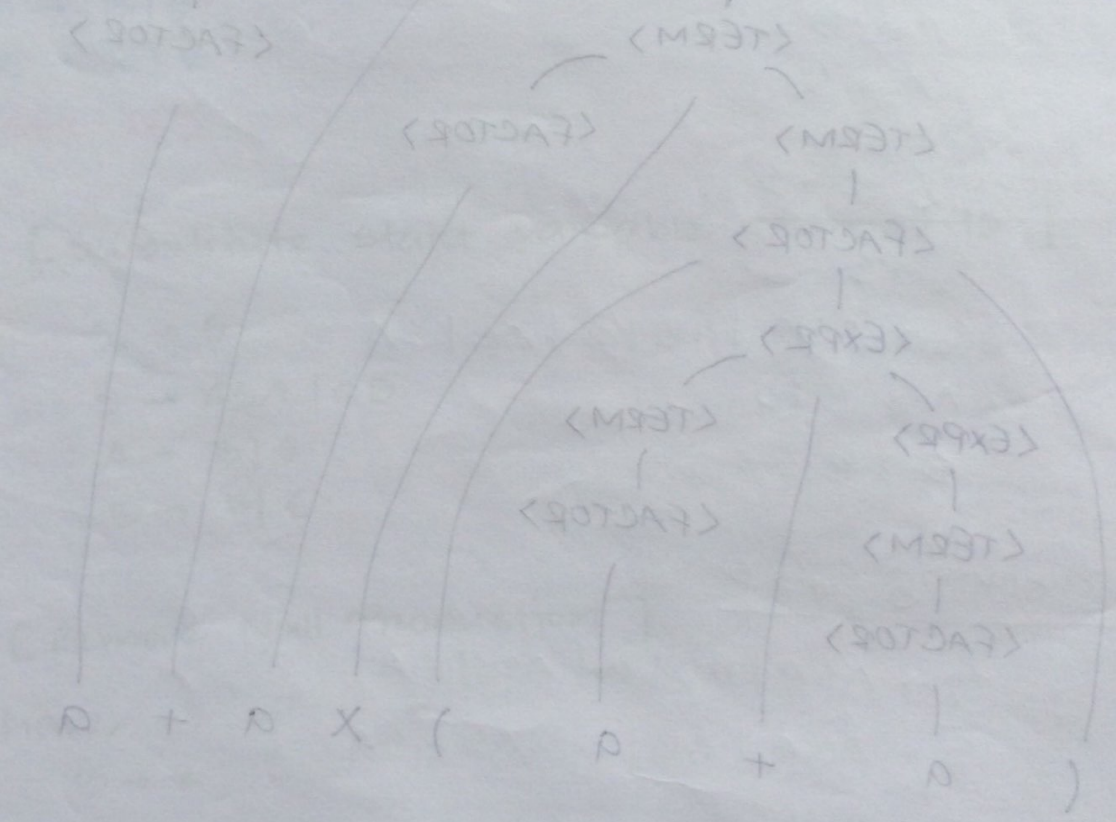
Context Free Grammar (CFG)

A CFG is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called variables,
2. Σ is a finite set, disjoint from V, called the terminals,
3. R is a finite set of rules, with each rules being a variable and a string of variables and terminals,
4. $S \in V$ is the start variable.



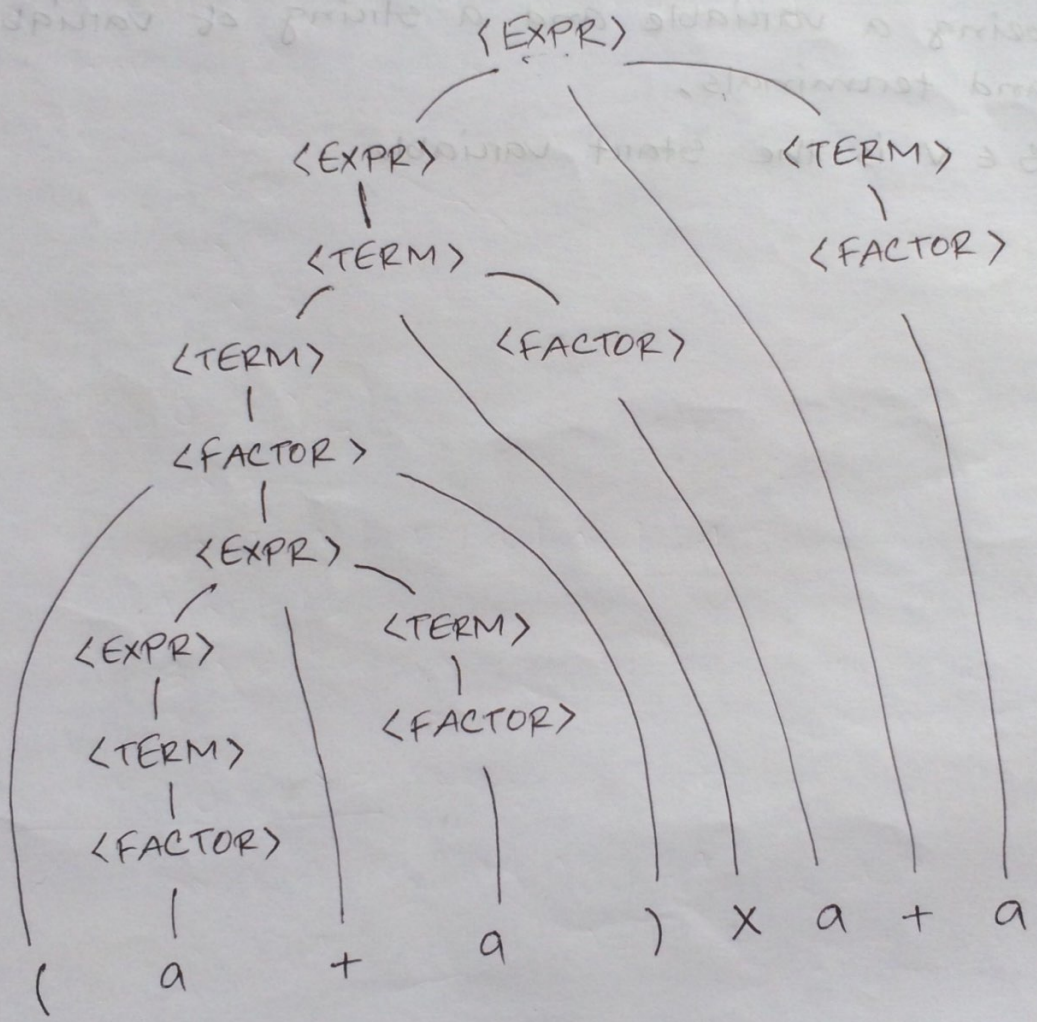
Consider the CFG :

⟨EXPR⟩ → ⟨EXPR⟩ + ⟨TERM⟩ | ⟨TERM⟩

⟨TERM⟩ → ⟨TERM⟩ × ⟨FACTOR⟩ | ⟨FACTOR⟩

⟨FACTOR⟩ → (⟨EXPR⟩) | a

Draw parse tree for the string (a + a) × a + a with this grammar. [SPTL - 18]



Nayeem Mahmood / 6AM / C161026

Chomsky Normal Form (CNF)

A CFG is in CNF if every rule is of the form

$$A \to BC$$
$$A \to a$$

where a is any terminal and A, B and C are any variables – except that B and C may not be the start variable. In addition, the rule $S \to \varepsilon$ is permitted where S is the start variable.

Convert the given CFG to CNF :

$S \to ASA \mid aB$
$A \to B \mid S$
$B \to b \mid \varepsilon$

[Sept-18]

Step 1: [Substitute start variable from RHS]

$S_0 \to S$
$S \to ASA \mid aB$
$A \to B \mid S$
$B \to b \mid \varepsilon$

Step 2: [Remove Null Production]

We have,

$B \to \varepsilon$

and $A \to B$
$\to \varepsilon$

Removing $B \to \varepsilon$

$S_0 \to S$
$S \to ASA \mid aB \mid a$
$A \to B \mid S \mid \varepsilon$

$B \to b$

Removing $A \to \epsilon$

$S_0 \to S$

$S \to ASA \mid aB \mid a \mid AS \mid SA \mid S$

$A \to B \mid S$

$B \to b$

Step 3: [Removing Unit Production]

We have,

$S_0 \to S$

$S \to S$

$A \to B$

$A \to S$

Removing $S \to S$

$S_0 \to S$

$S \to ASA \mid aB \mid a \mid AS \mid SA$

$A \to B \mid S$

$B \to b$

Removing $S_0 \to S$

$S_0 \to ASA \mid aB \mid a \mid AS \mid SA$

$S \to ASA \mid aB \mid a \mid AS \mid SA$

$A \to B \mid S$

$B \to b$

Nayeem Mahmood / 6AM / C161026

Removing $A \rightarrow B$

$S_0 \rightarrow ASA \mid aB \mid a \mid AS \mid SA$

$S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$

$A \rightarrow b \mid S$

$B \rightarrow b$

Removing $A \rightarrow S$

$S_0 \rightarrow ASA \mid aB \mid a \mid AS \mid SA$

$S \rightarrow ASA \mid aB \mid a \mid AS \mid SA$

$A \rightarrow b \mid ASA \mid aB \mid a \mid AS \mid SA$

$B \rightarrow b$

Step 4: [Finding out the Production rules that have more than 2 variables in RHS]

We have,

$S_0 \rightarrow ASA$

$S \rightarrow ASA$

$A \rightarrow ASA$

Let, $X \rightarrow SA$, So,

$S_0 \rightarrow AX \mid aB \mid a \mid AS \mid SA$

$S \rightarrow AX \mid aB \mid a \mid AS \mid SA$

$A \rightarrow b \mid AX \mid aB \mid a \mid AS \mid SA$

$B \rightarrow b$

$X \rightarrow SA$

Step 5: Changing the rules:

$S_0 \rightarrow aB$

$S \rightarrow aB$

$A \rightarrow aB$

Finally, we get:

$S_0 \rightarrow AX \mid YB \mid a \mid AS \mid SA$

$S \rightarrow AX \mid YB \mid a \mid AS \mid SA$

$A \rightarrow b \mid AX \mid YB \mid a \mid AS \mid SA$

$B \rightarrow b$

$X \rightarrow SA$

$Y \rightarrow a$

which is the required CNF.

Pumping Lemma for Context Free Language

If A is a CFL, then there is a number p (the pumping length) where, if s is any string in A of at least p, then s may be divided into 5 pieces $s = uvxyz$ satisfying the conditions:

1. for each $i \geq 0$, $uv^ixy^iz \in A$,
2. $|vy| > 0$, and
3. $|vxy| \leq p$

When s is being divided into uvxyz, condition 2 says that either v or y is not empty string. Otherwise the theorem would be trivially true. Condition 3 states that the pieces v, x and y

together have length at most p.

⊞ Using the pumping lemma, show that the following languages are not context free:

i) $A = \{a^n b^n c^n \mid n \geq 0\}$ [Spr-18]

Let us assume that A be a CFL.

Now, let us consider the pumping length, $p = 4$ and $s = a^4 b^4 c^4 \in A$.

Case 1: v and y each contain only one type of symbol.

$s = \underbrace{a}_{u}\underbrace{aaa}_{v}\underbrace{abbbbcc}_{x}\underbrace{c}_{y}\underbrace{cc}_{z}$

For $i = 2$,

$uv^2xy^2z = aaaaaabbbbccccc$

$= a^6 b^4 c^5 \notin A$

Thus, condition 1 is violated.

Case 2: Either v or y has more than one kind of symbols.

$s = \underbrace{aa}_{u}\underbrace{aabb}_{v}\underbrace{bb}_{x}\underbrace{cc}_{y}\underbrace{cc}_{z}$

For $i = 2$,

$uv^2xy^2z = aaaabbaabbbbccccc \notin A$

because it violates the pattern $a^n b^n c^n$ to be followed.

Hence, the given language is not context free.

ii) $B = \{ a^i b^j c^k \mid 0 \leq i \leq j \leq k \}$

Let us assume that B be a CFL.

Now let us consider the pumping length $p = 4$.

So, $s = a^4 b^4 c^4 \in B$.

case 1: v and y each contain only one type of symbol.

$s = \underbrace{aaaa}_{u}\underbrace{a}_{v}\underbrace{bbb}_{x}\underbrace{b}_{y}\underbrace{cccc}_{z}$

For $l=2$, $uv^l xy^l z = aaaaaabbbbbccccc$

$= a^6 b^5 c^4$

Since number of a's is greater than that of b's and number of b's is greater than that of c's, $a^6 b^5 c^4 \notin B$.

case 2: Either v or y has more than one kind of symbols.

$s = \underbrace{aaa}_{u}\underbrace{a}_{v}\underbrace{b}_{x}\underbrace{bbb}_{y}\underbrace{cccc}_{z}$

For $l=2$, $uv^l xy^l z = aaaaabababbbbcccc$

which doesn't contain the symbols in the correct order.

Thus, we have shown that s can't be pumped in violation of the pumping lemma and that B is not context free.

iii) $L = \{ww \mid w \in \{0,1\}^*\}$

Let us assume that L is a CFL.

Now let's consider the pumping length $p = 5$, and so

$s = 0^5 1^5 0^5 1^5 \in L$

Case 1: vxy doesn't straddle a boundary.

$s = \underbrace{00000}_{u} \underbrace{1\,1\,1\,1}_{v\,x\,y} \underbrace{1\,00000\,1\,1\,1\,1}_{z}$

For $i = 2$,

$uv^2xy^2z = 000001111111100000 11111$

$= 0^5 1^7 0^5 1^5 \notin L$

Case 2a: vxy straddle the first boundary.

$s = \underbrace{00000}_{u} \underbrace{1\,1\,1}_{v\,x} \underbrace{1\,1}_{y} \underbrace{00000\,1\,1\,1\,1}_{z}$

For $i = 2$,

$uv^2xy^2z = 000000011111111 00000 11111$

$= 0^7 1^7 0^5 1^5 \notin L$

Case 2b: vxy straddles the third boundary.

$s = \underbrace{00000\,11111\,00000}_{u}\;\underbrace{0}_{v}\;\underbrace{1}_{x}\;\underbrace{1}_{y}\;\underbrace{11}_{z}$

For $i = 2$,

$uv^2xy^2z = 00000\,11111\,000\,00000\,1\,1\,1\,11$

$= 0^5 1^5 0^7 1^7 \notin L$

Case 3: vxy straddles the midpoint.

$s = \underbrace{00000\,11}_{u}\;\underbrace{1}_{v}\;\underbrace{1}_{x}\;\underbrace{00}_{y}\;\underbrace{000\,1111}_{z}$

For $i = 2$,

$uv^2xy^2z = 00000\,11\,1\,1\,1\,00000\,000\,1\,1\,1\,1$

$= 0^5 1^7 0^7 1^5 \notin L$

Hence, the given language L is not context free.

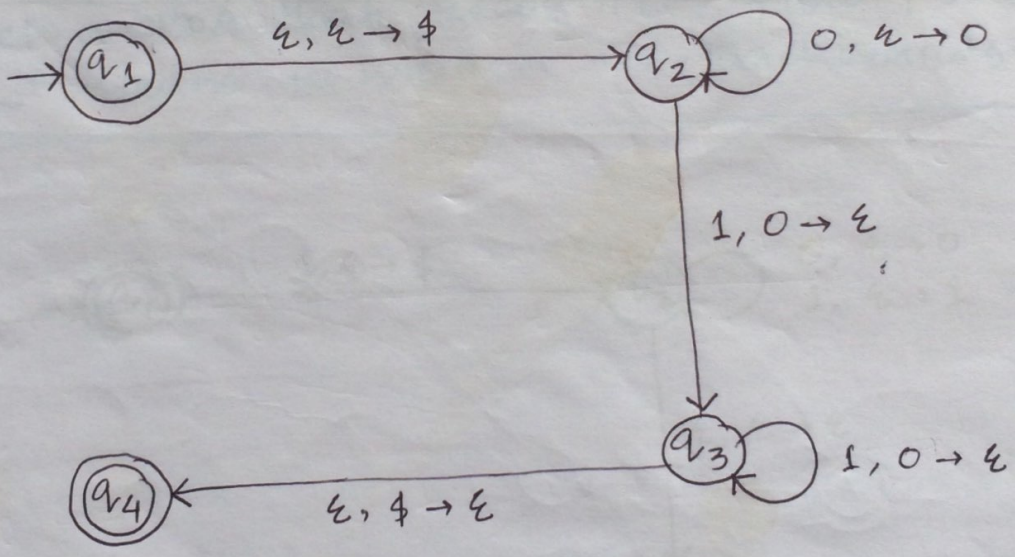
Nayeem Mahmood / 6AM / C161026

Formal Def^n of Pushdown Automata

A PDA is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where Q, Σ, Γ and F are all finite sets, and

1. Q is the set of states,
2. Σ is the set of input alphabets,
3. Γ is the stack alphabet,
4. $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow P(Q \times \Gamma_\varepsilon)$ is the transition function,
5. $q_0 \in Q$ is the start state, and
6. $F \subseteq Q$ is the set of accept states.

Construct a PDA that recognizes the language $L = \{0^n 1^n \mid n \geq 0\}$



Nayeem Mahmood / 6AM / C161026

Here,

$Q = \{q_1, q_2, q_3, q_4\}$, $q_0 = \{q_1\}$

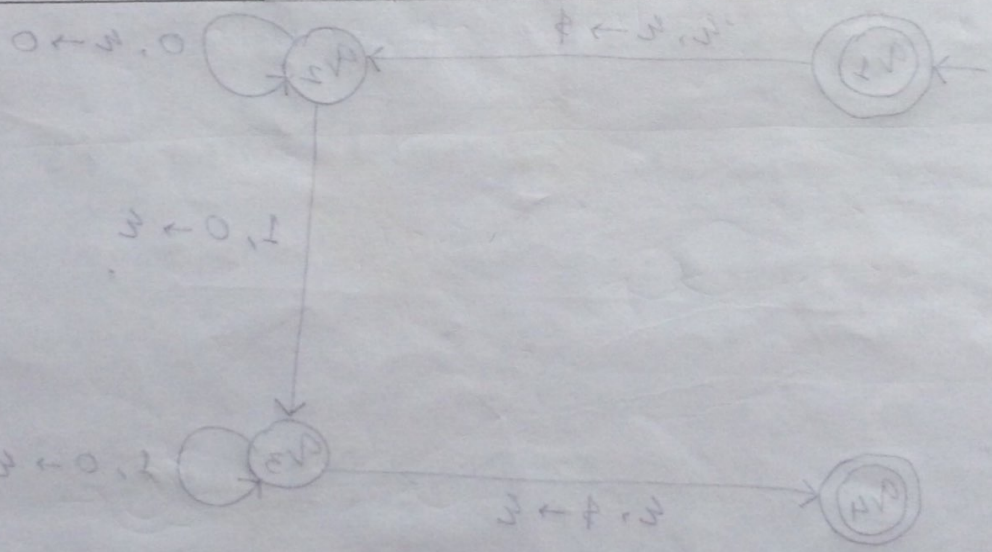
$\Sigma = \{0, 1\}$,

$\Gamma = \{0, \$\}$,

$F = \{q_1, q_4\}$, and

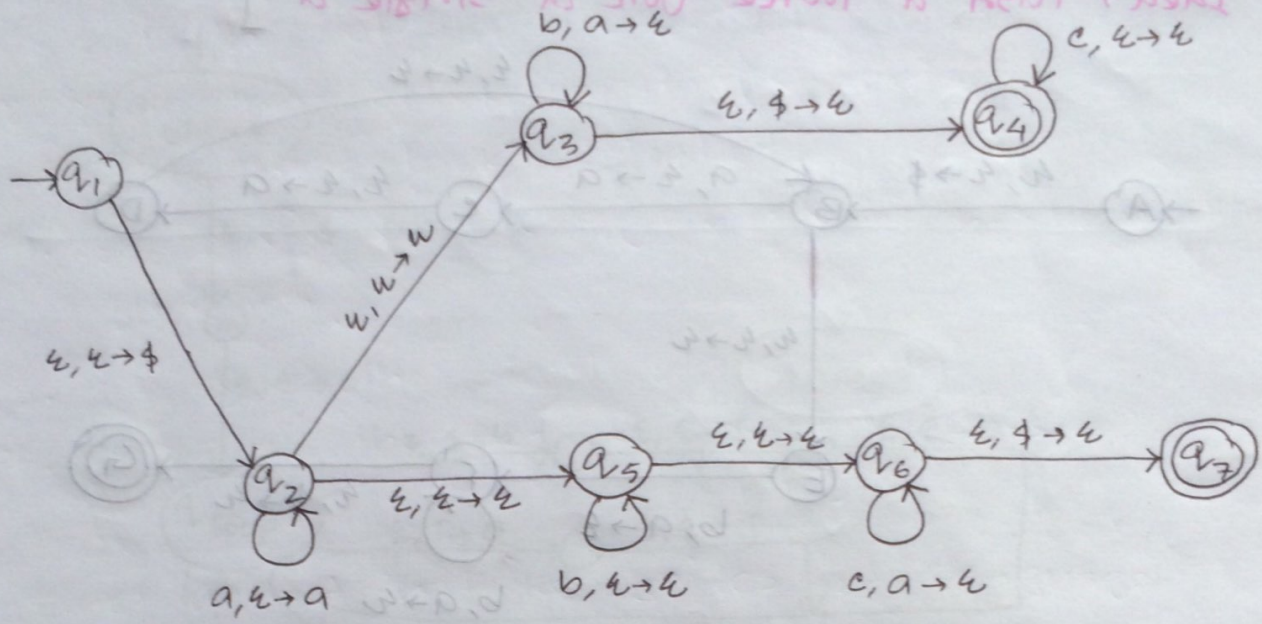
δ is given by the following table :

Input:	0			1			ε		
Stack:	0	$	ε	0	$	ε	0	$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$		$\{(q_3, \varepsilon)\}$				
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

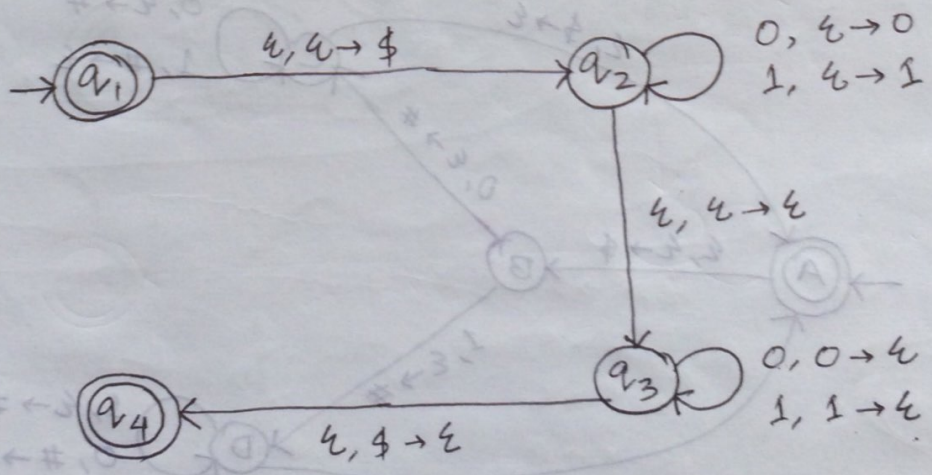


Construct a PDA that recognizes the language

$A = \{ a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k \}$

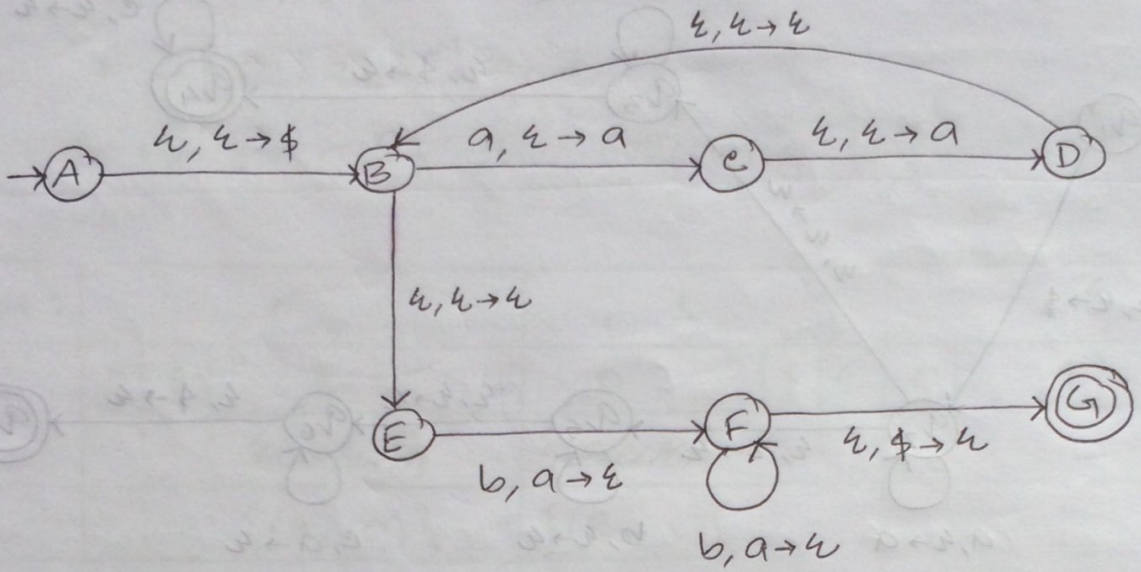


Construct a PDA that recognizes $\{ ww^R \mid w \in \{0,1\}^* \}$, where w^R means w written in backwards.

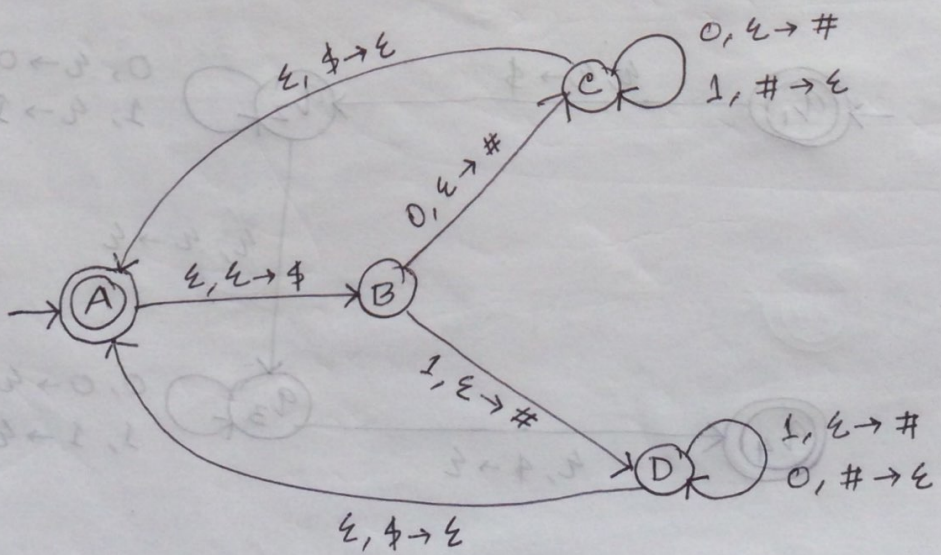


Construct a Pushdown automata for the language : $\{a^n b^{2n} \mid n \geq 1\}$

[* Idea : Push 'a' twice for a single 'a']



Construct a PDA for the language $L = \{w \mid w$ contains an equal number of 0s and 1s$\}$

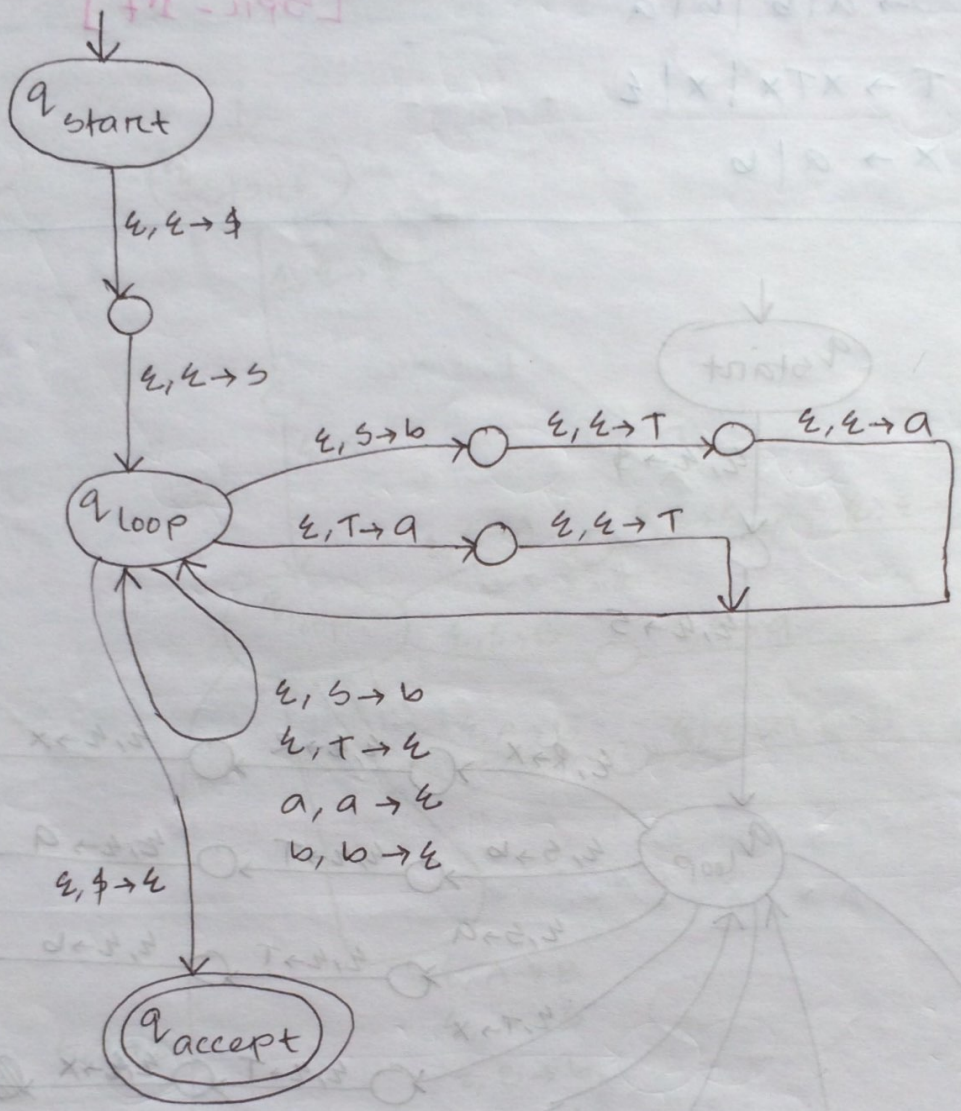


Nayeem Mahmood / 6AM / C161026

Construct a PDA for the following CFG:

$S \rightarrow aTb \mid b$

$T \rightarrow Ta \mid \varepsilon$



Nayeem Mahmood / 6AM / C161026

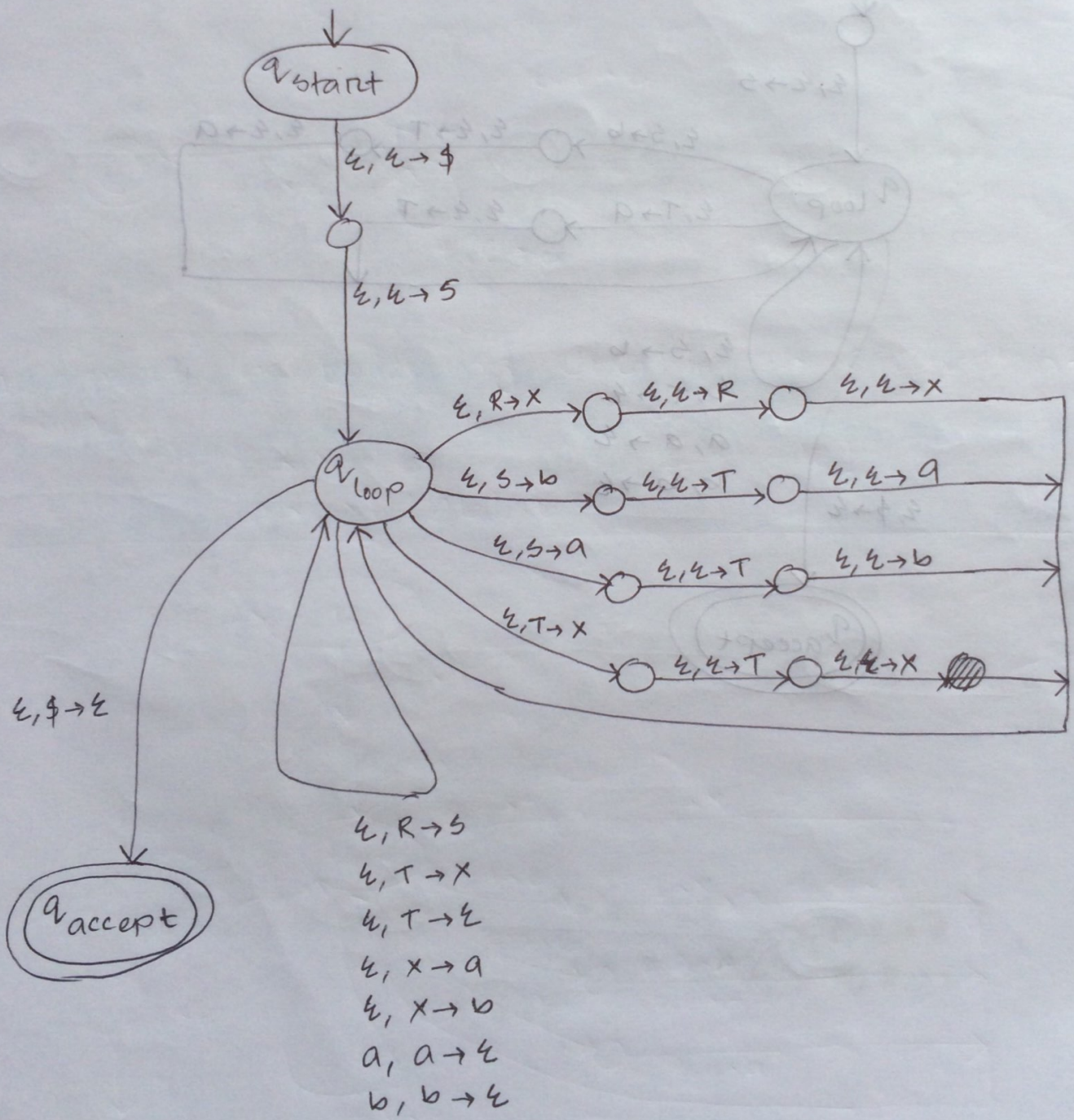
Convert the following CFG into an equivalent PDA :

$R \rightarrow XRX \mid S$

$S \rightarrow aTb \mid bTa$ [Spr - 17]

$T \rightarrow XTX \mid X \mid \varepsilon$

$X \rightarrow a \mid b$

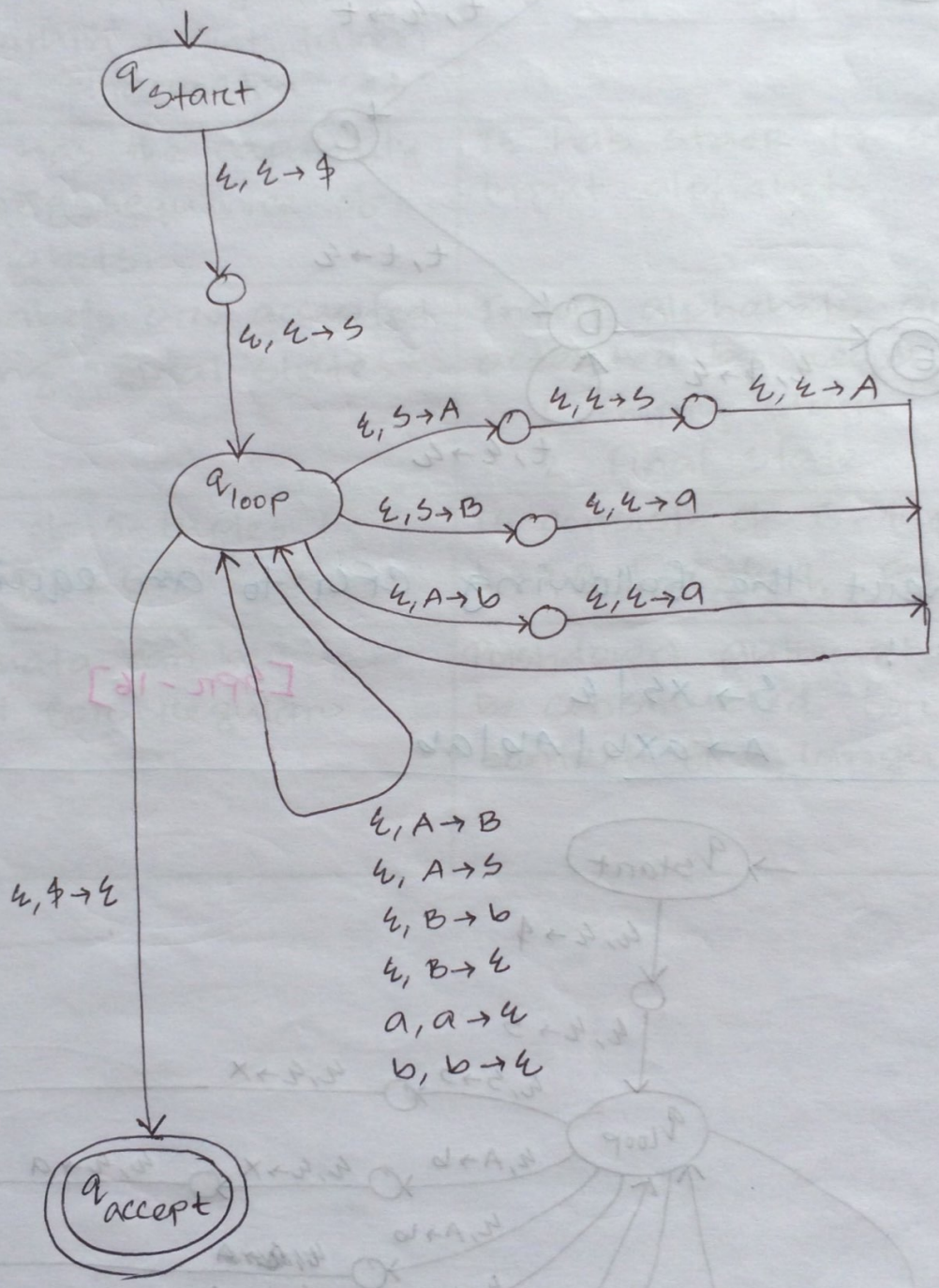


Nayeem Mahmood / 6AM / C161026

Convert the following CFG to an equivalent PDA:

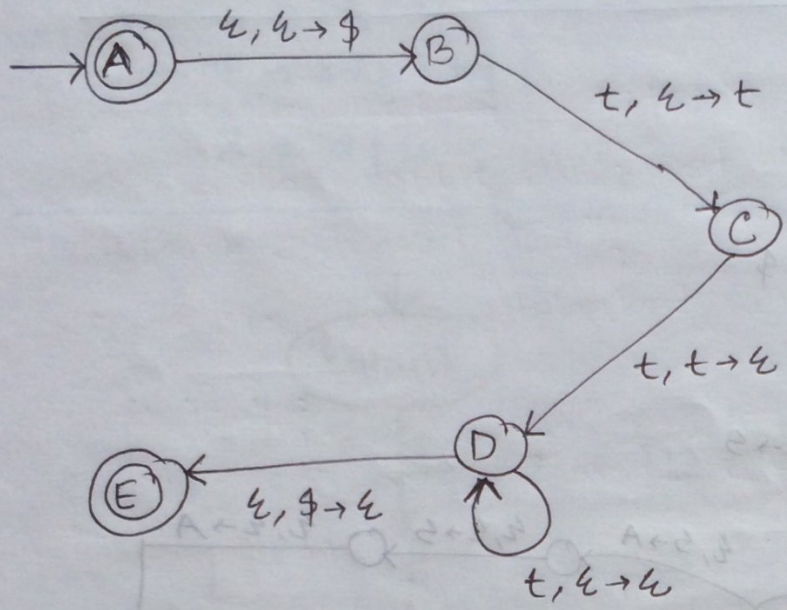
$S \rightarrow ASA \mid aB$
$A \rightarrow B \mid S \mid ab$
$B \rightarrow b \mid \varepsilon$

[Aut 16]



Nayeem Mahmood / 6AM / C161026

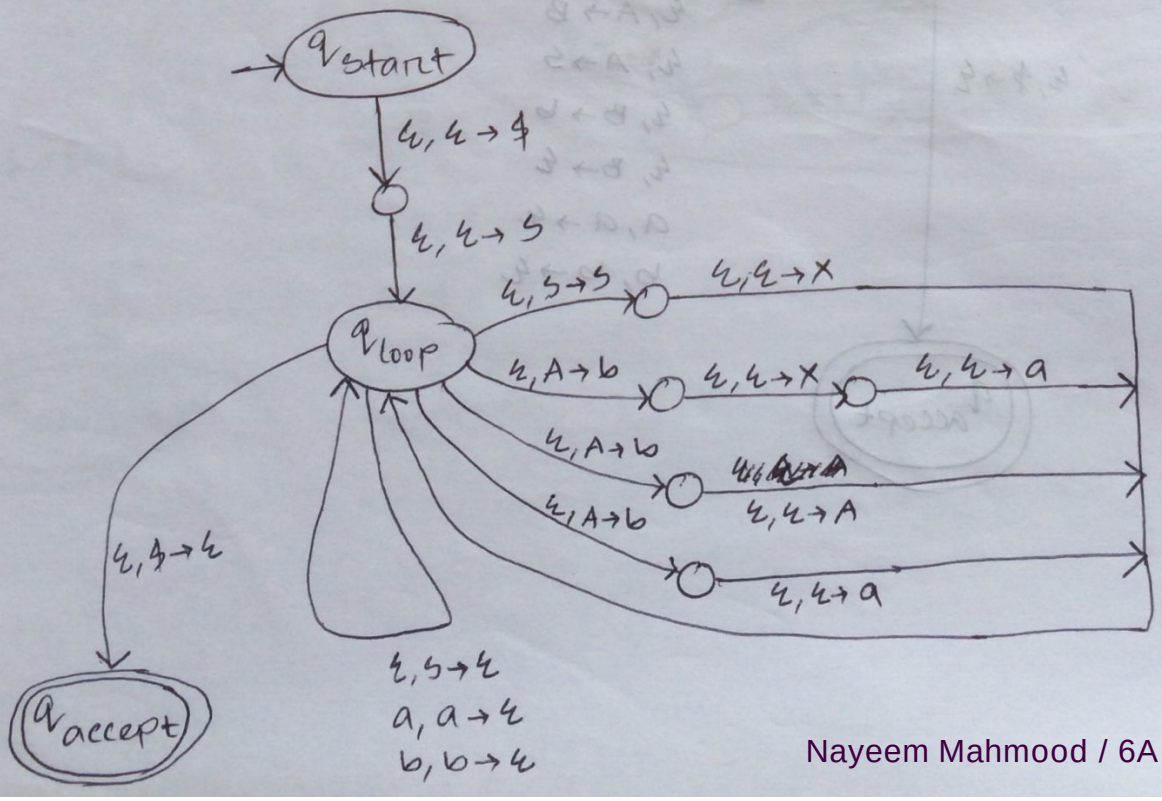
Construct PDA that recognizes the following language $L = \{t^p t^q \mid p \le q \le 2p\}$ [Aut-16]



Convert the following CFG to an equivalent PDA ! [Spr-16]

$S \to XS \mid \varepsilon$

$A \to aXb \mid Ab \mid ab$



Nayeem Mahmood / 6AM / C161026

Differentiate finite automation and pushdown automation. [Aut-17]

Finite Automation	Push Down Automation
A simple idealized machine that is used to recognize patterns within input taken from some character set.	A type of automation that employs stack.
It doesn't has the capability to store long sequence of input alphabets.	It has stack to store input alphabets.
Input alphabets are accepted by reaching "final states".	Input alphabets are accepted by reaching: 1. Empty stack 2. Final state
It consists of 5-tuples: $L = \{Q, \Sigma, \delta, q_0, F\}$	It consists of 6-tuples: $L = \{Q, \Sigma, \tau, \delta, q_0, F\}$
Finite automata can be constructed for regular language.	Pushdown automata can be constructed for context free language.

Nayeem Mahmood / 6AM / C161026

Formal def^{n} of Turing Machine

A Turing machine is a 7-tuple, $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where Q, Σ, Γ are all finite sets and -

1. Q is the set of states
2. Σ is the input alphabet not containing the blank symbol "$\sqcup$",
3. Γ is the tape alphabet, where $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$,
4. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function,
5. $q_0 \in Q$ is the start state,
6. $q_{accept} \in Q$ is the accept state, and
7. $q_{reject} \in Q$ is the reject state, where $q_{accept} \neq q_{reject}$.

Nayeem Mahmood / 6AM / C161026

Differences bet^{n} finite automata and Turing machines

1. A Turing machine can both write on the tape and read from it, but a finite automata doesn't have any tape.
2. The read/write head can move to the left & to the right.
3. The tape is infinite.
4. The special states for rejecting and accepting take effect immediately.

Design a Turing machine that decides $A = \{0^{2^n} \mid n \geq 0\}$, the language of all strings of 0s whose length is a power of 2.

Implementation level description

Let the Turing machine be M.

M = "On input string w:

1. Sweep left to right across the tape, crossing off every other 0.
2. If in stage 1 the tape contained a single 0, accept.
3. If in stage 1 the tape contained more than a single 0 and the number of 0s was odd, reject.

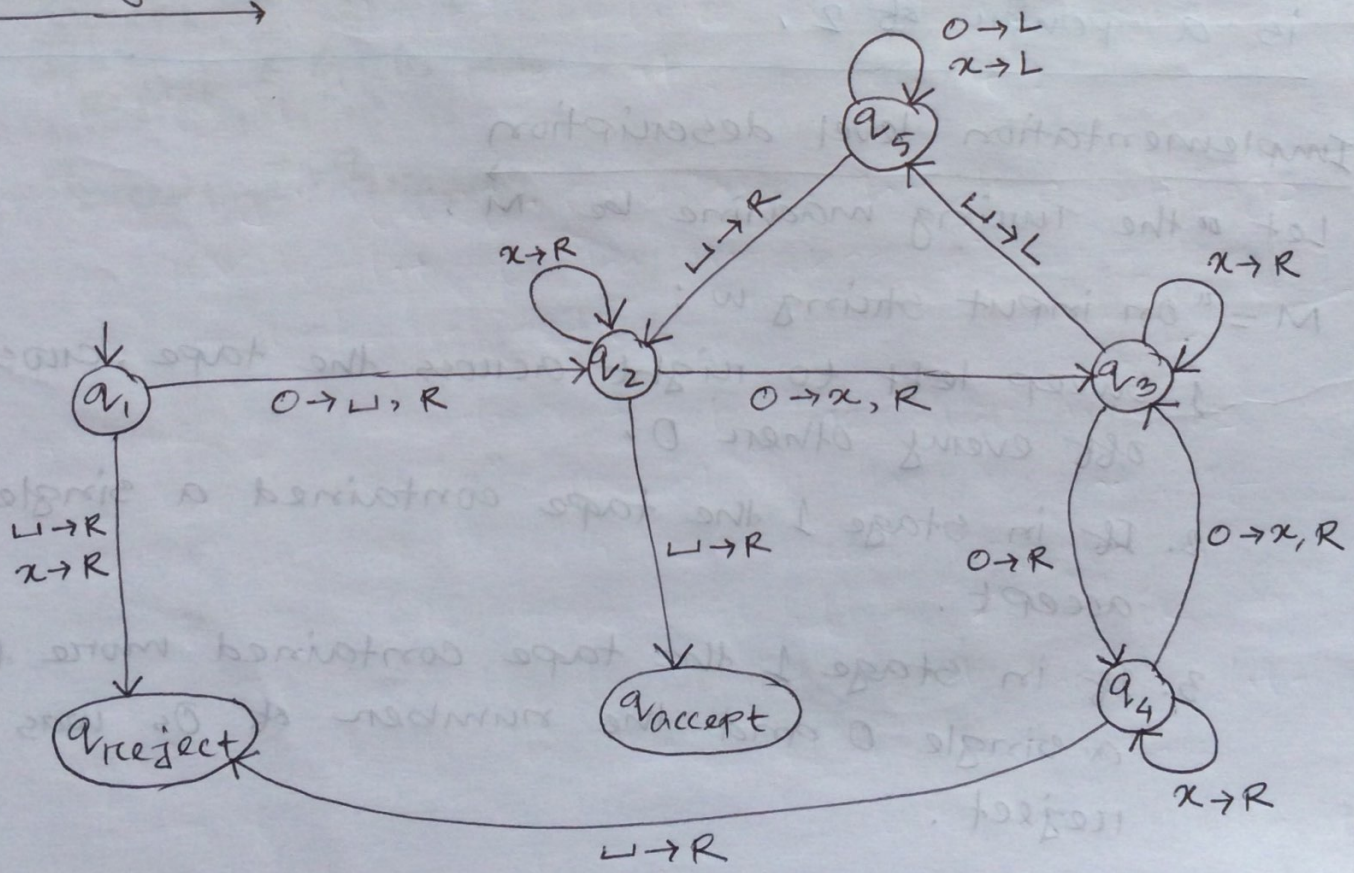
Nayeem Mahmood / 6AM / C161026

4. Return the head to the left-hand end of the tape.
5. Go to stage 1."

Formal Description →

- $\delta = \{q_1, q_2, q_3, q_4, q_5, q_{accept}, q_{reject}\}$,
- $\Sigma = \{0\}$,
- $\Gamma = \{0, x, \sqcup\}$,
- We describe δ with a state diagram below.
- The start, accept, and reject states are q_1, q_{accept}, q_{reject}, respectively.

State Diagram →



Nayeem Mahmood / 6AM / C161026

Design a Turing machine that decides the language $C = \{a^i b^j c^k \mid i \times j = k \text{ and } i, j, k \geq 1\}$

* Elementary Arithmetic

Implementation level description →

Let the required Turing machine be M.

M = " on input string w :

1. Scan the input from left to right to determine whether it is a member of $a^+b^+c^+$ and reject if it is not.
2. Return the head to the left-hand end of the tape.
3. Cross off an "a" and scan to the right until a "b" occurs. Shuttle between the b's and the c's, crossing off one of each until all b's are gone. If all c's have been crossed off and some b's remain, reject.
4. Restore the crossed off b's and repeat stage 3 if there is another "a" to cross off. If all a's have been crossed off, determine whether all c's also have been crossed off. If yes, accept ; otherwise, reject. "

Nayeem Mahmood / 6AM / C161026

Give implementation level description of a Turing machine for recognizing the following language, $E = \{\#x_1\#x_2\#...\#x_n \mid$ each $x_i \in \{0,1\}^*$ and $x_i \neq x_j$ for each $i \neq j\}$,

* Element Distinctness Problem

Let the required Turing machine be M.

M = " on input w :

1. Place a mark on top of the leftmost tape symbol. If that symbol was a "blank", accept. If that symbol was a "#", continue with the next stage. Otherwise reject.

2. Scan right to the next "#" and place a second mark on top of it. If no "#" is encountered before a blank symbol, only x_1 was present, so accept.

3. By zig-zagging, compare the two strings to the right of the marked #'s. If they are equal, reject.

4. Move the rightmost of the two marks to the next "#" symbol to the right. If no "#" symbol is encountered before a blank symbol, move the leftmost mark to the next "#" to its right and the rightmost mark to the "#" after that. This time, if no "#" is available for the rightmost mark, all the strings have been compared, so accept.

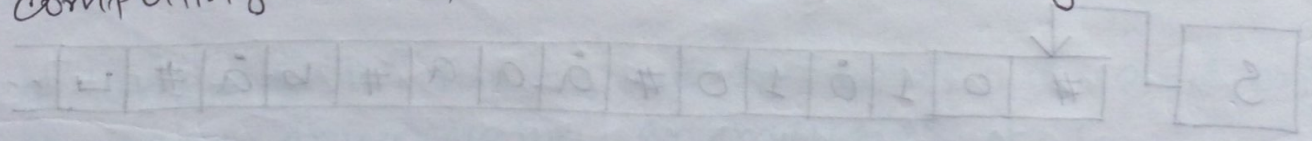
Nayeem Mahmood / 6AM / C161026

5. Go to stage 3."

State & explain Church-Turing Thesis.

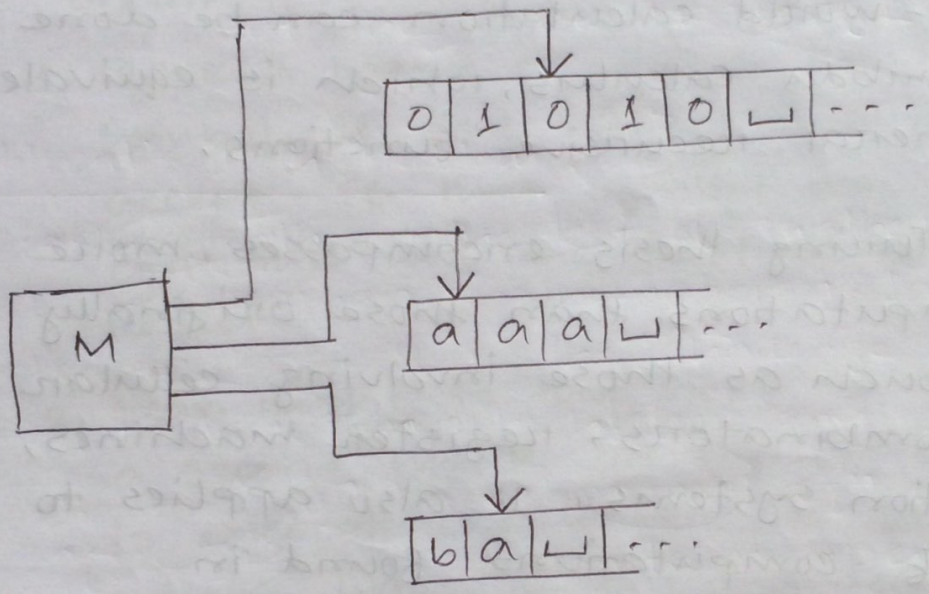
The Church-Turing thesis says that any real-world computation can be translated into an equivalent computation involving a Turing machine. In church's original formulation, the thesis says that the real-world calculation can be done using the lambda calculus, which is equivalent to using general recursive functions.

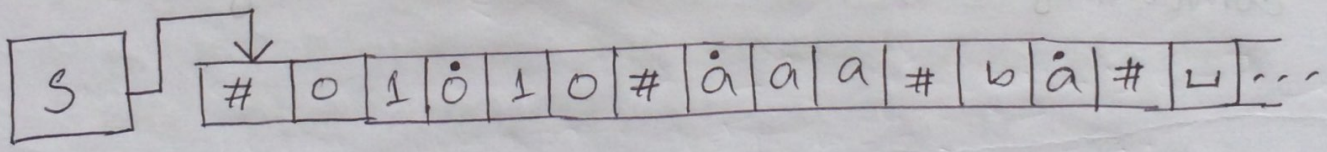
The Church-Turing thesis encompasses more kinds of computations than those originally envisioned, such as those involving cellular automata, combinators, register machines, and substitution systems. It also applies to other kind of computations found in theoretical computer science such as quantum computing and probabilistic computing.



Every multi-tape Turing machine has an equivalent single-tape Turing machine.

We show how to convert a multi-tape TM "M" to an equivalent single tape TM "S", the key idea is to show how to simulate "M" with "S".





S = " on input $w = w_1 \dots w_n$ ":

1. First S puts its tape into the format that represents all K tapes of M. The formatted tape contains

$$\#\dot{w}_1 w_2 \dots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \dots \#$$

2. To simulate a single move, S scans its tape from the first "#", which marks the left-hand end, to the (k+1)st "#", which marks the right-hand end, in order to determine the symbols under the virtual heads. Then S makes a second pass to update the tapes according to the way that M's transition function dictates.

3. If at any point S moves one of the virtual heads to the right onto a "#", this action signifies that M has moved the corresponding head onto the previously unread blank portion of the tape. So, S writes a blank symbol on this tape cell and shifts the tape contents, from this cell until the rightmost "#", one unit to the right. Then it continues the simulation as before."

A language is Turing-recognizable if and only if some enumerator enumerates it.

First we show that if we have an enumerator "E" that enumerates a language A, a TM "M" recognizes A. The TM "M" works in the following way:

M = "on input w:

1. Run E. Every that E outputs a string, compare it with w.
2. If w ever appears in the output of E, accept it."

Nayeem Mahmood / 6AM / C161026

E = " Ignore the input.

1. Repeat the following for $i = 1, 2, 3, \ldots$
2. Run M for i steps on each input s_1, $s_2, \ldots, s_i$.
3. If any computations accept, print out the corresponding s_j. "

Hilbert's 10th Problem

It is the tenth on the list of mathematical problems that the German mathematician David Hilbert posted in 1900. It is the challenge to provide a general algorithm which, for any given Diophantine eqn (a polynomial eqn with integer co-efficients and a finite number of unknowns), can decide whether the eqn has a solution with all unknowns taking integer values.

For example, the Diophantine eqn $3x^2 - 2xy - y^2z - 7 = 0$ has an integer solution : $x = 1, y = 2, z = -2$.

By contrast, the Diophantine eqn $x^2 + y^2 + 1 = 0$ has no such soln.

Hilbert's 10th problem has been solved, and it has a negative answer : such a general algorithm doesn't exist. This is the result of combined work of Martin Davis, Yuri Matiyasevich, Hilary Putnam and Julia Robinson which spans 21 years. The theorem is now known as Matiyasevich's

theorem or the MRDP theorem.

Differences between Turing recognizable and Turing decidable language:

A language is recognizable iff there is a Turing machine which will halt and accept only the strings in that language and for strings not in the language, the TM either rejects, or does not halt at all.

A language is decidable iff there is a Turing machine which will accept strings in the language and reject strings not in the language.

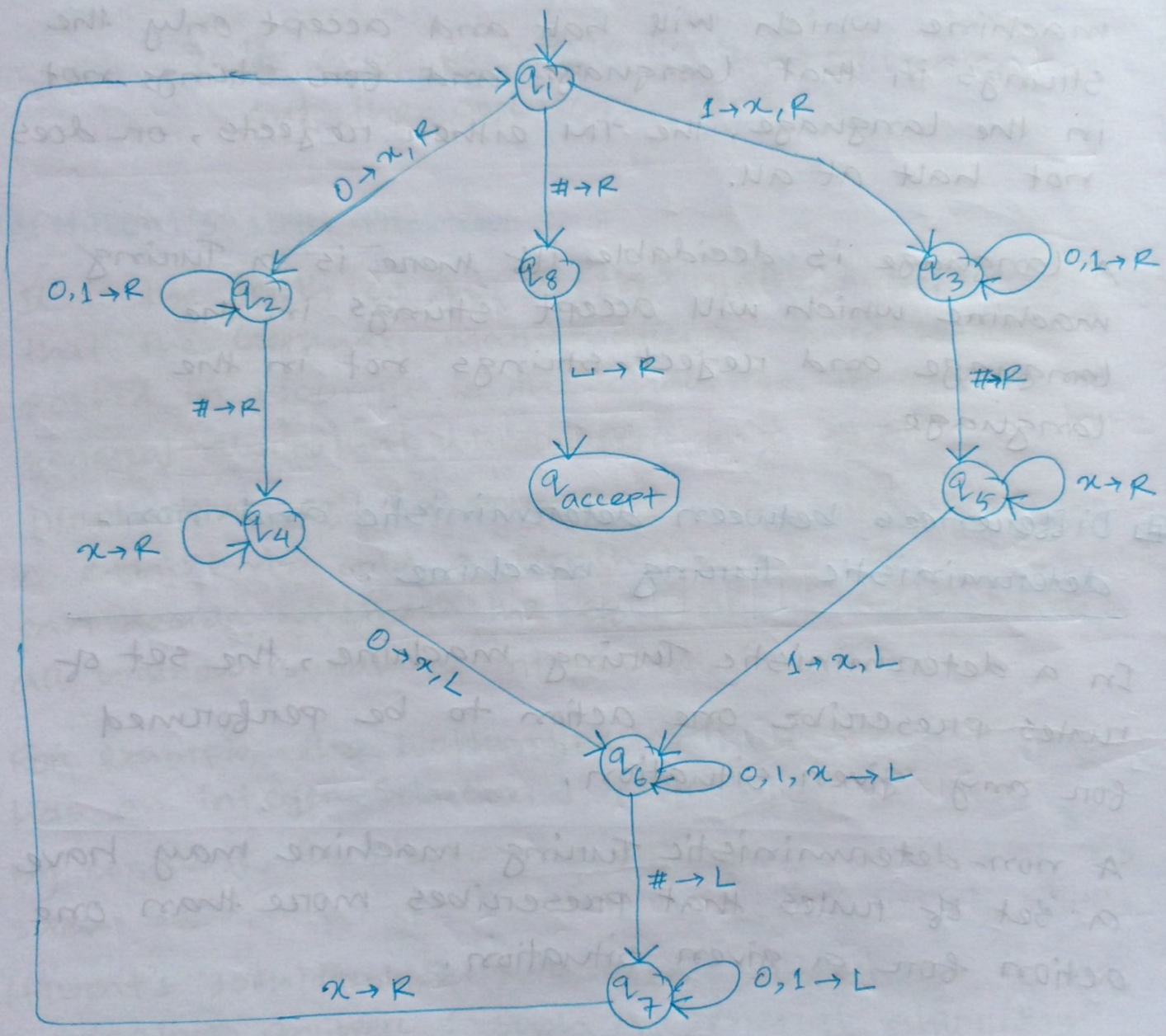
Differences between deterministic and non-deterministic Turing machine.

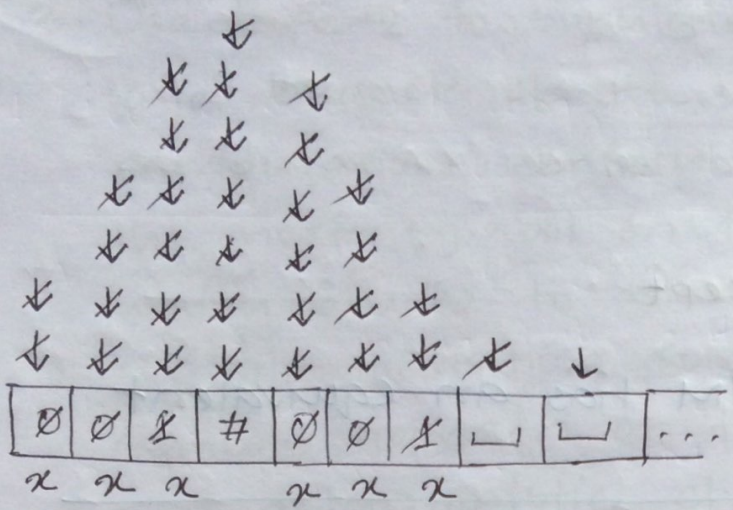
In a deterministic Turing machine, the set of rules prescribe one action to be performed for any given situation.

A non-deterministic Turing machine may have a set of rules that prescribes more than one action for a given situation.

Nayeem Mahmood / 6AM / C161026

Consider the Turing machine. Give the sequence of configurations that the machine traverse when started with the string 001#001





1. $q_1 001 \# 001$
2. $x q_2 01 \# 001$
3. $x0 q_2 1 \# 001$
4. $x01 q_2 \# 001$
5. $x01 \# q_4 001$
6. $x01 q_6 \# x01$
7. $x0 q_7 1 \# x01$
8. $x q_7 01 \# x01$
9. $q_7 x01 \# x01$
10. $x q_1 01 \# x01$
11. $xx q_2 1 \# x01$
12. $xx1 q_2 \# x01$
13. $xx1 \# q_4 x01$
14. $xx1 \# x q_4 01$
15. $xx1 \# q_6 xx1$
16. $xx1 q_6 \# xx1$
17. $xx q_7 1 \# xx1$
18. $x q_7 x1 \# xx1$
19. $xx q_1 1 \# xx1$
20. $xxx q_3 \# xx1$
21. $xxx \# q_5 xx1$
22. $xxx \# x q_5 x1$
23. $xxx \# xx q_5 1$
24. $xxx \# x q_6 xx$
25. $xxx \# q_6 xxx$
26. $xxx q_6 \# xxx$
27. $xx q_7 x \# xxx$
28. $xxx q_1 \# xxx$
29. $xxx \# q_8 xxx$

30. $xxx\#xq_8xx$

31. $xxx\#xxq_8x$

32. $xxx\#xxxq_8$

33. $xxx\#xxx\sqcup q_{accept}$

Every deterministic TM has an equivalent deterministic TM

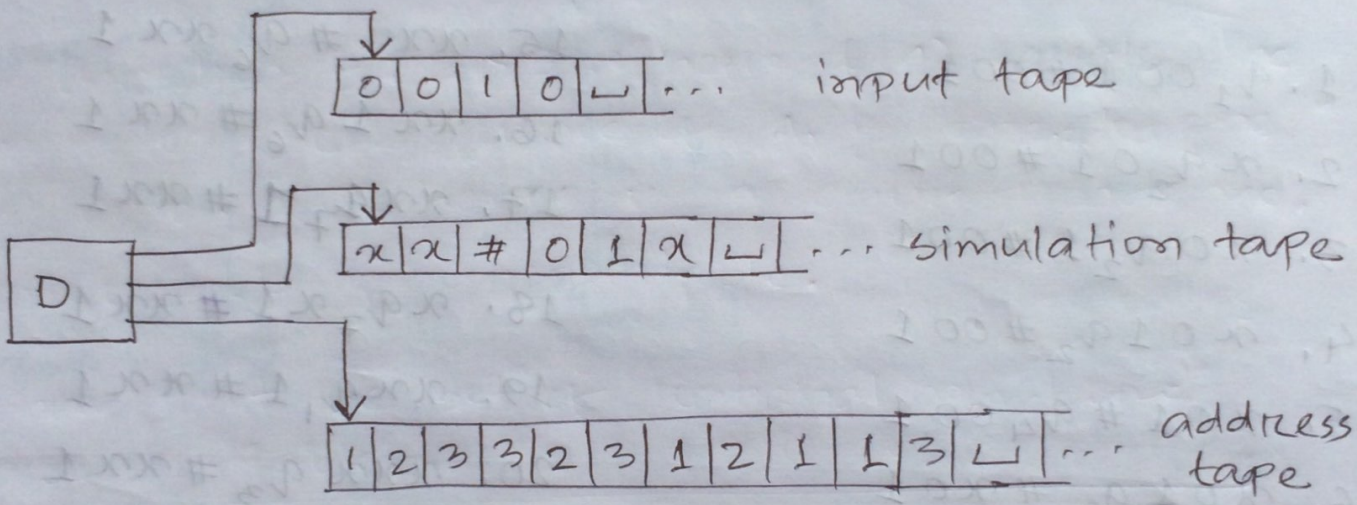


Fig: Deterministic TM "D" simulating non-deterministic TM "N".

1. Initially tape 1 contains the input w, and tapes 2 and 3 are empty.
2. Copy tape 1 to tape 2 and initialize the string on tape 3 to be ε.

3. Use tape 2 to simulate N with input w on one branch of its non-deterministic computation. Before each step of N, consult the next symbol on tape 3 to determine which choice to make among those allowed by N's transition function. If no more symbols remain on tape 3 or if this non-deterministic choice is invalid, abort this branch by going to stage 4. Also go to stage 4 if a rejecting configuration is encountered. If an accepting configuration is encountered, accept the input.

4. Replace the string on tape 3 with the next string in the string ordering. Simulate the next branch of N's computation by going to stage 2.

Show that set of infinitely long binary sequence is uncountable.

An infinite binary sequence is an un-ending sequence of 0s and 1s.

Let B be the set of all infinite binary sequences.

Let L be the set of all languages over alphabet Σ.

If A were the language of all strings starting with a 0 over the alphabet {0,1}, its characteristic sequence χ_A would be

$\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \ldots\}$

$A = \{0, 00, 01, 000, 001, \ldots\}$

$\chi_A = \{0, 1, 0, 1, 1, 0, 0, 1, 1 \ldots\}$

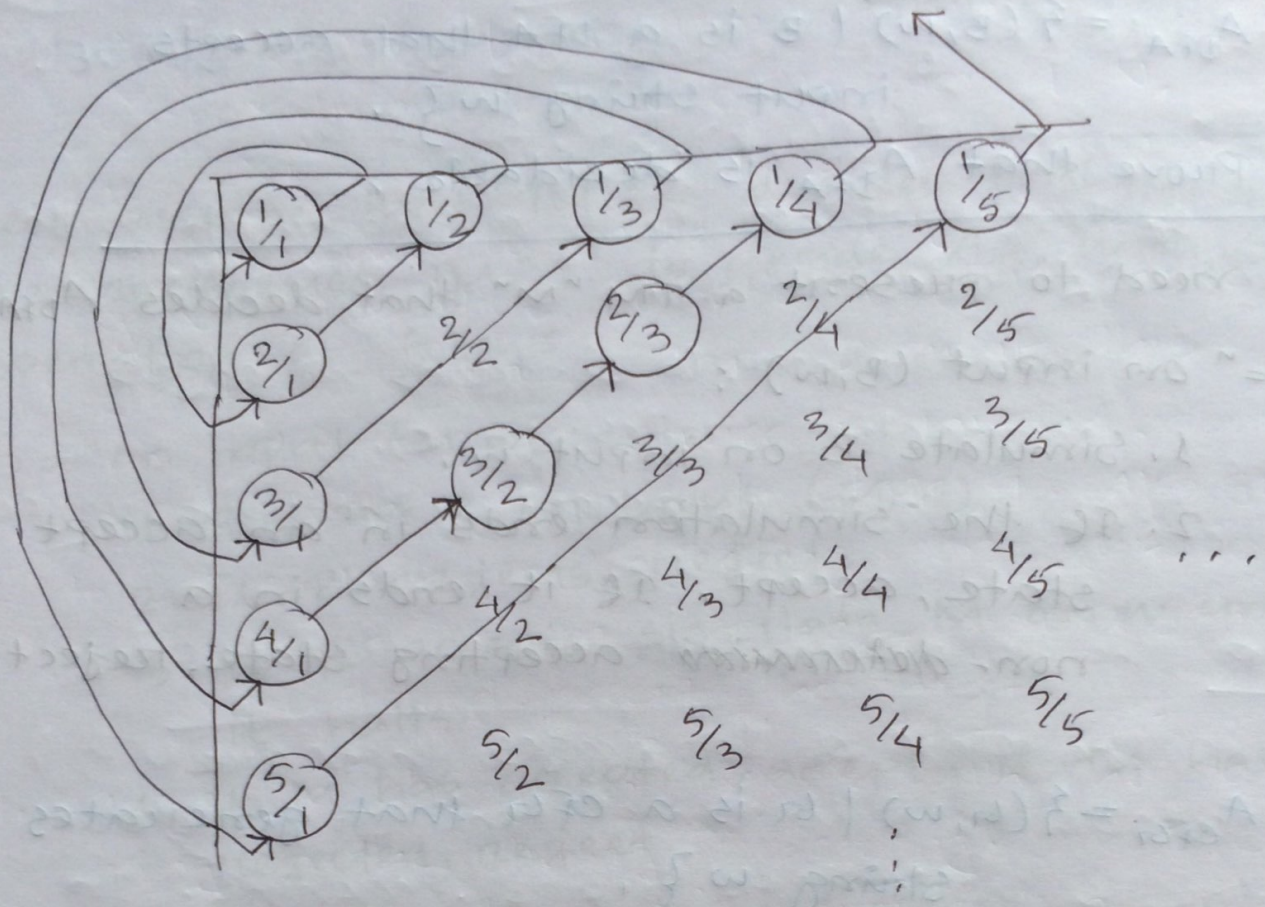
The function $f : L \to B$, where $f(A)$ equals the characteristic sequence of A, is one-to-one and onto, and hence is a correspondence, Therefore, B is uncountable.

Show that the set of rational numbers are countable.

In order to prove this theorem, we have to show how a complete list of the rational numbers can be formed.

We pair the first element on the list with the number 1 from N, the second element on the list with number 2 from N, and so on.

We must ensure that every member appear only once on the list. To get the list, we make an infinite matrix containing all the positive rational numbers.



So, the set of rational numbers are countable.

Nayeem Mahmood / 6AM / C161026

Let the language

$A_{DFA} = \{(B, w) \mid B \text{ is a DFA that accepts input string } w\}$.

Prove that A_{DFA} is decidable.

We need to present a TM "M" that decides A_{DFA}.

M = " On input (B, w) :

1. Simulate B on input w.
2. If the simulation ends in an accept state, accept. If it ends in a non-~~deterministic~~ accepting state, reject. "

$A_{CFG} = \{(G, w) \mid G \text{ is a CFG that generates string } w\}$.

Prove that A_{CFG} is a decidable language.

The TM for A_{CFG} follows;

S = " On input (G, w) :

1. Convert G to an equivalent grammar in CNF.
2. List all derivations with $2n-1$ steps, where n is the length of w; except $n=0$, then instead list all derivations with one step.
3. If any of these derivations generate w, accept, if not, reject. "

$HALT_{TM}$ is undecidable where

$HALT_{TM} = \{(M, w) \mid M \text{ is a TM and } M \text{ halts on input } w\}$,

---

Lets assume that TM "R" decides $HALT_{TM}$. We construct TM "S" to decide A_{TM}, with S operating as follows:

S = "on input (M, w):

1. Run TM R on input (M, w).
2. If R rejects, reject.
3. If R accepts, simulate M on w until it halts.
4. If M has accepted, accept; if M has rejected, reject."

clearly, if R decides $HALT_{TM}$, then S decides A_{TM}. Because A_{TM} is undecidable. $HALT_{TM}$ also must be undecidable.

Verifier

A verifier for a language A is an algorithm V, where

$A = \{w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c\}$.

We measure the time of a verifier in terms of the length of w, so a polynomial time verifier runs in polynomial time in the length of w.

Nayeem Mahmood / 6AM / C161026

CLIQUE is NP

The following is a verifier V for CLIQUE.

V = " on input ((G, k), c):

1. Test whether c is a subgraph with k nodes in G.
2. Test whether G contains all edges connecting nodes in c.
3. If both pass, accept, otherwise reject."

P

P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words,

$$P = \bigcup_k TIME(n^k)$$

NP

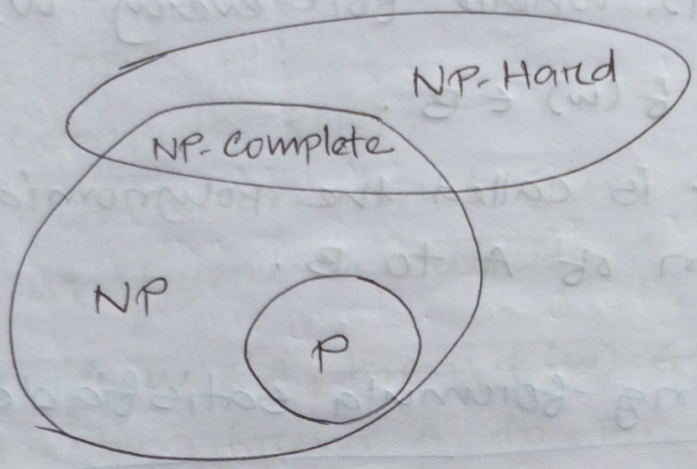
NP is the class of languages that have polynomial verifiers.

NP-Complete

A language B is NP-Complete if it satisfies two conditions:

Nayeem Mahmood / 6AM / C161026

1. B is NP,
2. Every A in NP is polynomial time reducible to B.



Subset sum problem is in NP

The following is a verifier V for Subset-Sum.

V = "on input ((S, t), c):

1. Test whether c is a collection of numbers that sum to t.
2. Test whether S contains all the number in c.
3. If both pass, accept; otherwise reject."

Polynomial Time Reduction

Language "A" is polynomial time reducible, to language "B", written $A \leq_P B$, if a polynomial time computable function $f : \Sigma^* \to \Sigma^*$ exists, where for every w,

$$w \in A \iff f(w) \in B.$$

The function f is called the polynomial time reduction of A to B.

Is the following formula satisfiable?

$$(x \vee y) \wedge (x \vee \bar{y}) \wedge (\bar{x} \vee y) \wedge (\bar{x} \vee \bar{y})$$

The formula isn't satisfiable. For any assignment of boolean values for x and y, it always makes one of the 4 clauses false. Therefore the formula, which is a conjunction of four clauses, is always false for any assignment of x and y.

NP-Hard

A problem "H" is NP-Hard when every problem in NP can be reduced in polynomial time to H.

If $A \leq_m B$ and B is decidable, then A is decidable.

We let M be the decider for B and f be the reduction from A to B. We describe a decider N for A as follows:

N = "on input w:

1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs."

Clearly, if $w \in A$, then $f(w) \in B$, because f is a reduction from A to B. Thus M accepts $f(w)$ whenever $w \in A$. Therefore, N works as desired.