CS 381	**Introduction to Theory of Computing**	Summer 2002
Prelim 2		June 21, 2002

10 points per problem.

1. For each of the following languages R, find a set of strings S_L that contains exactly one string from every equivalence class of $\equiv_R$.

a. $R = \{x \in \{a,b\}^* \mid \#a(x) = 2\}$

Solution: Either by inspection or by creating a minimal DFA to accept R, we see that the only thing that distinguishes two strings is the number of a's. One possibility for S_L is $\{\epsilon, a, aa, aaa\}$. The most common error was to omit aaa.

b. $R = \{x \in \{a,b\}^* \mid \#a(x) \text{ is even}, \#b(x) \equiv 0 \mod 3\}$

Solution: Again by using inspection or by creating a minimal DFA (e.g., using the product construction), we see that there are 6 equivalence classes corresponding to the ordered pairs (m,n), where m may be 0 or 1 ($\#a(x) \mod 2$), and n may be 0, 1, or 2 ($\#b(x) \mod 3$). Thus, one choice for S_L is $\{\epsilon, a, b, ab, bb, abb\}$.

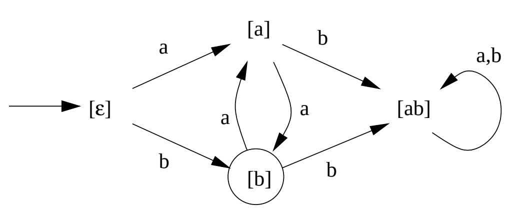
c. $R = \{a^n b a^n \mid n \geq 0\}$

Solution: If $j \neq k$, then $a^j \not\equiv_L a^k$ since $a^j b a^j \in L$ but $a^k b a^j \notin L$. Likewise, $a^j b \not\equiv a^k b$ if $j \neq k$, and neither of these is equivalent to any a^m. But $a^j ba \equiv_L a^{j-1}b$ for $j \geq 1$. Any string of the form $a^n b a^m$ with $n \geq m$ is represented by one of the strings just given, and any string not of this pattern must have 0 or 2 b's or more a's following the b than preceding it, in which case no string can be concatenated to yield a string in L. Thus, this last set of strings is represented by bb, and one possibility for S_L is $A \cup B \cup \{bb\}$, where $A = \cup_{n \geq 0}\{a^n\}$ and $B = \cup_{n \geq 0}\{a^n b\}$.

2. Let $R \subset \Sigma^*$ be a language so that the corresponding equivalence relation $\equiv_R$ has a total of 4 equivalence classes: $[\epsilon]$, $[a]$, $[b]$, and $[ab]$. Suppose also that $aa \in [b]$, $ba \in [a]$, $bb \in [ab]$, $aba \in [ab]$, $abb \in [ab]$.

a. Suppose that $R = [b]$. Draw the transition diagram for a DFA, M, with $L(M) = R$.

Solution:



b. True or false: In part (a), M is isomorphic to $M/\approx$. Explain.

Solution: True: Since $\equiv_R$ has only finitely many equivalence classes, it is a Myhill-Nerode relation, and from a theorem from class, it is the coarsest Myhill-Nerode relation. Hence the corresponding DFA is a DFA with the minimal number of states, hence is isomorphic to $M/\approx$.

c. Suppose $[R] = [a] \cup [b]$. Is this consistent with the restrictions on $\equiv_R$ given above? Explain.

Solution: This is not consistent since if $R = [a] \cup [b]$, then both $[a]$ and $[b]$ would be final states in the DFA, in which case $[a] \approx [b]$, and hence the DFA would not be minimal, contradicting the minimality of the DFA obtained from $\equiv_R$.

d. Suppose $[R] = [b] \cup [ab]$. Is this consistent with the restrictions on $\equiv_R$ given above? Explain.

Solution: This is consistent since in this case the 4 equivalence classes are still distinct. I.e., for any two states p and q, there is a string x so that starting from p and reading x will lead to a final state while starting from q will lead to a nonfinal state or vice versa. Thus the DFA is minimal, so this is consistent.

3. a. Give a CFG for $L = L(a^*bb^*aa^*b(a+b)^*)$. Give justification that your grammar generates exactly L.

Solution: One solution for this problem is to use the fact that L is regular together with the homework problem about converting a regular language to a right-linear grammar. This yields the following grammar:

$$
\begin{aligned}
S &\to aS \mid bA \\
A &\to bA \mid aB \\
B &\to aB \mid bC \\
C &\to aC \mid bC \mid \epsilon
\end{aligned}
$$

An alternative is to have nonterminals generating $L(a^*)$, $L(b^*)$, and $L((a+b)^*)$, then to combine these to get L. This gives

$$
\begin{aligned}
S &\to AbBaAbC \\
A &\to aA \mid \epsilon \\
B &\to bB \mid \epsilon \\
C &\to aC \mid bC \mid \epsilon
\end{aligned}
$$

Either a very simple induction or the previously mentioned homework problem show that A generates $L(a^*)$, B generates $L(b^*)$, and C generates $L((a+b)^*)$, so S generates L.

b. Give an NPDA for $L = \{a^m b^n \mid m \neq n\}$. Specify the states, stack symbols and transitions. You must provide comments to explain how your NPDA works, but you don't have to give formal proof that it accepts exactly L.

Solution: There were two common approaches to this problem: either to write out the transitions for an NPDA directly, or to make a grammar for the language, convert it to Greibach normal form, then use this to make a one-state NPDA. In the first case, one common error was not to change state after inputting a b to prevent strings of the form aba from being accepted. Another common error was to try to define acceptance by non-empty stack. An NPDA can accept either by final state or empty stack, but to accept by non-empty stack you have to do this with a transition to a final state or by emptying the stack.

Here is one possible way to construct the NPDA directly: The general idea is to keep track of $\#a - \#b$ by pushing A when an a comes in and either popping an A or pushing a B when a b comes in. Let $Q = \{s, q, f\}$, $\Gamma = \{A, B\}$, $F = \{f\}$, and let s be the start state. The following transitions define δ:

The first 2 rules push an A for every a.
$((s, a, \perp), (s, A\perp))$
$((s, a, A), (s, AA))$

The next rules switch to state q when a b comes in, and either pop an A or push a B.
$((s, b, \perp), (q, B\perp))$
$((s, b, A), (q, \epsilon))$

The next rules keep popping A's or pushing B's while b's come in.
$((q, b, \perp), (q, B\perp))$
$((q, b, A), (q, \epsilon))$
$((q, b, B), (q, BB))$

The final set of rules jump to the final state if the stack contains an A or a B (in the first case there are more a's than b's, and vice versa in the second case). Here we also allow a transition from state s if there is an A on the stack. In this case only a's have been read so far.
$((q, \epsilon, A), (f, \epsilon))$
$((q, \epsilon, B), (f, \epsilon))$
$((s, \epsilon, A), (f, \epsilon))$

Note that there are no transitions from state q with input symbol a, so any accepted string must have the form a^*b^*. Also, there is no transition out of state f, so the transition to f must occur when the string ends in order for it to be accepted. This insures that only strings with number of a's not equal to number of b's will be accepted.

For the approach using the Greibach normal form, a simple grammar to generate L is

$$S \rightarrow aSb \mid A \mid B$$
$$A \rightarrow aA \mid a$$
$$B \rightarrow bB \mid b$$

This can be converted to GNF by adding a new new nonterminal $C \to b$, and replacing the b in the first rule with C. This grammar then converts directly to a one-state NPDA as shown in class.

4. For each of the following languages, identify it as regular (R), context-free but not regular (C), or not context-free (N). All superscripts are assumed to be nonnegative integers. You may give answers only.

a. $\{a^i b^j c^k d^l e^m \mid i = j + k, l = m\}$

Solution: This is context-free. To make an NPDA, first push a's, then when b comes in, change states and pop a's. When c comes in, change states and pop a's. If the stack has $\perp$ when d comes in, change states again and push d's. Finally, when an e comes in, change states and pop d's. Accept only if the stack is $\perp$ when the last e comes in. All the state changes are used to keep track of the form $a^*b^*c^*d^*e^*$.

b. $\{a^i b^j c^k d^l e^m \mid i = m, j = k + l\}$

Solution: This is context-free. To make an NPDA, use an approach like that in part (a), but first push a's, then push b's, then pop b's while reading c and d, then pop a's while reading e.

c. $\{a^i b^j c^k d^l e^m \mid i = l, j = k + m\}$

Solution: This is not context-free. After taking the intersection with the regular set $L(a^*b^*d^*e^*)$, we get the set $\{a^m b^n d^m e^m \mid m, n \geq 0\}$, which is not context-free as shown in class.

d. $\{a^i b^j c^k d^l e^m \mid i + j + k \geq 100, l + m \geq 10\}$

Solution: This is a regular set since it can be written as the concatenation of $L(a^*b^*c^*) \cap \{x \in \{a, b, c\}^* \mid |x| \geq 100\}$ and $L(d^*e^*) \cap \{x \in \{d, e\}^* \mid |x| \geq 10\}$.

e. $\{a^i b^j c^k d^l e^m \mid i \geq j \geq k, l + m \geq 10\}$

Solution: This is not regular since it does not satisfy the pumping lemma for CFL's.

5. a. Prove that $A = \{a^n b^n c^m \mid m \leq 2n\}$ is not a CFL.

Solution: We prove this by playing the demon game. Let $k \geq 1$ and choose $z = a^k b^k c^{2k}$. Suppose $z = uvwxy$ with $vx \neq \epsilon$ and $|vwx| \leq k$. We base a winning strategy to find $i \geq 0$ so that $uv^i wx^i y \notin A$ on the possibilities for v and x. First note that if v or x has more than one type of letter, then uv^2wx^2y will not have the form $a^*b^*c^*$, hence will not be in A. So we may assume that v contains only one type of letter and likewise for x. If neither v nor x contains a c, then we choose $i = 0$, in which case we get a string $a^j b^l c^{2k}$ and even if $j = l$, we must still have $2k > 2l$, so this string is not in A. If exactly one of v or x consists of c's, then the other must contain only a's or only b's or be empty, so choosing $i = 0$ will either unbalance the number of a's and b's or else produce more than the allowable number of c's. Finally if v and x both contain c, we can choose $i = 2$ to produce a string with more than the allowable number of c's. Hence we have a winning strategy, so A is not CFL.

b. Give an example of 2 languages A_1 and A_2 so that neither is a CFL but $A_1 \cap A_2$ is a CFL with infinitely many strings. You must show that A_1 and A_2 are not CFL's and that $A_1 \cap A_2$ is a CFL and has infinitely many strings.

Solution: There are many possible solutions, most of which involve using letters that appear in only one of the two languages. One solution is to let $A_1 = \{a^n b^n c^m \mid m \leq 2n\}$ and $A_2 = \{a^n b^n d^m \mid m \leq 2n\}$. By part (a), both of these are not CFL's, but their intersection is $\{a^n b^n \mid n \geq 0\}$, which is a CFL as shown in class, and which clearly has infinitely many strings.