Additional CS103 Practice Final Exam Solutions

Here are the solutions to the first extra practice final exam. We strongly recommend working through the exam under realistic conditions (three hours, open-book, open-note, open-course-website, closed everything else) before reading these solutions.

Problem One: Discrete Mathematics (25 Points)

Prove that for any nonempty set S, we have $|2^S| = |\wp(S)|$. You may find the following definition useful: given two functions $f : A \to B$ and $g : A \to B$, we have $f = g$ iff for all $a \in A$, $f(a) = g(a)$. Your proof should work for all sets S, including infinite sets.

Proof: We exhibit a bijection $g : 2^S \to \wp(S)$, from which we get that $|2^S| = |\wp(S)|$. Define the function $g : 2^S \to \wp(S)$ as $g(f) = \{ x \in S \mid f(x) = 1 \}$. We claim that this function is a bijection and prove this by showing it is both surjective and injective.

First, we prove that g is surjective. To see this, consider any $T \in \wp(S)$. Then consider the function f defined as follows:

$$f(x) = \begin{cases} 1 & \text{if } x \in T \\ 0 & \text{otherwise} \end{cases}$$

We claim that $g(f) = T$. To see this, note that $g(f) = \{ x \in S \mid f(x) = 1 \}$. By construction, $f(x) = 1$ iff $x \in T$, so $\{ x \in S \mid f(x) = 1 \} = \{ x \in S \mid x \in T \} = T$. Therefore, $g(f) = T$, as required.

Next, we prove that g is injective. To see this, consider any f_1 and f_2 such that $g(f_1) = g(f_2)$. This means that $\{ x \in S \mid f_1(x) = 1 \} = \{ x \in S \mid f_2(x) = 1 \}$, which in turn means that for any $x \in S$, we have that $f_1(x) = 1$ iff $f_2(x) = 1$. We claim that from this fact, we can prove that $f_1 = f_2$. We will prove this by showing that $f_1(x) = f_2(x)$ for all $x \in S$.

Consider any $x \in S$. Then if $f_1(x) = 1$, we have $f_2(x) = 1$, and so $f_1(x) = f_2(x)$. Otherwise $f_1(x) = 0$, and so $f_1(x) \neq 1$. Therefore, $f_2(x) \neq 1$, so $f_2(x) = 0$, and we have that $f_1(x) = f_2(x)$. Therefore, for every $x \in S$ we see $f_1(x) = f_2(x)$, so by definition $f_1 = f_2$. Therefore, if $g(f_1) = g(f_2)$, then $f_1 = f_2$, as required.

Since g is injective and surjective, it is a bijection, so $|2^S| = |\wp(S)|$, as required. ■

The most common mistake we saw on this problem was giving an intuitive justification as to why $|\wp(S)| = |2^S|$ without formalizing the intuition as a bijection. One common explanation was that there are $2 \times 2 \times \ldots \times 2$ ($|S|$ times) possible subsets of S, since each element can either be chosen or not chosen, and there are $2 \times 2 \times \ldots \times 2$ ($|S|$ times) possible functions from S to $\{0, 1\}$, since each element can be assigned 0 or 1. This is a great intuition, but this reasoning doesn't scale up to the case where $|S|$ is infinite: what is $2 \times 2 \times \ldots$ infinitely many times?

Another common mistake was to try to prove the theorem by induction on the size of S. This proof will work for all finite sets S, but will not work for infinite sets. The reason for this is that a proof by induction can be used to prove that for any set S where $|S| \in \mathbb{N}$, the theorem holds. However, induction only proves that a result holds for all natural numbers, not that it holds for all infinite cardinalities.

Problem Two: Regular Languages (40 Points Total)

(i) Rock, Paper, Scissors (20 Points)

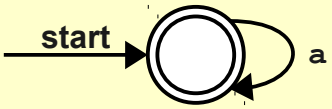
The number of characters in a regular expression is defined to be the total number of symbols used to write out the regular expression. For example, **(a|b)*** is a six-character regular expression, and **ab** is a two-character regular expression.

Let Σ = {**a**, **b**}. Find examples of all of the following:

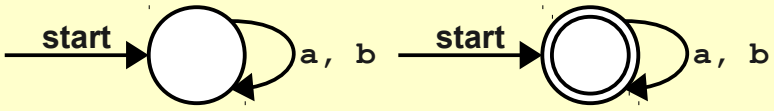
- A regular language over Σ with a one-state NFA but no one-state DFA.
- A regular language over Σ with a one-state DFA but no one-character regular expression.
- A regular language over Σ with a one-character regular expression but no one-state NFA.

Prove that all of your examples have the required properties.

The language $L = \{ \mathbf{a}^n \mid n \in \mathbb{N} \}$ has the following one-state NFA:



However, it has no one-state DFA. This is because there are only two possible one-state DFAs over Σ = {**a**, **b**}, which are given below:



The first of these has language $\emptyset \neq L$ and the second has language $\Sigma^* \neq L$, so there is no one-state DFA for L.

The language Σ* has a one-state DFA (given above). However, it has no one-character regular expression. This is because the only possible one-character regular expressions are **a**, **b**, **Σ**, **ε**, and **∅**, which have languages {**a**}, {**b**}, {**a**, **b**}, {**ε**}, and Ø, respectively, none of which are Σ*.

The language {**a**} has a one-character regular expression (namely, **a**). However, it has no one-state NFA. To see this, note that if the one state in an NFA is accepting, then the NFA accepts ε, but $\varepsilon \notin \{\mathbf{a}\}$. If it is not accepting, then the NFA never accepts, so it has language $\emptyset \neq \{\mathbf{a}\}$.

Common mistakes on this problem included forgetting that it is possible to build a one-state NFA or DFA for the empty language, or that it was not possible to build a one-character regular expression for the empty language. Another common mistake was to try to prove that no one-state automaton existed for a language by giving an example of an automaton for a language and claiming without proof that it was the minimal automaton for that language.

(ii) Nonregular Languages **(20 Points)**

A natural number $n > 1$ is called *composite* iff it can be written as $n = rs$ for natural numbers r and s, where $r \geq 2$ and $s \geq 2$. A natural number $n > 1$ is called *prime* iff it is not composite.

Let $\Sigma = \{ \mathtt{a} \}$ and consider the language $L = \{ \mathtt{a}^n \mid n \text{ is prime} \}$. For example:

- $\varepsilon \notin L$
- $\mathtt{a} \notin L$
- $\mathtt{a}^2 \in L$
- $\mathtt{a}^3 \in L$
- $\mathtt{a}^4 \notin L$
- $\mathtt{a}^5 \in L$
- $\mathtt{a}^6 \notin L$
- $\mathtt{a}^7 \in L$
- $\mathtt{a}^8 \notin L$

Prove that L is not regular. You may want to use the fact that for every natural number n, there is a prime number p such that $p > n$.

Review the proof in Problem Set 6 for a solution to this problem. When we gave out this exam last winter, we expected students to use the pumping lemma for regular languages in this proof. Since we didn't cover the pumping lemma here, the intended solution probably wouldn't be all that instructive.

Problem Three: Context-Free Languages (30 Points Total)

(i) Context-Free Grammars (20 Points)

Let $\Sigma = \{\mathbf{a}, \mathbf{b}\}$ and let $L = \{ \mathbf{a}^n\mathbf{b}^n \mid n \in \mathbb{N} \}$. The complement of this language is the language $\overline{L}$. For example:

- $\mathbf{abb} \in \overline{L}$
- $\mathbf{aab} \in \overline{L}$
- $\mathbf{baab} \in \overline{L}$
- $\mathbf{abab} \in \overline{L}$
- $\varepsilon \notin \overline{L}$
- $\mathbf{ab} \notin \overline{L}$
- $\mathbf{aabb} \notin \overline{L}$
- $\mathbf{aaabbb} \notin \overline{L}$

Write a context-free grammar that generates $\overline{L}$, then give derivations for the four strings listed in the left-hand column.

(Hint: There are several separate cases you need to consider. You might want to design the grammar to consider each of these cases independently of one another.)

There are *many* possible grammars that work here. One useful observation is that a string is in the language $\overline{L}$ iff it has the form $\mathbf{a}^m\mathbf{b}^n$, where $m \neq n$, or it contains a **b** before an **a**. This gives the following grammar:

$$S \rightarrow E \mid N\mathbf{ba}N$$

$$E \rightarrow \mathbf{a}E\mathbf{b} \mid A \mid B$$

$$A \rightarrow A\mathbf{a} \mid \mathbf{a}$$

$$B \rightarrow B\mathbf{b} \mid \mathbf{b}$$

$$N \rightarrow N\mathbf{a} \mid N\mathbf{b} \mid \varepsilon$$

Here, the nonterminal E generates a string of the form $\mathbf{a}^n\mathbf{b}^m$, where $n \neq m$, and and the nonterminal N produces any string in Σ^*.

The derivations with this grammar are

$$S \Rightarrow E \Rightarrow \mathbf{a}E\mathbf{b} \Rightarrow \mathbf{a}B\mathbf{b} \Rightarrow \mathbf{abb}$$

$$S \Rightarrow E \Rightarrow \mathbf{a}E\mathbf{b} \Rightarrow \mathbf{a}A\mathbf{b} \Rightarrow \mathbf{aab}$$

$$S \Rightarrow N\mathbf{ba}N \Rightarrow \mathbf{ba}N \Rightarrow \mathbf{ba}N\mathbf{b} \Rightarrow \mathbf{ba}N\mathbf{ab} \Rightarrow \mathbf{baab}$$

$$S \Rightarrow N\mathbf{ba}N \Rightarrow \mathbf{a}N\mathbf{ba}N \Rightarrow \mathbf{aba}N \Rightarrow \mathbf{abab}N \Rightarrow \mathbf{abab}$$

Common mistakes included accidentally missing one family of strings that should be generated, or inadvertently including some number of strings of the form $\mathbf{a}^n\mathbf{b}^n$.

(ii) Disjoint Unions **(10 Points)**

Let $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ and let L_1 and L_2 be arbitrary context-free languages over Σ. Prove that $L_1 \uplus L_2$ is context-free as well. As a reminder,

$$L_1 \uplus L_2 = \{\ \mathtt{0}w \mid w \in L_1\ \} \cup \{\ \mathtt{1}w \mid w \in L_2\ \}$$

Proof: Since L_1 and L_2 are CFLs, there exist grammars G_1 and G_2 for L_1 and L_2. Consider any two such grammars that have no nonterminals in common. Let S_1 and S_2 be the start symbols for these grammars, and consider the new CFG G defined by taking all productions from L_1 and L_2, then adding a new start symbol S with the following productions:

$$S \to \mathtt{0}S_1 \mid \mathtt{1}S_2$$

We claim that $\mathscr{L}(G) = L_1 \uplus L_2$. To see this, note that $w \in \mathscr{L}(G)$ iff it is generated by G. This in turn happens iff w is derived from $\mathtt{0}S_1$ or $\mathtt{1}S_2$. Since S_1 and S_2 are the start symbols for grammars G_1 and G_2, respectively, the strings derivable from S_1 are precisely the strings in L_1 and the strings derivable from S_2 are precisely the strings in L_2. Therefore, w is derived from $\mathtt{0}S_1$ or $\mathtt{1}S_2$ iff w has the form $\mathtt{0}x$ for some $x \in L_1$ or $\mathtt{1}x$ for some $x \in L_2$. This, in turn, is equivalent to the statement that $w \in L_1 \uplus L_2$. Combining these statements together, we see that $w \in \mathscr{L}(G)$ iff it's the case that $w \in L_1 \uplus L_2$. Therefore, $\mathscr{L}(G) = L_1 \uplus L_2$, as required. ■

The most common mistake on this problem was stating the construction without offering a proof that it was correct. When grading this problem, we were a bit lenient on the rigor of the proof, since we hadn't covered anything like it in lecture or the problem sets.

Problem Four: R, RE, and co-RE Languages (55 Points Total)

(i) The Halting Problem (15 Points)

Prove or disprove: for any TMs H and M and any string w, if H is a recognizer for $HALT$ and M loops on w, then H loops on $\langle M, w \rangle$.

Disproof: Let L be the TM "On input w, loop infinitely" and let H be the following TM:

H = "On input $\langle M, w \rangle$:

If $M = L$, reject.

Otherwise, run M on w.

If M halts, accept."

Then $\mathscr{L}(H) = HALT$, since H accepts $\langle M, w \rangle$ iff $M \neq L$ and M halts on w. Since L never halts, this is equivalent to saying that H accepts $\langle M, w \rangle$ iff M halts on w. However, H does indeed halt on any input of the form $\langle L, w \rangle$, even though L loops infinitely on w. ■

This was one of the trickiest questions on the exam. Although it is true that *in general* there is no way to learn what a TM will do on a string without running it, it is possible to determine what certain TMs will do without running them if you already know something about the TM.

Common mistakes included assuming that the only possible TM for $HALT$ must be a TM that runs M on w and accepts if M halts; or saying that if H were to reject $\langle M, w \rangle$, then it must be a decider (which is not quite true – if H rejects *every* $\langle M, w \rangle$ where M loops on w, then it would be a decider); or correctly noting that H could potentially reject an $\langle M, w \rangle$ but not demonstrating that this is actually possible.

(ii) RE Languages **(15 Points)**

A *palindrome number* is a number whose base-10 representation is a palindrome. For example, 1 is a palindrome number, as is 14941 and 7897987.

Consider the following language:

$L = \{ \langle n \rangle \mid n \in \mathbb{N}$ and there is a number $k \in \mathbb{N}$ where $k > 0$ and nk is a palindrome number $\}$

For example, $\langle 106 \rangle \in L$ because $106 \times 2 = 212$, which is a palindrome number. Also, $\langle 29 \rangle \in L$, because $29 \times 8 = 232$, which is a palindrome number.

Prove or disprove: $L \in$ **RE**.

This language is **RE** because we can build an NTM for it:

M = "On input $\langle n \rangle$, where $n \in \mathbb{N}$:

Nondeterministically guess a number $k > 0$.

Deterministically compute nk.

If nk is a palindrome, accept.

If nk is not a palindrome, reject."

This TM recognizes L, since if n has a multiple that is a palindrome L can nondeterministically guess that multiple and deterministically check it. If n does not have a multiple that is a palindrome, then L will never be able to guess a number k such that nk is a palindrome.

Common mistakes included inadvertent circular reasoning by assuming a TM existed for the language and then using that TM as a subroutine in a larger construction, or giving a valid TM but having a slight logic error in the proof, claiming that the language was not **RE** by attempting to reduce a language like L_D to it, or creating a machine like the one above but then continuing to argue that it is a decider, not just a recognizer.

Interestingly, it turns out that this language is actually decidable using some clever observations from number theory. As a challenge problem, try finding an algorithm for solving this problem!

(iii) Unsolvable Problems **(25 Points)**

Consider the following language *DECIDER*:

$$DECIDER = \{ \langle M \rangle \mid M \text{ is a decider} \}$$

Prove that *DECIDER* $\notin$ **RE** and *DECIDER* $\notin$ co-**RE**. We recommend using a mapping reduction involving the language A_{ALL} from Problem Set 8, which is neither **RE** nor co-**RE**. For reference:

$$A_{ALL} = \{ \langle M \rangle \mid M \text{ is a TM and } \mathscr{L}(M) = \Sigma^* \}$$

Proof: We will prove that $A_{ALL} \leq_M DECIDER$; since $A_{ALL} \notin$ **RE** and $A_{ALL} \notin$ co-**RE**, this means that *DECIDER* $\notin$ **RE** and *DECIDER* $\notin$ co-**RE** either.

For any TM $\langle M \rangle$, let $f(\langle M \rangle) = \langle N \rangle$, where N is the same machine as M but modified so that it loops whenever it would reject. (This is similar to the transformation we did in the proof relating A_{TM} and *HALT*). This function f is computable.

We claim that $\langle M \rangle \in A_{ALL}$ iff $\langle N \rangle \in DECIDER$. To see this, note that $\langle N \rangle \in DECIDER$ iff N is a decider, which happens iff N halts on all inputs. N halts on all inputs iff it accepts or rejects all inputs. By construction, N never rejects, so N halts on all inputs iff N accepts all inputs. Also by construction, N accepts an arbitrary input w iff M accepts w. Consequently, N accepts all inputs iff M accepts all inputs. This in turn happens iff $\mathscr{L}(M) = \Sigma^*$, which happens iff $\langle M \rangle \in A_{ALL}$. Thus $\langle M \rangle \in A_{ALL}$ iff $\langle N \rangle \in DECIDER$, so $A_{ALL} \leq_M DECIDER$, as required. ■

The most common mistake we saw on this problem was constructing the wrong TM as part of the reduction. Many solutions said that the reduction should be $f(\langle M \rangle) = \langle M \rangle$. While it is true that if $\mathscr{L}(M) = \Sigma^*$, then M is a decider, it is *not* necessarily the case that if M is a decider, then $\mathscr{L}(M) = \Sigma^*$. Some other solutions used the amplifier machine, which might be a decider even if M rejects w (and also is troublesome because there's no w to build into the amplifier!)

Problem Five: P and NP Languages (30 Points Total)

(i) Non-NPC Languages (15 Points)

There are exactly two languages in **NP** that we currently know are not **NP**-complete: Ø and Σ*.

Prove that Σ* is not **NP**-complete.

Proof: By contradiction; assume Σ* is **NP**-complete. Then there must be a polynomial-time mapping reduction from any **NP** language to Σ*. In particular, this means that there should be a polynomial-time reduction from Ø to Σ*. Thus there is a function f such that $w \in \emptyset$ iff $f(w) \in \Sigma^*$. Since all strings w satisfy $w \in \Sigma^*$, the right-hand side of this statement is true, and since all strings w satisfy $w \notin \emptyset$, the left-hand side of this statement is false. This is impossible. We have reached a contradiction, so our assumption must have been wrong, so Σ* is not **NP**-complete. ■

Common mistakes included confusing the language Σ* (all strings) with A_{ALL} (all TMs that accept every string), and thus concluding that Σ* is not in **NP**; correctly arguing that if Σ* is **NP**-complete, then **P** = **NP**, which is true but is not a formal proof that Σ* is definitely not **NP**-complete; claiming that Σ* is not in **NP** because it is impossible to accept all possible strings in polynomial time (which is false; there's a one-state DFA for Σ*); or arguing that it is not possible to reduce Ø to Σ* by claiming that $w \in \emptyset$ iff $f(w) = \Sigma^*$ is impossible because there would be no w to apply the function f to (remember that f is applied to all strings; we just need the earlier statement to be true.)

(ii) Resolving P $\stackrel{?}{=}$ NP **(15 Points)**

Suppose that we can prove the following statement:

For every pair of **NP** languages A and B (where neither A nor B is Ø or Σ^*), we have $A \leq_P B$.

Under this assumption, decide which of the following is true, then prove your choice is correct.

- **P** is necessarily equal to **NP**.
- **P** is necessarily not equal to **NP**.
- P may or may not be equal to NP.

Theorem: Under this assumption, **P** = **NP**.

Proof: Take any **P** language A (other than Ø or Σ^*). Under our assumption, SAT $\leq_P A$. This means that $SAT \in$ **P**. Therefore, there is an **NP**-complete problem (namely, SAT) that is in **P**, so **P** = **NP**. ■

Common mistakes included showing that all **P** languages must be **NP**-complete by showing that any arbitrary language in **NP** would reduce to any arbitrary **P** language (which is mostly correct, but misses the fact that Ø and Σ^* aren't necessarily known to reduce to that **P** language under the above assumption) and other similar mistakes that involved missing the fact that Ø and Σ^* are specifically excluded from the assumption.