Practice Final Exam 1

We strongly recommend that you work through this exam under realistic conditions rather than just flipping through the problems and seeing what they look like. Setting aside three hours in a quiet space with your notes and making a good honest effort to solve all the problems is one of the single best things you can do to prepare for this exam. It will give you practice working under time pressure and give you an honest sense of where you stand and what you need to get some more practice with.

This practice final exam is a (slightly modified) version of the final exam we gave out the last time we taught CS103 (Fall 2016). The exam policies are the same for the midterms – closed-book, closed-computer, limited note (one double-sided sheet of 8.5" × 11" paper decorated however you'd like).

You have three hours to complete this exam. There are 50 total points.

Question	Points	Graders
(1) Set Theory and Logic	/ 8	
(2) Graphs and Natural Numbers	/ 6	
(3) Binary Relations and Induction	/ 10	
(4) Regular and Context-Free Languages	/ 13	
(5) **R** and **RE** Languages	/ 10	
(6) **P** and **NP** Languages	/ 3	
	/ 50	

Problem One: Set Theory and Logic (8 Points)

An ***independence system*** over a set A is a <u>*nonempty*</u> set $I \subseteq \wp(A)$ with the following property:

$$\forall S \in I.\ \wp(S) \subseteq I.$$

This question explores some properties of independence systems.

i. **(3 Points)** Prove that if I is an independence system over a set A, then $\varnothing \in I$.

As a refresher from part (i) of this problem, an ***independence system*** over a set A is a nonempty set $I \subseteq \wp(A)$ with the following property:

$$\forall S \in I.\ \wp(S) \subseteq I.$$

ii. **(5 Points)** Let I_1 and I_2 be independence systems over the same set A. Prove that $I_1 \cap I_2$ is also an independence system over A.

For simplicity, you can use the fact that $I_1 \cap I_2 \subseteq \wp(A)$ without proof. However, since we haven't talked much about properties of set intersection in this course, if you want to use any other facts about set intersection, you'll need to prove them first.

Problem Two: Graphs and Natural Numbers (6 Points)

On Problem Set Four, you explored bipartite graphs. As a refresher, a graph $G = (V, E)$ is bipartite if there are sets V_1 and V_2 where all three of the following are true:

- V_1 and V_2 have no nodes in common (that is, $V_1 \cap V_2 = \varnothing$).
- Every node $v \in V$ belongs to at least one V_1 and V_2.
- Every edge in E has one endpoint in V_1 and the other endpoint in V_2.

Now, consider the graph $G_{\mathbb{N}}$ defined as follows: the nodes in $G_{\mathbb{N}}$ are the natural numbers, and there's an edge between a pair of nodes u and v if and only if $u + v$ is odd. This graph contains infinitely many nodes, which is unusual but nothing to worry about.

Prove that $G_{\mathbb{N}}$ is bipartite.

Problem Three: Induction and Binary Relations (10 Points)

Given a binary relation R over a set A and a natural number $n \geq 1$, we can define a new binary relation over A called the ***nth power of R***, denoted $\boldsymbol{R^n}$. This relation is defined inductively as follows:

$$xR^1y \quad \text{if} \quad xRy$$
$$xR^{n+1}y \quad \text{if} \quad \exists z \in A.\ (xRz \wedge zR^ny).$$

(Note that R^n is only defined for $n \geq 1$, and remember that "if" here means "is defined as.")

i. **(7 Points)** Let R be an arbitrary binary relation over a set A. Your task is to prove the following statement:

For any natural numbers $m, n \geq 1$,
and for any $a, b, c \in A$,
if aR^nb and bR^mc, then $aR^{n+m}c$.

To do so, we'd like you to use induction. Specifically, use induction to prove that the statement $P(n)$ defined below is true for all natural numbers $n \geq 1$:

$P(n)$ is the statement "for any natural number $m \geq 1$,
and for any $a, b, c \in A$,
if aR^nb and bR^mc, then $aR^{n+m}c$."

(Extra space for your answer to Problem Three, part (i), if you need it.)

The ***transitive closure*** of a binary relation R over a set A, denoted $\boldsymbol{R^+}$, is a binary relation over A defined as follows:

$$xR^+y \quad \text{if} \quad \exists n \in \mathbb{N}.\ (n \geq 1 \wedge xR^n y).$$

ii. **(3 Points)** Let R be an arbitrary binary relation over a set A. Using your result from part (i) of this problem, prove that R^+ is transitive. (You can use the result from part (i) even if you weren't able to prove it.)

Problem Four: Regular and Context-Free Languages (13 Points)

Consider the following language:

$L_1 = \{ w \in \{\texttt{a}, \texttt{b}, \texttt{c}\}^* \mid$ the last character of w appears nowhere else in w, and $|w| \geq 1 \}$.

This is a variant on one of the languages you built an NFA for in Problem Set Six. Here are some sample strings in L_1:

- `a`
- `b`
- `c`
- `aaaaac`
- `aabbaabbc`
- `ccbbccbba`
- `bac`
- `cbba`

Since this language is regular, it's possible to build a regular expression for it.

i. **(2 Points)** Write a regular expression for L_1. *(Hint: You probably don't have time to work through the state elimination algorithm on this exam. Try designing the regular expression from scratch.)*

In many programming languages (C, C++, Python, Java, JavaScript, etc.), a ***string literal*** is a piece of text enclosed in double quotes, such as `"Hi everybody!"` or `"Good luck on the exam!"`. Sometimes, you'll want to define a string that contains a double-quote character. In these languages, to do that, you escape the double-quote by preceding it with a backslash, like this:

```
"Quoth the raven \"Nevermore.\""
```

This lets the compiler distinguish between the double-quotes inside a string and the double-quotes delimiting a string.

As a consequence of this rule, any time you want to write a backslash character, you need to escape it as well by preceding it with a second backslash. For example, you might have a string like this:

```
"The notation \"A \\ B\" denotes the difference of the sets A and B."
```

Let $\Sigma = \{$ z, ", \ $\}$ and consider this language L_2:

$$L_2 = \{\ w \in \Sigma^* \mid w \text{ is a legal string literal } \}.$$

Here are some sample strings in L2:

- `""`
- `"z"`
- `"\""`
- `"\\"`
- `"zz\"zz\\zz\"z"`
- `"\\\""`
- `"\"zz\""`
- `"\"\""`

Here are some sample strings not in L2:

- `"z` *(this string isn't closed)*
- `"""` *(the quote in the middle needs to be escaped)*
- `"\"` *(this string is unterminated – that final double quote is escaped)*
- `\"zz"` *(the string doesn't begin with a double quote)*
- `"\z"` *(you cannot escape the letter z with a slash)*
- `"\\\z"` *(you cannot escape the letter z with a slash)*

This language happens to be regular, which is useful because many compilers use some form of finite automaton to find strings in source code.

ii. **(3 Points)** Design an NFA for L_2.

According to old-school Twitter rules, all tweets need to be 140 characters or less. Let Σ be the alphabet of characters that can legally appear in a tweet and consider the following language:

$$L_3 = \{ w \in \Sigma^* \mid |w| \leq 140 \}.$$

This is the language of all legal tweets. The good news is that this language is regular. The bad news is that building a DFA for it would be a frustrating endeavor.

iii. **(4 Points)** Prove that any DFA for L_3 must have at least 142 states. To do so, use the result you proved on Problem Set Seven that says that if you can find a set of 142 strings that are pairwise distinguishable relative to L_3, then any DFA for L_3 must have at least 142 states.

Let $\Sigma = \{\texttt{a}, \texttt{b}\}$ and consider the following language:

$L_4 = \{\ w \in \Sigma^* \mid |w| \equiv_4 0$, and the first quarter of the characters in w contains at least one $b\ \}$.

For example, baaa $\in L_4$, bbbb $\in L_4$, abbbbbba $\in L_4$, bbbaaabbbaaa $\in L_4$, abababbbbbbbbb $\in L_4$, but abbb $\notin L_4$, $\varepsilon \notin L_4$, b $\notin L_4$, aabbbbaa $\notin L_4$, and aaabbbbbbbbb $\notin L_4$. (For simplicity, I've underlined the first quarter of the characters in each string).

iv. **(4 Points)** Write a CFG for L_4.

Problem Five: R and RE Languages (10 Points)

Stanford's email system often tags messages with attachments as possible viruses by changing the subject to say something like [POSSIBLE VIRUS: ###]. You might wonder why the email system says something is a "possible" virus rather than just intercepting emails that really do contain viruses and blocking them from getting to their recipients. This question explores why.

Let's imagine that there's some method that, if called, will do something nefarious to your computer. Imagine it's this method:

```
private void doSomethingNefarious()
```

Your job is to prove that it's impossible to write a method

```
private boolean canDoSomethingNefarious(String program)
```

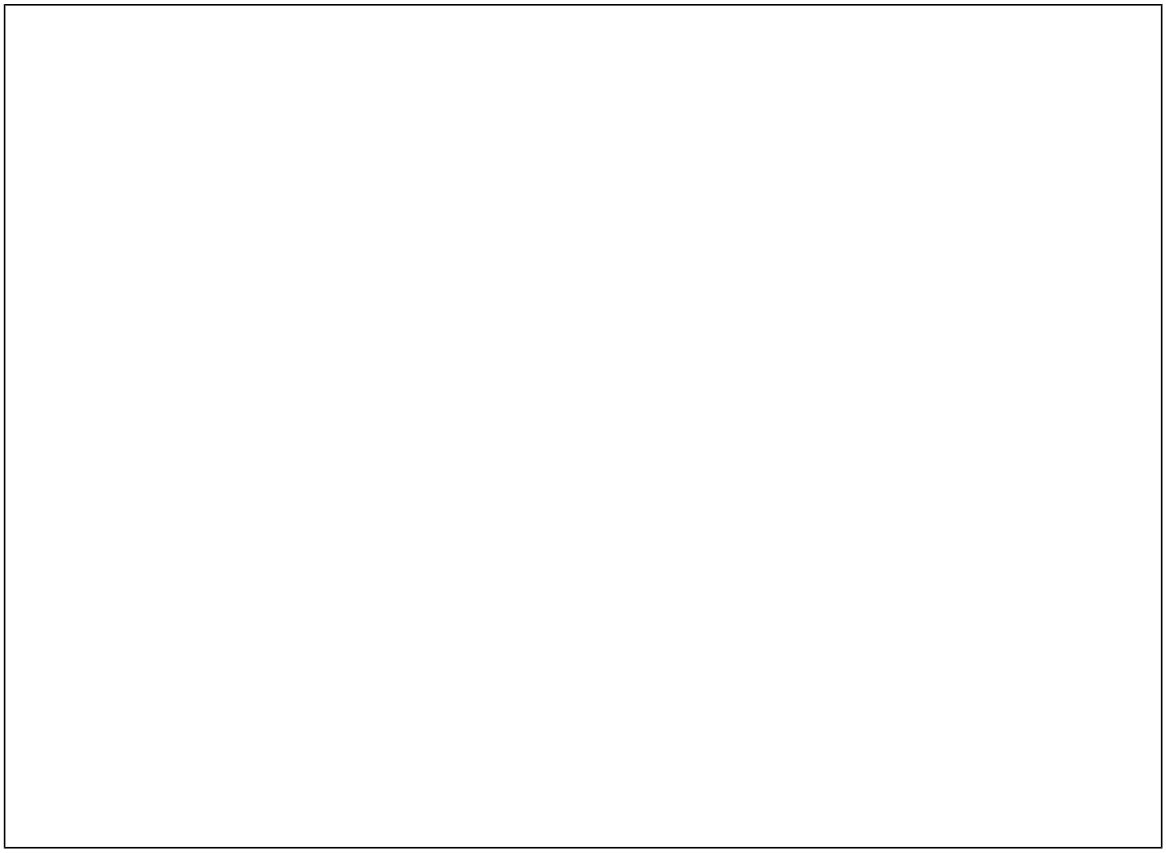
that takes as input the source code of a program, then returns true if the program under some circumstance can call the `doSomethingNefarious` method and returns false otherwise. (This partially explains why you get the "possible" virus warning over email – there's no general way to test whether a program can do nefarious things!)

i. **(4 Points)** In the interests of time, we don't want you to write out a full formal proof of this result. Instead, do the following:

- In the space below, write a self-referential program *P* that uses the `canDoSomethingNefarious` method such that *P* does something nefarious if and only if it doesn't do something nefarious. You can assume you have access to a method `mySource()` that returns the source code of your program.

- Briefly explain why *P* does something nefarious if and only if it doesn't do something nefarious, addressing each direction of the implication.

iii. **(6 Points)** Below is a Venn diagram showing the overlap of different classes of languages we've studied so far. We have also provided you a list of numbered languages. For each of those languages, draw where in the Venn diagram that language belongs. As an example, we've indicated where Language 1 and Language 2 should go. No proofs or justifications are necessary, and there is no penalty for an incorrect guess.



1. Σ^*
2. L_D
3. $\{ \texttt{1}^m\texttt{+1}^n\texttt{=1}^{m+n} \mid m, n \in \mathbb{N}$ and m and n are even $\}$
4. $\{ \texttt{1}^m\texttt{+1}^n\texttt{=1}^{m+n} \mid m, n \in \mathbb{N}$ and $m \leq 10^{137} \}$
5. $\{ \texttt{1}^m\texttt{+1}^n\texttt{=1}^{m+n} \mid m, n \in \mathbb{N}$ and $m + n \leq 10^{137} \}$
6. $\{ \langle M, w \rangle \mid M$ is a TM, w is a string, and M accepts w within $\lvert w \rvert^{137}$ steps $\}$
7. $\{ \langle M \rangle \mid M$ is a TM and M halts on infinitely many inputs $\}$
8. $\{ \langle M, w \rangle \mid M$ is a TM, w is a string, M accepts at least one substring of $w \}$

Problem Six: P and NP Languages (3 Points)

We briefly covered the **P** and **NP** languages in our last week of class. Here's a quick series of true/false questions about them. Each correct answer is worth one point, and there is no penalty for an incorrect guess. You do not need to justify your answers.

i. **NP** stands for "not polynomial time" and is the class of decision problems that cannot be solved in polynomial time.

☐ True ☐ False

ii. All **NP**-hard problems are in **NP**.

☐ True ☐ False

iii. If the halting problem is decidable, then **P** ≠ **NP**.

☐ True ☐ False

We have one final question for you: do *you* think **P** = **NP**? Let us know in the space below. There are no right or wrong answers to this question – we're honestly curious to hear your opinion!

☐ I think **P** = **NP** ☐ I think **P** ≠ **NP**