CS5003 **Foundations of C.S.** Spring, 2023

Final Exam

PRINT NAME: _____

SIGN: _____

You may use your textbook or two pages of notes.

Do both problems 1 and 2. From the remainder do your choice of 4 problems. Write your answers clearly and neatly. Use the back of the pages if necessary. The last page is blank.

1. Let L be the language on $\{a, b\}$ with recursive definition

BASIS: $aba \in L$, and $babab \in L$

RECURSIVE STEP: If $u, v \in L$ then $uvu \in L$.

CLOSURE: Every element in L can be obtained from the basis after a finite number of applications of the recursive step.

Prove by induction that every string in L is a palindrome.

♣ This is by induction on the number of recursive steps applied. Specifically, we will show that, for all $n \geq 0$, every element in L_n is a palindrome.

Base Case: L_0 consists of *aba* and *babab*, both palindromes.

Inductive Step. Let $n \geq 0$ be given and suppose every element of L_n is a palindrome.

Those elements of L_{n+1} which are in L_n are palindromes by the inductive hypothesis.

Let $w \in L_{n+1}$ and $w \notin L_n$. Then w is obtained from applying the rule to two elements u and v of L_n. By the inductive hypothesis, both u and v are palindromes, so $u^R = u$ and $v^R = v$. The element w is given by $w = uvu$, and the reverse satisfies $w^R = u^R v^R u^R = uvu$, so w is also a palindrome. Thus all elements of L_{n+1} are palindromes, as required.

Therefore, all elements of L are palindromes by induction.

Note 1: uvu is not a "palindrome in L". uvu is a concatenation of strings in the recursive construction. If $u = abb$ and $v = bba$, then $uvu = abbbbaabb$ which is not a palindrome.

Note 2: You cannot do a reasonable induction . . . unless you know what you are inducting on! ♣

2. Let N be the language of the regular expression $(a \cup b)^*((cb)^2 \cup a^*)(b \cup c)^*$ and M the language of the regular expression $(a \cup b)^*(b \cup c)^*$.

Prove that $N = M$ using the double inclusion method.

♣ First we show $N \subseteq M$. Let $w \in N$, so $w = xyz$ where x matches $(a \cup b)^*$, y matches $(cb)^2 \cup a^*$ and z matches $(b \cup c)^*$.

If $y = (cb)^2$, then yz matches $(b \cup c)^*$ and and $w = x(yz)$ matches $(a \cup b)^*(b \cup c)^*$, so $w \in M$.

On the other hand, if y matches a^*, then xy matches $(a \cup b)^*$ and $w = (xy)z$ matches $(a \cup b)^*(b \cup c)^*$ and $w \in M$ in this case as well.

So, either way, $w \in M$ and $N \subseteq M$.

Now we show $M \subseteq N$. Let $w' \in M$, so $w' = uv$, with u matching $(a \cup b)^*$ and v matching $(b \cup c)^*$. Note that λ matches $((cb)^2 \cup a^*)$, since λ matches a^*. So $w' = xy = x\lambda y$ matches $(a \cup b)^*((cb)^2 \cup a^*)(b \cup c)^*$, so $w' \in N$, and we have $M \subseteq N$.

This proves $N = M$.

Note 1: The double inclusion method requires you to show that first $N \subseteq M$, by let letting $w \in N$ and showing $w \in M$, and then the reverse. If you are not doing that, it is not the double inclusion method.

Note 2: N and M are sets. A regular expression is *not* a set, so writing $ab \in (ab \cup ba)^*$ or $ab = (ab \cup ba)^*$ look correct but are incorrect and vague. If you insist on sets, convert $(a \cup b)^*((cb)^2 \cup a^*)(b \cup c)^*$ to $(\{a, b\}^*(\{cb\}^2 \cup \{a\}^*)\{b, c\}^*$ which is a set, but that usually causes more work. ♣

3. a) Find a regular expression OR define a DFA for the set of strings on $\Sigma = \{a, b, c\}$ which contain both $babab$ and b^3 as substrings.

In either case, your design should be clear or explained.

♣ At this point, you should have anticipated that the DFA for this is much harder, and done the regular expression.

The required strings can be separated in either order or jammed together in either order:
$(a \cup b \cup c)^*[(babab(a \cup b \cup c)^*bbb) \cup (bbb(a \cup b \cup c)^*babab) \cup bababbb \cup bbbabab](a \cup b \cup c)^*$

A DFA requires, at a minimum, 15 states, and the product construction takes 24. The states you need are S, N_b, N_{bb}, N_{ba}, N_{bab}, N_{baba} for *neither string in prefix, but partial progress*, A, A_b A_{bb} for *prefixes contains the alternating string and some partial progress toward the cube*, C, C_b, C_{ba}, C_{bab}, C_{baba} for *prefixes having the cube with partial progress toward the alternating string*, and H, the only final state, for *hurrah we're done.*

The final DFA can be drawn without edge crossings, which is at least something. ♣

b) Find a regular expression OR a regular grammar for the set of strings on $\{a, b, c\}$ which do not contain a^4 as a substring.

In either case, your design should be clear or explained.

♣ If any a's then the power has to be restricted and separated non-trivially from any other a's by b's or c's, and the possibility of initial and final b's and c must be allowed as well. The alternative is no a's at all.

$[(b \cup c)^*((a \cup a^2 \cup a^3)(b \cup c)^+)^*(a \cup a^2 \cup a^3)(b \cup c)^*] \cup (b \cup c)^*$

The regular grammar and regular expression seem about equally easy to write down.

The roles of the variables in the regular grammar take note of how many trailing a's are in the prefix.

$$\begin{array}{rcl} G: S & \rightarrow & aP \mid bS \mid cS \mid a \mid b \mid c \mid \lambda \\ P & \rightarrow & aQ \mid bS \mid cS \mid a \mid b \mid c \\ Q & \rightarrow & aR \mid bS \mid cS \mid a \mid b \mid c \\ R & \rightarrow & bS \mid cS \mid b \mid c \end{array}$$

and requires only 4 variables. ♣

4. Convert the grammar G below to an equivalent grammar below which contains no useless symbols.

$$\begin{aligned} G: S &\to aPQRV \mid VV \\ P &\to a \mid b^2 \\ Q &\to TU \mid aT \mid bU \\ R &\to b \mid a^2 \\ T &\to QU \mid aU \mid bQ \\ U &\to TQ \mid aQ \mid bT \\ V &\to a^2 \mid b^2 \end{aligned}$$

♣ $\text{TERM}_0 = \{P, R, V\}$, $\text{TERM} = \text{TERM}_1 = \{P, R, V, S\}$, So G is equivalent to

$$\begin{aligned} G_1: S &\to VV \\ P &\to a \mid b^2 \\ R &\to b \mid a^2 \\ V &\to a^2 \mid b^2 \end{aligned}$$

$\text{REACH}_0 = \{S\}$, $\text{REACH} = \text{REACH}_1 = \{S, V\}$, So G' is equivalent to

$$\begin{aligned} G_2: S &\to VV \\ V &\to a^2 \mid b^2 \end{aligned}$$

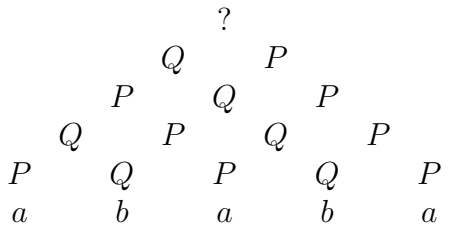
So $L(G) = \{aaaa, aabb, bbaa, bbbb\}$. ♣

5. Consider the Chomsky grammar

$$\begin{array}{rcl} G: S & \to & PQ \\ P & \to & QP \mid a \\ Q & \to & PQ \mid b \end{array}$$

a) Use the CYK algorithm to decide if *ababa* is in the language.

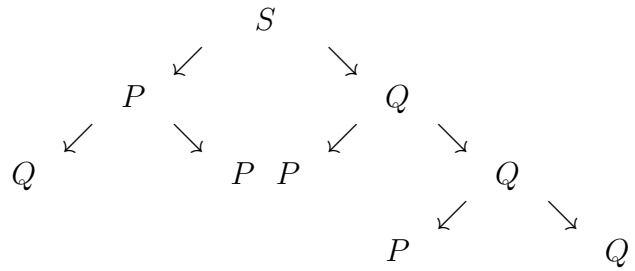
Easy to fill, we ignore S until the very top.



The only pairs for the top position are PP and QP, neither of which are S rules, so $ababa \notin L(G)$.

b) Exhibit a derivation tree corresponding to a left-derivation for any string of length 5 in the language.

♣ If you were working to make your's a *left* derivation, there was no need. There is no way to detect the type a derivation, (leftmost, rightmost, in between) from the tree, that is one one the reasons derivations trees are used.



This corresponds to *baaab*.

You should be able to show that $L(G)$ is the set of all strings on $\{a, b\}$ which end in ab. Hint: Change that to a statement about sentential forms.

Using that, we can conclude that the language has a regular grammar, in fact, and easy one. If you compare that to the grammar you are getting from the next problem, you see that the Greibach grammar you get from the conversion procedure may not be the easiest Greibach grammar for that language at all. ♣

6. Consider the Chomsky grammar

$$\begin{array}{rcl} G: S & \to & PQ \\ P & \to & QP \mid a \\ Q & \to & PQ \mid b \end{array}$$

Convert to Greibach normal form. Show all steps in the process.

♣ Really? Well, there are only three variables and five rules, how bad can it be? Substitute first-variable forward on the ordering $S < P < Q$:

$$\begin{array}{rcl} G_1: S & \to & PQ \\ P & \to & QP \mid a \\ Q & \to & QPQ \mid aQ \mid b \end{array}$$

Removing left recursion when necessary, only Q needed modification:

$$\begin{array}{rcl} G_2: S & \to & PQ \\ P & \to & QP \mid a \\ Q & \to & aQ \mid b \mid aQR \mid bR \\ R & \to & PQ \mid PQR \end{array}$$

Now Q is Greibach compliant and we have to first-variable substitute backwards,

$$\begin{array}{rcl} G_3: S & \to & aQPQ \mid bPQ \mid aQRPQ \mid bRPQ \mid aQ \\ P & \to & aQP \mid bP \mid aQRP \mid bRP \mid a \\ Q & \to & aQ \mid b \mid aQR \mid bR \\ R & \to & PQ \mid PQR \end{array}$$

fixing the new recursive variable, R, last:

$$\begin{array}{rcl} G_3: S & \to & aQPQ \mid bPQ \mid aQRPQ \mid bRPQ \mid aQ \\ P & \to & aQP \mid bP \mid aQRP \mid bRP \mid a \\ Q & \to & aQ \mid b \mid aQR \mid bR \\ R & \to & aQPQ \mid bPQ \mid aQRPQ \mid bRPQ \mid \\ & & aQ \mid aQPQR \mid bPQR \mid aQRPQR \mid bRPQR \mid aQR \end{array}$$

And that is that.

Note: You cannot pick a rule and replace an initially variable with *some* of it's consequences, that alters the language.

Note: You cannot pick and alter rules corresponding to whole sets of variables, particular which refer to one another. That may alter the language. The established produce works on variables one at at time in a specified order to avoid this problem. ♣

7. Let C be a grammar Chomsky-Normal form and let R be a regular grammar. For each of the following, label the statement as TRUE or FALSE, and provide a short explanation.

__ If $L(C) \subseteq L(R)$, then $L(C)$ is regular.

♣ F: Σ^* is regular, so this inclusion implies nothing about the regularity of $L(C)$, which may be a non-regular language, context free language. ♣

__ If $L(R) \subseteq L(C)$, then $L(R)$ is context free.

♣ T: For the simple reason that every regular language is context free, since it can be expressed by a regular grammar. ♣

__ $\{w \in \Sigma^* \mid w \notin L(R)\}$ is regular.

♣ T: This was one of our surprises at the end. If you switch final states on a DFA, then that action exactly switches the collection of accepted and non-accepted strings. (No so for non-deterministic machines!) ♣

__ The language of every regular grammar is recognized by a NDFA-λ.

♣ T: Regular grammars generate regular languages, and those are also the ones recognizing automata, specifically NDFA-λ's. ♣

__ $(L(C)L(R) \cup \{a\})^*$ is countably infinite.

♣ T: Since the set of strings Σ^* is countable, every language is countable.

And, as we've pointed out a thousand times, it is the *collection of all languages on* Σ which is uncountable,

$$\mathcal{L} = \{L \mid L \subseteq \Sigma^*\},$$

not the individual languages, each of which is an element of the uncountable set $\mathcal{L}$. ♣

Just an extra page.