Theoretical Part

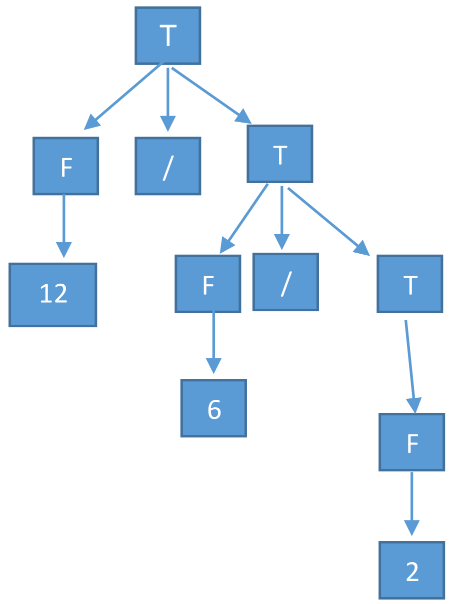
a. The grammar for the NFA:

1. $A \rightarrow aC \mid \varepsilon$
2. $B \rightarrow aA$
3. $C \rightarrow aB$
4. $(start)D \rightarrow aC \mid aE \mid \varepsilon$
5. $E \rightarrow aF$
6. $F \rightarrow aE \mid \varepsilon$

b. For any NFA, we set the grammar according to the states:
Given an NFA with the 5-tuple: $\{Q(states), \delta: Qx\Sigma \rightarrow P(Q)(State\ transition\ function), q_s(Initial\ state), F(Finite\ state)\}$
we add the following rules:

1. If state q belongs to F (Finite states): $q \rightarrow \varepsilon$
2. If state q_{next} $belongs\ to\ \delta(q, \sigma\epsilon\Sigma) : q \rightarrow \sigma q_{next}. This\ goes\ also\ in\ the\ case\ where\ \sigma\ is\ \varepsilon$

c. With these grammar rules, associativity of arithmetic operations doesn't follow the convention. Adding or multiplying don't pose a problem, subtracting and diving do. For example:
12/6/2. The Parsing tree:



The result will be 12/(6/2)=12/3=4
But expected result is (12/6)/2=2/2=1

d. Since C is left recursive, meaning it has a non-terminal left of a production (CD while C is non-terminal), the grammar isn't LL(1) – We may loop infinitely with this grammar.

To solve it, we take the following steps:

We make new, non-left-recursive rules with the following steps:

1. $C' \to D$
2. $C' \to DC'|\varepsilon$
3. $C \to cC'$

Now the new set of rules is:

1. $S \to AC$
2. $A \to aA \mid b$
3. $C' \to DC'|\ \varepsilon$
4. $C \to cC'$
5. $D \to d$

Next step – Make the table of Terminals Vs States:

	a	b	c	d	$
S	S->AC	S->AC			
A	A->aA	A->b			
C			C->cC'		
C'				C'->DC'	C'->ε
D				D->d	

The First of all states:

First(S) = First(A)

First(A) = {a, b}

First(C) = {c}

First(C') = First(D) ∪ {ε}

First(D) = {d}

The Follow of all states:

Follow(S) = {$}

Follow(A) = {c}

Follow(C) = {$}

Follow(C') = {$}}

Follow(D) = {d, $}

For the input abcdd:

Input	Step#	Stack (top is on left)
abcdd$	1	S
abcdd$	2	AC
abcdd$	3	aAC
bcdd$	4	AC
bcdd$	5	bC
cdd$	6	C
cdd$	7	cC'
dd$	8	C'
dd$	9	DC'
dd$	10	dC'
d$	11	C'
d$	12	DC'
d$	13	dC'
$	14	C'
$	15	ε

e.

i. The CFG for which this CFSM was constructed:
$S \to bAb$
$A \to (B\ |a$
$B \to Aa)$

ii. The transitions in the CFSM don't implement the reduce actions, so when reading the input, b gets us to state I1 and a to I5. This state doesn't make a transition with b, the next letter in the input and so the CFSM doesn't accept.

iii. The LR(0) will work on the input as follows:

Stack	Input	Action
I_0	b ((a a) a) b$	Shift, ->I1
I_0bI1	((a a) a) b$	Shift, ->I6
I_0bI1(I6	(a a) a) b$	Shift, ->I6
I_0bI1(I6(I6	a a) a) b$	Shift, ->I5
I_0bI1(I6(I6aI5	a) a) b$	Reduce A->a
I_0bI1(I6(I6AI8	a) a) b$	Shift, ->I9
I_0bI1(I6(I6AI8aI9	) a) b $	Shift, I10
I_0bI1(I6(I6AI8aI9I)10	a) b$	Reduce B->Aa)
I_0bI1(I6(I6BI7	a) b$	Reduce A->(B
I_0bI1(I6AI8	a) b$	Shift, ->I9

I_0bI1(I6AI8aI9	) b$	Shift, ->I10
I_0bI1(I6AI8aI9)I10	b$	Reduce B-> Aa)
I_0bI1(I6BI7	b$	Reduce A->(B
I_0bI1AI2	b$	Shift, ->I3
I_0bI1AI2b	$	Reduce S->bAb
I_0SI4	$	Reduce S'->S
I_0	$	done