Another 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 However, please do not cite results that are beyond the scope of what we've covered in CS103.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 is formed from questions that have been given on final exams in previous quarters (though not all at the same time), so it's a good representative of what you might expect to get on the actual final exam.

(signed) ____

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

Question	Points	Graders
(1) Induction	/ 4	
(2) First-Order Logic	/ 5	
(3) Sets and Functions	/ 5	
(4) Regular Languages	/ 12	
(5) Context-Free Languages	/ 3	
(6) **R** and **RE** Languages	/ 15	
(7) **P** and **NP** Languages	/ 4	
	/ 48	

Problem One: Induction (4 Points)

(Final Exam, Spring 2015)

If you're hungry, you can make yourself a cheese sandwich by taking two pieces of bread and putting a slice of cheese between them. If you're really hungry, you can make a *cheese metasandwich* by making two cheese sandwiches and putting a slice of cheese between them. If you're really, really hungry, you can make a *cheese meta-metasandwich* by making two cheese metasandwiches and putting a slice of cheese between them.

Formally, we can define a hierarchy of sandwich variants as follows:

- An ***order 0 metasandwich*** is a normal cheese sandwich.
- An ***order n+1 metasandwich*** consists of two order n metasandwiches with a piece of cheese between them.

Determine formulas for the number of pieces of bread and slices of cheese in an order n metasandwich, then prove by induction that your formulas are correct. Your formulas should not be recurrence relations, by the way – it should be easy to directly evaluate your formulas to see how much bread and cheese is necessary to make an order n metasandwich.

Problem Two: First-Order Logic (5 Points)

(Final Exam, Spring 2015)

Suppose we have the predicates

- $Person(p)$, which states that p is a person, and
- $Loves(p, q)$, which states that p loves q.

Below are a series of five English statements paired with a statement in first-order logic. For each statement, decide whether the corresponding formula in first-order logic is a correct translation of the English statement and check the appropriate box. There is no penalty for an incorrect guess.

Everyone loves themselves.	$\forall p.\ (Person(p) \to \forall q.\ (Loves(p, q) \to p = q))$	☐ ***Correct*** ☐ ***Incorrect***
There are two people that everyone loves.	$\forall r.\ (Person(r) \to \exists p.\ (Person(p) \land \exists q.\ (Person(q) \land q \neq p \land Loves(r, p) \land Loves(r, q))))$	☐ ***Correct*** ☐ ***Incorrect***
Love is a transitive relation over the set of people.	$\forall p.\ (Person(p) \land \forall q.\ (Person(q) \land \forall r.\ (Person(r) \land (Loves(p, q) \land Loves(q, r) \to Loves(p, r)))))$	☐ ***Correct*** ☐ ***Incorrect***
No two people love exactly the same set of people.	$\forall p.\ (Person(p) \to \forall q.\ (Person(q) \land q \neq p \to \exists r.\ (Person(r) \land (Loves(p, r) \leftrightarrow \neg Loves(q, r)))))$	☐ ***Correct*** ☐ ***Incorrect***
Someone doesn't love anyone.	$\neg\forall p.\ (Person(p) \to \exists q.\ (Person(q) \land Loves(p, q)))$	☐ ***Correct*** ☐ ***Incorrect***

Problem Three: Sets and Functions (5 Points)

(Final Exam, Winter 2013)

There can be many functions from one set A to a second set B. This question explores how many functions of this sort there are.

For any set S, we will denote by 2^S the following set:

$$2^S = \{ f \mid f : S \to \{0, 1\} \}$$

That is, 2^S is the set of all functions whose domain is S and whose codomain is the set $\{0, 1\}$. Note that 2^S does *not* mean "two raised to the Sth power." It's just the notation we use to denote the set of all functions from S to $\{0, 1\}$.

Prove that if S is a nonempty set, then $|2^S| = |\wp(S)|$. To do so, find a bijection from 2^S to $\wp(S)$, then prove that your function is a bijection. Your proof should work for all sets S, including infinite sets.

You may find the following definition useful: if $f : A \to B$ and $g : A \to B$ are functions with the same domain and the same codomain, then we say that $f = g$ if $f(a) = g(a)$ for all $a \in A$.

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

Problem Four: Regular Languages (12 Points)

(Final Exam, Fall 2011)

Consider the following language over $\Sigma = \{0, 1\}$:

$$SANDWICH = \{ w \mid w \text{ starts and ends with the same symbol, and } |w| > 0 \}$$

For example, `010` ∈ *SANDWICH,* `111` ∈ *SANDWICH,* `1` ∈ *SANDWICH,* and `00` ∈ *SANDWICH.* However, ε ∉ *SANDWICH* (because it has length zero), `10` ∉ *SANDWICH* (because it doesn't begin and end with the same symbol), and `0101` ∉ *SANDWICH.*

i. **(3 Points)** Design a DFA that accepts *SANDWICH*.

ii. **(2 Points)** Design a regular expression for *SANDWICH*.

(Final Exam, Fall 2014)

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

$$L = \{ \ w \in \Sigma^* \mid w \text{ has odd length and its middle character is } a \ \}$$

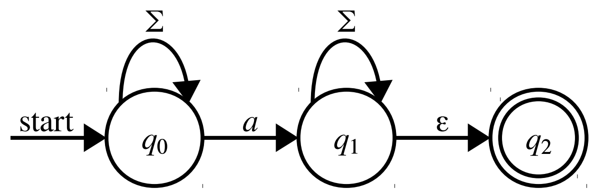
iii. **(5 Points)** Prove that L is not a regular language.

(Final Exam, Fall 2014)

As a reminder, the language L over $\Sigma = \{a, b\}$ from the previous page was defined as follows:

$$L = \{ \ w \in \Sigma^* \mid w \text{ has odd length and its middle character is } a \ \}$$

You just proved that this language is not regular. However, below is an NFA that purportedly has language L:



Here is a line of reasoning that claims that this NFA has language L:

> "Intuitively, this NFA will sit in state q_0 following its Σ transition until it nondeterministically guesses that it's about to read the middle a character. When it does, it transitions to q_1, where it keeps following the Σ transition as long as more characters are available. Finally, once it's read all the characters of the input, the NFA follows the ε transition from q_1 to q_2, where the NFA then accepts."

Of course, this reasoning has to be incorrect, since L is not a regular language.

iv. **(2 Points)** Without using the fact that L is not a regular language, explain why the above NFA is not an NFA for the language L.

Problem Five: Context-Free Languages (3 Points)

(Final Exam, Spring 2015)

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

$$L = \{\ w \in \Sigma^* \mid |w| \equiv_3 0 \text{ and all the characters in the first third of } w \text{ are the same }\}$$

For example, $\underline{\texttt{aa}}\texttt{baba} \in L$, $\underline{\texttt{bbb}}\texttt{aaaaaa} \in L$, $\underline{\texttt{a}}\texttt{aa} \in L$, and $\varepsilon \in L$, but $\underline{\texttt{ab}}\texttt{bbbb} \notin L$ and $\texttt{aaaaa} \notin L$. (For convenience, I've underlined the first third of the characters in each string.)

Write a context-free grammar for L.

Problem Six: R and RE Languages (15 Points)

(Final Exam, Spring 2015)

Let $\Sigma = \{$`(`, `)`$\}$. Consider the language $L = \{ w \in \Sigma^* \mid w$ is a string of balanced parentheses $\}$. For example, `(())` $\in L$, `(())(()())` $\in L$, and $\varepsilon \in L$, but `))((` $\notin L$, `())` $\notin L$, and `(()` $\notin L$.

i. **(5 Points)** Design a TM that is a decider for the language L. Please draw out an actual TM consisting of states and transitions rather than providing a high-level description of the TM. No justification is necessary.

   Some hints:

   - Our solution doesn't use very many states. If you find yourself drawing out a huge TM, you might want to reevaluate your solution.
   - Rather than searching for an open parenthesis and trying to find the close parenthesis that matches it, instead search for a close parenthesis and work backwards to find the nearest open parenthesis.
   - Remember that if a state in a TM has no transition defined for a particular character in the tape alphabet, the TM will automatically reject if it reads that character in that state.

(Final Exam, Spring 2015)

Although we typically haven't treated the class **RE** as a set, it is indeed a set of languages, so we can speak of subsets of the **RE** languages.

Let $S \subseteq$ **RE** be an arbitrary subset of the **RE** languages where $\emptyset \in S$ and $\Sigma^* \notin S$. We can then define a new language L_S as follows:

$$L_S = \{ \langle M \rangle \mid M \text{ is a TM and } \mathscr{L}(M) \in S \}$$

In other words, L_S is the set of all TMs whose language is one of the languages included in set S.

ii. **(5 Points)** Prove that L_S is not an **RE** language. As a hint, think about the following self-referential program:

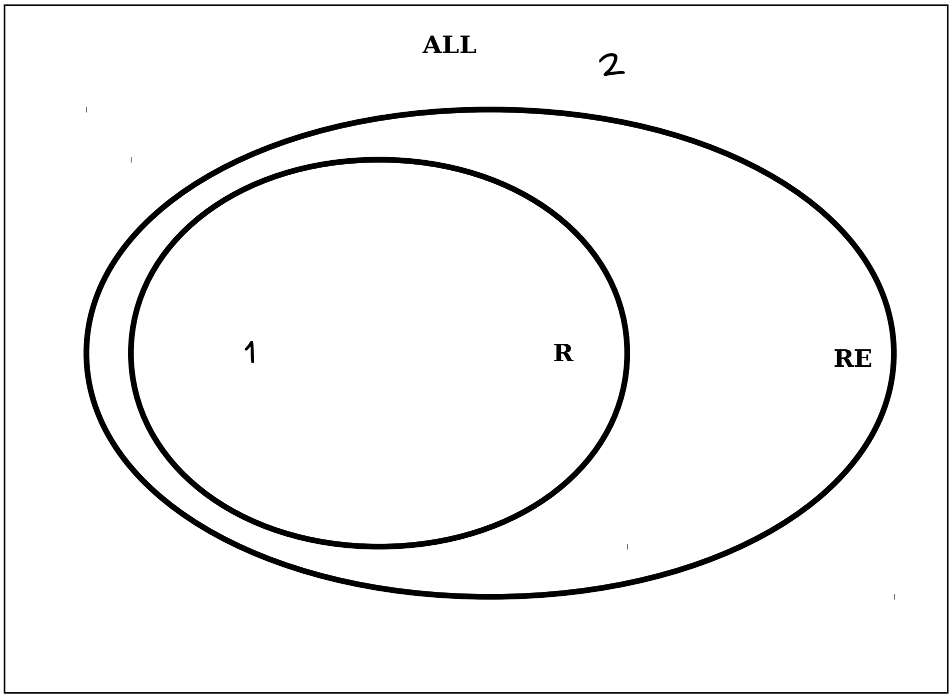
```
int main() {
    string me = mySource();
    string input = getInput();

    for (int i = 0 to ∞) {
        for (each string c of length i) {
            if (imConvincedIsInLS(me, c)) {
                accept();
            }
        }
    }
}
```

(More space for Problem 7.ii, if you need it)

(Final Exam, Spring 2015)

iii. **(5 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 seven 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. { $\langle D, w \rangle$ | D is a DFA and D does not accept w }
4. { $\langle R, w \rangle$ | R is a regular expression and R does not match w }
5. { $\langle M, w \rangle$ | M is a TM and M does not accept w }
6. { $\langle M \rangle$ | M is a TM and M accepts $\langle M \rangle$ }
7. { $\langle M \rangle$ | M is a TM and there is no verifier for $\mathscr{L}(M)$ }

Problem Seven: P and NP Languages (4 Points)

(Final Exam, Spring 2015)

In class, we saw that the language *INDSET* is **NP**-complete. If you'll recall, *INDSET* is defined as

$$INDSET = \{ \langle G, k \rangle \mid G \text{ is an undirected graph that has an independent set of size at least } k \}$$

Now, consider the following language:

$$INDEPENDENCE = \{ \langle G, k \rangle \mid G \text{ is an undirected graph and } \alpha(G) = k \}$$

(Recall from Problem Set Four that $\alpha(G)$ is the size of the largest independent set in G). It turns out that it's known that *INDEPENDENCE* is **NP**-hard, but it's not known whether it's **NP**-complete because it's not known whether *INDEPENDENCE* $\in$ **NP**.

Using the verifier intuition for **NP**, explain why it's unlikely that *INDEPENDENCE* $\in$ **NP** even though the related language *INDSET* is in **NP**. Specifically, explain at a high level how a polynomial-time verifier for *INDSET* would work, then explain at a high level why a similar polynomial-time verifier would not work for *INDEPENDENCE*. No formal proof is necessary.