University of Virginia - cs3102: Theory of Computation	Spring 2010
Exam 1 - Final Comments	**4 March 2010**

Before revisions:
Average: 71.6
Distribution: 90-100: 8; 80-89: 12; 70-79: 14; 60-69: 10; below 60: 11

After revisions:
Average: 79.625
Distribution: 90-100: 17; 80-89: 19; 70-79: 12; 60-69: 5; below 60: 8

Problem 1: Definitions. For each question, provide a correct, clear, precise, and concise answer from the perspective of a theoretical computer scientist.

a. (Average: 4.9 / 5) What is a *language*?

Answer: A language is a *set of strings*.

b. (4.7 / 5) How do we measure the *power* of a type of machine such as a DFA or NPDA?

Answer: We measure the power of a machine type by the *set of languages it can recognize*.

Note that it is not correct to say the "number" of languages. All of the machine types we have defined can recognize infinitely many languages.

c. (4.3 / 5) What does it mean for a machine model to be *nondeterministic*?

Answer: A nondeterministic machine can have more than one possible action for a given configuration; it always chooses an action that leads to an accepting state if there is such an action. (Alternately, it tries all possible actions, and if any sequence of actions leads to an accept state it accepts.) Just saying that there can be more than one transition at each step was worth 4 points, but is not a satisfactory definition since the always guesses correctly property is crucial to understanding nondeterministic machine models.

Problem 2: Language Classification.

For each of these questions, provide a convincing argument supporting the proposition. You may use any technique you wish to make your argument, and may assume any of the properties we have proved in class or in the book.

a. (2.9 / 5, semi-trick question) Show that the language produced by the context-free grammar below is a regular language:

$$S \quad \rightarrow \quad 0S0 \mid 1S1$$

Answer: Since there is no rule that replaces S with only terminals, the grammar cannot produce *any* strings. This means the language it describes is the empty language. This is a regular language since it is recognized by a DFA with no accepting states ($F = \emptyset$).

b. (4.7 / 5) Show that the language produced by the context-free grammar below is a regular language:

$$S \quad \rightarrow \quad 0S \mid S1 \mid \epsilon$$

Answer: The grammar describes the language 0^*1^*. This is a regular language since we can describe it with a regular expression.

c. (4.0 / 5) Show that the language $NOTREPEATING$ defined below is **not** a regular language:

$$NOTREPEATING = \{w | w \in \{0,1\}^* \wedge \text{there is no } x \text{ such that } w = xx\}$$

Answer: The easiest way to prove $NOTREPEATING$ is not a regular language is to use the property that the regular languages are closed under complement. This means that if we can prove that the complement of $NOTREPEATING$ is not a regular language, that proves that $NOTREPEATING$ itself is not a regular language. The complement of $NOTREPEATING$ is the language,

$$REPEATING = \{w | w \in \{0,1\}^* \wedge \text{there is an } x \text{ such that } w = xx\}$$

which is the same as the language, $\{ww | w \in \{0,1\}^*\}$. We have proven that this language is not context-free, so it must be non-regular.

It was also possible to get full credit for this question with an informal argument that explains that recognizing the language requires an infinite amount of state since we need to keep track of the entires first half of the input to know if the second half matches.

Another way to prove $NOTREPEATING$ is non-regular is to use the pumping lemma for regular languages. The easiest way to do this is to use the reverse version of the pumping lemma that starts with a string not in the language and pumps to produce a string in the language (which is just like using the complement property):

Assume $NOTREPEATING$ is a regular language with pumping length p. Choose $s = 0^p10^p1$. $s \notin NOTREPEATING$ since $s = vv$ where $v = 0^p1^p$. To satisfy the reverse pumping lemma, there must be a way to break s into $s = xyz$ with $|xy| \leq p$ such that $xy^iz \notin NOTREPEATING$ for all i. But, however we choose xyz, y must be a subset of the first group of 0s, $y = 0^m$ for some $m \geq 1$. If we select $i = 2$, the resulting string is $s_{i=2} = 0^{2m}0^{p-m}10^p1$. This cannot be divided into two equal strings, so $s_{i=2} \in NOTREPEATING$. Thus, we have a violation of the reverse pumping lemma and $NOTREPEATING$ must not be regular.

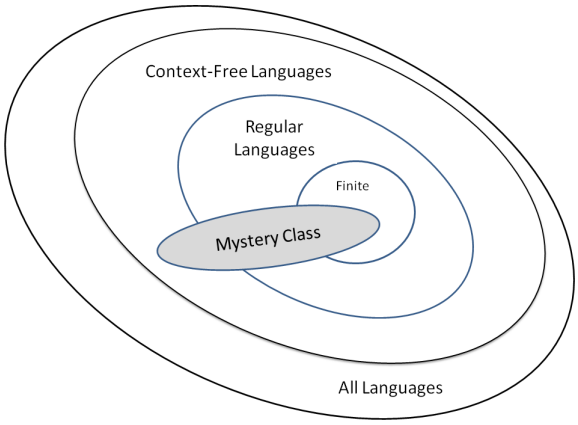
A much harder way to prove this is to use the standard pumping lemma. The challenge here is you need to find a string that is in the language $NOTREPEATING$, but can be pumped to produce a string that is not in the language. This means the original string is not of the form ww, but no matter how it is divided into xyz there is some string $xy^iz = vv$ that is repeating. No one provided a really satisfactory answer to this in the revisions, so this is now a challenge problem.

d. (3.7 / 5 + 5) A *squarefree* sequence is a sequence that contains no adjacent repeating subsequences of any non-zero length. For example, ab and $abcba$ are squarefree, but aa is not since it contains a^2 and $abcbacbab$ is not since it contains $(cba)^2$. Show the language consisting of all squarefree strings in $\{a,b,c\}^*$ is not context-free.

Answer: Discussed in Class 17.

Problem 3: Language Classes.

(7.4 / 15) **Revision Opportunity!** Define a type of machine that recognizes the set of languages in the *Mystery Class* ellipse depicted below:



Your class should include infinitely many non-regular languages, but there should be infinitely many finite languages that are not included in your class. For full credit, your answer

should include a convincing argument why your class includes infinitely many non-regular languages and excludes infinitely many finite languages.

Answer: An example of a machine that would produce the *Mystery Class* is a DPDA with the restriction that q_0 and any state reachable from q_0 following only ϵ-transitions is excluded from F. We'll call this a *Empty-Rejecting-DPDA* (ERDPDA). This restriction means the empty string cannot be in the language described by any ERDPDA. There are infinitely many finite languages that do not include the empty string, but also infinitely many finite languages that do include the empty string. Similarly, for regular and non-regular context free languages.

The reason the limited-state PDA can accept all regular languages is that we can use the stack to keep track of extra states. We can construct a PDA that uses one state to encode the complete transition function for a DFA, with three additional states needed to set up the stack with a bottom symbol and to check the bottom of the stack. Here's how to construct the DPDA equivalent to DFA $D = (Q, \Sigma, \delta, q_0, F)$:

$P = (\{q_a, q_b, q_c, q_d\}, \Sigma, \Gamma = Q \cup \{\$\}, \delta_P, q_a, \{q_d\})$ where:

$\delta_P(q_a, \epsilon, \epsilon) = (q_b, \$)$ (push a \$ on the stack at the beginning)

$\delta_P(q_b, \epsilon, \epsilon) = (q_c, q_0)$ (push q_0, the label of the DFA's initial state, on the stack)

$\delta_P(q_c, \epsilon, \$) = (q_d, \epsilon)$ (move to accepting state if at end of input and bottom of stack)

(We have not included the permanent rejecting state. Any input in q_d leads to a permanent rejecting state.)

All the real work is done in state q_c:

If $\delta(q_i, a_i) = q_j$ then: $\delta_P(q_c, a_i, q_i) = (q_c, q_j)$

(stay in state q_c, the top of the stack represents the simulated DFA's state)

Problem 4: Broken Proofs.

Each of these "proofs" claims to prove a conjecture that is false. For each proof, identify the *first step* that is wrong, and briefly and clearly explain why. The explanation is more important than the step you identify.

a. (4.1 / 5) **False Conjecture:** The intersection of two context free languages is context free.

Claimed Proof.

1. We define the intersection of two languages
$$A \cap B = \{w | w \in A \wedge w \in B\}.$$
2. The language $X = A \cap B$ is a subset of the language A: $X \subseteq A$. This is true since every string in X must also be in A.

3. Since X is a subset of A, and A is context free, X is context free.

   Answer: Step 3: knowing X is a subset of a context-free language does not tell us anything about X. For example, $\{0,1\}^*$ is a context-free language (it is also a regular language), but the non-context-free language $\{ww|w \in \{0,1\}^*\}$ is a subset of $\{0,1\}^*$.

b. (4.0 / 7) **Revision Opportunity! False Conjecture:** All whole numbers are even.

Claimed Proof. We use induction on the numbers to prove the conjecture.

1. A number n is *even* if $2m = n$ for some integer m.
2. *Basis:* 0 is even. Select $m = 0$, then $2m = 0$.
3. *Induction:* Assume the conjecture holds for all $i < n$. We show that it holds for n.
4. $n = k + 2$ for some integer $k < n$.
5. By the induction hypothesis, $k = 2m$ for some integer m.
6. $n = k + 2 = (2m) + 2 = 2(m + 1)$. Thus, n is even.

   Answer: The problem is with step 5 (as it connects with step 4).

   The induction hypothesis is about *whole numbers*, but this step uses integers. It is true that $n = k + 2$ for some integer $k < n$, but the induction hypothesis only applies to whole numbers. It is not true that $n = k + 2$ for some whole number, since if $n = 1$ there is no non-negative number that satisfies this property.

   Many people identified issues that are not problems, but reflected a very rigid notion of induction proofs. For example, you may expect the induction hypothesis to be written like, *Assume the conjecture holds for all* $i \leq n$. *Show it holds for* $n + 1$. If n is an integer, this means the exact same thing as showing it holds for n, assuming it holds for all $i < n$. It is also not the case that all induction proofs have to increment by one. The requirement is that if we want to prove some property holds for all members of a given set, the generation method used in the induction has to produce all members of the set.

   The revision opportunity for this question is to demonstrate that you understand this by providing a *correct* induction proof of the following proposition:

   > All even numbers greater than zero can be expressed as the sum of two odd numbers.

   We prove by induction on the set of even numbers:

Basis. The proposition is true for the first even number greater than zero: 2 = 1 + 1.

Induction. The induction hypothesis is that all even numbers $0 < e \leq n$ can be expressed as the sum of two odd numbers. We show that the next greater even number, $n+2$, can be expressed as the sum of two odd numbers. By the induction hypothesis, $n = d_1 + d_2$ where d_1 and d_2 are odd numbers. Then, $n+2 = d_1 + d_2 + 2 = d1 + (d2+2)$. We know $d1$ is odd. Since $d2$ is odd, $(d2+2)$ is also odd. Hence, $n+2$ is also the sum of two odd numbers.

c.(6.0 / 8, *tricky*) **False Conjecture:** The language $TRIPLES$ is not regular.

$$TRIPLES = \{1^{3n} | n \geq 0\}$$

Claimed Proof. We use the pumping lemma for regular languages to obtain a contradiction.

1. Assume $TRIPLES$ is regular. Then, there exists some DFA M with pumping length p that recognizes $TRIPLES$.
2. Choose $s = 1^p$.
3. The pumping lemma requires that there is a way to divide s into $s = xyz$ where $|y| \geq 1$ and $|xy| \leq p$ and $xy^iz \in TRIPLES$ for all $i \geq 0$.
4. Since s is all 1s, we know y can contain only 1s.
5. Choose $y = 1$.
6. Choose $i = 2$.
7. Since xy^2z now has $p+1$ ones, it is not in the language $TRIPLES$.
8. Thus, we have a contradiction of the pumping lemma and $TRIPLES$ must not be regular.

Answer: The first problem is with step 2. To use the pumping lemma, we need to choose a string that is in the language. We don't know if 1^p is or is not in the $TRIPLES$. A better choice would be to start with 1^{3p} which we do know is in the language.

Step 5 is also broken since it picks a specific choice for y. For our proof to be correct, it needs to work for *all possible* choices for y.

d. (*Bonus*; don't work on this one until you finish the rest of the exam)

In the 2010 annual Latkes (fried potato cakes) v. Hamentash (triangular cookies) debate at MIT, Michael Sipser presented the proof described below (taken from *Faculty fling fake facts in food fight*, The Tech, 26 February 2010, with numbers added):

> Sipser wrapped things up for Team Hamentash with the HamenTheorem, which proves by contradiction that the hamentash is better than the latke. (1) First, the proof assumes latkes are best. (2) Then by obviousness, he claimed that hamentashen are better than nothing, and (3) by first assumption, claimed that nothing is better than latkes. (4) Therefore, Sipser argued that the HamenTheorem proved that hamentashen are better than latkes.

Since everyone knows latkes are better, the proof must be flawed. Explain the flaw in Sipser's proof.

> *Answer:* The best answer to this would be:
>
> > Nothing!
>
> There is nothing technically wrong with the proof, but there is a problem with the way Sipser uses the informal English word "Nothing". In step 2, it means "not having anything". In step 3, it means "no one thing".

Problem 5: Leaky PDAs.

Consider a new machine model known as a *LeakyPDA*. A LeakyPDA is similar to a deterministic PDA except that the stack has a limited depth. A LeakyPDA is specified by the 7-tuple $(Q, \Sigma, D, \Gamma, \delta, q_0, F)$ where $D \in \mathcal{N}$ is a natural number giving the maximum stack depth and the other elements have the same meaning as for a deterministic PDA. When the stack contains D elements, if a transition pushes another element on the stack, the bottom element of the stack is removed. That is, the result of following the transition rule,

$$\delta(q_i, a, \epsilon) \rightarrow (q_t, h_p)$$

starting with a stack $\gamma_0\gamma_1, \ldots, \gamma_{D-2}\gamma_{D-1}$ is the stack $h_p\gamma_0\gamma_1, \ldots, \gamma_{D-2}$.

(5.4 / 10) **Revision Opportunity!** Precisely describe the computing power of a LeakyPDA. For full credit, your answer should include a convincing proof supporting your answer.

> *Answer:* The computing power of a LeakyPDA is precisely that of a DFA: it can recognize the class of Regular Languages. Since the stack is finite, there is a finite number of possible stack configurations: $|\Gamma|^{D+1}$ (the exponent is $D+1$ since we need to consider non-full stacks also). The configuration of a LeakyPDA is its current state (an element of Q) and current stack. Thus, we can simulate a LeakyPDA with a DFA where there are $|Q| * |\Gamma|^{D+1}$ states to represent all possible configurations of the LeakyPDA.

Problem 6: Language Class Differences.

a. (7.9 / 10) If A and B are regular languages, is $A - B$ always a regular language? Recall that the difference of two sets is defined as

$$A - B = \{s : s \in A \wedge x \notin B\}$$

(For full credit, your answer should include a convincing proof supporting your answer.)

Answer: One way to prove this is to use the property that $A - B = A \cap \bar{B}$. Since we already proved the regular languages are closed under intersection and complement, this proves that $A - B$ is regular.

Another way to prove it would be to show how to construct a DFA that recognizes $A - B$:

> Since A and B are regular languages, there exists DFA $M_A = (Q_A, \Sigma, \delta_A, q_{0A}, F_A)$ and $M_B = (Q_B, \Sigma, \delta_B, q_{0B}, F_B)$ that recognize A and B respectively.
>
> We can produce a machine M_{A-B} that recognizes the language $A - B$ by simulating both machines and accepting when M_A would accept and M_B would reject.
>
> $M_{A-B} = (Q_A \times Q_B, \Sigma, \delta_{AB}, (q_{0A}, q_{0B}), F_{AB})$ where:
>
> $\delta_{AB}(q_a, q_b, a) = (\delta_A(q_a, a), \delta_B(q_b, a))$
> $F_{AB} = \{(q_a, q_b) | q_a \in F_A \wedge q_b \notin F_B\}$

b. (5.3 / 10) **Revision Opportunity!** If A and B are context-free languages, is $A - B$ always a context-free language? (For full credit, your answer should include a convincing proof supporting your answer.)

Answer: No. We prove $A - B$ is not necessarily a context-free language by identifying context-free languages A and B whose difference is not a context-free language.

One set of choices is:

- $A = \{a^n b^n c^m | n \geq 0, m \geq 0\}$ (we know this is context-free since we could recognize it with a PDA that pushes to count the as, pops to count the bs, and then goes to a state that accepts any number of cs).
- $B = \{a^j b^k c^k | j \geq 0, k \geq 0\}$ (we know this is context-free since we could recognize it with a PDA that reads any number of as, then pushes to count the bs, and pops to count the cs).
- $A - B = \{a^i b^i c^t | i \geq 0, i \neq t\}$ This language results, since when we remove the strings in B from A, we remove all the strings where $n = m$ since those strings have the same number of bs and cs. But, this language is not context-free! We can see that since it requires counting the number of as, bs, and cs (to know $t \neq i$).

An easier way is:

- $A = \Sigma^*$ (we know this is context-free since it is regular and all regular languages are context-free)
- $B = \{w | w \neq xx, x \in \Sigma^*\}$ (we know this is context-free from Class 9)

- $A - B = \{w | w = xx, x \in \Sigma^*\}$ But, we know this is not context-free from Class 9.