CS 480 Syllabus and Portfolio Winter 2019

Bryan Muller

February 1, 2019

Portfolio

Course Tracker

You are required to track your progress in the course using this table.

Note: Currently, you see full credit for week one's work. (✓ means yes. Blank means no.) Update the table for week 2 and all subsequent weeks each class day and week during the semester.

Week	CRU	PFP	CDL	SAQ	PAQ	CDL	PPL
1	✓	✓	✓	✓	✓	✓	100%
2	✓	✓	✓	✓	✓	✓	100%
3	✓	✓	✓	✓	✓	✓	85%
4	✓	✓	✓	✓	✓	✓	100%
5							
6							
7							
8							
9							
10							
11							
12							
13							

This is an honest and true record of my work for this course.

Signature: _____

Grade Claims

On the week indicated, bring this updated document to my office and make your claim.

Claim Week	Grade Claim	Instructor Grade	Adjusted Grade
5	A-		
9			
13 - 14			

Evidences

Fill in your evidences here each week to build your portfolio. The number of pieces of evidence are determined by you. However, the more you have the better off you will be.

Week 1

Initial Take Home Exam

I have included another copy of my Initial Take Home Exam.

Week 2

Selections from Chapter 2 Exercises

Exercises 2.1.3: Language Operations

1. No, it cannot. Our definition of an alphabet is *finite* and *non-empty* set of symbols. While *Nat* may be a *non-empty* set, it's cardinality is $\aleph_0$ (infinite).

2. *symbols* = {"H", "e", "l", "o", " ", "t", "h", "r", "!" } The smallest alphabet underlying this string would be the set *symbols*. It meets our definition of an alphabet; a finitie and non-empty set.

3. While perhaps not every palindrome string is initially created using a concatenation of a string with its reverse, any palindrome could certainly be defined that way.

Exercises 2.1.4: Zero, One, Exp

1. s = *abacaca* number of a's = 4 number of b's = 1 number of c's = 2

s^4 = abacacaabacacaabacacaabacaca number of a's = 4 * 4 = 16 number of b's = 1 * 4 = 4 number of c's = 2 * 4 = 8 number of d's = 0 * 4 = 0

2. The *One* element would be the Universal set (or as we defined it, *Nat*). The intersection of any subset with it's parent set would return just the subset, aka the *One* element in multiplication.

s $\cap$ *Nat* = s $\cong$ s $\cap$ *One* = s

The *Zero* element would be the empty set (). The intersection of any set with the empty set returns the empty set. This behaves the same way as the *Zero* element in multiplication.

s $\cap$ = $\cong$ s $\cap$ *Zero* = *Zero*

Exercises 2.2: Languages

1. As we've defined language, it must equal the empty set () or possibly infinite set of finite strings which must meet the constraint /$a^i b^j$: i,j $\geq$ 0, and i $<$ j.

This means our language either has *no* strings (ϵ is a string, albiet empty) OR the string must match the contraints. The constraints specify that the

number of *i's* and *j's* must be greater than or equal to zero AND that there are less *b's* than *a's*.

For those constraints to be valid, there must always be at least 1 *b*, meaning that ϵ would never be a valid string in our language.

2. For ϵ to be a valid string in this language, we would need to modify second part of the condition. If we change the condition to be $a^i b^j$ *: i,j* $\geq$ *0, and* $i \leq j$ ϵ would be a valid string in our language (note: change $i < j$ to $i \leq j$)

Exercises 2.2: Languages- Python

1.

```
substrings_s = { "a" * i + "b" * j + "c" * k for i in range(2) for j in range(2) for k
print(substrings_s)
```

```
(-flatten (loop for i from 0 to 1
      collect (loop for j from 0 to 1
          collect (loop for k from 0 to 1
              collect (concat (make-string i ?a) (make-string j ?b) (make-string k ?c)
```

(c b bc a ac ab abc)

2.

```
print({"(" * i + ")" * j for i in range(6) for j in range(6) if i == j})
```

```
(-flatten
    (loop for i from 0 to 5
        collect (loop for j from 0 to 5
            if (= i j)
            collect (concat (make-string i ?\() (make-string j ?\)))))))
```

(() (()) ((())) (((()))) ((((())))) ((((()))))))

3.

```
p = "abcde"
q = "fghij"

print((p+q)[::-1])
print(q[::-1] + p[::-1])
```

On the left side of the equation, we are adding the strings p and q and then reversing that concatenated string. On the right side of the equation, we are reversing the strings p and q and then adding them together. This works due to the commutative property of reverse function. In integer arithmetic, we can see this same property like so.

Given an three integers a, b, and c: (a + b)(c) = (a * c) + (b * c)

```
a = 5
b = 3
c = 2

print((a+b)*(c))
print((a * c) + (b * c))
```

4. L_1 describes a language that contains pairs of opposing balanced parantheses and the empty string. By opposing balanced parantheses, I mean that the string is equally split with all of the opening parentheses on the left side of the string, and the closing parentheses on the right (e.g. '()' '(())' '((()))' '(((())))' '((((()))))')

L_2 describes all strings that contain a balanced set of parentheses. This means there always an opening parenthesis which precedes a matching closing parenthesis. There may be nested pairs of opening and closing parentheses, but every opening parenthesis has a matching closing parenthesis and vice versa. (e.g. '()' '(())' '(()())')

$L_1 \subset L_2$.

L_3 describes all strings with an equal number of opposing parentheses. The opening and closing parentheses are not required to be balanced. The string is valid as long as there is the same number of opening as closing parentheses. (e.g. '()' '(())' '()()()' '))()((')

$L_1 \subset L_2 \subset L_3$

Exercises 2.2.5: Languages(review)

1. $\Sigma = \{0,1\}$
a. $\Sigma^2 = \{00, 01, 10, 11\}$
b. $\Sigma^0 = \{\epsilon\}$ (see pg 24)
c. $\Sigma^1 = \{0, 1\}$
d. $\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

2. $M = \{0, 10\}$
a. $M^2 = \{00, 010, 100, 1010\}$
b. $M^0 = \{\epsilon\}$ (see pg 24)
c. $M^1 = \{0, 10\}$
d. $M^3 = \{000, 0010, 0100, 01010, 1000, 10010, 10100, 101010\}$

3.

```
(defun l-1 (n)
   (-flatten
      (loop for i from 0 to n
            collect (loop for j from 0 to n
                          if (= i j)
                          collect (concat (make-string i ?\() (make-string j ?\)))))))))

(l-1 3)
```

a. ϵ, '()', '(())'
b. '()()()'
c. '()'.

We previously established that $L_1 \subset L_2 \subset L_3$, so we would need to take the smallest member of L_1. We could go with ϵ, but that seemed a little too much of a given :)

Exercises 2.2.6

1. a. $L_1 \cup L_2$ would match L_2 because $L_1 \subset L_2$
b. $L_1 \cup L_3$ would match L_3 because $L_1 \subset L_3$
c. $L_1 \cap L_2$ would match L_1 because $L_1 \subset L_2$

2. a. star({0, 1}, 2) = {ϵ, 0, 1, 00, 01, 10, 11}

b. star({0, 1}, 0) = {ϵ}

c. star({0, 1}, 1) = {ϵ, 0, 1}

d. star({0, 1}, 3) = {ϵ, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111}

e. star({0, 10}, 2) = {ϵ, 0, 10, 00, 010, 100, 1010}

f. star({0, 10}, 0) = {ϵ}

g. star({0, 10}, 1) = {ϵ, 0, 10}

h. star({0, 10}, 3) = {ϵ, 0, 10, 00, 010, 100, 1010, 000, 0010, 0100, 01010, 1000, 10010, 10100, 101010}

i. star({0, 1, 00, ϵ}, 2) = {ϵ, 0, 1,00, 01, 000, 10, 11, 100, 001, 0000}

j. star({0, 10}, 0) = {ϵ}

k. star({0, 10}, 1) = {ϵ, 0, 10}

l. star({0, 10}, 3) = {ϵ, 0, 10, 00, 010, 100, 1010, 000, 0010, 0100, 01010, 1000, 10010, 10100, 101010}

m. It is $\sum_0^n 2^n$. The size of each set which results from $L^n = 2^n$. Because *star* unions these sets together, we some the cardinalities of each exponentiated set to find the total number of items

n. An arbitrary string that is finite/infinite and each symbol in the string is a combintation of the characters 1 or 0

Exercises 2.3: Slippery Concepts

1. $L_E = \{0^{2i} : i \geq 0\}$ each string character will be 2i characters long. 2 times any number will always result in an even number, thus every string produced by this constraint will also be even

2. $(00) = 0^2$ so $(00)^i = 0^{2i}$ thus $\{0^{2i} : i \geq 0\} = \{(00)^i : i \geq 0\}$

3. L_O - $\{0^{2i+1} : i \geq 0\}$ contains all strings with odd number of characters. So union would be all strings of a finite/infinite number of zeros. Is that what $\{0\}^*$ represents?

4. A language which contains all strings beginning with a finite/infinite number of zeros, and ending with a finite/infinite number of ones.

5. a. They are equal. The only thing that is different is the variable used to represent the exponenent.

b. They are equal. The first constraint creates a finite/infinite list of zeros, which is multiplied together with a finite/infinite list of ones formed by the second contstraint. This is an equivalent definition.

c. They are equal. Both sets create words with a finite/infinite number of preceding zeros followed by a finite/infinite number of ones. Both sets have a cardinality of $\aleph_0$, so they are equal.

d. They are not equal. L_4 does not contain ϵ

e. Yes. This is a union of L_3 and $\{\epsilon\}$ (which should already be in L_3) so we've already established that they are the same.

f. Yes. Although the variable names have changed, they are fixed to be equal to each other, rendering the same result

6. No. as L_7 is defined, i and j can be different values, which allows there to be unequal numbers of ones and zeros. $\{0^i : i \geq 0\}\{1^i : i \geq 0\}$ is using the samve value, which means there will always be an equal number of ones and zeros.

7. a. No, it is not the true complement of L_6

b. 10, 110, 11100000, 1010101

c. Any string with a 1 coming before a 0. Any string with alternating 1's and 0's. $[\{1^j0^i\}: i, j > 0]$ $[\{w : w \in (\{0\}^i \{1\}^j)^n, i,j,n > 0)$

d. No, $L_8 \subset$ 'L$_6$

Week 3

I was quite ill this week which is why I was not able to complete all of the exercises. Once my schedule has settled down a bit, I plan on going back and completing the ones I skipped. Even though I did not complete all of the exercises, I did leave the chapters confident I understood all of the material well enough to apply it in future problems.

Selections from Week 3 exercises:

Exercise 3.2: Star Properties

1. { ϵ, "((((((", ")))))))", "()()()()" ")))(((", "()())()(" }

2. Yes. {0}* indicates the set of all strings of only repeating zeros. Concated to that is {1}* which is the set of all strings of only repeating ones. This is equivalent to the definition of L_7

3. The Empty Language where L_1 * = Unit language
The Unit Language, where L_2 * = Unit language

4.

- Languages in English L_{P0}: All binary strings.

  L_{P1}: All binary strings which are palindromes.

  L_{P2}: All binary strings with some word and it's reverse split by either a 1,0, or empty string.

  L_{P3}: All binary strings made up by word and it's reverse split by either a 1 or a 0.

  L_{WW}: All binary strings made up of a word and it's copy without any modification.

- Solutions Context-free:

  L_{P1}: It produces the language of all palindromes over the alphabet {0,1}, which is context-free

  Context-sensitive:

  L_{ww}: Produces the language of all words with a pattern of 0's and 1's up to some length which is then followed by a carbon copy of the same pattern without any reversal.

L_{P2}: Produces the language of a word w followed by a 1,0, or empty string which is in turn followed by the reverse of w

L_{P3}: Produces the language of a word w followed by a 1,0 which is in turn followed by the reverse of w.

5. a. Yes. L_E defines the language containing all strings of a repeated even number of 0's L_O defines the language containing all strings of a repeated odd number of 0's. $L_E \cup L_O$ would then contain all strings of repeated 0's. This is also the definition $\{0\}^*$.

b. Yes. L = LL indicates that a language is the same when concatenated with itself. This would be possible with L_E. Concatenating strings of even length will result in even lengthed strings. L_E contains all even lengthed strings of repeated 0's, so it would equal itself when concatenated with itself.

c. I believe that $L_E = L_E^*$. Both contain the empty string, and concatenating two strings of an even number of zeros will result in another string of even zeros.

d. No.

i. $\{\}$

ii. $\{\epsilon\}$

e. No. L_E^* would only contain strings of zeros which are even in length.

f. No.

6. Claim $L^* = L^{**}$

For every language M, $M \subseteq M^*$, thus $L^* \subseteq (L^*)^*$.

If $w \in (L^*)^*$ then $w = w_1 \dots w_x$ for some $w_1 \dots, w_x \in L^*$.

Then for each i, $w_i = w_{i,1} \dots w_{i,x}$ where $w_{i,j} \in L$.

Then $w = w_{1,1} \dots w_{1,x_1} \dots w_{x,1} \dots w_{x,xx} \in L^*$

Therefore, $(L^*)^* \subseteq L^*$

This can therefore be represented as $L() = L^*$

Exercise 3.4.1: Language Puzzles

1. a. L_x is the subset of {a,b,c}* where each $s \in L_x$ has the same number of a, b, and c, and is arranged in alphabetical order.

   b. $L_x = \{a^i b^i c^i : i > 0\}$

   c. L_y is the subset of {a,b,c}* where each $s \in L_y$ begins with 0 or more c, followed by 1 or more a or b, followed by 0 or more c, followed by 1 or more a or b, and ending with 0 or more c.

Exercise 3.5: Homomorphism

1. Yes. It meets both conditions. The reversal of ϵ is ϵ. And given strings a and b, rev(ab) = rev(a)rev(b).

2. No. function f would not meet condition two. If f(ab) = c and f(a)f(b) = de, then f(ab) $\neq$ f(a)f(b) so it is not a homomorphism.

Exercise 3.6: Lexicographic Order

First Python, then elisp :)

```
from itertools import product

def lexlt(s, t):
    if (s==""):
        return True
    if (t==""):
        return False
    if (s[0] < t[0]):
        return True

    return (s[0] == t[0]) & lexlt(s[1::], t[1::])

L1 = {"abacus", "bandana", "pig", "cat", "dodo", "zulu", "physics"}
L2 = {"dog", "zebra", "zzxyz", "pimento"}

def list_pairs(L1, L2):
    prod = list(product(L1, L2))
    filtered_pairs = set(filter(lambda s: lexlt(s[0], s[1]), prod))
    for i in filtered_pairs:
        print(i)

list_pairs(L1, L2)
```

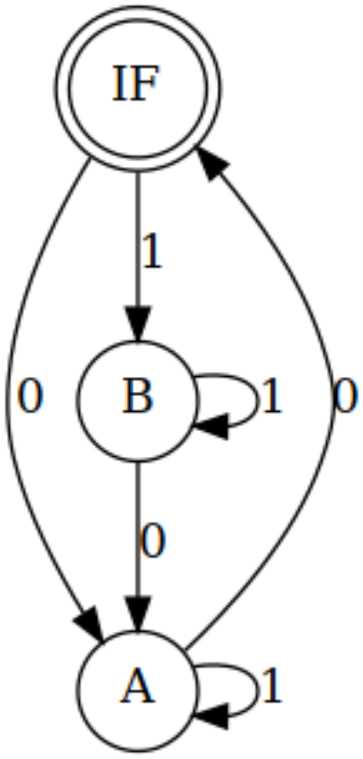
In ELISP!!!! :)

```
(defun cartesian-product (x y)
  "Produces the Cartesian product of two lists"
  (mapcan (lambda (x-item)
            (mapcar (lambda (y-item)
                      (if (listp x-item)
```

```
                        (append x-item (list y-item))
                      (list x-item y-item)))
                y))
          x))

(defun list-pairs (L1 L2)
  (-filter (lambda (pair)
             (string-lessp (car pair) (cadr pair)))
           (cartesian-product L1 L2)))

(list-pairs
   '("abacus" "bandana" "pig" "cat" "dodo" "zulu" "physics")
   '("dog" "zebra" "zzxyz" "pimento"))
```

Exercise 4.2: DFA Basics

1.



2. State table

State	Input	Next State
I	0	A
I	1	F
A	0	I
A	1	I
F	0	1
F	1	1

It is not so simple as the string must end with a 1. Yes, to exit the state machine the string must end with a 1, but there are also rules regarding the number of zeros. The string must contain no, or 2+ zeros. The string may not contain only one zero.

I don't think it is possible. There would need to be a way to enforce that if a 0 is entered, at least on more is entered as well. I don't think there is a way to do that with only a two state DFA.

Exercise 4.7.1: Regular or not?

1. $L_{br} = \{ \{^i\}^i : i \geq 0\}$

All strings in L_{br} must have some number of opening brackets followed by an equal number of closing brackets.

- Steps If L is a regular language, then L has a pumping length P such that any string S where $|S| \geq P$ may be divided into 3 parts S = xyz such that the following conditions must be true:

```
1. xy^{i}z \in L for every i \ge 0
2. |y| \gt 0
3. |xy| \le P
```

Prove that a language is not Regular with pumping Lemma:

1. Assume that L is regular
2. It has to have a pumping length P
3. All strings longer than P can be pumped $|S| \geq P$
4. Now find a string 'S' in L such that $|S| \geq P$
5. Divide S into x,y,z
6. Show that $xy^iz \in L$ for some i

7. Consider all ways that S can be divided into x, y, z.

- Proof setup Proof:

  Assume L_{br} is regular with pumping length P.

  let $S = \{^{P}\}^{P}$

  let P = 3

  If P = 3, then S = '{{{}}}'

  Case 1: y contains only '{' x = '{', y = '{{', z = '}}}'

  Case 2: y contains only '}' x = '{{{', y = '}}', z = '}'

  Case 3: y contains both '{' and '}' x = '{{', y = '{}', z = '}}'

- Case 1 let i = 2 $xy^{i}z \rightarrow xy^{2}z$ then $\rightarrow$ '{{{{{}}}}'

  - Condition 1 $xy^{i}z \in L$ for every $i \geq 0$

    This string does not belong to L_{bc} as there are more opening brackets than closing brackets.

    if $L_{br} = \{ \{^{j}\}^{j} : j \geq 0\}$ and $S = xy^{2}z =$ '{{{{{}}}}', then S is not $\in L_{bc}$

  - Condition 2

    $$y \quad > 0$$

    $$y \quad = 4. \text{ This condition is met}$$

  - Condition 3

    $$xy \quad \leq P$$

    $$xy \quad = 5$$

    P = 3

    This condition is not met

- Case 2 let i = 2 $xy^{i}z \rightarrow xy^{2}z$ then $\rightarrow$ '{{{}}}}}'

  - Condition 1 $xy^{i}z \in L$ for every $i \geq 0$

    This string does not belong to L_{bc} as there are more closing brackets than opening brackets.

    if $L_{br} = \{ \{^{j}\}^{j} : j \geq 0\}$ and $S = xy^{2}z =$ '{{{}}}}}}', then S is not $\in L_{bc}$

- Condition 2

$$\begin{aligned} y &> 0 \\ y &= 4 \end{aligned}$$

This condition is met

- Condition 3

$$\begin{aligned} xy &\leq P \\ xy &= 5 \end{aligned}$$

$P = 3$

This condition is not met

- Case 3 let i = 2 $xy^iz \rightarrow xy^2z$ then $\rightarrow$ '{{{}{}}}'

- Condition 1 $xy^iz \in L$ for every $i \geq 0$

This string does not belong to L_{bc} as it does not follow the pattern of any number of opening brackets followed by the same number of closing brackets.

if $L_{br} = \{ \{^j\}^j : j \geq 0\}$ and $S = xy^2z =$ '{{{}{}}}', then S is not $\in L_{bc}$

- Condition 2

$$\begin{aligned} y &> 0 \\ y &= 4 \end{aligned}$$

This condition is met

- Condition 3

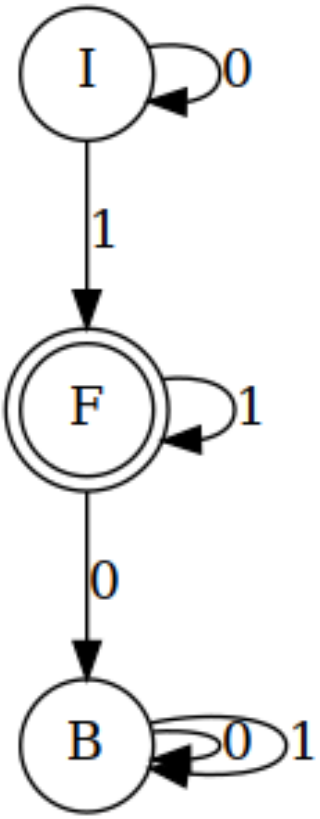
$$\begin{aligned} xy &\leq P \\ xy &= 5 \end{aligned}$$

$P = 3$

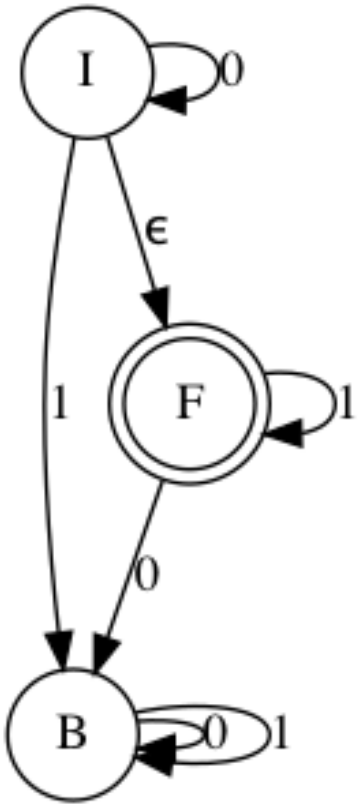
This condition is not met

- Conclusion For every given partition of xyz, all three conditions of a regular language are not met. Therefore, L_{bc} is not a regular language by proof of contradiction using the pumping lemma.

2.



3.



Exercise 4.9

1. If L is regular than that implies that there is a natural number N such that for any string w ∈ L where w is at least length of N, we must be able to read out w as hmt, were h and t are strings of aribitrary length and m is length N and m can be split into strings xyz where y is non-empty and xy is confined to the first N steps of m and furthermore, for all i≥0, xy^i z ∈ L must be true.

2. Original:

```
\exists N \in Nat:
\forall w \in L : [|w| \ge N \rarr
               \exist x,y,z \in \Sigma^* :
                    w = xyz
```

```
                \land   |xy| \le N
                \land   y \ne \epsilon
                \land   \forall i \ge 0 : xy^{i}z \in L ]
```

Negated Condition:

```
\forall N \in Nat:
\forall w \in L : [ |w| \ge N \rarr
                \forall x, y, z \in \Sigma^*:
                     w = xyz
                \land    |xy| \le N
                \land    y \ne \epsilon
                \land    \exist i : xy^{i}z \notin L.
```

Week 4

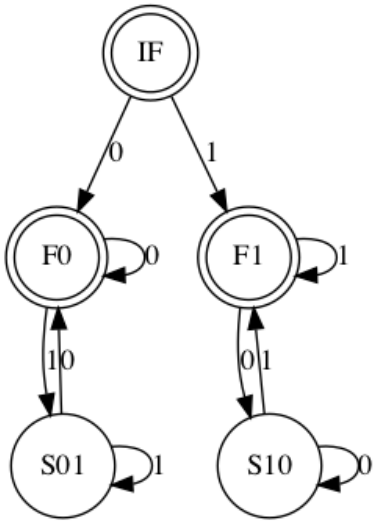
A lot of this week's work was in Jove. I tried to include as much evidence for the work I did in Jove without actually importing the notebooks. This is shown in the copied Python code and imported DFA images for some of the exercises.

Selections from Week 4 exercises:

Exercise 5.1.1: Equal Change DFA

1. It is missing the fact that there must be a strict equal number of transitions between $0 \to 1$ and $1 \to 0$. Not every string which belongs to the alternative definition also belongs to the original definition. Also ϵ is not included in the second language.

2.



Test Strings

String	In	Out	Correct
010	✓		✓
10101	✓		✓
0111		✓	✓
10100		✓	✓

This DFA handles the case of the empty string, strings of only 0's or 1's, and forces there to be a balanced number of $1 \to 0$, $0 \to 1$ switches based on

the number of states required to pass through to get back to a finish state once a switch is made.

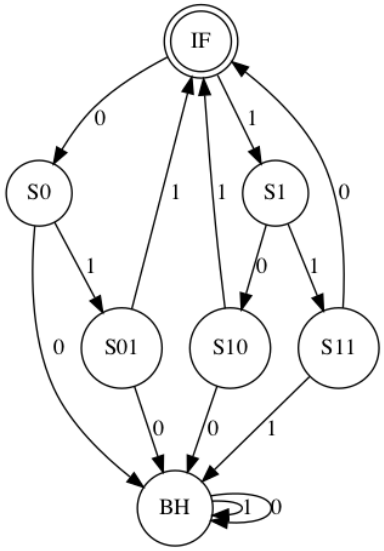
Exercise 5.2.1: Block-of-3 DFA

1.

State	to	New State
S	0	S0
S0	0	BH
S0	1	S01
S01	0	BH
S01	1	S
S	1	S1
S1	0	S10
S10	1	S
S10	0	BH
S1	1	S11
S11	0	S
S11	1	BH

```
from graphviz import Digraph

d = Digraph("5.2.1.1", filename='5.2.1.1.gv', engine='dot', format='png')

d.attr('node', shape='doublecircle')
d.node('IF')

d.attr('node', shape='circle')
d.node('BH')
d.node('S0')
d.node('S1')
d.node('S01')
d.node('S10')
d.node('S11')

d.edge('IF', 'S0', label='0')
d.edge('S0', 'BH', label='0')
d.edge('S0', 'S01', label='1')
d.edge('S01', 'BH', label='0')
d.edge('S01', 'IF', label='1')
```

```
d.edge('IF', 'S1', label='1')
d.edge('S1', 'S10', label='0')
d.edge('S10', 'IF', label='1')
d.edge('S10', 'BH', label='0')
d.edge('S1', 'S11', label='1')
d.edge('S11', 'IF', label='0')
d.edge('S11', 'BH', label='1')
d.edge('BH', 'BH', label='1')
d.edge('BH', 'BH', label='0')
d
```



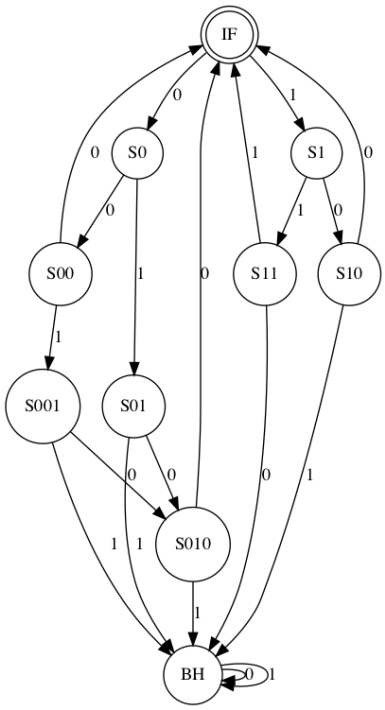
Treat every string as if it is a 3 bit word. We know that the valid 3 bit strings are 011, 110, 101. Make paths for these strings, and send anything else to the black hole.

2. The complement of L_{b3} would be 'L_{bc} = { x: Every contiguous block of 3 bits in x must have $>$ or $<$ than 2 1s.

State	New State	to
IF	S0	0
S0	S0	0
S0	S01	1
S01	BH	1
S01	S010	0
S010	IF	0
S010	BH	1
IF	S1	1
S1	BH	1
S1	S10	0
S10	BH	1
S10	IF	0
BH	BH	0
BH	BH	1

```
from graphviz import Digraph

d = Digraph("5.2.1.2", filename='5.2.1.2.gv', engine='dot', format='png')

d.attr('node', shape='doublecircle')
d.node('IF')

d.attr('node', shape='circle')
d.node('BH')
d.node('S0')
d.node('S01')
d.node('S010')
d.node('S1')
d.node('S10')

d.edge('IF', 'S0', label='0')
d.edge('S0', 'S00', label='0')
d.edge('S00', 'IF', label='0')
d.edge('S00', 'S001', label='1')
d.edge('S001', 'S010', label='0')
d.edge('S001', 'BH', label='1')
d.edge('S0', 'S01', label='1')
d.edge('S01', 'BH', label='1')
d.edge('S01', 'S010', label='0')
```

```
d.edge('S010', 'IF', label='0')
d.edge('S010', 'BH', label='1')
d.edge('IF', 'S1', label='1')
d.edge('S1', 'S11', label='1')
d.edge('S11', 'IF', label='1')
d.edge('S11', 'BH', label='0')
d.edge('S1', 'S10', label='0')
d.edge('S10', 'BH', label='1')
d.edge('S10', 'IF', label='0')
d.edge('BH', 'BH', label='0')
d.edge('BH', 'BH', label='1')
d.view()
```



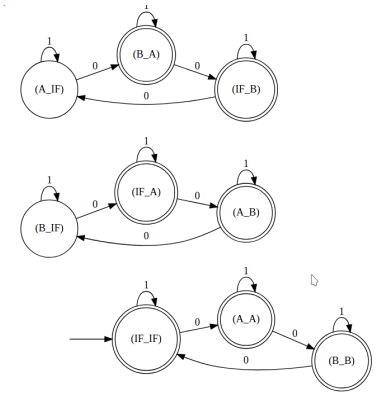
3. I followed the same process, but it was much quicker now that I knew what I was looking for. I just plotted out the different state switches that would happen, and built the DFA from that.

Exercise 6.2: DFA Jove $\cup$, $\cap$

1. Complement:

```
DFA_fig47_comp = {'Q': {'A', 'B', 'IF'},
 'Sigma': {'0', '1'},
 'Delta': {('IF', '0'): 'A',
  ('IF', '1'): 'IF',
  ('A', '0'): 'B',
  ('A', '1'): 'A',
  ('B', '0'): 'IF',
  ('B', '1'): 'B'},
 'q0': 'IF',
 'F': {'A', 'B'}}
```

Union of complement and base:



Yes, this is still a DFA. A DFA is allowed to have disconnected states.

2. It begins from the initial state and moves through every state transition in the language for each state it comes ac. It then constructs a new DFA from only the states it encountered, removing all of the unreachable, and therefore unneeded states.

Exercise 6.5: DFA, DeMorgan's Laws

1. The isomorphic property indicates that not only are two DFA language equivalent, but that they have the same number of states. Two DFA can be language equivalent without being isomorphic. Take for instance, a bloated

and minimal DFA for a given language. Although the two DFA are language equivalent, they are not isomorphic because they do not have the same number of states. The bijection mentioned in Myhill-Nerod Theorem is a byproduct of the isomorphic nature of language equivalent minimal DFA and being able to map a minimal DFA state-to-state with its isomorphic sibling. Therefore, if two DFA are not isomorphic to each other, they will not have a bijection even if they are language equivalent.

2.

```
DFA_6.5.2 = {

   I        : 0 -> I
   I        : 1 -> S1
   S1       : 0 -> S10
   S1       : 1 -> I
   S10      : 0 -> S10
   S10      : 1 -> S101
   S101     : 0 -> S1010
   S101     : 1 -> S01
   S1010    : 0 -> I
   S1010    : 1 -> F10101
   F10101   : 0 -> S10
   F10101   : 1 -> I

}
```

3.

- Check intersection:

String	In Language	Accepted?	Correct?
10101	✓	✓	✓
110101			✓
1000101	✓	✓	✓
111000101	✓	✓	✓
100000			✓
00000101			✓

- Using Tools:

  Done in Jove

4.

```
DFA_oa = {
   I : a -> F
   I : b -> I
   F : a -> I
   F : b -> I
}


DFA_eb = {
   IF : a -> Sa
   IF : b -> IF
   Sa : a -> IF
   Sa : b -> Sa
}
```

$D_{ea} \cup D_{ob} = (D_{oa} \cap D_{eb})^c$

Steps followed in Jove:

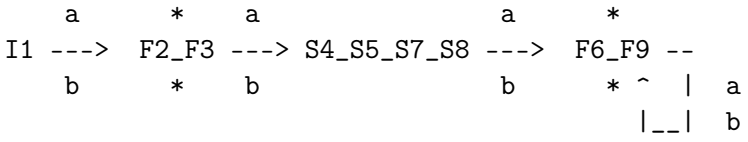
```
inter_Doa_Deq = intersect_dfa(DFA_oa, DFA_eb)

comp_inter_Doa_Deq = comp_dfa(inter_Doa_Deq)

min_comp_inter_Doa_Deq = min_dfa(comp_inter_Doa_Deq)

iso_dfa(min_comp_inter_Doa_Deq, union_dfa(comp_dfa(D_oa), comp_dfa(D_eq)))

# True
```

5.

	I1	F2	F3	S8	S5	S7	S4	F6	F9
I1	x	x	x	x	x	x	x	x	x
F2	✓	x	x	x	x	x	x	x	x
F3	✓	-	x	x	x	x	x	x	x
S8	+	✓	✓	x	x	x	x	x	x
S5	+	✓	✓	-	x	x	x	x	x
S7	+	✓	✓	-	-	x	x	x	x
S4	+	✓	✓	-	-	-	x	x	x
F6	✓	+	+	✓	✓	✓	✓	x	x
F9	✓	+	+	✓	✓	✓	✓	-	x

Pair	Input	Output	Marked?
I1, S8	a	F2, F6	Yes
I1, S5	a	F2, F6	Yes
I1, S7	a	F2, F6	Yes
I1, S4	a	F2, F6	Yes
F2, F3	a	S8, S7	No
F2, F3	b	S5, S4	No
S5, S8	a,b	F6, F6, F6, F9	No
S7, S8	a	F6, F6	No
S7, S8	b	F6, F9	No
S7, S5	a,b	F6, F6	No
S4, S8	a	F6, F6	No
S4, S8	b	F6, F9	No
S4, S5	a,b	F6, F6	No
S4, S7	a,b	F6, F6	No
F6, F2	a	F6, S5	Yes
F6, F3	a	F6, S7	Yes
F9, F2	a	F9, S8	Yes
F9, F3	a	F9, S7	Yes
F9, F6	a	F9, F6	No
F9, F6	b	F6, F6	No

Combine: (F3, F2), (S5, S8), (S7, S8), (S7, S5), (S4, S8), (S4, S5), (S4, S7), (F9, F6)



Output from Jove: