Practice CS103 Final Exam

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 in the last offering of CS103 (Winter 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 47 total points.

Question	Points	Graders
(1) Logic and Binary Relations	/ 6	
(2) Sets and Functions	/ 5	
(3) Induction and Graphs	/ 6	
(4) Regular and Context-Free Languages	/ 12	
(5) **R** and **RE** Languages	/ 14	
(6) **P** and **NP** Languages	/ 4	
	/ 47	

Problem One: Logic and Binary Relations (6 Points)

The ***axiom of choice*** is an axiom of set theory that can be stated as follows:

> If R is an equivalence relation over A, then there is a set $S \subseteq A$ with the following property:
> for any $a \in A$, there is exactly one $b \in S$ such that aRb.

This question explores a few properties of the axiom of choice.

i. **(3 Points)** To begin with, what does the axiom of choice look like in first-order logic? Let's assume that R is an equivalence relation over the set A. Given the predicates

   $x \in y$, which states that x is an element of y;
   $Set(S)$, which states that S is a set; and
   xRy, which states that the binary relation R holds between x and y,

   along with the constant symbol A, write a statement in first-order logic that says "there is a set $S \subseteq A$ with the following property: for any $a \in A$, there is exactly one $b \in S$ such that aRb." Your translation can assume that R is an equivalence relation over the set A and you don't need to explicitly state that.

As a refresher from the previous page, the ***axiom of choice*** states that

> if R is an equivalence relation over A, then there is a set $S \subseteq A$ with the following property:
> for any $a \in A$, there is exactly one $b \in S$ such that aRb.

If R is an equivalence relation over a set A, recall that for any $a \in A$, the ***equivalence class of a***, denoted $[a]_R$, is the set

$$[a]_R = \{\ b \in A \mid aRb\ \}$$

ii. **(3 Points)** Let R be an arbitrary equivalence relation over a set A and let S be one of the sets guaranteed by the axiom of choice. Prove that every equivalence class of A contains exactly one element of S. As a reminder, to prove that there is exactly one object satisfying some property, you need to prove that there is *at least one* object with that property and *at most one* object with that property.

Problem Two: Sets and Functions (5 Points)

Below is a series of statements. For each statement, decide whether it's true or false. No justification is required, and there is no penalty for an incorrect guess.

i. **(1 Point)** For any sets A and B, if $A \in B$, then $A \subseteq \wp(B)$.

☐ True ☐ False

ii. **(1 Point)** For any sets A and B, if $A \subseteq B$, then $A \in \wp(B)$.

☐ True ☐ False

iii. **(1 Point)** For any set A, there is a set B where $A \subseteq B$.

☐ True ☐ False

iv. **(1 Point)** There is a set B where, for any set A, we have $A \subseteq B$.

☐ True ☐ False

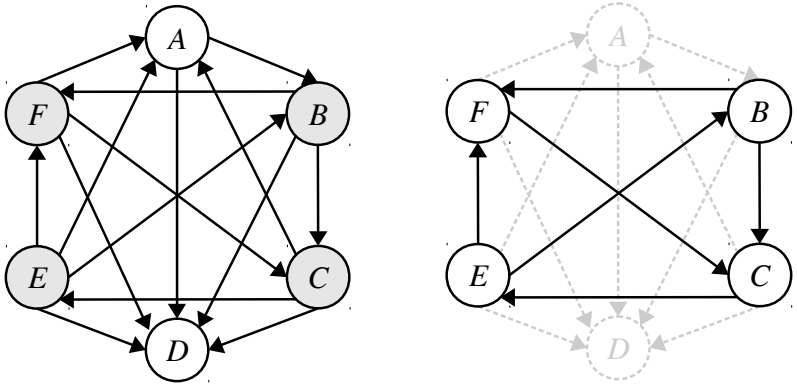
v. **(1 Point)** For any sets A and B where $|A| = |B|$, every function $f : A \to B$ is a bijection.

☐ True ☐ False

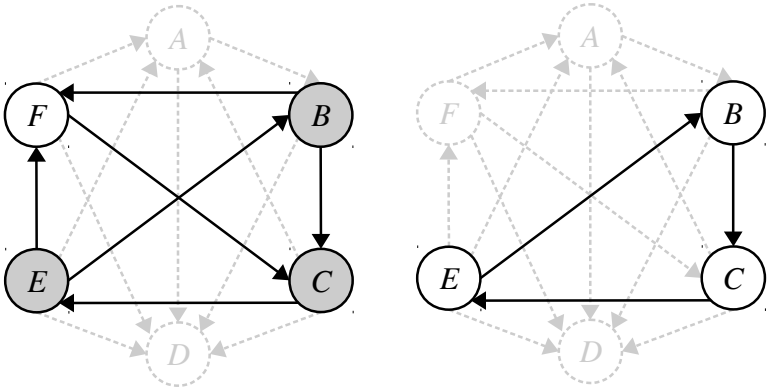
Problem Three: Induction and Graphs (6 Points)

A ***tournament*** is a directed graph where, for every distinct pair of nodes, there is exactly one edge between those nodes. We can think of a tournament as a way of representing the result of a contest in which each player plays exactly one game against each other player and there are no ties. A ***tournament winner*** is a player in a tournament who, for each other player, either won her game against that player, or won a game against a player who in turn won his game against that player (or both).

Now, let's suppose that we start with a tournament and remove from it all the players except the winners in the tournament. For example, given the tournament on the left (with the winners highlighted), we might then form the subtournament on the right:



Now, let's focus on the four-player subtournament we've discovered this way. Notice that while player *F* was a winner in the original tournament, player *F* is no longer a winner in this remaining subtournament because she didn't beat player *B* and didn't beat anyone who beat *B*. (Although in the original tournament *F* beat *A* and *A* beat *B*, player *A* is no longer present in this reduced tournament). However, the other three players (*B*, *C*, and *E*) are all still winners in this smaller subtournament. We can therefore think about once again removing from this subtournament all the players who are no longer winners. If we do, we get this subtournament:



At this point, we can see that the players *B*, *C*, and *E* are all winners of this final tournament, so if we were to delete from this subtournament all players who aren't winners, we'd end up with exactly the same subtournament we started with.

Since the players *B*, *C* and *E* were winners in the original tournament, and also winners in the tournament formed by removing all non-winners from the original tournament, and *also* winners of the tournament formed by removing all non-winners from *that* tournament, they must have done extremely well in the original tournament! Let's introduce some terminology to capture this idea.

Given a tournament *T*, we'll say that the ***winner subtournament*** of a tournament *T*, denoted $W(T)$, is the tournament formed by starting with *T* and removing from it all the players who aren't winners. A tournament *T* where $W(T) \subsetneq T$ is called a ***reducible tournament*** (all reducible tournaments must have at least one player in them who isn't a winner), and a tournament *T* where $W(T) = T$ is called an ***irreducible tournament*** (every player in an irreducible tournament is a winner.)

Finally, we'll (inductively) say that a player *p* is ***grandmaster*** of *T* if *p* is a winner in *T* and

- *T* is irreducible, or
- *T* is reducible and *p* is a grandmaster of $W(T)$.

Intuitively, a grandmaster is a player who's a winner in *T*, a winner in $W(T)$, a winner in $W(W(T))$, a winner in $W(W(W(T)))$, etc. In the example tournament *T* given on the previous page, players *B*, *C*, and *E* are all grandmasters of *T*. Player *A* isn't a grandmaster of *T* because player *A* isn't a winner in *T*, and player *F* isn't a grandmaster of *T* because, while she was a winner in *T*, she wasn't a winner in $W(T)$.

Let *T* be an arbitrary tournament with $n \geq 1$ players and let *p* be an arbitrary player in *T*. Prove by complete induction on the number of players in *T* that if *p* is the ***only*** grandmaster in *T*, then *p* is the only winner in *T*. Depending on your proof strategy, you may find it useful to use the following theorems in the course of your proof.

> ***Theorem A:*** If *p* is a player in a tournament and *p* lost a game, then at least one of the players who beat *p* is a tournament winner.
>
> ***Theorem B:*** If *p* is a player in a tournament, then *p* is the only winner if and only if *p* won all her games.

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

Problem Four: Regular and Context-Free Languages (12 Points)

On Problem Set Six, we had you design a number of DFAs, NFAs, and regular expressions, which we then graded using an autograder one of the TAs (Kevin) put together. You might be wondering how exactly that autograder works. In this problem, you'll get an answer to that question!

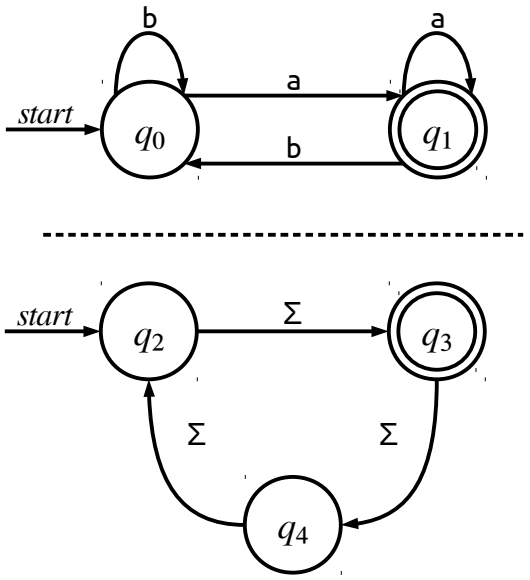
Let's suppose that you have two DFAs D_1 and D_2. It's possible to build a new DFA called the ***XOR automaton*** of D_1 and D_2, denoted $\boldsymbol{D_1 \times D_2}$, that simulates the execution of D_1 and D_2 simultaneously. While there are many different ways to define the XOR automaton, one simple definition involves a minor variation on the subset construction.

Given two DFAs D_1 and D_2, you can construct $D_1 \times D_2$ as follows:

1. Add a new start state q_s to the automaton with ε-transitions to the start states of D_1 and D_2.
2. Perform the subset construction on the resulting NFA to produce a new DFA.
3. For each state in the resulting DFA, mark that state as an accepting state if it corresponds to a set containing either (1) an accepting state in D_1 and a rejecting state in D_2, or (2) a rejecting state in D_1 and an accepting state in D_2. Then, mark each other state as rejecting. The resulting automaton is $D_1 \times D_2$.

The automaton $D_1 \times D_2$ has the useful property that every string it accepts is accepted by exactly one of D_1 and D_2. It's therefore ideal for use in an autograder. We can take a student submission as D_1, our reference solution as D_2, then build $D_1 \times D_2$. If $D_1 \times D_2$ accepts any strings, then we know that the submitted DFA D_1 is incorrect because it either rejects something it should have accepted or accepts something it should have rejected.

i. **(4 Points)** Apply the above algorithm to the two DFAs given below (each of which has alphabet {a, b}) and give us an example of a string accepted by exactly one of D_1 and D_2. Give the resulting DFA as a transition table in the space provided. To be nice, we've given you exactly the number of rows you'll need.



	a	b

String accepted by exactly one of D_1 and D_2: _____

Let $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}\}$ and consider the following language L_1 over Σ:

$$L_1 = \{\ w \in \Sigma^* \mid \text{no two consecutive characters in } w \text{ are the same } \}$$

For example, $\texttt{abcaba} \in L_1$, $\texttt{baba} \in L_1$, $\texttt{a} \in L_1$, and $\varepsilon \in L_1$, but $\texttt{a}\underline{\texttt{bb}}\texttt{a} \notin L_1$, $\texttt{acba}\underline{\texttt{cc}} \notin L_1$, and $\underline{\texttt{aa}} \notin L_1$.

ii. **(4 Points)** Write a context-free grammar for L_1.

Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. Consider the following language L_2 over Σ:

$$L_2 = \{\ \mathtt{a}^n\mathtt{b}^m \mid m, n \in \mathbb{N} \text{ and } m \leq 2n\ \}$$

For example, $\mathtt{aa} \in L_2$, $\mathtt{aab} \in L_2$, $\mathtt{aabb} \in L_2$, $\mathtt{aabbb} \in L_2$, and $\mathtt{aabbbb} \in L_2$, but $\mathtt{aabbbbb} \notin L_2$.

iii. **(4 Points)** Prove that L_2 is not a regular language. In your proof, if you claim that a particular string does or does not belong to L_2, please be as specific as possible in your justification; the requirements on strings in L_2 are a bit harder to determine from context than for some of the other languages we've encountered in the past.

Problem Five: R and RE Languages (14 Points)

In lecture, we proved that L_D is not an **RE** language using diagonalization. How else might we prove a language is not in **RE**? Using the Double Verification problem from Problem Set Nine, there's an elegant way to discover non-**RE** languages by starting with recognizable but undecidable problems.

i. **(3 Points)** Let L be a language in **RE – R**. Prove that $\overline{L} \notin$ **RE**.

In lecture, we proved that the problem of checking whether a program is a secure voting machine is undecidable. On Problem Set Nine, you proved that the problem of checking whether a program is a valid password checker was also undecidable. These results are actually special cases of a broader theorem that's the focus of this problem.

Let $L \in \mathbf{R}$ be an arbitrary decidable language. Consider the following language *Rec*(*L*):

$$Rec(L) = \{ \langle M \rangle \mid M \text{ is a TM and } \mathscr{L}(M) = L \}.$$

In other words, *Rec*(*L*) is the language of all TMs whose language is precisely the decidable language *L*. The two problems described above are just special cases of *Rec*(*L*) for particular choices of *L*.

ii. **(5 Points)** Prove that if *L* is any decidable language, then $Rec(L) \notin \mathbf{R}$. As a hint, if *Rec*(*L*) is decidable, then you can write a method that represents a decider for *Rec*(*L*):

```
bool recognizesL(string program)
```

Similarly, since *L* is decidable, you can write a method representing a decider for *L*:

```
bool isInL(string w)
```

Then, consider what the following program does:

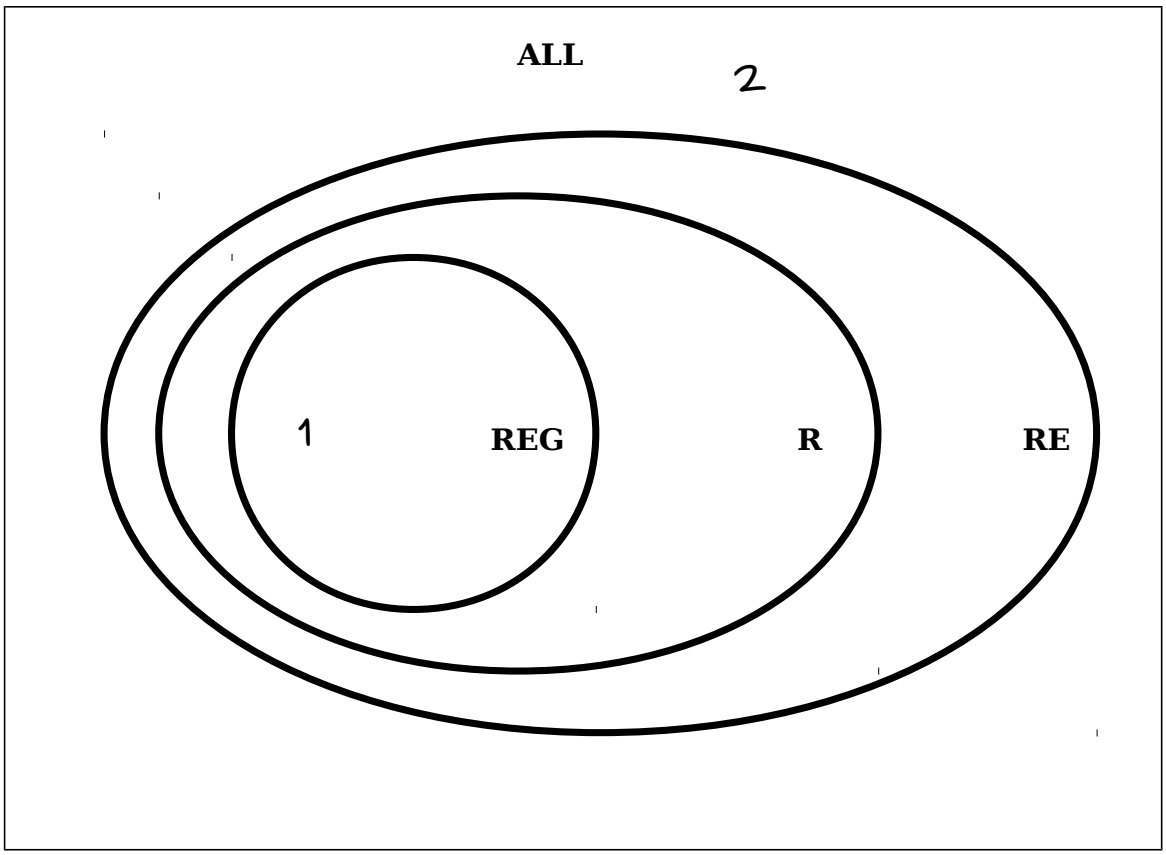
```
int main() {
    string me = mySource();
    string input = getInput();

    bool answer = isInL(input);
    if (recognizesL(me)) {
       answer = !answer;
    }

    if (answer) {
        accept();
    } else {
        reject();
    }
}
```

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

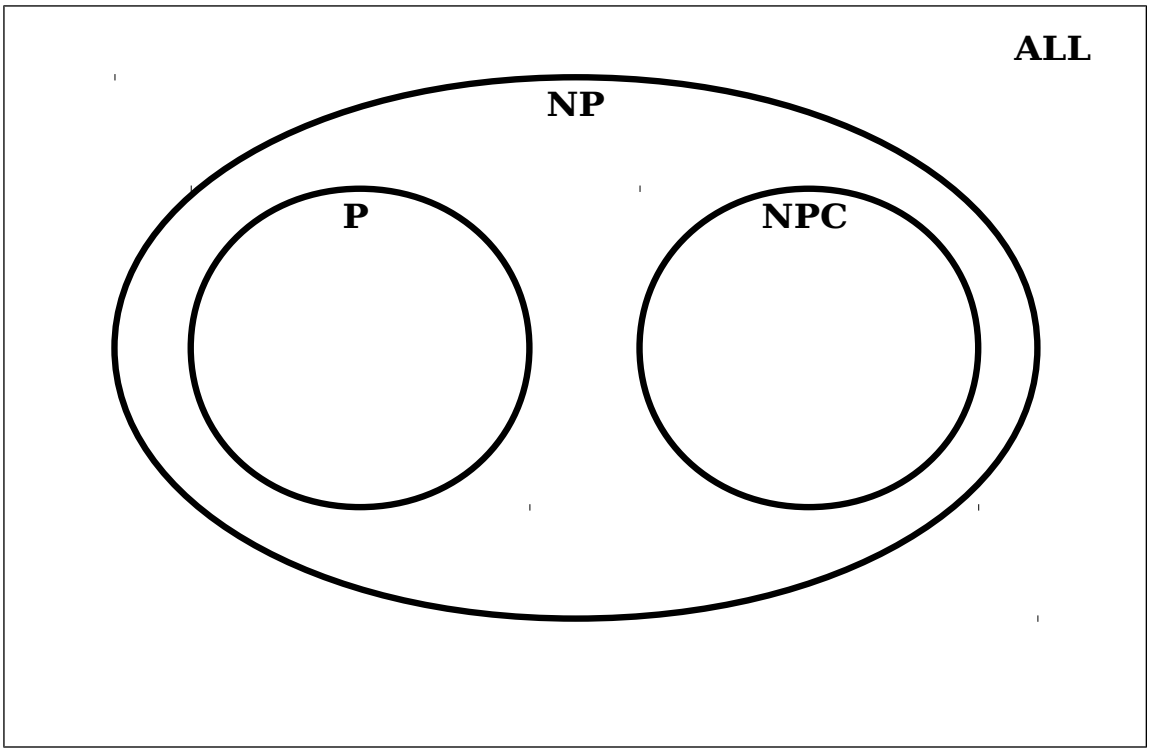
iii. **(6 Points)** Below is a Venn diagram showing the overlap of different classes of languages we've studied so far. We have also provided you a list of numbered languages. For each of those languages, draw where in the Venn diagram that language belongs. As an example, we've indicated where Language 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. { $w \in$ {a, b, c, …, z, A, B, C, …, Z, 0, 1, …, 9}* | w contains at least two lower-case letters, at least two upper-case letters, and a digit; w doesn't end with a digit; and $|w| \geq 8$ }
4. { $\langle P \rangle$ | P is a syntactically correct Java program } *(If you don't know Java, don't worry! The answer is the same if you replace Java with any of C, Python, JavaScript, C#, Visual Basic, or Scheme, so feel free to reason about those languages instead.)*
5. {$\langle U_{TM} \rangle$} (As a reminder, U_{TM} is the universal Turing machine.)
6. $\mathscr{L}(U_{TM})$
7. The complement of language (6)
8. { $\langle M, w \rangle$ | M is a TM, w is a string, and U_{TM} rejects $\langle M, w \rangle$ }

Problem Six: P and NP Languages (4 Points)

Let's suppose that someone does manage to prove that **P** ≠ **NP**. In that case, the landscape of **P**, **NP**, and the languages outside of **NP** looks something like this:



Note that there are four regions in this Venn diagram: **P**, **NP**, **NPC**, and **ALL**.

i. **(2 Points)** Let *L* be a language where $SAT \leq_p L$. Assuming **P** ≠ **NP**, where could *L* be in the above Venn diagram? Indicate your answer by writing a (1) in each possible location. No proof or justification is necessary.

ii. **(2 Points)** Let *L* be a language where $L \leq_p SAT$. Assuming **P** ≠ **NP**, where could *L* be in the above Venn diagram? Indicate your answer by writing a (2) in each possible location. No proof or justification is necessary.

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**