180170107030
Gabu siddharth

FOREVER
Date: / / Page

1. What is regular language? write regular definition for following:

a. The set of identifiers of Java language.
b. comments consisting of a string surrounded by /* and */ without an intervening */ unless it appears inside the quotes "and".
c. All strings of 0's and 1's with even number of 0's and an odd number of 1's.

→ Regular expressions are used to denote regulers are used to denote regulers language. A language is regular if it can be expressed in term of regular expression.

a) alpha → a|b|...z|A|B|...Z|_|$
digit → 0|1|...9
alphas → alpha alpha*
digits → digit digit*
id → alphas(digits* | alphas*)*

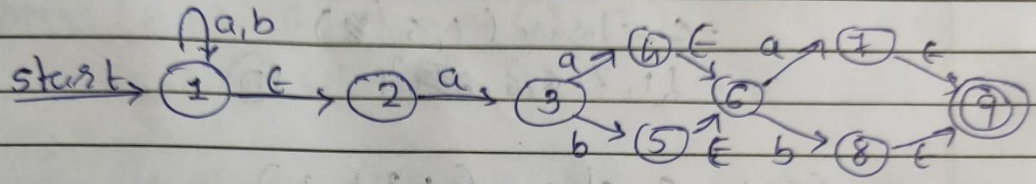
(b). S → /*
E → */
Q → "
EST → [^"*"]
ES2 → [^"/"]
comments → S(EST|*tES1)|(Q(ES1)(E)(EST)Q))E

(c) E → (00+11)*(01+10)
C → [00+11+01E+10E+0E]
E0 → I(C)+(0I

~~Regular expression using syntax tree~~

2. Construct DFA from following regular expression using Thompson's construction method.

$(a|b)^* a (a|b)(a|b)\#$ also minimize it.



		a	b
ε-closure(1) = {1, 2}	1	1	1
ε-closure (2) = {2}	2	3	ϕ
ε-closure (3) = {3}	3	4	5
ε-closure (4) = {4, 6}	4	ϕ	ϕ
ε-closure (5) = {5, 6}	5	ϕ	ϕ
ε-closure (6) = {6}	6	7	8
ε-closure (7) = {7, 9}	7	ϕ	ϕ
ε-closure (8) = {8, 9}	8	ϕ	ϕ
ε-closure (9) = {9}	9	ϕ	ϕ

$\therefore q_0 = \epsilon\text{-closure}(q_0')$

$q_0 = \{1, 2\}$ — (A)

$\text{Move}(A, a) = \epsilon\text{-closure}\{\delta(1, a) \cup \delta(2, a)\}$

$= \epsilon\text{-closure}\{1, 3\}$

$= \{1, 2, 3\}$ — (B)

move (A, b) = ε-closure (δ(1,b) ∪ δ(2,b))
= ε-closure (2)
= {1, 2}

move (B, a) = ε-closure (δ(1,a) ∪ δ(2,a) ∪ δ(3,a))
= ε-closure (1, 3, 4)
= {1, 2, 3, 4, 6} — (C)

move (B, b) = ε-closure (δ(1,b) ∪ δ(2,b) ∪ δ(3,b) ~~∪ δ(4,b) ∪ δ(6,b)~~)
= ε-closure (1, 5, 8)
= {1, 2, 5, 6, 8} — (D)

move (C, a) = ε-closure (1, 3, 4, 7)
= {1, 2, 3, 4, 6, 7, 9} — (E)

move (C, b) = ε-closure (5, 1, 8)
= {1, 2, 8, 9, 5, 6} — (F)

move (D, a) = ε-closure (1, 3, 7)
= {1, 2, 3, 7, 9} — (G)

move (D, b) = ε-closure (1, 8)
= {1, 2, 8, 9} — (H)

move (E, a) = ε-closure (1, 3, 4, 7)
= {1, 2, 3, 4, 6, 7, 9}

move (E, b) = ε-closure (1, 5, 8)
= {1, 2, 5, 6, 8, 9}

move (F, a) = ε - closure (~~1, 8, 7~~) (1, 3, 7)
= ~~{1, 3, 8, 7}~~ = {1, 2, 3, 7, 9}

move (F, b) = ε - closure (1, 8)
= {1, 2, 8, 9}

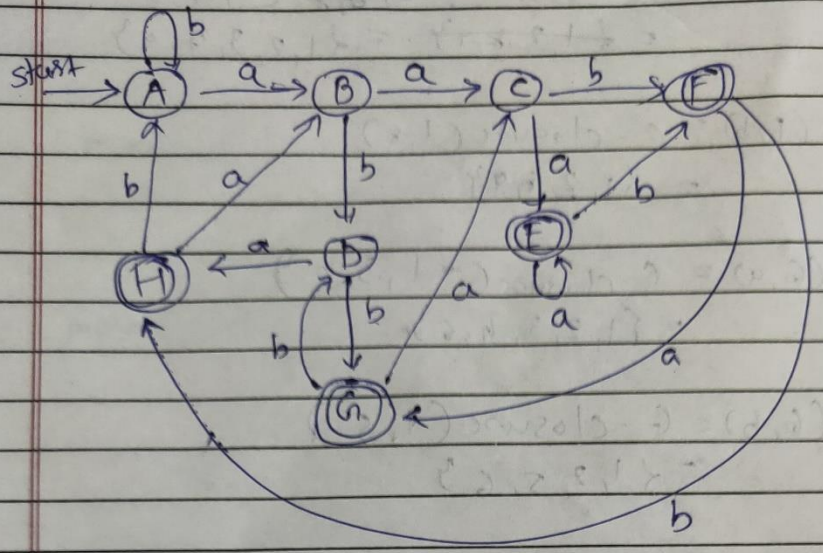
move (G, a) = ε - closure (1, 3, 4)
= {1, 2, 3, 4, 6}

move (G, b) = ε - closure (9, 5)
= {1, 2, 5, 6}

move (H, a) = ε - closure (1, 3)
= {1, 2, 3}

move (H, b) = ε - closure (1)
= {1, 2}

	a	b
A = {1, 2}	B	A
B = {1, 2, 3}	C	D
C = {1, 2, 3, 4, 6}	E	F
D = {1, 2, 5, 6}	G	H
(E) = {1, 2, 3, 4, 6, 7, 9}	E	F
(F) = {1, 2, 8, 9, 5, 6}	G	H
(G) = {1, 2, 3, 7, 9}	C	D
(H) = {1, 2, 8, 9}	B	A



3. ~~Construct DFA for following regular expression using syntax tree method.~~ Also minimize

DFA Optimization -

{A, B, C, D, E, F, G, H}

Non accepting states
{A, B, C, D}

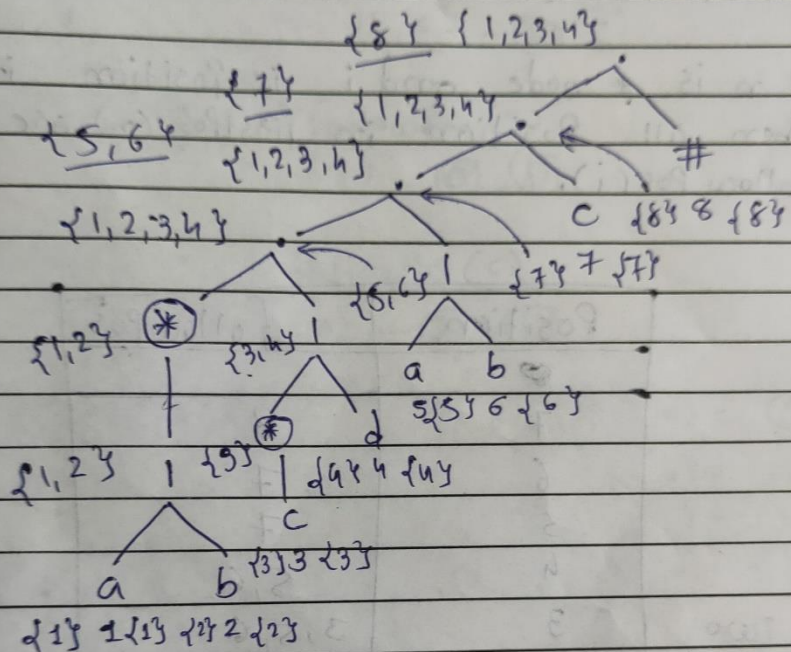
Accepting states
{E, F, G, H}

→ So In Non accepting states subgroups, doesn't have any states s and t such that for all input symbols a, states s and t have transitions on a states in same group. So DFA is already optimized.

Date: / / Page

3. Construct DFA for following regular expression using syntax tree method. also minimize it.

$(a|b)^*(c^*|d)(a|b)c\#$



→ Nullable node
Here * is only nullable node

→ A leaf with position $i = d_i$
firstPos = lastPos = d_i

⇒ if lastPos differ from firstPos then it will be displayed with underline.

→ if n is concatenation node with left child c1 and right child c2 and i is a position in lastPos(c1), then all position in firstPos(c2) are in followPos(i).

→ if n is * node and i is position in lastPos(n) then all position in firstPos(n) are in followPos(i).

	Position	followPos
	7	8
	6	7
	5	7
	4	5, 6
above two rules are applied because of '*'	3	3, 5, 6
	2	1, 2, 3, 4
	1	1, 2, 3, 4

Initial state = firstPos of root = {1, 2, 3, 4} — (A)

State A

$\delta((1,2,3,4), a)$ = followpos(1)
= {1, 2, 3, 4} — A

$\delta((1,2,3,4), b)$ = followpos(2)
= {1, 2, 3, 4} — (A)

FOREVER
Date: / / Page

$\delta((1,2,3,4), c)$ = followpos (3)
= {5, 6} — (B)

$\delta((1,2,3,4), d)$ = followpos (4)
= {5, 6} — (C)

$\delta((5,6), a)$ = followpos (5)
= {7} — (D)

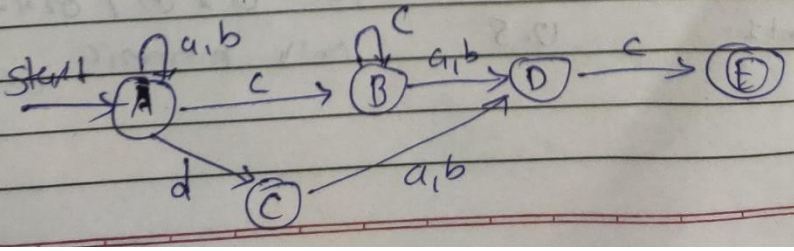
$\delta((5,6), b)$ = followpos (6)
= {7} — (D)

$\delta((5,6), c) = \{\}$
$\delta((5,6), d) = \{\}$

$\delta(7, a) = \{\}$
$\delta(7, b) = \{\}$
$\delta(7, c)$ = followpos (7) = {8} — (E)
$\delta(7, d) = \{\}$

$\delta((3,5,6), a)$ = (5) = 7 — (D)
$\delta((3,5,6), b)$ = (6) = 7 — (D)
$\delta((3,5,6), c)$ = (3) = (3,5,6) — (B)
$\delta((3,5,6), d) = \{\}$

State	a	b	c	d
A = {1, 2, 3, 4}	A	A	B	C
B = {5, 6}	D	D	B	-
C = {5, 6}	D	D	B	-
D = {7}	-	-	E	-
E	-	-	-	-



Date: / /

5. Define token. Explain How it differs from Pattern and lexeme. Find token, Pattern and lexeme from following expressions.

Token:
Sequence of characters having a collective meaning is known as token.

Pattern:
The set of rules called Pattern associated with a token.

Lexemes:
The sequence of characters in a source Program matched with a Pattern for a token is called lexeme.

→ A lexeme is a string of characters that is a ~~st~~ lowest-level syntatic unit in the Programming language.

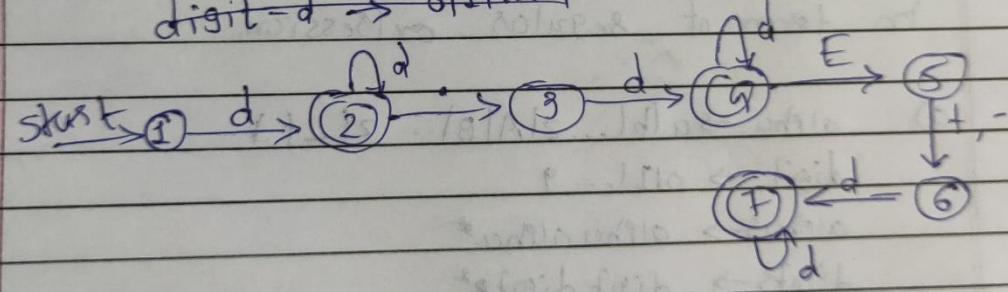
b). total = sum + 12.5

Token	lexeme	Pattern
identifier 1	total	letter followed by letter digit
operator 1	=	< or <= or = or <> or =>
identifier 2	sum	letter followed by digit or letter
operator 2	+	for - or * or / or %.
constant 1	12.5	only numeric constant

a.) if (x <= 5)

Token	Lexem	Pattern
identifier1	if	if
operator1	(	(
identifier2	x	letter followed by letter or digit
operator2	<=	< or > or <= or >= or <>
constant1	5	Numeric constant
operator3	)	)

b. Construct a DFA from recognizing the unsigned integers & unsigned real numbers with fraction.



digit = d → 0|1|...9
digits → d d*
optional_fraction → . digits | ε
optional_exponent → (E(+|-|ε) digits)|ε
num → digits optional_fraction optional_exponent