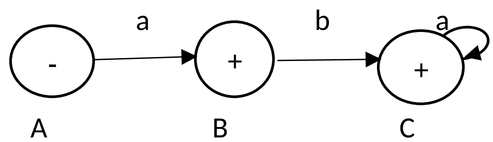
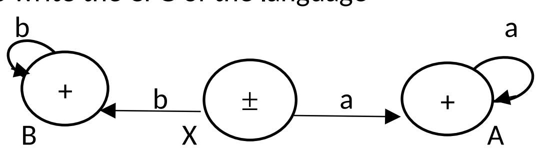
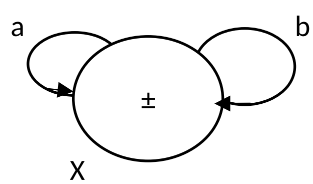
Compiler Design, Handout No.2, Assignment No.2

Tracing Grammars and constructing FA for some instructions in C++

Recall: To find a grammar of a language, the best is to construct its FA and then use FA to write its grammar.

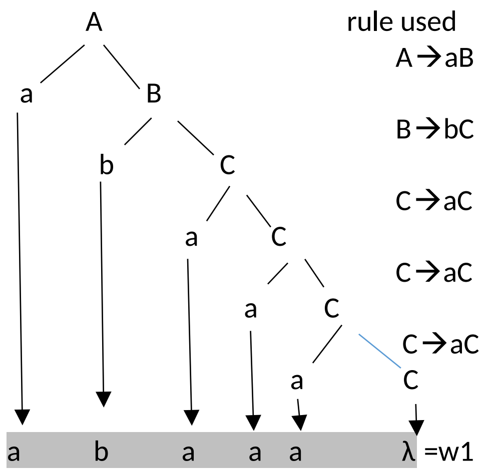
Examples. Write the grammar of the following FAs.

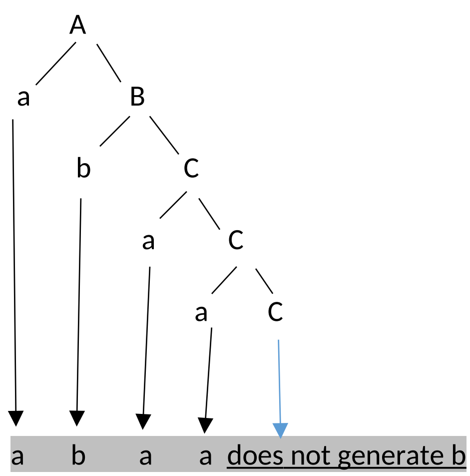
FA(Finite Automata)	Grammar or CFG(Context Free Grammar)
i)Σ={a,b }, L=aba*  	A→aB B→bC |λ C→aC C→λ Terminals or alphabets = {a,b }, Non-terminals or states = {A,B,C}
ii)Given L = a* + b*, find its CFG 1st, construct FA for the language. 2nd use the FA to write the CFG of the language  X is final: L1 = λ A is final: L2=aa* , B is final: L3=bb* L=L1+L2+L3 = λ + aa* + bb* , write λ= λ + λ = (λ +aa*) + (λ+bb*)=a* + b*	X→aA |bB | λ A→aA | λ B→bB | λ
iii) Given L (a+b)* , write the CFG of L  	X→aX | bX | λ

Tracing grammars: **T**o determine whether a given word is accepted or rejected by a grammar, we use one of the following two methods:

a. Parse tree
b. Left-most-derivation

Example. Trace the following CFG to determine whether word w is accepted or rejected:
CFG: A→aB, B→bC, C→aC | λ. Terminals={a,b,c}, Non-terminals={A,B,C}

<table>
<tr><th>Parse tree method</th><th>Left-most-derivation method</th></tr>
<tr><td>
i) w1 = abaaa: Start with the first rule: A→aB. Since "a" is terminal we stop, but "B" is nonterminal, we expand it by using rule: B→bC. Stop when there are no more non-terminal letters exist.


ii)Trace w2=abaab:


CFG rejects w2
</td><td>
i)w1=abaaa: Start with the first rule A→aB, go from left-to-right, a is terminal but B is non-terminal, replace B with B→bC. Skip all terminals until you see a non-terminal, expand the non-terminal. Do this until there are no more non-terminals left in the grammar, the word is accepted if there are no more non-terminals left and reject otherwise.

A→ aB , replace B by bc
⟹abC , skip a,b and replace C with aC
⟹abaC , replace C with aC
⟹abaaC ,replace C with aC
⟹abaaaC ,replace C with λ
⟹ abaaaλ = w1 stop, w1 is accpeted

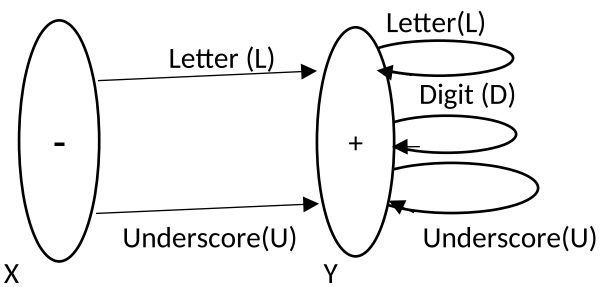
Note-1. In left-most-derivation you ***must*** only do one substitution for each non-terminal from left-to-right.
For example:
A→ aB , you can replace B with bC
⟹abC, with only one substitution

But, use both B→aC and C→aC to get
A→aB
⟹abaC
is illegal, two substitutions at once!
Note-2:
In left-most-derivation, only the first line is the actual given grammar and we use A→ aB, using single arrow, after that is all substitutions and we must use A ⟹ abC, arrow with two lines ⟹
ii)Trace w2=abaab
A→ aB
⟹ abC
⟹abaC
⟹abaaC
⟹abaa(does not generate b)

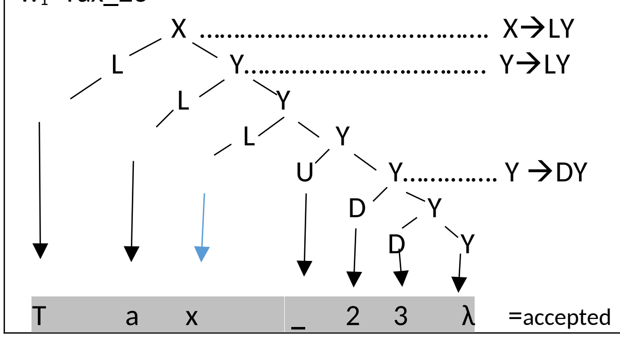
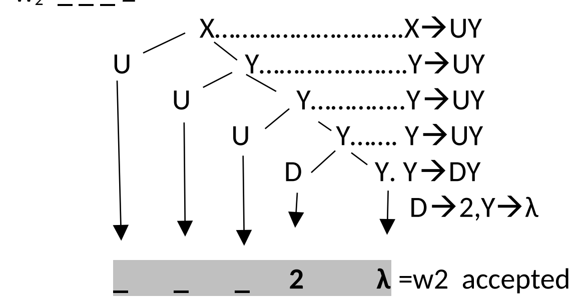
CFG rejects w2
</td></tr>
</table>

Now, lets see the applications of what we have covered so far.

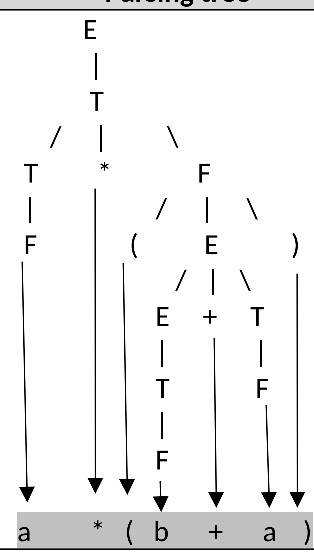
Example. An Identifiers in C++ could be string of letters, digits, or underscores. Identifiers must begin with a letter or underscore. Construct a grammar for identifiers in C++

FA representing the grammar of identifiers in C++	**CFG**
 Must begin with a letter of underscore, followed by more letters, digits, or underscores	X →LY | UY Y→L Y| DY |UY | λ D→0|1|2|….|9, digits are 0,1,…,9 U→_ ,underscore is _ L → 'A' |'B'| ….|'Z'|'a'|'b'|…..|'z' Letters are 'A'..'Z', 'a'..'z' Terminals = {a..z, A..Z, 0..1, _} Non-terminals={ X, U, L, D, Y }

Example: Trace the above CFG to determine whether each identifier w_i is accepted or not:

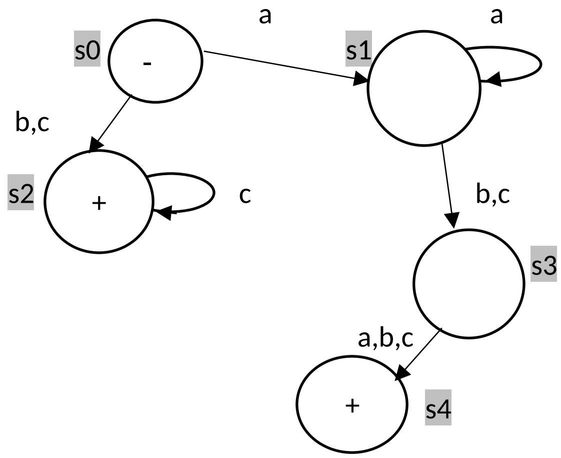
w_1=Tax_23 	w_2=_ _ _ 2 

Example. Use the given CFG. Trace the grammar to determine whether w=a*(b+a) is accepted or not.

CFG	Parsing tree	Left-most-derivation
E→E+T E→ E- T E→ T T→T*F T→T/F T→ F F→(E) F→ a F→ b		E→ T ⟹T * F ,T→T*F ⟹F * F ,T→ F ⟹a * ,F→a ⟹ a * (E) ,F→(E) ⟹ a * (E + T) ,E→E+T ⟹a * (T + T) ,E→T ⟹ a * (F + T) ,T→F ⟹ a * (b + T) ,F→b ⟹a * (b + F) ,T→F ⟹a * (b + a) ,F→a Note: ONLY one substitution at each step

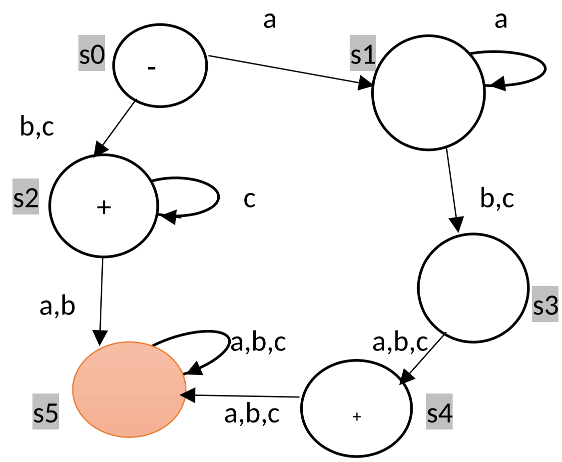
Example. Given the following FA with Σ={a, b, c}. We want to write a program to determine whether a given word w=abbcc$ ($ marks the end of string) is accepted or rejected by an FA.

1. Given the following FA

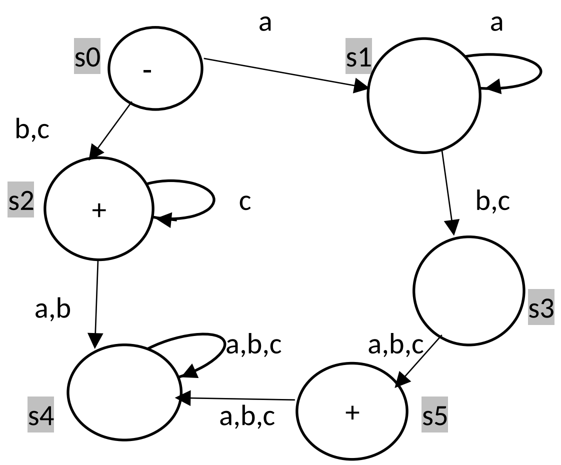


First, we must complete the FA, means at each state what will be happened if the input is a, b, or c. If an input letter it is not defined, we will add a new NULL state, and dump that input into the NULL state.

2. Complete the given FA. Add a new state s5, and dump any input a, b, or c which are not defined at existing states into the new state s5. Now, since s5 is part of this FA, complete that state as well by a loop with labels a ,b, and c.



3.Construct a transition table for this new FA



Input / States	a=0	b =1	c = 2
0	1	2	2
1	1	3	3
2	4	4	2
3	5	5	5
4	4	4	4
5	4	4	4

Initial state=s0 or just 0

Final states=s2 and s5 or just 2 , 5

At state o, input a takes us to state 1, input b to state 2 and input c to also state 2, The same for other states.

introduce the transition table to your program using a 2-dimensional array: Table[6][3]

```
int  Table[6][3]={ {1,2,2}, { 1,3,3},{4,4,2},
                        {5,5,5},{4,4,4},{4,4,4} };
int state=0, i=0, col=0; // start at state 0
string w="abbcc$";//input sting to test
while ( i < w.length() )
{    switch( w[i] )//find col for each input
    {   case 'a' : col=0; break;
        case 'b' : col=1; break;
        case '$':   if (state==2||state==5)
                            cout<<w<<" is accepted\n";
                    else
                            cout<<w<<" is rejected\n";
                    break;
    }//end of switch
    state= Table[state][col]; //go to the next state
    ++i; //go to the next input letter
```

case ‘c’ : col=2; break;	}// end of while loop

Trace the program segment for input: abbcc$

Transition table

Input State(s)	a=0	b =1	c = 2
0	1	2	2
1	1	3	3
2	4	4	2
3	5	5	5
4	4	4	4
5	4	4	4

C++ code

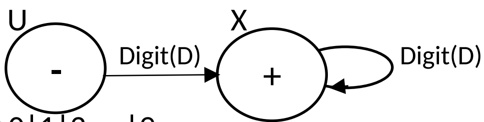
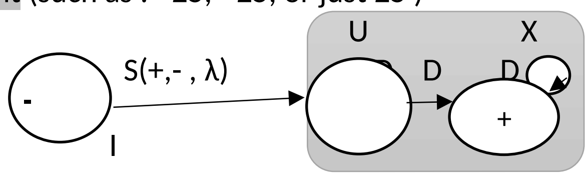
```
int  Table[6][3]={ {1,2,2},
 { 1,3,3},{4,4,2},   {5,5,5},
 { 4,4,4}, {4,4,4} };
int state=0, i=0, col=0;
// start at state 0
string w=”abbcc$”;

while ( i < w.length() )
{ switch( w[i] )
  { case ‘a’ : col=0; break;
     case ‘b’ : col=1; break;
    case ‘c’ : col=2; break;
    case ‘$’:
       if (state==2||state==5)
             cout<<w<<” accepted”;
       else
              cout<<w<<” rejected”;
       break;
  }//end of switch
      //go to the next state
      state= Table[state][col];
       //go to the next input letter
      ++i;
}// end of while
```

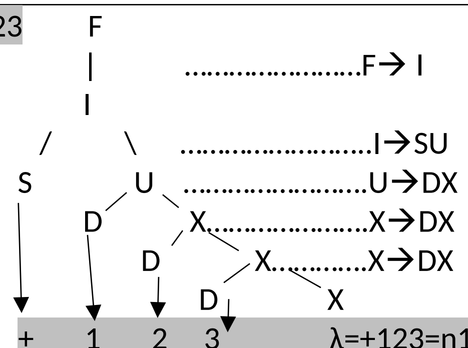
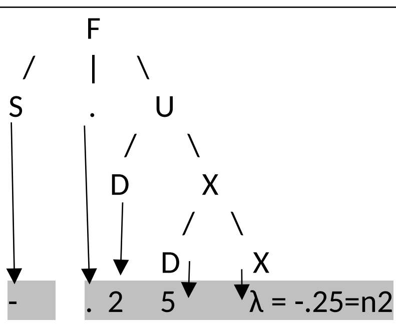
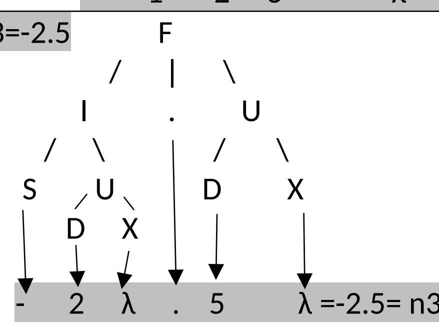
Tracing the C++ codes for w=abbcc$

state	W[i]	col	[state][col]
0	**a**	**0**	**[0][0]=1**
1	**b**	**1**	**[1][1]=1**
1	**b**	**1**	**[1][1]=1**
1	**c**	**2**	**[1][2]=3**
3	**c**	**2**	**[3][2]=5**
5	**$**	**State=5, the Input is accepted**	

Example. Write a CFG for numbers of types **int** and **float** in C++

<table>
<tr><th>FA</th><th>CFG</th></tr>
<tr><td>Unsigned int (only positive integers with no sign, such as 123)

D→0|1|2,….|9</td><td>U→D X ; one digit
X→DX | λ ;or more digits
D→0|1|2|3|…..|9</td></tr>
<tr><td>int (such as : +25, -25, or just 25)
</td><td>I→ SU ; sign or no sign followed by
U→DX ;unsigned digit
X→DX | λ;or more digits
D→0|1|2|……..|9
S→+|-|λ</td></tr>
<tr><td colspan="2">float (such as: +25., -25., 25.3, +.123)
Integer followed by .
25. +25. -25.
Integer . unsigned integer
+25.13, -25.13, 25.13
F→ I. | I. U | I | S.U
Signed . unsigned integer
+.25, -.25, .25
I→SU; U→DX; X→DX|λ;
D→ 0|1|2|3|………|9
S→+| - | λ
Integer numbers are also considering as floating numbers</td></tr>
</table>

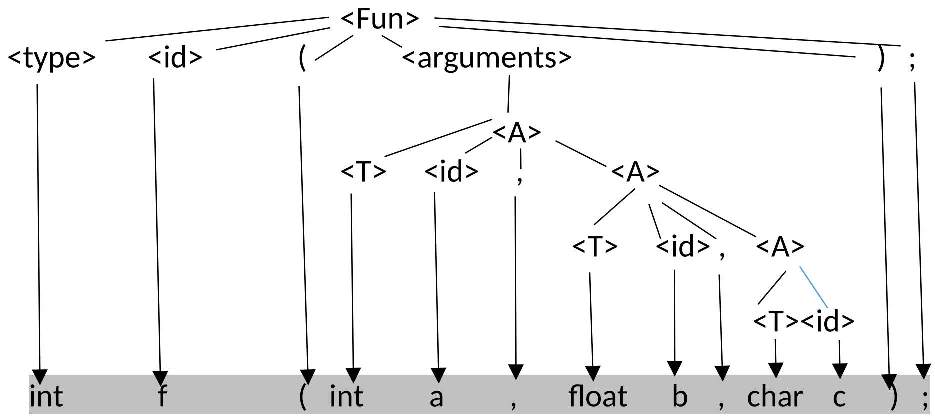
Example. Trace the above CFG for n1=+123, n2= -.02, n3= -2.5

<table>
<tr><td>CFG: F is the initial state
F→ I. | I. U | S.U | I | U
I→SU
U→DX
X→DX | λ
D→0| 1| 2|…………|9
S→ + | - | λ</td><td>n1=+123
</td></tr>
<tr><td>n2 = -.25
</td><td>n3=-2.5
</td></tr>
</table>

Example. Write a CFG for simple function’s prototypes in C++

<table>
<tr><th>Examples of function’s prototype in C++</th><th>CFG for function’s prototype in C++</th></tr>
<tr><td>a. void f( )
b. void f(int a, int b, int c);
c. int f(int a, int b int c);
) at the end
List of 0 or more arguments(A)
(left parenthesis
Identifier(ID)
Type(T)</td><td><Fun>→<T> <id> (<arguments>) ;
<T>→void| int | char|…….
<id>→few examples before
<arguments> →<A>
<A> →<T><id>, <A> | <T><id> | λ</td></tr>
</table>

Example. Trace the following function’s prototype: int f(int a, float b, char c);

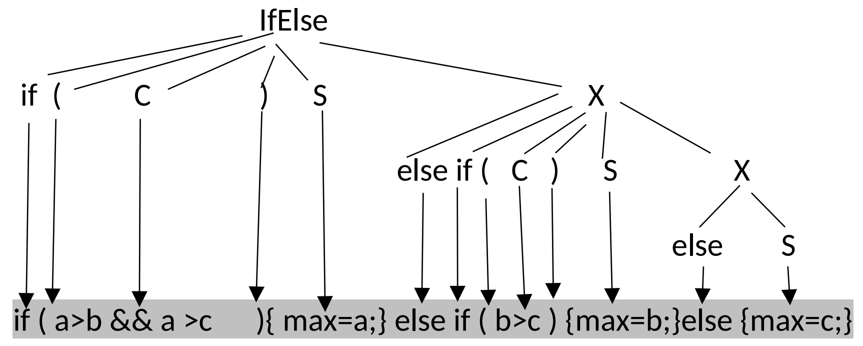


Example. Write a CFG for if-else statements in C++

<table>
<tr><th>Examples of if-else statement in C++</th><th>CFG for if-else statements in c++</th></tr>
<tr><td>i. if(a)
cout<<a;
ii. if(a> b)
{
Max=a;
cout<<Max<<endl;
}
else
{
<statements>
}
iii. if(<condition>)
{ <statements> }
else if (<condition>)
{ <statements> }
else if (<condition>)
{ <statement> }
else
{ <statements> }</td><td><if-else>→if (<condition>) {<statements> }
→if(<condition>){statements>}else{<statements>}
→if(<condition>){ <statements> }
else if (<condition>) { <statement> }
else if (<condition>){ <statement> }
else {<statements> }
or
IfElse →if (C) SX
X → λ | else S | else if(C) SX
C → condition
S → statement</td></tr>
</table>

Example. Trace the following :

```
if( a>b && a > c )
{          max= a;    }
else if (b > c )
{          max= b;    }
else
{         max= c;     }
```



(CFG of some statements in C++)

50 points

1. (8 points) Consider the following grammar:
   S→aSbB | A | c
   A→cA | c
   B→ d | A
   Trace the grammar to determine which of the following words are accepted or rejected?

   i. accbc (use parse tree) ii. acccdd (use left-most-derivation)

2. (8 points) Given the following CFG:
   S→ I = E
   E→E+T | E-T | T
   T→T*F | T/F | F
   F→(E) | I
   I → a | b
   Use parsing tree to trace the grammar and decide which of the following statements are accepted or rejected.
   i. a=a*(b - a*a) ii. b=a*b – b*(a + b)

3. (8 points) Find the language of the following grammars:

a. S→aS | bB | aA |λ B→ bB | aA A→aA | bA | λ	b. S→aS | bA |λ A→aA | bX |λ X→aX | bX |λ

4. (9 points) Find a CFG for each of the following languages
   (i) L=a* + b* (ii) L=a*b*c* (iii) L=ab* + ba*+c

Programming assignment

1. (17 points) Write a program to find the value of a postfix expression. Variables are one or more characters each. We might have some integer numbers as part of the expression.(Hint. Read each part of the expression as a token of type string, if the first character of the token is a letter that indicates the token is a variable name, push its value in stack. If the token is an integer number use the predefined function **stoi (member of <string> library)** to return its numeric value of the token.

Sample I/O:

Enter a postfix expression with a $ at the end: 20 jerry 45 + tom - * $ Enter the value of jerry: 10 Enter the value of tom: 5 Expression's value is 1000 CONTINUE(y/n)? y	Enter a postfix expression with a $ at the end: myscore yourscore 45 + 100 + * $ Enter the value of myscore: 3 Enter the value of yourscore: 5 Expressions value is 450 CONTINUE(y/n)? n

As input, please use the given expressions and the given values.

2. Given the language L=a*c*b(a+c)* with alphabet={a,b,c}. (1) Construct an FA to accept L. (2) complete the FA (3) construct a transition table for the new FA. (4) Write a program to decide whether the following words are members of L or not

$W1 = bac$ $W2 = aa\ b\ a^3$ $W3 = baaccb$