1a) Find a regular expression for the set of strings on $\{a, b, c\}$ which contain $abbb$, ba, and bbc as substrings.

$\clubsuit$ *There are several possibilities and we have to cover all of them. The three strings might be come in any of six orders, and may be mashed together, with one ordering there are* 6 *ways to mash!*

$[(a \cup b \cup c)^* abbb(a \cup b \cup c)^* ba(a \cup b \cup c)^* bbc(a \cup b \cup c)^*] \cup$
$[(a \cup b \cup c)^* abbba(a \cup b \cup c)^* bbc(a \cup b \cup c)^*] \cup$
$[(a \cup b \cup c)^* abbb(a \cup b \cup c)^* bbc(a \cup b \cup c)^* ba(a \cup b \cup c)^*] \cup$
$[(a \cup b \cup c)^* abbbbc(a \cup b \cup c)^* ba(a \cup b \cup c)^*] \cup$
$[(a \cup b \cup c)^* abbbc(a \cup b \cup c)^* ba(a \cup b \cup c)^*] \cup$
$[(a \cup b \cup c)^* ba(a \cup b \cup c)^* abbb(a \cup b \cup c)^* bbc(a \cup b \cup c)^*] \cup$
$[(a \cup b \cup c)^* babbb(a \cup b \cup c)^* bbc(a \cup b \cup c)^*] \cup$
$[(a \cup b \cup c)^* ba(a \cup b \cup c)^* abbbbc(a \cup b \cup c)^*] \cup$
$[(a \cup b \cup c)^* babbbbc(a \cup b \cup c)^*] \cup$
$[(a \cup b \cup c)^* ba(a \cup b \cup c)^* abbbc(a \cup b \cup c)^*] \cup$
$[(a \cup b \cup c)^* babbbc(a \cup b \cup c)^*] \cup$
$[(a \cup b \cup c)^* ba(a \cup b \cup c)^* bbc(a \cup b \cup c)^* abbb(a \cup b \cup c)^*] \cup$
$[(a \cup b \cup c)^* bbc(a \cup b \cup c)^* ba(a \cup b \cup c)^* abbb(a \cup b \cup c)^*] \cup$
$[(a \cup b \cup c)^* bbc(a \cup b \cup c)^* babbb(a \cup b \cup c)^*] \cup$
$[(a \cup b \cup c)^* bbc(a \cup b \cup c)^* abbb(a \cup b \cup c)^* ba(a \cup b \cup c)^*] \cup$
$[(a \cup b \cup c)^* bbc(a \cup b \cup c)^* abbba(a \cup b \cup c)^*] \cup$

$\clubsuit$

b) Give a recursive definition for language of all strings on $\{a, b\}$ which have an equal number of a's and b's.

$\clubsuit$ *The justification is that every string with an equal number of a's and b's has, somewhere, either the substring ab or the substring ba.*

BASIS: $\lambda \in L$.

RECURSIVE STEP: If $uv \in L$ *then* $uabv$ *and* $ubav$ *are in* L.

CLOSURE: Every element in the set is obtained from the basis after a finite number of applications of the recursive step. $\clubsuit$

2. Let G be the grammar

$$\begin{array}{lcl} S & \rightarrow & aSc \mid bSc \mid aAc \mid bAc \\ A & \rightarrow & c \mid c^2 \mid c^3 \end{array}$$

and let $L' = \{uv = \{a,b,c\}^* \mid u \in \{a,b\}^+, v \in c^*, 1 \leq \text{length}(v) - \text{length}(u) \leq 3\}$.

Prove that $L' = L(G)$.

Your proof should be careful, clear and complete, and will be graded on exposition.

♣ *On this page we show $L(G) \subseteq L'$. We will show that the Sentential forms belong the the set of forms of the following types:*

uSc^k, *where where* $u \in \{a,b\}^*$ *and* $\mathit{length}(u) = k$

uAc^k, *where where* $u \in \{a,b\}^+$ *and* $\mathit{length}(u) = k$

uc^{k+j}, *where where* $u \in \{a,b\}^+$ *and* $\mathit{length}(u) = k$ *and* $1 \leq j \leq 3$.

This will imply that $L(G) \subseteq L'$, *since the third types have* $(k+j) - \mathit{length}(u) = j$.

We show that the sentential forms belong to this class by induction on the number N of rules applied.

Base Case: $N = 0$, *no rules applies so the sentential form is S, which is of the first type with* $k = 0$, *(and* $u = \lambda$*).*

Inductive step. Let $N \geq 0$ be given and suppose that after N rule have been applied that the sentential form belongs to the set described. We apply one more rule.

First, if the rule is in of the first type, uSc^k, *then applying the first two S rules gives either* $uaSc^{k+1}$ *or* $ubSc^{k+1}$, *both of which are in the first type because* $\mathit{length}(ua) = \mathit{length}(ub) = k+1$. *If we apply one of the other two rules we get* $uaAc^{k+1}$ *or* $ubAc^{k+1}$ *which both of which are of the second type because ua and ub are in* $\{a,b\}^+$ *since* $u \in \{a,b\}^*$, *and* $\mathit{length}(ua) = \mathit{length}(ub) = k+1$, *just as before.*

The only other type which allows a rule is the second type, where we have type apply one of the three A rules to get either ucc^k, uc^2c^k, *or* uc^3c^k. *Taking j to be the power in the middle,* $1 \leq j \leq 3$, *we have the form* uc^{j+k}, *with* $u \in \{a,b\}^+$ *and* $\mathit{length}(u) = k$ *and* $1 \leq j \leq 3$, *so the form is of the third type.*

So, by induction, all forms belong to one of the three types, so all strings in the language belong to the third type, so belong to L'.

♣

This page intentionally blank

♣ *On this page we show $L' \subseteq L(G)$.*

Let $w \in L'$, so $w = uv$ with $u \in \{a,b\}^+$ and $v = c^{k+j}$ with $k = length(u)$ and $1 \leq j \leq 3$.

We provide a derivation sequence for w. First write $u = u'x$ where $u' \in \{a,b\}^*$ and x is either a or b. So $w = u'(x(c^j)c)c^{k-1}$. Then

$$S \overset{k-1}{\Longrightarrow} u'Sc^{k-1} \overset{1}{\Longrightarrow} u'xAc^k \overset{1}{\Longrightarrow} u'xc^jc^k = w$$

is the required derivation, with the $k-1$ S rules chosen to generate the correct sequence of a's and b's to create u', the next rule chosen depending whether x is a or b, and the last rule according to the power of c desired.

So every element of L' can be derived in G, so $L' \subseteq L(G)$.

This completes the proof that $L' = L(G)$.

♣

3. Write a **regular** grammar for the set of all strings on $\{a, b, c\}$ which contain c^2 and are of even length.

♣ *(I give a couple solutions here, but by now, you should know that just writing a grammar without giving the reader any idea of how it is constructed, or how it is supposed to work is not a complete solution. You shouldn't be offering the reader a puzzle to figure out, even though I personally like such puzzles.)*

Our grammar needs to keep track of prefixes with the following issues, evenness or oddness of length, whether they do contain c^2, or, if not, if they end in c, or not. For evenness or oddness, we will have the variable take the subscript 0 *or* 1 *respectively. C will have the role for prefixes containing c's, B, for ending in c, and A for those not.*

The grammar then follows the realization of these roles:

$$
\begin{array}{rcl}
S & \rightarrow & aA_1 \mid bA_1 \mid cB_1 \\
A_1 & \rightarrow & aA_2 \mid bA_2 \mid cB_2 \\
A_2 & \rightarrow & aA_1 \mid bA_1 \mid cB_1 \\
B_1 & \rightarrow & aA_2 \mid bA_2 \mid cC_2 \mid c \\
B_2 & \rightarrow & aA_1 \mid bA_1 \mid cC_1 \\
C_1 & \rightarrow & aC_2 \mid bC_2 \mid cC_2 \mid a \mid b \mid c \\
C_2 & \rightarrow & aC_1 \mid bC_1 \mid cC_1
\end{array}
$$

This grammar is unambiguous.

Here is another method: A string in the language has c^2 with strings of either both even, or both odd length on either side, A will strings in the language with initial even strings, and B with initial odd. C derives c followed by an even string, and C' derives c followed by an odd string, Variable E derives any string of even length at least 2, variable O derives any string of odd length. So need additionally C to derive c^2, and A and B for the two ways of continuing. Lastly, we need to allow for the even bounding strings to be empty.

$$
\begin{array}{rcl}
S & \rightarrow & aB \mid bB \mid cB \mid cC \\
A & \rightarrow & aB \mid bB \mid cB \mid cC \\
B & \rightarrow & aA \mid bA \mid cA \mid cC' \\
C & \rightarrow & c \mid cE \\
C' & \rightarrow & cO \\
E & \rightarrow & aO \mid bO \mid cO \\
O & \rightarrow & aE \mid bE \mid cE \mid a \mid b \mid c
\end{array}
$$

This gives an ambiguous grammar with different derivations of ac^2bc^2ba corresponding to the factorizations $ac^2bc^2ba = a(c^2)bc^2ba = ac^2b(c^2)ba$.

♣

4. Suppose we have a grammar which contains variables A, B, and C and all the A and B rules are

$$\begin{array}{rcl} A & \rightarrow & AB \mid AACab \mid BAaC \mid AACCAACC \mid ba \mid ABABA \\ B & \rightarrow & BB \mid BBB \mid bbb \mid AABBCC \mid aa \mid BBCCAA \end{array}$$

Introduce new variables X and Y to remove the direct left recursion.

♣ *This little problem is only here because so many people fouled it up on the quiz. First I circle strings replicated by the direct left recursive parts:*

$$\begin{array}{rcl} A & \rightarrow & A\boxed{B} \mid A\boxed{ACab} \mid BAaC \mid A\boxed{ACCAACC} \mid ba \mid A\boxed{BABA} \\ B & \rightarrow & B\boxed{B} \mid B\boxed{BB} \mid bbb \mid AABBCC \mid aa \mid B\boxed{BCCAA} \end{array}$$

with the initial variable eventually replaced with one of the "escapes".

$$\begin{array}{rcl} A & \rightarrow & BAaC \mid ba \mid BAaCX \mid baX \\ X & \rightarrow & B \mid ACab \mid ACCAACC \mid BABA \mid BX \mid ACabX \mid ACCAACCX \mid BABAX \\ B & \rightarrow & bbb \mid AABBCC \mid aa \mid bbbY \mid AABBCCY \mid aaY \\ Y & \rightarrow & B \mid BB \mid BCCAA \mid BY \mid BBY \mid BCCAAY \end{array}$$

♣

5. Design a grammar in Chomsky-Normal form whose language is the set of all strings on $\{a, b, c\}$ which factor as $u_1 u_2 u_3 \ldots u_k$ with $u_i = a^n b^m c^n$, with k, n, $m > 0$. (So $a^5b^6c^5a^5b^2c^5a^7b^{22}c^7a^3b^3c^3$ is in the language.)

♣ *The problem says to design it, so I really am obligated to explain the roles and functions of the variables. A key variable will be U, whose role, together with V and W, is to derive the strings u_i in the language. U and V recursively generate the matching a's and c's, with at least one pair created, and W derives at least one b for the interior string of u_i.*

S therefore, must derive a sequence of at least one U, so by the restrictions of the form, for S to derive a single U, we must replicate for S all the U rules. Also, an alternative S' is written to avoid a recursive start.

$$\begin{array}{rcl}
S & \to & US' \mid U_aV \\
S' & \to & US' \mid U_aV \\
U & \to & U_aV \\
V & \to & UU_c \mid WU_c \\
W & \to & WU_b \mid b \\
U_a & \to & a \\
U_b & \to & b \\
U_c & \to & c
\end{array}$$

Another reasonable method is to create an ordinary grammar simple enough, and with well chosen variable names, so that it obviously generates the language required, and then convert that grammar to Chomsky Normal Form. ♣

6. Consider the Chomsky-Normal Form grammar

$$\begin{aligned}
S &\rightarrow \lambda \mid FA \mid EB \mid b \\
A &\rightarrow FA \mid EB \mid b \\
B &\rightarrow FB \mid EC \mid a \\
C &\rightarrow FC \mid ED \mid b \\
D &\rightarrow FB \mid EA \mid a \\
E &\rightarrow a \\
F &\rightarrow b
\end{aligned}$$

a) Use the CYK algorithm to trace out a^3ba^3b and determine whether or not it it belongs to the language.

♣ *Even though the string is rather long to trace out, we can see in grammar a feature which will save us a lot of work in our search.* E *and* F *only derive strings of length* 1*, so will never occur "above" the first row. On the other hand, all binary rules either start with* E *or* F*, so in looking for binary matches, the first variable must be on the bottom row, and it must be* E *or* F*.*

Another help is the repetition, in the string.

As we discussed in class, I need not consider the variable S *until the very last, but note that* A *and* S*, except for* λ*, derive the same strings, so any position without an* A *on the 4'th row from the bottom yields a string of length* 4 *not in the language, taking care of part* b*.*

ACS							
AC	BD						
BD	BD	AC					
AC	AC	AC	BD				
BD	BD	BD	BD	BD			
BD	AC	AC	AC	BD	AC		
AC	AC	BD	BD	AC	AC	BD	
BDE	BDE	BDE	ACF	BDE	BDE	BDE	ACF

So the trace proves that aaabaaab is in the language, but that aaab is not, and neither is aaba, abaa, nor baaa. ♣

b) Find a word of length 4 not in the language and prove that it is not with the CYK algorithm.

7. Consider the language of the Chomsky-Normal Form grammer

$$\begin{array}{lcl} S & \to & \lambda \mid AF \mid BE \mid b \\ A & \to & BF \mid BE \mid b \\ B & \to & FC \mid CE \mid a \\ C & \to & DF \mid DE \mid b \\ D & \to & FA \mid AE \mid a \\ E & \to & a \\ F & \to & b \end{array}$$

Convert to an equivalent language in Greibach-Normal Form.

♣ *Because the initial variables of A, B, C, and D cycle around, there is no simple shortcut which avoids the long procedure.*

If we take the standard ordering, the first variable in each rule is larger than the variable it is produced from for every rule except $D \to AE$, and not seeing a better ordering, I prepare for that rule to explode and we move from A to D, and finally introduce a new variable G to eliminate the left D recursion, and lastly, substitute for the initial E's and F'. So the offensive rule, by our procedure, evolves into:

$$\begin{array}{lcl} D & \to & AE \\ D & \to & BFE \mid BEE \mid bE \\ D & \to & FCFE \mid CEFE \mid aFE \mid FCEE \mid CEEE \mid aEE \mid bE \\ D & \to & FCFE \mid DFEFE \mid DEEFE \mid bEFE \mid aFE \mid FCEE \mid \\ & & DFEEE \mid DEEEE \mid bEEE \mid aEE \mid bE \\ \hline D & \to & FCFE \mid bEFE \mid aFE \mid FCEE \mid bEEE \mid aEE \mid bE \mid \\ & & FCFEG \mid bEFEG \mid aFEG \mid FCEEG \mid bEEEG \mid aEEG \mid bEG \\ G & \to & FEFE \mid EEFE \mid FEEE \mid EEEE \mid FEFEG \mid EEFEG \mid FEEEG \mid EEEEG \end{array}$$

With final form of rules D, E, F, and G:

$$\begin{array}{lcl} D & \to & bCFE \mid bEFE \mid aFE \mid bCEE \mid bEEE \mid aEE \mid bE \mid \\ & & bCFEG \mid bEFEG \mid aFEG \mid bCEEG \mid bEEEG \mid aEEG \mid bEG \\ E & \to & a \\ F & \to & b \\ G & \to & bEFE \mid aEFE \mid bEEE \mid aEEE \mid bEFEG \mid aEFEG \mid bEEEG \mid aEEEG \end{array}$$

C is converted to

$$\begin{array}{lcl} C & \to & bCFEF \mid bEFEF \mid aFEF \mid bCEEF \mid bEEEF \mid aEEF \mid bEF \mid \\ & & bCFEGF \mid bEFEGF \mid aFEGF \mid bCEEGF \mid bEEEGF \mid aEEGF \mid bEGF \\ & & bCFEE \mid bEFEE \mid aFEE \mid bCEEE \mid bEEEE \mid aEEE \mid bEE \mid \\ & & bCFEGE \mid bEFEGE \mid aFEGE \mid bCEEGE \mid bEEEGE \mid aEEGE \mid bEGE \mid b \end{array}$$

B is converted to

$$\begin{array}{lcl} B & \to & bC \mid bCFEFE \mid bEFEFE \mid aFEFE \mid bCEEFE \mid bEEEFE \mid aEEFE \mid bEFE \mid \\ & & bCFEGFE \mid bEFEGFE \mid aFEGFE \mid bCEEGFE \mid bEEEGFE \mid aEEGFE \mid bEGFE \\ & & bCFEEE \mid bEFEEE \mid aFEEE \mid bCEEEE \mid bEEEEE \mid aEEEE \mid bEEE \mid \\ & & bCFEGEE \mid bEFEGEE \mid aFEGEE \mid bCEEGEE \mid bEEEGEE \mid aEEGEE \mid bEGEE \mid bE \mid a \end{array}$$

A is converted to the B rules ending with an F or an E or a lone b.

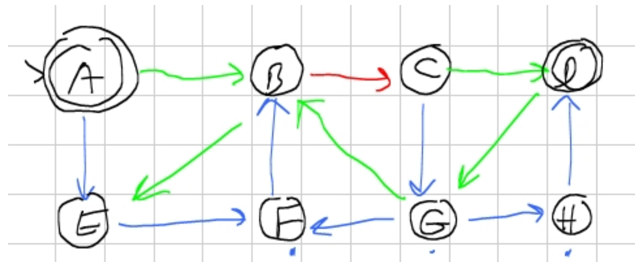
$$\begin{aligned}
A \;\rightarrow\;& bCF \mid bCFEFEF \mid bEFEFEF \mid aFEFEF \mid bCEEFEF \mid bEEEFEF \mid aEEFEF \mid bEFEF \mid \\
& bCFEGFEF \mid bEFEGFEF \mid aFEGFEF \mid bCEEGFEF \mid bEEEGFEF \mid aEEGFEF \mid bEGFEF \\
& bCFEEEF \mid bEFEEEF \mid aFEEEF \mid bCEEEEF \mid bEEEEEF \mid aEEEEF \mid bEEEF \mid \\
& bCFEGEEF \mid bEFEGEEF \mid aFEGEEF \mid bCEEGEEF \mid bEEEGEEF \mid aEEGEEF \mid bEGEEF \\
& \mid bEF \mid aF \\
& bCE \mid bCFEFEE \mid bEFEFEE \mid aFEFEE \mid bCEEFEE \mid bEEEFEE \mid aEEFEE \mid bEFEE \mid \\
& bCFEGFEE \mid bEFEGFEE \mid aFEGFEE \mid bCEEGFEE \mid bEEEGFEE \mid aEEGFEE \mid bEGFEE \\
& bCFEEEE \mid bEFEEEE \mid aFEEEE \mid bCEEEEE \mid bEEEEEE \mid aEEEEE \mid bEEEE \mid \\
& bCFEGEEE \mid bEFEGEEE \mid aFEGEEE \mid bCEEGEEE \mid bEEEGEEE \mid aEEGEEE \mid bEGEEE \\
& \mid bEE \mid aE \mid b
\end{aligned}$$

Lastly S is converted to short ones, and the A rules followed by F and the B rules followed by E:

$$\begin{aligned}
S \;\rightarrow\;& \lambda \mid b \\
& bCFF \mid bCFEFEFF \mid bEFEFEFF \mid aFEFEFF \mid bCEEFEFF \mid bEEEFEFF \mid aEEFEFF \mid bEFEFF \mid \\
& bCFEGFEFF \mid bEFEGFEFF \mid aFEGFEFF \mid bCEEGFEFF \mid \\
& bEEEGFEFF \mid aEEGFEFF \mid bEGFEFF \\
& bCFEEEFF \mid bEFEEEFF \mid aFEEEFF \mid bCEEEEFF \mid bEEEEEFF \mid aEEEEFF \mid bEEEFF \mid \\
& bCFEGEEFF \mid bEFEGEEFF \mid aFEGEEFF \mid bCEEGEEFF \mid \\
& bEEEGEEFF \mid aEEGEEFF \mid bEGEEFF \mid bEFF \mid aFF \\
& bCEF \mid bCFEFEEF \mid bEFEFEEF \mid aFEFEEF \mid bCEEFEEF \mid bEEEFEEF \mid \\
& aEEFEEF \mid bEFEEF \mid \\
& bCFEGFEEF \mid bEFEGFEEF \mid aFEGFEEF \mid bCEEGFEEF \mid \\
& bEEEGFEEF \mid aEEGFEEF \mid bEGFEEF \\
& bCFEEEEF \mid bEFEEEEF \mid aFEEEEF \mid bCEEEEEF \mid bEEEEEEF \mid aEEEEEF \mid bEEEEF \mid \\
& bCFEGEEEF \mid bEFEGEEEF \mid aFEGEEEF \mid bCEEGEEEF \mid \\
& bEEEGEEEF \mid aEEGEEEF \mid bEGEEEF \mid bEEF \mid aEF \mid bF \\
& bCE \mid bCFEFEE \mid bEFEFEE \mid aFEFEE \mid bCEEFEE \mid bEEEFEE \mid aEEFEE \mid bEFEE \mid \\
& bCFEGFEE \mid bEFEGFEE \mid aFEGFEE \mid bCEEGFEE \mid bEEEGFEE \mid aEEGFEE \mid bEGFEE \\
& bCFEEEE \mid bEFEEEE \mid aFEEEE \mid bCEEEEE \mid bEEEEEE \mid aEEEEE \mid bEEEE \mid \\
& bCFEGEEE \mid bEFEGEEE \mid aFEGEEE \mid bCEEGEEE \mid bEEEGEEE \mid aEEGEEE \mid bEGEEE \\
& \mid bEE \mid aE
\end{aligned}$$

and we see what that one lone recursion in the original led to. ♣

8. Consider the non-deterministic finite automaton with λ-rules, (a is red, b is blue and λ is green):



a) Find the λ closure of each state.

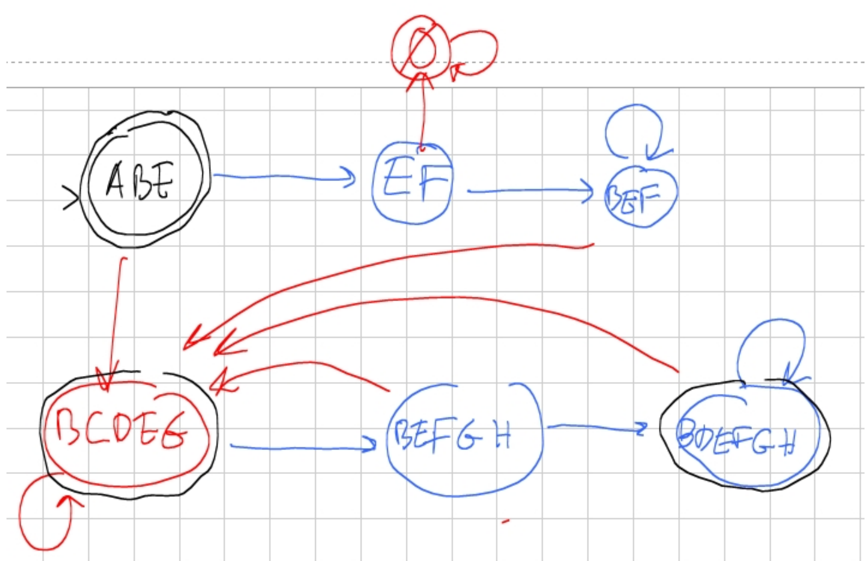
V	$\lambda(V)$	V	$\lambda(V)$
A	$\{A, B, E\}$	E	$\{E\}$
B	$\{B, E\}$	F	$\{F\}$
C	$\{B, C, D, E, G\}$	G	$\{B, E, G\}$
D	$\{B, D, E, G\}$	H	$\{H\}$

b) Use the λ-closure fill in the transition relation

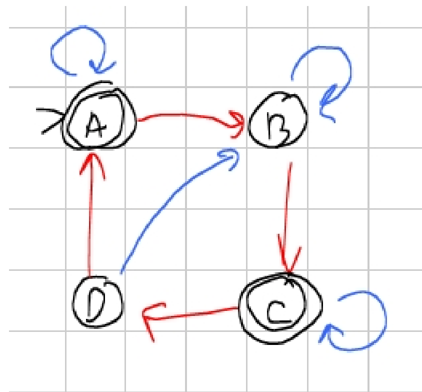
	a	b
A	$\{B, C, D, E, G\}$	$\{E, F\}$
B	$\{B, C, D, E, G\}$	$\{F\}$
C	$\{B, C, D, E, G\}$	$\{B, E, F, G, H\}$
D	$\{B, C, D, E, G\}$	$\{F, H\}$
E	$\emptyset$	$\{F\}$
F	$\emptyset$	$\{B, E\}$
G	$\{B, C, D, E, G\}$	$\{F, H\}$
H	$\emptyset$	$\{B, D, E, G\}$

If you miss transitions here, it will go badly later. A transition is any path which processes a single letter, so $\lambda(x(\lambda(V)))$, not just $x(\lambda(V))$ or $\lambda(x(V))$.

c) Use parts a) and b) to construct an equivalent deterministic finite automaton.



9.For the automaton below, red is a and blue is b:

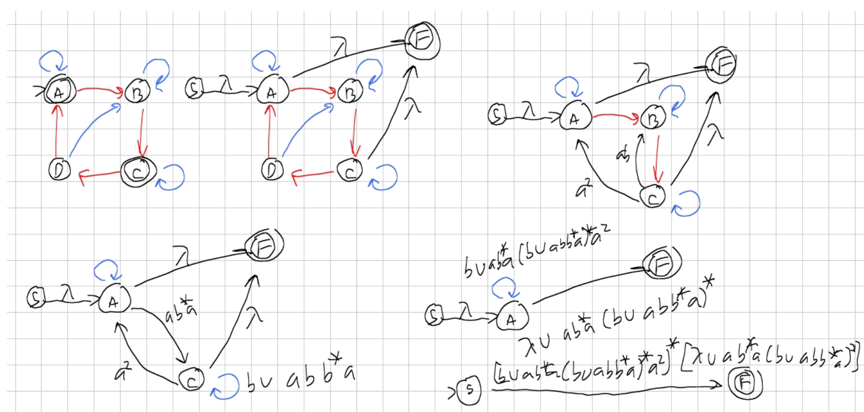


a) Find a regular expression for the language, and justify why your expression is correct.

♣

$$[b \cup ab^*a(b \cup abb^*a)^*a^2]^*[\lambda \cup ab^*a(b \cup abb^*a)^*]$$

The answer is justified by the following sequence of expression graphs:



♣

b) Construct a grammar in Chomsky normal form for this language. Explain why your solution is correct.

♣ *Since you have a deterministic automaton, you should immediately convert to a regular grammar, which is simple to convert. If you are using the regular expression, or starting from scratch, you are not thinking about the various conversion options open to you now:*

$$\begin{array}{rcl} S & \to & \lambda \mid bA \mid aB \mid b \\ A & \to & bA \mid aB \mid b \\ B & \to & bB \mid aC \mid a \\ C & \to & bC \mid aD \mid b \\ D & \to & bB \mid aA \mid a \end{array}$$

$$\begin{array}{rcl} S & \to & \lambda \mid U_bA \mid U_aB \mid b \\ A & \to & U_bA \mid U_aB \mid b \\ B & \to & U_bB \mid U_aC \mid a \\ C & \to & U_bC \mid U_aD \mid b \\ D & \to & U_bB \mid U_aA \mid a \\ U_a & \to & a \\ U_b & \to & b \end{array}$$

♣

10. Show that the language $L = \{u \in \{a, b, c\}^* \mid n_a(u) + n_b(u) = 2^k, k \in \mathbb{N}\}$ is not context free.

♣ *Let K be given. consider the string a^{2^K} which is in the language. (It is very long. If there is a factorization with $a^{2^K} = uxwyv$ with xwy of with length less than K, x and y non-trivial and simultaneously pumpable, L would have a word of length $2^K + length(x) + length(y)$, but $length(x) + length(y) \leq K < 2^K$, and L has now strings whose length is between 2^K and $2^{K+1} = 2^K + 2^K$. So x and y are non simultaneously pumpable, and L is not context free.* ♣