Additional Practice CS103 Final Exam

This additional practice final exam **is not for extra credit**, but is designed to give you extra practice for the final exam. This exam was given in the Spring 2012 offering of CS103 and is similar in structure to the exam for this quarter.

Question		Points	Grader
(1) Relations Revisited	(30)	/ 30	
(2) Regular Languages	(30)	/ 30	
(3) Context-Free Languages	(25)	/ 25	
(4) **R** and **RE** Languages	(60)	/ 60	
(5) **P** and **NP** Languages	(35)	/ 35	
	(180)	**/ 180**	

Problem One: Relations Revisited (30 Points)

Recall that a binary relation R over a set A is formally defined as a subset of $A \times A$; that is, R is a set of ordered pairs (x, y) where xRy iff $(x, y) \in R$. This means that in addition to our previous treatment of relations, we can consider relations from a set-theoretic perspective.

(i) Properties of Equivalence Relations (15 Points)

Prove or disprove: Every binary relation R over a set A is a subset of some equivalence relation over the set A.

(ii) Properties of Partial Orders (15 Points)

Prove or disprove: Every binary relation R over a set A is a subset of some partial order relation over the set A.

Problem Two: Regular Languages (40 Points Total)

Given two strings of `0`s and `1`s, we say that those strings have the same `1`-parity iff both of the strings contain an odd number of `1`s or both of the strings contain an even number of `1`s.

Consider the following language over Σ = {`0`, `1`, `M`}:

1PARITY = { *w* `M`*x* | *w*, *x* ∈ {`0`, `1`}* and *w* and *x* have the same `1`-parity }

For example, `01M111` ∈ *1PARITY*, `0011M111111` ∈ *1PARITY*, and `M` ∈ *1PARITY*. However, `1M0` ∉ *1PARITY*, `MM` ∉ *1PARITY*, `00M01` ∉ *1PARITY*, and ε ∉ *1PARITY*.

(i) Finite Automata (10 Points)

Design a DFA for *1PARITY*.

(ii) Regular Expressions (10 Points)

Write a regular expression for *1PARITY*.

Consider the following language over the alphabet Σ = {`1`, `≥`}:

$GE = \{ \mathtt{1}^m \geq \mathtt{1}^n \mid m, n \in \mathbb{N} \text{ and } m \geq n \}$

(iii) The Pumping Lemma (15 Points)

Using the pumping lemma for regular languages, prove that *GE* is not regular. To help out, we have sketched out a part of the proof; you should fill in the appropriate blanks.

Proof: By contradiction; assume that *GE* is regular. Let *n* be the length guaranteed by the pumping lemma. Then consider the string *w* = [ ] We then have that *w* ∈ *GE* and |*w*| ≥ *n*, so by the pumping lemma we can write *w* = *xyz* such that |*xy*| ≤ *n*, *y* ≠ ε, and for any *i* ∈ ℕ, xy^iz ∈ *GE*.

(finish the proof in the box below)

We have reached a contradiction, so our assumption was wrong and CONTAINS is not regular. ■

Problem Three: Context-Free Languages (25 Points Total)

(i) Writing CFGs (10 Points)

Let Σ = {`0`, `1`}. Consider the language *NEP* defined as follows:

$$NEP = \{ w \in \Sigma^* \mid w \text{ is not an even-length palindrome } \}$$

For example, `0111` ∈ *NEP*, `101` ∈ *NEP*, `101010` ∈ *NEP*. However, `10100101` ∉ *NEP*, ε ∉ *NEP*, and `0000` ∉ *NEP*. Write a CFG for *NEP*.

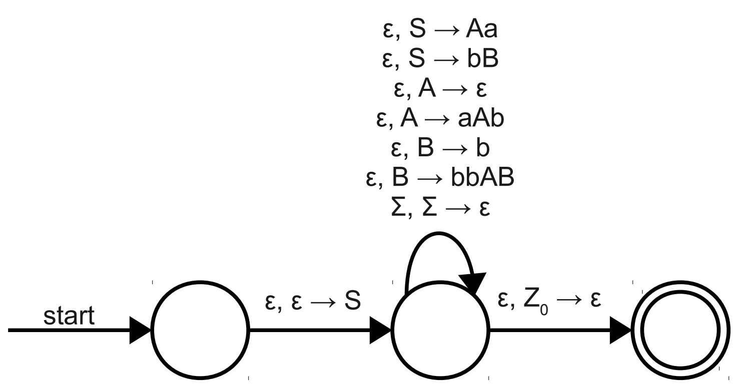
In lecture, we sketched a proof that if a language is context-free, there is a PDA for that language. Our proof constructed a PDA from an arbitrary CFG that tried to simulate a derivation of the input string from the start symbol. The construction built a PDA with three states:

- A **start** state that sets up the PDA's stack with the start symbol of the grammar.
- A **parsing** state where each transition simulates either *predicting* which production to use or *matching* a predicted symbol with the next character of the input.
- An **accepting** state entered when we find that the input string could be parsed.

The only part of the PDA that depends on the grammar is the set of transitions from the parsing state to itself. This state always has the transition Σ, Σ → ε to itself. Additionally, it has one transition for every production in the grammar. For example, given the following grammar:

S → Aa | bB
A → ε | aAb
B → b | bbAB

The resulting PDA is as follows:



Given a grammar *G*, let's denote by P(*G*), the automaton constructed this way. For most grammars *G*, P(*G*) is an NPDA. However, there are grammars *G* for which P(*G*) is a DPDA.

(ii) Deterministic Parsing Automata (15 Points)

What property or properties must a context-free grammar *G* have for P(*G*) to be a DPDA? Explain why P(*G*) is a DPDA iff *G* has the property or properties that you describe, though you don't need to formally prove it. Make sure to address both directions of implication.

Problem Four: R and RE Languages (60 Points Total)

(i) RE and Verifiers (20 Points)

Recall that a verifier for a language L is a Turing machine V such that

$$w \in L \quad \text{iff} \quad \exists x \in \Sigma^*.\ V \text{ accepts } \langle w, x \rangle$$

In the context of **NP** we considered polynomial-time verifiers, verifiers that run in time polynomial in the size of w. Let's relax this description and define a *deciding verifier* to be a verifier V for a language L such that V is a decider (that is, V halts on all inputs).

Prove that if there is a deciding verifier for a language L, then $L \in$ **RE**.

(ii) Unsolvable Problems (25 Points)

Consider the following language L_{some}:

$$L_{\text{some}} = \{ \langle M \rangle \mid \mathscr{L}(M) \neq \emptyset \text{ and } \mathscr{L}(M) \neq \Sigma^* \}$$

Prove that $L_{\text{some}} \notin$ **RE**.

(iii) Properties of Reductions (15 Points)

Prove or disprove: If $L_1 \leq_M L_2$ and $L_1 \leq_M L_3$, then $L_1 \leq_M L_2 \cap L_3$.

Problem Five: P and NP Languages (35 Points Total)

Just how hard are the **NP**-complete problems? In a sense, they are the "hardest" problems in **NP**, because a solution to any one of them can be used to solve all other **NP** problems. How accurate is that intuition?

It turns out that is possible to construct languages that are **NP**-complete but which can be decided efficiently in many cases. One way to do this uses the disjoint union operation that you saw in Problem Set 8. Recall that given language L_1 and L_2 over $\{0, 1\}^*$, the disjoint union $L_1 \uplus L_2$ is

$$L_1 \uplus L_2 = \{ 0w \mid w \in L_1 \} \cup \{ 1w \mid w \in L_2 \}$$

(i) Relatively Easy NP-Complete Languages (20 Points)

Let L_1 be an **NP**-complete language and let L_2 be any language in **P**. Prove that $L_1 \uplus L_2$ is **NP**-complete. This new language, while **NP**-complete, is easy for many inputs; we can decide in polynomial-time whether any string starting with a `1` is contained within $L_1 \uplus L_2$.

(ii) $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ (15 Points)

What would it take to prove whether P = NP?

Below are ten statements. For each statement, if the statement would definitely prove that **P** = **NP**, write "**P** = **NP**" on the appropriate line. If the statement would definitely prove that **P** ≠ **NP**, write "**P** ≠ **NP**" on the appropriate line. If the statement would not prove either result, write "neither" on the appropriate line. No explanation is necessary.

There is a regular expression for SAT. __________

There is no regular expression for SAT. __________

There is a ***deterministic*** polynomial-time algorithm for SAT. __________

There is a ***nondeterministic*** polynomial-time algorithm for SAT. __________

Every $f(n)$-time ***single-tape*** NTM can be converted into a $f(n)^8$-time ***single-tape*** TM. __________

Every $f(n)$-time ***multitape*** NTM can be converted into a $f(n)^8$-time ***multitape*** TM. __________

For any k, there is a language in **NP** that cannot be decided in time $O(n^k)$. __________

There is a language in **NP** that, for any k, cannot be decided in time $O(n^k)$. __________

There is a polynomial-time TM that correctly decides SAT for all strings of length ***at most*** 10^{100}, but might give incorrect answers for longer strings. __________

There is a polynomial-time TM that correctly decides SAT for all strings of length ***at least*** 10^{100}, but might give incorrect answers for shorter strings. __________