the universal machine. Axioms 3–5 are very easy and done just as they are done for Turing machines.

(b) Let A = HALT be the Halting Problem (consisting of a Turing machine, p, and an input, i, to the Turing machine such that p halts on input i). If we endow Turing machines with the oracle A, then is HALT and its complement recognizable by such machines?

Solution: Yes: one oracle call to HALT can tell us if an input to the Halting Problem will halt or will not, so we can use this recognize both HALT and its complement.

(c) Describe a sequence of oracles, $A_0 = \emptyset, A_1, A_2, \ldots$ such that for any positive integer i, the set of languages recognizable by Turing machines with oracle A_{i-1} is a proper subset of those recognized by Turing machines with oracle A_i.

Solution: Our approach will be to show that for any language, A, there is a language $B = B(A)$ such that the set of languages recognizable by Turing machines with oracle B is a strict superset of those with oracle A. Then we set $A_{i+1} = B(A_i)$ for $i = 0, 1, 2, \ldots$.

For any oracle, A, let $H_A = \text{HALT}^A$ be the halting problem for oracle Turing machines with oracle A. By the first part of this exercise, the complement of the Acceptance Problem for Turing machines with oracle A is not recognizable by any Turing machine with oracle A (since the set of such oracle Turing machines satisfy Axioms 1–5); hence the complement of the Halting Problem with oracle A, namely H_A, also cannot be recognized by Turing machines with oracle A. It follows that if we have a Turing machine that can "call both the oracles A and H_A," then machines with this oracle can recognize a strict superset of the

languages recognizable with oracle A. It suffices to construct an oracle that allows one to make oracle query calls to "both A and H_A" (in the evident sense, i.e., simulate the capability of writing a string on the oracle tape and querying either A or H_A).
So fix a Turing machine, and assume that 1 and 2 are letters in the worktape alphabet of the machine (if not, then just enlarge this alphabet). Let B be the language consisting of strings $1w$ such that $w \in A$ and $2w$ such that $w \in H_A$. Then by prepending either a 1 or a 2 to a string, we can call the oracle B to give an oracle call on w to either A or H_A.
It follows that for any language A, there exists a language $B = B(A)$ such that the set of langauges recognizable with oracle B is a strict superset of those recognizable with oracle A.

(7) (a) Outline the proof of the Time Hierarchy Theorem.
Solution: There are a number of possible variants; I will describe the proof we gave in class. We will need the existence of a universal Turing machine that can simulate cn^α steps of a Turing machine, M, on an input w of length $n = |w|$ in time $C\log(n)n^\alpha$, where $C = C(M, c, \alpha)$; in class we proved a weaker form of this bound, with $C\log(n)n^\alpha$ replaced with $Cn^{2\alpha}$, and this weaker result can be used to get a weaker form of the Time Hierarchy Theorem.
The result is that for real numbers $1 \ge \alpha \ge \beta$, TIME(n^α) is a strict subset of TIME(n^β). To prove this we fix a rational γ between α and β and build a Turing machine M' as follows: on input i, we see if i begins by describing a Turing machine, i.e., $i = \langle M\rangle w$ where M is a Turing machine and w is an additional "dummy padding" of i, and if so we simulate M on input i. We run our simulation for time $|i|^\gamma$ in M' (since M' may need an average of roughly $\log(|i|)$ steps to simulate each step of M, the number of simulated M steps will be fewer). If M halts on input i after this simulation, then if M accepts i we have M' reject i, and if M rejects then M' accepts. (We insist that γ is rational so that we can compute the function $f(n) = n^\gamma$ in time order n^γ, so that we don't need to spend excessive time computing just computing n^γ.)
Clearly M' runs within time order $n^\gamma \le n^\beta$. However if M_0 is a fixed Turing machine running in time order n^α, then for $i = \langle M_0\rangle w$ with $|w|$ sufficiently large, then M' differ on some inputs. The only subtle point is that the universal Turing machine simulating M_0 will never look at w to perform each step of M_0, and so the in the time $C\log(n)n^\alpha$ bound for the simulation of cn^γ steps of M_0, the constant C, which depends on the description of M_0, is unaffected by the added padding of the description of M_0 in the input. (In class we noted that we could alternatively pad the Turing machine M_0 with some "dummy states" which does the same type of padding that we need.)

(b) Explain how to modify this proof to show that for any positive real numbers $b > a \ge 1$ there is a language in SPACE(n^b) that is not in SPACE(n^a). [The only technical point is to show that a Turing machine that runs in space $O(n^a)$ can be simulated in space $O(n^b)$;

this is not difficult, and much easier than the same claim for "space" replaced by "time."]

Solution: By the above solution, given real numbers $1 \leq \alpha < \beta$ with γ rational, we need to show that there is a universal Turing machine that simulates a machine running M using space bounded by order n^α in a simlutation that doesn't use more than order n^β space. So fix a rational number, γ, and simulate M on input w, making sure to stop the simulation once any of the following hold: (1) M halts; (2) M takes more than n^γ space; and/or (3) M takes more than 2^{n^γ} time. Since M has a fixed number of tapes, (2) requires at most space $C_M n^\gamma$, and (3) requires at most space n^γ (for a counter from 1 to 2^{n^γ}. Notice that the above universal Turing machine on bounded space is much easier and stronger than the time bound. Simulating $f(n)$ steps of a Turing machine, M, can be done easily in time $C_M(f(n))^2$, and by a more difficult simulation in time $C_M f(n)\log(f(n))$. But a simulation of a Turing machine, M, for space $f(n)$, can be done in time $C_M f(n)$ (where the constant C_M involves the space of tape of the simulating machine and a counter so that we stop after $C^{f(n)}$ steps to avoid infinite loops); further this proof is quite easy. So the Space Hierarchy Theorem gets rid of the $log(f(n))$ factor, and formally says that if $f(n)$ is computable in space $f(n)$, then SPACE($f(n)$) contains SPACE($g(n)$) for any function $g(n)$ with $g(n) = o(f(n))$.

(8) Let $L_{\text{NP easy}}$ consist of all descriptions of a triple, $\langle M, i, t\rangle$ where M is a non-deterministic Turing machine that accepts input i, running in time t where t is expressed in unary. Write out a careful proof that shows that this language is NP-complete (with respect to polynomial time reductions).

(9) Let $L_{\text{PSPACE easy}}$ consist of all descriptions of a triple, $\langle M, i, s\rangle$ where M is a Turing machine that accepts input i, running in space s where s is expressed in unary. Write out a careful proof that shows that this language is PSPACE-complete (with respect to polynomial time reductions).

Solution: First we show that $L_{\text{PSPACE easy}}$ is in PSPACE. In fact, this follows from the universal Turing machine that simulates a Turing machine, M, on input i, of bounded space s in space as described in Problem 7(b). We just need to be careful about the resources needed. First we extract i and copy i onto the worktape of the universal machine; this takes space at most $\langle M, i\rangle$, since we have arranged the input as $\langle M, i, s\rangle$, and we do not need to scan past i. Now the simulation described in Problem 7(b) requires space $C_M|s|$; we just have to be careful to bound C_M by a polynomial in $\langle M\rangle$ (which would therefore be bounded by a polynomial in the size of the input, which is $\langle M, i, s\rangle$).

The constant C_M comes from the number of tapes of M; since the transition function is

$$\delta\colon Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, S, R\}^k,$$

multiplexing k tapes with tape symbols Γ is polynomial in the description of δ. Strictly speaking, if M on input i and space s loops infinitely, then we are allowed to loop infinitely; hence we don't have to worry about counting the number steps to detect an infinite loop (but we could easily do so).

Next we show that if $L \in PSPACE$, then $L \le L_{\text{PSPACE easy}}$. Indeed, if M is Turing machine recognizing L in cn^a for some integers c, a, then given input w let

$$f(w) = \langle M, w, 1^{c|w|^a} \rangle.$$

The function f can be computed in the sum of the following time: (1) constant time to write down a description of M, (2) time order $|w|$ to tack on w, (3) time polynomial in $|w|$ to compute $c|w|^a$ and tack on this many 1's to the description of M and w. Hence f can be computed in polynomial time in w, and clearly

$$w \in L \iff f(w) \in L_{\text{PSPACE easy}}.$$

(10) For any positive integer, n, we may interpret a word of $\{0,1\}^{n(n+1)}$ as a sequence of $n+1$ non-negative integers, each of which is specified in base 2 by a sequence of exactly n binary digits. For any positive integer, n, let

$$\text{SUBSET} - \text{SUM}_n \subset \{0,1\}^{n^2+n}$$

be those words whose associated sequence of integers (as above), m_1, m_2, m_n, t, lie in SUBSET-SUM, i.e., there is an $I \subset \{1, \ldots, n\}$ such that

$$\sum_{i \in I} m_i = t.$$

Let $f(n)$ be the minimum sized circuit that computes, for a word in $\{0,1\}^{n(n+1)}$, whether or not the word lies in $\text{SUBSET} - \text{SUM}_n$. Show that if we can prove that for any integer, c, we have $f(n) \ge n^c$ for sufficiently large n, then P $\ne$ NP.

Solution: The essential idea is that if SUBSET-SUM is in P, then there are polynomial sized circuits that take a description of a word, w, and compute whether or not w lies in SUBSET-SUM. Hence there are polynomial size circuits to compute SUBSET-SUM; hence a superpolynomial lower bound on the size of circuits would contradict that the fact that SUBSET-SUM is in P.

The only technicality to consider is that w is a word in some alphabet like $\Sigma = \{0, 1, \ldots, 9, \#\}$, and the input Boolean variables in the aforementioned circuits are a list y_{ij} which is true iff the i-th letter of w is the j-th letter of Σ. It is straightforward, albeit a bit tedious, to show that the above y_{ij} give us the x_{ij} we desire.

In more detail, any SUBSET-SUM problem in Σ^n has all its integers having at most n base 10 digits (recall we are using base 10 for SUBSET-SUM, but binary x_{ij} involve a base 2 representation) and has a total of at most $(n+1)/2$ different integers (the integers of this problem are separated by #'s). Given the x_{ij} of such a SUBSET-SUM problem, we can compute the corresponding y_{ij} in a circuit of polynomial size in n (this circuit involves converting base 2 numbers to base 10). Then we can feed the y_{ij} into polynomial size circuits for the $\text{SUBSET} - \text{SUM}_n$ problem, assuming that SUBSET-SUM is in P. Hence a superpolynomial lower bound on the size of such circuits would be impossible.

(11) For any positive integer, n, we may interpret a word of $\{0,1\}^{n(n-1)/2}$ as describing a graph on n vertices, by describing which of the $n(n-1)/2$ possible edges are contained in the graph (so by a *graph* we mean undirected

graphs that have no multiple edges or self-loops). For any positive integer, n, let

$$3\mathrm{COLOR}_n \subset \{0,1\}^{n(n-1)/2}$$

be those words whose associated graph (as above) is 3-colourable. Let $f(n)$ be the minimum sized circuit that computes, for a word in $\{0,1\}^{n(n-1)/2}$, whether or not the word lies in $3\mathrm{COLOR}_n$. Show that if we can prove that for any integer, c, we have $f(n) \geq n^c$ for sufficiently large n, then $\mathrm{P} \neq \mathrm{NP}$.

(12) Explain why the set of languages that are decided by a list of circuits $C = (C_0, C_1, C_2, \ldots)$ in size $n+1$ is uncountable. Is the same true with n replaced by $\sqrt{n}+1$? Is the same true with n replaced by 3?

Solution: The same is true with n replaced by 1: consider the Boolean functions that depend only on the size of the input. If a function for a given size of input is constant, then it is computed by a circuit of size 1 (provided that we don't count the inputs $x_1, \ldots, x_n$ as part of the size; if we do, then these are circuits of size $n+1$). But the number of such functions is as large as the number of subsets of the positive integers, which uncountable.

(13) Let $3\mathrm{COLOR}_n$ be the subset of $\{0,1\}^{n(n-1)/2}$ described in Problem 11. If we can prove that $f(n) \leq n^3$ for n sufficiently large, does it necessarily follow that $\mathrm{P} = \mathrm{NP}$?

Solution: Not in general. If the circuits are simple enough to be generated by some Turing machine in polynomial time (such circuits are usually called "uniform circuits"), then yes. However, there is no reason that this need be true; indeed, byt last problem, the number of different functions that a sequence $C_0, C_1, C_2, \ldots$ can compute is uncountable; so there certainly are many circuit sequences that cannot be generated by Turing machines. [Most people believe that there aren't polynomial sized circuits for any NP-complete problem, so showing the contrary would be of great interest.]

Department of Computer Science, University of British Columbia, Vancouver, BC V6T 1Z4, CANADA, and Department of Mathematics, University of British Columbia, Vancouver, BC V6T 1Z2, CANADA.

E-mail address: `jf@cs.ubc.ca or jf@math.ubc.ca`

URL: `http://www.math.ubc.ca/~jf`