Practice Final Exam I

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 (Spring 2017). 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 55 total points.

Question	Points	Graders
(1) Binary Relations and Functions	/ 10	
(2) Graphs and the Pigeonhole Principle	/ 10	
(3) Induction and Set Theory	/ 11	
(4) Regular and Context-Free Languages	/ 11	
(5) **R** and **RE** Languages	/ 10	
(6) **P** and **NP** Languages	/ 3	
	/ 55	

Problem One: Binary Relations and Functions (10 Points)

*(We recommend spending about **40** minutes on this problem.)*

Over the course of the quarter, you've gotten a lot of practice working with different classes of binary relations and their properties. This problem explores two subtypes of strict orders and how they relate to one another.

Let's begin with a new definition. A binary relation R over a set A is called ***trichotomous*** if for any x, $y \in A$, exactly one of the following statements is true:

$$xRy \qquad x=y \qquad yRx$$

A binary relation R is called a ***strict <u>total</u> order*** if R is a strict order and R is trichotomous.

As a refresher from Problem Set Three, if R be a strict order over some set A, then the ***incomparability relation*** of R, denoted $\sim_R$, is a binary relation over A defined as follows:

$$x \sim_R y \quad \text{if} \quad x \not R y \wedge y \not R x$$

(As a reminder, the "if" in the above context means "is defined to mean." Also, note that those R's on the right-hand side of the definition have slashes through them.)

A binary relation R over a set A is called a ***strict <u>weak</u> order*** if R is a strict order and R's incomparability relation $\sim_R$ is transitive.

Let A and B be arbitrary sets. Let $f : A \to B$ be an arbitrary function from A to B and let R be an arbitrary strict ***<u>total</u>*** order over B. Consider the relation S over the set A defined below:

$$xSy \quad \text{if} \quad f(x) \; R \; f(y)$$

Prove that S is a strict ***<u>weak</u>*** order over A.

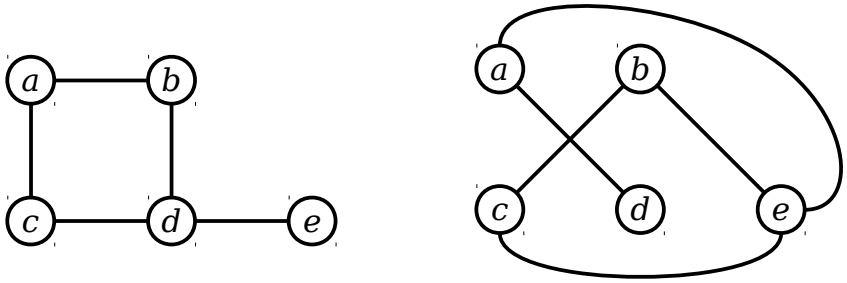
(Extra space for your answer to Problem One, if you need it)

Problem Two: Graphs and the Pigeonhole Principle (10 Points)

*(We recommend spending about **40** minutes on this problem.)*

On Problem Set Four, you explored bipartite graphs. As a refresher, a graph $G = (V, E)$ is called ***bipartite*** if its nodes can be partitioned into two sets V_1 and V_2 where every edge $e \in E$ has one endpoint in V_1 and the other in V_2. This question explores some additional properties of bipartite graphs.

Let's begin with a new definition. If $G = (V, E)$ is an undirected graph, the ***complement of G***, denoted $\boldsymbol{G^c}$, is a graph related to the original graph G. Intuitively, G^c has the same nodes as G, and its edges consist of all the edges missing from graph G. Formally speaking, G^c is the graph with the same nodes as G and with edges determined as follows: the edge $\{u, v\}$ is present in G^c if and only if $u \neq v$ and the edge $\{u, v\}$ is not present in G. As an example, here's a sample graph and its complement:



Prove that if G is a graph with at least five nodes, then at least one of G and G^c is ***not*** bipartite.

(Extra space for your answer to Problem Two, if you need it.)

Problem Three: Induction and Set Theory (11 Points)

*(We recommend spending about **40** minutes on this problem.)*

When you were first learning algebra, you probably learned a family of techniques to solve equations in which a variable x was on both sides of an equals sign. For example, you probably learned how to look at a formula like

$$x^2 = ax + b$$

and to use the quadratic formula to solve for x.

It's also possible to set up equations involving some unknown that appears on both sides of an equals sign, but where the quantities involved are *languages* rather than numbers. For example, if A and B are languages, you may want to determine what languages X satisfy the equality

$$X = AX \cup B.$$

Just as the quadratic formula is a useful tool for solving for x given a quadratic equation, in formal language theory there's a result called ***Arden's lemma*** that's useful for solving for X in an equation of the above form. Specifically, Arden's lemma says that, given the equality $X = AX \cup B$, you are guaranteed that

$$A^*B \subseteq X.$$

In this problem, we're going to ask you to prove Arden's lemma.

Let's begin with a refresher of the key terms and definitions involved. As a reminder, if L_1 and L_2 are languages over an alphabet Σ, then the ***concatenation of L_1 and L_2***, denoted L_1L_2, is the language

$$L_1L_2 = \{\ wx \mid w \in L_1 \text{ and } x \in L_2\ \}.$$

From concatenation, we can define ***language exponentiation*** of a language L inductively as follows:

$$L^0 = \{\varepsilon\} \qquad\qquad L^{n+1} = LL^n$$

You may find these formal terms helpful in the course of solving this problem.

i. **(8 Points)** Let A and B be arbitrary languages over some alphabet Σ. Prove, by induction, that if $X = AX \cup B$, then $A^nB \subseteq X$ for every $n \in \mathbb{N}$. Please use the formal definitions of concatenation, language exponentiation, union, and subset in the course of writing up your answer.

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

If you'll recall, we formally defined the ***Kleene closure*** of a language L over Σ to be the language

$$L^* = \{ w \in \Sigma^* \mid \text{there is some } n \in \mathbb{N} \text{ such that } w \in L^n \}.$$

ii. **(3 Points)** Let A and B be arbitrary languages over some alphabet Σ. Using your result from part (i) of this problem and the formal definition of L^*, prove that if $X = AX \cup B$, then $A^*B \subseteq X$.

Problem Four: Regular and Context-Free Languages (11 Points)

*(We recommend spending about **35** minutes on this problem.)*

Let $\Sigma = \{\texttt{a}, \texttt{b}\}$ and let L_1 be the language over Σ given by the regular expression `(ab U ba)*`.

i. **(3 Points)** Design a DFA for L_1.

Let $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{:}\}$ and consider the following language L_2:

$$L_2 = \{ w\texttt{:}x \mid w, x \in \{\texttt{a}, \texttt{b}\}^* \text{ and } w \text{ is a substring of } x \}$$

Here are some strings in L_2. For convenience, we've underlined where the substring appears:

- `aabb:aaaabbbb`
- `bbb:aabbbaa`
- `abab:aaabbbabab`
- `aaa:aaaaa`
- `aaa:aaa`
- `abb:abb`
- `:abab`
- `:`

Here are some sample strings not in L_2:

- `aaa:ababa`
- `aaa:bbb`
- `bbb:bb`
- `aabb:abab`
- `ab:`
- `a:aa:aaa:aaaa:aaaaa`
- `::`
- ε

This language is not regular, and in this problem we'd like you to convince us why this is.

ii. **(4 Points)** To save you time, rather than having you write out a full proof that L_2 is not regular, we'd instead like you to answer the following questions:

1. The Myhill-Nerode theorem requires you to choose an infinite set S of strings that are pairwise distinguishable relative to L_2. In the space below, write such a set S.

2. Suppose you choose two arbitrary, distinct strings x and y from the set S. What string w will you append to x and y to show that they are distinguishable relative to L_2?

3. Briefly explain why the choice of w you found above shows that x and y are distinguishable relative to L_2.

Let $\Sigma = \{\texttt{1}, \texttt{+}, \texttt{=}\}$ and consider the language L_3 defined as follows:

$$L_3 = \{\texttt{1}^m\texttt{+}\texttt{1}^n\texttt{=}\texttt{1}^{m+n} \mid m, n \in \mathbb{N} \text{ and } m + n \text{ is even } \}$$

Here are some sample strings in L_3:

- `1+1=11`
- `+11=11`
- `+=`
- `111+11111=11111111`
- `1111+11=111111`
- `11111+11111=1111111111`

Here are some sample strings not in L_3:

- `1+1=111`
- `11+1=111`
- `1111+1=11111`
- `111+1111=1111111`
- `111+111+11=11111111`
- `11+11=111+1`
- `1111=11+11`
- `111++1==1++111`

The language L_3 happens to be context-free.

iii. **(4 Points)** Write a context-free grammar for L_3.

Problem Five: R and RE Languages (10 Points)

*(We recommend spending about **20** minutes on this problem.)*

A ***buffer overflow*** is an error where a program accidentally tries to write data to an index in an array that's beyond the end of that array. For example, writing to array index 10 in an array of size 5 would be a buffer overflow, as would writing to the 137th index of an array of size 137 (assuming the array is zero-indexed). This sort of error is an extremely common source of bugs and security vulnerabilities. For example, the recent WannaCry worm, which infected over 200,000 machines in a single day, used a buffer overflow to spread.

You might be wondering why, if there's a specific program behavior that we know leads to security vulnerabilities (writing to an array index past the end of the array), we can't just write a method

```
boolean hasBufferOverflowOnInput(string program, string input);
```

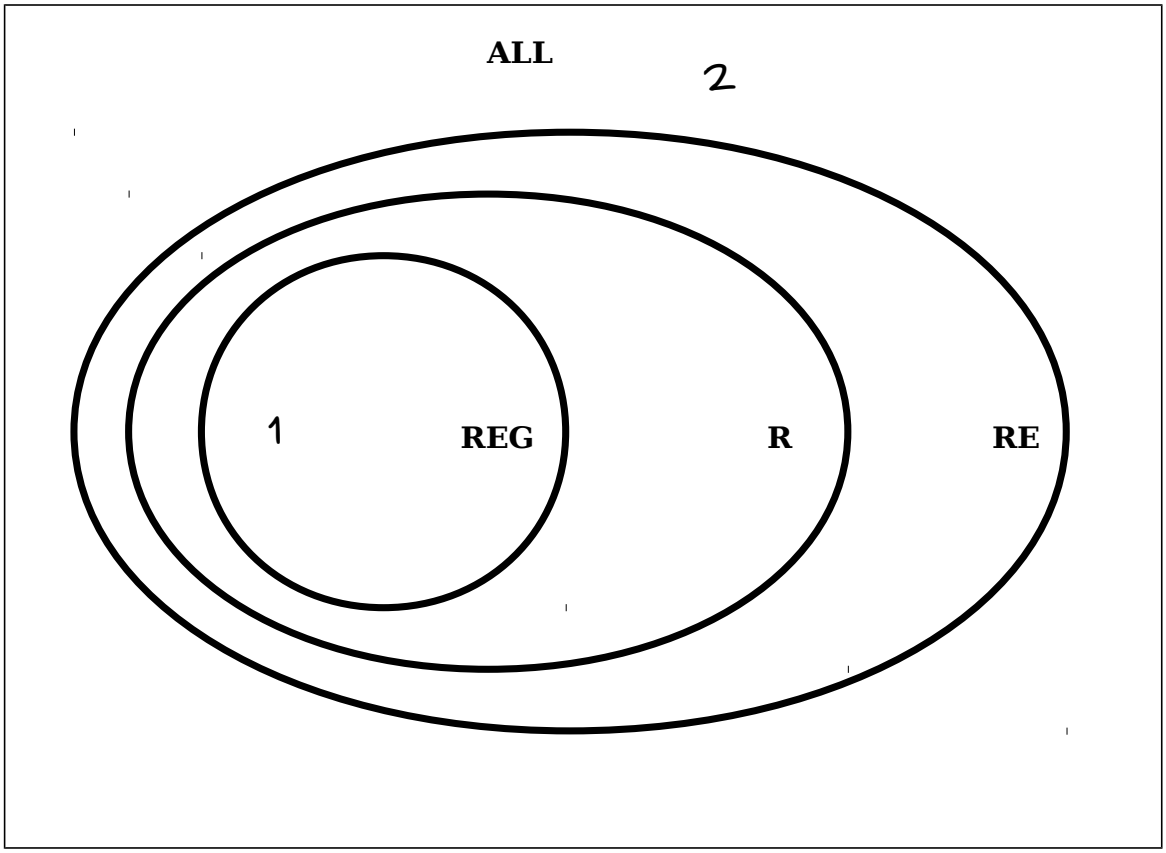
that takes as input the source code of a program and the input to that program, then returns whether, when that program is run on that input, that program ***executes*** a line of code that writes beyond the end of an array. Unfortunately, there's no way to implement the above method.

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:

   1. In the space below, write code for a self-referential program *P* that uses the `hasBufferOverflowOnInput` method such that running *P* on some input *w* causes a buffer overflow if and only if running *P* on *w* does not cause a buffer overflow. You can assume you have access to a method `mySource()` that returns the source code of your program, and you can assume that the `hasBufferOverflowOnInput` method itself doesn't trigger a buffer overflow.

   2. Briefly explain why running your program *P* on some input *w* triggers a buffer overflow if and only if running *P* on *w* does not trigger a buffer overflow.

ii. **(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. $\{ \langle M, w \rangle \mid M$ is a TM, w is a string, and M accepts w^n for ***every*** natural number $n \}$
4. $\{ \langle M, w \rangle \mid M$ is a TM, w is a string, and M accepts w^n for ***at least one*** natural number $n \}$
5. $\{ w \in \{\texttt{a}, \texttt{b}\}^* \mid w$ contains the same number of copies of the substrings `aabb` and `bbaa` $\}$
6. $\{ w \in \{\texttt{a}, \texttt{b}\}^* \mid w$ contains the same number of copies of the substrings `ab` and `ba` $\}$
7. $\{ \langle M, w \rangle \mid M$ is a TM, w is a string, M loops on w, and $|\langle M, w \rangle| \leq 10^{137} \}$
8. $\{ \langle M \rangle \mid M$ is a TM and M loops on $\langle M \rangle \}$

Problem Seven: P and NP Languages (3 Points)

*(We recommend spending about **5** minutes on this problem.)*

Here's a quick series of true/false questions about the **P** and **NP** languages. 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. If a language is in **P**, then there is no polynomial-time verifier for it.

☐ True ☐ False

ii. There is a language *L* where *L* is in **P** if and only if **P** = **NP**.

☐ True ☐ False

iii. All **NP**-complete problems are polynomial-time reducible to one another.

☐ 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**