Harvard University
Computer Science 121
Problem Set 3

Due Friday, October 8, 2010 at 1:20 PM.
Submit a single PDF (lastname+ps3.pdf) of your solutions to cs121+ps3@seas.harvard.edu
Late problem sets may be turned in until Monday, October 11, 2010 at 1:20 PM with a 20% penalty.
See syllabus for collaboration policy.

Name

Problem set by !!! Your Name Here !!!

with collaborators !!! Collaborators' names here !!!!

Notes

When drawing your parse tree you may want to draw by hand or use a graphics program and import like we did for ps1.
When not stated elsewhere assume the alphabet $\Sigma = \{a, b\}$.
Use the provided template for your answers. Custom formatted solutions are difficult to grade.
Read instructions carefully. DRAW means exactly that.
We denote the reverse (defined naturally) of a string w to be w^R.

PROBLEM 1 (2+2+2+2 points)

Which of the following languages are regular? Support each of your answers with a proof.

(A) $\{a^ib^ja^jb^i\colon i, j \geq 0\}$.
(B) $\{a^{i^2}\colon i \geq 0\}$.
(C) $\{wxw^R : w, x \in \Sigma^*\}$
(D) $\{ww^R\}$.

(A) This language is not regular. Assume it is regular and let p be the pumping length. Consider the string $a^pb^pa^pb^p$. By the pumping lemma we know strings of the form $a^{p+y|k|}b^pa^pb^p$ $\forall k \geq 0$ must be in the language. From the lemma $|y| > 0$, so for any such $k \geq 1$ the string is not in the language since the first sets of a's and b's are unbalanced. $\square$
(B) This language is not regular. Assume it is regular and let p be the pumping length. Consider the string a^{p^2}. By the pumping lemma we know strings of the form $a^{p^2+k|y|}$ $\forall k \geq 0$ must be in the language. However, these must all be perfect squares if the language is regular. We can verify this isn't possible:

$$p^2 + ky = (p+1)^2$$

$$p^2 + (k+1)y = (p+2)^2$$

for any $k \geq 0$. Doing some math:

$$ky = 2p + 1;\ k = (2p + 1)/y$$

$$(k + 1)y = 4p + 4;\ k = (4p + 3)/y$$

Which has no solution for $p \geq 0$. Thus we have a contradiction—we can't fit the growth of perfect squares with a regular language. $\square$

(C) This language is regular. Any string matches this format. Consider $w = \epsilon$, then $w^R = \epsilon$. This leaves the pattern $\epsilon x \epsilon$ where $x \in \Sigma^*$, which is any string. We can construct a simple DFA for this consisting of one state, the start state, which is also an accept state. It has a self loop for any character seen. Since we have a DFA recognizing the language we know it's regular. $\square$

(D) This language is not regular. Assume it is regular and let p be the pumping length. Consider the string $a^p bba^p$, which is clearly in the language. By the pumping lemma we know strings of the form $a^p a^{k|y|} bba^p\ \forall k \geq 0$ must be in the language, too. However for any $k \geq 1$ this results in a string which isn't reversible, since our string only contains two b's w must contain only one. Thus this string isn't in the language, a contradiction. $\square$

PROBLEM 2 (4+1 points)

(A) Prove the following strong form of the pumping lemma: there is a number p where, if s is any string in L of length at least p, then s may be written as $s = xyz$ such that

1. for each $i \geq 0$, $xy^i z \in L$,
2. $|y| > 0$, and
3. $|xy| \leq p$.

(B) The pumping lemma states every regular language L has these properties. Is the converse true? That is, if a language has these properties is it regular? Describe why or why not briefly.

(A) See the proof in Sipser.

(B) No. The converse of a theorem is not generally true and a language which is not regular may satisfy these properties, too.

PROBLEM 3 (4 points)

Show that a DFA with n states accepts an infinite language if and only if it accepts some string of length at least n and less than $2n$.

(A) We have to prove two statements: (1) if a DFA with n states accepts an infinite language a string of length x, $n \leq x < 2n$ and (2) if a DFA with n states accepts some string of length x, $n \leq x < 2n$ it accepts an infinite language..

(1) If a DFA accepts strings of arbitrary length (required to accept an infinite language over a finite alphabet) then it must have a cycle. See problem set 1. Additionally, this cycle must have an accept state reachable from it, otherwise we can not revisit past states and still accept a string properly. Let $m < n$ be the size of the cycle and k be the number of states from the cycle to an

accept state (any one will do). This particular cycle we're considering (and we only need consider one) must be reached in $r < n$ states. Once reached, it accepts (at least) strings with lengths $\forall i \geq 1, r + m^i + k$, since we can have a constructed string loop through the cycle i times before reaching the final state. Since $r + m + k \leq n$ we have $\exists i > 1 n \leq r + m^i + r < 2n$.

(2) If a DFA accepts a string of length greater than the number of states it has then by the pigeonhole principle at least one state must have been visited twice and, by definition, the DFA must have a cycle. Further, since the string was accepted there must be an accept state reachable from the cycle. Since we have a cycle and an accept state reachable from the cycle we can accept strings greater than any finite length (thus generating an infinite language) by iterating around the cycle a sufficient number of times before reaching the accept state.

PROBLEM 4 (2+2 points)

(A) Prove the set of finite languages over a fixed alphabet is countable.

(B) Prove that if A is an uncountable set and B is a countable set, then $A - B$ is uncountable.

(A) (Cool version) We know from class that Σ^* is countable. This allows us to map any string into a number, and any finite language into a set of numbers. Since two finite languages must differ on at least one string these sets are unique. We can map these numbers to primes such that a number i maps to the ith prime. Multiplying these primes uniquely numbers each language since prime factorizations are unique.

(Regular version) Languages with $d = n$, for any fixed n, strings comprise a finite set. The set of all finite languages is the union of these sets from zero to infinity, where sets are labelled with valid lengths 0, 1, 2 .... This is the set of natural numbers, so we have a union of finite sets over a countable space, which results in a countable set.

(B) Assume the statement is false. Then $A - B$ is countable. Also note that $A \cap B$is countable since it's a subset of a countable set (we can use the same function mapping the subset to the natural numbers as we use for the superset). However, $A = (A - B) \cup (A \cap B)$, which is the union of two countable sets and therefore a countable set. This is a contradiction, so our assumption must be wrong and A - B is uncountable.

PROBLEM 5 (4+2+2 points)

A satisfied boolean expression is a string over the alphabet $\Sigma = \{0, 1, (,), \neg, \wedge, \vee\}$ representing a boolean expression that evaluates to 1, where $\neg$ is the *not* operator, $\wedge$ is the *and* operator, and $\vee$ is the *or* operator (see Sipser p.14). For instance, 1, $(1 \wedge (\neg 0))$, and $(\neg(\neg 1))$ are satisfied boolean expressions.

The following are examples of strings that are *not* satisfied boolean expressions:

)(1
$(0\neg)$
$\wedge(0 \vee 1)$
$(0 \wedge 1)$
$(\neg(1 \vee 0))$

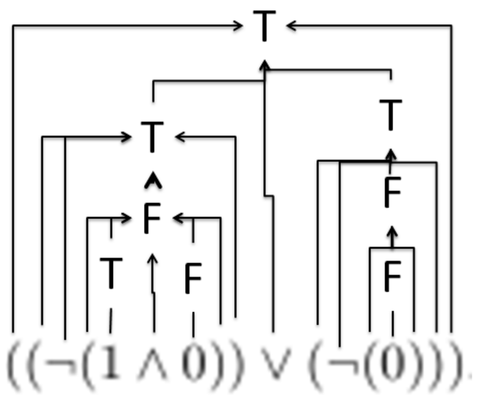

Figure 1: Parse tree for $((\neg(1 \wedge 0)) \vee (\neg(0)))$.

The first three examples are strings that aren't valid expressions and simply don't make sense. The last two are well-formed expressions, but they evaluate to 0, rather than to 1.

(A) Write a context-free grammar that generates the language of satisfied boolean expressions. You may assume that expressions must be completely parenthesized. For instance, your grammar need not generate $\neg(0 \wedge \neg 0 \wedge 1)$ but should generate its equivalent, $(\neg(0 \wedge ((\neg 0) \wedge 1)))$.

(B) In several sentences, explain why the language of satisfied boolean expressions is not regular.

(C) Draw the parse tree in your grammar for the expression $((\neg(1 \wedge 0)) \vee (\neg(0)))$.

(A) Let the start symbol be T. We have the following rules (with the remainder of the grammar defined naturally):

$$T \to 1|(T)|(\neg F)|(T \wedge T)|(T \vee T)|(T \vee F)|(F \vee T)$$
$$F \to 0|(F)|(\neg T)|(F \wedge F)|(F \vee F)|(F \wedge T)|(T \wedge F)$$

(B) Assume it is regular with pumping length p and consider the string $(^p1)^p$, which is in the language but when pumped would create unmatched parentheses.
(C) See figure.

PROBLEM 6 (4+4+2 points)

A context-free grammar G is **ambiguous** if there exists a string $w \in L(G)$ with two distinct leftmost derivations in G. (That is, two distinct derivations in which the leftmost nonterminal is replaced at each step).

(A) Show that the context-free grammar $G = (V, \Sigma, R, S)$, where $V = \{S, T\}$, $\Sigma = \{a, b\}$, and

$$R = \{S \to aT,\ T \to S,\ T \to Tab,\ T \to a\}$$

is ambiguous.

(B) A language L is **inherently ambiguous** if all context-free grammars G such that $L = L(G)$ are ambiguous. Show that if L is a regular language, then L is **not** inherently ambiguous. (*Hint*: think about regular grammars).

(C) Show that $L(G)$, where G is the context-free grammar in Part (A), is **not** inherently ambiguous.

(A) We can show it's ambiguous by deriving the same string two different ways. Consider aaaab:

$$(1) S \rightarrow aT$$

$$aT \rightarrow aTab$$

$$aTab \rightarrow aaTab$$

$$aaTab \rightarrow aaaab$$

$$(2) S \rightarrow aT$$

$$aT \rightarrow aaT$$

$$aaT \rightarrow aaTab$$

$$aaTab \rightarrow aaaab$$

(B) We can build a grammar by examining the DFA for a regular language. For every state $q \in Q$ we build a rule in our grammar $q \rightarrow \Sigma_{q,q'} q'$ where $q' \in Q$ is reachable from q by $\Sigma_{q,q'}$. For accepting states $q^* \in \Sigma$ we also add the rule $q^* \rightarrow \epsilon$, and we have our start symbol S point to the start state q_0. Having constructed a grammar from the DFA we now need to prove this regular is not ambiguous. Since we constructed it from a DFA, however, there is one transition only for each character from each state. The same string must be created only one way then, otherwise it would violate the rules of the DFA. As such, this generated grammar is not ambiguous.
(C) Per part B it's sufficient to demonstrate the language is regular. We can do this by writing the regular expression for the language: $aaa^*(ab)^*$.

PROBLEM 7 (Challenge!! 1 points)

Show that for any subset L of $\{a\}^*$, the language L^* is regular.

(A) This problem is really asking you to demonstrate that any infinite sequence of integers with a sufficient number of them linearly independent form a basis for all natural numbers greater than a finite value. If there is only a finite set of linearly independent values than the language is regular since each basis vector may be encoded in the DFA along with however many multiplications are needed to arrive at other elements (if a finite number) or an infinite loop permitted (if that vector generates an infinite number). Proving the former case is not trivial and unlikely to be of interest in isolation, but we suggest students take a look at the literature of basis for the natural numbers if interested. There's a particularly interesting paper by Erdös entitled "Minimal Asymptotic Bases for the Natural Numbers" which is closely related.