1. *Finite automata and regular languages:*

50/50 = (10/10)

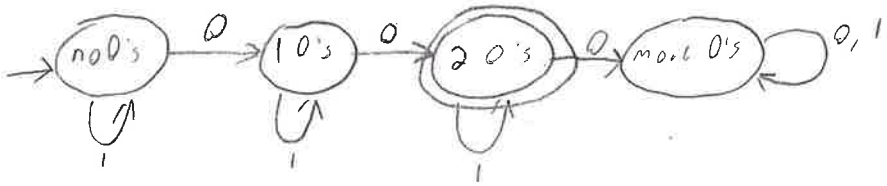
10/10 1a. Design a finite automaton for recognizing binary sequences that have exactly two zeros. Assume that the input strings contain only symbols 0 and 1. The easiest is to have 4 states: no-zeros, 1-zero, 2-zeros, and more-than-2-zeros, you just need to describe transitions between these states, and which states are final. Show, step-by-step, how your automaton will accept the string 01101.

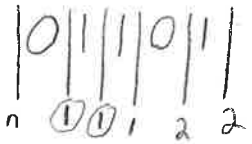
10/10 1b. On the example of this automaton, show how the word 01101 can be represented as xyz in accordance with the pumping lemma.

10/10 1c. Use a general algorithm to describe a regular expression corresponding to the finite automaton from the Problem 1a.

20/20 1d-e. The resulting language can also be described by a regular expression $1^*01^*01^*$. Use a general algorithm to transform this regular expression into a finite automaton: first a non-deterministic one, then a deterministic one.

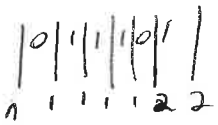
1a)





accepted

1b) x = 0
y = 1
z = 101

xyyz =  still accepted

49/50 = (9.8/10)

1. *Finite automata and regular languages:*

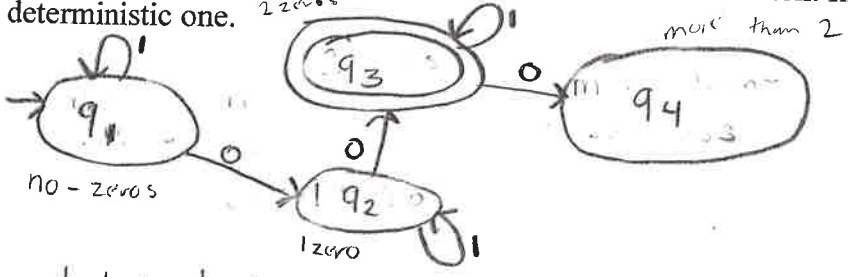
10/10 1a. Design a finite automaton for recognizing binary sequences that have exactly two zeros. Assume that the input strings contain only symbols 0 and 1. The easiest is to have 4 states: no-zeros, 1-zero, 2-zeros, and more-than-2-zeros, you just need to describe transitions between these states, and which states are final. Show, step-by-step, how your automaton will accept the string 01101.

10/10 1b. On the example of this automaton, show how the word 01101 can be represented as xyz in accordance with the pumping lemma.

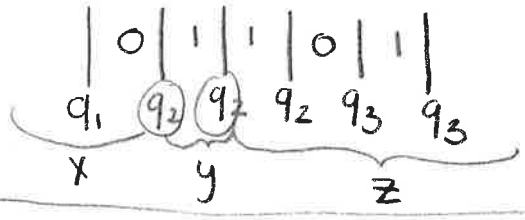
9/10 1c. Use a general algorithm to describe a regular expression corresponding to the finite automaton from the Problem 1a.

20/20 1d-e. The resulting language can also be described by a regular expression $1^*01^*01^*$. Use a general algorithm to transform this regular expression into a finite automaton: first a non-deterministic one, then a deterministic one.

1a.)

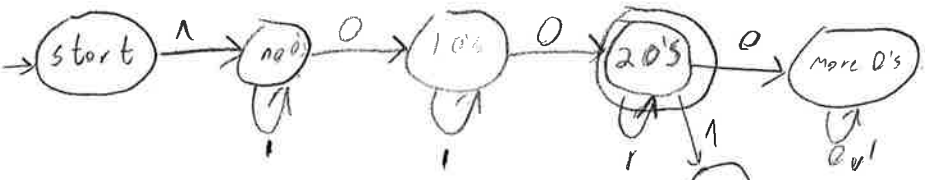


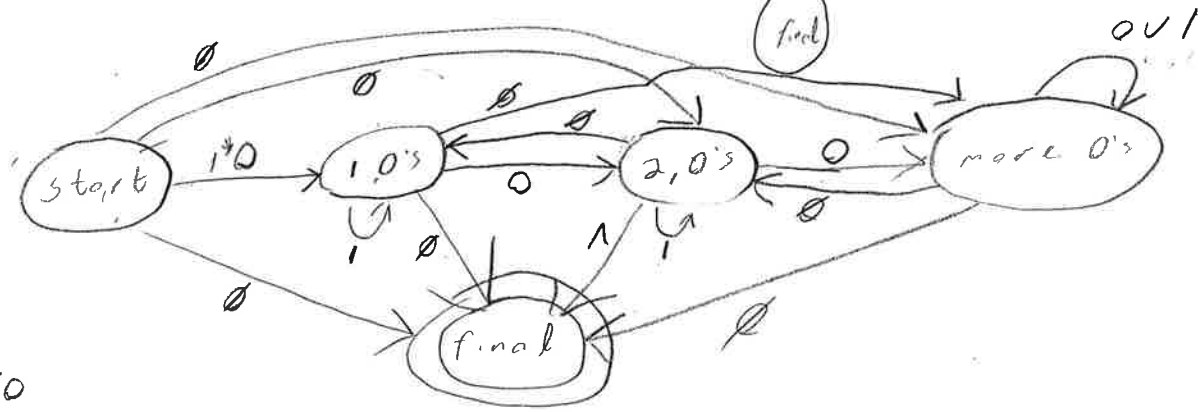
1b.)



x = 0
y = 1
z = 101

1c)





$R_{s1} = \emptyset \cup \lambda 1^*0$
$= 1^*0$

$R_{s2} = \emptyset \cup \lambda 1^*\emptyset$
$= \emptyset$

$R_{sm} = \emptyset \cup \lambda 1^*\emptyset$
$= \emptyset$

$R_{sf} = \emptyset \cup \lambda 1^*\emptyset$
$= \emptyset$

$R_{11} = 1 \cup \emptyset$
$= 1$

$R_{12} = 0 \cup \emptyset$
$= 0$

$R_{1m} = \emptyset \cup \emptyset$
$= \emptyset$

$R_{1f} = \emptyset \cup \emptyset$
$= \emptyset$

$R_{21} = \emptyset \cup \emptyset$
$= \emptyset$

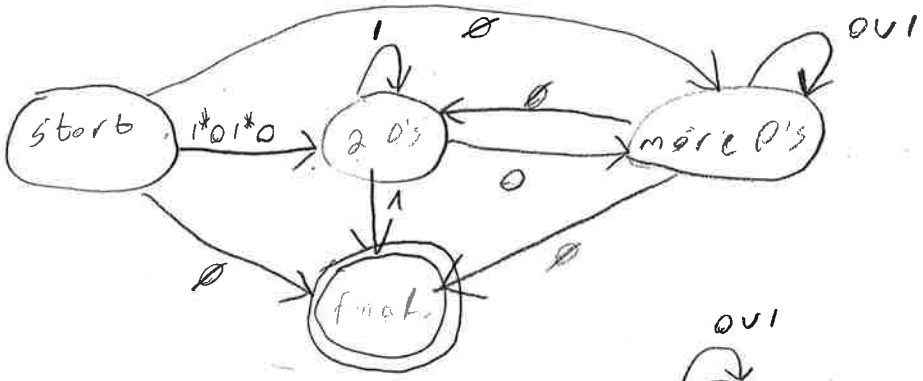
$R_{2m} = 0 \cup \emptyset$
$= 0$

$R_{22} = 1 \cup \emptyset$
$= 1$

$R_{2f} = \lambda \cup \emptyset$
$= \lambda$

$R_{mm} = (0 \cup 1) \cup \emptyset$
$= 0 \cup 1$

$R_{mf} = \emptyset \cup \emptyset$
$= \emptyset$



$R_{s2} = \emptyset \cup (1^*0)1^*0$
$= 1^*01^*0$

$R_{sm} = \emptyset \cup (1^*0)1^*\emptyset$
$= \emptyset$

$R_{sf} = \emptyset \cup 1^*01^*\emptyset$
$= \emptyset$

$R_{2f} = \lambda \cup \emptyset$
$= \lambda$

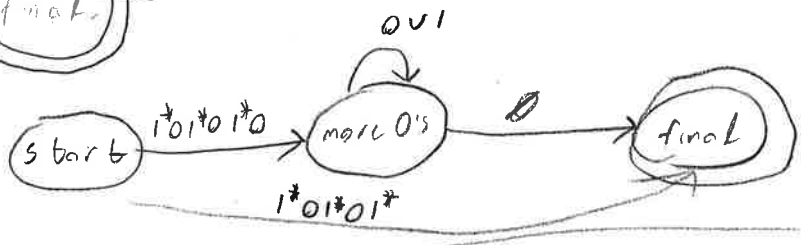
$R_{22} = 1 \cup \emptyset$
$= 1$

$R_{2m} = 0 \cup \emptyset$
$= 0$

$R_{m2} = \emptyset \cup \emptyset$

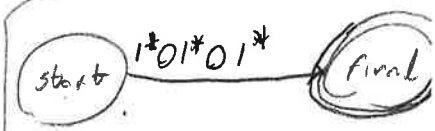
$R_{mf} = \emptyset \cup \emptyset$

$R_{mm} = (0 \cup 1) \cup \emptyset$



$R_{sm} = \emptyset \cup 1^*01^*01^*0$
$= 1^*01^*01^*0$

$R_{sf} = \emptyset \cup 1^*01^*01^* \lambda$
$1^*01^*01^*$

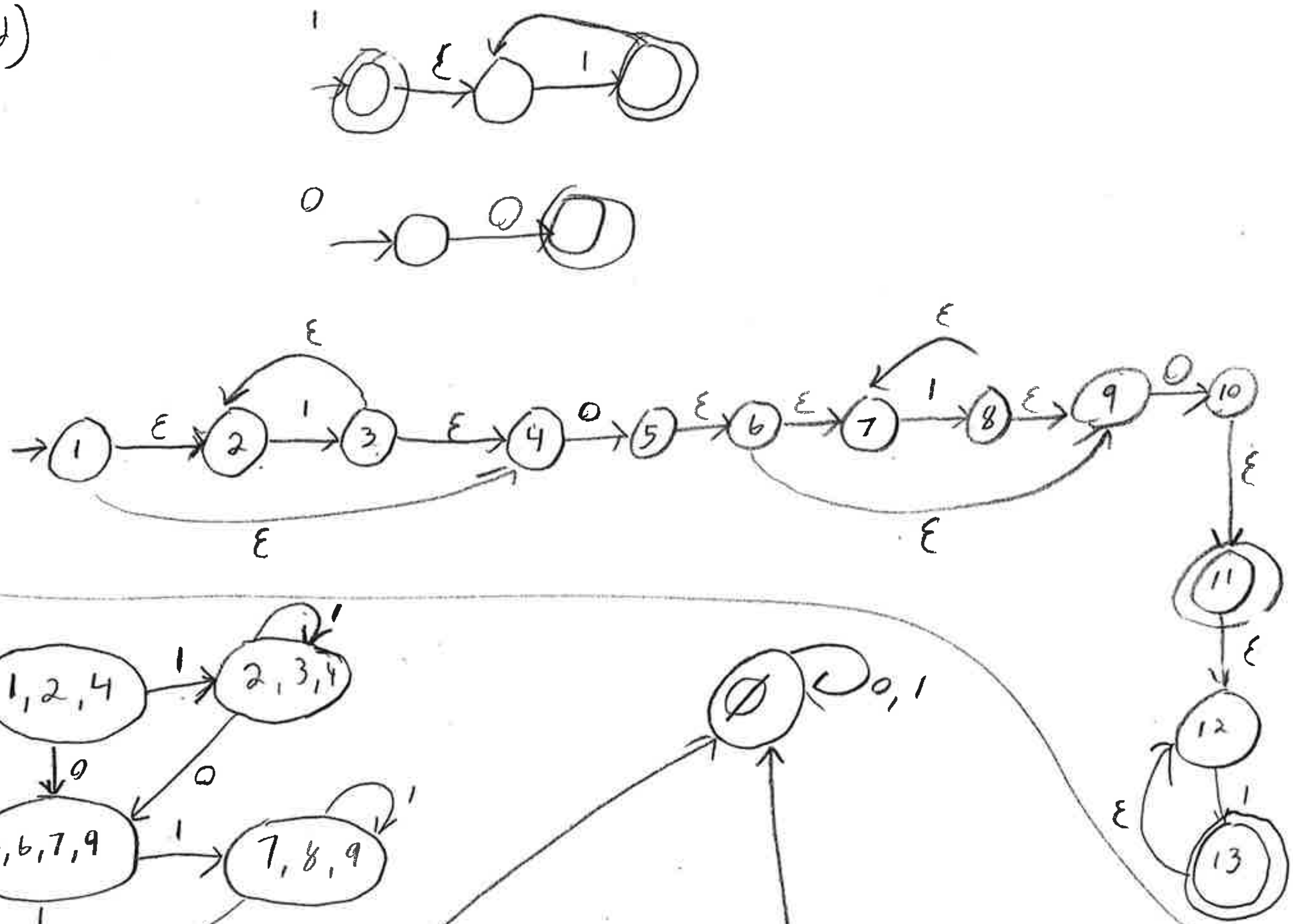


$R_{sf} = 1^*01^*01^* \cup 1^*01^*01^*0(0 \cup 1)^*\emptyset$

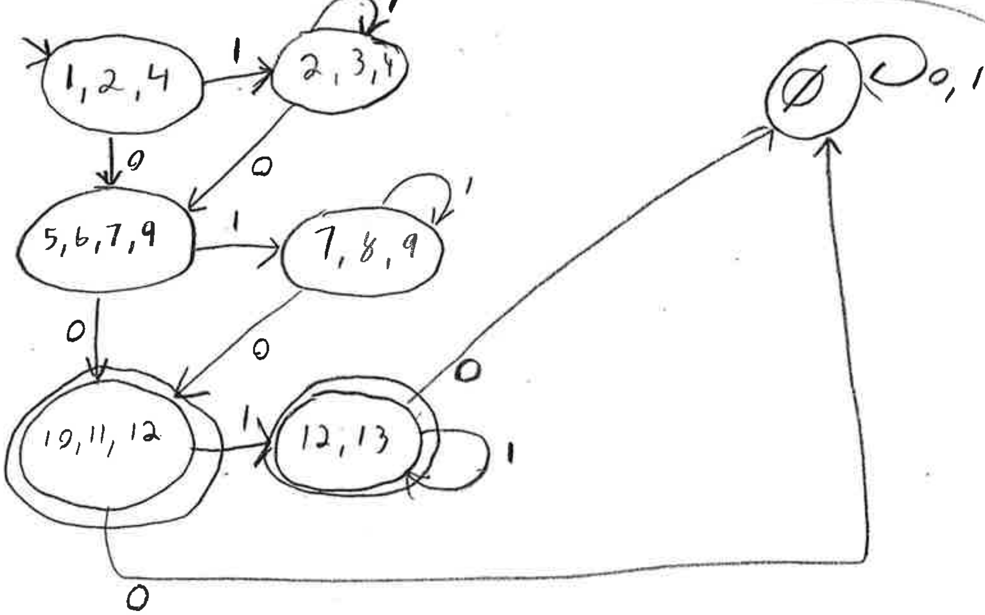
$= 1^*01^*01^*$

$1^* 0 1^* 0 1^*$

1d)

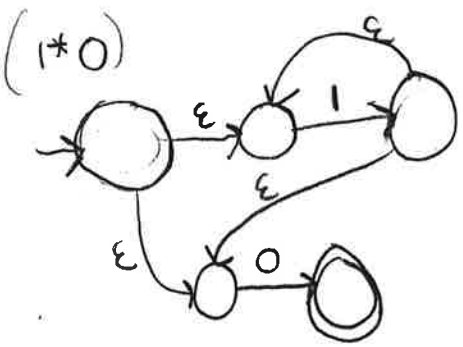


1e)

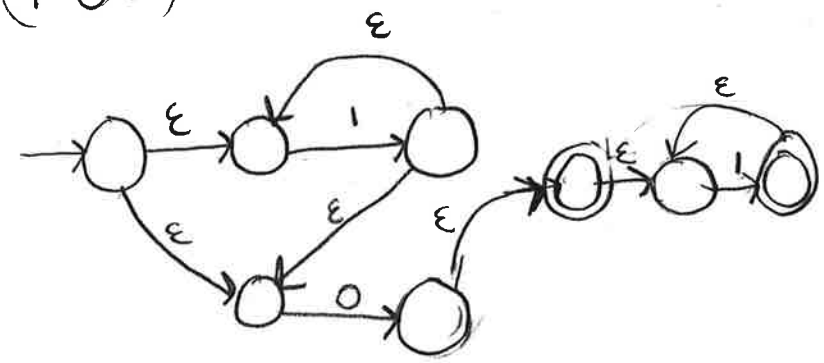


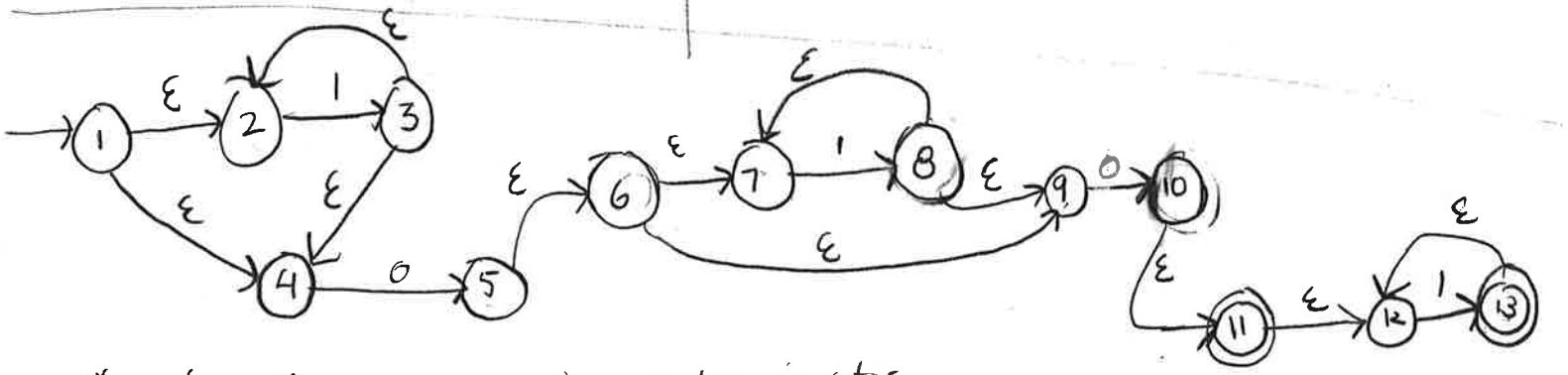
$1^*01^*01^*$

(1^*0)

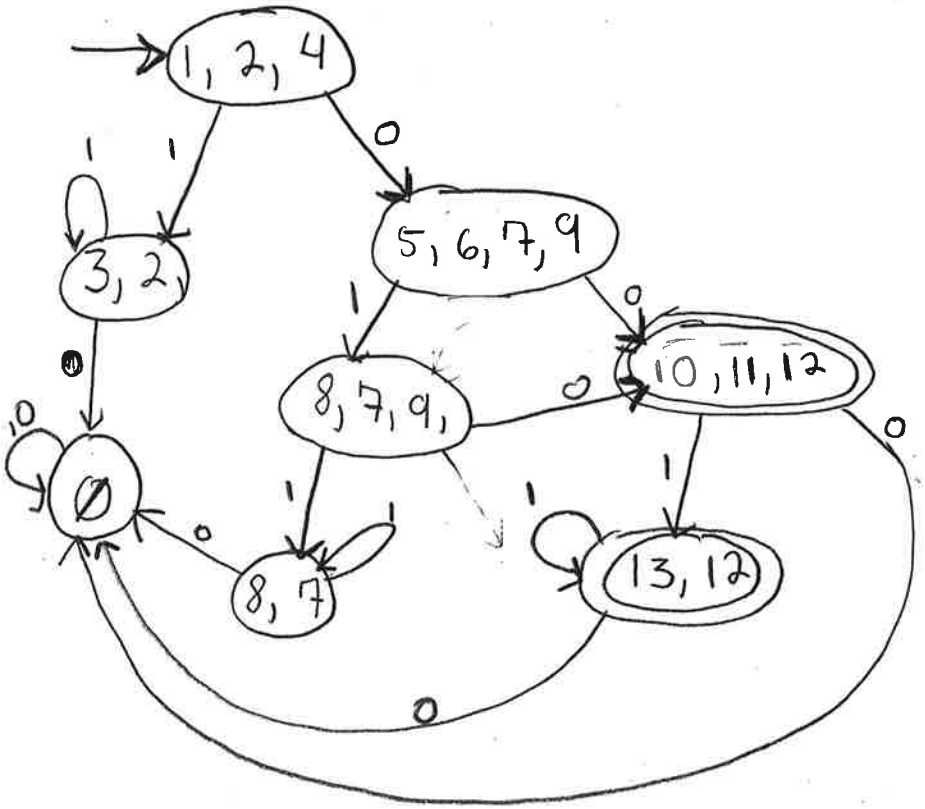


(1^*01^*)





$1^*01^*01^*$ Non-Deterministic



Deterministic

59/60 = (9.8/10)

2. *Beyond finite automata: pushdown automata and context-free grammars:*

9/10 2a. Prove that the language consisting of all expressions that contain twice as many a's as b's is not regular.

10/10 2b. Use a general algorithm to transform the finite automaton from the Problem 1a into a context-free grammar (CFG). Show, step-by-step, how this CFG will generate the word 01101.

10/10 2c. For the context-free grammar from the Problem 2b, show how the word 01101 can be represented as uvxyz in accordance with the pumping lemma.

10/10 2d. Use a general algorithm to translate the CFG from 2b into Chomsky normal form.

10/10 2e. Use a general algorithm to translate the CFG from 2b into an appropriate push-down automaton. Explain, step-by-step, how this automaton will accept the word 01101.

2f. Use the general stack-based algorithms to show:

10/10
- how the compiler will transform the expression (2 − 3) * (1 + 2) into inverse Polish notation, and
- how it will compute the value of this expression.

2a) $L = \{a^{2n} b^n, n = 1, 2, 3, \ldots\}$ Not exactly

Let's assume that L is regular then by pumping lemma $\exists p$ such that every word s from L whose length $\geq p$ can be represented as $s = xyz$ where $len(y) > 0$, $len(xy) \leq p$ & for every i $xy^i z \in L$

Let's take $s = a^{2p} b^p$, here $len(s) = 3p \geq p$ so by pumping lemma there exists a corresponding xyz

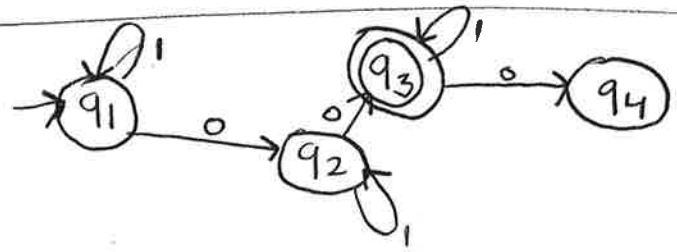
since s starts xy & $len(xy) \leq p$ this means that x & y are in a's

so when we take $xyyz$ we add a's but the number of b's does not change so the number of a's in $xyyz$ is no longer twice the number of b's so $xyyz \notin L$ but by pumping lemma $xyyz \in L$ - a contradiction this contradiction shows that our assumption is wrong so L is not regular

adding more a's to the string by pumping lemma shows that now the new s will not be in the language and so we get a contradiction.

Therefore the language L is not regular.

2.b)



01101

$Q_1 \rightarrow 1Q_1$

$Q_1 \rightarrow 0Q_2$

$Q_2 \rightarrow 1Q_2$

$Q_2 \rightarrow 0Q_3$

$Q_3 \rightarrow 1Q_3$

$Q_3 \rightarrow 0Q_4$

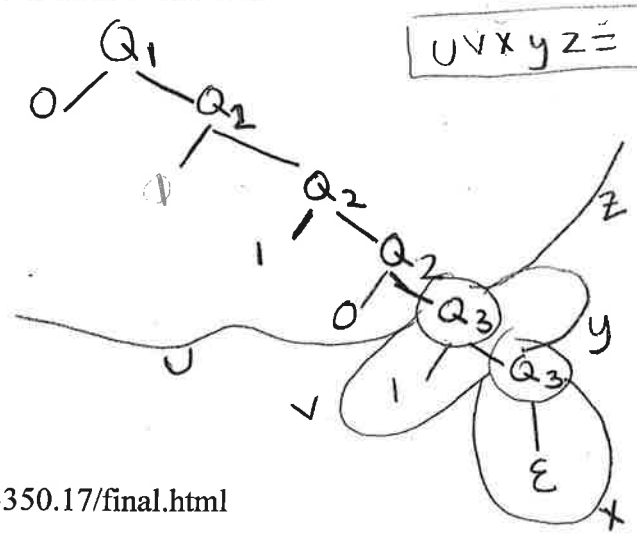
$Q_3 \rightarrow \varepsilon$

$Q_1 \rightarrow 0Q_2 \rightarrow 01Q_2 \rightarrow 011Q_2 \rightarrow 0110Q_3 \rightarrow 01101Q_3 \rightarrow 01101$

accepted

2.c)

uvxyz = 01101



u = 0110

v = 1

x = ∧

y = ∧

z = ∧

2d)

$S_0 \to Q_1$

$Q_1 \to 1Q_1$

$Q_1 \to 0Q_2$

$Q_2 \to 1Q_2$

$Q_2 \to 0Q_3$

$Q_3 \to 1Q_3$

$Q_3 \to 0Q_4$

$Q_3 \to \varepsilon$

Step 2

$Q_2 \to 0$ ✓

$Q_3 \to 1$ ✓

Step 3

$S_0 \to 1Q_1$

$S_0 \to 0Q_2$

Step 4

$U_0 \to 0$ ✓

$U_1 \to 1$ ✓

$Q_1 \to U_1 Q_1$ ✓

$Q_1 \to U_0 Q_2$ ✓

$Q_2 \to U_1 Q_2$ ✓

$Q_2 \to U_0 Q_3$ ✓

$Q_3 \to U_1 Q_3$ ✓

$Q_3 \to U_0 Q_4$ ✓

$S_0 \to U_1 Q_1$ ✓

$S_0 \to U_0 Q_2$ ✓

2b) $Q_0 \to 1 Q_0$

$Q_0 \to 0 Q_1$

$Q_1 \to 1 Q_1$

$Q_1 \to 0 Q_2$

$Q_2 \to 1 Q_2$

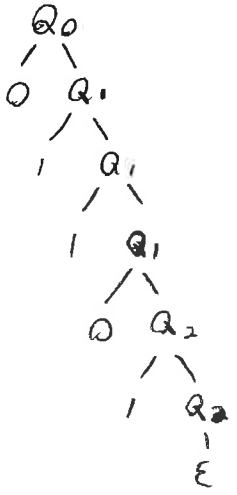
$Q_2 \to 0 Q_m$

$Q_m \to 0 Q_m$

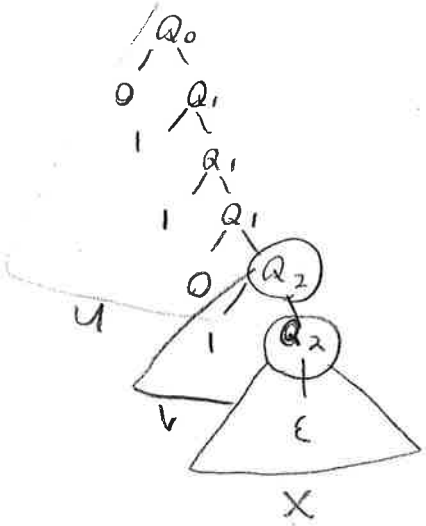
$Q_m \to 1 Q_m$

$Q_2 \to \varepsilon$

01101



2c)



u = 0 110

v = 1

x = ε

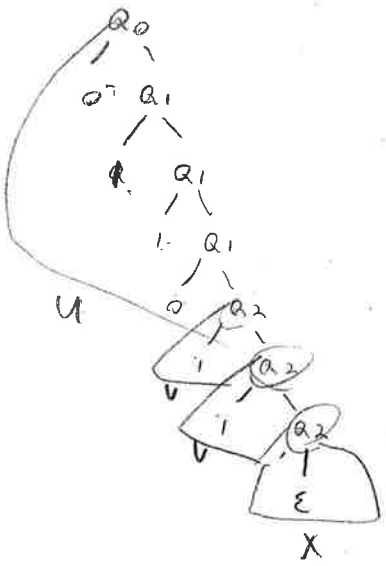
y = ε

z = ε

u v v x y y z

0 1 1 0 1 1 1 ε ε ε

still accepted



2d)

~~$Q_0 \to 1 Q_0$~~

~~$Q_0 \to 0 Q_1$~~

~~$Q_1 \to 1 Q_1$~~

~~$Q_1 \to 0 Q_2$~~

~~$Q_2 \to 0 Q_m$~~

~~$Q_2 \to 1 Q_2$~~

~~$Q_1 \to \varepsilon$~~

~~$Q_m \to 1 Q_m$~~

~~$Q_m \to 0 Q_m$~~

~~$S_0 \to Q_0$~~

$Q_1 \to 0$

$Q_2 \to 1$

~~$S_0 \to 1 Q_0$~~

~~$S_0 \to 0 Q_1$~~

$V_0 \to 0$

$V_1 \to 1$

$Q_0 \to V_1 Q_0$

$Q_0 \to V_0 Q_1$

$Q_1 \to V_1 Q_1$

$Q_1 \to V_0 Q_2$

$Q_2 \to V_1 Q_2$

$Q_2 \to V_0 Q_m$

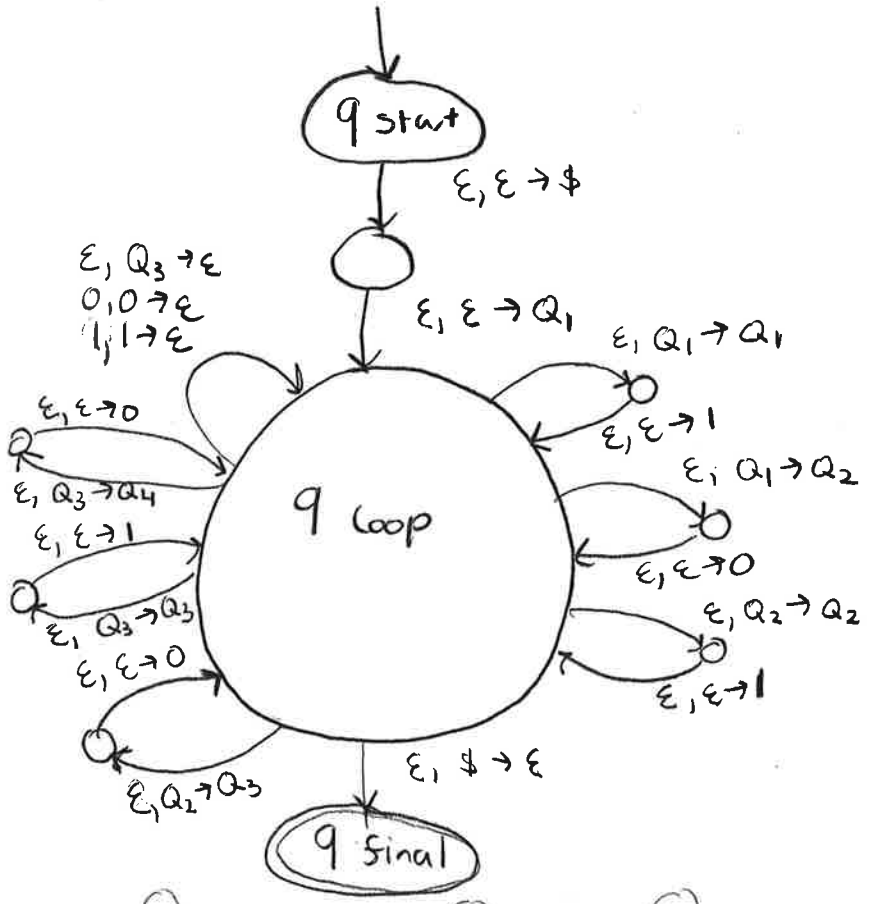
$Q_m \to V_1 Q_m$

$Q_m \to V_0 Q_m$

$S_0 \to V_1 Q_0$

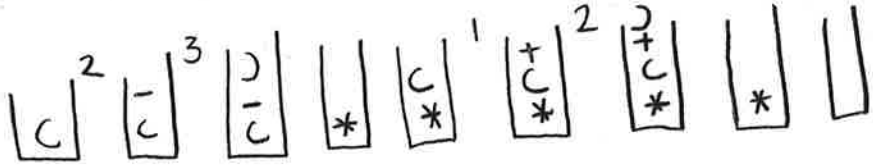
$S_0 \to V_0 Q_1$

2e) $Q_1 \to 1\,Q_1$
$Q_1 \to 0\,Q_2$
$Q_2 \to 1\,Q_2$
$Q_2 \to 0\,Q_3$
$Q_3 \to 1\,Q_3$
$Q_3 \to 0\,Q_4$
$Q_3 \to \varepsilon$



0 1 1 0 1

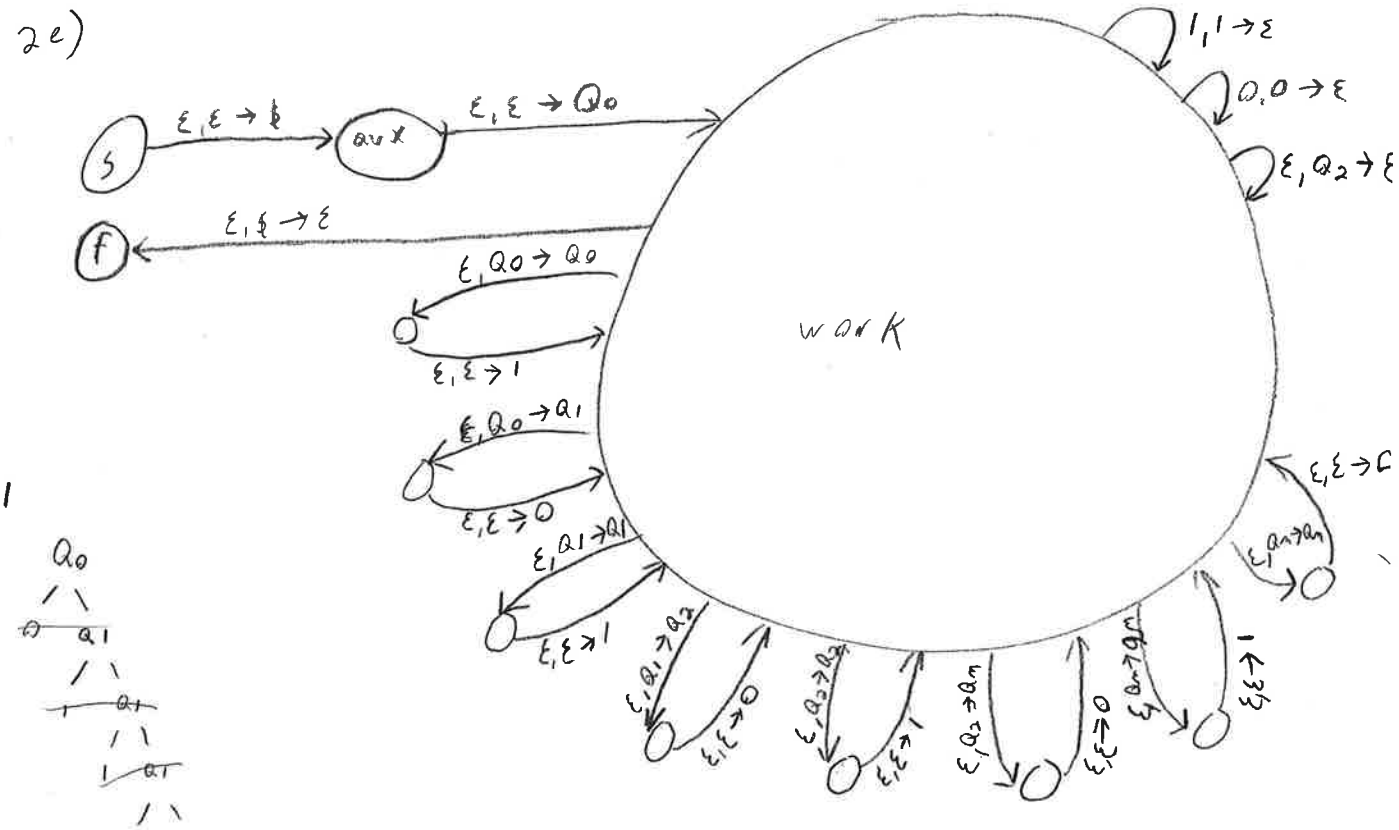
2 f) (2-3)* (1+2) =



2 3 - 1 2 + *

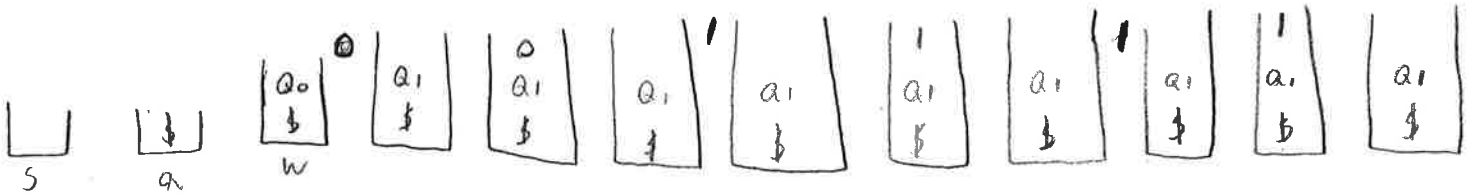
answer : -3

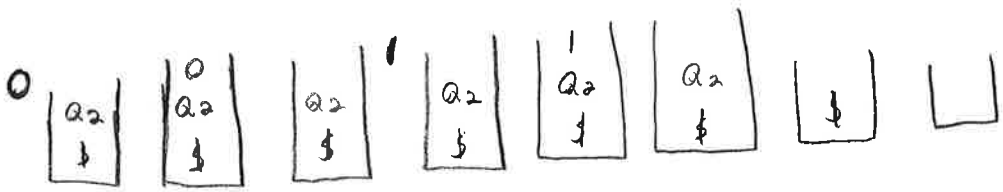
2e)


S
aux
F
work
ε, ε → $
ε, ε → Q0
ε, $ → ε
ε, Q0 → Q0
ε, ε → 1
ε, Q0 → Q1
ε, ε → 0
ε, Q1 → Q1
ε, ε → 1
ε, Q1 → Q2
ε, ε → 0
ε, Q2 → Q2
ε, ε → 1
ε, Q2 → Qa
ε → 0
ε, Qa → Qa
ε → 1
1, 1 → ε
0, 0 → ε
ε, Q2 → ε
ε, ε → ε
ε, Qa → ε

01101

Q0
Q1
Q1
Q1
Q2
Q2
ε

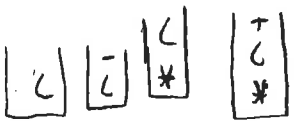

S
q
w
Q0 $
0
Q1 $
0
Q1 $
Q1 $
1
Q1 $
1
Q1 $
1
Q1 $
1
Q1 $
1
Q1 $
1
Q1 $

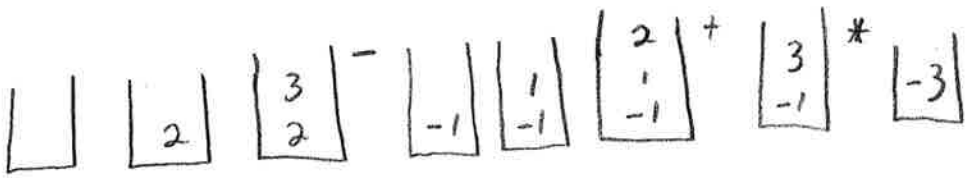

0
Q2 $
0
Q2 $
Q2 $
1
Q2 $
1
Q2 $
Q2 $
$
$

2 f)

$(2-3)*(1+2)$

$2\ 3\ -\ 1\ 2\ +\ *$





52/40 = (13/10)

3. *Beyond pushdown automata: Turing machines*

14/10

3a. Prove that there exists a language that is not context-free and therefore, cannot be recognized by a pushdown automaton. You can use the same language we had in class or -- for extra credit -- some other language.

28/20

3b-c. Use a general algorithm to design a Turing machine that accepts exactly all sequences accepted by a finite automaton from Problem 1a. Show, step-by-step, how this Turing machine will accept the word 01101. Describe, for each step, how the state of the tape can be represented in terms of states of two stacks.

10/10

3d. Design a Turing machine for computing n + 2 in binary code. Trace it for the binary number n = 10_2 (which is 2 decimal); the result of the computation should $2 + 2 = 4_{10}$, i.e., 100_2

3a) $L=\{1^n0^n1^n0^n, n=0,1,2,3\}$

Lets assume L can be recognized by CFG, then by pumping lemma there exists a p such that every word s from L whose length is at least p can be represented as $s = uvxyz$ where $len(xy) > 0$ $len(vxy) \leq p$ & for i $uv^ixy^iz \in L$

Lets take $s = 1^p0^p1^p0^p \in L$ here $len(s) = 4p \geq p$ so it can be represented as uvxyz by pumping lemma Lets consider all possible cases that hold $len(vxy) \leq p$. we take $uvvxyyz \in L$ but

if vxy is in the first 1's first 0's, second is or second 0's means that when we pump we will add more of that symbol disrupting the balance

if vxy is in first 0's & first 1's then if we pump we will have more first 1's & first 0's than second is & second 0's

similarly if vxy is in (first 0's & second 1's & (second 1's & second 0's)

any combination gives us a contradiction. since the word is $\notin L$ it cannot be recognized by PDA & is not context free

44/40 = 11/10

3. *Beyond pushdown automata: Turing machines*

14/10

3a. Prove that there exists a language that is not context-free and therefore, cannot be recognized by a pushdown automaton. You can use the same language we had in class or -- for extra credit -- some other language.

20/20

3b-c. Use a general algorithm to design a Turing machine that accepts exactly all sequences accepted by a finite automaton from Problem 1a. Show, step-by-step, how this Turing machine will accept the word 01101. Describe, for each step, how the state of the tape can be represented in terms of states of two stacks.

10/10

3d. Design a Turing machine for computing n + 2 in binary code. Trace it for the binary number n = 10_2 (which is 2 decimal); the result of the computation should $2 + 2 = 4_{10}$, i.e., 100_2.

3a) Proof by Contradiction

Let's assume that L can be recognized by CFG.

This means there exists a p in all words of the language L with $len(w) \geq p$ and $\exists$ u, v, x, y, z with $w = uvxyz$ and $len(xy) > 0$ and $len(vxy) \leq p$ and $\forall i$ in $uv^i xy^i z$ is still in the language

Take $w = a^{3p} b^{2p} c^p$

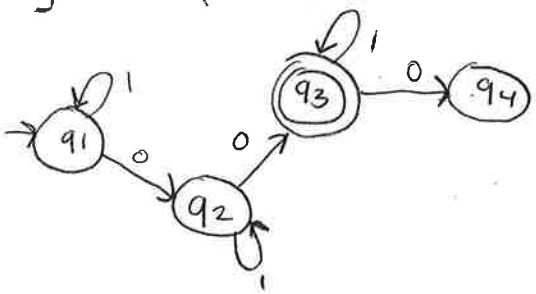
We get a contradiction if the repetition vy is in the single letters a's, b's, or c's because the word will not be in the language if we pump more of that single letter, because we will have too much of that single letter, and it won't be a part of the language.

We get a contradiction if the repetition is in between the a's and b's because if we pump a's and b's, there will be too little c's so the new word won't be in the language.

We get a contradiction if the repetition vy is in between the b's and the c's because pumping b's and c's will make it not part of the language since it will have too many b's and c's in comparison to a's.

We get a contradiction because by pumping lemma, we will get a word that is not part of the language; therefore L cannot be recognized by a pushdown automata.

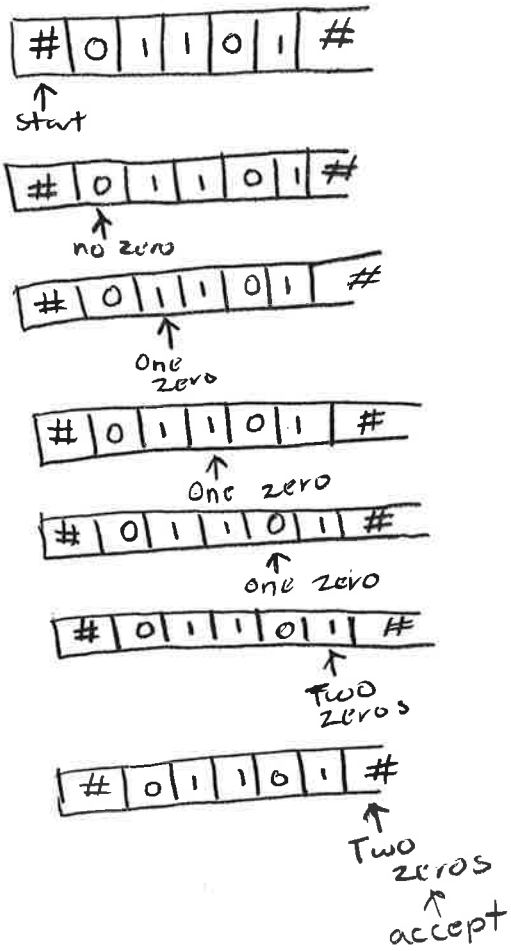
3b-c)



0 1 0 1

Start, # → no zero, R

no zero, 0 → One zero, R
no zero, 1 → no zero, R
no zero, # → reject
One zero, 0 → Two zeros, R
One zero, 1 → One zero, R
One zero, # → reject
Two zeros, 0 → Too many, R
Two zeros, 1 → Two zeros, R
Two zeros, # → accept
Too many, 1 → Too many, R
Too many, 0 → Too many, R
Too many, # → reject



3b)

start $\rightarrow R_1 Q_0$

$Q_0 \; \_ \rightarrow$ reject

$Q_0 \; 1 \rightarrow R$

$Q_0 \; 0 \rightarrow R_1 Q_1$

$Q_1 \; \_ \rightarrow$ reject

$Q_1 \; 1 \rightarrow R$

$Q_1 \; 0 \rightarrow R_1 Q_2$

$Q_2 \; \_ \rightarrow$ accept

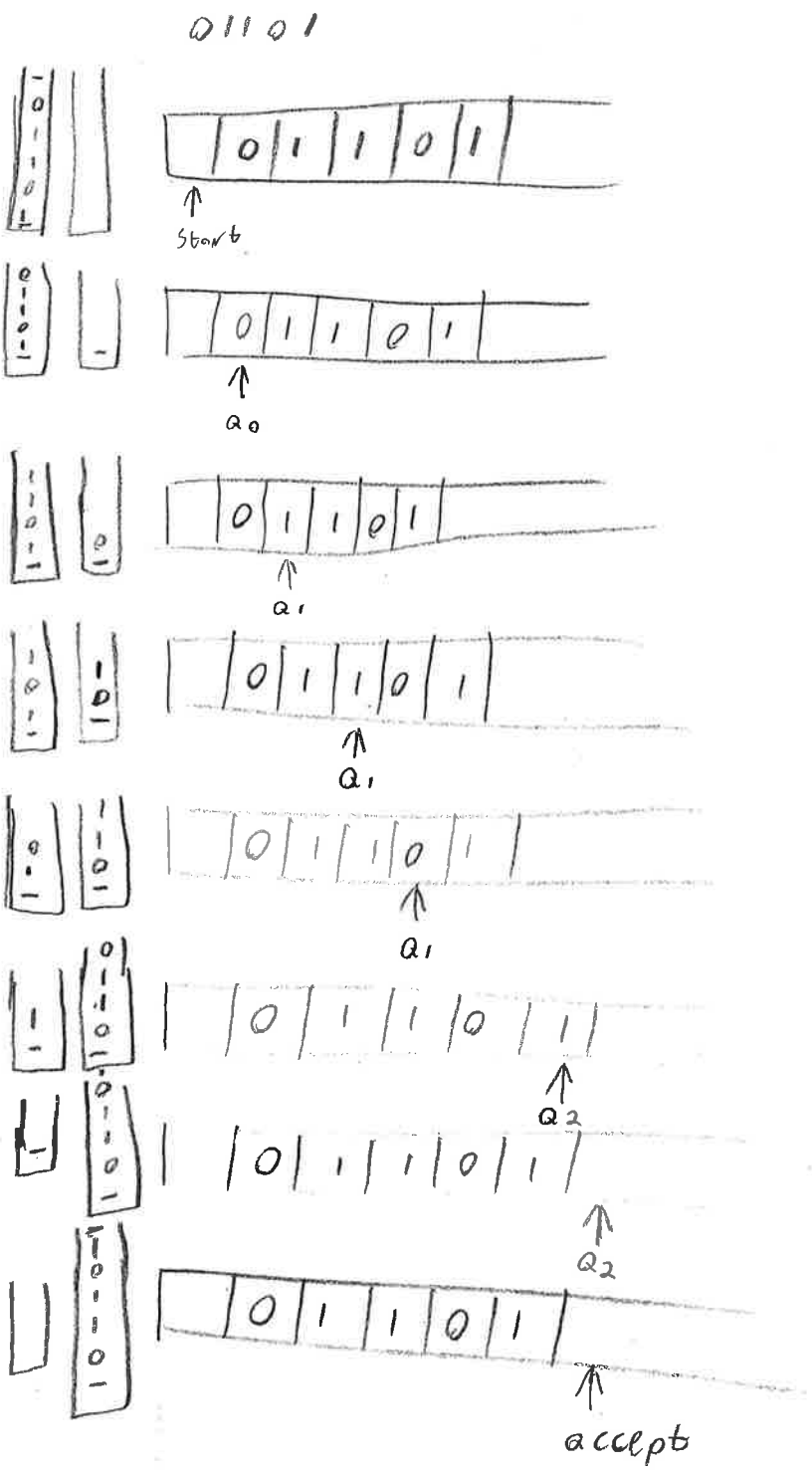
$Q_2 \; 1 \rightarrow R$

$Q_2 \; 0 \rightarrow R_1 Q_m$

$Q_m \; \_ \rightarrow$ reject

$Q_m \; 1 \rightarrow R$

$Q_m \; 0 \rightarrow R$

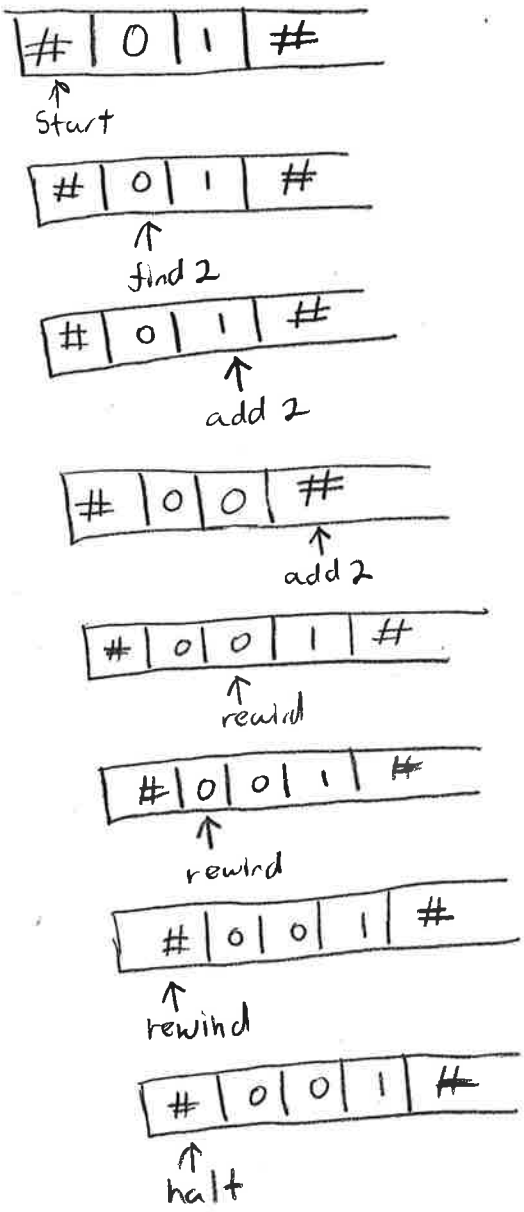


3d) $n+2$

$n = 10_2 = 2$

$2+2 = 4 \rightarrow 100_2$ | 001

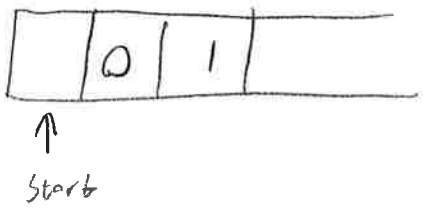
start, # → find 2, R
find 2, 1 → add 2, R
find 2, 0 → add 2, R
find 2, # → 0, add 2, R
add 2, 0 → rewind, 1, L
add 2, # → rewind, 1, L
add 2, 1 → add 2, 0, R
rewind, 0 → rewind L
rewind, 1 → rewind, L
rewind, # → halt

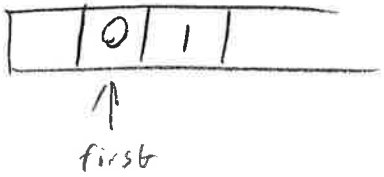


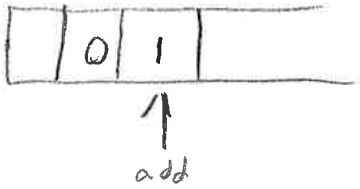
3d)

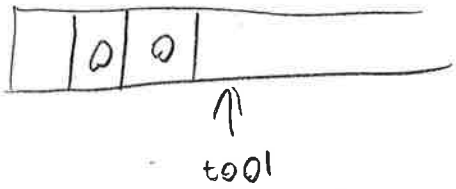
start _ → R, first
first _ → 0, R, add
first 0 → R, add
first 1 → R, add
add 0 → 1, R, Left
add 1 → 0, R, to01
to01 0 → 1, L, done
to01 1 → 0, R
to01 _ → 1, L, done
add _ → 1, L, done
done 0 → L
done 1 → L
done _ → halt

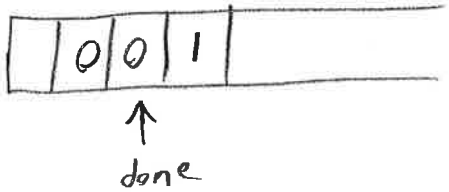
10_2.

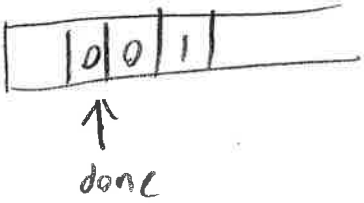


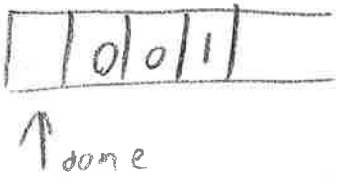


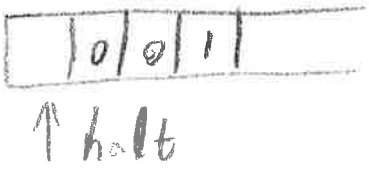












50/50 = (10/10)

4. *Beyond Turing machines: computability*

10/10 4a. Formulate Church-Turing thesis. Is it a mathematical theorem? Is it a statement about the physical world?

10/10 4b. Prove that the halting problem is not algorithmically solvable.

10/10 4c. In the proof that the halting problem is not algorithmically solvable, we use a diagonal function f(n) which was defined as follows:

- $f(n) = j_n(n) + 1$ if n is a valid code of a Java program and j_n halts on n;
- $f(n) = 0$ otherwise.

Write down the first 3 values f(0), f(1), and f(2) of the diagonal function f(n) for the following case:

- $j_0(n) = n + 2$;
- 1 is not a valid numerical code of a program; and
- $j_2(n) = 4 * n$.

10/10 4d. Not all algorithms are feasible, but, unfortunately, we do not have a perfect definition is feasibility. Give two examples:

- an example of an algorithm which is feasible according to the current definition but not practically feasible, and
- an example of an algorithm which is practically feasible but not feasible according to the current definition.

10/10 4e. Briefly describe what is P, what is NP, and what is NP-hard.

4a) Church-Turing Thesis says that anything that can be computed on any physical device can also be computed on a turing machine (or Java program).

Church-turing thesis is a statement about the physical world, not a mathematical theorem.

50/50 = 10/10

4. *Beyond Turing machines: computability*

10/10 4a. Formulate Church-Turing thesis. Is it a mathematical theorem? Is it a statement about the physical world?

10/10 4b. Prove that the halting problem is not algorithmically solvable.

4c. In the proof that the halting problem is not algorithmically solvable, we use a diagonal function f(n) which was defined as follows:

10/10

- $f(n) = j_n(n) + 1$ if n is a valid code of a Java program and j_n halts on n;
- $f(n) = 0$ otherwise.

Write down the first 3 values f(0), f(1), and f(2) of the diagonal function f(n) for the following case:

10/10

- $j_0(n) = n + 2$;
- 1 is not a valid numerical code of a program; and
- $j_2(n) = 4 * n$.

10/10 4d. Not all algorithms are feasible, but, unfortunately, we do not have a perfect definition is feasibility. Give two examples:

- an example of an algorithm which is feasible according to the current definition but not practically feasible, and
- an example of an algorithm which is practically feasible but not feasible according to the current definition.

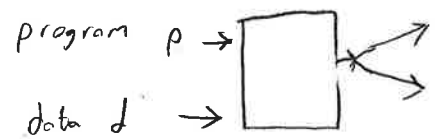
10/10 4e. Briefly describe what is P, what is NP, and what is NP-hard.

4a) anything that can be computed on any physical device can be computed on a turing machine / Java program.

this is not a math theorem it is a statement about the physical world

Halt checker No algorithm is possible that given a program P & data d checks whether P halts on d

we want to prove that halt checker is impossible



problem $001_2 = 1_2$

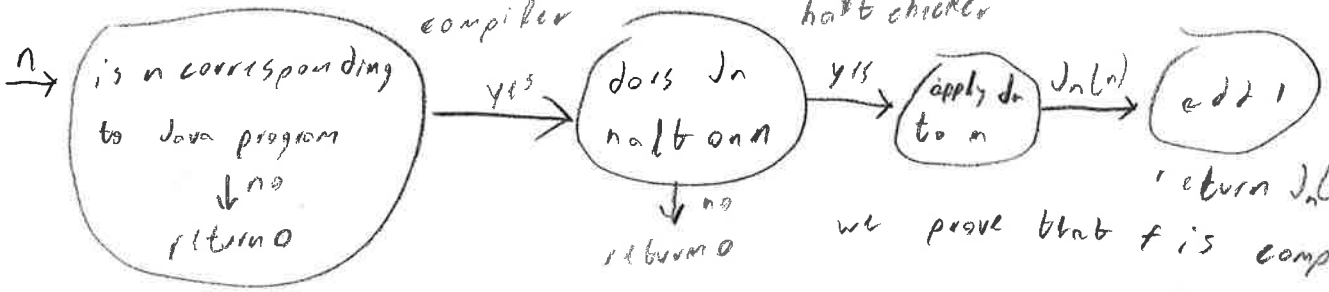
different binary strings may correspond to same numbers
to avoid this we append 1 to the front, only then interpret string as binary
now $1001_2 \neq 11_2$

there is an algorithm that given a natural number n checks whether n corresponds to a Java program, by stripping the first 1 then applying compiler & creating an executable file denoted as J_n

proof by contradiction

lets assume that halt checker exists then we define the following function

$$f(n) \begin{cases} J_n(n)+1 & \text{if } n \text{ is a number corresponding to a Java program \& } J_n \text{ halts on } n \\ 0 & \text{otherwise} \end{cases}$$



we prove that f is computable

f(n) always halts then n denote the number corresponding to Java program that computes f(n), this means that for every input n $J_n(n) = f(n)$ $J_{n_0}(n_0) = f(n_0)$ by definition n_0 is a # corresponding to a Java program J_n is f, it halts so it halts on n_0 as well so

$$f(n_0) = J(n_0)+1$$

4b) No algorithm is possible that given a problem p and data d checks if p halts on d. Prove that a halt checker is impossible.

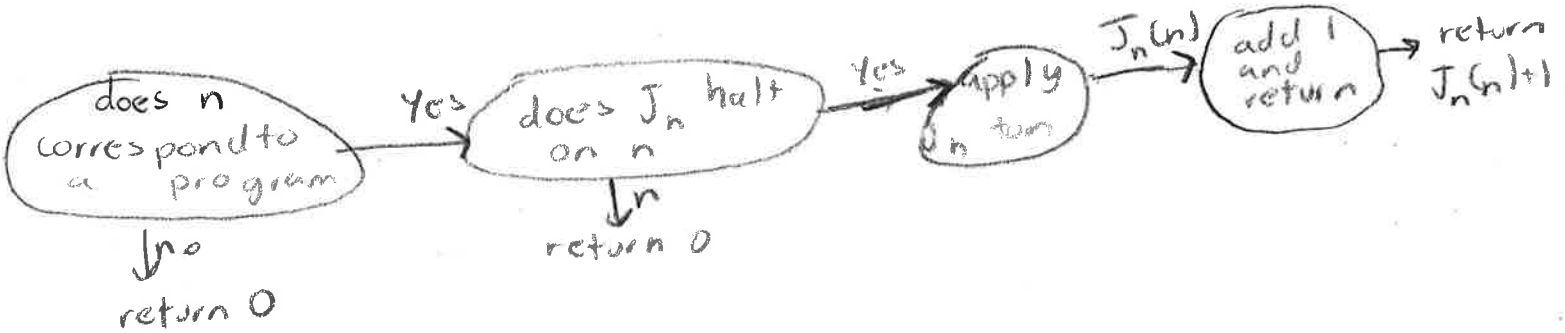
{ true if p halts on d
{ false otherwise

A program is a series of ASCII symbols which are used to turn every symbol into 0's and 1's, that when put together form a long sequence of 0's and 1's. This sequence is what is stored in the pc when typing a program. Different binary strings may lead to the same number. To avoid that, we append a 1 to the beginning of the string and interpret it as a binary number.

$001_2 = 1_2$ ① $001_2 = 9$ ① $1_2 = 3$

There is an algorithm that given a number n, checks if n corresponds to a program. If n is valid, a compiler generates an executable file denoted by J_n

By contradiction
Assume a halt checker exists.
Define function f(n) { $J_n(n)+1$ → If n corresponds to a program and halts on n
{ 0 → otherwise



Let n_0 denote the number corresponding to the program that computes $f(n)$.

For every n, $J_{n_0}(n) = f(n)$

for $n = n_0$

$J_{n_0}(n) = F(n_0)$

By definition: n_0 is a number corresponding to a program.

J_0 is F, it always halts, so it halts on n_0 too.

Therefore, $J_0(n_0) = J_{n_0}(n_0) + 1$

$0 \neq 1$

This is a contradiction, thus a halfchecker does not exist.

4c)

$J_0(n) = n + 2$

1 not valid

$J_2(n) = 4 * n$

	0	1	2
J_0	(2)	3	4
1	–	(–)	–
J_2	0	4	(8)
$f(n)$	3	0	9

4c)

J_n	0	1	2
0	(2)	3	4
1	–	(–)	–
2	9	4	(8)
$f(n)$	3	0	9

4d)

$f_A(n) = 10^{301} n$ feasible by definition but not by practice

$f_A(n) = e^{10^{-101} n}$ feasibly by practice but not by definition

4e)

P: problems that can be solved in feasible time

NP: Once we have a candidate for a solution we can check in feasible time whether it is indeed a solution

NP-hard: every problem from np can be reduced to this problem

4 d)

$F_A(n) = 10^{301} n$, feasible by definition, not feasible in practice.

$F_A(n) = e^{10^{-10} n}$, not feasible by definition, feasible in practice

4 e) P- are problems that can be solved with algorithms that run in polynomial time.

NP- Once there is a candidate for a solution, we can check, in feasible time, whether it is indeed a solution.

NP-Hard - A problem is said to be NP-hard if an algorithm for solving it can be translated into one for solving any other NP-problem, meaning every problem from the class NP can be reduced to this program.