Practice CS103 Final Exam

This exam is closed-book and closed-computer. You may have a double-sided, 8.5" × 11" sheet of notes with you when you take this exam. You may not have any other notes with you during the exam. You may not use any electronic devices during the course of this exam without prior authorization from the course staff. Please write all of your solutions on this physical copy of the exam.

You are welcome to cite results from the problem sets or lectures on this exam. Just tell us what you're citing and where you're citing it from. However, please do not cite results that are beyond the scope of what we've covered in CS103.

> On the actual exam, there'd be space here for you to write your name and sign a statement saying you abide by the Honor Code. We're not collecting or grading this exam (though you're welcome to step outside and chat with us about it when you're done!) and this exam doesn't provide any extra credit, so we've opted to skip that boilerplate.

This practice exam was the final exam given out in Fall 2015. The problems here are therefore a good representative sample of what you could expect to see on this quarter's final exam.

(signed) _____

You have three hours to complete this exam. There are 48 total points.

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

Problem One: Logic and Relations (6 Points)

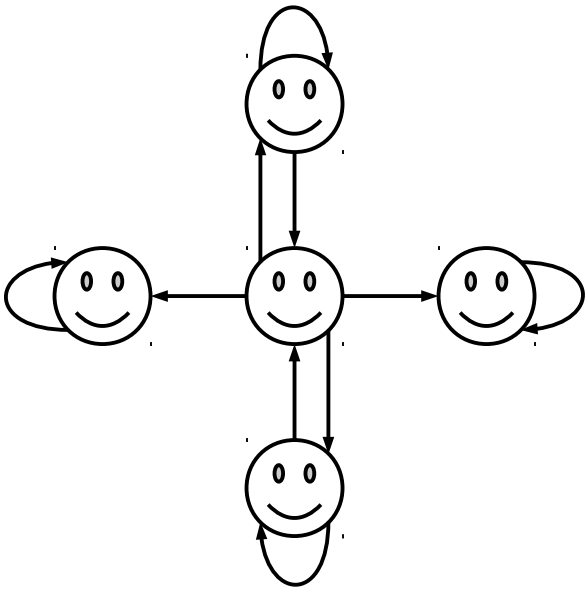
Suppose that you want to prove the implication $P \to Q$. Here are two possible routes you can take:

- Prove the implication by contradiction.
- Take the contrapositive of the implication, then prove the contrapositive by contradiction.

It turns out that these two proof approaches are completely equivalent to one another.

i. **(2 Points)** State, in propositional logic, which statements you will end up assuming if you were to use each of the above proof approaches, then *briefly* explain why they're equivalent.

ii. **(4 Points)** Below is a drawing of a binary relation R over a set of people A:



For each of the following first-order logic statements about R, decide whether that statement is true or false. No justification is required, and there is no penalty for an incorrect guess.

1. $\forall p \in A.\ \exists q \in A.\ pRq$

   ☐ True ☐ False

2. $\exists p \in A.\ \forall q \in A.\ pRq$

   ☐ True ☐ False

3. $\exists p \in A.\ (pRp \rightarrow \forall q \in A.\ qRq)$

   ☐ True ☐ False

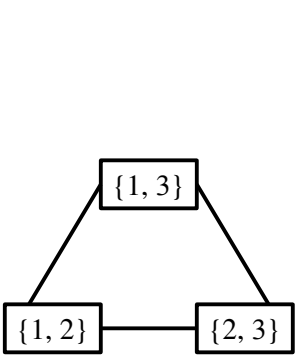
4. $\neg\forall p \in A.\ \forall q \in A.\ (p \neq q \rightarrow \exists r \in A.\ (pRr \wedge qRr))$

   ☐ True ☐ False

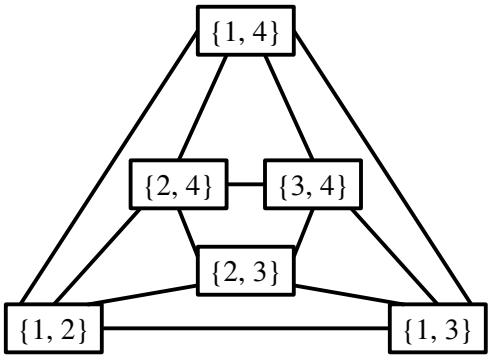
Problem Two: Graphs and Sets (6 Points)

Recently, there's been a major development in complexity theory: an “almost” efficient algorithm for the graph isomorphism problem. The algorithm relies on a special class of graphs that are the focus of this problem.

The ***triangular graph of order n***, denoted T_n, is a graph defined as follows. Begin with the set $\{1, 2, 3, \ldots, n\}$. The nodes in T_n are the two-element subsets of $\{1, 2, 3, \ldots, n\}$, and there's an edge between any two sets that have exactly one element in common. For example, below are the graphs T_3 and T_4:



The graph T_3



The graph T_4

Recall from Problem Set Four that an ***independent set*** in an undirected graph $G = (V, E)$ is a set $I \subseteq V$ such that if $x \in I$ and $y \in I$, then $\{x, y\} \notin E$. Intuitively, an independent set in G is a set of nodes where no two nodes in I are adjacent. The ***independence number*** of a graph G, denoted $\alpha(G)$, is the size of the largest independent set in G.

Prove that if $n \in \mathbb{N}$ and $n \geq 1$, then $\alpha(T_{2n}) \doteq n$. *(Hint: You need to prove two separate results: first, that there's an independent set of size n in T_{2n}; second, that no larger independent set exists in T_{2n}.)*

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

Problem Three: Induction and Cardinality (6 Points)

Consider the following series:

$$-1 + 2 - 3 + 4 - 5 + 6 - 7 + 8 - 9 + 10 - 11 + 12 - 13 + 14 - 15 \ldots$$

We can think about evaluating larger and larger number of terms in the summation. For example, the sum of the first five terms is $-1 + 2 - 3 + 4 - 5 = -3$, and the sum of the first eight terms works out to $-1 + 2 - 3 + 4 - 5 + 6 - 7 + 8 = 4$. For notational simplicity, let's define A_n to be the sum of the first n terms in the summation. For example, A_0 is the sum of the first zero terms in the summation (that's the empty sum, which is zero). A_1 is the sum of the first term (-1), A_2 is the sum of the first two terms ($-1 + 2 = 1$), A_3 is the sum of the first three terms ($-1 + 2 - 3 = -2$), etc.

When we covered cardinality in lecture, we gave the following piecewise function as an example of a bijection $f : \mathbb{N} \to \mathbb{Z}$:

$$f(n) = \begin{cases} \dfrac{n}{2} & \text{if } n \text{ is even} \\ -\dfrac{n+1}{2} & \text{otherwise} \end{cases}$$

It turns out that this function is closely connected to the above series. Specifically, for every natural number n, the following is true:

$$A_n = f(n)$$

In other words, you can form a bijection from $\mathbb{N}$ to $\mathbb{Z}$ by considering longer and longer alternating sums of the natural numbers. Weird, isn't it?

Prove by induction on n that if $n \in \mathbb{N}$, then $A_n = f(n)$.

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

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

A ***hard reset sequence*** for a DFA is a string *w* with the following property: starting from any state in the DFA, if you read *w*, you end up in the DFA's start state.

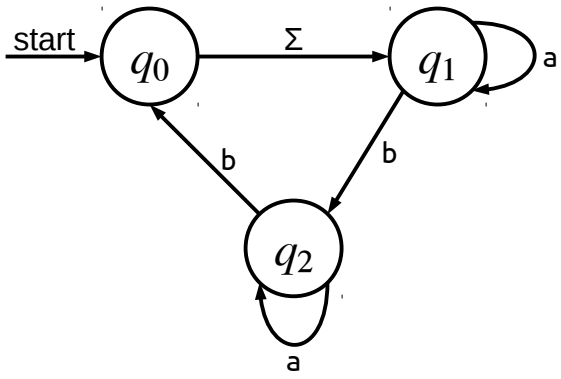
Hard reset sequences have many practical applications. For example, suppose you're remotely controlling a Mars rover whose state you're modeling as a DFA. Imagine there's a hardware glitch that puts the Mars rover into a valid but unknown state. Since you can't physically go to Mars to pick up the rover and fix it, the only way to change the rover's state would be to issue it new commands. To recover from this mishap, you could send the rover a hard reset sequence. Regardless of what state the rover got into, this procedure would guarantee that it would end up in its initial configuration.

Here is an algorithm that, given any DFA, will let you find every hard reset sequence for that DFA:

1. Add a new start state q_s to the automaton with ε-transitions to every state in the DFA.
2. Perform the subset construction on the resulting NFA to produce a new DFA called the ***power automaton***.
3. If the power automaton contains a state corresponding solely to the original DFA's start state, make that state the only accepting state in the power automaton. Otherwise, make every state in the power automaton a rejecting state.

This process produces a new automaton that accepts all the hard reset sequences of the original DFA. It's possible that a DFA won't have any hard reset sequences (for example, if it contains a dead state), in which case the new DFA won't accept anything.

i. **(4 Points)** Apply the above algorithm to the following DFA and give us a hard reset sequence for that DFA. For simplicity, please give the subset-constructed DFA as a transition table rather than a state-transition diagram. We've given you space for the table over to the right, and to be nice, we've given you exactly the number of rows you'll need.



	a	b

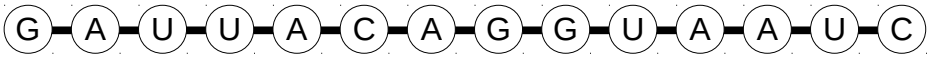
Sample hard reset sequence: _____

RNA strands consist of strings of *nucleotides*, molecules which encode genetic information. Computational biologists typically represent each RNA strand as a string made from four different letters, `A`, `C`, `G`, and `U`, each of which represents one of the four possible nucleotides.

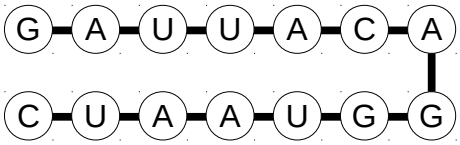
Each of the the four nucleotides has an affinity for a specific other nucleotide. Specifically:

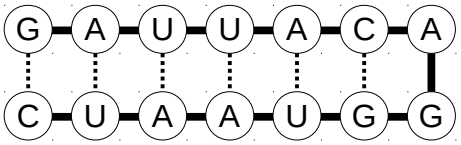
`A` has an affinity for `U` (and vice-versa) `C` has an affinity for `G` (and vice-versa)

This can cause RNA strands to fold over and bind with themselves. Consider this RNA strand:



If you perfectly fold this RNA strand in half, you get the following:





Notice that each pair of nucleotides – except for the `A` and the `G` on the far right – are attracted to the corresponding nucleotide on the other side of the RNA strand. Because of the natural affinities of the nucleotides in the RNA strand, the RNA strand will be held in this shape. This is an example of an *RNA hairpin*, a structure with important biological roles.

For the purposes of this problem, we'll say that an RNA strand forms a hairpin if

- it has even length (so that it can be cleanly folded in half);
- it has length at least four (there is at least one pair holding the hairpin shut); and
- all of its nucleotides, except for the middle two, have an affinity for its corresponding nucleotide when folded over. (The middle two nucleotides in a hairpin might coincidentally have an affinity for one another, but it's not required. For example, `CAUG` forms a hairpin.)

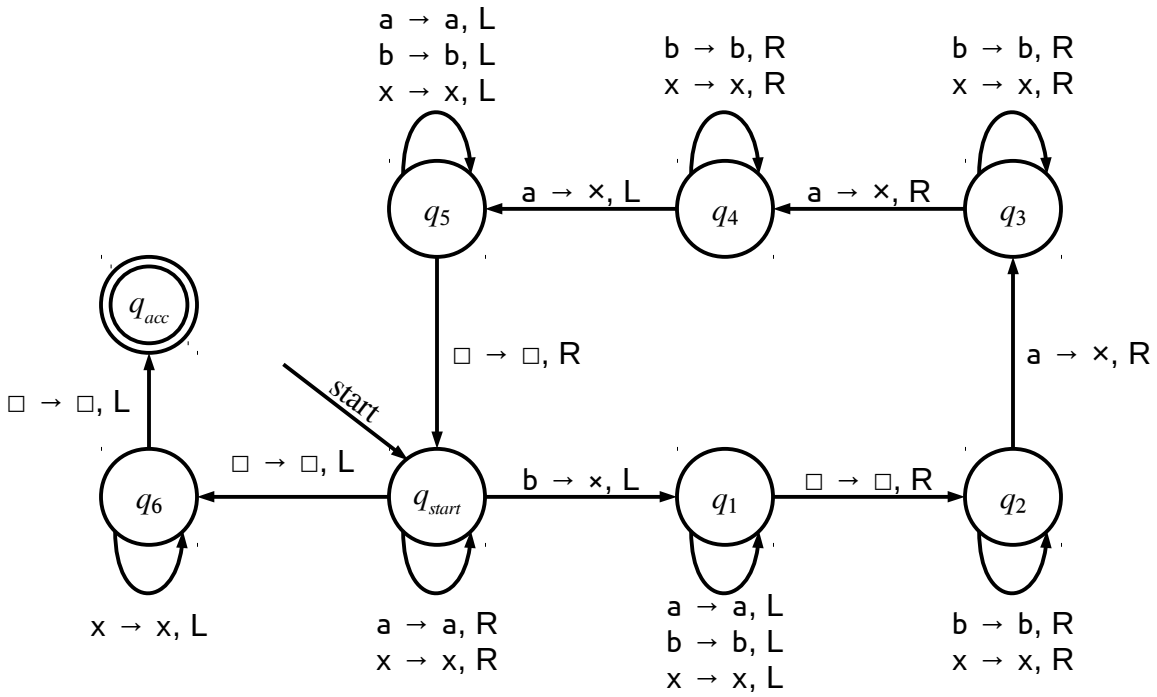
Let $\Sigma = \{$`a`, `c`, `g`, `u`$\}$ and let $L_{RNA} = \{\ w \in \Sigma^* \mid w$ represents an RNA strand that forms a hairpin $\}$. For example, the strings `gacccguc`, `guac`, `uuuuuaaaaa`, and `ccaaccuugg` are all in L_{RNA}, but the strings `au`, `aaaacuuuu`, `ggc`, and `guuuuaaaag` are all not in L_{RNA}.

ii. **(4 Points)** Write a context-free grammar for L_{RNA}. It should fit into the space below.

iii. **(4 Points)** Use the Myhill-Nerode theorem to prove that the language L_{RNA} from part (ii) of this problem is not regular. Since this language imposes a lot of requirements on the strings it contains, if in the course of your proof you want to claim that a particular string is or is not in L_{RNA}, please articulate clearly why the string does or does not meet all of the requirements of strings in L_{RNA}.

Problem Five: R and RE Languages (14 Points)

Consider the following TM, which we'll call TM_6:



Here, q_{start} is the start state, and q_{acc} is the accepting state. As usual, we assume that all missing transitions implicitly cause *M* to reject.

TM_6's input alphabet is $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ and its tape alphabet is $\Gamma = \{\mathtt{a}, \mathtt{b}, \times, \square\}$.

i. **(3 Points)** Fill in the following blank to let us know what the language of TM_6 is. You may find it useful to run this TM on a few small sample inputs to get a feel for how it works. No justification is necessary.

$\mathscr{L}(TM_6) = \{\ w \in \Sigma^* \mid$ _____ $\}$

Let Σ be an arbitrary alphabet and consider the following language:

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

In other words, A_{ALL} is the language of all descriptions of TMs that accept every string.

ii. **(5 Points)** Prove that $A_{ALL} \notin$ **RE**. As a hint, if A_{ALL} is in **RE**, then you can build a verifier for it, which corresponds to a function like this one:

```
bool imConvincedAcceptsEverything(string program, string certificate)
```

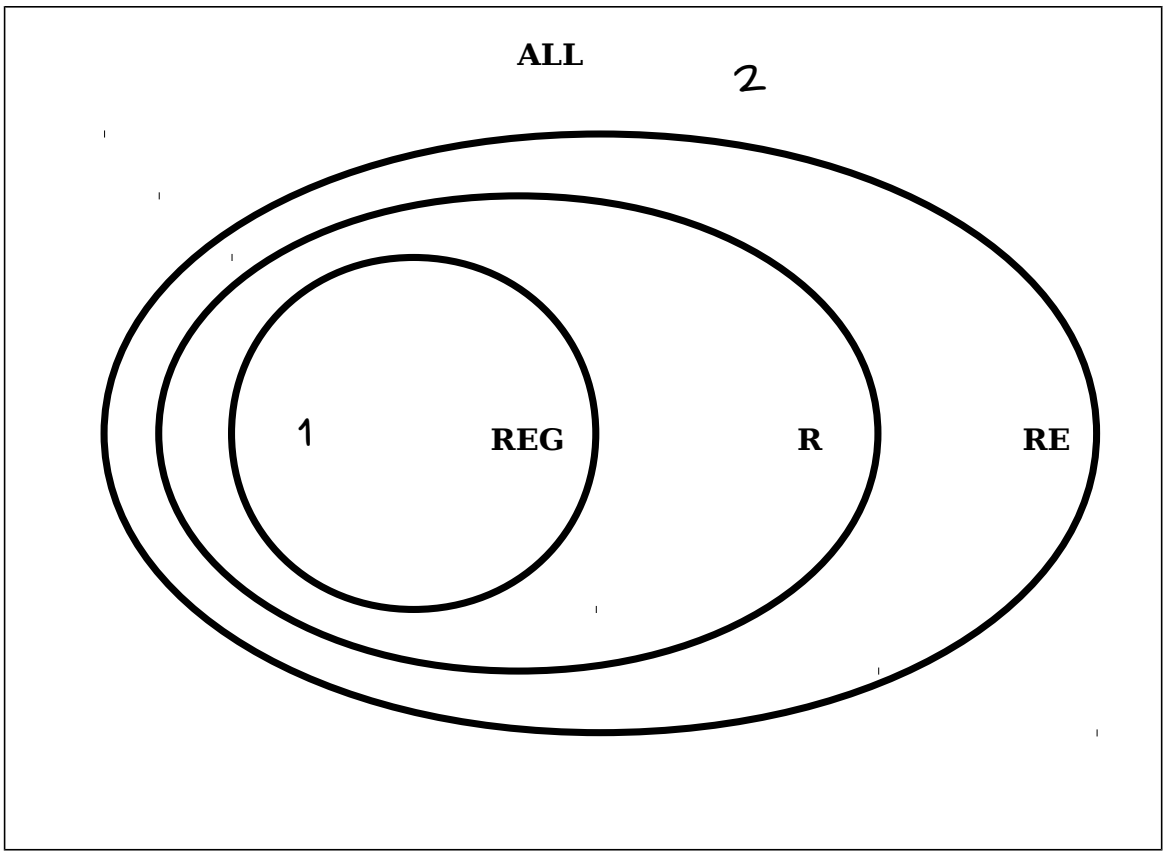
Then, think about what this program does:

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

    if (imConvincedAcceptsEverything(me, input)) {
        reject();
    } else {
        accept();
    }
}
```

(Extra space for your answer to Problem 5.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. { $w \in \{\texttt{a}, \texttt{b}\}^*$ | $|w| \geq 100$ and the first 50 characters of w are the same as the last 50 characters of w }
4. { $\langle M_1, M_2, M_3 \rangle$ | M_1, M_2, and M_3 are TMs over the same alphabet Σ and every string in Σ^* belongs to exactly one of $\mathscr{L}(M_1)$, $\mathscr{L}(M_2)$, or $\mathscr{L}(M_3)$ }
5. $HALT - A_{TM}$
6. $A_{TM} - HALT$
7. { $\langle V, w \rangle$ | V is a TM and there is a string c such that V accepts $\langle w, c \rangle$ }
8. { $w \in \{\texttt{r}, \texttt{d}\}^*$ | w has more r's than d's }

Problem Six: P and NP Languages (4 Points)

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

i. If **P** = **NP**, there are no **NP**-***complete*** problems in **P**.

☐ True ☐ False

ii. If **P** = **NP**, there are no **NP**-***hard*** problems in **P**.

☐ True ☐ False

iii. If **P** ≠ **NP**, there are no **NP**-***complete*** problems in **P**.

☐ True ☐ False

iv. If **P** ≠ **NP**, there are no **NP**-***hard*** problems in **P**.

☐ 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**