Assignment 5 - Chapters 10, 11, 12

By Dylan Gartin.

Note for class: non-terminal -> capital letter, terminals are lowercase

**Q1) (Total 15 Points)**

**Use the pumping lemma to show that this language is nonregular:**

1. {a^n b^n a^n } = {aba aabbaa aaabbbaaa aaaabbbbaaaa…….}
2. { a^n ba^n } = {aba aabaa aaabaaa……}
3. { a^n b^2n } = {abb aabbbb aaabbbbbb ……}

**(Part 1)**

for [a^n b^n a^n ]

In order to see if this is regular on non-regular we will look at “aabbaa”

Using pumping lemma equation on this where ( x y^n z ) should be where there are three words such that all words are in the language.

If we choose x=a y=a and then w = bbaa we then can apply pumping to this to make the word “aaabbaa”

x = a

y= a

w= bbaa

x y^n z = x y^2 z = a (a)^2 aabb = aaabbaa

this string does not belong in the language when made there is a conflict with the pumping of lemma function, thus making it non-regular.

**(Part 2)**

for [a^n ba^n ]

In order to see if this is regular on non-regular we will look at “aabaa”

If we choose x=a y=a and then w = baa we then can apply pumping to this to make the word “aaabaa”

X= a

Y= a

W=abb

x y^n z = x y^2 z = a (a)^2 baa = aaabaa

this string does not belong in the language when made there is a conflict with the pumping of lemma function, thus making it non-regular.

**(Part 3)**

for [a^n b^2n]

In order to see if this is regular on non-regular we will look at “aabbbb”

Use the same methodology as before we can prove this one also

If we choose x=a y=a and then w = bbbb we then can apply pumping to this to make the word “aaabbbb”

X= a

Y= a

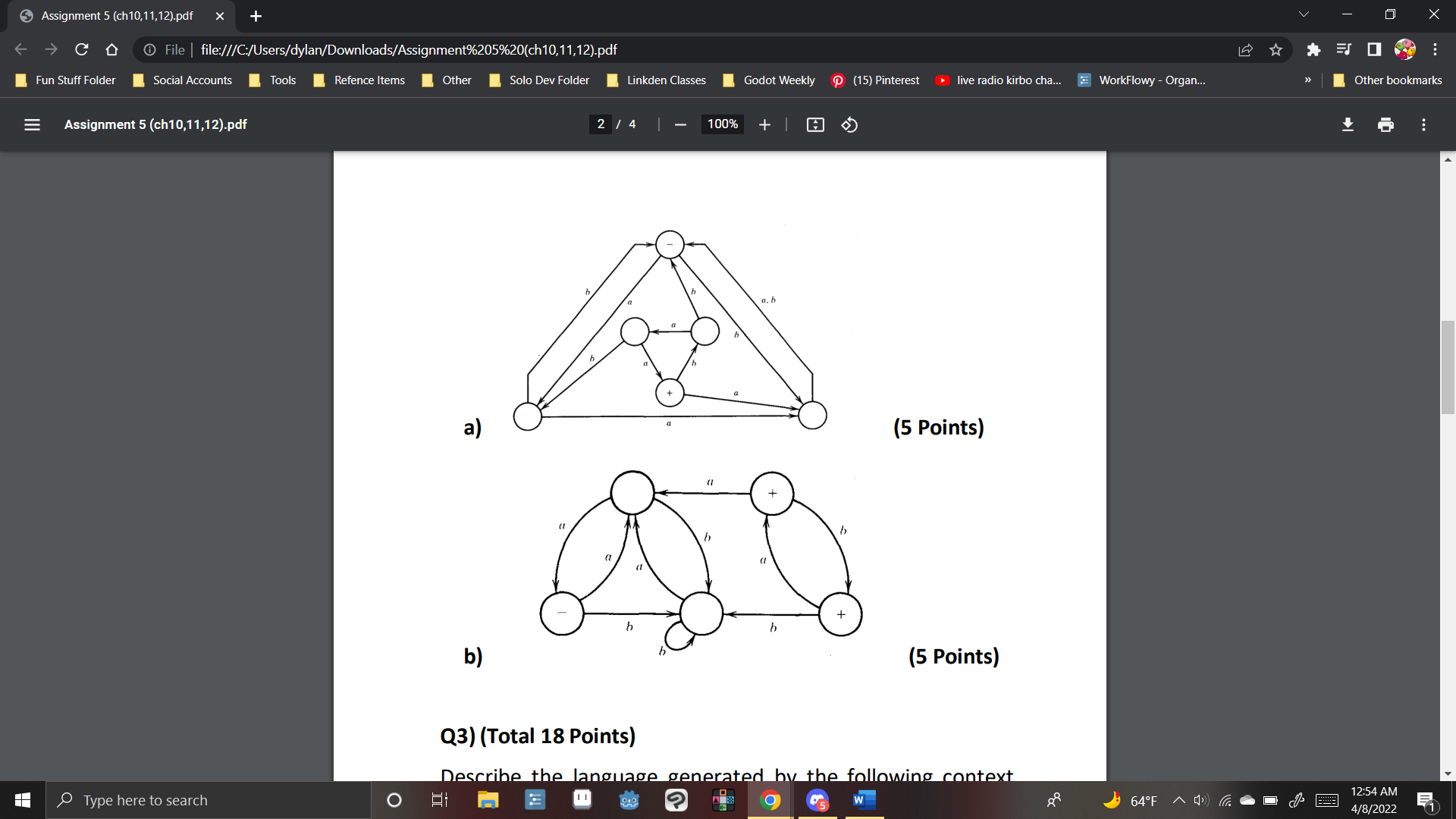
W=bbbb

x y^n z = x y^2 z = a (a)^2 bbbb = aaabbbb

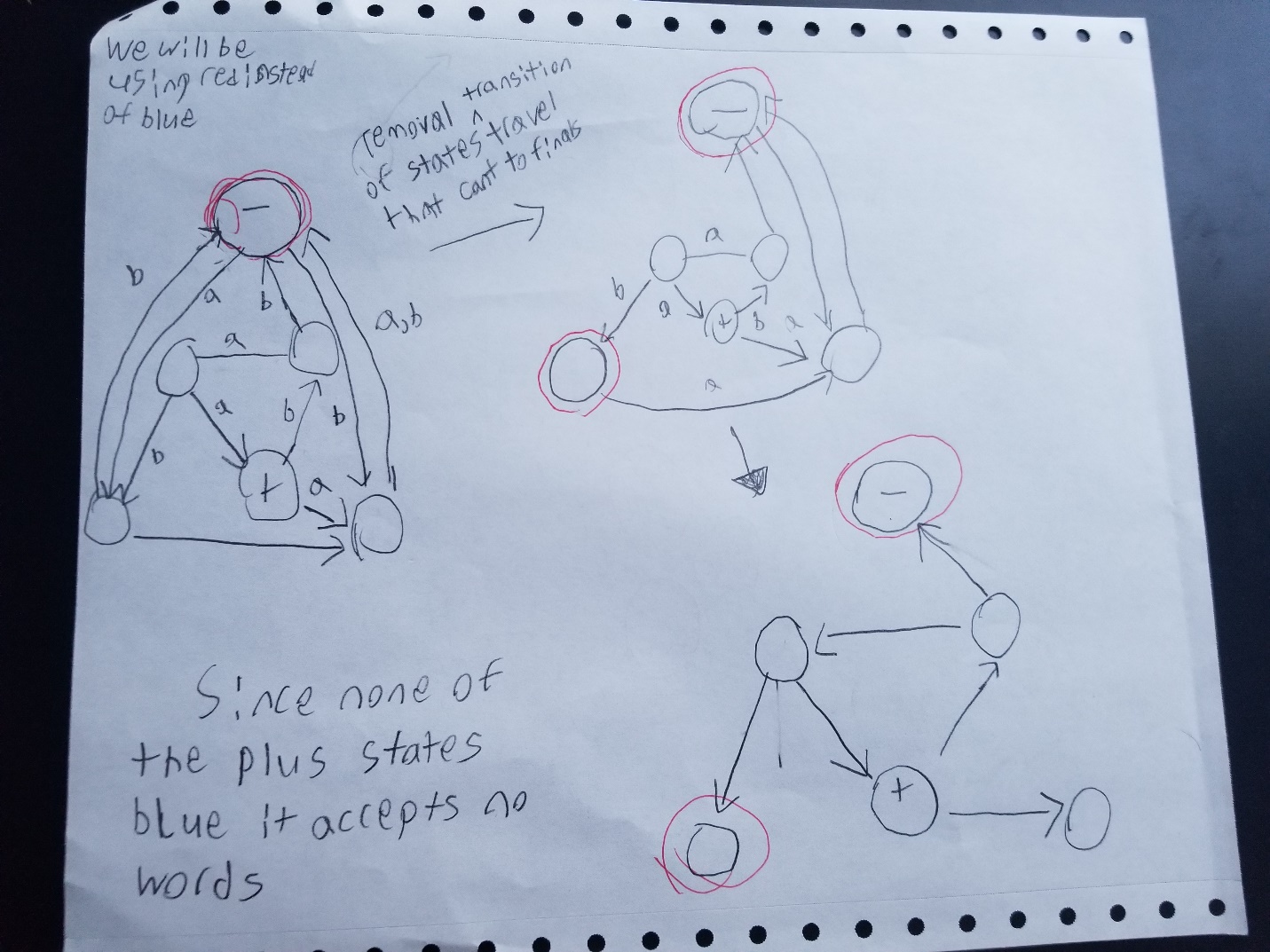
this string does not belong in the language when made there is a conflict with the pumping of lemma function, thus making it non-regular.

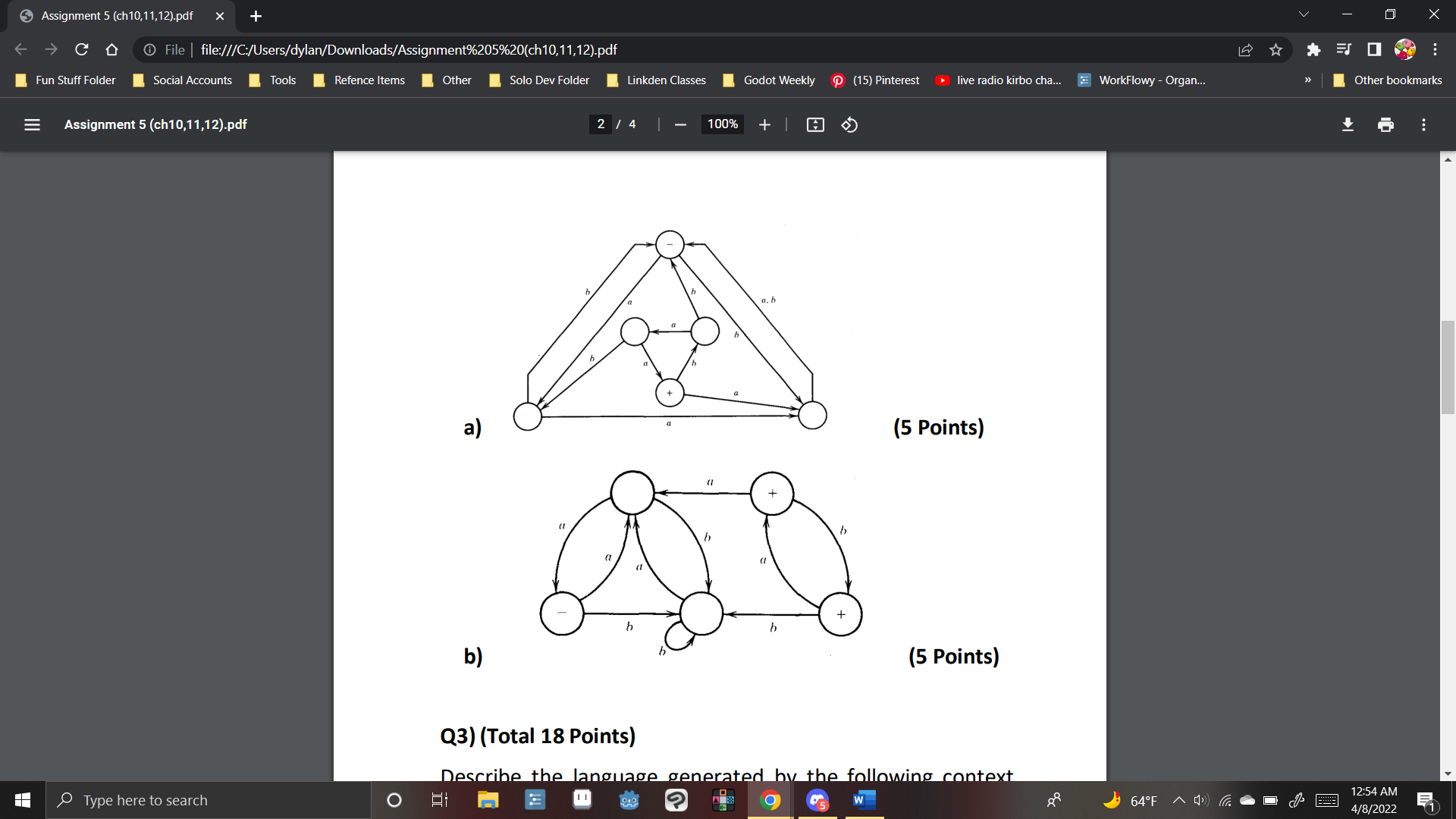
**Q2) (Total 10 Points)**

**By using blue paint, determine which of the following FA’s accept any words:**

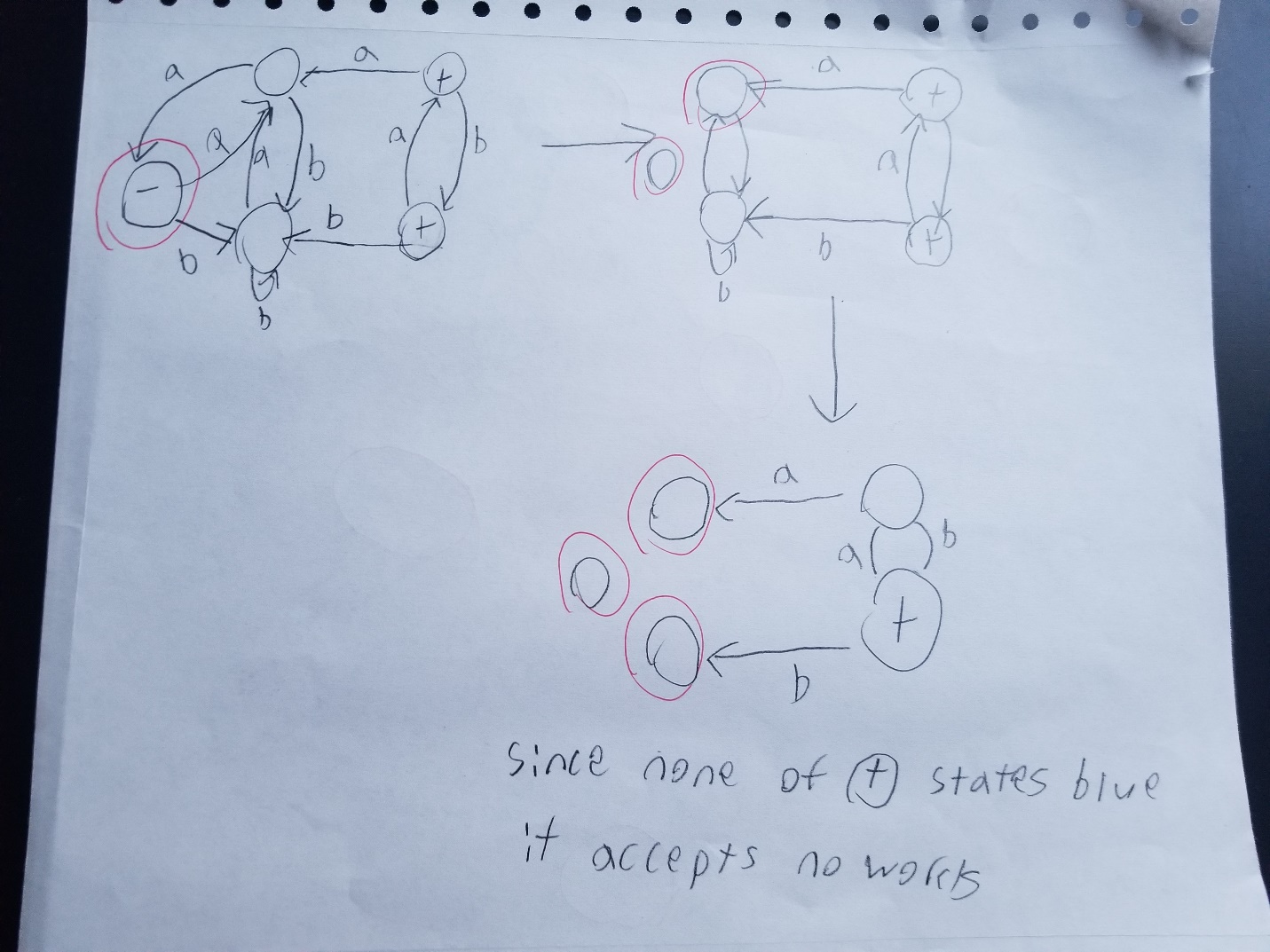


Part A)





Part B)



Q3) (Total 18 Points)

Describe the language generated by the following context free grammar (CFG)in English and write regular expressions:

a)

S → SS

S → ZZZ

Z → bZ

Z → Zb

Z → a

From the Z rules Z → bZ, Z → Zb, Z → a we can combine them to the regular expression (b\* a b\*) after that we then have the other two rules, thus Z = (b\* a b\*)

S → SS means that the whole machine can be duplicated (S)\* essentially

S → ZZZ means that the Z’s must come in set’s of three thus meaning ZZZ = (b\* a b\*) (b\* a b\*) (b\* a b\*)

Combine these together and we get the following regular expression

**((b\* a b\*) (b\* a b\*) (b\* a b\*))\***

**Thus this is a language of a and b where a is always in multiples of 3, with any number of b’s**

b) S → aS

S → bb

S → aS is equivalent to a\* being at the front of the word.

S → bb and this is equivalent well to bb

Combine these together and we get the following regular expression

**a\*bb**

**Thus this is the language of a’s and b’s in which you can have any number of a’s at the start of the word ended by the letters bb**

c) S → XYX

X → aX

X → bX

X → Λ

Y → bbb

First by looking at the rules we have for X: X → aX, X → bX, and X → Λ this allows us to generate any combination of a and b thus this is (a+b)\*

As for y it directly translates too Y → bbb

Lastly the start state S → XYX tells us to put the above X and Y states in this pattern seen here X – Y – X

Combine these together and we get the following regular expression

**(a+b)\* bbb (a+b)\***

**Thus this is any word using a and b that contains the substring bbb within it**

**Q4) (Total 15 Points)**

**Find CFG for the following languages over the alphabet**

∑ = {a, b}:

* All words that have different first and last letters.

This language can be made by simply having a starting rule that defines words that start with a must end with b and words that start with b must end with a. then in the middle

S -> aXb | bXa (starting rule that makes sure it follows the different letter rules)

X-> a| b | XX | **Λ** (a way to write (a+b)\* )

**Thus the CFG is the following**

**S -> aXb | bXa**

**X-> a| b | XX | Λ**

* All words in which the letter b is never tripled.

To do this we simply need to make sure bbb and any strings past that cannot be reached

For our first rule we can have **S -> Λ | b | bb** as these are the words that contain only b’s or no b’s that still fill this condition.

After that we simply need to add in a way to have a\* to the language.

**S -> Sa** (allows for any number of a’s in the language)

And then also have where there can be a\* in between the b and bb segments of the words

**S -> Sab**

**S -> Sabb**

**Thus the CFG is the following**

**S -> Λ | b | bb**

**S -> Sa**

**S -> Sab**

**S -> Sabb**

* All words that do not have substring ab.(iii)

We need to avoid generating the ab substrings therefore

Start with the words that we know don’t generate ab **S -> Λ |a| b |ba** Knowing this we simply need to generate the word a\*b\* subset as that will allow there to be no ab

**Thus the CFG is the following**

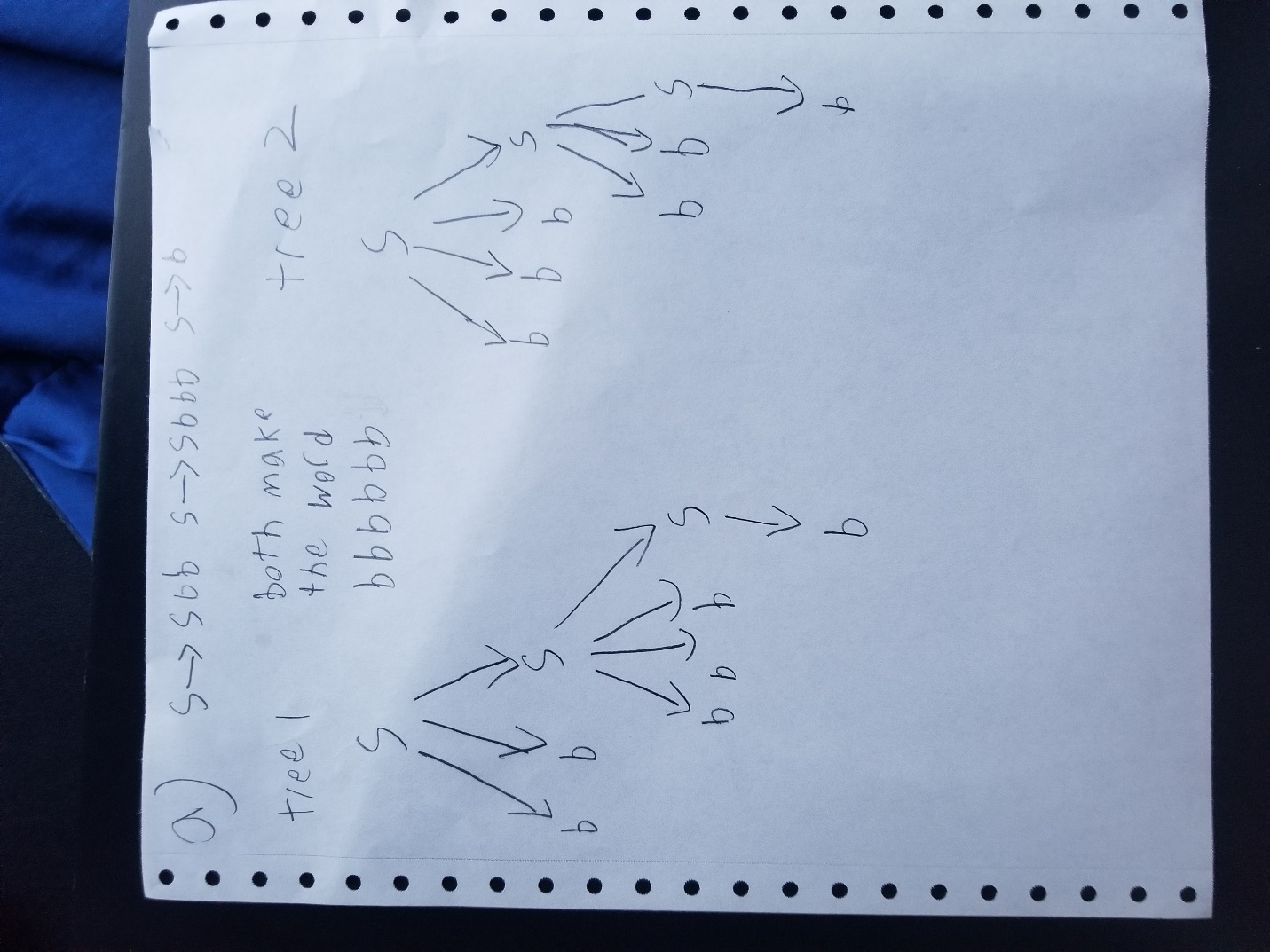
**S -> Λ | bS | aX (generates the base ba cases, b repeats on start state)**

**X -> aX | Λ (and to repeat a we use this one)**

**Q5) (Total 15 Points)**

**Show that the CFG below is ambiguous by finding a word with two distinct syntax trees. Show both syntax trees.**

1. S → Sbb S → Sbbb S → b



1. S → AA A →AAA|a|bA|Ab

