40/40 = (100/100)

CS 3350 Automata, Computability, and Formal Languages
Fall 2016, Final Exam

Name: [redacted]

General comments:

- you are allowed up to 10 pages of handwritten notes;
- if you need extra pages, place your name on each extra page;
- the main goal of most questions is to show that you know the corresponding algorithms; so, if you are running of time, just follow the few first steps of the corresponding algorithm;
- each question will be graded on its own merit; so, for example, if on one stage, you got a wrong context-free grammar, but on the second stage, you correctly apply the Chomsky normal form algorithm to the resulting grammar, you will get full credit for the second stage.

Good luck!

12/0

1. *Finite automata and regular languages:*

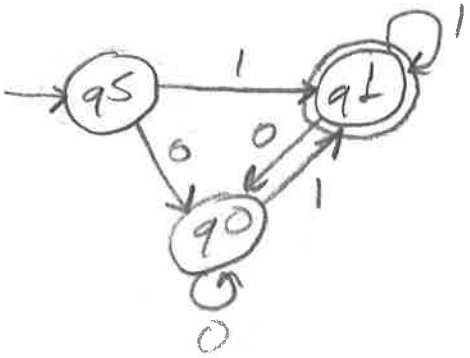
1a. Design a finite automaton for recognizing binary sequences that end with 1. Assume that the input strings contain only symbols 0 and 1. The easiest is to have 3 states: start, accept, and reject, you just need to describe transitions between these states. Show, step-by-step, how your automaton will accept the string 01101.

1b. On the example of this automaton, show how the word 01101 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.

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

1a)



0	1	1	0	1
q5	q0	q1	q1	q0	q1

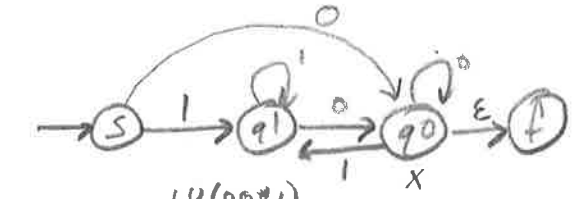
1b)

0	1	1	0	1
q5	q0	q1	q1	q0	q1

x y z

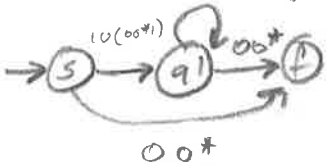
$x = 01 \quad y = 1 \quad z = 01$

1c)



$R_{ij}' = R_{ij} \cup (R_{ik} R_{kk}^* R_{kj})$

$1 \cup (00^*1)$



00^*

$R_{sf}' = R_{sf} \cup (R_{s0} R_{00}^* R_{0f})$

$00^* = \phi \cup (0\ 0^*\ \Lambda)$

$R_{s1}' = R_{s1} \cup (R_{s0} R_{00}^* R_{01})$

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

$R_{11}' = R_{11} \cup (R_{10} R_{00}^* R_{01})$

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

$R_{1f}' = R_{1f} \cup (R_{10} R_{00}^* R_{0f})$

$00^* = \phi \cup (0\ 0^*\ \Lambda)$

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

$= 00^* \cup (1 \cup (00^*1)(1 \cup (00^*1))^* 00^*)$

$00^* \cup (1 \cup (00^*1)(1 \cup (00^*1))^* 00^*)$

10/10

1. *Finite automata and regular languages:*

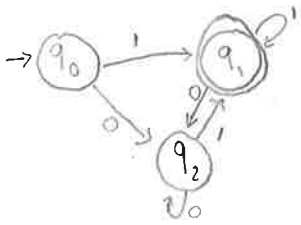
2/2 1a. Design a finite automaton for recognizing binary sequences that end with 1. Assume that the input strings contain only symbols 0 and 1. The easiest is to have 3 states: start, accept, and reject, you just need to describe transitions between these states. Show, step-by-step, how your automaton will accept the string 01101.

2/2 1b. On the example of this automaton, show how the word 01101 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.

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

1.a.



0 1 1 0 1 1
q_0 q_2 q_1 q_1 q_2 q_1 ✓

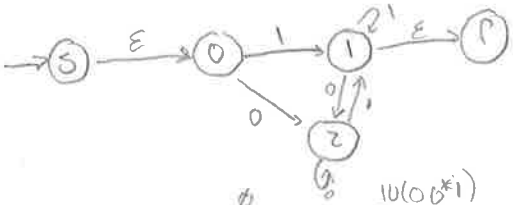
1.b.

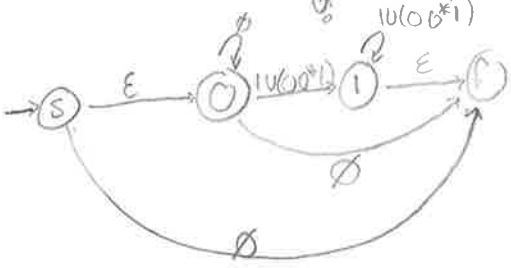
0 1 1 0 1 1
q_0 q_2 q_1 q_1 q_2 q_1
x y z

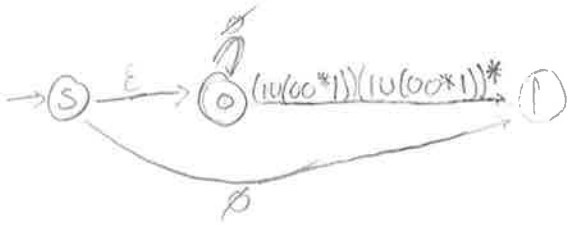
x = 0 1
y = 1
z = 0 1

x - before 1st repeating state
y - between 1st and 2nd rep
z - after 2nd rep

1.c







$R'_{s0} = R_{s0} \cup (R_{s2} R_{22}^* R_{20})$
$\Lambda \cup (\emptyset \ldots) = \Lambda$

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

$R'_{01} = R_{01} \cup (R_{02} R_{22}^* R_{21})$
$1 \cup (0\ 0^*\ 1)$

$R'_{0f} = R_{0f} \cup (R_{02} R_{22}^* R_{2f})$
$\emptyset \cup (0\ 0^*\ \emptyset) = \emptyset$

$R'_{1f} = R_{1f} \cup (R_{12} R_{22}^* R_{2f})$
$\Lambda \cup (0\ 0^*\ \emptyset) = \Lambda$

$R'_{00} = R_{00} \cup (R_{02} R_{22}^* R_{20})$
$\emptyset \cup (0\ 0^*\ \emptyset) = \emptyset$

$R'_{11} = R_{11} \cup (R_{12} R_{22}^* R_{21})$
$1 \cup (0\ 0^*\ 1)$

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

$R'_{s0} = R_{s0} \cup (R_{s1} R_{11}^* R_{10})$
$\Lambda \cup (\emptyset \ldots) = \Lambda$

$R'_{00} = R_{00} \cup (R_{01} R_{11}^* R_{10})$
$\emptyset \cup ((1 \cup (00^*1))(1 \cup (00^*1))^* \emptyset) = \emptyset$

$R'_{0f} = R_{0f} \cup (R_{01} R_{11}^* R_{1f})$
$\emptyset \cup ((1 \cup (00^*1))(1 \cup (00^*1))^* \Lambda) =$
$(1 \cup (00^*1))(1 \cup (00^*1))^*$

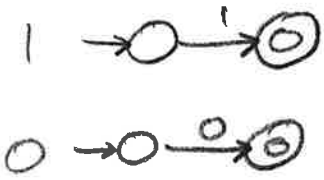
$R_{sf} = R_{sf} \cup (R_{s0} R_{00}^* R_{0f})$
$\emptyset \cup (\Lambda\ \emptyset^*\ (1 \cup (00^*1))(1 \cup (00^*1))^*)$

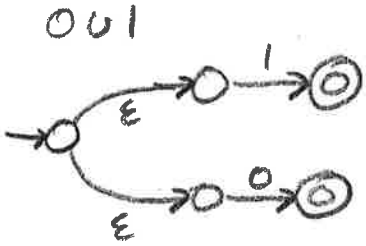
$= (1 \cup (00^*1))(1 \cup (00^*1))^*$

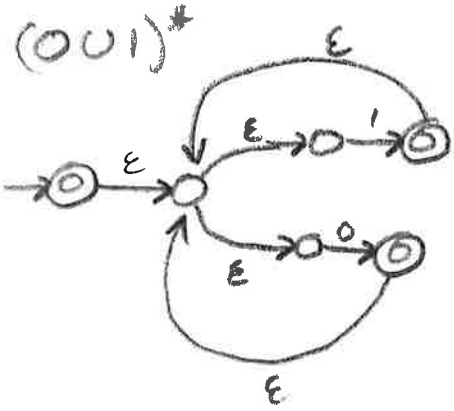
$\rightarrow S\ \underline{(1 \cup (00^*1))(1 \cup (00^*1))^*}\ (F)$

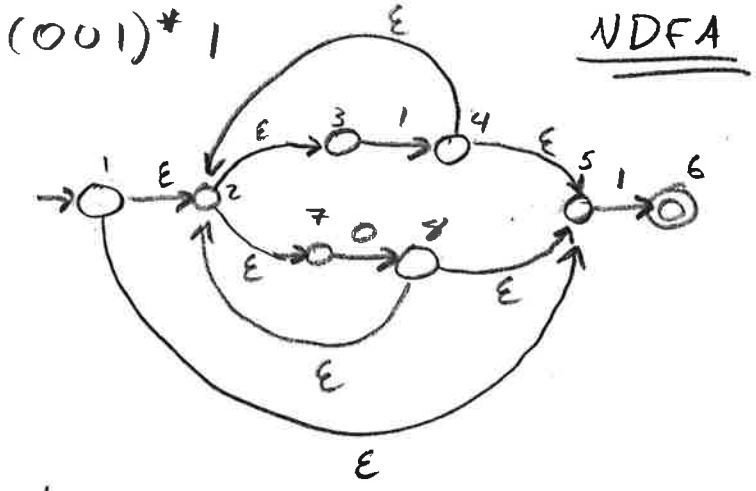
1d-e)

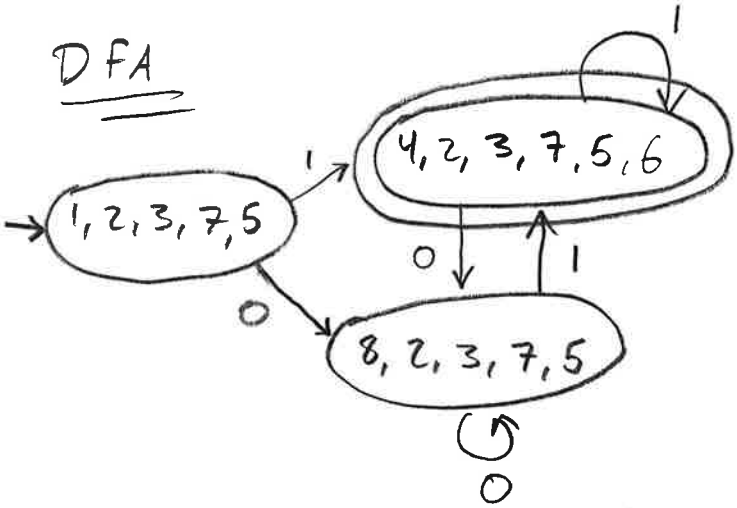
$(0 \cup 1)^* 1$


1
0


0 U 1
1
ε
0
ε


(0 U 1)*
ε
ε
1
ε
0
ε
ε


(0 U 1)* 1
NDFA
ε
ε
3
1
4
ε
5
1
6
1
ε
2
7
0
8
ε
ε
ε
ε


DFA
1
1
4, 2, 3, 7, 5, 6
1, 2, 3, 7, 5
0
1
0
8, 2, 3, 7, 5
0

10/10

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

2a. Prove that the language consisting of all expressions that contain half as many a's than 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 01101.

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.

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 01101.

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

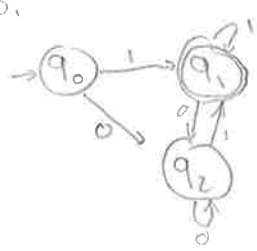
- how the compiler will transform the expression 2 − 3 * (1 + 2) into inverse Polish notation, and
- how it will compute the value of this expression.

2a. $L = \{ a^n b^{2n} , n = 0, 1, \ldots \}$ is not regular

Proof by contradiction: Let us asume that L is regular, then by pumping lemma $\exists p$ such that every word s from L where length is $\geq p$ can be represented as $s = xyz$, where $len(y) \geq 0$, $len(xy) \leq p$ and for every i, $xy^i z \in L$

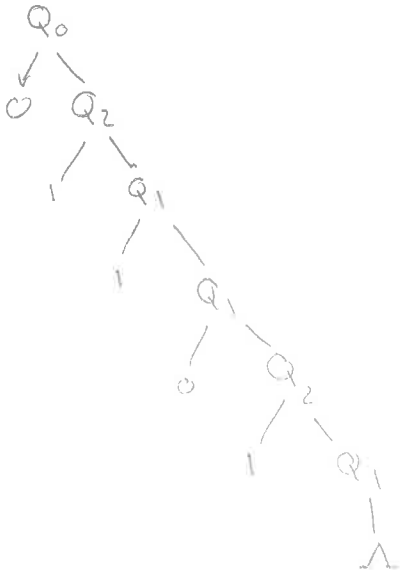
Lets take $s = a^p b^{2p}$. Here the $len(s) = 3p \geq p$ so by pumping lemma there exist corresponding x, y, z. Since s starts with xy and $len(xy) \leq p$ this means that x and y are in as. So when we take $xy^i z$ we will get but the number of bs does not change, so the number of as is no longer half of the number of bs, so $xyyz \notin L$ but according to pumping lemma $xyyz \in L$, a contradiction. This contradiction shows that our asumption is wrong so L is not regular.

2.b.

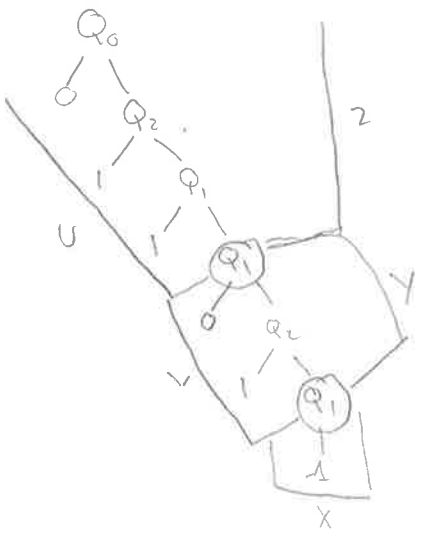


$Q_0 \to 1Q_1$
$Q_0 \to 0Q_2$
$Q_1 \to 1Q_1$
$Q_1 \to 0Q_2$
$Q_2 \to 0Q_2$
$Q_2 \to 1Q_1$
$Q_1 \to \Lambda$

0 1 1 0 1



2.c.



U = 0 1 1
V = 0 1
X = Λ

2.d.

$Q_0 \to 1Q_1$
$Q_0 \to 0Q_2$
$Q_1 \to 1Q_1$
$Q_1 \to 0Q_2$
$Q_2 \to 1Q_1$
$Q_2 \to 0Q_2$
~~$Q_1 \to \varepsilon$~~

Pre-Step
~~$S_0 \to Q_0$~~

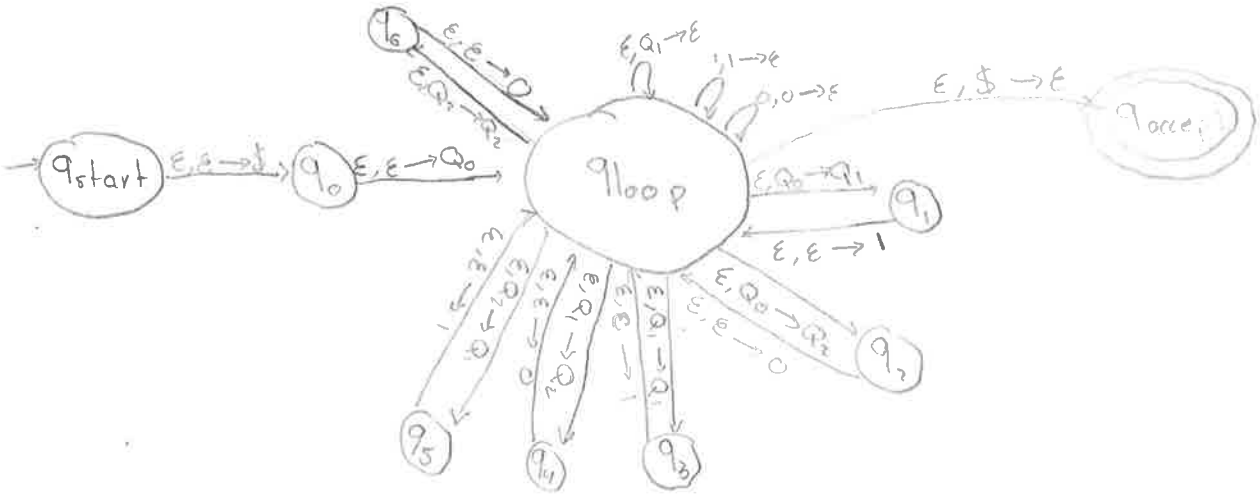
Step 0
$Q_0 \to 1$
$Q_1 \to 1$
$Q_2 \to 1$

Step 1
$S_0 \to 1Q_1$ $S_0 \to 1$
$S_0 \to 0Q_2$

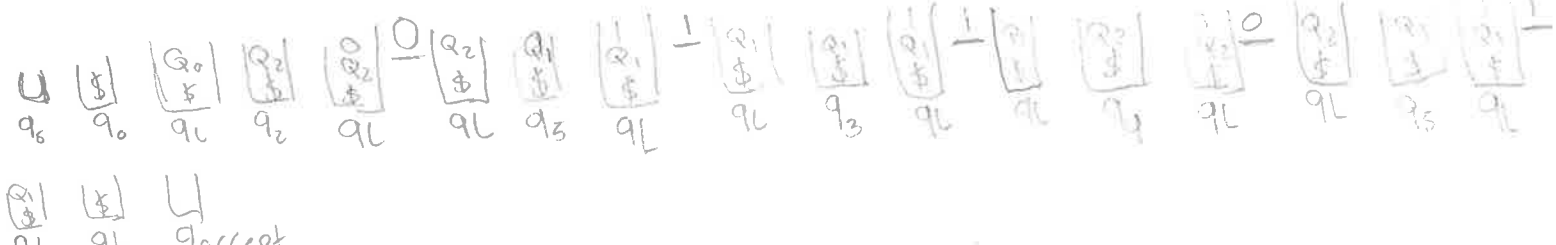
Step 2
$V_0 \to 0$ $V_1 \to 1$
$Q_0 \to V_1Q_1$
$Q_0 \to V_0Q_2$
$Q_1 \to V_1Q_1$
$Q_1 \to V_0Q_2$
$Q_2 \to V_1Q_1$
$Q_2 \to V_0Q_2$
$Q_0 \to 1$
$Q_1 \to 1$

$Q_2 \to 1$
$S_0 \to V_1Q_1$
$S_0 \to V_0Q_2$
$S_0 \to 1$

2.e

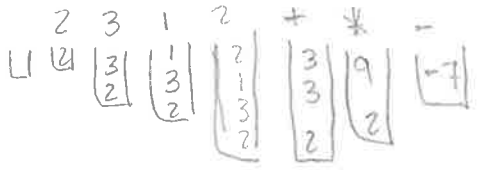


0 1 0 1



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

2312+*-



10/10

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

2a. Prove that the language consisting of all expressions that contain half as many a's than 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 01101.

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.

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 01101.

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

- 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) $L = \{ a^n b^{2n} : n = 0, 1, 2, \ldots \}$ is not regular

Proof: By contradiction. Let us assume that L is regular. Then, by pumping lemma $\exists p$ such that every words from L whose length $\geq p$ can be represented as $s = xyz$, where $len(y) > 0$, $len(xy) \leq p$, and for every i, $xy^i z \in L$.

Let's take $s = a^p b^{2p}$. Here, $len(s) = 3p \geq p$ so by pumping lemma, there exist corresponding x, y, and z.

Since s starts with xy and $len(xy) \leq p$, this means that x and y are in a's. So when we take $xyyz$, we add a's but the number of b's does not change. So # of a's in $xyyz$ is no longer half 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. the proposition is proven.

2b)

01101

$Q_s \rightarrow 1Q_1$

$Q_s \rightarrow 0Q_0$

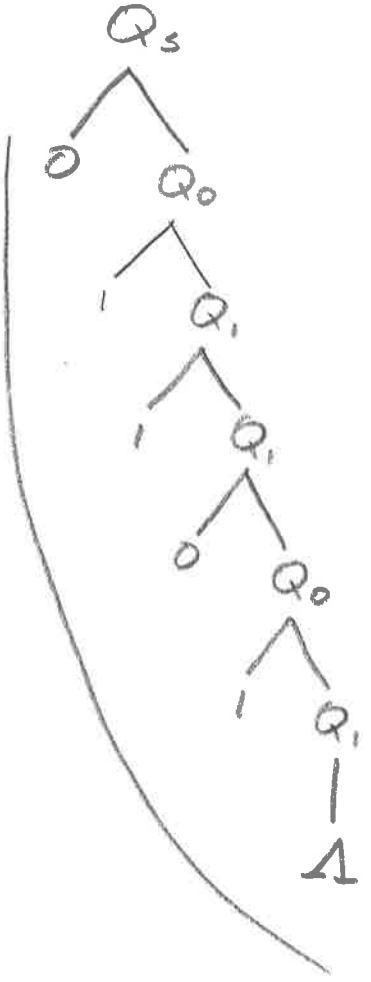
$Q_1 \rightarrow 1Q_1$

$Q_1 \rightarrow 0Q_0$

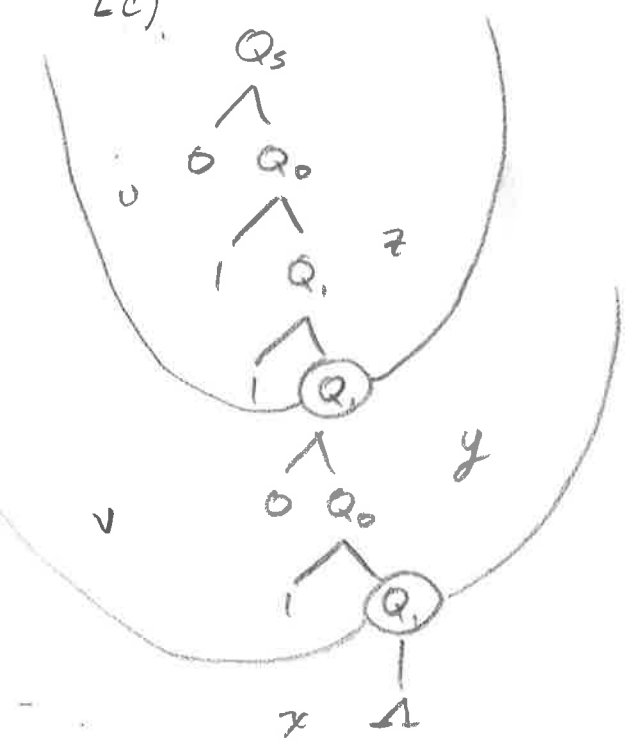
$Q_0 \rightarrow 1Q_1$

$Q_0 \rightarrow 0Q_0$

$Q_1 \rightarrow \Lambda$



2c)



$u = 01$

$v = 01$

$x = \Lambda$

$y = \varepsilon$

$z = \varepsilon$

2d)

$Q_S \to 1\,Q_1$ ✓

$Q_S \to 0\,Q_0$ ✓

$Q_1 \to 1\,Q_1$ ✓

$Q_1 \to 0\,Q_0$ ✓

$Q_0 \to 1\,Q_1$ ✓

$Q_0 \to 0\,Q_0$ ✓

~~$Q_1 \to \varepsilon$~~

preliminary step

~~$S_0 \Rightarrow Q_S$~~

step 0

$Q_S \to 1$ ✓

$Q_1 \to 1$ ✓

$Q_0 \to 1$ ✓

step 1

$S_0 \to 1\,Q_1$ ✓

$S_0 \to 0\,Q_0$ ✓

step 2

$V_1 \to 1$

$V_0 \to 0$

$Q_S \to V_1\,Q_1$

$Q_S \to V_0\,Q_0$

$Q_1 \to V_1\,Q_1$

$Q_1 \to V_0\,Q_0$

$Q_0 \to V_1\,Q_1$

$Q_0 \to V_0\,Q_0$

$Q_S \to 1$

$Q_1 \to 1$

$Q_0 \to 1$

$S_0 \to V_1\,Q_1$

$S_0 \to V_0\,Q_1$

2e)

$Q_S \to 1\,Q_1$

$Q_S \to 0\,Q_0$

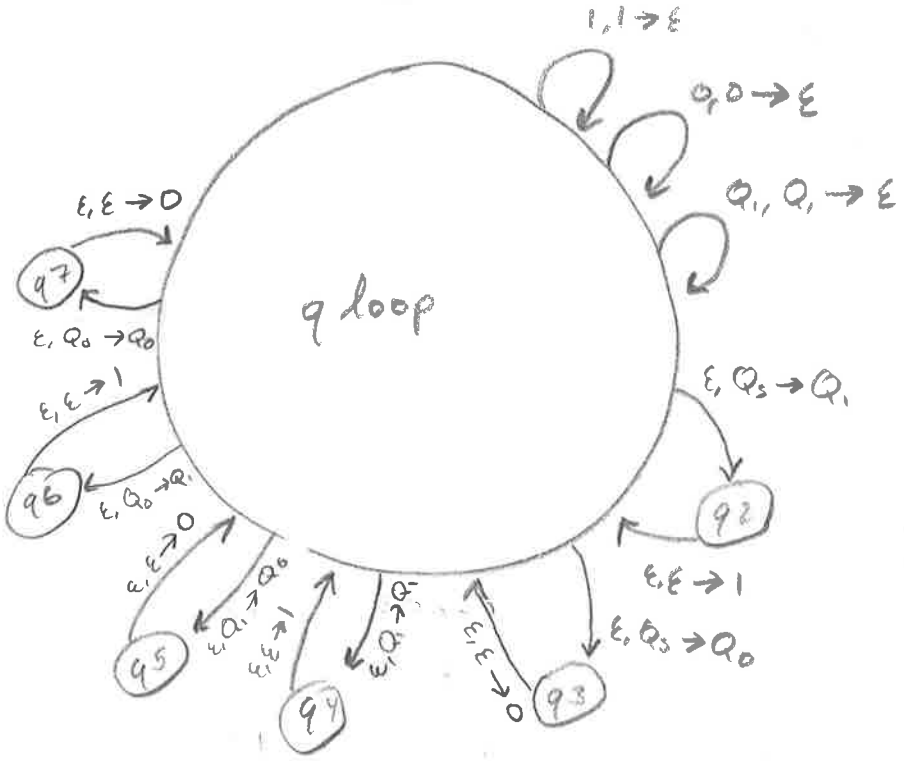
$Q_1 \to 1\,Q_1$

$Q_1 \to 0\,Q_0$

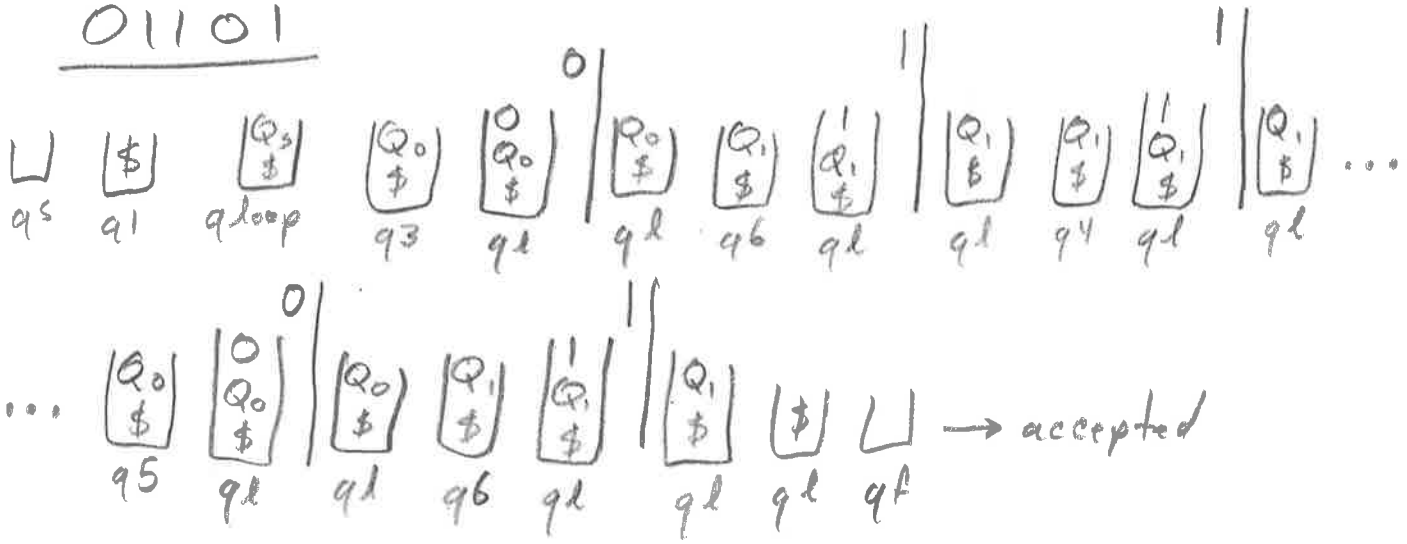
$Q_0 \to 1\,Q_1$

$Q_0 \to 0\,Q_0$

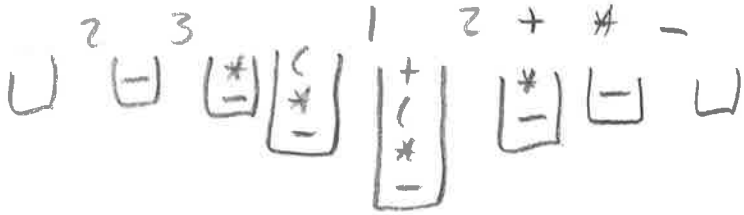
$Q_1 \to \Lambda$

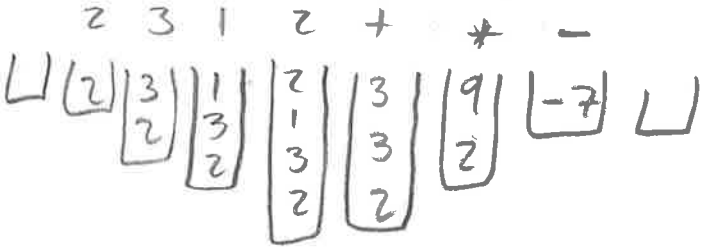


01101



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





10/10

3. *Beyond pushdown automata: Turing machines*

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 01101. 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 4 * n in binary code. Trace it for the binary number n = 10_2 (which is 2 decimal); the result of the computation should $4 * 2 = 8_{10}$, i.e., 1000_2

3.a. $L = \{a^n b^n c^n, n=0,1,2\ldots\}$ is not CFG

Proof by contradiction

Lets assume that L is CFG. Then according to pumping lemma for CFG there exist a p such that every word from L whose length at least p can be represented as uvxyz, where len(vy)>0, len(vxy)≤p, and $uv^i xy^i z \in L$ for all i.

Lets take $s = a^p b^p c^p \in L$. Here len(s)=3p≥p so by pumping lemma s can be represented as uvxyz. We know that (vxy)≤p. Lets consider all possible locations of vxy within s, and lets show that in all possible cases we get a contradiction. Here, s = a...a b...b c...c (p, p, p) where can vxy be?

It can be in a's, b's, c's, it can be in a's and b's, it can be in b's and c's. It cannot be in a's and c's because it would include all b's and len(vxy)>p.

Lets consider all possible cases one by one

1) if vxy is in a's, then we go to uv^0xy^0z, we still have to the original word so we have more a's than b's and c's thus $uv^0xy^0z \notin L$, but by pumping lemma it should be in L, a contradiction

2) similar to 1) vxy cannot be in b's

3) similar to 1) vxy cannot be in c's

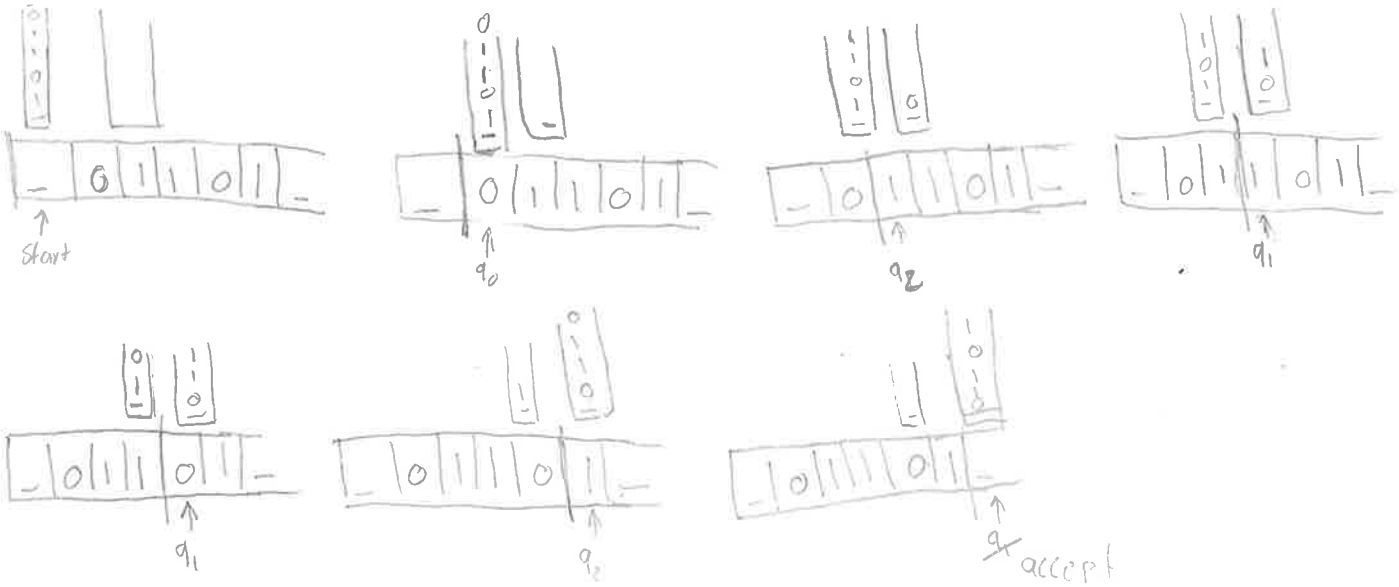
4) if vxy is in a's and b's then pumping added a's and b's but not c's so in uv^2xy^2z there are fewer c's than a's and b's. So $uv^2xy^2z \notin L$, a contradiction

5) similar to 4) vxy cannot be in b's and c's

In all possible cases we get a contradiction so our assumption is wrong, so L is not CFG

Start, _ → q_0, R
Start, 0 → reject
Start 1 → reject
q_0, 1 → q_1, R
q_0, 0 → q_2, R
q_0, _ → reject
q_1, 1 → q_1, R
q_1, 0 → q_2, R
q_1, _ → accept
q_2, 1 → q_1, R
q_2, 0 → q_2, R
q_2, _ → reject

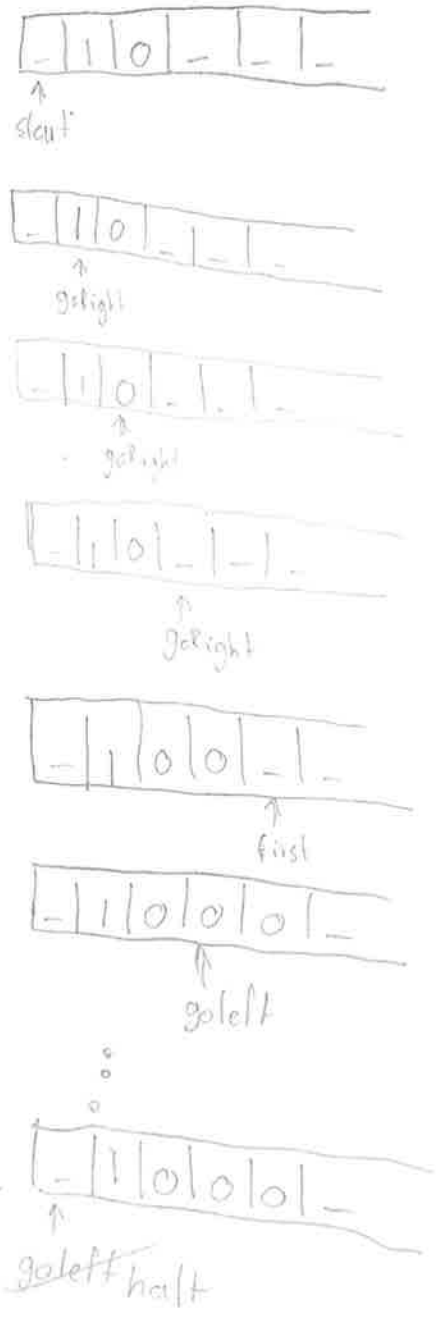
01101



3.d

4 * n binary code

start, _ → goright, R
goright, 0 → goright, R
goright, 1 → goright, R
goright, _ → 0, first, R
first, _ → 0, goleft, L
goleft, 0 → goleft, L
goleft, 1 → goleft, L
goleft, _ → halt



+1 11/10

3. *Beyond pushdown automata: Turing machines*

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 01101. 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 4 * n in binary code. Trace it for the binary number n = 10_2 (which is 2 decimal); the result of the computation should $4 * 2 = 8_{10}$, i.e., 1000_2

3a) $L = \{a^n b^n c^n d^n : n = 0, 1, 2, \ldots\}$ is not a CFG

Proof: By contradiction. Let's assume that L is CFG, then according to pumping lemma for CFGs, $\exists p$ such that every word s from L whose length is at least p can be represented as $s = uvxyz$, where $len(vy) > 0$, $len(vxy) \leq p$ and $uv^i xy^i z \in L$ for all i.

Let's take $s = a^p b^p c^p d^p \in L$. Here $len(s) = 4p \geq p$. So by pumping lemma s can be represented as $uvxyz$. We know that $len(vxy) \leq p$. Let's consider all possible locations of vxy within s, and let's show that in all possible cases we get a contradiction. Here, $s = \underbrace{a\ldots a}_{p \text{ times}} \underbrace{b\ldots b}_{p \text{ times}} \underbrace{c\ldots c}_{p \text{ times}} \underbrace{d\ldots d}_{p \text{ times}}$

where can vxy be? It can be in a's, b's, c's, d's, a's and b's, b's and c's, c's and d's. It cannot be in a's and c's, neither b's and d's because then it would include all between those and $len(vxy) > p$.

Let consider all possible cases one by one.

1) If vxy is in a's, then when we go to $uvvxyyz$, we add a's to the original word, now having more a's than b's, c's, and d's, thus $uv^2xy^2z \notin L$ — but by pumping lemma it should. so in this case, we get a contradiction.

2) Similarly, if vxy is in b's — we get a contradiction.

3) Similarly, vxy cannot be in c's.

4) Similarly, vxy cannot be in d's.

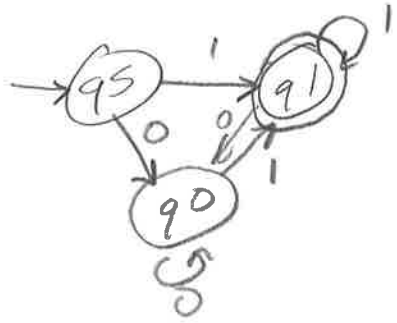
5) If vxy is in a's and b's, then pumping adds a's and/or b's but not c's and d's, so in uv^2xy^2z, there are fewer c's and d's than a's and b's so $uv^2xy^2z \notin L$ — a contradiction.

6) Similarly, vxy cannot be in b's and c's.

7) Similarly, vxy cannot be in c's and d's.

In all possible cases we get a contradiction, so our assumption is wrong, so L is not a CFG.

3b-c)



start, _ → begin, R

begin, 1 → stateOne, R

begin, 0 → stateZero, R

stateOne, 1 → stateOne, R

stateOne, 0 → stateZero, R

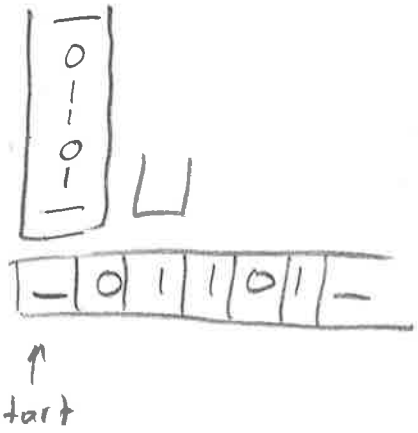
stateZero, 1 → stateOne, R

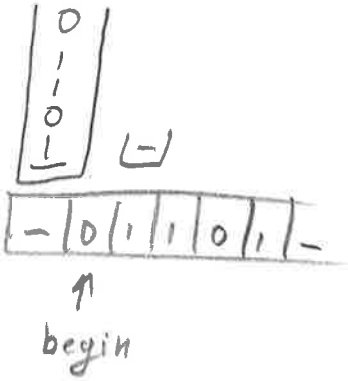
stateZero, 0 → stateZero, R

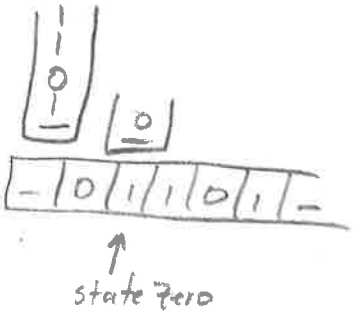
stateOne, _ → accept

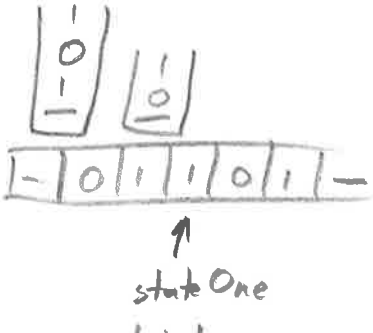
stateZero, _ → reject

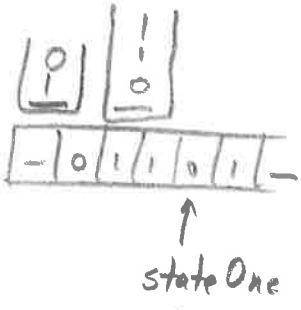
01101

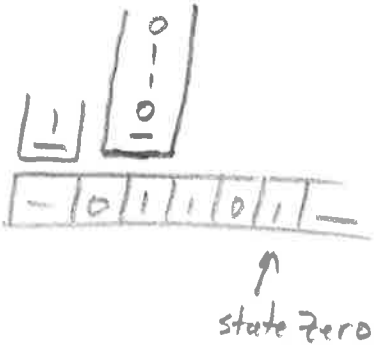


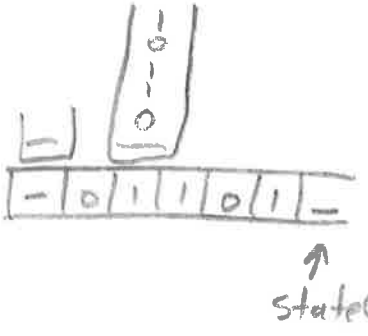


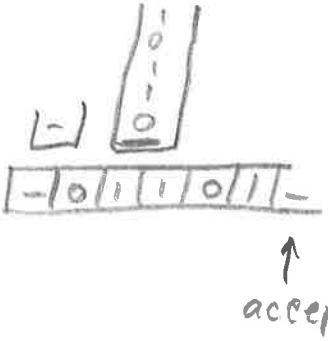






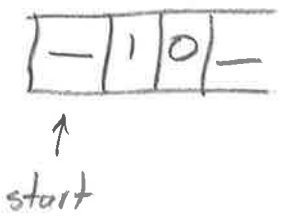


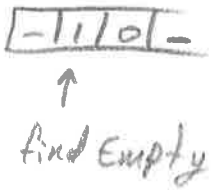


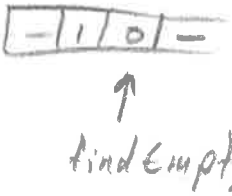


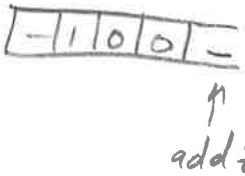
3d)

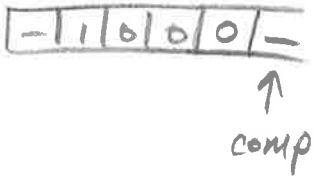
start, _ → findEmpty, R
findEmpty, 1 → findEmpty, R
findEmpty, 0 → findEmpty, R
findEmpty, _ → 0, addZero, R
addZero, _ → 0, complete, R
complete, _ → goBack, L
goBack, 0 → goBack, L
goBack, 1 → goBack, L
goBack, _ → halt.

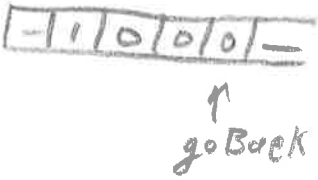


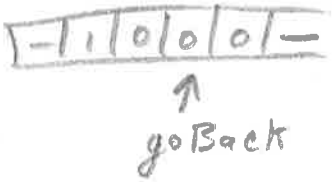


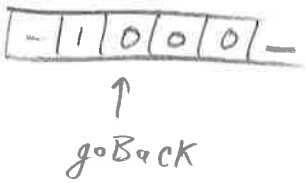


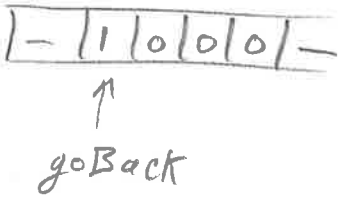


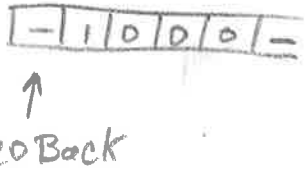


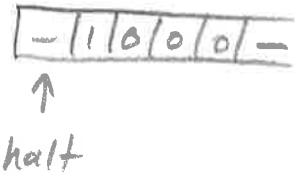












10.5/10

4. *Beyond Turing machines: computability*

9.5/2

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

2/2

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 j (n) which was defined as follows:

2/2

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

2/2

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.

2/2

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

4.a. Church-Turing thesis: Anything that can be computed on any physical device can also be computed on a turing machine
Its a statement about the physical world, not a mathematical proof

4.b Halting problem: There is no algorithm that given a program $\underline{p}$ and data $\underline{d}$, checks whether p halts on d

Proof by contradiction. Let's assume that there exist a halt checker(p,d) = true if p halts on d, false if p does not halt on d. Design a new auxiliary program

$$f(n) = \begin{cases} P_n(n)+1 & \text{if } n \text{ is a valid code of the program and halts}(P_n, n) = \text{true} \\ 0 & \text{otherwise} \end{cases}$$

this f is computable, let n_0 be the natural number corresponding to this program

$f(n_0) = P_{n_0}(n_0)+1$ by definition of $f(n)$, on the other hand $P_{n_0}(n) = f(n)$

$P_{n_0}(n) = f(n_0)$ in particular for $n = n_0$ we get $P_{n_0}(n_0) = f(n_0)$

So $P_{n_0}(n_0) = P_{n_0}(n_0)+1 \Rightarrow 0 = 1$ we get a contradiction

4.c

	0	1	2
j_0	(0)	-	4
j_1	-	(1)	-
j_2	4	-	(6)
$f(n)$	1	0	7

$j_0(n) = 2 * n$

$j_1(n) = n + 4$

4.d

$F_A(n) = 10^{30^5} n$, feasible by definition, not feasible in practice

$F_B(n) = e^{10^{-10^1} n}$, not feasible by definition, feasible in practice

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

NP- are problems that can be verified in polynomial time, that is if a problem can be solved and has a proof, we can check that it is correct in polynomial time

NP-Hard- a problem X is NP-Hard if there is an NP complete problem y such that y is reducible to x in polynomial time, but since NP-complete problems can be reduced to any other NP-complete problem in polynomial time, all NP-complete problems can be reduced to any NP-Hard problem in polynomial time. Then if there is a solution to one NP-Hard problem in polynomial time, all NP problems can be solved in polynomial time