UNIT-I Finite Automata & Regular Expression

1. Regular Expression for valid identifies

Let ΣA be the set of alphabets & ΣA be the set of digits. The regular expression R for all valid identifies (alphabet followed by any sequence of alphabets or digits) is

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

The keywards (for, while, if) are excluded in the eenical analysis phase following taken 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)

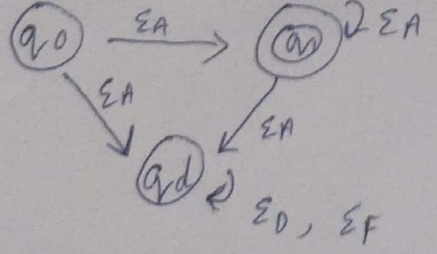
$\Sigma = \Sigma A \cup \Sigma D$

$A = \{q\}$ (final state)

Transition Table

State	Input ΣA	Input ΣD	Input (other)
q_0	q'	q_d	q_d
q'	q'	q_d	q' (loop)
q_d	q'	q_d	q_d

DFA (diagram)



Scanned with OKEN Scanner

3> Embedding the DFA is a lexical Analysis

The DFA acts as the state machine for recognizing the part of an identifier

1) DFA recognition : The lexer consumes input characters, tracking DFA's state. when the input stream forces the DFA out of q_1 (e.g. 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.

Q2 UNIT 2 PDA & Context free language.

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

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

$S \to OSC$ handles nested structures (eg <open> .. </close>)

$S \to SS$ handles concatenated structures (eg <open> </close> <open>

$S \to \epsilon$ 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

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

	Input	Top of stack	New state	stack operation	Rationale
q_0	0	Z_0	q_0	x = 0	Push x for first 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

5> Demonstrate the parse tree

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

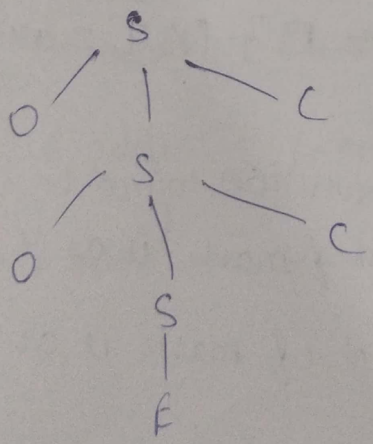
Derivation

$S \Rightarrow 0Sc$

$S \Rightarrow 0(0Sc)c \quad (S \to 0Sc)$

$S \to 0(0\epsilon c)c \,(S \to \epsilon)$

parse tree :



Q3 UNIT - 3 Turing Machine & Chomsky hierarchy

1) Justify why $L = \{a^n b^n c^n / n \geq 1\}$ is not context free

we use pumping lemma for CFLs.

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

3) Decompose S: $S = uvwxy$ where $|vwx| \leq P$ & $|vx| \geq 1$

4) Pumping argument since $|vwx| \leq p$ the pumpable segments vwx can only contain symbols from atmost two blocks (only a's & b's, or only b's & c's)

• case (vwx is in a's & b's) Pumping up (setting i=2) ↑ the no of a's and/or b's, but leaves the number of c's flexed at p.

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

5) Conclusion $s' \notin L$, since the conditions of the pumping lemma are violated, L can't be a CFL

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

The TM mark one a, one b & one c in the cycle all symbols are marked.

• Tape alphabets $\Gamma = \{a, b, c, x, Y, Z, \square\}$ (x, Y, Z are markers)

Core logic

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

qa : find the leftmost unmarked b & mask it as Y, then transition to find c

qb : find the leftmost unmarked c & mask it as Z, then transition to return

qret : Soon tell to the starting point (x)

qcheck : After all a's are marked, scan right to ensure the rest of the tape is first Yb, Z's & finally □ 'B'

Step by step configuration for 'aaabbbccc'

The TM cycles three times to mark the three pairs

i) Cycle 1 (Mark a_1, b_1, c_1) : q_0, aaabbbccc $\rightarrow$ q_a, Xaabbbccc (Mark a) $\rightarrow$ q_b, XaaYbbcc (Mark b) $\rightarrow$ q_{ret} $\rightarrow$ XaaYbbZcc (Mark c, return left) $\rightarrow$ q_0, XaaYbbZcc (Restore)

Cycle 2 (Mark $a_2 b_2 c_2$) : q_0, Xaabbccc $\rightarrow$ q_{ret}, XXaYYYZZZ

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 □ q_{check};

XXXYYYZZZ□ $\rightarrow$ q_{accept} (Accept)

UNIT-4 Code generation & optimization

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

Syntax - Directed translation scheme (s attributed) using a simple precedence - based grammar

Production	Semantic Rules
$\rightarrow E_1 + T$	E.addr = new_temp ( ) ; E.code = E_1.code ‖ T.code ‖ E.addr = E_1.addr + T.addr
$\rightarrow T_1 * F$	T.addr = new_temp ( ) ; T.code = T_1.code ‖ F.code ‖ T.addr = T_1.addr * F.addr
$\rightarrow (E_1)$	F.addr = E_1.addr ; F.code = E_1.code
$\rightarrow$ id	F.addr = id . lexeme ; F. code = ε

2) Generate three address code (TAC)

The TAC is generated based on expression's evaluation order precedence : parthesis → multiplication → addition

1) $t_1 = A + B$

2) $t_2 = C - D$

3) $t_3 = t_1 * t_2$

4) $t_4 = t_3 + E$

3) Optimize the generated TAC

There is no duplicate expression (common subexpressions) in lines 1 & 2. The code is already optimal wrt to CSE.

Dead Code Removal

Assume the final result t_4 is used, all intermediate variable (t_1, t_2, t_3) are nescessary inputs for subsequent line. No dead code can be removed.

Optimized TAC (unchanged)

1) $t_1 = A + B$

2) $t_2 = C - D$

3) $t_3 = t_1 * t_2$

4) $t_4 = t_3 + E$

Q3 (Commulative

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

i) Prove that L is content free but not regular

• Not Regular : Use the pumping lemma for regular languages Choose $S = 0^P 1^P$. Pumping down ($i=0$) gives 0^{P-L}, P ($L \geq 1$) which has unequal counts voilating 1. Thus L is not Regular

• Content free: The language L is accepted by a pushdown Automaton (PDA) shown below) which demonstrates its contentss free nature. The PDA's stack is essential for counting and compairing the non local sequ dependencies (0s vs 1s)

Provide a CFG for his language (!)
The Grammar G must enforce that every 1 is matched by a preceding 0

$S \rightarrow 0S1S \mid \epsilon$

This CFG generate all valid dyck paths

3) Design a PDA & Trace '0011'

A) PDA Design (M)

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) : $\delta(q_0, 1, x) = \{(q_0, \epsilon)\}$ (pops only if 0's are in excess.

B) Trace the acceptance of '0011'

Input	State	Stack (1→R)	Transition	Condition check (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	

Scanned with OKEN Scanner