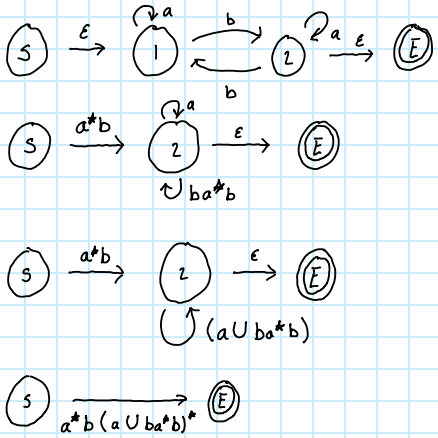
Assignment #2

Monday, February 7, 2022 6:07 PM

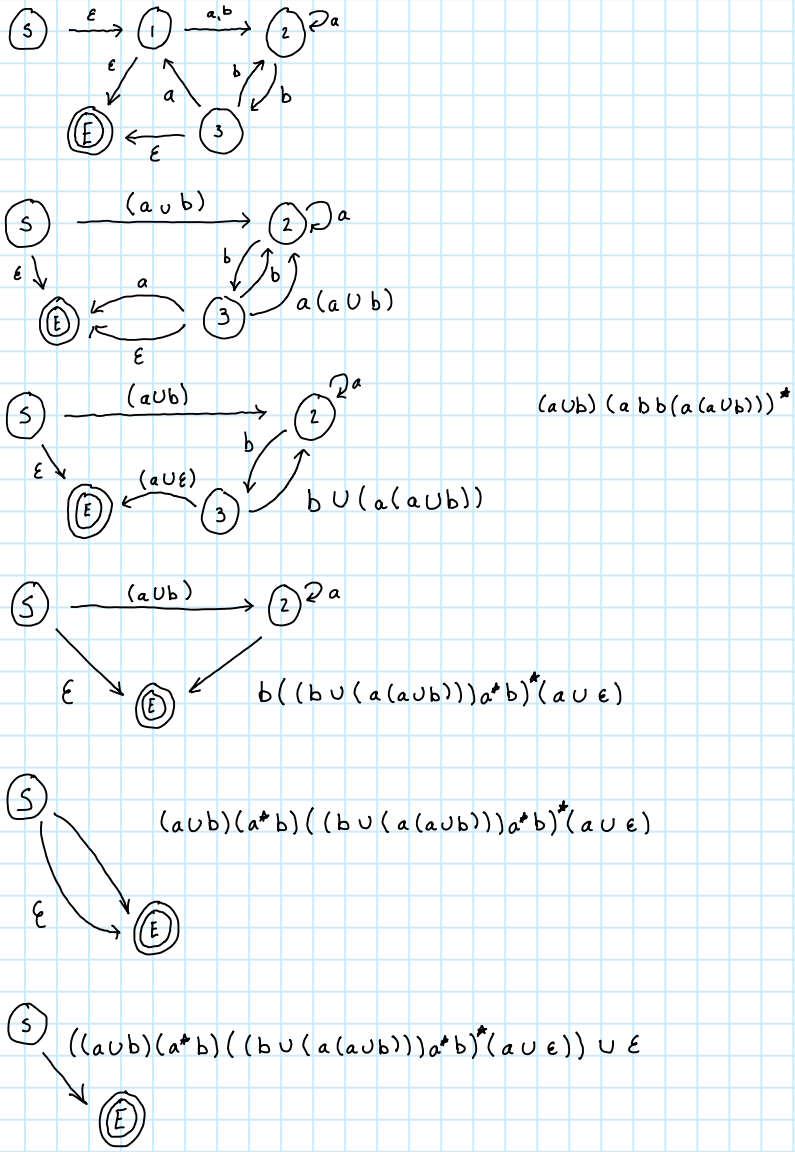
1.21

a.



Reg: $a^*b(a \cup ba^*b)^*$

b.



Reg: $((a \cup b)(a^* b)((b \cup (a(a \cup b)))a^* b)^*(a \cup \epsilon)) \cup \epsilon$

1.47 We will prove this using the Pumping Lemma

We start by assuming *Y* is a regular language and observe the string $w = x_1 \# x_2$ where $x_1 \neq x_2$ and $w \in Y$. We can define $x_1 = 111^p$ and $x_2 = 1^p 111$ where $p > 0$, which means that our string $w = 111^p \# 1^p 111$. This establishes the initial premise of the pumping lemma in that if *Y* is regular, then if $w \in Y$ and at least of length *p*, then we may split it and satisfy three conditions. First, we must split *w* into three parts, which we will do as follows: $x = 11, y = 1, z = \#1111$. Of note is the first condition, of the pumping lemma, which states that for each $i \geq 0, xy^i z \in Y$. However, we can prove this is untrue for our string. For instance, if *i=2*, then $y = 1^2 = 11$, meaning *w* becomes $w = xy^2 z = 1111\#1111$, which is not in language *Y* as it violate the principle that $x_1 \neq x_2$. Thus, as the condition is not true, the pumping lemma is violated and it must mean that *Y* cannot be a regular language.

1.55 We will prove this using the Myhill-Nerode Theorem

We define some infinite set of strings where $X = \{10^n \mid n \geq 0\}$ and $x_n \in X$. For all n, m *where* n, 10^n is distinguishable from all 10^m because there exists some z, $z = +0 = 10^n$, such that:

$$(10^n z \text{ is equivalent to } 10^n + 0 = 10^n) \in B$$
$$(10^m z \text{ is equivalent to } 10^m + 0 = 10^n) \notin B$$

where $B = \{x = y + z \mid x, y, z \text{ are binary inytegers and } x \text{ is the sum of } y \text{ and } z\}$. Since X is an infniite set, it means that B has a infinite index, thus proving it is not regular.

1.59

a. We will prove that *F* is not regular using the pumping lemma

There are three different cases that combine to form *F*. One where *i = 0, i = 1,* and $i \geq 1$, which we will refer to as F_0, F_1, and F_2. We can say that languages F_0 and F_2 are regular, since their accepting states accept any value for the variables of *j* and *k,* with F_2 also accepting any value of *i.* Since we know that regular languages are closed under union and complement (Theorem 1.25 and Theorem 2.42), we can say that $\overline{F_0 \cup F_2}$ is also regular. We can also say that $F_1 = F - (F_0 \cup F_2)$, which can then be altered using our complement to say $F_1 = F \cap \left(\overline{F_0 \cup F_2}\right)$. Applying the same logic of intersection earlier, we can then say that if *F* is regular, then so is F_1.
In F_1 there exists some string *w*, where $w = ab^p c^p$. This establishes the premise of the pumping lemma, as the length of *w* will be greater than whatever integer *p* we choose. We now divide the string into three pieces: $x = a, y = b^p, z = c^p$. We will now pump 0 times, per the first condition of the lemma, and receive the following:
$xy^0 z = a(b^p)^0 c^p = ac^p$
We see that there are more c's than b's, which means that this new string is not a member of F_1 and F_1 must be irregular. As F_1 is a subset of *F*, this therefore means that the string is not within *F,* proving that *F* is not regular.

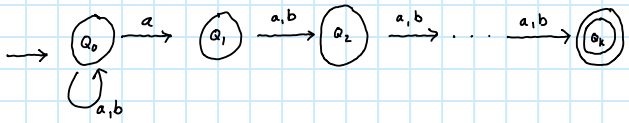
b. We say there is some string *w* in *L* of length at least *p=2*, which then provides the following cases:

- $w = b^m c^n$: We then define the pumping lemma variables *x,y,* and *z* to be as follows: $x = \epsilon, y = b$, $z = b^{m-1} c^n$. We note that $|xy| \leq 2$ and $|y| > 0$, satisfying conditions two and three of the pumping lemma. Finally, we see for all $q \geq 0$, $xy^q z = b^q b^{m-1} c^n = b^{m-1+q} c^n$, which is in *F*
- $w = a^l b^m c^n, i \neq 2, i \geq 1$: We then define the pumping lemma variables *x,y,* and *z* to be as follows: $x = \epsilon, y = a$, $z = a^{l-1} b^m c^n$. We note that $|xy| \leq 2$ and $|y| > 0$, satisfying conditions two and three of the pumping lemma. Finally, we see for all $q \geq 0$, $xy^q z = a^q a^{l-1} b^m c^n = a^{l-1+q} b^m c^n$, which is in *F*.
- $w = aab^m c^n$: We then define the pumping lemma variables *x,y,* and *z* to be as follows: $x = \epsilon, y = aa$, $z = b^m c^n$. We note that $|xy| \leq 2$ and $|y| > 0$, satisfying conditions two and three of the

pumping lemma. Finally, we see for all $q \geq 0$, $xy^q z = (aa)^q b^m c^n$, which is in *F*.

Therefore, we have proven that for all possible cases in *p = 2*, we have shown *F* does not contradict the pumping lemma and is a regular language.

1.60



A NFA is described as follows $N = (Q, \Sigma, \delta, q_0, F)$:

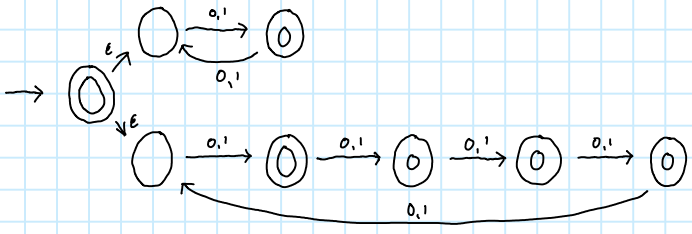
- *Q* = Set of states in the NFA = $\{q_0, q_1, q_2, \ldots q_k\}$
- Σ = Set of the alphabet = {a,b}
- q_0 = Start state = $\{q_0\}$
- *F* = Set of the final states = $\{q_k\}$
- δ = Transition function works as follows:
  - $\delta(q_i, a) = \begin{cases} \{q_0, q_1\} \text{ if i} = 0 \\ \{q_{i+1}\} \text{ if } 0 < i < k \\ \emptyset \text{ if } i = k \end{cases}$
  - $\delta(q_i, b) = \begin{cases} \{q_0\} \text{ if i} = 0 \\ \{q_{i+1}\} \text{ if } 0 < i < k \\ \emptyset \text{ if } i = k \end{cases}$
  - $\delta(q_i, \epsilon) = \emptyset \forall i$

1.61 If a DFA enters two different states after reading two strings, which we will name *xz* and *yz*, then we can assume that it also enters two distinct states after reading *x* and *y* individually. We will then define *x* and *y* as strings of length *k* such that $x = x_0 x_1 \ldots x_k$ and $y = y_0 y_1 \ldots y_k$. We will then let *i* be some position such that $x_i \neq y_i$. One of the strings will contain an *a* in its i^{th} position and the other string will contain a *b* in its i^{th} position. We then consider our *z* to be $z = b^{i-1}$. With the combined strings of *xz* and *yz,* one of them has the k^{th} bit from the end as an *a*. The total number of strings of length *k* over the alphabet is 2^k. Thus, we would need 2^k states to distinguish 2^kstrings, proving the DFA that recognizes C_k has at least 2^k states.

1.64

a. If A is not empty, then there must be a start and accept state , which we'll call q_0 and q_a. We will call *x* the string that is accepted by *N* that gives the shortest path from q_0 to q_a,which is of length *n*. This means there are *n+1* states including the start and end states in this path. All the states must be distinct, otherwise there would be a shorter path. Since there can only be *k* states and we know *n* is the length of the shortest path, we know then that $n \leq k$. Thus, *A* contains some string of at least length *k.*

b.



We will define some NFA *N* that has as many states as it does transitions, and say that *A* is some language recognized by *N*. We can see a few things about language *A*: *N* accepts the empty string, it will reject any string of length divisible by 2 or 5. Thus, the inverse of this language *A* $(\bar{A})$ is all non empty strings divisble by 10, with the shortest string of length *10 > K*. This a contradiction of part a, where we have replaced *A* with $\bar{A}$.

c. As per theorem 1.39, we can covert any NFA into a corresponding DFA. We can then convert our NFA *N* into a DFA *X* that also recognizes language *A*. We then swap the accept and non-accept states of *X* to obtain the DFA $\bar{X}$ that accepts $\bar{A}$, which has 2^k states, where k is the amount of states in N, as seen in Lecture 4. Using part a, we then see that if we replace $\bar{X}$ with *N*, then we can see that $\bar{A}$ is non emptied, and thus must contain a string of length at most 2^k.

d. Using part b, we can generate a bound near to that of an exponential. We first let $2 \leq Y_1 \leq Y_2 \leq \cdots \leq Y_n \leq \sqrt{k}$ be all the prime numbers between 1 and k. Then, for each *Y*, we can construct a DFA X_i with Y_i states that will reject only strings of length Y_i. We then construct an NFA *P* by unioning all X_i's as follows:

First we create a new starting state that has a ϵ transition to each other X_i starting states using Kleene's Theorem, except we will also mark this new start state as an accepting state. This means the large machine *P* will have $1 + Y_1 + Y_2 + \cdots Y_j \leq k$ states. We see then that *P* rejects a string if and only if *A* is nonempty and the string length is disivisble by $Y_1 Y_2 \ldots Y_j$. Therefore, like in part b, the shortest string rejected by *P* has length $Y_1 Y_2 \ldots Y_j$. By prime number theorem, there are approximately $\ln n$ prime numbers between 0 and *n*. In this case, we can say then that $j \approx \frac{1}{2}\ln k$ and since there are $\frac{1}{4}\ln k$ primes between 0 and $\sqrt[4]{k}$, then there are $j - \frac{1}{4}\ln k \approx \frac{1}{4}\ln k$ primes between $\sqrt[4]{k}$ and $\sqrt{k}$.

Therefore $Y_1 Y_2 \ldots Y_j \geq \left(\sqrt[4]{k}\right)^{\frac{1}{4}\ln k} = k^{\frac{1}{16}\ln k}$

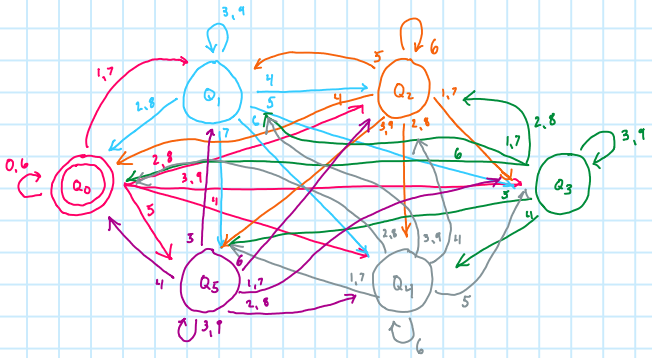
1.18

a. [0-9]*(50|00)

b. [0-9]*(([13579]20)|([02468]40)|([13579]60)|([02468]80)|([2468]00))

c. $(10)0^*$

d.



e. Myhill Nerode, sequence of nines

f. We will prove irregularity using the Pumping Lemma. We first assume that L_f is regular. We choose the string $w = 4096$ and divide as follows: $x = 4, y = 09, z = 6$ for $p = 3$. We confirm the latter two conditions of the pumping lemma, namely that $|y| > 0 \; and \; |xy| \leq 3$. However, for the first condition, $\forall\, i \geq 0, xy^i z \in L_f$, is not true. We can see this with $i = 2, where \; 4(09)^2 6 = 409096 \notin L_f$. Thus, this is a contradiction to our starting statement, and L_f is not regular

g.

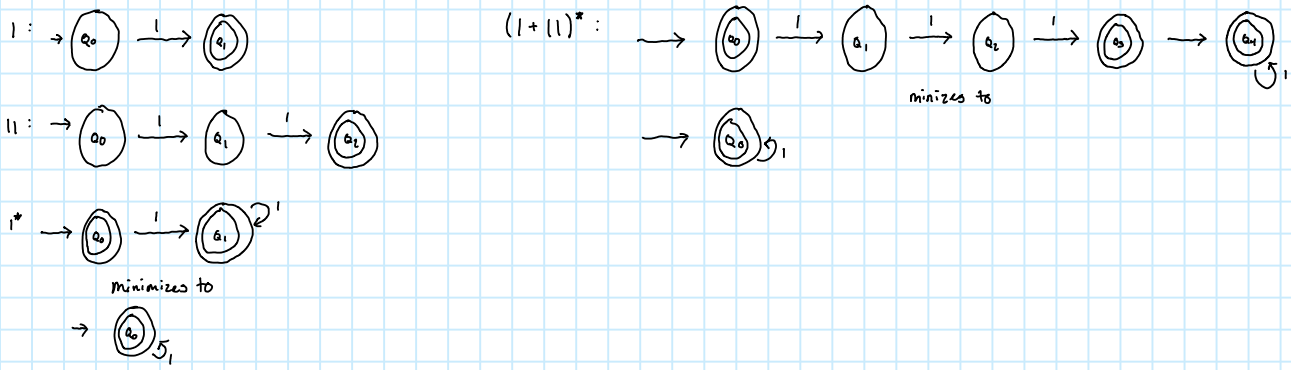
h. We will prove irregularity using the Pumping Lemma. We first assume that L_h is regular. We choose the string w = 122/35 and divide as follows: x = 1, y = 22, z =/35 for p = 3. We confirm the latter two

string $w = 122/35$ and divide as follows: $x = 1, y = 22, z = /35$ for $p = 3$. We confirm the latter two conditions of the pumping lemma, namely that $|y| > 0$ and $|xy| \leq 3$. However, for the first condition, $\forall\, i \geq 0, xy^i z \in L_f$, is not true. We can see this with $i = 2$, where $1(22)^2/35 = 12222/35 \notin L_h$. Thus, this is a contradiction to our starting statement, and L_h is not regular

1.111 Minimizing a regular expression is a PSPACE-hard problem, meaning there is no general method that could be applicable to apply to every regex *R*. However, a trivial solution could then be when given such a regex *R*, you enumerate all of the possible regular expressions (given the symbols and operators in *R)* in ascending order. You then test each one, in ascending order, to see if its equivalent NFA is equivalent to *R*'s NFA. For instance, let us say we have some regular expression $R = (1 + 11)^*$ and wish to find some minimization. We then enumerate all the possible expressions, up to *R:*

1. 1
2. $11, 1^*$
3. $111, 11^*, 1^*1, (11)^*, 1 + 1$
4. *All expressions of length 4*
5. *All expressions of length 5, including*

Then we create an NFA of each in ascending order, compute an equivalent DFA, then use part(b) of the Myhill-Nerode theorem to construct minimal DFA's for both. We know that two regular expressions are equivalent when they have the same minimal DFA.



Thus, we have found the most minimal regex, R_{min}, as $1^* = (1 + 11)^*$.