Name : Abhishek Srivastava.

Reg. No. : 19BCE10071

Class No. : CSE 2004

Subject : Theory of Computation and Compiler design.

Date : 23rd Jan, 2021

Answers

(1)

(b) $(a^* ab + ba)^* a^* = (a + ab + ba)^*$

To prove equivalence of two above relations, we should try to reduce one.

$LHS = (a^* ab + ba)^* a^*$

$RHS = (a + ab + ba)^*$

Keeping $a^* = \varepsilon$, we obtain LHS as : $(ab + ba)^*$ — (i)

Now, Let $a1 = \varepsilon$, so $a1^* = \varepsilon$, hence LHS term is reduced to : a^* — (ii)

Hence overall expression considering the above two cases can be briefed as : $(a + ab + ba)^*$ since $(ab + ba)^*$ is already obtained in (ii).

And using 2nd term we get any combination of 'a' and hence it can be inside $(a + ab + ba)^*$

Hence Proved

(a) Let $L = T(M)$ where $M = (Q, \Sigma, \delta, z_0, F)$ is a p- (2)
finite automata.

We can modify Σ, Q, and δ like →

(i) If $a \in \Sigma_1 - \Sigma$ the symbol 'a' will ~~not~~ appear in any string of $T(M)$.

∴ we can ~~delete~~ 'a' delete 'a' ~~from~~ from Σ_1 and all transitions defined by a.

Here $T(M)$ is not ~~affected~~ affected.

(ii) If $\Sigma - \Sigma_1 \neq \phi$ we can add dead state d to Q.

Let $\delta(d, a) = d$ for all 'a' in Σ, ~~and~~ and $\delta(q, a) = d$, for all q in Q & 'a' in $\Sigma - \Sigma_1$. ∴ $T(M)$ is not affected.

Lets say M can be ~~written~~ obtained by applying (a) and (b) to Σ, Q, and δ.

So a new M' can be written as $(Q, \Sigma, \delta, q, F)$.

Now, lets define a new automation 'M' such that $M' = (Q, \Sigma, \delta, Q, F)$ where M' differs from M only in final state.

∴ $w \in T(M')$ iff $\delta(q_0, w) \in Q - F$ and $w \notin T(M)$

∴ $q^p - L = T(M')$ is Regular.

(3)

(b) Chomsky hierarchy of Languages.

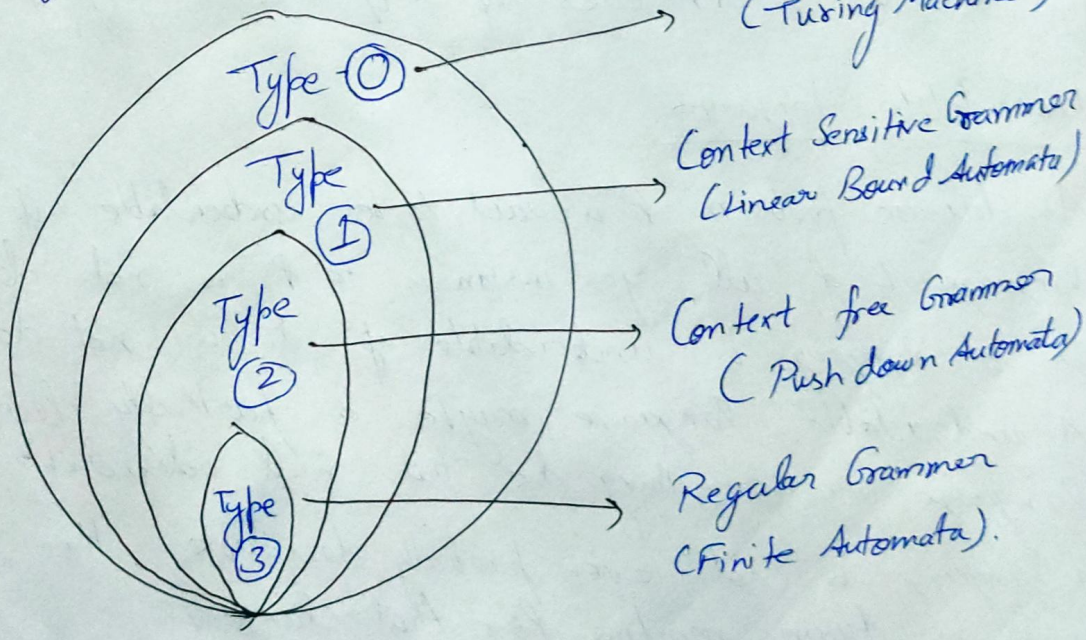
According to chomsky hierarchy, grammer is divided into 4 types :-

Type 0 known as Unrestricted Grammer.

Type 1 known as Context Sensitive Grammer.

Type 2 known as Context free Grammer.

Type 3 Regular Grammer.



Decidability and Undecidability

Decidable Language

A decision problem P is said to be decidable (ie, have an algorithm) if the language L of all yes instances to P is decidable.

Eg :-

1) Given a DFA. Does it accept a given word?
2) Given a DFA. Does it accept a word?
3) Given 2 DFA. Does they accept same Language?

Undecidable Language

A decision problem P is said to be undecidable if the language L of all yes instances to P is not decidable. or a language is undecidable if it is not decidable. A undecidable language maybe a partially decidable language or something else but not decidable. It a language is not even partially decidable, then there exist no turing machine for that language.

Recursive Language → TM always Halt.
Recursively Enumerable Language → TM may or may not Halt.
Decidable Languages → Recursive language.
Partially decidable Language → Recursively Enumerable language.
Undecidable → NO TM for this language.

4)

DFA	NFA
It stands for Deterministic finite Automata.	It stands for non-determi-nistic finite Automata.
It cannot use empty string transition.	It can use empty string transition.
Can be understood as 1 machine.	Can be understood as multiple little machine.
In DFA, next state is distinct set.	In NFA, each pair of state and input symbol can have many possible next state.
It is difficult to construct.	It is easier to construct.
Execution time is less	Execution time is more
Requires more space	Requires less space.

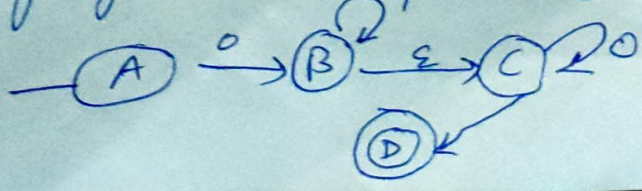
Epsilon NFA

ε → NFA

└→ Empty Symbol.

$\{Q, \Sigma, q_0, \delta, F\}$.

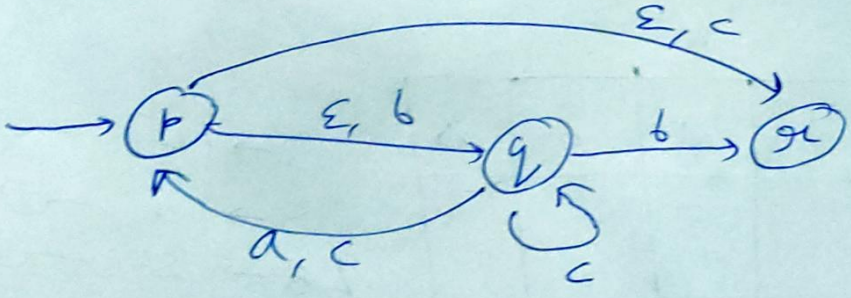
$\delta : Q \times \Sigma \cup \epsilon \rightarrow 2^Q$.



Taking Union of above Table.

	a	b	c	ε
p		q	q	{q, ϕ}
q	ϕ	ϕ	{p, q}	ϕ
ε	ϕ	ϕ	q	ϕ

Constructing State Diagram.



Making

	ε* a ε*	ε* b ε*	ε* c ε* table.
p	q p {p,q,r}	q r r	q p {p,q,r}
	r ϕ ϕ	r ϕ ϕ	r ϕ ϕ
	p ϕ ϕ	p q q	{r} r p
q	q p {p,q,r}	q {r,r}	q p {p,q,r}
			q {q,r}
r	r ϕ ϕ	r ϕ ϕ	r ϕ ϕ

(5)

Halting Problem in Turing Machine.

Halting means that the program on certain input will accept it and halt or reject it and halt and it would never go into an infinte loop. Basically, halting means terminating. So can we have an algorithm that will tell that the given program will halt or not. In terms of turing machine, will it terminate when run on some machine with some particular given input.

The answer is no, we cannot design a generalized algorithm which can appropiately say that a given problem will ever halt or not.

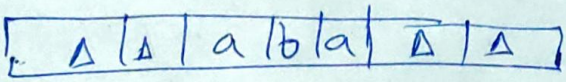
1. In General, we cant always know.
2. The best we cane, is run a program and see.
3. For many programs, we see it will always halt or some time loop.

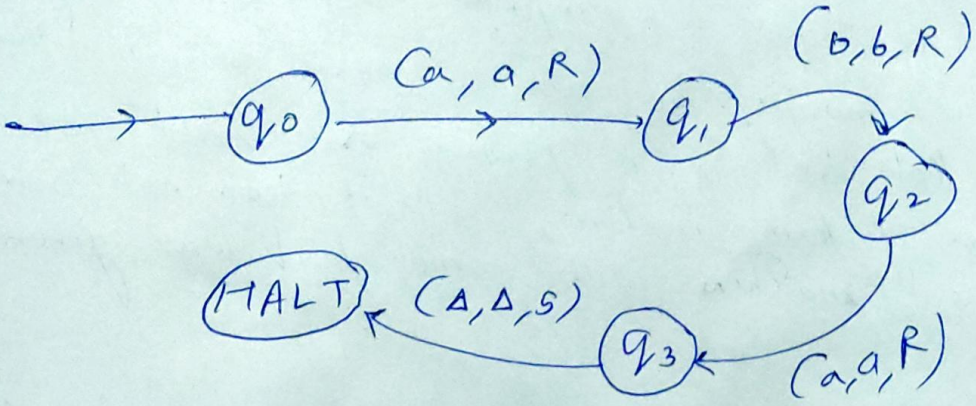
BUT FOR PROBLEM INGENERAL, THE QUESTION IS UNDECIDABLE

$\Sigma = \{a, b\}$

aba

Δ : for blank cells.





---

(1) (a) PDA to CFG.

$A = (\{q_0, q_1\}, \{a, b\}, \{Z_0, Z_1\}, \delta, Z_0, \phi)$

$\delta(q_0, b, Z_0) = (q_0, ZZ_0)$
$\delta(q_0, \varepsilon, Z_0) = (q_0, \varepsilon)$
$\delta(q_0, b, Z) = (q_0, \varepsilon)$
$\delta(q_0, b, Z) = (q_0, ZZ)$
$\delta(q_0, a, Z) = (q_1, Z)$
$\delta(q_1, b, Z) = (q_1, \varepsilon)$
$\delta(q_1, a, Z_0) = (q_0, Z_0)$

For starting Symbol :-

$S \to [q_0 z_0 q_0]$ ← (S → A) → P_1

$S \to [q_0 z_0 q_1]$ ← (S → B) → P_2

$\delta(q_0, b, z_0) = (q_0, z z_0)$

$[q_0 z_0] \to b[q_0 z][z_0]$

$[q_0 z_0] \to b[q_0 z][z_0]$

$[q_0 z_0] \to b[q_0 z][z_0]$

$[q_0 z_0] \to b[q_0 z][z_0]$

⇓

$[q_0 z_0 q_0] \to b[q_0 z q_0][q_0 z q_0]$ — P_3

$B = \{q_0, q_1\}$ $[q_0 z_0 q_0] \to b[q_0 z q_1][q_1 z q_0]$ — P_4

$[q_0 z_0 q_1] \to b[q_0 z q_0][q_0 z q_1]$ — P_5

$[q_0 z_0 q_1] \to b[q_0 z q_0][q_0 z q_1]$ — P_6

$\Rightarrow \delta(q_0, \varepsilon, z_0) = (q_0, \varepsilon)$

$[q_0 z_0 q_0] \longrightarrow \varepsilon$ — P_7

$\Rightarrow \delta(q_0, b, z) = (q_0, zz)$ | z^0

$\Rightarrow \delta(q_0, a, z) = (q_1, z)$ | z^1

$[q_0 z q_0] \to a[q_1 z q_0]$ — P_{12}

$[q_0 z q_1] \to a[q_1 z q_1]$ — P_{13}

$\Rightarrow \delta(q_1, b, z) = (q_1, z)$

$[q_1 z q_1] \rightarrow b$ ——— P_{14}

$\Rightarrow \delta(q_1, a, z_0) = (q_0, z_0)$ 2'

$[q_1 z_0 q_0] \rightarrow a[q_0 z_0 q_0] \longrightarrow P_{15}$

$[q_1 z_0 q_1] \rightarrow a[q_0 z_0 q_1] \longrightarrow P_{16}$

$\Rightarrow \delta(q_0, b, z) = (q_0 z z)$

$[q_0 z q_0] \rightarrow b[q_0 z q_0][q_0 z q_0]$ - P_8

$[q_0 z q_0] \rightarrow b[q_0 z q_1][q_1 z q_0]$ - P_9

$[q_0 z q_1] \rightarrow b[q_0 z q_0][q_0 z q_1]$ - P_{10}

$[q_0 z q_1] \rightarrow b[q_0 z q_1][q_1 z q_1]$ - P_{11}

(2)

(b)

To check whether grammer is LL(1)

		First
(1)	$S \to (L) \mid a$	(1) {(, a) First (1)
(2)	$L \to SL'$	(2) {(, a} First (L)
(3)	$L' \to ,\, L \mid \varepsilon$	(3) {, , ε) First(L')

Parse Table

	(	)	a	,	$
S	1		2		
L	3		3		
L'		4		5	

$S \to (L)$ — (1)
$S \to a$, — (2)
$L \to SL$ — (3)
$L' \to \varepsilon$ — (4)
$L' \to ,SL'$ — (5)

If parse table contains more than 1 entry within a cell position, then the grammer is NOT LL(1).