Capstone Asss.

SUBMITTED BY:

NAME: ANMOL

ROLLNO: 2301010371

CLASS: BTECH CSE SEC-F

SUBMITTED TO: MANSI KATAL MA'AM

T-1 (Finite Automata & Regular Expression)

Regular Expression for valid Identifiers

Let Σ_A be the set of alphabets & Σ_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 $M = \{Q, \Sigma, \delta, q_0, F\}$

$Q = \{q_0, q_1, q_d\}$

q_0 = start state

q_1 = accepting state (valid identifier started)

q_f = dead state (invalid start)

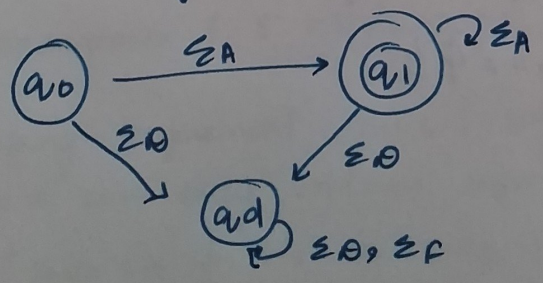
$\Sigma = \Sigma_A \cup \Sigma_D$

$F = \{q_1\}$ (final state)

δ: Transition Table

State	Input Σ_A	Input Σ_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:-



3) Embedding the DFA in a lexical Analyzer
The DFA acts as the state machine for recognizing the pattern of an identifier

1) DFA recognition : The lexer consumes input characters, tracking DFA's state. When the input stream forces the DFA out of q1 (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 KEYWORD token is generated
- otherwise, an IDENTIFIER token is generated & its entry (lexeme & type) is stored in the symbol table

UNIT-2 PDA & context free language

1) Formulate a CFG for well formed queries
Let O = '<open>' & C '</close>'. The grammar G models balanced nesting

$S \longrightarrow OSC \mid SS \mid \in$

$S \longrightarrow OSC$ handles nested structure (eg <open>.... </close>)
$S \longrightarrow SS$ handles concatenated structure (eg <open></close> <open></close>)
$S \rightarrow \in$ handles empty query

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

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

State	Input	Top of stack	New State	stack operation	Rationale
q_0	0	z_0	q_0	$X z_0$	push X for 0
q_0	0	X	q_0	X X	push X for nested 0
q_0	C	X	q_0	ϵ	pop X for matching C
q_0	ϵ	z_0	q_0	ϵ	accept by empty stack

3) Demonstrate the parse tree

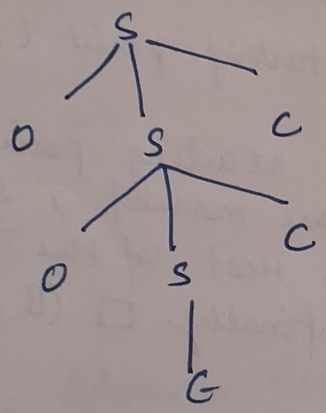
Query : <open><open><close><close>

$S \rightarrow OSC$

$S \rightarrow O(OSC)C \rightarrow (S \rightarrow OSC)$

$S \rightarrow O(O\epsilon C)C \rightarrow S \rightarrow \epsilon$

parse tree:-



Q3 UNIT-3 Turing Machine & chomsky Hierarchy

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

- choose string s: Let p be the pumping length choose

$$s = a^p b^p c^p \in L$$

- decompose s: $s = uvwxy$ where $|vwx| \leq p$ & $|vx| \geq 1$

- pumping arguement since $|vwx| \leq p$ the pumpable segment vwx can only contain symbols from atmost two blocks (only a's & b's or only b's & c's)

4) conclusion $s' \notin L$ $s' =$ ur wrong
of a's b's & c's (specifically, count(a) + count(b) ≠ count(c

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

The TM make one a, one b, one c in the cycle until all symbols are marked

Tape alphabets Γ: Σ a, b, c, X, Y, Z, □ } (X, Y, Z are markers)

Core logic

q_0 : Mark leftmost as X, & to find b

q_a : Find leftmost unmarked b & mark it as Y, then transition to find c

q_b : Find the leftmost unmarked c & mark it as Z, then transition to return

q_{ret} : scan left to the starting point (X)

q_{check} : scan left to the starting point (X) & right too after all a's are marked, scan right again to ensure the rest of the tape is just Y's, Z's & finally □ (B)

(Q4) UNIT(4) - code generation & optimization

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

1) Syntax - Directed Translation scheme (S attributed) using a simple precedence - based grammar

Production	Semantic Rules
$E \rightarrow E_1 + T$	E.addr = new_Temp(); E.code = E1.code ‖ T.code ‖ E.addr = E1.addr + T.addr
$T \rightarrow T_1 * F$	T.addr = new_Temp(); T.code = T1.code ‖ F.code T.addr = T1.addr * F.addr
$F \rightarrow (E_1)$	F.addr = E1.addr ; F.code = E1.code
$F \rightarrow (id)$	F.addr = id.lexeme, F.code = ∈

generate address code (TAC)

The TAC is generated based on expressions evaluation order (precedence : paranthesis → multiplication → addition)

1) $t_1 \rightarrow A+B$
2) $t_2 \rightarrow C-D$
3) $t_3 \rightarrow t_1 * t_2$
4) $t_4 \rightarrow t_3 + E$

3) Optimize the generated TAC
There is no duplicate expression present (common subexpressions). The code is already optimal wrt to CSE

Optimized TAC (unchanged)

1) $t_1 = A+B$
2) $t_2 = C-D$
3) $t_3 = t_1 * t_2$
4) $t_4 = t_3 + E$

Qs (Cumulative - Advanced Reasoning & Application)

1) Prove that L is content free but not regular
L = equal no of 0's & 1's & no prefix has more 1's than 0's (By cycle paths)

1) Not Regular: Use the pumping lemma for regular languages.
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

2) Content free : accepted by PDA shown below which demonstrates its content free nature. The PDA stack is essential for counting & comparing the non local

dependencies.

2) Provide a CFG for this language (L)
The grammar must enforce every one is matched by a preceding 0

$S \rightarrow 0S1S \mid \epsilon$

3) Design a PDA & Trace '0011'

A) PDA design

M accepts by the 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: $\delta(q_0, 1, X) = \{(q_0, \epsilon)\}$ (pop only if 0's are in excess)

Accept: $\delta(q_0, \epsilon, z_0) = \{(q_0, \epsilon)\}$

B) Acceptance of '0011'

Input	state	stack (L-R)	δ	condition check
		z_0	push 0	($0's \geq 1's$)
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	$2 \geq 2$
0011	q_0			
		z_0	Empty stack	$2 = 2$
001ϵ	q_0			
		ϵ	accept	