Additional Practice CS103 Final 2 Solutions

Here are the solutions to the second 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 (15 points total)

Prove, by induction on n, that for any $n \in \mathbb{N}$ with $n \geq 2$, that

$$\log_2 3 \cdot \log_3 4 \cdot \log_4 5 \cdot \ldots \cdot \log_n (n + 1) = \log_2 (n + 1)$$

You might want to use the change of base formula for logarithms: for any $c \in \mathbb{R}$ with $c > 1$,

$$\log_a b = \frac{\log_c b}{\log_c a}$$

Proof: By induction on n. Let $P(n)$ be "$\log_2 3 \cdot \log_3 4 \cdot \ldots \cdot \log_n (n + 1) = \log_2 (n + 1)$". We prove that $P(n)$ holds for all $n \in \mathbb{N}$ with $n \geq 2$ by induction.

As a base case, we prove $P(2)$, that $\log_2 3 = \log_2 3$. Since this statement is true, $P(2)$ holds.

For the inductive step, assume that for some $n \in \mathbb{N}$ with $n \geq 2$ that $P(n)$ holds. We will prove $P(n + 1)$: that $\log_2 3 \cdot \log_3 4 \cdot \ldots \cdot \log_n (n + 1) \cdot \log_{n+1} (n + 2) = \log_2 (n + 2)$. To see this, note that by our inductive hypothesis, that

$$\log_2 3 \cdot \log_3 4 \cdot \ldots \cdot \log_n (n + 1) \cdot \log_{n+1} (n + 2) = \log_2 (n + 1) \log_{n+1} (n + 2)$$

By the change of base formula,

$$\log_2(n+1)\log_{n+1}(n+2) = \log_2(n+1)\frac{\log_2(n+2)}{\log_2(n+1)} = \log_2(n+2)$$

Thus $P(n + 1)$ holds, completing the induction. ■

Common Mistakes: The most common mistakes we saw with this problem were trying to prove a result about the *sum* of these logarithms, rather than their product, or not proving the result inductively as required. Otherwise, most people did very well on this problem.

Problem Two: Regular Languages (40 Points Total)

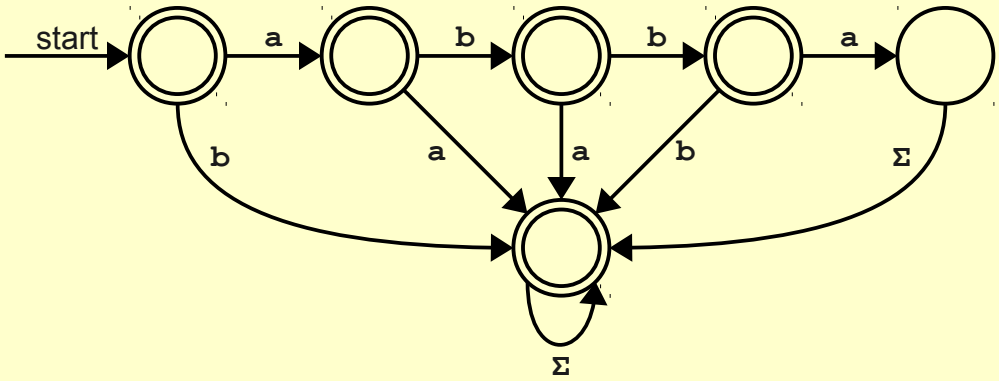
Suppose that you really, *really* don't like the string **abba** and want to build a language of everything except that string. Let $\Sigma = \{\mathbf{a}, \mathbf{b}\}$ and consider the language $NOT_{\mathbf{abba}}$ defined as follows:

$$NOT_{\mathbf{abba}} = \{ w \in \Sigma^* \mid w \neq \mathbf{abba} \}$$

For example, $\varepsilon \in NOT_{\mathbf{abba}}$ and $\mathbf{abbabb} \in NOT_{\mathbf{abba}}$, but (unsurprisingly) $\mathbf{abba} \notin NOT_{\mathbf{abba}}$.

(i) Finite Automata (10 Points)

Design a DFA for the language $NOT_{\mathbf{abba}}$.



This machine is formed by starting with a machine that accepts *only* the string **abba**, then flipping all the accept and reject states so that it accepts everything *except* **abba**.

(ii) Regular Expressions (10 Points)

Write a regular expression for $NOT_{\mathbf{abba}}$.

One option is

ε | a | ab | abb | (b | aa | aba | abbb | abbaΣ)Σ*

The first four terms of this regular expression match all prefixes of **abba**. The remaining terms of the regular expression match strings whose prefixes aren't prefixes of **abba**, or which consist of exactly **abba** with at least one character tacked on to the end.

You can interpret this regular expression as listing all paths to an accepting state in the DFA.

(iii) Nonregular Languages **(20 Points)**

Let $L = \{\texttt{0}^n\texttt{1}^n \mid n \in \mathbb{N}\}$. Prove that no infinite subset of L is a regular language. This result shows that not only is it impossible to write a regular expression for L, but it is also impossible to write a regular expression that matches infinitely many strings from L without also matching at least one string not in L.

Proof: Let L' be any infinite subset of L. Consider the set $S = \{ \texttt{0}^n \mid \texttt{0}^n\texttt{1}^n \in L' \}$. This set is infinite, since there are infinitely many strings in L'. We claim that any pair of strings in S are distinguishable relative to L'. To see this, consider any two strings $\texttt{0}^n, \texttt{0}^m \in S$ where $\texttt{0}^n \neq \texttt{0}^m$. Then $\texttt{0}^n\texttt{1}^n \in L'$ (because, by construction, $\texttt{0}^n \in S$ iff $\texttt{0}^n\texttt{1}^n \in L'$) but $\texttt{0}^m\texttt{1}^n \notin L'$, since $\texttt{0}^m\texttt{1}^n \notin L$ and we know $L' \subseteq L$. Since our choice of $\texttt{0}^n$ and $\texttt{0}^m$ was arbitrary, this means that any two strings in S are distinguishable relative to L'. Therefore, by the Myhill-Nerode theorem, L' is not regular. Thus no infinite subset of L is regular. ■

Common Mistakes: Almost everyone came up with a correct DFA for this problem. The regular expression was tricky to come up with (when writing the exam, my initial regular expression was incorrect!). We saw many regular expressions that generated some but not all of strings that were not the string **abba**, and others that accidentally generated **abba** due to mistakes with the Kleene star operator.

Part (iii) of this problem was particularly difficult. The most common mistakes we saw were arguments of the form "since L is not regular, no infinite subset of L is regular." This isn't true for all nonregular languages; take the language $\{ w \in \{\texttt{0}, \texttt{1}\}^* \mid w$ has the same number of **0**s and **1**s$\}$, which isn't regular (do you see why?) but has the infinite regular language **(01)*** as an infinite subset.

Another common mistake in part (iii) was to not choose the set S correctly. Many solutions chose $S = \{ \texttt{0}^n \mid n \in \mathbb{N} \}$ as an infinite set of distinguishable strings, arguing that $\texttt{0}^n$ is distinguishable from $\texttt{0}^m$ by appending $\texttt{1}^n$ to each. The reason for this is that $\texttt{0}^n\texttt{1}^n$ is not necessarily in L', so it might be the case that neither $\texttt{0}^n\texttt{1}^n$ nor $\texttt{0}^m\texttt{1}^n$ are in L'.

Problem Three: Context-Free Languages (25 points)

(i) Context-Free Grammars (15 Points)

Consider the following language L over the alphabet $\Sigma = \{\mathbf{a}, \mathbf{b}\}$:

$L = \{ w \in \Sigma^* \mid |w| \equiv_4 0$, and the first quarter of the characters in w contains at least one **b** $\}$

For example, **baaa** $\in L$, **abbbbbba** $\in L$, **bbbaaabbbaaa** $\in L$, **ababbbbbbbbb** $\in L$, but **abbb** $\notin L$, **aabbbbbaa** $\notin L$, $\varepsilon \notin L$, **bbb** $\notin$ L, and **aaabbbbbbbbb** $\notin L$. Here, the first quarter of the characters in each string has been underlined.

Write a context-free grammar for L.

One option is

$$S \rightarrow \mathbf{a}SXXX \mid \mathbf{b}TXXX$$

$$T \rightarrow XTXXX \mid \varepsilon$$

$$X \rightarrow \mathbf{a} \mid \mathbf{b}$$

The idea is to build the string from the inside out by placing three characters on the right and one on the left. This ensures that the string has a length that is a multiple of four. We place arbitrary characters on the right and **a**s on the left until we construct the necessary **b**. At that point, we can place arbitrary characters on either side of the string.

Another option is

$$S \rightarrow TXXX$$

$$T \rightarrow TXXXX \mid XTXXX \mid \mathbf{b}$$

$$X \rightarrow \mathbf{a} \mid \mathbf{b}$$

The idea behind this grammar is to start off by dropping down four characters. The T symbol represents the position in which the **b** in the first quarter will ultimately be placed. The productions for T then say that it can either remain where it is as the string grows longer (production one), or move one step to the right as the string grows longer (production two), or expand to **b** immediately.

Common Mistakes: This problem was tricky. Coming up with a working grammar requires you to understand the way in which a string in L could be constructed.

By far the most common mistake was not being able to generate all strings in L. Most solutions correctly generated only strings in L, but not all of the strings in L. For example, many grammars would require the first character of the string to be a **b**, or would require the last character of the first quarter of the string to be a **b**.

Problem Four: R and RE Languages (60 Points)

(i) Properties of Reductions (15 Points)

A language L over an alphabet Σ is called *nontrivial* iff $L \neq \emptyset$ and $L \neq \Sigma^*$.

Prove or disprove: All nontrivial **RE** languages are mapping reducible to one another.
Prove or disprove: All nontrivial co-**RE** languages are mapping reducible to one another.

Disprove claim 1: Not all nontrivial **RE** languages are mapping reducible to one another. Take A_{TM} and `0*1*`. Then if $A_{TM} \leq_M$ `0*1*`, we would have $A_{TM} \in$ **R** because `0*1*` $\in$ **R**. But this is impossible, since $A_{TM} \notin$ **R**. Thus A_{TM} is not mapping reducible to `0*1*`.

Disprove claim 2: Not all nontrivial co-**RE** languages are mapping reducible to one another. Take L_D and `0*1*`. Then if $L_D \leq_M$ `0*1*`, we would have $L_D \in$ **R** because `0*1*` $\in$ **R**. But this is impossible, since $L_D \notin$ **R**. Thus L_D is not mapping reducible to `0*1*`.

(ii) co-RE Languages **(20 Points)**

Consider the following language, which consists of all Turing machines that reject at least one string:

$$L_{R1} = \{ \langle M \rangle \mid M \text{ rejects at least one string} \}$$

Prove that $L_{R1} \in \mathbf{RE}$.

Proof: We will give an NTM N where $\mathscr{L}(N) = L_{R1}$; this proves that $L_{R1} \in \mathbf{RE}$, as required.

Consider the following NTM N:

N = "On input $\langle M \rangle$:

Nondeterministically guess a string $w \in \Sigma^*$.

Run M on w.

If M rejects w, accept.

If M accepts w, reject."

To prove that $\mathscr{L}(N) = L_{R1}$, we prove that there is some choice N can make such that N accepts $\langle M \rangle$ iff $\langle M \rangle \in L_{R1}$. To see this, note that there is some choice N can make such that N accepts $\langle M \rangle$ iff there is some choice of w such that M rejects w. Moreover, there is some choice of w such that M rejects w iff $\langle M \rangle \in L_{R1}$. Thus $\mathscr{L}(N) = L_{R1}$, as required. ■

(iii) Unsolvable Problems **(20 Points)**

Prove that $L_{R1} \notin$ co-**RE**. As a reminder, L_{R1} is defined as follows:

$$L_{R1} = \{ \langle M \rangle \mid M \text{ rejects at least one string} \}$$

Proof: We will prove that $A_{TM} \leq_M L_{R1}$. Since $A_{TM} \notin$ co-**RE**, this proves that $L_{R1} \notin$ co-**RE** either.

We exhibit a mapping reduction from A_{TM} to L_{R1}. For any TM/string pair $\langle M, w \rangle$, consider the machine M' defined as follows:

M' = "On input x:

Ignore x.

Run M on w.

If M accepts w, reject.

If M rejects w, accept."

Let $f(\langle M, w \rangle) = \langle M' \rangle$. This function is computable, since we can make the amplifier machine Amp(M, w) and then switch its accepting and rejecting states. We further claim that $\langle M, w \rangle \in A_{TM}$ iff $\langle M' \rangle \in L_{R1}$. To see this, note that $\langle M' \rangle \in L_{R1}$ iff M' rejects at least one string. We claim that M' rejects at least one string iff M accepts w. To see this, note that if M accepts w, then M' rejects all inputs, so it rejects at least one input. Otherwise, if M does not accept w, then either M rejects w, in which case M' accepts all inputs, or M loops on w, in which case M' loops on all inputs. In either case, M' does not reject any strings. Finally, note that by definition of A_{TM}, we see that M accepts w iff $\langle M, w \rangle \in A_{TM}$. Thus $\langle M, w \rangle \in A_{TM}$ iff $\langle M' \rangle \in L_{R1}$, so f is a mapping reduction from A_{TM} to L_{R1}, so $A_{TM} \leq_M L_{R1}$. Since $A_{TM} \notin$ co-**RE**, $L_{R1} \notin$ co-**RE**, as required. ■

Common Mistakes: For part (i), by far the most common mistake was trying to find an explicit way to mapping reduce any pair of **RE** or co-**RE** languages to one another. This is impossible, since languages in **RE** – **R** or co-**RE** – **R** can't be reduced to languages in **R**.

For part (ii), many solutions attempted reductions that would not prove that $L_{R1} \in$ **RE**, such as trying to show that $A_{TM} \leq_M L_{R1}$. Other incorrect solutions attempted reductions that would prove that $L_{R1} \in$ **RE**, but made small mistakes in setting up the reductions.

For part (iii), the most common mistakes were (as in part (ii)) setting up reductions that could not prove that $L_{R1} \notin$ co-**RE**, such as proving $L_{R1} \leq_M A_{TM}$ or $L_D \leq_M A_{TM}$.

Problem Five: P and NP Languages (40 Points Total)

(i) Mutual Reducibility (15 Points)

Prove that for any language L, L is **NP**-complete iff $L \leq_P$ 3SAT and 3SAT $\leq_P L$.

Proof: Consider any arbitrary language L. We prove both directions of implication.

First, we prove that if L is **NP**-complete, then $L \leq_P$ 3SAT and 3SAT $\leq_P L$. To see that $L \leq_P$ 3SAT, note that since L is **NP**-complete, $L \in$ **NP**. Since 3SAT is **NP**-complete, any **NP** language is polynomial-time reducible to 3SAT, so in particular $L \leq_P$ 3SAT. To see that 3SAT $\leq_P L$, note that since 3SAT $\in$ **NP** and L is **NP**-complete, there must be a polynomial-time reduction from 3SAT to L. Thus 3SAT $\leq_P L$, as required.

Next, we prove that if $L \leq_P$ 3SAT and 3SAT $\leq_P L$, then L is **NP**-complete. To see this, note that $L \leq_P$ 3SAT, then $L \in$ **NP**. Next, note that if 3SAT $\leq_P L$, then L is **NP**-hard. Thus L is **NP**-hard and is in **NP**, so L is **NP**-complete. ■

(iii) Resolving P $\stackrel{?}{=}$ NP **(25 Points)**

Suppose that V is a polynomial-time verifier for an **NP**-complete language L. That is,

$$w \in L \quad \text{iff} \quad \exists x \in \Sigma^*.\ V \text{ accepts } \langle w, x \rangle$$

and

$$V \text{ runs in time polynomial in } |w|$$

Now, suppose that V is a "superverifier" with the property that for any string $w \in L$, V accepts $\langle w, x \rangle$ for almost all choices of x. Specifically, for any $w \in L$, there are at most five strings x for which V rejects $\langle w, x \rangle$. Under these assumptions, prove that **P** = **NP**.

Proof: We will show that $L \in$ **P**; since L is **NP**-complete, this proves **P** = **NP**.

To show that $L \in$ **P**, we will construct a polynomial-time decider for L. To do so, consider the following TM M:

> M = "On input w:
>
> Run V on $\langle w, \varepsilon \rangle$, $\langle w, \texttt{0} \rangle$, $\langle w, \texttt{1} \rangle$, $\langle w, \texttt{00} \rangle$, $\langle w, \texttt{01} \rangle$, and $\langle w, \texttt{10} \rangle$, one after another.
>
> If V accepts any of these strings, accept.
>
> If V rejects all of these strings, reject."

First, we claim that $\mathscr{L}(M) = L$. To see this, note that M accepts w iff V accepts one of six different strings of the form $\langle w, x \rangle$. We claim that this happens iff $w \in L$. First, note that if $w \notin L$, then since V is a verifier for L, it will reject all strings of the form $\langle w, x \rangle$. In particular, this means that it will reject each of the six strings that M runs it on. Therefore, M will reject w, so M does not accept w. On the other hand, if $w \in L$, then by definition V can reject $\langle w, x \rangle$ for at most five choices of x. Consequently, since M runs V on six strings of the form $\langle w, x \rangle$ and $w \in L$, we know that V must accept at least one of these strings. Therefore, M accepts w. Consequently, we have that M accepts w iff $w \in L$, so $\mathscr{L}(M) = L$.

Next, we claim that M is a decider. To see this, note that M runs the machine V on six different strings, one after the other, then accepts or rejects. Since V is a decider, V always halts on all inputs, so every time V is run it will halt. Therefore, M must halt on all inputs.

Finally, we claim that M runs in polynomial time. To see this, note that M runs V on six different strings of the form $\langle w, x \rangle$. Since V runs in time polynomial in $|w|$, this means that running the machine takes polynomial time each time. The machine does only polynomial work to construct the six strings of the form $\langle w, x \rangle$, and does at most polynomial work afterwards to determine whether to accept or reject. Thus the machine M runs in polynomial time.

This means that M is a polynomial-time decider for L, so $L \in$ **P**. Since $L \in$ **NPC**, this means **P** = **NP**, as required. ■

Common mistakes: The most common mistakes on part (i) of this problem was failing to prove both directions of implication, using informal justifications about "no easier than" or "no harder than" to explain why the problems were reducible to one another, or some combination of these errors.

For part (ii), the most common mistake was thinking that $\mathscr{L}(V)$ was "close" to the language L. However, this is not the case – note that $\mathscr{L}(V)$ is a language of pairs of strings in L and some certificate for L. Consequently, it's not possible to tweak V by hardcoding in the five strings that it will be wrong about, since (a) even if this were done, the resulting machine would not have language L, and (b) those strings could vary based on which w was provided as input.

Another common mistake was only running V on *five* strings rather than the six necessary to determine whether a string w belonged to L.