10/10

1. *Finite automata and regular languages:*

1a. Design a finite automaton for recognizing binary sequences that have exactly one or exactly two 1s. 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 1100.

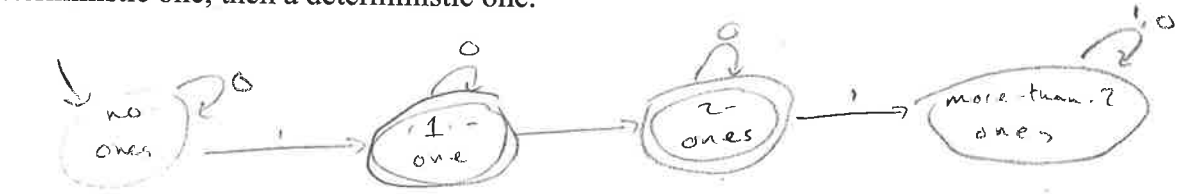
1b. On the example of this automaton, show how the word 1100 can be represented as xyz in accordance with the pumping lemma.

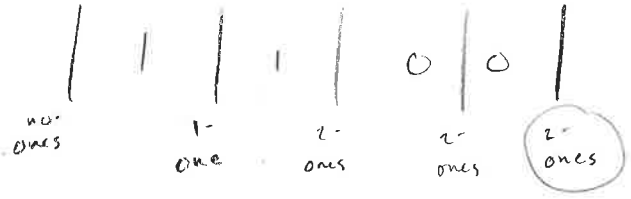
1c. Use a general algorithm to describe a regular expression corresponding to the finite automaton from the Problem 1a.

3.5/4

1d-e. The resulting language can also be described by a regular expression $0^*1(0^* \cup 0^*10^*)$. Use a general algorithm to transform this regular expression into a finite automaton: first a non-deterministic one, then a deterministic one.

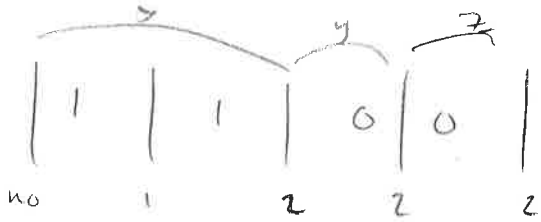
1a.





2 ones is final, 1100 is accepted

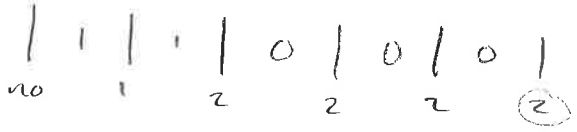
1b.



p = 4 len(w) = 4 ≥ 3 len(xy) = 3 ≤ 4

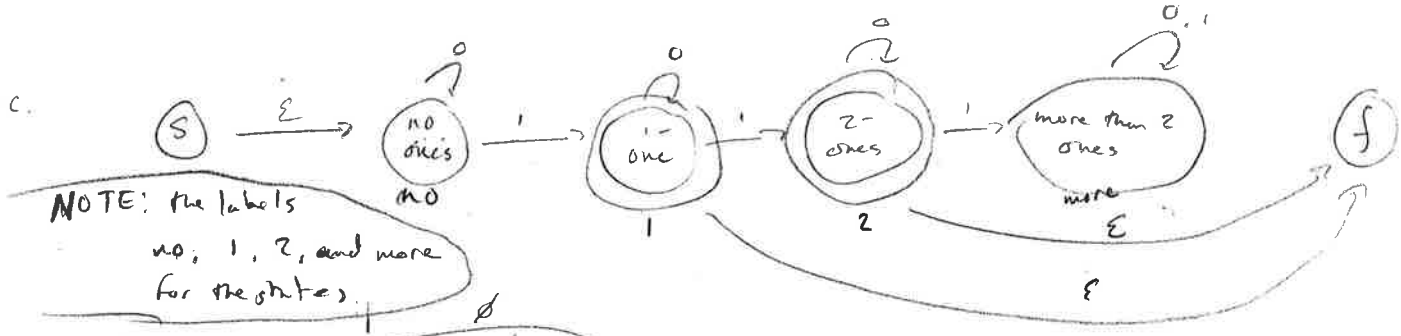
x = 11 y = 0 z = 0

x y y z = 11 0 0 0

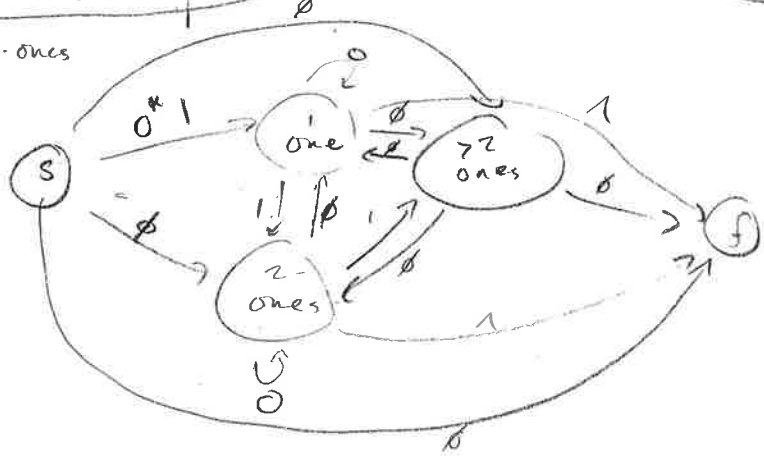


accepted

1c.



K = no-ones



$R'_{s1} = R_{s1} \cup (R_{sno} R_{nono}^* R_{no1})$

$= \emptyset \cup (\wedge \quad 0^* \quad 1$

$= 0^*1$

$R'_{s2} = R_{s2} \cup (R_{sno} R_{nono} R_{no2})$

$\emptyset \cup (\wedge \quad 0^* \quad \emptyset)$

$= \emptyset$

$R'_{2f} = R_{2f} \cup (R_{2no}$

$1 \cup (\emptyset$

$R'_{smore} = R_{smore} \cup ($ $R_{nomore})$

$\emptyset \cup (\quad \cdot \cdot \emptyset)$

$= \emptyset$

$R'_{11} = R_{11} \cup (R_{1no} R_{nono}^*$

$0 \cup (\emptyset \cdots)$

$= 0$

$R'_{12} = R_{12} \cup (R_{1no}$

$1 \cup (\emptyset \cdots)$

$= 1$

$R'_{2more} = R_{2more} \cup (R_{2no}$

$\cup (\emptyset \cdots)$

$R'_{more2} = R_{more2} \cup (R_{moreno}$

$\emptyset \cup (\emptyset \cdots)$

$R'_{21} = R_{21} \cup (R_{2no}$

$= \emptyset) \cup (\emptyset \cdots)$

$R'_{1more} = R_{1more} \cup (R_{1no}$

$\emptyset \cup (\emptyset \cdots)$

$R'_{sf} = R_{sf} \cup (R_{sno} R_{nono}^* R_{nof}$

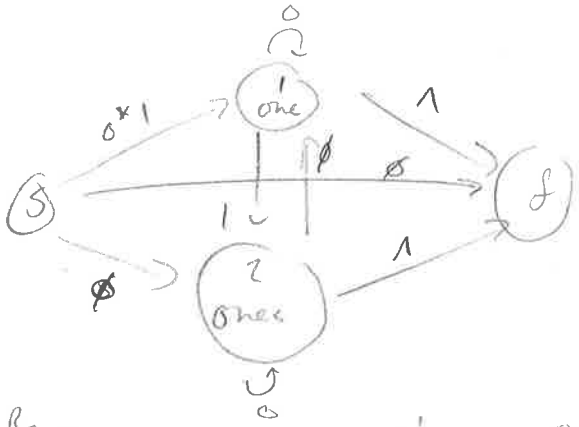
$\emptyset \cup (\wedge \quad 0^* \quad \emptyset)$

$= \emptyset$

$R'_{2f} = R_{2f} \cup (R_{2no}$

$1 \cup (\emptyset$

L = more than 2 ones



$R'_{sf} = R_{sf} \cup (R_{smore}$

$\phi \cup (\phi \cdots \phi)$

$R'_{s1} = R_{s1} \cup (R_{smore}$

$0^*1 \cup (\phi \cdots)$

$R'_{s2} = R_{s2} \cup (R_{smore} R_{more}^* R_{more2})$

$\phi \cup (\phi \quad (01)^* \quad \phi)$

$R'_{11} = R_{11} \cup (R_{1more}$

$0 \cup (\phi$

$R'_{12} = R_{12} \cup (R_{1more}$

$1 \cup (\phi \cdots)$

$R'_{21} = R_{21} \cup (R_{2more} R_{moremore}^* R_{more1}$

$= \phi \cup (1 \quad (01)^* \quad \phi)$

$= \phi$

$R'_{22} = R_{22} \cup (R_{2more} R_{moremore}^* R_{more2})$

$0 \cup (1 \quad (01)^* \quad \phi)$

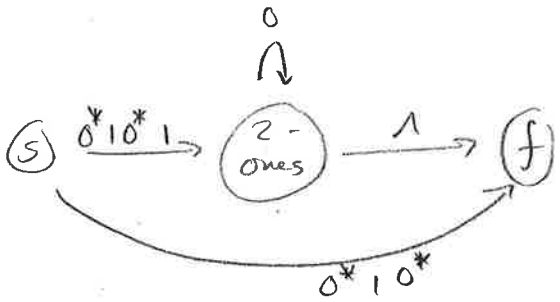
$R'_{2f} = R_{2f} \cup (R_{2more} R_{moremore}^* R_{more f})$

$1 \cup (1 \quad (01)^* \quad \phi)$

$R'_{1f} = R_{1f} \cup (R_{1more}$

$1 \cup (\phi \cdots)$

$k = 1$: one



$R'_{sf} = R_{sf} \cup (R_{s1} R_{11}^* R_{1f})$

$= \emptyset \cup (0^*10^* \lambda)$

$= 0^*10^*$

$R'_{s2} = R_{s2} \cup (R_{s1} R_{11}^* R_{12}$

$\emptyset \quad (0^*10^*1$

$R'_{22} = R_{22} \cup (R_{21}$

$0 \cup (\emptyset \cdots)$

$R'_{2f} = R_{2f} \cup (R_{21}$

$\lambda \cup (\emptyset \cdots)$

$k = 2$ ones

$R'_{sf} = R_{sf} \cup (R_{s2} R_{22}^* R_{2f})$

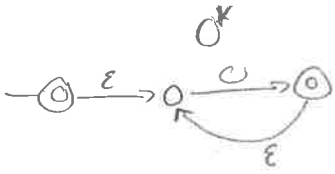
$(0^*10^*) \cup (0^*10^*10^* \lambda)$

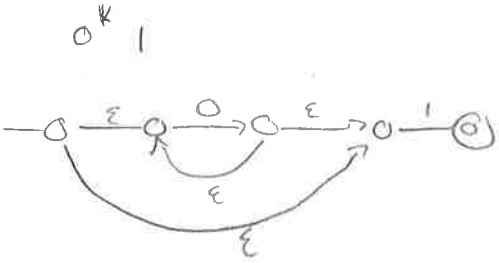
Regular language

$(0^*10^*) \cup (0^*10^*10^*)$

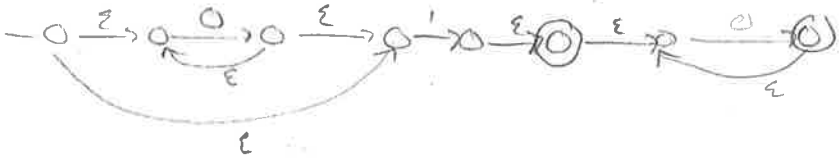
1d. Non deterministic FTC

$0^* 1 (0^* \cup 0^* 1 0^*)$

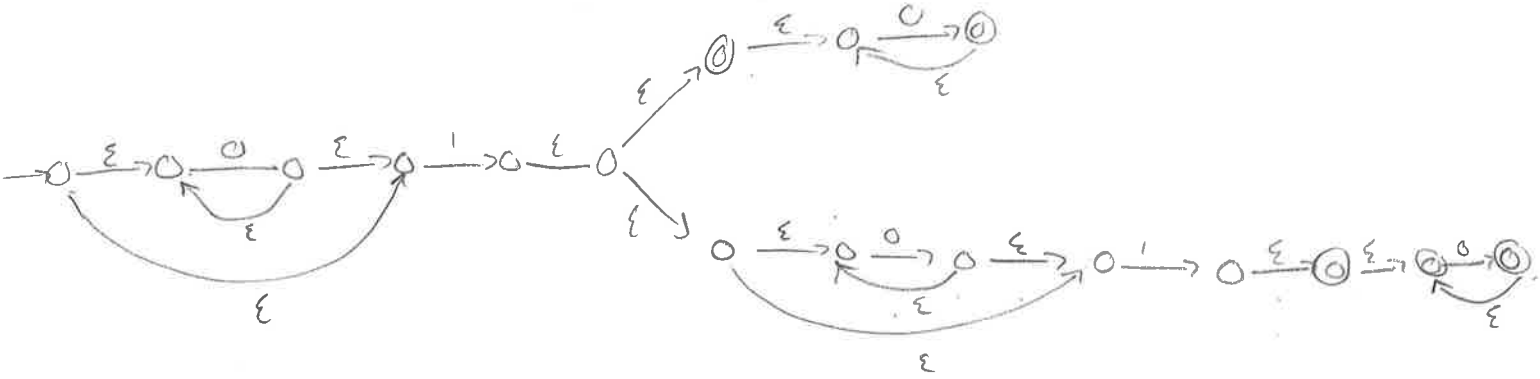




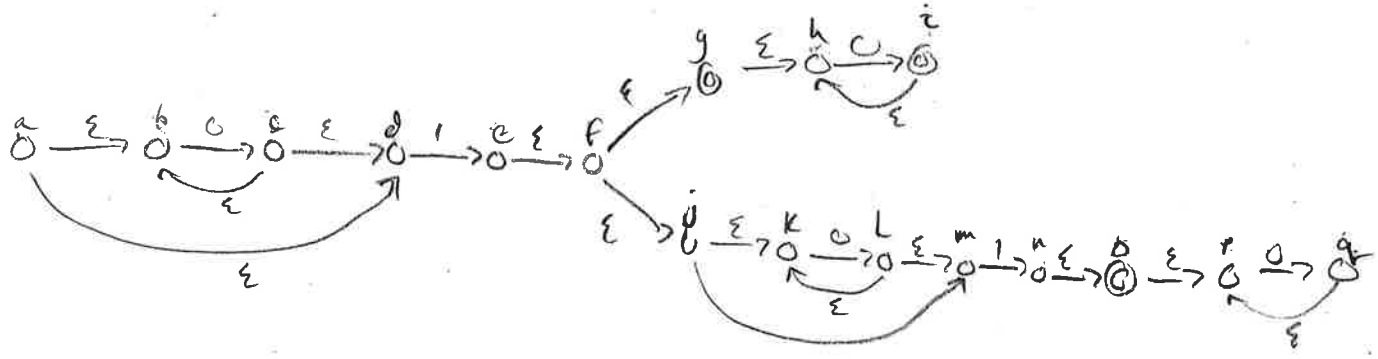
$0^* 1 0^*$



$0^* 1 (0^* \cup 0^* 1 0^*)$

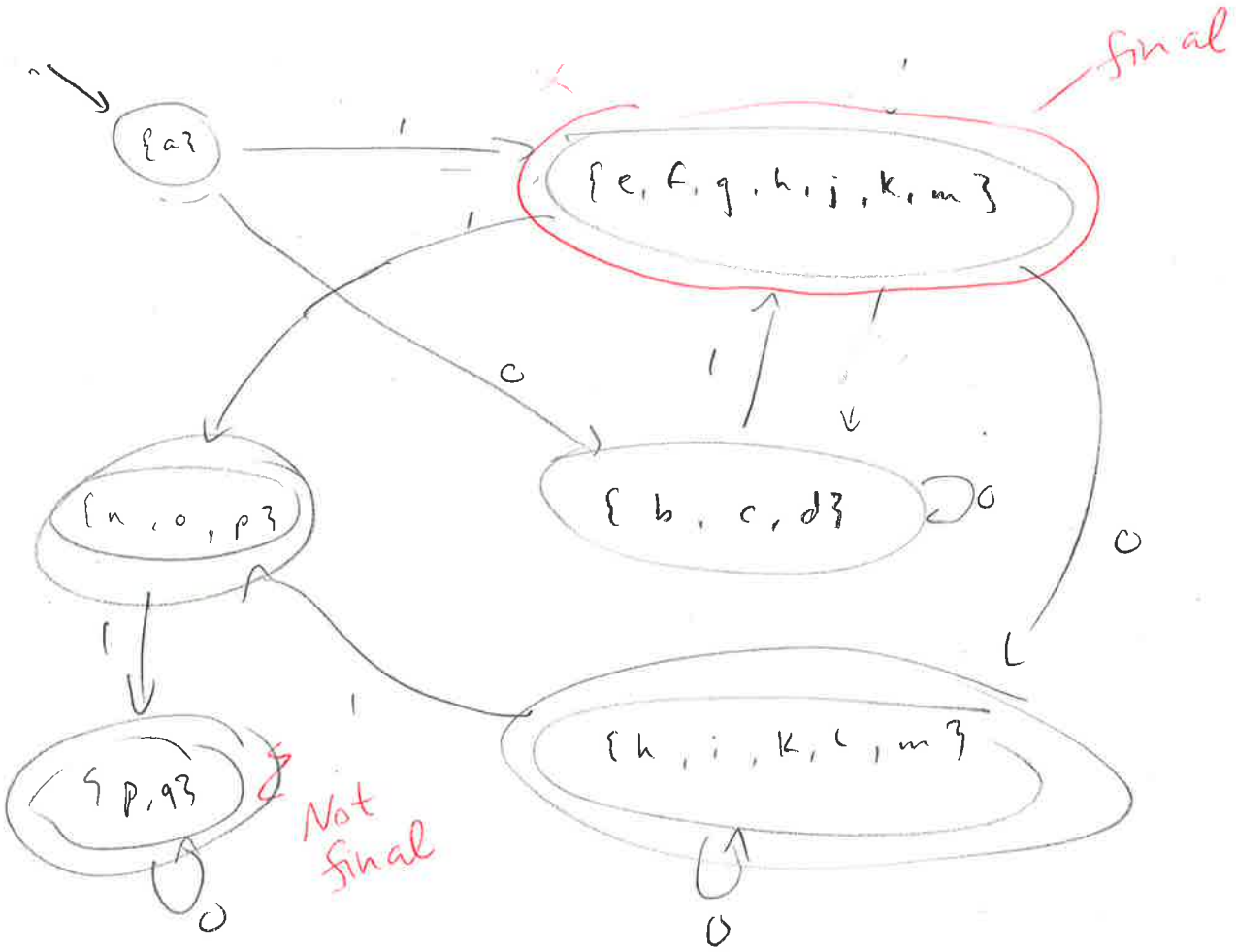


1e.



	1	0
a	e, F, g, h, J, K, m	b, c, d
b	–	b, c, d
c	e, F, g, h, J, K, m	c
d	e, F, g, h, J, K, m	–
e	n, o, p	h, I
F	n, o, p	h, I
g	–	h, I
h	–	h, I
I	–	h, I
J	n, o, p	K, L, m
K	–	K, L, m
L	n, o, p	K, L, m
m	n, o, p	–
n	–	p, q
o	–	p, q
p	–	p, q
q	–	p, q

	1	0
{a}	{e, f, g, h, j, k, m}	{b, c, d}
{e, f, g, h, j, k, m}	{n, o, p}	{h, i, k, l, m}
{b, c, d}	{e, f, g, h, j, k, m}	{b, c, d}
{n, o, p}	-	{p, q}
{h, i, k, l, m}	{n, o, p}	{h, i, k, l, m}
{p, q}		{p, q}



10/12

2. *Beyond finite automata: pushdown automata and context-free grammars:* 1.7/1.7

2a. Prove that the language consisting of all expressions that contain three times as many a's as b's is not regular.

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 1100.

2c. For the context-free grammar from the Problem 2b, show how the word 1100 can be represented as uvxyz in accordance with the pumping lemma.

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

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 1100.

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

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

2a $L = \{a^{3n} b^n\}$ Proof by contradiction. Let's assume that L is a regular language.

$\exists p$ s.t. $\forall w \in L (len(w) \geq p \rightarrow \exists xyz (w = xyz \;\&\; len(y) \geq 0 \;\&\; len(xy) \leq p) \;\&\; \forall i (xy^i z \in L))$

Let us take $w = a^{3p} b^p$ $w = \underbrace{a \ldots a}_{3p\text{-times}} \underbrace{b \ldots b}_{p\text{-times}}$

$len(xy) = 4p \geq p$ so $w = xyz$ $len(xy) \leq p$. so xy is in the first p symbols

$\Rightarrow$ y contains only a's

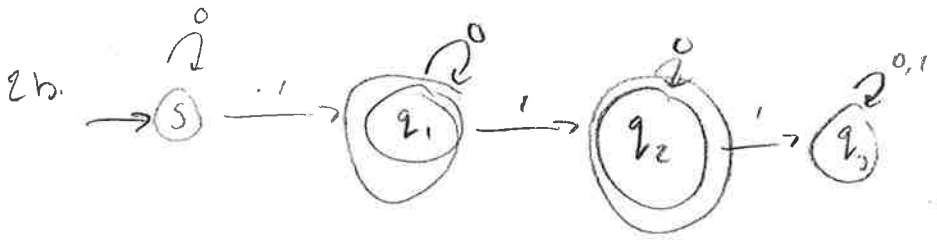
xyyz → we added only a's, now w contains more than 3 times a's than b's

$xyyz \notin L$, but by pumping lemma $xyyz \in L$

We have a contradiction, therefore our assumption is false, and

L is not a reg lang.

Eb.



$S \rightarrow 0S$

$S \rightarrow 1Q_1$

$Q_1 \rightarrow 1Q_2$

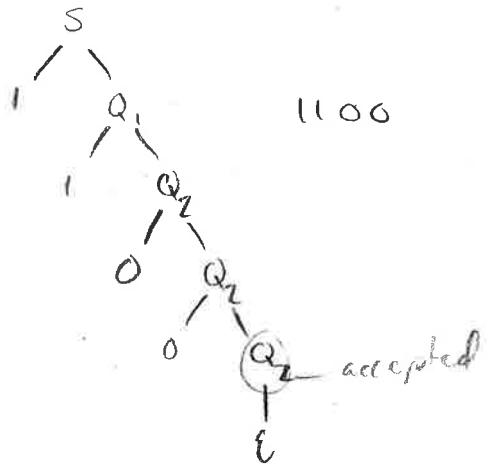
$Q_1 \rightarrow 0Q_1$

$Q_1 \rightarrow \varepsilon$

$Q_2 \rightarrow 0Q_2$

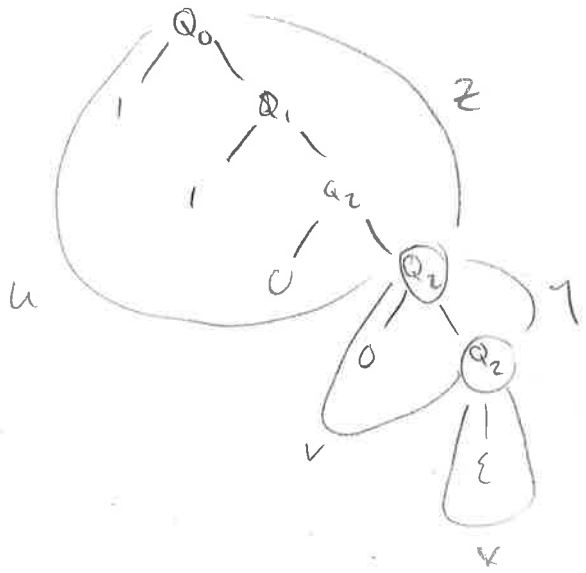
$Q_2 \rightarrow \varepsilon$

$w = 1100$



1100

2c.



$u = 110$

$v = 0$

$x = \varepsilon$

$y = \varepsilon$

$z = \varepsilon$

2d. Using Q_0 instead of S to avoid problems w/ pretim step.

$S \to Q_0$

$Q_0 \to 0Q_0 \mid 1Q_1$

$Q_1 \to 0Q_1 \mid 1Q_2$

$Q_1 \to \varepsilon$

$Q_2 \to 0Q_2$

$Q_2 \to \varepsilon$

$Q_0 \to 1 \mid 0Q_0 \mid 1Q_1$

$Q_1 \to 1 \mid 0Q_1$

$Q_2 \to 0$

$S \to 1 \mid 0Q_0 \mid 1Q_1$

$V_0 \to 0$

$V_1 \to 1$

$Q_0 \to V_0 Q_0$

$Q_0 \to V_1 Q_1$

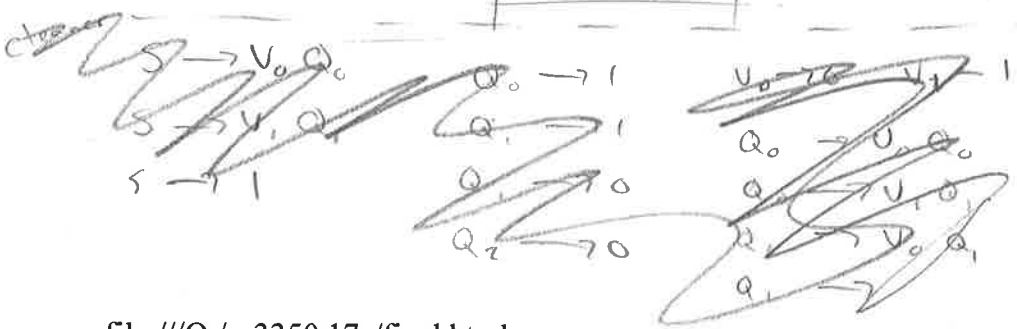
$Q_1 \to V_0 Q_1$

$Q_1 \to V_1 Q_2$

$Q_2 \to V_0 Q_2$

$S \to V_0 Q_0$

$S \to V_1 Q_1$

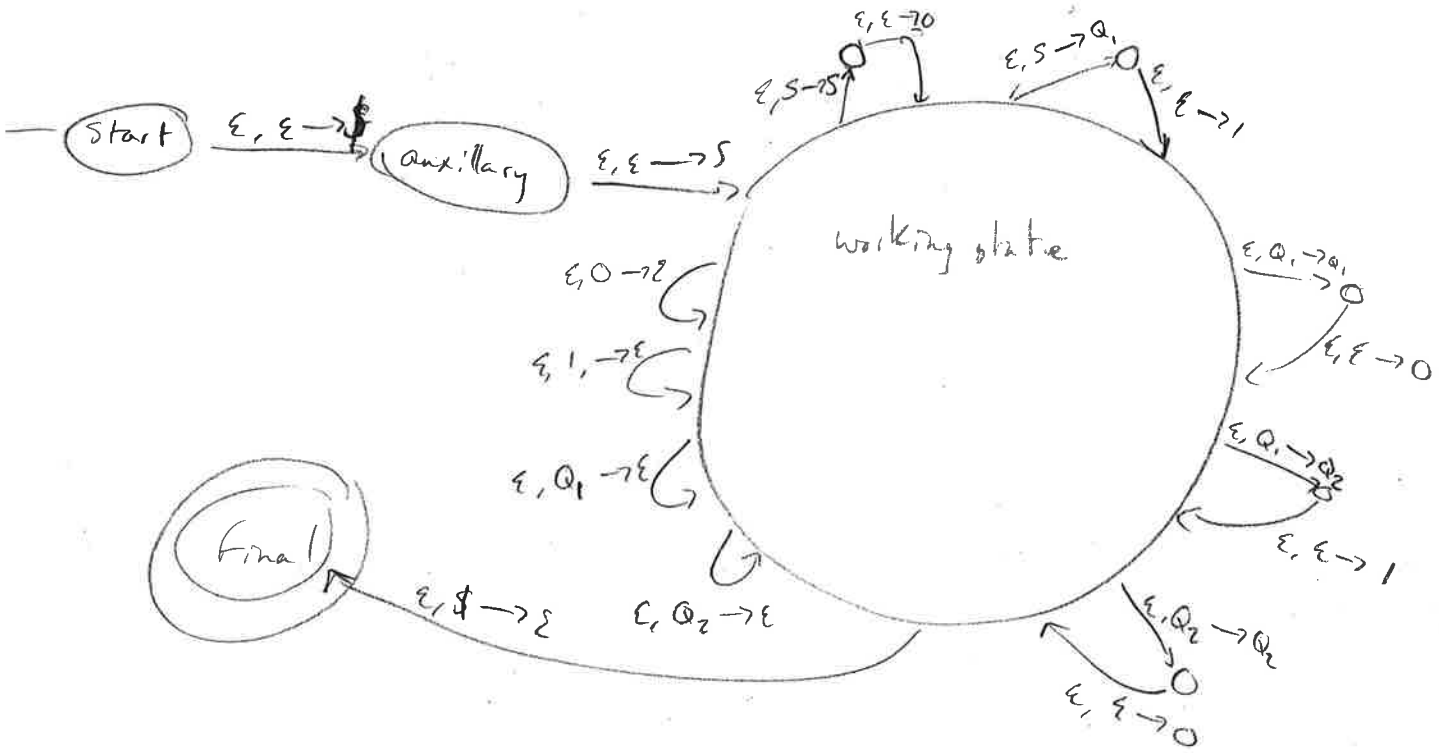


2e.

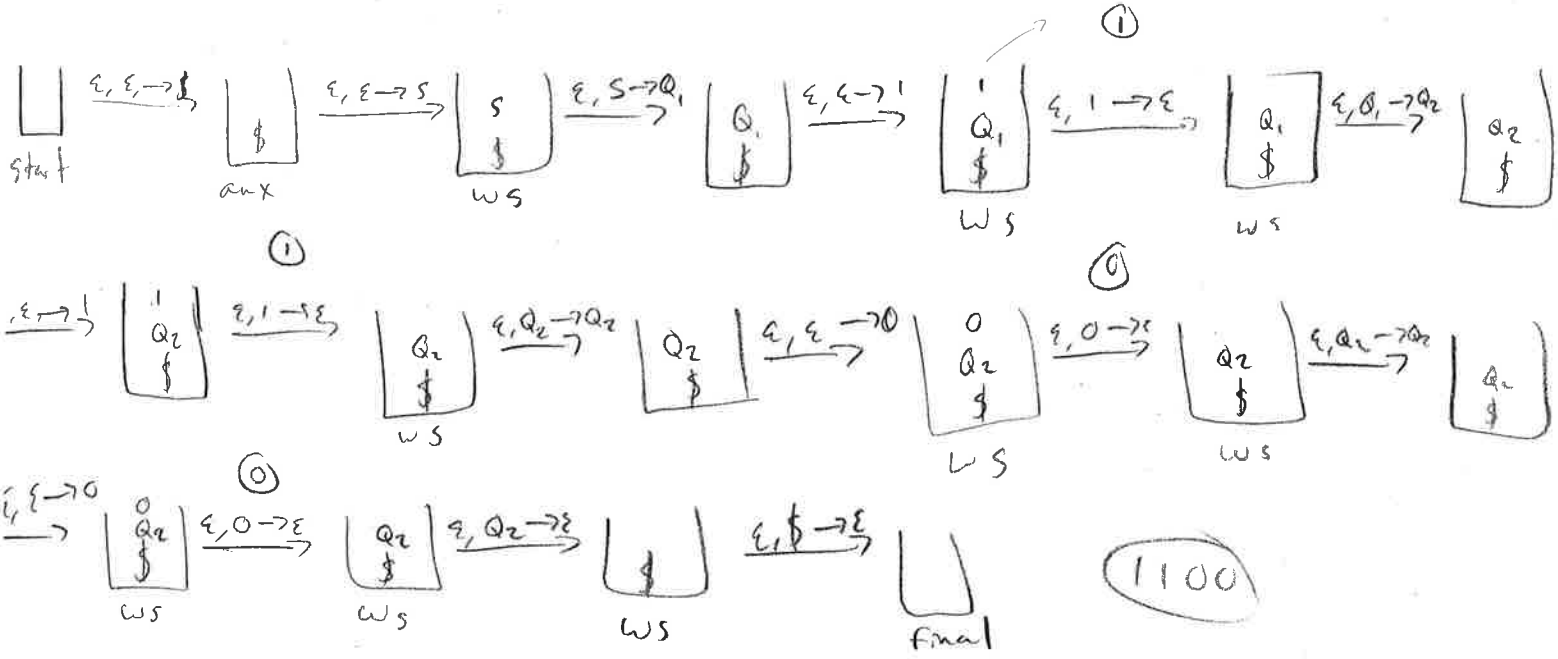
$S \to 0S$ $\quad Q_1 \to 1Q_2$ $\quad Q_2 \to 0Q_2$

$S \to 1Q_1$ $\quad Q_1 \to 0Q_1$ $\quad Q_1 \to \varepsilon$

$Q_2 \to \varepsilon$



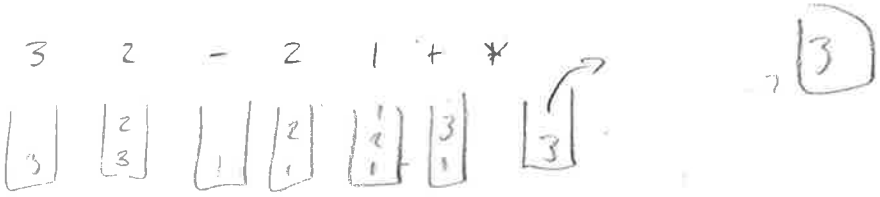
$w = 1100$



2 f.

(3 - 2) * (2 + 1)

3 2 - 2 1 + *



3. *Beyond pushdown automata: Turing machines*

3.5/2.5 (11/12)

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.

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 1100. Describe, for each step, how the state of the tape can be represented in terms of states of two stacks.

3d. Design a Turing machine for computing a + b in unary code. Trace it for the numbers a = 111 (i.e., 3), and b = 1 (i.e., 1); the result of the computation should 3 + 1 = 4, i.e., 1111.

sorry that the whole thing is squished together. :-(

3a. $L = \{ a^n b^{2n} c^{3n} \}$ Proof by contradiction.

Let's assume that L is context free, then by pumping lemma,

$\exists p$ s.t. $\forall w \in L [(len(w) \geq p \rightarrow \exists u,v,x,y,z (w = uvxyz \wedge len(vy) > 0 \wedge len(vxy) \leq p \wedge \forall i (uv^i x y^i z \in L))$

Let's take $w = a^p b^{2p} c^{3p}$

a ... a b ... b c ... c $\longrightarrow len(w) = 6p \geq p$ so

p-times 2p-times 3p-times

$w = uvxyz$

$len(vxy) \leq p$, so where can vxy be? Let's look at uv^2xy^2z in each case.

case 1: it can be in a's: we only add a's, not b's or c's and change the 1:2:3 ratio

case 2: it can be in a's and b's: we only add a's and b's, but no c's and change the 1:2:3 ratio

case 3: it can be in b's; we only add b's not a's or c's, and change the 1:2:3 ratio

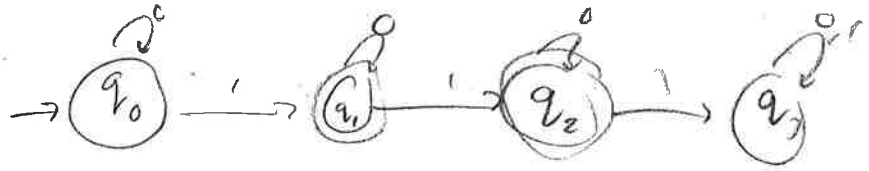
case 4: it can be in b's and c's; we only add b's and c's, but no a's, and change the 1:2:3 ratio

case 5: it can be in c's: we add c's but no a's, or b's, and change the 1:2:3 ratio

in all cases, $uv^2xy^2z \notin L$ but by pumping lemma, it should be.

We have a contradiction, therefore our assumption is false and L is not context free.

3 b - c



start, # $\rightarrow q_0, R$

$q_0, 0 \rightarrow q_0, R$

$q_0, 1 \rightarrow q_1, R$

$q_1, 0 \rightarrow q_1, R$

$q_1, 1 \rightarrow q_2, R$

$q_2, 0 \rightarrow q_2, R$

$q_2, 1 \rightarrow q_3, R$

$q_3, 0 \rightarrow q_3, R$

$q_3, 1 \rightarrow q_3, R$

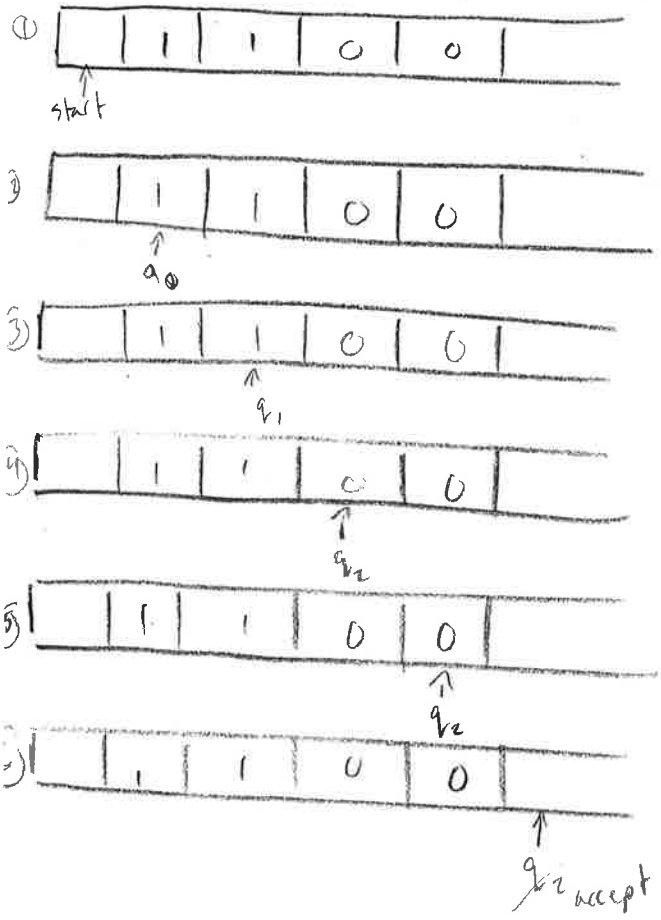
$q_0, \# \rightarrow$ reject

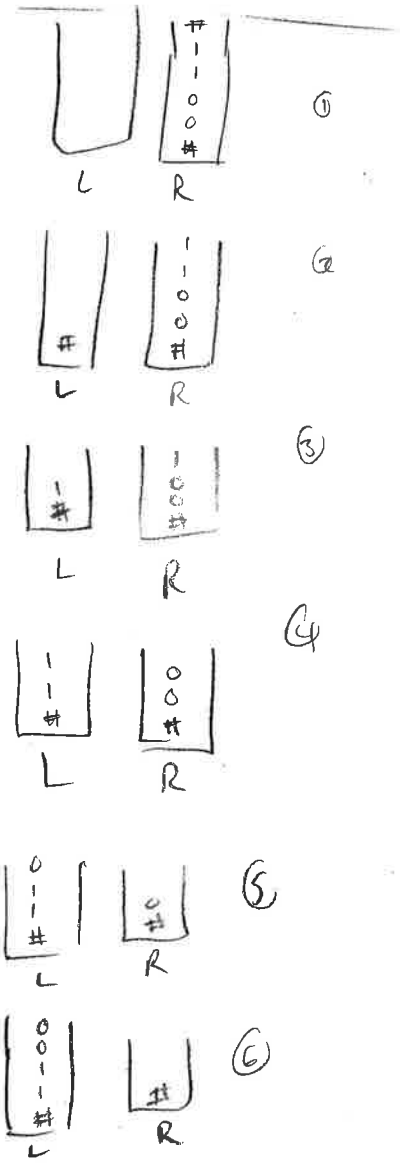
$q_1, \# \rightarrow$ accept

$q_2, \# \rightarrow$ accept

$q_3, \# \rightarrow$ reject

trace





30.

(start, #) → (in1, R)

(in1, 1) → R

(in1, #) → (in2, R)

(in2, 1) → R

(in2, #) → ready, 1

(ready, 1) → (#, L, wait)

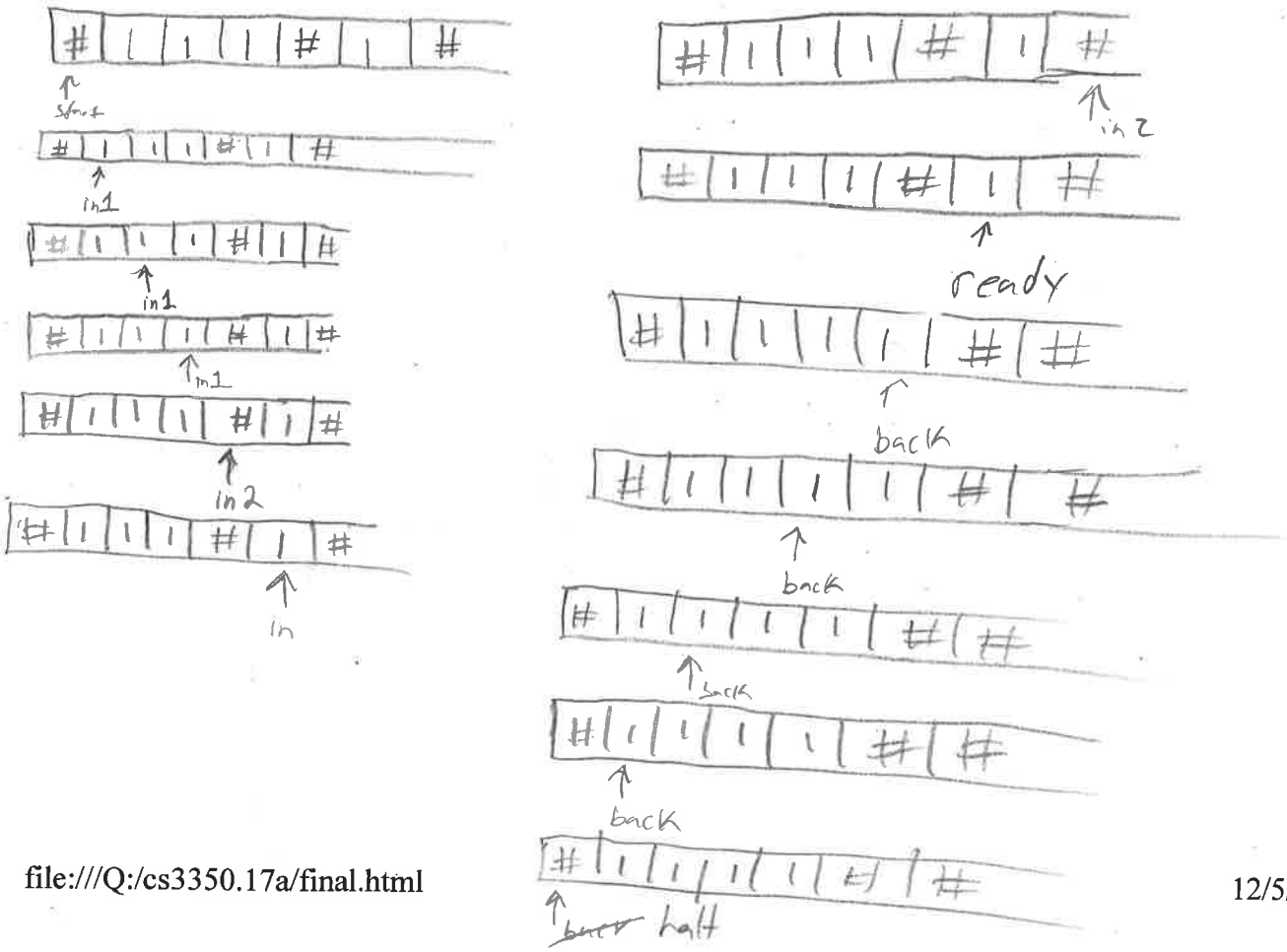
(wait, 1) → L

(wait, #) → 1, L, back

(back, 1) → L

(back, #) → halt.

(ready, #) → L, back



4. *Beyond Turing machines: computability*

4a. Formulate Church-Turing thesis. Is it a mathematical theorem? Is it a statement about the physical world? Anything that can be computed on any physical device can also be computed on a turing machine.

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

4c. Not all algorithms are feasible, but, unfortunately, we do not have a perfect definition is feasibility. Give a current formal definition of feasibility and give two examples:

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

4d. Briefly describe what is P, what is NP, what is NP-hard, and what is NP-complete. Is P equal to NP?

4e. Give an example of an NP=complete problem: what is given, and what we want to find.

4b) In a computer, a program p or datat d is a seqvence of 0s & 1's

$0100101\ldots_2$ diff strings

1_2
01_2 } same int
001_2

to avoid ambiguity, we can append 1 in front

$0001101\ldots0_2$

Proof by contradiction:

Let's assume there exists a halt-checker; the number corresponding to this program is P_0. Does P_0 halt on P_0?

halt-checker(p, d) = { true if P halts on d; false otherwise }

```
public static void proof(int n){
    if(halt-checker(n,n){
        while(true){
            n=n;
        }
    }}
```

Case 1) P_0 halts on P_0, then halts(P_0,P_0) is true,
So P_0 won't halt on P_0.

Case 2) P_0 doesn't halt on P_0, halt(P_0,P_0) is false,
So P_0 halts on P_0

So we have a contradiction and our assumption is false.

formal def.

feasibility ≡ poly time

4c.) feasibility:
There exists a polynomial $P(n)$ such that for every input x, the running time $t_A(x)$ of A on x is $\leq P(\text{len}(x))$

Practically but not theoretically

$2^{(10^{-10000} n)}$

Theoretically but not practically

$10^{10000} n$

4d.) P - class of problems solvable in polynomial time

NP - class of problems for which, once there is a candidate for a solution can be checked to verify whether it a solution in polynomial time

NP hard - a problem P_a is NP hard if every problem from the class NP can be reduced to P_a

NP complete - is NP hard and is in NP. A problem P_a from the class NP is NP complete if every class NP can be reduced to it.

P = NP is an open problem

4e) example of NP complete: propositional satisfiability given: a propositional formula

$(x_1 \& \& x_2) \,||\, (x_3 \ldots)$

Find: value of $x_1, x_2, \ldots, x_n$ that make it true