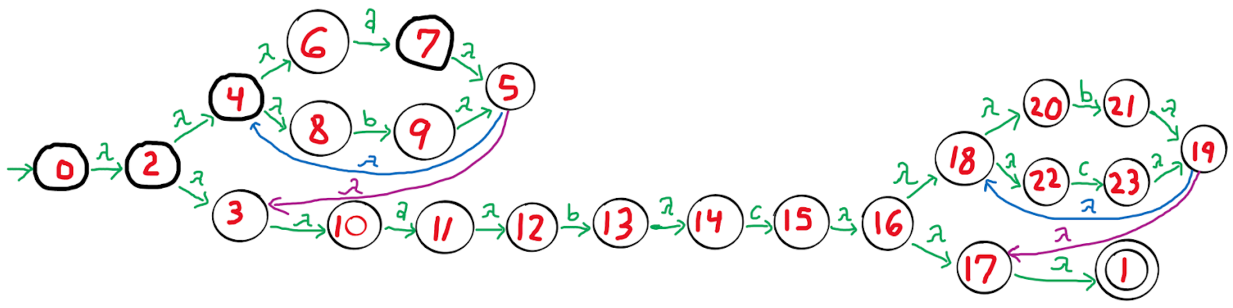
Midori Lynch
Homework 4
Problem 1

NDFSA Diagram



Input file input.ndfsa

```
A a,b,c
S 0,0,1,1,2,0,3,0,4,0,5,0,6,0,7,0,8,0,9,0
S 10,0,11,0,12,0,13,0,14,0,15,0,16,0,17,0
S 18,0,19,0,20,0,21,0,22,0,23,0
B 0
D 0,@,2
D 2,@,3
D 2,@,4
D 3,@,10
D 4,@,6
D 4,@,8
D 5,@,4
D 5,@,3
D 6,a,7
D 7,@,5
D 8,b,9
D 9,@,5
D 10,a,11
D 11,@,12
D 12,b,13
D 13,@,14
D 14,c,15
D 15,@,16
D 16,@,17
D 16,@,18
D 18,@,20
D 18,@,22
D 19,@,18
D 19,@,17
```

```
D 17,@,1
D 20,b,21
D 21,@,19
D 22,c,23
D 23,@,19
T abc
O
T bbabcbb
O
T aaabbabaaabababbabaaaaabc
O
T babc
O
T abcbc
O
T abaaabbaac
O
T bacb
O
T aaaabbabaababaabaabbbc
O
T bbaabccb
O
T baaabbaababbabbabc
O
```

I plugged this input file into the code from problem four of the previous homework.
Here is the file output.ndfsa that I got after the sample run.

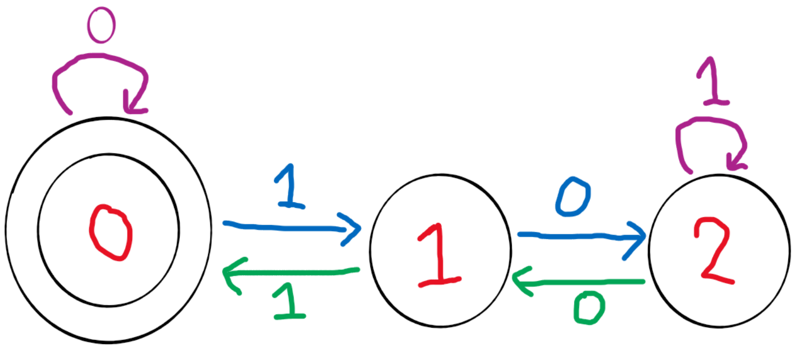
```
A a,b,c
S
0,0,1,1,2,0,3,0,4,0,5,0,6,0,7,0,8,0,9,0,10,0,11,0,12,0,13,0,14,0,15,0,16,0,17,0,18,0,19,0,20,0,21
,0,22,0,23,0
B 0
D 0,@,2
D 2,@,3
D 2,@,4
D 3,@,10
D 4,@,6
D 4,@,8
D 5,@,4
D 5,@,3
D 6,a,7
D 7,@,5
D 8,b,9
```

```
D 9,@,5
D 10,a,11
D 11,@,12
D 12,b,13
D 13,@,14
D 14,c,15
D 15,@,16
D 16,@,17
D 16,@,18
D 18,@,20
D 18,@,22
D 19,@,18
D 19,@,17
D 17,@,1
D 20,b,21
D 21,@,19
D 22,c,23
D 23,@,19
T abc
O accepted
T bbabcbb
O accepted
T aaabbabaaabababbabaaaaabc
O accepted
T babc
O accepted
T abcbc
O accepted
T abaaabbaac
O rejected
T bacb
O rejected
T aaaabbabaababaabaabbbc
O rejected
T bbaabccb
O accepted
T baaabbaababbabbabc
O accepted
```

Problem 2

Problem 2a

This language can be recognized by a FSA as shown below.



Problem 2b

According to the pumping lemma, there is no FSA that recognizes this language because all possible prime numbers can not be represented using a regular grammar.

- W_p is the unary expression of all strings, p, that represent prime numbers.
- W_p can be split into three parts. These parts will be X, Y, and X. So, $|W_p| = |XYZ| = |X| + |Y| + |Z|$
- $|XY^nZ|$ is the same as doing Y part of the word W_p n times.
- The amount of '1's inside the word is represented by the following equation. $|XY^nZ| = |X| + n * |Y| + |Z| = (|X| + |Y| + |Z|) + (n-1) * |Y| = |W_p| + (n-1) * |Y|$
- We can make $n=|W_p|+1$. The equation becomes $|XY^nZ| = |W_p| + (n-1) * |Y| = |W_p| + |W_p| * |Y| = (|Y|+1) * |W_p|$
- This implies that XY^nZ is a composite number. For any possible way that we split up W_p (which is any prime number), we can also chose an n that will make XY^nZ be a composite number.
- This implication contradicts the pumping lemma. So, we conclude that the language of all unary strings that represent prime numbers can't be expressed using a regular grammar. Further, a language with irregular grammar can not be represented by an FSA diagram.

Problem 2c

According to the pumping lemma, there is no FSA that recognizes a language of all unary strings that represent prime numbers. This is because this language can not be represented using a regular grammar.

- Suppose that L the language of unary strings that represent composite numbers is regular.
- If L is regular, then the complement of L' which represents all unary strings that are prime numbers must all be regular due to regular languages being closed under complementation.
- We can then apply the pumping lemma to L'.
- We can say p is the pumping length, and begin with $x=0^n$ for a prime number $n >= p$.
- x can be split into three parts: u, v, and w. So, $|uv|<=p$, $|v|>0$, and $uv^kw \in L$ for $k>=0$

- If a=|uw| and m=|v|>0 for k>=0, then for every k>=0 $|uv^kw| = a + km$
- This is an arithmetic sequence beginning at a, with a difference of m from term to term. This sequence must contain a composite number. Since we arrived at a prime number instead of a composite number, this shows that L' is not a regular language. Since we assumed that L is a regular language and showed that L' is not regular, then our assumption is wrong. L must also not be a regular language and can't be represented by a FSA.

Problem 2d

It is not possible to make an FSA for this language. FSAs only allow you to pump one variable and you would have to be pumping forward and backward at the same time in order to check for a palindrome.

Example using pumping lemma:

- Suppose that the language L is the set of all palindromes made with the alphabet $\{a, b, c\}^*$. Let's take a look at w = bbbccbbb which is a palindrome.
- By the pumping lemma, there must exist strings x, y, and z that satisfy all the constraints.
- Thus, we can pick any x, y, and z such that $w = xyz$, $|xy| <= n$, and |y|>=1. Since |xy|<=the value from the pumping lemma, xy is completely contained in the b^n at the start of w. So, x and y consist completely of b's.
- Then, we look at xz. According to the pumping lemma, xz needs to be in the language; however, xz can't be a palindrome. So, the set of palindromes does not satisfy the pumping lemma. Therefore, the set of palindromes made up of the given alphabet can not be regular.
- If this language is not regular, then no FSA can be created for it.

Problem 3

Step 1: Define the problem

- Let D = N = { 0, 1,... }
- Let G be the given CFG. We let P(n) be equivalent to G generates a string { w ∊ $\{a, b\}^*$ | the number of a's in w = the number of b's in w } for every n∊N.

Step 2: Check stopping values and two more

1. The stopping value is 0 which is in D. a^0b^0 : S⟶ ε = ε
    a. Thus, P(0) is true.
2. Check P(1)
    a. a^1b^1 : aSbS = aSbS, S⟶ ε = abS, S⟶ ε = ab
    b. Thus, P(1) is true.
3. Check P(2)
    a. a^2b^2 : S⟶ aSbS = aSbS, S⟶ aSbS = aaSbbS, S⟶ ε = aabbS, S⟶ ε = aabb
    b. Thus, P(2) is true.

Step 3: Prove for all n>s, if P(n-1) is true, then P(n) is true.

- 1st, we must check the case where G generates a string w ∊ $\{a, b\}^*$ | the number of a’s in w > the number of b’s in w for every n∊N.
    - If we assume that P(n-1) is true, then this means that the word generated by P(n-1) has the same number of a’s and b’s in it.
    - We must then look at the transitions. The only transitions from such a state are S⟶ aSbS|bSaS|ε. The transition from S⟶ aSbS creates an equal number of a’s and b’s. The transition from S⟶ bSaS also creates an equal number of a’s and b’s. So does the transition from S⟶ ε. Therefore, regardless of which of these transitions is executed, there will always be an even number of a’s and b’s generated at any time, and no excess a’s will be possible in the final string.
- 2nd, we must check the case where G generates a string w ∊ $\{a, b\}^*$ | the number of a’s in w < the number of b’s in w for every n∊N.
    - If we assume that P(n-1) is true, then this means that the word generated by P(n-1) has the same number of a’s and b’s in it.
    - We must then look at the transitions. The only transitions from such a state are S⟶ aSbS|bSaS|ε. The transition from S⟶ aSbS creates an equal number of a’s and b’s. The transition from S⟶ bSaS also creates an equal number of a’s and b’s. So does the transition from S⟶ ε. Therefore, regardless of which of these transitions is executed, there will always be an even number of a’s and b’s generated at any time, and no excess b’s will be possible in the final string.
- Thus, by the principle of induction, P(n) is true for every n∊N, because it is impossible for a string with the number of a’s > the number of b’s or a string with the # of b’s > the # of a’s to be created.

Step 4: Conclude the proof.

- Since we verified all the requirements for proof by induction, we have shown that for ∀ G a string w ∊ $\{a, b\}^*$ | the number of a’s in w = the number of b’s for every n∊N that is possible.

Problem 4

Problem 4a

$\{a^{n!} | n \in N\},\ alphabet = \{a\}$

- Putting n=1,2,3,4,5…...L={n,2n,6n,24n,120n}
- This language is not in arithmetic progression. FSA or push down automata can’t traverse a language if it is not in arithmetic progression.
- So, the language is neither regular nor context free.
- Answer is not context-free

Problem 4b

$\{a^p | p \in N, p > 0, and\ p\ is\ a\ prime\},\ alphabet = \{a\}$

- Language produced is L={a2,a3,a5,a7,a11}
- This language is not in arithmetic progression. FSA or push down automata can’t traverse a language if it is not in arithmetic progression.

- So, the language is neither regular nor context free.
- Answer is not context-free

Problem 4c

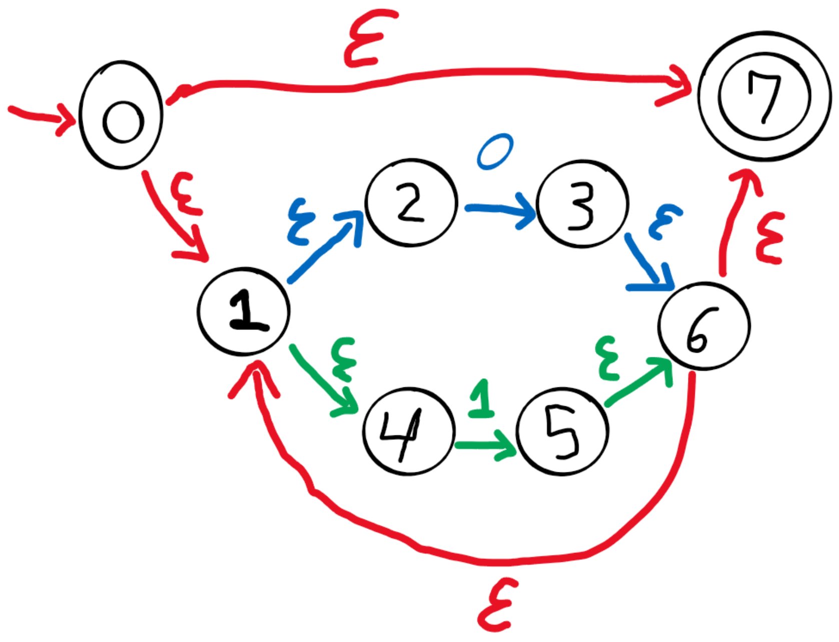
$\{a^{2n+1}b^{3n+2} | n \in N\},\ alphabet = \{a, b\}$

- L={a3b5,a5b8,a7b11,a9b14,...}
- Both a and b are dependent on n. So, we have to parse a and then compare it to b. Since there is this comparison, a FSA will not accept this language (due to the two traversals). This also means it is not a regular language.
- For push down automata, there is a relation between a and b. In the first string there are two extra b's.
- In the next relation, there are three extra b's in the string. In the following relation, there are four extra b's in the string.
- Since there is no pattern in these relations, there is nothing we can follow to push or pop a and b to be accepted by a push down automata. So, this language is also not a context free language.
- Answer is not context-free.

Problem 4d

$\{w \in \{0, 1\}^* | w\ does\ not\ contain\ 01101\ as\ a\ substring\}.$

- L={0,1,00,01,10,11,000,001,010,101,1010,0111,1100,...} any combination of 1's and 0's that does not contain 01101 as a substring
- We can create a FSA that accepts the language which contains 01101 as a substring
- We can complement that FSA by switching the final states to be non-final and the non-final states to be final. This will create an FSA to recognize the language of all 1's and 0's except for ones that contain 01101 as a substring.
- It is known that the complement of a regular language is also regular which makes this language regular.
- So, the answer is regular.



Problem 5

Step 1: Eliminate ε/empty productions.

- N_1 = { T, U, X, R, Y, V, W, Z }. (can reach ε)
- $N_2 = N_1$
- New rules:
  - S→ aXT|YbT|UbZ|UWc|PQT|aX|Yb|bZ|Wc|aT|PQ|a|bT|Uc|Ub.
  - P→ aT|a.
  - Q→ QT|aQ.
  - T→ c|cT.
  - U→ a|aU.
  - X→ a|aX|R.
  - R→ aRb|ab.
  - Y→ b|Yb|R.
  - V→ bVc|bc.
  - W→ Wc|c|V.
  - Z→ b|bZ|V.

Step 2: Eliminate unit productions.

- We will remove the following productions: X→ R, Y→ R, W→ V, Z→ V
- We will add the following productions:
  - X→ aRb|ab.
  - Y→ aRb|ab.
  - W→ bVc|bc.
  - Z→ bVc|bc.

Step 3: Eliminate variables that derive no terminal string.

- N_0 = { a, b, c }.
- N_1 = { a, b, c, S, P, T, U, X, R, Y, V, W, Z }.
- N_2=N_1. So, on variable Q is eliminated.

Step 4: Eliminate variables not reached from the start symbol.

- Since variable P is the only variable that can't be produced starting from S, it must be removed.
- New Rules/Productions =
    - S→ aXT|YbT|UbZ|UWc|aX|Yb|bZ|Wc|aT|a|bT|Uc|Ub.
    - T→ c|cT.
    - U→ a|aU
    - X→ a|aX|aRb|ab.
    - R→ aRb|ab.
    - Y→ b|Yb|aRb|ab.
    - V→ bVc|bc.
    - W→ Wc|c|bVc|bc.
    - Z→ b|bZ|bVc|bc.

Step 5: Present new equivalent grammar G in CNF.

- Vars = { S, P, T, U, X, R, Y, V, W, Z }.
- Alph = { a, b, c }.
- Start = S.
- Rules:
    - S→ aXT|YbT|UbZ|UWc|PQT.
    - T→ c|cT.
    - U→ a|aU.
    - X→ a|aX|aRb|ab.
    - R→ aRb|ab.
    - Y→ b|Yb|aRb|ab.
    - V→ bVc|bc.
    - W→ Wc|c|bVc|bc.
    - Z→ b|bZ|bVc|bc.

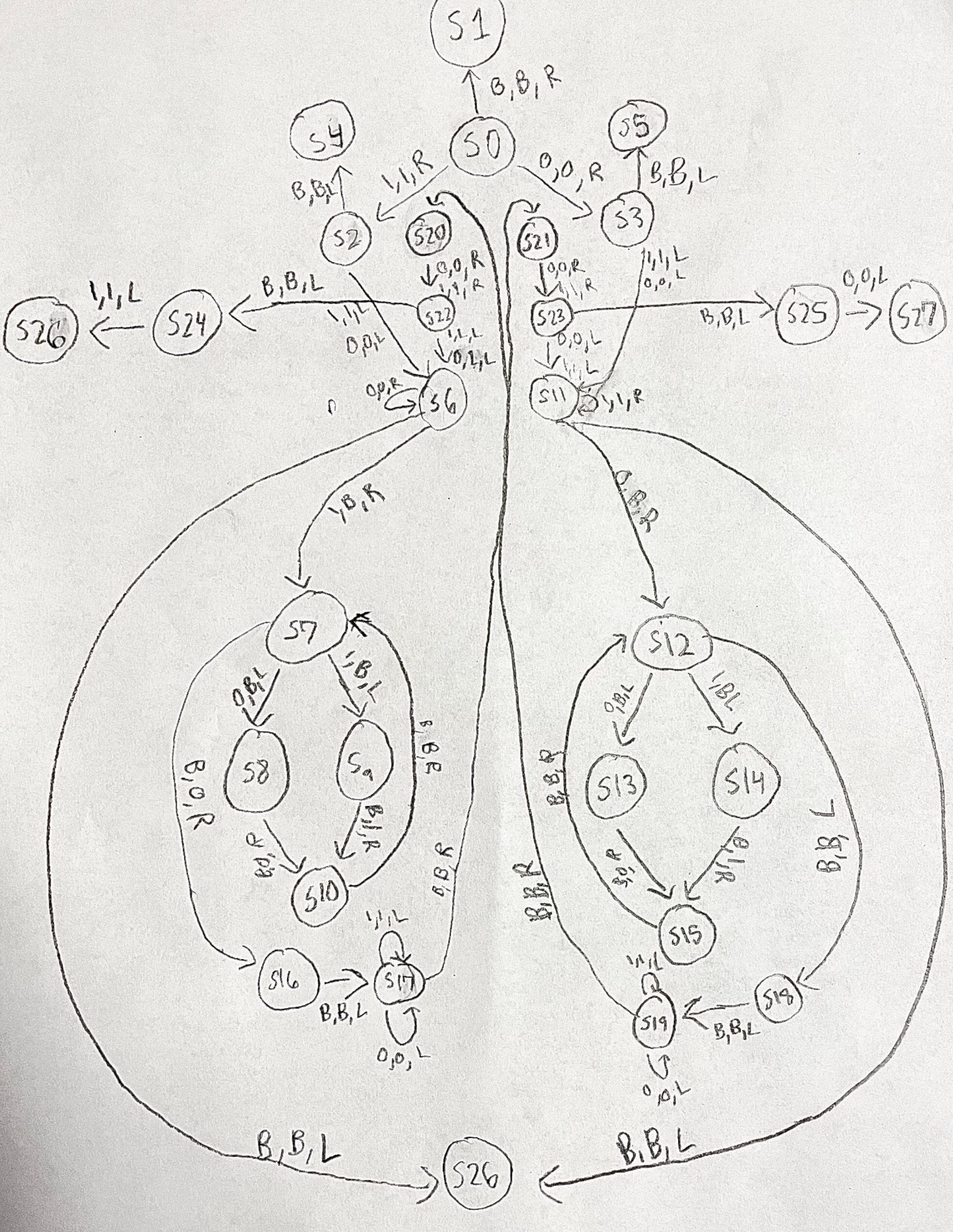
Problem 6

Example 1: 010101

- S0→ S3, S3→ S11, S11→ S12, S12→ S18, S18→ S19, S19→ S21, S21→ S23, S23→ S11, S11→ S26 (Accepting state BBBBBB)

Example 2: 0010100

- S0→ S3, S3→ S11, S11→ S11, S11→ S12, S12→ S18, S18→ S19, S19→ S21, S21→ S23, S23→ S25, S25→ S27


S1
B,B,R
S0
1,1,R
0,0,R
S4
S5
B,B,L
B,B,L
S2
S3
S20
S21
0,0,R
1,1,R
0,0,R
1,1,R
1,1,L
0,0,L
B,B,L
1,1,L
S26
S24
S22
1,1,L
0,0,L
S23
B,B,L
S25
0,0,L
S27
1,1,L
0,1,L
0,0,L
1,1,L
0,0,R
S6
S11
1,1,R
1,B,R
0,B,R
S7
0,B,L
1,B,L
S8
S9
B,B,R
B,0,R
B,0,R
B,1,R
S10
B,B,R
S12
0,B,L
1,B,L
S13
S14
B,B,R
B,0,R
B,1,R
B,B,L
S15
B,B,R
1,1,L
S16
S17
B,B,L
0,0,L
1,1,L
S18
S19
B,B,L
0,0,L
B,B,L
B,B,L
S26

Problem 7

Turning Machine 1:

CS	CC	NS	NC	M
1	β	0	β	L
1	0	0	β	L
1	1	0	β	L

Turning Machine 2:

CS	CC	NS	NC	M
1	β	0	β	L
1	0	0	β	L
1	1	0	β	N

Turning Machine 3:

CS	CC	NS	NC	M
1	β	0	β	L
1	0	0	β	L
1	1	0	β	R

Turning Machine 4:

CS	CC	NS	NC	M
1	β	0	β	L
1	0	0	β	L
1	1	0	0	L

Turning Machine 5:

CS	CC	NS	NC	M
1	β	0	β	L
1	0	0	β	L
1	1	0	0	N