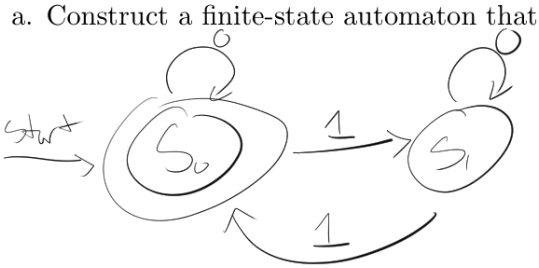
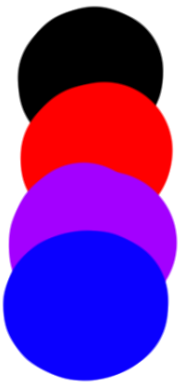
CSC 404 - Homework 3 - Name: Chris Glanzer

Problem 1 (RegEx < TM)**.** Anything that can be done in RE can be done on a TM, so let's go redo a small machine! Consider the regular language of all bit strings that contain an even number of 1s.

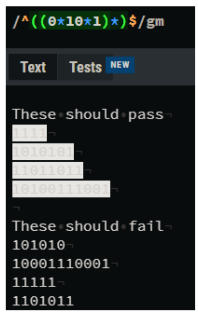
a. Construct a finite-state automaton that recognizes this language.





b. Represent this language with a regular expression.

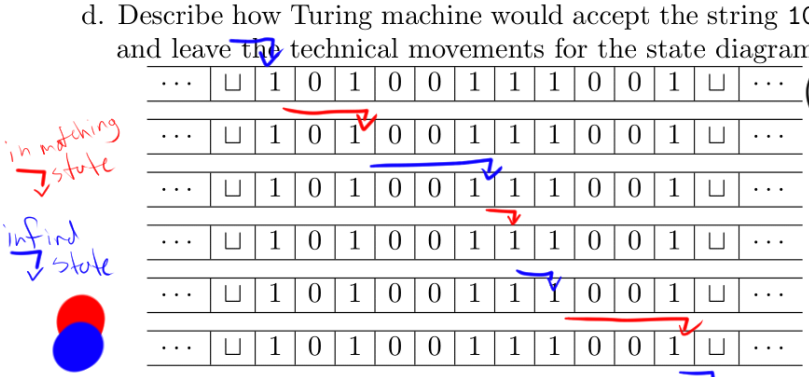
$/^((0*10*1)*)$/gm$

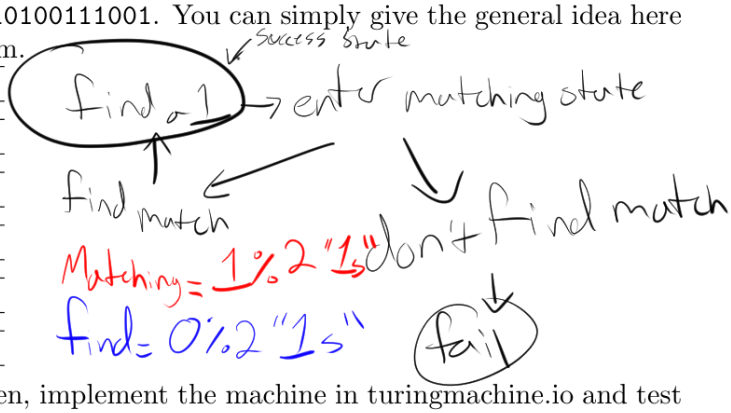


c. Implement the regular expression and test it against the sets:

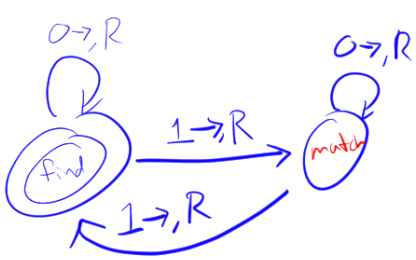
Accepted: `1111, 1010101, 11011011, 10100111001` Rejected: `101010, 10001110001, 11111, 1101011`

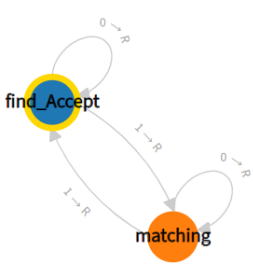
d. Describe how Turing machine would accept the string `10100111001`. You can simply give the general idea here and leave the technical movements for the state diagram.





e. Construct a state diagram for this Turing machine. Then, implement the machine in turingmachine.io and test it :-) (Upload a screenshot of your machine - you do not need to worry about importing it into the PDF).






```
#A turing machine for binary palindromes (i.e., made of the symbols '0' and '1')

input: '10100111001'
#input: '0010100'

blank: ' '
start state: find_Accept
table:
  find_Accept:
    0: R
    1: {R: matching}
  matching:
    0: R
    1: {R: find_Accept}
```

Problem 2 (Doubling, i.e., $f(n) = 2n$, yay!)**.** In this exercise, we consider the problem of doubling a non-negative integer input. That is, we work towards constructing a Turing machine that computes the function $f(n) = 2n = n + n$. As with (early) addition, we will work with the unary representation of integers. That is, we represent integer n by a string of n 1s (e.g., we represent 4 as 1111 and 2 as 11.)

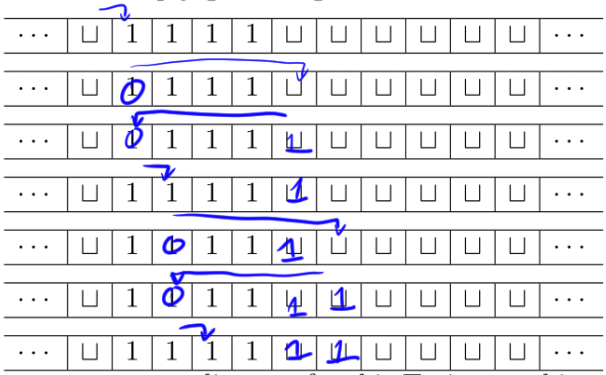
a. Describe how a Turing machine would compute the function $f(4) = 2 \cdot 4 = 8$. That is, we begin with a tape with 4 1s (to represent 4)

⋯	⊔	1	1	1	1	⊔	⊔	⊔	⊔	⊔	⊔	⋯

and need to produce a tape with 8 1s, i.e.,

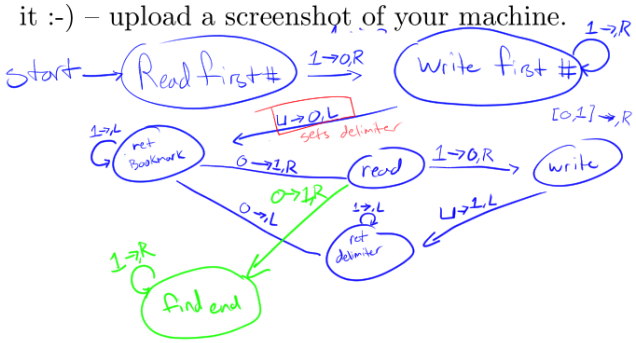
⋯	⊔	1	1	1	1	1	1	1	1	⊔	⊔	⋯

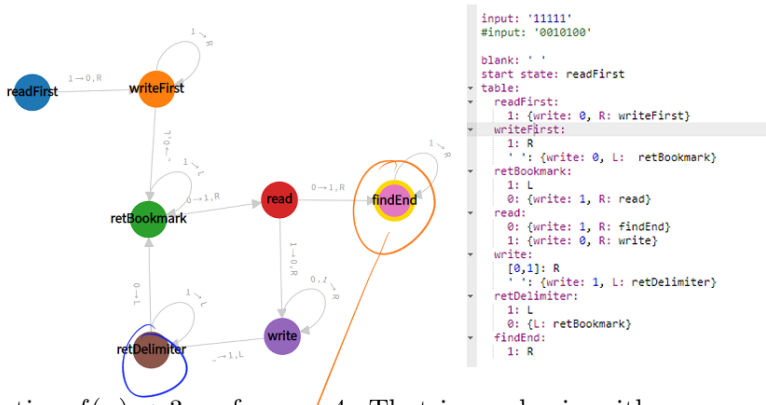
You can simply give the general idea here and leave the technical movements for the state diagram.



since 1 is the only significant digit we can just flip the bit to indicate a read ⇒ copy each bit before flipping them back

b. Construct a state diagram for this Turing machine. Then, implement the machine in turingmachine.io and test it :-) – upload a screenshot of your machine.





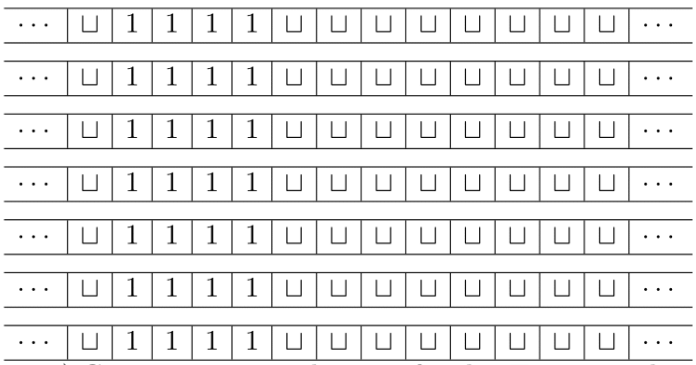
c. Describe how a Turing machine would compute the function $f(n) = 3 \cdot n$ for $n = 4$. That is, we begin with a tape with 4 1s (to represent 4)

⋯	⊔	1	1	1	1	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⊔	⋯

and need to produce a tape with $3 \cdot 4 = 12$ 1s, i.e.,

⋯	⊔	1	1	1	1	1	1	1	1	1	1	1	1	⋯

You can simply give the general idea here and leave the technical movements for the state diagram.



add a retDelimiter state to detect when we have copied enough times (3x) ⇒ remove the findEnd state... once we have enough copies, divert to another branch which fills in the delimiters ⇒ finds the end

d. (Bonus) Construct a state diagram for this Turing machine. Then, implement the machine in turingmachine.io and test it :-).

e. (Bonus) For a fixed non-negative integer m (i.e., you know the m before), describe how a Turing machine would compute the function $f(n) = mn$. for static M, just have a total of m-1 retDelimiter states

Problem 3 (Addition - Again!)**.** In Section 2.2, we explored addition on a Turing Machine, through unary addition. That is, we essentially represented numbers as tally marks (1s) and smashed the two strings together. Unary addition comes with obvious 'storage' issues in that if we wanted to add 2048 and 128 we would use 2048 1s followed by a $*$ and 128 1s creating 2177 tape symbols!

Writing $2048 = 2^{11} = \texttt{0b100000000000}$ and $128 = 2^7 = \texttt{0b10000000}$ and performing binary addition

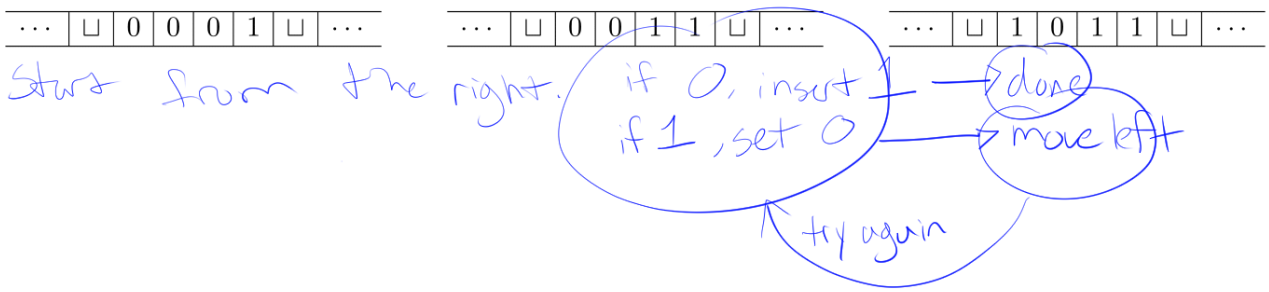
$$\texttt{0b100000000000} + \texttt{0b10000000} = \texttt{0b100010000000} = 2176$$

should reduce the number of symbols tremendously! (That and unary numbers are boring/not as fun to work with :-)). So, let's switch to adding in binary.

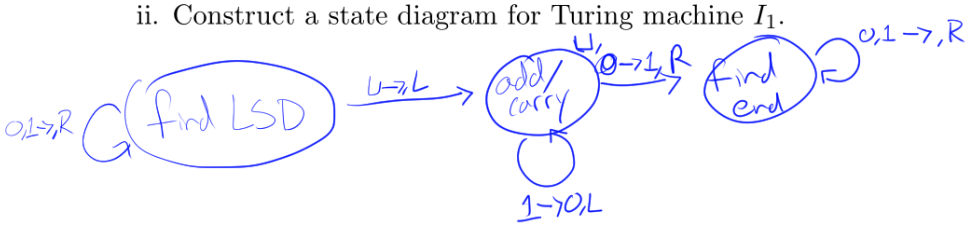
a. First up is Binary Increment - that is, adding a binary number by 1. For example,

$0000 + 1 = 0001 \mid 0001 + 1 = 0010 \mid 0010 + 1 = 0011 \mid 0011 + 1 = 0100 \mid 0100 + 1 = 0101 \mid 0101 + 1 = 0110$

i. Describe how a Turing machine, I_1, would perform binary increment on the strings 0001, 0011, and 1011



ii. Construct a state diagram for Turing machine I_1.

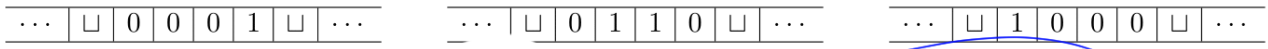


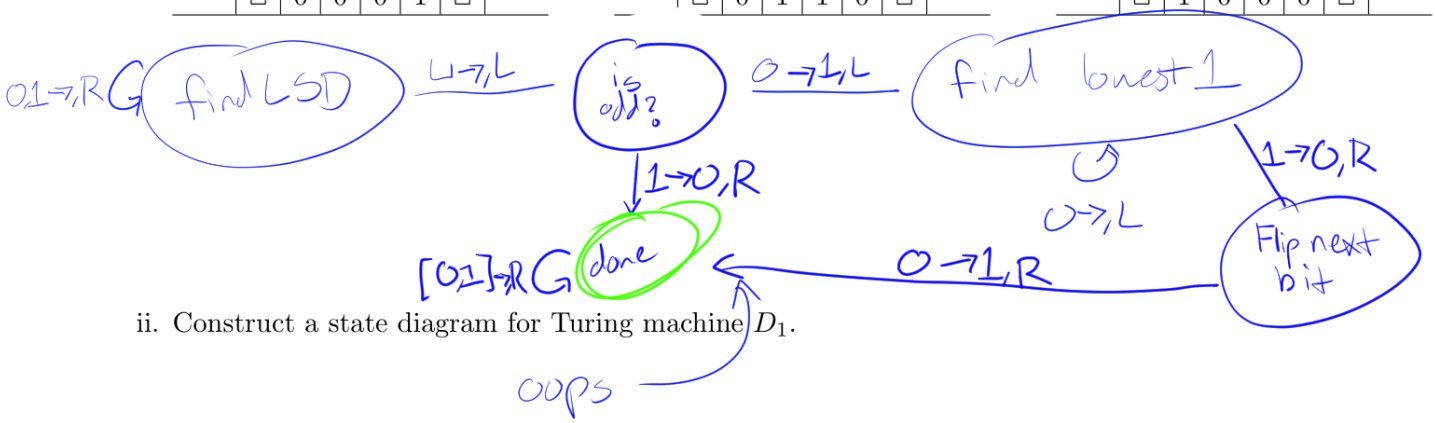
(Hint: You should see that when you increment, you just need to turn all the 1s on the right into 0s and turn the first 0 into a 1.)

b. Next up is Binary Decrement - that is, subtracting a binary number by 1. For example,

$0001 - 1 = 0000 \mid 0010 - 1 = 0001 \mid 0011 - 1 = 0010 \mid 0100 - 1 = 0011 \mid 0101 - 1 = 0100 \mid 0110 - 1 = 0101$

i. Describe how a Turing machine, D_1, would perform binary decrement on the strings 0001, 0110, and 0100





ii. Construct a state diagram for Turing machine D_1.

find least significant 1, flip to 0

↓

if there are less significant bits → flip the 0/1 bit

c. Now that we have the pieces in place, we are ready to do addition – Wee! With machines built for increment and decrement addition is pretty straight forward. We describe the process through an example. Here we seek to add `x` $= 5 =$ `0b101` and `y` $= 3 =$ `0b011`. We pad these with an additional bit (`0`) and place `0101` $+$ `0011` on the tape. If the head finds a `1` before the `+` it decrements the string to the left of the `+` and increments the string to the right of the `+`. This continues until there are no `1`s to the left of the `+`. The string to the right of the `+` is then $x + y$ - woo!

$\cdots$	$\sqcup$	0	1	0	1	+	0	0	1	1	$\sqcup$	$\cdots$
$\cdots$	$\sqcup$	0	1	0	0	+	0	1	0	0	$\sqcup$	$\cdots$
$\cdots$	$\sqcup$	0	0	1	1	+	0	1	0	1	$\sqcup$	$\cdots$
$\cdots$	$\sqcup$	0	0	1	0	+	0	1	1	0	$\sqcup$	$\cdots$
$\cdots$	$\sqcup$	0	0	0	1	+	0	1	1	1	$\sqcup$	$\cdots$
$\cdots$	$\sqcup$	0	0	0	0	+	1	0	0	0	$\sqcup$	$\cdots$
$\cdots$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	$\sqcup$	1	0	0	0	$\sqcup$	$\cdots$

Notice that `0b1000` $= 8 =$ `5` $+$ `3` is left on the tape, so it worked! This also highlights the need to pad each of the inputs (we needed the 4th bit). In general, if x is an n bit number and y is an m bit number we will write each of these as a $\max\{m, n\} + 1$ bit number on the tape. (For example, 5 is 3 bits and 3 is 2 bits, so we write these as $\max\{3, 2\} + 1 = 3 + 1 = 4$ bit numbers.)

Now it is your turn! Add `7` $=$ `0b111` (3 bit number) and `11` $=$ `0b1011` (4 bit number) using the above procedure.

$\cdots$	$\sqcup$	0	0	1	1	1	+	0	1	0	1	1	$\sqcup$	$\cdots$
$\cdots$	$\sqcup$			1	1	0			1	1	0	0	$\sqcup$	$\cdots$
$\cdots$	$\sqcup$			1	0	1			1	1	0	1	$\sqcup$	$\cdots$
$\cdots$	$\sqcup$			1	0	0			1	1	1	0	$\sqcup$	$\cdots$
$\cdots$	$\sqcup$			0	1	1			1	1	1	1	$\sqcup$	$\cdots$
$\cdots$	$\sqcup$				1	0		1	0	0	0	0	$\sqcup$	$\cdots$
$\cdots$	$\sqcup$				0	1		1	0	0	0	1	$\sqcup$	$\cdots$
$\cdots$	$\sqcup$					0		1	0	0	1	0	$\sqcup$	$\cdots$
$\cdots$	$\sqcup$												$\sqcup$	$\cdots$

6+2=18=9 7+11 ✓

d. Construct a state diagram for the Turing machine, ADD (addition) – Have fun :-).

e. (Bonus) Implement the TM in turingmachin.io and test it on fun examples (or examples you are willing to let run – it may take a while).

f. (Super Bonus) What about multiplication of binary numbers?