CS5003 Final Exam

Foundations of C.S.

Summer, 2023
Good Luck!

You may use your textbook or anything appearing on our canvas pages. You may not collaborate or use other sources..

Do any five of the following eight problems. Write your answers clearly and neatly. Upload a pdf of your answers. It should be a single file, but may have several pages.

---

1. Let $\Sigma = \{a, b\}$ and let
$X = \{w \in \Sigma^* \mid n_a(w) + n_b(w) = 2k; k \in \mathbb{N}\}$, and
$Y = \{uv \mid u \in \Sigma^*; v \in \Sigma^*; \text{length}(u) = \text{length}(v)\}$ define two languages.
Prove using the double inclusion method that $X = Y$.
[It is important the your argument is complete, uses the double inclusion method, and also that you not make it any more complicated than necessary.]

♣

First we show $X \subseteq Y$. Let w be any element of X. So $n_a(w) + n_b(w) = 2k$. Since $\Sigma = \{a, b\}$, $\text{length}(w) = 2k$. Let u the string consisting of the first k characters and v the string consisting of the final k characters. So $w = uv$ and $\text{length}(u) = \text{length}(v)$, so $w \in Y$.

Now we show $Y \subseteq X$. Let $uv \in Y$, so we have $\text{length}(u) = \text{length}(v)$. The concatenation uv satisfies $\text{length}(uv) = \text{length}(u) + \text{length}(v) = 2\text{length}(u)$. Also, since $\{a, b\}$ is the whole alphabet, $\text{length}(uv) = n_a(uv) + n_b(uv)$. So $n_a(uv) + n_b(uv) = 2\text{length}(u)$, and $uv \in X$. ♣

2. Let $\Sigma = \{a, b\}$ and let the language $L \subseteq \Sigma^*$ be defined recursively by
BASIS: $a^3 \subseteq L$, and $b^3 \subseteq L$, and
RECURSIVE STEP: If $w = uaav \in L$, then $ubaaabv \in L$ and $uaabaav \in L$.
If $w = ubbv \in L$, then $ubbabbv \in L$ and $uabbbav \in L$.
CLOSURE: All elements of S are obtained from the basis after a finite number of applications of the recursive step.

a) List all the elements in S_0 and all the elements of S_1.

♣ $S_0 = \{aaa, bbb\}$

$S_1 = \{aaa, bbb, abaaab, aaabaa, baaaba, aabaaa, abbbab, bbabbb, babbba, bbbabb\}$

(Remember, the new elements of S_1 are obtained from S_0 by applying one rule. For $uaav \in S_0$ it must be $uaab = aaa$, with $u = \lambda$ or $v = \lambda$. $a(aa)\lambda$ gives $a(aabaa)$ and $a(baaab)$. $\lambda(aa)a$ gives $(aabaa)a$ and $(baaab)a$. There are also four from bbb, and don't forget to include aaa and bbb.) ♣

b) Prove carefully by induction that every string $w \in L$ has either aaa or bbb as a substring, that is, w can be written either as $w = paaaq$, or $w = pbbbq$ for some $p, q \in \Sigma^*$.

[Your proof must be an induction proof, and must be clear and carefully written.]

♣ We show, for all $N \in \mathbb{N}$, that every element of L_N is in the proper form.

Base Case: $N = 0$. Then $L_0 = \{a^3, b^3\}$ and both strings are in the proper form with $p = q = \lambda$.

Inductive step: Let N be given and suppose that every element of L_N is in the proper form.

Let $w \in L_{N+1}$. If $w \in L_N$ then w is in the proper form by the inductive hypothesis. Otherwise w is the result of applying one of the rules to $uxxv \in L_N$ with $x \in \Sigma$. Since $uxxv \in L_N$, we have $uxxv$ contains either a^3 or b^3 as a substring by the induction hypothesis. If u or v contains a^3 or b^3, then so does w, and we are done. Otherwise, the substring a^3 or b^3 in $uxxv$ must intersect with the xx. So the only case remaining is that $uxxv$ has either u ending in the letter x or v beginning in the letter x.

If $x = a$, then $uaav$ is transformed to either $uaabaav$ or $ubaaabv$. The second one, $ubaaabv$, contains a^3. For the first one, since u ends in a or v begins in a, the string w must have a substring a^3 in this case as well.

If $x = b$, the situation in this last case is just the same. $ubbv$ is transformed to either $uabbbav$ or $ubbabbv$. The first one, $uabbbav$, contains b^3. For the second one, since u ends in b or v begins in b, the string w must have a substring b^3 in this case as well.

Thus every element of L_{N+1} contains a^3 or b^3 and the result is true by induction. ♣

3. Let L begin the set of strings on $\Sigma = \{a, b, c\}$ which do contain abc but which do not contain bbb as substrings.

Design a regular expression for the language L.

Make sure that your expression matches every string in L and does not match any string not in the language.

Your design principles should make your expression readable and understandable.

♣ Since a bbb substring cannot interact with with an abc, every string in L can be factored as a string with no bbb, followed by abc followed again by a string with no bbb. So mainly our task is to design a regular expression for strings with no bbb. So all b's and bb's must be separated by a non-empty string of only a's and c's, so by $(a \cup c)^+$

The expression $(b \cup b^2)[(a \cup c)^+(b \cup b^2)]^*$ gives the initial b's and optionally any following ones with the required separation. The string can additionally start and end with a string without b's, or have no simply have no b's at all:

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

Now we need the required abc in the middle of two of them: $[(a \cup c)^* \cup [(a \cup c)^*(b \cup b^2)[(a \cup c)^+(b \cup b^2)]^*(a \cup c)^*]]abc[(a \cup c)^* \cup [(a \cup c)^*(b \cup b^2)[(a \cup c)^+(b \cup b^2)]^*(a \cup c)^*]]$. ♣

4. Let $\Sigma = \{a, b\}$. Consider the grammar

$$\begin{aligned} G : S &\rightarrow A \mid B \mid C \\ A &\rightarrow aaB \mid aaC \mid aa \\ B &\rightarrow bbA \mid bbC \mid bb \\ C &\rightarrow ccA \mid ccB \mid cc \end{aligned}$$

a) Draw a derivation tree for $ccbbaa \in L(G)$.
b) Describe the language of this grammar in words, or set theoretically.
c) Design a regular expression for the language of the grammar. (It is much more important that the expression is correct and well designed than being short.)

♣ a)

```
        S
        |
        C
      / | \
     c  c  B
          / | \
         b  b  A
              / |
             a  a
```

And you read the derived word left to right in the embedded tree.

b) It is all non-empty strings of a^2, b^2 and c^2 in which a^4, b^4 and c^4 do not occur.

A set theoretic description can be tricky, but thinking about it, like thinking about verbal description, should help you with the next part. Here is one:

$$L(G) = \{w \in \{a, b, c\}^* \mid w = x_1^2 x_2^2 \cdots x_k^2; k > 0; x_i \in \{a, b, c\}; x_i \neq x_{i+1}\}$$

c) The c^2's, if any must be separated by non-empty strings of alternating a^2 and b^2, so we design that first: $[(a^2(b^2a^2)^*(b^2 \cup \lambda)) \cup (b^2(a^2b^2)^*(a^2 \cup \lambda))]$, with two cases distinguishing if string starts with a^2 or b^2.

Setting $W = [(a^2(b^2a^2)^*(b^2 \cup \lambda)) \cup (b^2(a^2b^2)^*(a^2 \cup \lambda))]$ to be the separator, the final expression will be $W \cup [(W \cup \lambda)c^2(Wc^2)^*(W \cup \lambda)]$.

Here is the whole thing with the W's copied in:
$[(a^2(b^2a^2)^*(b^2\cup\lambda))\cup(b^2(a^2b^2)^*(a^2\cup\lambda))]\cup[([(a^2(b^2a^2)^*(b^2\cup\lambda))\cup(b^2(a^2b^2)^*(a^2\cup\lambda))]\cup\lambda)c^2$
$([(a^2(b^2a^2)^*(b^2\cup\lambda))\cup(b^2(a^2b^2)^*(a^2\cup\lambda))]c^2)^*([(a^2(b^2a^2)^*(b^2\cup\lambda))\cup(b^2(a^2b^2)^*(a^2\cup\lambda))]\cup\lambda)]$.

Here is a shorter one. Do you see how it was built?

$$(cc \cup \lambda)[(a(abba)^*a(bb \cup \lambda) \cup b(baab)^*b(aa \cup \lambda))cc]^*(a(abba)^*a(bb \cup \lambda) \cup b(baab)^*b(aa \cup \lambda))$$

There are many other ways to get a regular expression. You don't have to focus on the c's first, like as above. Or you can convert the grammar above to a regular grammar, (not always possible but for this one it can be done), or convert to an automaton, and convert from there – but those usually lead to long ugly expressions, and the many steps make it likely that errors are introduced. ♣

5. Given the grammar

$$
\begin{aligned}
G : S &\rightarrow C \mid D \mid E \mid \lambda \mid ABCDEFGH \\
A &\rightarrow aAA \mid AaA \mid AAa \\
B &\rightarrow b \mid bb \mid bbb \mid F \\
C &\rightarrow aD \mid \lambda \\
D &\rightarrow bE \mid E \\
E &\rightarrow cC \mid C \\
F &\rightarrow b \mid bb \mid bbb \mid B \\
G &\rightarrow aaa \mid H \\
H &\rightarrow bbb \mid \lambda
\end{aligned}
$$

Compute REACH and TERM and use them to construct a new grammar G' with no useless symbols.

Compute NULL and CHAIN on G' to convert to and equivalent G'', and essentially non-contracting grammar.

♣ There are many steps in this problem, and doing them in the correct order can save a lot of time.

$TERM_0 = \{S, B, C, F, G, H\}$, so $TERM_1 = \{S, B, C, E, F, G, H\}$, and $TERM_2 = \{S, B, C, D, E, F, G, H\}$, so the only unterminal symbol is A, so remove all rules which reference the useless A, giving G_1:

$$
\begin{aligned}
G_1 : S &\rightarrow C \mid D \mid E \mid \lambda \\
B &\rightarrow b \mid bb \mid bbb \mid F \\
C &\rightarrow aD \mid \lambda \\
D &\rightarrow bE \mid E \\
E &\rightarrow cC \mid C \\
F &\rightarrow b \mid bb \mid bbb \mid B \\
G &\rightarrow aaa \mid H \\
H &\rightarrow bbb \mid \lambda
\end{aligned}
\qquad\qquad
\begin{aligned}
G' : S &\rightarrow C \mid D \mid E \mid \lambda \\
C &\rightarrow aD \mid \lambda \\
D &\rightarrow bE \mid E \\
E &\rightarrow cC \mid C
\end{aligned}
$$

Now on G_1 compute $REACH_0 = \{S\}$, and $REACH_1 = \{C, D, E\}$, but $REACH_2 = \{C, D, E\} = REACH$, so B, F, G, and H are all unreachable. So all references to them are removed in G', above.

Now, on G' compute $\text{Null}_0 = \{S, C\}$;, $\text{Null}_1 = \{S, C, E\}$ and $\text{Null}_2 = \{S, C, D, E\}$, so all symbols are nullable. That allows us to convert to G'_1:

$$
\begin{aligned}
G'_1 : S &\rightarrow C \mid D \mid E \mid \lambda \\
C &\rightarrow aD \mid a \\
D &\rightarrow bE \mid E \mid b \\
E &\rightarrow cC \mid C \mid c
\end{aligned}
\qquad\qquad
\begin{aligned}
G'' : S &\rightarrow cC \mid c \mid aD \mid a \mid bE \mid b \mid \lambda \\
C &\rightarrow aD \mid a \\
D &\rightarrow bE \mid b \mid aD \mid a \mid cC \mid c \\
E &\rightarrow cC \mid c \mid aD \mid a
\end{aligned}
$$

Now the chains must be computed on G'_1: $\text{Chain}(S) = \{S, C, D, E\}$, $\text{Chain}(C) = \{C\}$, $\text{Chain}(D) = \{C, D, E\}$, $\text{Chain}(E) = \{C, E\}$,

Now we can convert to G'' above. All done.

♣

6. a) Suppose you have a grammar with the following rules. Find equivalent rules with no left recursion.

$$\begin{array}{rcl} A & \to & a \mid aA \mid Aa \mid BaA \mid AaB \\ B & \to & bbb \mid BBB \\ C & \to & bbb \mid BBB \mid ccc \mid CCC \mid cba \mid CBA \end{array}$$

If you do not use the methods presented in the text or lectures, then you must justify your procedure.

♣ This should be a quick procedure. A has 2 left-recursive rules and 3 'escapes'. So there will be $2 \cdot 3$ new A rules and $2 \cdot 2$ rules for the new R_A variable. B has 1 of each, so there are 2 new B rules, and 2 new R_B rules. C has 2 left-recursive rules and 4 'escapes'. So there will be $2 \cdot 4$ new C rules and $2 \cdot 2$ rules for the new R_C variable.

$$\begin{array}{rcl} A & \to & a \mid aA \mid BaA \mid aR_a \mid aAR_a \mid BaAR_a \\ B & \to & bbb \mid bbbR_b \\ C & \to & bbb \mid BBB \mid ccc \mid cba \mid bbbR_c \mid BBBR_c \mid cccR_c \mid cbaR_c \\ R_A & \to & aB \mid a \mid aBR_A \mid aR_A \\ R_B & \to & BB \mid BBR_B \\ R_C & \to & BA \mid CC \mid BAR_C \mid CCR_C \end{array}$$

♣

b) Convert to the following grammar to an equivalent grammar in Chomsky normal Form

$$\begin{array}{rcl} G : S & \to & aa \mid P \\ P & \to & aa \mid aQaQ \\ Q & \to & bb \mid bPbP \\ R & \to & cc \mid bPQPRPQb \mid aPRQRaPaQaR \mid abc \end{array}$$

If you do not use the methods presented in the text or lectures, then you must justify your procedure.

♣ S has a chain rule, but the chain is very short and we can do in the end. R is unreachable, so can be deleted. So we convert first to G',

$$\begin{array}{rcl} G' : S & \to & AA \mid P \\ P & \to & AA \mid AQAQ \\ Q & \to & BB \mid BPBP \\ A & \to & a \\ B & \to & b \end{array}$$

$$\begin{array}{rcl} G'' : S & \to & AA \mid CC \\ P & \to & AA \mid CC \\ Q & \to & BB \mid DD \\ A & \to & a \\ B & \to & b \\ C & \to & AQ \\ D & \to & BP \end{array}$$

and then cut (and follow the chain) to make G'', above. ♣

7. Design a Deterministic Finite Automaton for the language of all strings on {a,b,c} which contain the substring *aa* or the substring *bb* but not both.

Your design should be clear and explained.

♣ States of interest

S start.

P_a, P_b, P_c – prefix has neither, but ends in the subscript letter. None are final states.

A_b, A_n – prefix has aa, not bb, and ends in b (or $n$ot). Both are final states.

B_a, B_n – prefix has bb, not aa, and ends in a (or $n$ot). Both are final states.

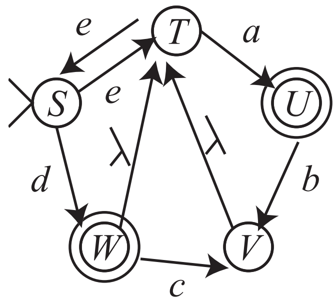
F – prefix has both squares, and so is rejected. Not a final state.

	a	b	c
S	P_a	P_b	P_c
P_a	A_n	P_b	P_c
P_b	P_a	B_n	P_c
P_c	P_a	P_b	P_c
A_b	A_n	F	A_n
A_n	A_n	A_b	A_n
B_a	F	B_n	B_n
B_n	B_a	B_n	B_n
F	F	F	F

You can do it with one fewer state (which one?), and of course you can do it with many more. So without designing the machine in advance, it is very hard not to lose track of the several aspects.

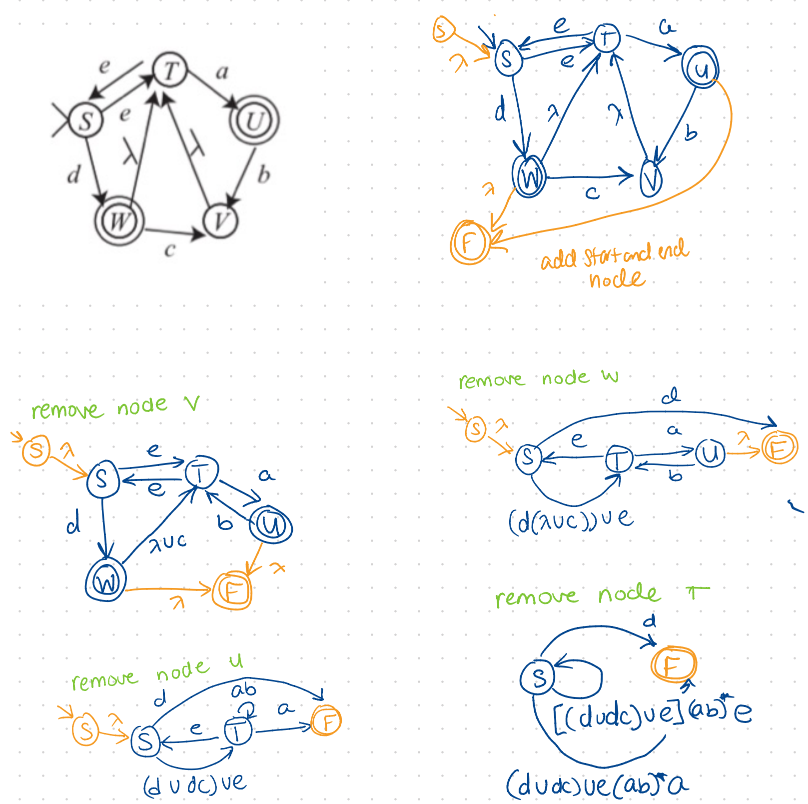
[Sorry about the table. I don't have my drawing program. :-( ] ♣

8. Here is a graph of an automaton. Find a regular expression for the language, or explain why the regular expression cannot exist.



♣ I hope you did not try to explain why none exists. We proved that the languages of all automata, deterministic or not, are regular sets, so all such languages have regular expressions. That is not to say that it is easy to find one, but actually, the procedure using expression graphs is straightforward, and for small examples, actually quite fun: ♣

8) Since NDFA-λ $\Leftrightarrow$ Regular sets we should be able to find a regular expression



final expression

$$[((d\cup dc)\cup e)(ab)^*e]^*[d \cup [((d\cup dc)\cup e)(ab)^*a]]$$