CS 480 Syllabus and Portfolio Winter 2019

Bryan Muller

February 28, 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	✓	✓	✓	✓	✓	✓	100%
6	✓	✓	✓	✓	✓	✓	100%
7	✓	✓	✓	✓	✓	✓	100%
8	✓	✓	✓	✓	✓	✓	100%
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	A		
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 5

Chapter 7 Exercises [100%]

DONE Exercise 7.5: NFA to DFA

- 1.

  ```
  NFA
  I    : 0|1   -> I
  I    : 0|''  -> S0
  S0   : 1     -> S01
  S01  : 0     -> S010
  S010 : 1     -> F
  F    : 0|1   -> F
  ```

  ```
  strings = ["{0:b}".format(i).zfill(4) for i in range(0, 20)]
  accepts = ["accepts" if accepts_nfa(nfa7_5, s) else "rejects" for s in strings]
  list(zip(strings, accepts))
  ```

- 2. A. Define language described in 5.2.1 in formal terms: L = {w : $\in \Sigma^*$: $\forall$ x $\in$ {p $\in$ parts(w): len(p) = 3 } : count(x, 1) = 2 }

  Any word w in {0,1}* where for all partitions of w with length 3 have exactly two 1's.

  Negated: L = {w: $\in \Sigma^*$: $\exists$ x $\in$ {p $\in$ parts(w): len(p) = 3} : count(x, 1) != 2 }

  Any word w wherin exists a partition x of length three which does not have exactly two 1's

  So yes, this is a correct negation. B.

  ```
  complement_of_blocks_of_3 = md2mc('''
  NFA
  I : 0 -> I
  I : 1 -> I
  I : 0 -> S0
  I : 1 -> S1
  S0 : 0 -> S00
  S0 : 1 -> S01
  S1 : 0 -> S10
  ```

```
S01 : 0 -> F
S10 : 0 -> F
S00 : 0 -> F
S00 : 1 -> F
F : 0 -> F
F : 1 -> F
''')
dotObj_nfa(complement_of_blocks_of_3)

dotObj_dfa(min_dfa(comp_dfa(min_dfa(nfa2dfa(complement_of_blocks_of_3)))))
```

C.

```
nums = []
it = 1
while len(nums) < 20:
  for i in itertools.product([0,1],repeat=it):
    nums.append(i)
  it += 1
values = []
for each in nums:
  word = ""
  for i in each:
    word += str(i)
  values.append(word)
for each in values:
    print("String: ", each, " Accepted NFA: ", accepts_nfa(complement_of_blocks_of
```

- 3

  Worked on it with Daniel, Matt, Seth. Couldn't figure it out.

DONE Exercise 7.6.1: Brzozoski's DFA minimization

- 1. Beginning DFA

```
bloated_dfa = md2dc('''
DFA
IS1 : a -> FS2
IS1 : b -> FS3
```

```
FS2 : a -> S4
FS2 : b -> S5
FS3 : a -> S5
FS3 : b -> S4
S4 : a | b -> FS6
S5 : a | b -> FS6
FS6 : a | b -> FS6
''')
```

Reverse turning it into an NFA

```
rev_bloated_nfa = md2mc('''
NFA
IS6 : a | b -> IS6
IS6 : a | b -> S4
IS6 : a | b -> S5
S4  : a -> IS2
S4  : b -> IS3
S5  : a -> IS3
S5  : b -> IS2
IS2 : a -> FS1
IS3 : b -> FS1
''')
```

Turn NFA into DFA

```
rev_bloated_dfa = md2mc('''
DFA
IS0 : a | b -> FS1
FS1 : a | b -> S2
S2  : a | b -> FS3
FS3 : a | b -> FS3
''')
```

Reverse reversed DFA

```
min_nfa = md2mc('''
NFA
IS1 : a | b -> IS1
```

```
IS1 : a | b -> S2
S2 : a | b -> IS3
IS3 : a | b -> FS4
''')
```

Convert back to dfa

```
min_dfa = md2mc('''
DFA
IS0 : a | b -> FS1
FS1 : a | b -> S2
S2 : a | b -> FS3
FS3 : a | b -> FS3
''')
```

- 3.

```
blimp = md2mc('''
DFA
I1 : a -> F2
I1 : b -> F3
F2 : a -> S8
F2 : b -> S5
F3 : a -> S7
F3 : b -> S4
S4 : a | b -> F6
S5 : a | b -> F6
F6 : a | b -> F6
S7 : a | b -> F6
S8 : a -> F6
S8 : b -> F9
F9 : a -> F9
F9 : b -> F6
''')
min1 = min_dfa(blimp)
min2 = min_dfa_brz(blimp)
iso_dfa(min1, min2)
```

True

Chapter 8 Exercises [100%]

DONE Exercise 8.2: NFA Operations

- 1. 001100100 001000101
- 2.

  ```
  re_8_5_nfa = md2mc('''
  NFA
  I1 : '' -> St1
  I1 : '' -> St2
  St1 : '' -> I1
  St2 : a -> St3
  St3 : '' -> I1
  IF2 : '' -> St4
  IF2 : '' -> St5
  St4 : c -> St6
  St6 : '' -> St7
  St7 : d -> St8
  St8 : '' -> IF2
  St5 : b -> St9
  St9 : '' -> IF2
  I1 : '' -> IF2
  ''')
  dotObj_nfa(re_8_5_nfa)
  ```

- 3.

  ```
  re_8_5_nfa = re2nfa("(''+a)*(b+cd)*")
  dotObj_nfa(re_8_5_nfa)

  re_8_5_nfa_hand = md2mc('''
  NFA
  I1 : '' -> St1
  I1 : '' -> St2
  St1 : '' -> I1
  St2 : a -> St3
  St3 : '' -> I1
  IF2 : '' -> St4
  IF2 : '' -> St5
  ```

```
St4 : c -> St6
St6 : '' -> St7
St7 : d -> St8
St8 : '' -> IF2
St5 : b -> St9
St9 : '' -> IF2
I1 : '' -> IF2
''')
dotObj_nfa(re_8_5_nfa_hand)

iso_dfa(nfa2dfa(re_8_5_nfa), nfa2dfa(re_8_5_nfa_hand))
```

True!

DONE Exercise 8.8: Sylvester's Formula

- 1. No. Any linear combination of 3 and 6 will always have to be a multiple of three. This means there are infinitely many natural numbers which cannot be expressed by 3 and 6.
- 2.

  The requirement that the greatest common divisor (GCD) equal 1 is necessary in order for the Frobenius number to exist. If the GCD were not 1, every integer that is not a multiple of the GCD would be inexpressible as a linear, let alone conical, combination of the set, and therefore there would not be a largest such number. For example, if you had two types of coins valued at 4 cents and 6 cents, the GCD would equal 2, and there would be no way to combine any number of such coins to produce a sum which was an odd number. On the other hand, whenever the GCD equals 1, the set of integers that cannot be expressed as a conical combination of { a1, a2, ..., an } is bounded according to Schur's theorem, and therefore the Frobenius number exists.

  – Wiki

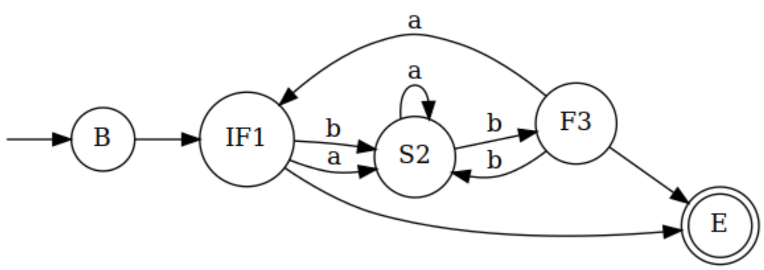
Exercise 8.8.5: Postage Stamp

- 1. a. p,q = 5, 11 F(p,q) = (5*11) - 5 - 11 = 55 - 5 - 11 = 39 b. p,q = 5,1 F(p,q) = 39 p,q,r = 5,7,11 F(p,q,r) = 13
- 2.

Week 6

Monday, Feb 11

CDL



Chapter 9 [100%]

DONE Exercise 9.2: NFA to RE

- 1. Deleting the “X” state last produced the smaller regular expression. This is due to the nature of how states are removed from the NFA. If we remove the “busy” state last, then most of the states leading to it have already been collapsed, leading to less states needing to be collapsed in the final iteration.
  - a. Delete Order:

    ```
    ["I1", "I2","I3", "F1","F2", "X"]
    ```

    Produced Regex:

    ```
    '(((c + ((b + a) + a)) + ((b + a) + a)) (((p + q))* ((n + m) + m)))'
    ```

    Delete Order
  - b Delete order:

    ```
    ["X", "I1", "I2", "I3", "F1", "F2"]
    ```

    Produced Regex:

    ```
    '(((((c (((p + q))* n)) + (((b (((p + q))* n)) + (a (((p + q))* n))) + (a (((
    ```

- 2. Heuristically, it seems better to eliminate the busy state last. This is due to the reason I explained above. This could perhaps change based on the nfa and the number of “busy” states.

- 3. We could always convert them back to an nfa and check for language equivelance.

DONE Exercise 9.5: nfa2re: RE Size

- 1.

  ```
  md2mc('''
  NFA
  I : '' -> A
  I : '' -> G
  A : '' -> C
  A : '' -> B
  B : 1 -> D
  C : 0 -> E
  D : '' -> G
  E : '' -> A
  E : '' -> G
  G : 1 -> F
  ''')
  ```

  Delete Order: I, A, B, C, D, E, G, F.

  - Step 1. Add real I (S) and real F (Q)

    ```
    md2mc('''
    NFA
    S : '' -> I
    I : '' -> A
    I : '' -> G
    A : '' -> C
    A : '' -> B
    B : 1 -> D
    C : 0 -> E
    D : '' -> G
    E : '' -> A
    E : '' -> G
    G : 1 -> F
    F : '' -> Q
    ''')
    ```

$$
\begin{aligned}
\text{SI} &= \epsilon \\
\text{IA} &= \epsilon \\
\text{IG} &= \epsilon \\
\text{AB} &= \epsilon \\
\text{AC} &= \epsilon \\
\text{BD} &= 1 \\
\text{CE} &= 0 \\
\text{DG} &= \epsilon \\
\text{EA} &= \epsilon \\
\text{EG} &= \epsilon \\
\text{GF} &= 1 \\
\text{FQ} &= \epsilon
\end{aligned}
$$

- Remove I i = S, j = a $\mathrm{R1_a} = \epsilon$, $\mathrm{R2_a} =$, $\mathrm{R3_a} = \epsilon$, $\mathrm{R4_a} =$ i = S, j = g $\mathrm{R1_g} = \epsilon$, $\mathrm{R2_g} =$, $\mathrm{R3_g} = \epsilon$, $\mathrm{R4_g} =$

$$
\begin{array}{lclcl}
\text{SA} &=& (\epsilon)()^*(\epsilon) \cup () &=& \epsilon \\
\text{SG} &=& (\epsilon)()^*(\epsilon) \cup () &=& \epsilon \\
\text{AB} &=& \epsilon &=& \epsilon \\
\text{AC} &=& \epsilon &=& \epsilon \\
\text{BD} &=& 1 &=& 1 \\
\text{CE} &=& 0 &=& 0 \\
\text{DG} &=& \epsilon &=& \epsilon \\
\text{EA} &=& \epsilon &=& \epsilon \\
\text{EG} &=& \epsilon &=& \epsilon \\
\text{GF} &=& 1 &=& 1 \\
\text{FQ} &=& \epsilon &=& \epsilon
\end{array}
$$

- Remove A i = S, j = b $\mathrm{R1_B} = \epsilon$, $\mathrm{R2_B} =$, $\mathrm{R3_B} = \epsilon$, $\mathrm{R4_B} =$ i = S, j = c $\mathrm{R1_C} = \epsilon$, $\mathrm{R2_C} =$, $\mathrm{R3_C} = \epsilon$, $\mathrm{R4_C} =$ i = E, j = E $\mathrm{R1_E} = \epsilon$, $\mathrm{R2_E} =$, $\mathrm{R3_E} =$, $\mathrm{R4_C} =$

$$
\begin{array}{lclcl}
\text{SB} &=& (\epsilon)()^*(\epsilon) \cup () &=& \epsilon \\
\text{SC} &=& (\epsilon)()^*(\epsilon) \cup () &=& \epsilon \\
\text{SG} &=& (\epsilon)()^*(\epsilon) \cup () &=& \epsilon \\
\text{BD} &=& 1 &=& 1 \\
\text{CE} &=& 0 &=& 0 \\
\text{DG} &=& \epsilon &=& \epsilon \\
\text{ES} &=& (\epsilon)()^*(\epsilon) \cup () &=& \epsilon \\
\text{EG} &=& \epsilon &=& \epsilon \\
\text{GF} &=& 1 &=& 1 \\
\text{FQ} &=& \epsilon &=& \epsilon
\end{array}
$$

- Remove B i = S, j = D R1 = ϵ, R2 = , R3 = 1, R4 =

$$\begin{array}{lclcl} SC & = & (\epsilon)()^*(\epsilon) \cup () & = & \epsilon \\ SD & = & (\epsilon)()^*(1) \cup () & = & 1 \\ SG & = & (\epsilon)()^*(\epsilon) \cup () & = & \epsilon \\ CE & = & 0 & = & 0 \\ DG & = & \epsilon & = & \epsilon \\ ES & = & (\epsilon)()^*(\epsilon) \cup () & = & \epsilon \\ EG & = & \epsilon & = & \epsilon \\ GF & = & 1 & = & 1 \\ FQ & = & \epsilon & = & \epsilon \end{array}$$

- Remove C i = S, j = E R1 = ϵ, R2 = , R3 = 0, R4 =

$$\begin{array}{lclcl} SD & = & (\epsilon)()^*(1) \cup () & = & 1 \\ SE & = & (\epsilon)()^*(0) \cup () & = & 0 \\ SG & = & (\epsilon)()^*(\epsilon) \cup () & = & \epsilon \\ DG & = & \epsilon & = & \epsilon \\ ES & = & (\epsilon)()^*(\epsilon) \cup () & = & \epsilon \\ EG & = & \epsilon & = & \epsilon \\ GF & = & 1 & = & 1 \\ FQ & = & \epsilon & = & \epsilon \end{array}$$

- Remove D i = S, j = G R1 = 1, R2 = , R3 = ϵ, R4 = ϵ

$$\begin{array}{lclcl} SE & = & (\epsilon)()^*(0) \cup () & = & 0 \\ SG & = & (1)()^*(\epsilon) \cup (\epsilon) & = & 1 \\ ES & = & (\epsilon)()^*(\epsilon) \cup () & = & \epsilon \\ EG & = & \epsilon & = & \epsilon \\ GF & = & 1 & = & 1 \\ FQ & = & \epsilon & = & \epsilon \end{array}$$

- Remove E i = S, j = S R1 = 0, R2 = , R3 = ϵ, R4 = i = S, j = G R1 = 0, R2 = , R3 = ϵ, R4 = 1

$$\begin{array}{lclcr} SS & = & (0)()^*(\epsilon) \cup () & = & 0 \\ SG & = & (0)()^*(\epsilon) \cup (1) & = & 0+1 \\ GF & = & 1 & = & 1 \\ FQ & = & \epsilon & = & \epsilon \end{array}$$

- Remove G i = S, j = F R1 = 0+1, R2 = , R3 = 1, R4 =

$$\begin{array}{lclcl} SS & = & (0)()^*(\epsilon) \cup () & = & 0 \\ SF & = & (0+1)()^*(1) \cup () & = & (0+1)1 \\ FQ & = & \epsilon & = & \epsilon \end{array}$$

- Remove F i = S, j = Q R1 = (0+1)1, R2 = , R3 = ϵ, R4 =

$$\begin{aligned} SS &= (0)()^*(\epsilon) \cup () &&= 0 \\ SQ &= (0+1)1()^*(\epsilon) \cup () &&= (0+1)1 \end{aligned}$$

- Final REGEX ((((0 ((0)* ((1 + “”) + “”))) + ((1 + “”) + “”)) + ((1 + “”) + “”)) 1)

$$\begin{aligned} SI &= \epsilon \\ IA &= \epsilon \\ IG &= \epsilon \\ AC &= \epsilon \\ AB &= \epsilon \\ BD &= 1 \\ CE &= 0 \\ DG &= \epsilon \\ EA &= \epsilon \\ EG &= \epsilon \\ GF &= 1 \\ FQ &= \epsilon \end{aligned}$$

Week 7

DONE Wednesday, FEB 20, 2019 [100%]

DONE Warmup CDL the boy sees a flower

Noun Phrase	Verb Phrase
Complex Noun	Complex Verb

Arcticle	Noun	Verb	Noun Phrase

the	boy	sees	Article	Noun
the	boy	sees	a	flower

SENTANCE => NOUN-PHRASE VERB-PHRASE
=> COMPLEX-NOUN COMPLEX-VERB
=> ARTICLE NOUN VERB NOUN-PHRASE
=> the boy sees COMPLEX-NOUN
=> the boy sees ARTICLE NOUN
=> the boy sees a flower

DONE 2nd Warmup CDL a girl with a flower likes the boy SENTANCE
=> NP VP
=> CN PP VP
=> A N P CN CV
=> a girl with A N V NP
=> a girl with a flower likes CN
=> a girl with a flower likes A N
=> a girl with a flower likes the boy

DONE 3rd Warmpup CDL the girl touches the boy with the flower S
=> NP VP
=> NP CV PP
=> CN V NP P CN
=> CN V CN P CN
=> A N V A N P A N
=> the girl touches the boy with the flower

DONE 4th Warmup CDL

S -> aB
B -> ”
bbB
B -> bb
bbB

DONE 5th Warmup CDL

S -> ”
S -> aSbb

DONE 6th Warmup CDL

S -> λ
S -> B
B -> bB
B -> λ

DONE REAL CDL L = $\{s : s \in \{a, b\}^*$ and $\#_b(s) = 2\#_a(s)\}$

S -> a S b S b
S -> b S a S b
S -> b S b S a
S -> ϵ

REAL CDL pt 2 L = $\{s : s \in \{a, b\}^*$ and $\#_b(s) = 2\#_a(s) + 3\}$

S -> A
S -> ϵ
A -> B b b b
A -> b B b b
A -> b b B b
A -> b b b B
B -> a B b B b
B -> b B a B b
B -> b B b B a
B -> ϵ

DONE Friday, FEB 22, 2019 [100%]

DONE CDL Build two PDA

One to recognize the language of EvenPalindromes over $\Sigma = \{0, 1\}$, $\{ww^R : w \in \Sigma\}$ (4 states)

The other to recognize the language of MarkedPalindromes over $\Sigma = \{0, 1\}$ with # as the the **marked** character.

$\{w\#w^R : w \in \Sigma\}$ (4 states)

```
I: 0, # : 0# -> A
I: 1, # : 1# -> A
A : 0, 0 : 00 -> A
A : 1, 0 : 01 -> A
A : 0, 1 : 01 -> A
A : 1, 1 : 11 -> A
A : #, 0 : 0  -> B
A : #, 1 : 1  -> B
B : 1, 1 : '' -> B
B : 0, 0 : '' -> B
B : '', #: # -> F
```

DONE Chapter 11 Exercises [100%]

DONE 11.5.1 [100%]

- **DONE** 11.5.1.1 Sentance 1 + 2 * 3

  ```
  E -> E+T | T
  T -> T*F | F
  F -> 1 | 2 | 3 | ~F | (E)
  ```

  E => E + T => F + T * F => 1 + F * 3 => 1 + 2 * 3

  as parse tree:

  ```
      E
     /|\
    E + T
   /   /|\
  F   T * F
  |   |   |
  1   F   3
      |
      2
  ```

- **DONE** 11.5.1.2 Sentence: 1 + ~ 2 * 3

  - CFG1

    #+NAME CFG 1

    ```
    E -> 1 | 2 | 3 | ~E | E+E | E*E | (E)
    ```

    Parse Tree 1

    ```
      E
     /|\
    E + E
    |  /|\
    1 E * E
      |   |
     ~E   3
      |
      2
    ```

    Parse Tree 2

```
     E
    /|\
   E * E
  /|\  |
 E + E 3
 |   |
 1  ~E
     |
     2
```

- CFG2 #+NAME CFG 2

```
E -> E+T | T
T -> T*F | F
F -> 1 | 2 | 3 | ~F | (E)
```

```
    E
   /|\
  / | \
 E  +  T
 |    /|\
 T   T * F
 |   |   |
 F   F   3
 |   |
 1  ~F
     |
     2
```

- **DONE** 11.5.1.3 I would argue that they denote the same context free language because they contain the same set of terminals and transitions. While the transition functions are not the same, (CFG1 is ambigious), they can produce language equivalent parse trees, meaning any sentance that can be turned into a parse tree with CFG1 can also be turned into a parse tree with CFG2 (and vice versa)

DONE 11.10.1 [100%]

- **DONE** 11.10.1.5
  - Case 1 (OP is AND) $L_{abcd} = \{a^i b^j c^k d^l : i,j,k,l \geq 0 \text{ and } ((i = j) \text{ AND } (k = l))\}$

```
S => '' | AB
A => '' | aAb
B => '' | cBd
```

- Case 2 (OP is OR)
  $L_{abcd} = \{a^i b^j c^k d^l : i,j,k,l \geq 0 \text{ and } ((i = j) \text{ OR } (k = l))\}$

  ```
  S => '' | XY | ABY | XCD
  X => '' | aXb
  Y => '' | cYd
  A => '' | aA
  B => '' | bB
  C => '' | cC
  D => '' | dD
  ```

- **DONE** 11.10.1.6 $L_{acbd} = \{a^i\ c^k\ b^j\ d^l : i,j,k,l, \geq 0 \text{ and } ((i = j) \text{ OP } (k = l))$
  - Case 1 OP=AND Assume $L_{acbd} = \{a^i\ c^k\ b^j\ d^l : i,j,k,l, \geq 0 \text{ and } ((i = j) \text{ AND } (k = l))$ is context free.

    Pumping Lemma applies & garuntees an $N > 0$

    Pick N of the pumping lemma. Pick $z = a^n\ c^n\ b^n\ d^n$. Break z into uvwxy, with $|vwx| \leq n$ and $vx \neq \epsilon$. Then vwx contains one or two different symbols. In both cases, the string uwy connot be in L.

    Context Free Languages cannot match two substrings of arbitrary length over an alphabet of at least two symbols.
  - Case 2 OP=OR
    $L_{acbd} = \{a^i\ c^k\ b^j\ d^l : i,j,k,l, \geq 0 \text{ and } ((i = j) \text{ OR } (k = l))$
    This isn't complete, but I feel like it is close

    ```
    S => '' | AcBd | aCbD
    A => '' | aA
    B => '' | bB
    C => '' | cC
    D => '' | dD
    ```

Week 8

DONE Chapter 13 Exercises [100%]

DONE Exercise 13.8: DTM and NDTM Design [100%]

- **DONE** 1

Current State	Symbol Read	Next State	Symbol Written	Move Direction
i0	.	fhalt	0	S
i0	0	fhalt	1	S
i0	1	q1	0	R
q1	0	fhalt	1	S
q1	1	q1	0	R
q1	.	fhalt	1	S

```
(princ "TM\n")
(loop for (cs sr ns sw md) in (cddr table)
      do (princ (format "%s : %s ; %s , %s -> %s\n" cs sr sw md ns)))
```

```
TM
i0 : . ; . , S -> fhalt
i0 : 0 ; 1 , S -> fhalt
i0 : 1 ; 0 , R -> q1
q1 : 0 ; 1 , S -> fhalt
q1 : 1 ; 0 , R -> q1
q1 : . ; 1 , S -> fhalt
```

- **DONE** 2 Assuming there is a '#' at the start of the string and a '$' at the end. If they do not exist, I would just add the states to add them before starting the "count".

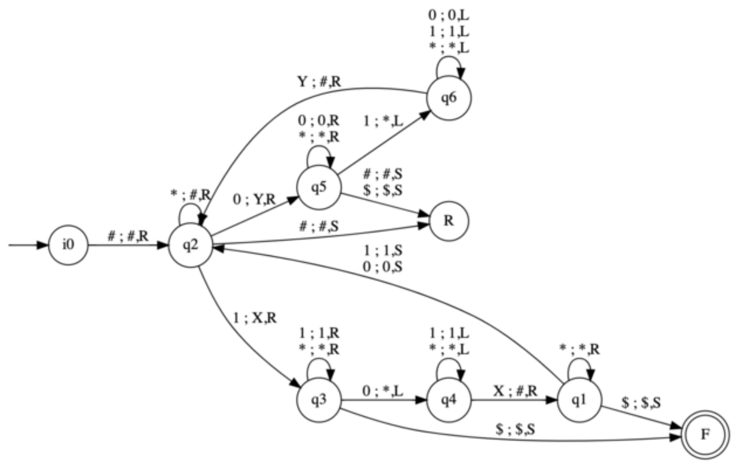
  Basic idea, bounce back and forth "matching up" pairs of 1's and 0's until you run out. Depending on which number you've started on, you will need to accept or reject upon reaching a specific side of the tape. q2 handles switching between matching an initial 0 to a 1, or an initial 1 to a 0. Note, that a string such as 10011 would start with an initial 1, match it to the first zero, and then "restart" on the second zero. The q2 state handles that restart.

  I am making no claim that this is the "best" program for this particular problem, but it has worked on every string I've thrown at it.

Current State	Symbol Read	Symbol Written	Move Direction	Next State
i0	#	#	R	q2
q1	*	*	R	q1
q1	1	1	S	q2
q1	$	$	S	A
q1	0	0	S	q2
q2	1	X	R	q3
q2	0	Y	R	q5
q2	*	#	R	q2
q2	#	#	S	R
q3	0	*	L	q4
q3	1	1	R	q3
q3	*	*	R	q3
q3	$	$	S	A
q4	X	#	R	q1
q4	1	1	L	q4
q4	*	*	L	q4
q5	0	0	R	q5
q5	1	*	L	q6
q5	#	#	S	R
q5	*	*	R	q5
q5	$	$	S	R
q6	0	0	L	q6
q6	1	1	L	q6
q6	Y	#	R	q2
q6	*	*	L	q6

```
(princ "TM\n")
(loop for (cs sr sw md ns) in (cddr table)
      do (princ (format "%s : %s ; %s , %s -> %s\n" cs sr sw md ns)))
```

TM Graph



0 ; 0,L
1 ; 1,L
* ; *,L
q6
Y ; #,R
0 ; 0,R
* ; *,R
1 ; *,L
q5
* ; #,R
0 ; Y,R
; #,S
$; $,S
R
i0
; #,R
q2
; #,S
1 ; 1,S
0 ; 0,S
1 ; X,R
1 ; 1,R
* ; *,R
1 ; 1,L
* ; *,L
* ; *,R
q3
0 ; *,L
q4
X ; #,R
q1
$; $,S
F
$; $,S