Title Date

Page No.

26-11

Name :- Vedant

Roll No :- 2301010369

Course :- B. Tech CSE

Subject :- Theory Of Computation

$\boxed{\text{CAPSTONE ASSIGNMENT}}$

DELTA

unit -1 Finite Automata and Regular Expression

Regular Expression for valid Identifiers
Let $\Sigma A$ be the set of alphabets and $\Sigma d$ be the set of digits
The Regular Expression R for all valid identifiers (alphabet followed by any sequence of alphabets or digits ) is

$$R = (\Sigma A)(\Sigma A \cup \Sigma d)^*$$

The Keywords (for, while, if ) are excluded in the lexical analysis phase following token Recognition

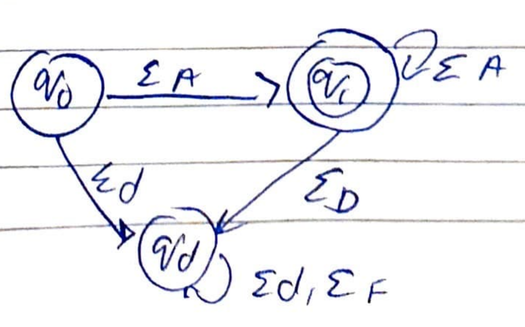
2 Design a DFA equivalent to R
The DFA $M = \{Q, \Sigma, \delta, q_0, F\}$

$Q = (q_0, q_1, q_f)$
$q_0$ = start state
$q_1$ = accepting state (valid identifier started)
$q_f$ = dead state (invalid start)
$\Sigma = \Sigma_A \cup \Sigma_D$
$f = \{ q_1$ is (final state) $\}$

Transition Table

| State | Input $\Sigma A$ | Input $\Sigma d$ | 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



Title Date Page No.

3 Embedding The DFA in a lexical Analyze The DFA acts as The state machine for recognizing The pattern of an Identifier

Q 1 DFA recognition: The lexer consume input character reaching DFA's state o when The input stream ~~pass~~ forces The DFA out of $q_1$ (encountering a space or operator) The operator reader to that point is identified as a potential Token

2 Keyword check: The recognized string is Then checked against a small finite list of reserved keywords (for while, if) This is typically done via a fast hash Table lookup

3 Token generation

- If The string is found in The keyword list, a key word Token is generated
- otherwise a Identifier Token is generated and its entry (lemme and type) is stored in The Symbol Table

Q2 unit 2 DDA and context free language

Formulate a CFG for well formed queries
let O = <open> and C = </close>', The grammar and module balanced nesting

$S \to OSC / SS / \epsilon$

$S \to OSC$ handles nested structure (eg <open> .... </loose>)
$S \to SS$ handles concatenated structure (eg <open> </close> <open>)

DELTA

Title ______________________ Date ________
Page No. ________

$S \to \epsilon$ handles empty query

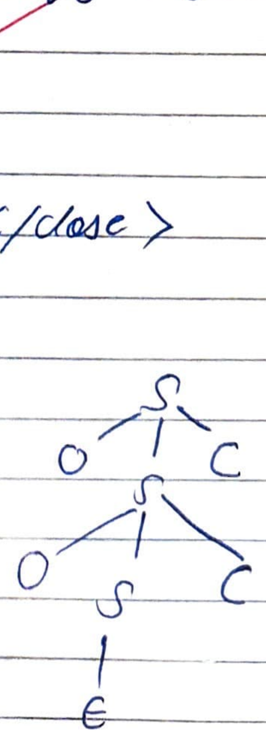
2 Construct a PDA that accept such queries
The PDA accept the language by empty stat. It uses the stack to track unmatched <open> tags

$M(\{q_0\}, \{O, C\}, \{Z_0, X\}, \{, q_0, \phi\})$

| State | Input | Top of stack | New state | stack operation | Rational |
|---|---|---|---|---|---|
| $q_0$ | O | $Z_0$ | $q_0$ | $XZ_0$ | Push X for first O |
| $q_0$ | O | X | $q_0$ | XX | Push X for nested O |
| $q_0$ | C | X | $q_0$ | $\epsilon$ | Pop X for matching C |
| $q_0$ | $\epsilon$ | $Z_0$ | $q_0$ | $\epsilon$ | Accept by empty stack |

3 ~~Demons~~ Demonstrat the parse tree
Query : <open> <open> </close> </close>

Derivation
$S \Rightarrow OSC$
$S \Rightarrow O(OSC)C \quad (S \to OSC)$
$S \Rightarrow O(O\epsilon C)C \quad (S \to \epsilon)$
parse tree :



Title ____________________ Date ________
Page No. ________

Q3 unit 3 Turing machine and chomsky Hierarchy

(1) Justify why $L = \{a^n b^n c^n \mid n \geq 1\}$ is not context free use pumping lemma for CFL

choose string s: let P be the pumping length choose s = a
$s = a^P b^P c^P \in L$

Decompose S: $S = uvwxy$ where $|vwx| \leq p$ and $|vx| \geq 1$

Pumping arguement: since $|vwx| \leq P$ the Pumpable segment vwx can only contain symbol from at most two blocks (only a's and b's, or only b's and c's)

Case (vwx is in a's and b's) pumping up (setting $i=2$) ↑ the no. of a's and/or b's but leaves the number of c's fixed at P

Resulting string $s' = uv^2wx^2y$ has unequal counts of a's, b's and c's (specifically, count(a) + count(b) > count(c))

conclusion $s' \notin L$, since the condition of the pumping lemma are violated, L can't be a CFL

2 Design a Turing machine (TM) that accepts
The TM mark one a, one b, one c in the cycle until all symbol are marked

Tape alphabet $\Gamma: \{a, b, c, x, y, z\ \square\}$ (x, y, z are markers)

Core logic

Title ______________________ Date ________
Page No. ________

$q_0$: mark the left most a as x and transition to find b

$q_b$: Find the left most ~~um~~ unmarked b and mark it as y then transition to find c

$q_c$: Find the left most unmarked c and mark it as z then transition to return

$q_{ret}$: scan left the starting point (x)

$q_{check}$: After all are marked, scan right to ensure the rest of the tape is ~~ju~~ just y's z's and finally □ (B)

Step by Step configuration for 'aaabbbccc'

The TM cycles three times to mark the three points

cycle 1 (mark a, b, c): $q_0$ aaabbbccc → $q_a$ xaabbccc (mark a) → $q_b$, xaa ybbcc (mark b) → $q_{ret}$ xaa ybb ccc (mark c), return left) →
$q_0$, xaa ybbzcc (Restart)

cycle 2 (mark $a_2 b_2 c_2$) and $q_0$ xaa bbccc → $q_{ret}$ xxa yyb zz (tape becomes xxa yybzzc)

cycle 3 (mark $a_3 b_3 c_3$): $q_0$ xxayybzzc → $q_{ret}$ xxxyyyzzz

Final check ($q_{check}$): $q_0$ read the marked a's (x's) transition to $q_{check}$. $q_{check}$ scans y's and z's until it hits [ ] check xxxyyyzzz

DELTA®

Title ______ Date ______
Page No. ______

Unit-4 code generation and optimization

~~Explain~~

Expression (A+B) * C (C-D) + G

Syntax - Directed Translation Scheme (S attributed)
using a simple ~~free~~ procedure-based grammer

Production semantic Rules

E1 + T    E.addr = new temp(): t1code || T.code ||
          E.addr = E1addr + T.addr

T1 * F    T.addr = new_temp(); T.code = T1.code || F.code ||
          T.addr = T1.addr * F.addr

→ (E1)    F.add = e1.addr; f.code = E1.code

→ id      F.add = id.Lexeme, F.code = G

2 Operate Three address code (TAC)
The TAC is generated based on expression's evaluation
order procedure: Paranthesis - multiplication → addition

1 $T_1 = A+B$
$t_2 = C-d$
$T_3 = t_1 * T_2$
$T_4 = T_3 + e$

3 Optimize the generate TAC
There is no duplicate expression (common sub expression)
in ~~this~~ lines 1 and 2. the code is already optimal
w.r.t to CSE

DELTA®

Assume the final result $T_4$ is used, all intermediate variables $(T_1, T_2, T_3)$ are necessary inputs for sub-sequent lines. No dead code can be removed

Optimized TAC (unchanged)

$T_1 = A + B$

$T_2 = C - D$

$T_3 = T_1 * T_2$

$T_4 = T_3 + E$

Q5 Cumulative - Advanced Reasoning and Application

Language L = (Equal no. of 0's and 1's no prefix has more 1's than 0's Dyck Paths)

Prove that L is context free but not regular

Not regular: Use the pumping lemma for regular language. Choose $S = 0^p 1^p$. Pumping down $(i=0)$ gives $0^{p-b} 1^p$ $(b \geq 1)$ which has unequal counts violating L. Thus L is not regular

Context free: The language L is accepted by a Push down Automation (PDA) shown below which demonstrates its context free nature. The PDA's stack is essential for counting and comparing the non local dependencies (0's vs 1's)

Provide a ~~CFG~~ CFG for this Language (L)

The grammar G must enforce that every 1 is matched by a preceding 0

$S \rightarrow 0S1S / \epsilon$

Title ______ Date ______
Page NO. ______

3 Design a PDA and trace 0011'

PDA Design (M)

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

Start 0: $\delta(q_0, 0, Z_0) = \{(q_0 X Z_0)\}$
Push 0: $\delta(q_0, 0, \cancel{X}) = \{(q_0, X, X)\}$

Pop 1 (prefix check): $\delta(q_0, 1, X) = \{(q_0\ \overset{\epsilon}{\cancel{X}})\}$
(pops only if 0's are in excess

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

B Trace the acceptance of '0011'

| Input | State | Stack (1→k) | Transition | Condition |
|---|---|---|---|---|
| 0011 | $q_0$ | $Z_0$ | push 0 | 2 > 0 |
| 0011 | $q_0$ | $X Z_0$ | Push 0 | 2 > 1 |
| 0011 | $q_0$ | $X X Z_0$ | Pop 1 | 2 > 2 |
| 0011 | $q_0$ | $X Z_0$ | Empty Stack ~~Pop~~ 1 | 2 = 2 |
| 0011 | $q_0$ | $\epsilon$ | accept ~~Empty~~ | |

DELTA®