CSE 105: Final Solution

June 14, 2006

No books, no calculators. One 8.5x11 page of handwritten notes.

Name: ____________________________

Student ID:_______________________

Section (circle one): 8AM 1PM 2PM 3PM

<table>
<thead>
<tr>
<th>Problem</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/15</td>
</tr>
<tr>
<td>2</td>
<td>1/18</td>
</tr>
<tr>
<td>3</td>
<td>1/18</td>
</tr>
<tr>
<td>4</td>
<td>1/14</td>
</tr>
<tr>
<td>5</td>
<td>1/15</td>
</tr>
<tr>
<td>6</td>
<td>1/15</td>
</tr>
<tr>
<td>7</td>
<td>1/5</td>
</tr>
<tr>
<td>Extra Credit</td>
<td>1/15</td>
</tr>
<tr>
<td>Total</td>
<td>1/100</td>
</tr>
</tbody>
</table>
1. **15 pts.** Determine whether regular languages are closed under the following operations (for example, if L is a regular language, is \(T hree(L)\) necessarily a regular language?). Prove your answer.

(a) \(T hree(L) = \{x| x \in L \text{ and } |x| \text{ is a multiple of 3}\}\)

*Solution:* The set of all strings whose length is a multiple of 3 is a regular language, call it S. \(T hree(Lx)\) is the intersection of \(L\) and \(S\), and since regular languages are closed under intersection, \(T hree(L)\) is regular.

(b) \(Double(L) = \{x| x = vw, v \in L, w \in L, \text{ and } |v| = |w|\}\)

*Solution:* Not regular.

We’ll use a counter-example. Let \(L = 1^*0\). Then \(Double(L) = 1^n01^n0\).

Assume for the sake of contradiction that \(Double(L)\) is regular. Let \(p\) be as in the pumping lemma. Then \(1^n01^n0 \in Double(L)\). But, by the pumping lemma, that means there exists a \(x, y, z\) such that \(|xy| \leq p, |y| > 0, \text{ and } \forall i, xy^iz \in Double(L)\). But since \(|xy| \leq p\), that means \(y\) is completely contained within the first 1’s. So, letting \(i = 0\), \(xy^iz = xz \notin Double(L)\) since it’s of the form \(1^q01^p0\) where \(q < p\). Thus, our assumption that \(Double(L)\) is regular is incorrect.
2. 18 pts. Prove that the following language is undecidable (you may use the fact that $A_{TM}$ and $PCP$ are undecidable, but may not use Rice’s theorem):

The set of all standard one-tape Turing Machines that write a different symbol on their leftmost cell than was there on the original input.

**Solution:** Let’s give a formal description: $L = \{ \langle M \rangle | M$ is a standard one-tape TM that writes a symbol on its leftmost cell that is different from what was there on the original input $\}$. 

Assume for the sake of argument that $L$ is decidable (recursive). Then, there’s a Turing Machine $TM_L$ that decides $L$.

Create a new machine $S$:

$S =$

(a) On input $\langle M, w \rangle$, create a TM $M'$:

$M' =$ “On input $\langle w' \rangle$:

i. Move the tape contents to the right two cells, and then write a # on the second tape cell (# is some tape symbol not in the input alphabet of $M'$).

ii. Simulate $M$ on $w$ using only the portion of the tape after the #.

iii. If the simulation accepts, move to the leftmost tape cell and write a #.”

(b) Simulate $TM_L$ on $M'$. If the simulation accepts, accept, else reject.”

$S$ decides $A_{TM'}$:

- If $\langle M, w \rangle \in A_{TM}$, $M'$ will write a # on its leftmost tape cell (different from what was there on the original input), and so $S$ will accept.

- If $\langle M, w \rangle \notin A_{TM}$, $M'$ won’t write on the leftmost tape cell a different symbol (in any of its steps), and thus $S$ will reject.
3. **18 pts.** Classify the following languages as recursive (decidable), recursively enumerable (Turing-recognizable), or not recursively enumerable (not recognizable). You need not justify your answer; just specify what class it is in.

(a) The set of all Turing Machines that accept a recursively enumerable (Turing-recognizable) language.

*Solution:* Recursive. By definition, what a TM accepts is a recursively enumerable language, thus this is just the set of all Turing Machines, which is certainly recursive (just verify the encoding is valid).

(b) The set of all Turing Machines that write a different symbol on any cell than was there on the original input.

*Solution:* Recursive. Let $L$ be the given language. $L^C$ is those Turing Machines that don’t write any different symbols on any cell than were there on the original input. This is decidable because there are only a finite number of different configurations the TM can be in (input length + 1 times number of states). If a configuration is repeated, the TM is in an infinite loop.

Thus, $L^C$ can be decided by simulating the given TM for the max number of configurations. If it accepts within that time, accept, otherwise reject. Since recursive languages are closed under complement, $L$ is recursive.

(c) The set of all pairs of context-free grammars, $G$, and regular expressions $R$ such that their languages are the same.

*Solution:* Not recursively enumerable. Not recursive since we could reduce PCP to this problem (use $\Sigma^*$ as $R$). Not recursively enumerable since the complement is recursively enumerable (non-deterministically generate a string and see whether its in $L(G)$ and $L(R)$; if they’re different, accept).

(d) The set of all context-free grammars that generate a finite language.

*Solution:* Recursive. Here’s a decision procedure: given $G$, convert it to Chomsky Normal Form. Remove all useless symbols (unreachable non-terminals, or non-terminals that don’t lead to terminals). Then, see whether there’s a cycle from any non-terminal to itself. If so, $G$ is infinite, otherwise it is finite.
(e) The set of all Turing Machines that don’t accept themselves.
Solution: Not recursively enumerable. If it were, then since its complement (see next item) is recursively enumerable, both would be recursive. But it’s easy to show that these languages are undecidable.

(f) The set of all Turing Machines that accept themselves.
Solution: Recursively enumerable. Here’s a decision procedure: given $M$, simulate it on itself. If it accepts, accept.

(g) The set of all Turing Machines that accept themselves within 100 steps.
Solution: Recursive. Simulate $M$ on $M$ for 100 steps. If it accepts within those 100 steps, accept; otherwise reject.

(h) The set of all Turing Machines that accept something.
Solution: Recursively enumerable. Easy to show undecidable. Here’s a recognizer. Given $M$, simulate all inputs (starting with length 0, then 1, etc.). Run inputs of length $\leq 1$ for 1 step, then of length $\leq 2$ for 2 steps, etc. If any accepts, accept.

(i) The set of all Turing Machines that accept nothing.
Solution: Non-recursively enumerable. Otherwise, the last item, its complement, would be recursive.
4. **14 pts.** Circle all of the following that are *known* to be true:

(a) There is no polynomial-time deterministic algorithm to solve the 3-SAT problem.

*Solution:* False. There is no known algorithm. If $P = NP$, then there is such an algorithm.

(b) Graph isomorphism ($\{(G_1, G_2) | G_1 \text{ is isomorphic to } G_2\}$) is a problem in NP (guess a mapping from $G_1$ to $G_2$ and verify), but is not known to be NP-complete. If $P \neq NP$, then graph isomorphism $\notin P$.

*Solution:* False. It could be that Graph isomorphism problem is in $P$ but we just don’t know it: nobody has discovered a deterministic polynomial time algorithm for it. If one were discovered, it wouldn’t shed any light on the $P = NP$ question, since graph isomorphism isn’t known to be NP-complete.

(c) If a language $L$ is NP-complete and a polynomial-time deterministic Turing machine exists that decides $L$, then $P=NP$.

*Solution:* True. If $L$ is NP-complete, then there is a deterministic polynomial-time reduction from any problem in NP to $L$. Thus, any problem in NP has a deterministic polynomial-time algorithm.

(d) $P$ is closed under complement.

*Solution:* True. A language in $P$ has a decidable TM that runs in deterministic polynomial time. Simply create a new TM that accepts when the first rejects, and vice-versa. This new one also runs in deterministic polynomial time.

(e) NP is closed under complement.

*Solution:* Not known to be true (would be true if $P=NP$). A language in NP has a decidable TM that runs in nondeterministic polynomial time. Thus, at least one computation path accepts (an OR of all computation paths). The complement would need an AND.

As an example, graph isomorphism is in NP, but graph non-isomorphism is not known to be.

(f) Any deterministic Turing Machine that uses at most $S(n)$ space uses at most $S(n)$ time.
Solution: False. It can use up to $k^{S(n)}$ space.

(g) Any deterministic Turing Machine that uses at most logarithmic space uses at most polynomial time.

Solution: True. There are at most polynomial different tape configurations; the machine must halt before re-entering a configuration.
5. 15 pts. Give a grammar to recognize the following language:
0^i1^j where \( i < j \).

Solution:

\[
S \rightarrow S1 | E \\
E \rightarrow 0E1 | 1
\]

\( E \) generates strings of the form \( 0^n1^{n+1} \). \( S \) adds on extra 1s at the end.
6. **15 pts.** Give a pushdown automata to recognize the following language: \(0^n1^m2^{n+m}\) where \(n,m \geq 0\).

**Solution:** We'll push an X symbol when we see a 0 or 1, and then pop it off when we see a 2. If the stack is empty when we reach the end of the string, we accept. We must be careful to deal with empty 0’s 1’s, or 0’s, 1’s, and 2’s.
7. 5 pts.

(a) What are three things you liked most about this class?

(b) What are three things you liked least about this class?

(c) What would you like to see changed?

(d) What did you think of having weekly quizzes comprised directly from homework questions? Would you have preferred traditional turned-in homework?
8. 15 pts. Extra Credit.

Let \( L = \{ x_1 x_2 \# \ldots \# x_m | x_i \in \{0, 1\}^*, \text{ and } x_i = x_j \text{ for some } i \neq j \} \). For both parts of this question, give a high-level English description of the TM. You need not give states or transitions, but do specify how it works, and what is stored on the tape.

(a) Give an efficient deterministic two-tape TM that decides \( L \). What are the space and time requirements of this machine (in \( O \)-notation)?

**Solution:** The TM will copy the tape contents to the second tape, and then go through checking \( x_1 \) from the first tape against all the \( x_i \)'s on the second tape. Then, it’ll check \( x_2 \) from the first tape against all the \( x_i \)'s \((i > 2)\) on the second tape, and so on. If it ever finds a pair that are equal, it’ll accept. If it gets all the way to the end of the first tape, it’ll reject.

i. Copy the first tape to the second tape, leaving the tape head at the left on both tapes.
ii. Mark the current \( x_i \) on the second tape. If at end of tape, reject
iii. Scan the current \( x_j \) on the first tape, matching it against each of the \( x_i \)'s on the second tape in turn. If a match occurs, accept.
iv. Move the second tape to the right past either a \# or a blank.
v. Go to step ii).

The space requirements are \( O(n) \). No tape cells outside of the original tape contents are written and a copy of the first tape is written to the second.

The time requirements are \( O(n^2) \). In the worst-case, when no \( x_i \)'s are equal, and \( m = n/2 \), there are \( n - 1 + n - 3 + \cdots + 1 \) steps = \( \Theta(n^2) \) steps.

(b) Give a non-deterministic two-tape TM that decides \( L \), but saves time by using non-determinism. What are the space and time requirements of this machine (in \( O \)-notation)?

**Solution:** The TM will use the second tape as a copy of the first tape. It’ll guess the \( i \) and \( j \) and then verify \( x_i = x_j \).
i. Copy the first tape to the second tape, leaving the tape head at the left on both tapes.

ii. Move to the right on the input tape and second tape simultaneously. When a # is reached, nondeterministically choose to keep moving the second tape.

iii. When a # is reached on the second tape, nondeterministically choose to continue with step iii) or go on to step iv).

iv. Compare the 0’s and 1’s under the two tapes until a # (or blank) is reached. If they are the same, accept, otherwise, reject.

The space requirements are $O(n)$. No tape cells outside of the original tape contents are written and the second tape writes $n$ tape cells.

The time requirements are $O(n)$. In the worst-case, it copies the original input once, and then goes through almost all of the first tape and all of the second tape.