NAME \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

CS 420/520 Theory of Computation, Spring 2019 at Indiana State University,   
taught by Jeff Kinne

Exam 2 - Context Free Languages

Points - each part is graded as 0, 1, 2, or 3. There are 8 questions total, with each worth 3 points (though they’re not equally difficult). You should skip one. So 21 points total.

1. Give a DFA, NFA, RE for a regular language.

L = {w1.w2 | strings of a's and b's where   
 (# a's in w1 mod 2) = (# b's in w2 mod 2)}  
 In L: a.b, aa.bb, a.aab, ab.baaa, .bb  
 Not in L: a, .b, aa.a, ab.b

See exam2\_1.jff

Grading: -1/2 for tiny mistake, -1 if correctly did the language {#a’s = #b’s mod 2}

2. Give a CFG for a CFL language.

L = {w1.w2 | w2 is a substring of w1R and w1, w2 are strings of a’s and b’s}  
 In L: abb.bba, abb.ba, abb.b, abb., aabaab.baaba  
 Not in L: abb.bbb, abb.ab  
  
S -> UV  
U -> aU | bU | epsilon  
V -> aVa | bVb | T.  
T -> aT | bT | epsilon

Grading: this was a tough one. -1 if you had something heading in the right direction (need to have a rule like V -> aVa somewhere) but still with a major problem.

3. Give a PDA for a CFL language.

L = {w1.w2 | w1 has same # of a’s as w2 and w1,w2 are strings of a’s and b’s}  
 In L: aa.aa, aba.aa, bb.b, ababa.baaa

Not in L: a.b, bba.bb, babaa.aab  
  
See exam2\_3.jff

4. Prove that a non-CFL language is not CF.

L = {w1.w2.w3 | w1 has same # a’s as w2 and w2 has same # b’s as w3}

Strings in L: aba.bbaa.abbaaaaa, abb.a., abbbaaa.baabaa.bb

Strings not in L: ab.bb.bb, ab.ab.bb, ab.bb.aa

Note that this is a little tricky – the first few things I tried for “s” in the proof did not work (s could be pumped and still would be in the language). If you try a few s that you figure out don’t work, I can give you partial credit for that.  
  
Suppose L is CF, and let p be the pumping length for L from the pumping lemma for CF for L. Let s = ap.bpap.bp. Note that |s| >= p and s is in L. Then we can write s as uvxyz where |vxy| <= p, |vy| >= 1, and uvixyiz is in L for all i >= 0. For convenience we refer to the 4 different parts of s as A1, B1, A2, B2.

Consider the cases for where vxy can lie within s, and for each we say what happens with the string s’ = uvixyiz for i=2.

Case 1: xvy inside of A1 – s’ has more a’s in the w1 portion than in the w2 portion, meaning s’ would not be in L.

Case 2: xvy inside of A2 – s’ has more a’s in the w2 portion than in the w1 portion, meaning s’ would not be in L.

Case 3: xvy inside of B1 or B2 – similar to cases 1 and 2.

Case 4: xvy crosses between A1 and B1 – s’ either has too many “.” characters or has too many a’s in the w1 portion or too many b’s in the w2 portion. In any case, s’ is not in L.

Case 5: xvy crosses between B1 and A2, or between A2 and B2 – similar to case 4.

5. Given the following CFG -

DOC -> LINES

LINES -> LINE\n LINES | epsilon

LINE -> NAME,PHONE,AGE

NAME -> FIRST-LAST

PHONE -> AREA-PREFIX-LINENUMBER

FIRST -> [a-zA-Z]+

LAST -> [a-zA-Z]+  
 AGE -> \d\d\d

AREA -> \d\d\d

PREFIX -> \d\d\d

LINENUMBER -> \d\d\d\d

Note - variables in the grammar are DOC, LINES, LINE, NAME, PHONE, AGE, FIRST, LAST, AREA, PREFIX, LINENUMBER. epsilon is the empty string. The last five rules give regular expressions for the variable. - and , are terminal symbols in the other rules. \n is the newline character in the second rule.

Give a string that is in the language:  
Jeff-Kinne,111-222-3333,042

Give a string that is not in the language:

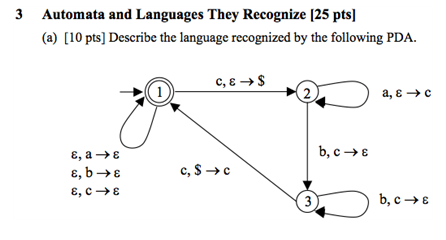
Jeff-Kinne

Is this language itself closed under star? That is, if a string w is in the language, are ww, www, wwww, etc. also in the language?

Yes. A string in the language is some number of lines. Repeating a string would still be some number of lines (twice as many).

6. Write out the transition table for the following PDA from

<https://i.stack.imgur.com/xEQqP.png>  
 Note – ignore the text in the image, just look at the PDA.



State, input symbol, stack symbol -> state, stack-symbol-push

--------------------------------------------------------------  
 1 epsilon a 1 epsilon

1 epsilon b 1 epsilon

1 epsilon c 1 epsilon

1 a epsilon 2 $

2 a epsilon 2 c

2 b c 3 epsilon

3 b c 3 epsilon  
 3 c $ 1 epsilon

Give a string that is accepted by the PDA: empty string, caabbc

Give a string that is not accepted by the PDA: a, b, c, cabbc

What is the shortest string accepted by the PDA? Empty string, after that cabc

If the set of accept states was swapped (so 2 and 3 are accept states) - what would the language be? Language would be somewhat similar to what it is now. In the picture, the language is (cai+1bi+1c)\*. If we made 2 and 3 accept states instead of state 1, then the accepted strings would be some repetitions of cai+1bi+1c where the last time through it is somewhere in the middle of the a’s and b’s. For example, the following would be accepted: caab, caabb, c, cabccaa

~~7. Give the proof idea for the following statement that is true.~~

~~Statement:~~

8. The following statement is not true in general for CF's. Give an example language where the statement is indeed true, and give another example where the statement is not true.

Statement: any language L that is CF is itself closed under concatenation; that is, if x is in L and y is in L then xy is in L.

Language where this is true: language from problem 5. Another example is {w | w has an equal number of a’s and b’s}

Language where this is not true: language from problem 2 (because xy would have two “.” characters which definitely is not allowed). Another example is {anbn | n >= 0}

9. Convert the following CFG into Chomsky normal form

S -> 01 U 11 T 01

U -> 00 U | 01 U

T -> 10 T | 11 T | V

V -> 1001 | 0110 | epsilon

Grading notes: note that CNF requires rules to either be like X -> U V or like X -> a. Many put rules that are not CNF.

Note: the following are variables in my grammar -   
S, S1, S2, …, U, U1, …, V, V1, …, T, T1, …

First attempt –

S -> O S1  
O -> 0  
I -> 1  
S1 -> I S2  
S2 -> U S3  
S3 -> I S4  
S4 -> I S5

S5 -> T S6  
S6 -> O I  
U -> O U1  
U1 -> O U  
U -> O U2  
U2 -> I U  
T -> I T1  
T1 -> O T  
T -> I T2  
T2 -> I T  
T -> V # still need to deal with this one  
V -> I V1  
V1 -> O V2  
V2 -> O I  
V -> O V3  
V3 -> I V4  
V4 -> I O

V -> epsilon # still need to deal with this one

Fix those two issues

\* To remove T -> V, we take all of the rules where T was present and add a copy where we have V instead, so we add the following rules  
 T2 -> I V  
and remove the rule T -> V

\* To remove V -> epsilon, we take all of the rules where V was present and add a copy where V is not present, so we add the following rules  
 T2 -> I  
and remove the rule V -> epsilon  
  
Now we need to fix the rule T2 -> I, so we add the following rules  
 T -> I I  
and remove the rule T2 -> I

That should be it. Probably I have at least one mistake in there. Let me know.