TOPIC _____ DATE _____

→ Sachin Kumar
→ 23010103466

26-11

Capstone

Theory of Computation

Unit II Finite Automata & Regular Expression.

1) Regular Expression for valid identifiers

Let Σ_A be the set of alphabets & Σ_D be the Set of digits regular expression R for all valid identifies (alphabet followed by any sequence of alphabet or digit) is

$$R = (\Sigma_A)(\Sigma_A \cup \Sigma_D)^*$$

- the keyboard (for, while if) are excluded in the lexical analysis phase following token recognition.

2) Design a DFA equivalent to R.

the DFA $M = \{Q, \Sigma_0, S, q_0, f\}$.

$Q = \{q_0, q_1, q_f\}$.

q_0 = Start State.

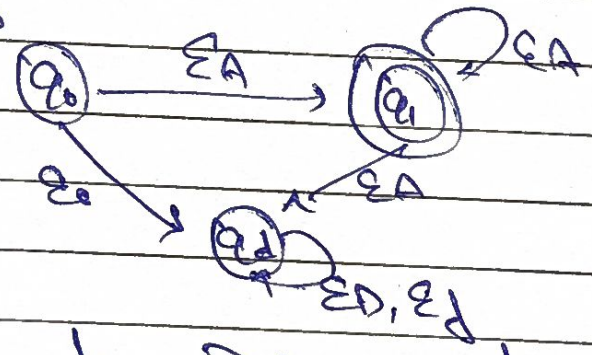
TOPIC _____ DATE _____

q_1 :- Accepting State. (valid identification Started).
q_d :- dead State.
Σ :- $\Sigma_A \cup \Sigma_0$
f :- $\{q_1\}$ (final State).

Transition table :-

State.	input Σ_A	input Σ_0	input (other)
q_0	q_1	q_d	q_d
q_1	q_1	q_1	q_1 (loop).
q_d	q_d	q_d	q_d

DFA diagram :-



(B.) Embedding the DFA in lexical analysis.

The DFA acts as State machine for recognizing the part of an identifier :-

i) DFA recognition :-

The lexer consumes input characters, tracking DFA State. When the input stream forces the DFA out of q_1 (encounting a space (or) operators) the operands before to that point is identified

TOPIC ________ DATE ________

as a Potential token.

2.) Keyword checker

The recognised String is then checked against a small, finite Set of reversed keyword (for, While, if). This is typically do one via fast table lookup.

3.) Token generation:-

- if the String is found in the keyword list, a keyword token generated.
- Otherwise, A identifier token is generated & its entry (name & type) is Stored in the Symbool table.

Q2) Unit-2 PDA & Context free language:-

1) formulate a CFG for well formed queries

Let O = <'opens'> & C = </close>. The Grammar is G models blanced nesting

$S \to OSC \,|\, SS \,|\, \varepsilon$

S → OSC handles nested Structure (eg <opens> .... </close>

TOPIC _____ DATE _____

$S \to SS$ handles Concatend Structure. (<open> <close> <open>)
$S \to \varepsilon$ handles empty query.

(2) Construct a PDA that accept Such queries—

The PDA accepts the language by empty stack. It uses the stack to tracks unmatched <open> tags

$M = (S_0, \{0, c\}, \{z_0, x\}, \delta, q_0, \phi)$.

State	input	top of stack	Next state	Action	Remarks
q_0	0	z_0	q_0	xz_0	Push x for 1st 0.
q_0	0	x	q_0	xx	Push x for N 0.
q_0	c	x	q_0	ϵ	pop x for Mc
q_0	ϵ	z_0	q_0	ϵ	Accept by empty Stack.

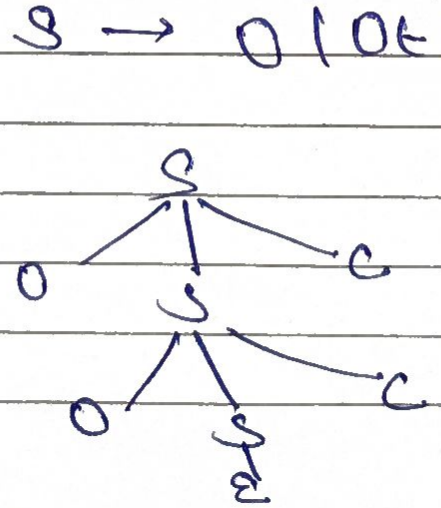
(3) Demonstrate the phrase tree.

Query :- <open> <open> </close> </close>

Derivation :-

$S \to 0Sc$
$S \to 0(0Sc)c$. $\{S \to 0Sc\}$
$S \to 0(0\epsilon c)\epsilon$ $\{S \to \epsilon\}$

Phrase tree.



TOPIC _____ DATE _____

Q) Unit-3 - Turing Machine & chomsky heirarchy

1) Justify why = $\{a^n b^n c^n \mid n \geq 1\}$ 3 is not context free.

We can Pumping lemma for CFL.

2) Choose String S:

Let p be the Pumping length

Choose $a^p b^p c^p$ etc.

3) Decompose S: S =

$uvxyz$ where $|vxy| \leq p$ & $|vy| \geq 1$

4) → Pumping lemma Since $|vwy| \leq p$ the Pumpable segments vwy can only contain symbol from atmost two blocks (only a's & b's or only b's & c's).

→ Case (vwy is in a's & b's) Pumping up (setting i=2) the no. of a's another b's but leaves the number of c's fixed at p.

→ Resulting String s' = uv²wy has unequal count of a's, b's

5) Conclusion: S ∉ L Since the condition of the Pumping lemma are violated, L can't be a CFL.

TOPIC ______ DATE ______

2.) Design a Turing Machine (TM) that accepts L.

The TM mark one a, one b & more one c. In the cycle untill all symbool are marked.

- Tape alphabet Γ: {a, b, c, x, y, z, □} {x, y, z and
- Core logic:

q0: Mark the leftmost a as x & transition to find b.

q1: find the 1st unmarked b & make it as y, then transition to find c.

q2: find the leftmost unmarked c & make it as z, then transition.

qret: Scan left to the starting point (x).

q check: After all a's marked, scan right to ensure the rest of the tape is just y's & finally □ (B).

2) Generate three address (TAC):

The TAC is generated based on expressions evolutin order Procedure → Parenthesis → multiplication → addition

i) $t_1 = A + B$
(ii) $t_2 = C - D$
(iii) $t_3 = t_1 * t_2$
(iv) $t_4 = t_3 + E$

3). Optimize the generated TAC:

There is no duplicate expression (common subexpression) in lines 1 & 2.

TOPIC ______________ DATE ______

Step by Step Configuration for "aaabbbccc"
The TM cycles three times to mark the pairs:-

Cycle 1 (Mark a, b, c): q_0, aaabbbccc → q_0 Xaabbccc
(Mark a) → q_b, Xaaybbzcc (Mark b) → q_{end}, Xaaybbcc
(Mark c) → q_0, Xaaybbzcc (~~mark b~~) for start).

Cycle 2 (Mark a_2 b_2 c_2):- q_0 Xaabbcc → q_{end} XXaYYbZZc
(Tape becomes XXaYYbZZcc)

Cycle 3 Mark($a_3 b_3 c_3$):- q_0 XXaYYbZZc → q_{end} XXXYYYZZZ

final check (q_{check}):- q_0 reads the marked a's (X's), transition to q_{check}. q_{check} scans over Y's & Z's until it hits D.

q_{check}, XXXYYYZZZ D → q(accept).

Unit-5 Code generation & optimization.
Expression (A+B) * (C-D)+E

Syntax:- Directed translation Scheme (s optimization).
Using a simple procedure-based grammar.

Production	Semantic rules:
→ E_1 + T	E.addr = new_temp() Ecol=line E.addr = E_1.addr + T.addr.
→ T_1 * F	T.addr = E_1.addr, d address
→ E_1	F.addr = E_1.addr; F.addr = Eon
→ id	F.addr = id.ke.

TOPIC ______ DATE ______

The code is already optimal wrt to CSE.

Dead Code removal :-

Assume the final result t_4 is used, all intermediate values (t_1, t_2, t_3) are necessary input for subsequent lines. No dead ~~state~~ code can be removed.

Optimised (TAC (unchanged)

i) $t_1 = A+B$
(ii) $t_2 = C-D$
(iii) $t_3 = t_1 * t_2$
(iv) $t_4 = t_3 + t$

(5.) Commulative - Advance reasoing & Application

Language L = Equal no. of 0's & 1's & No prefix has more than 1's than 0's.

Not regular :-
use the Pumping lemma for regular languages. Choose $S = 0^p 1^p$. Pumping lemma down ($i=0$) guess $0^{p-q} 1^p$ ($q \geq 1$) which has unequal counts violating L. this L is not regular.

Cumulative - Advance Reasoning & Application

Ans 1 Language L = Equal no. of 0's & 1's & no prefix has more than 1's ~~than~~ than 0's.

→ Not Regular : use the pumping lemma for regular language. Choose $s = 0^p 1^p$ pumping down ($i=0$) given $0^p = p$, $\therefore (b \geq 1)$ which has unequal counts violating L. Thus L is not regular.

→ Context free : The language L is accepted by a PDA which demonstrates its context free nature. The PDA's stack is essential for counting & comparing the non local dependencies (0_s ie 1_s)

Ans 2. The Grammer must enforce that every 1 is matched by a preceding 0.

$$S \rightarrow 0S1 \mid SS \mid \epsilon$$

This CFG generates all valid Dyck paths.

Ans 3 Design a PDA & trace "0011"

M accepts by empty stack, using X to count the excess no. of 0's.

Start Z_0 : $\delta(q_0, 0, Z_0) = \{(q_0, XZ_0)\}$

Push 0: $\delta(q_0, 0, X) = \{(q_0, XX)\}$

pop 1 (prefix check) : $\delta(q_0, 1, X) = \{(q_0, \epsilon)\}$

Accept : $\delta(q_0, \epsilon, Z_0) = \{q_0, \epsilon\}$

Trace the acceptance of "0011"

TOPIC _____ DATE _____

Input	state	stack (L→R)	Transition	Cond^n chuck (0's ≥ 1's)
0011	q_0	Z_0	Push 0	
0011	q_0	XZ_0	Push 0	$2 \geq 0$
0011	q_0	XXZ_0	Pop 1	$2 \geq 1$
0011	q_0	XZ_0	Pop 1	$2 \geq 2$
0011ϵ	q_0	Z_0	Empty stack	$2 = 2$
		ϵ	accept	