1. (10) Given the grammar below, identify which sentences are in the language (which are valid sentence).
   1. Baab

derivations

<A> -> b

<B> -> a

Strings as sentences baab is derived by grammar so baab is valid sentence

* 1. Bbbab

<A> -> <A> b

<A> -> <A>b

<A> -> b

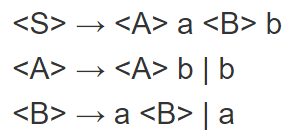
Because of <B> String is not generated by grammar so invalid sentences.

* 1. Bbaaaaaa

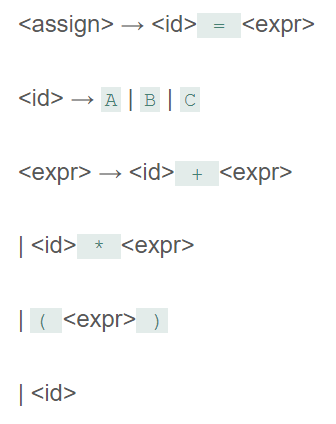
Because of this b. String is not generated by grammar. So invalid

* 1. Bbaab

Bbaab is derived by grammar so valid sentence.



1. (10) Identify all of the tokens (categories of lexemes) in the grammar below, and which lexemes they categorize. Put them in a table.



Tokens

Identifiers: {<assign>, <id>, Expr>}

Operations: {=, +, \*}

Separtators { ->, |, (,)}

1. (10) Given the grammar from question 2, show a left-most derivation and draw the parse tree for the following statement.
   1. B = B + (C + (A \* A) )

Left most derivation: parse tree

<assign> <assign>

<id> = <expr> <id> = <expr>

B = <expr> <id> + <expr>

B = <id> + <expr> (<expr>)

B = B + <expr> <id> ++ <Expr>

B = B + (<expr>) (<expr>)

B = B + (<id> + <expr>) <id> \* <expr>

B = B + ( c + <expr>) <id>

B = B + (c + (<expr>))

B = B + (c (<id> \* <expr>))

B = B + (c + (a \* <expr>))

B = B + (c+ (a \* a)) B = B + ( C + ( A \* A))

1. (10) Remove all of the recursion from the following grammar:

S -> Aa | Bb

A -> Aa | AbC | C

B -> S | bb

C -> c

The recursion is A -> Aa | AbC | C

This is can be written as:

A -> Aa | AbC  
A -> C

So the grammar becomes:

S -> Aa | Bb  
A -> Aa | AbC  
A -> C  
B -> S | bb  
C -> c

The remaining recursion is:

A -> Aa | AbC  
i.e. A -> A (a | bC)  
i.e. A -> AA' and A' -> a | bC

Hence, the non-recursive form of the grammar is:

S -> Aa | Bb  
A -> AA'  
A' -> a | bC  
A -> C  
B -> S | bb  
C -> c

1. (10) Use left factoring to resolve the pairwise disjointness problems in the following grammar:

A -> aBc | ac | a

B -> b | aB

In A -> aBc/ ac/ a -> a is common so

A =->aA’

A’ -> Bc /c /epsilon

Therfore grammar will be

A -> aA’

A’ -> Bc/ c/ epsilon

B ->b/ ab

1. (20 pts) Create an LR(0) parse table for the following grammar. Show all steps (creating closures, the DFA, the transition table, and finally the parse table):

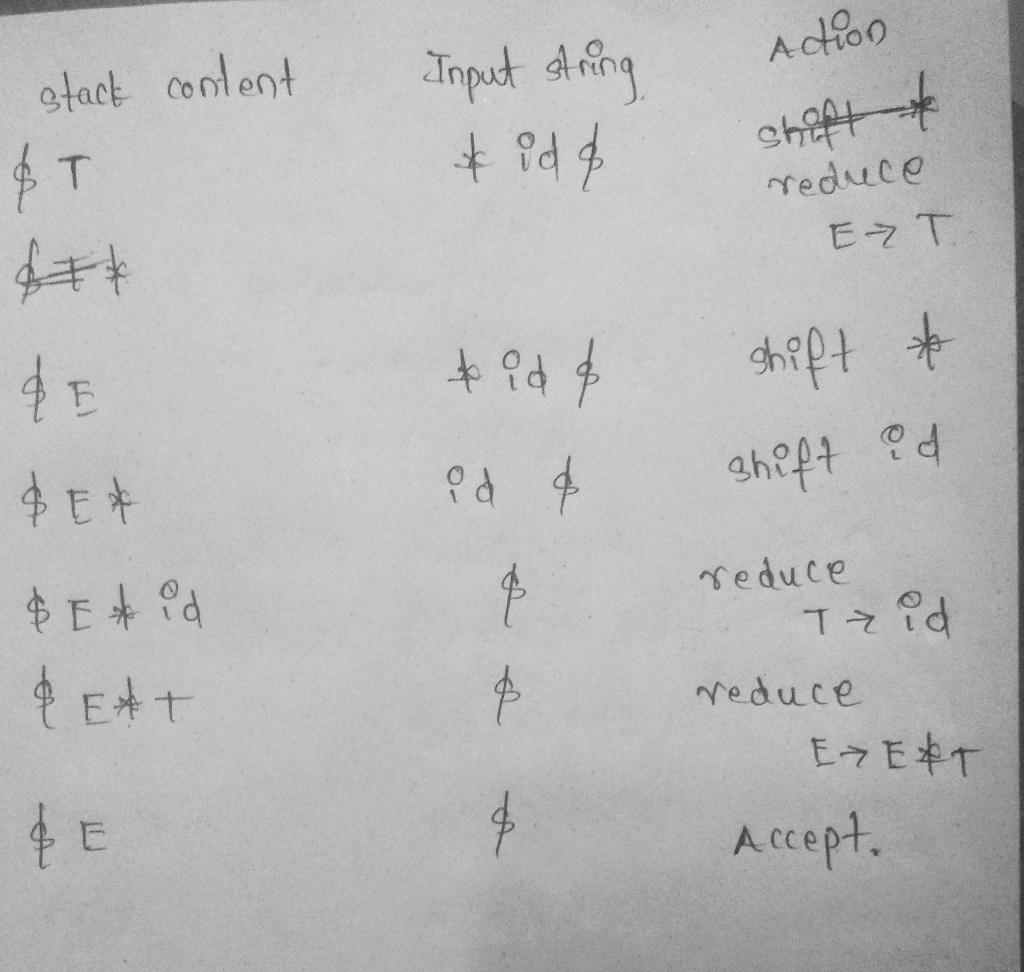
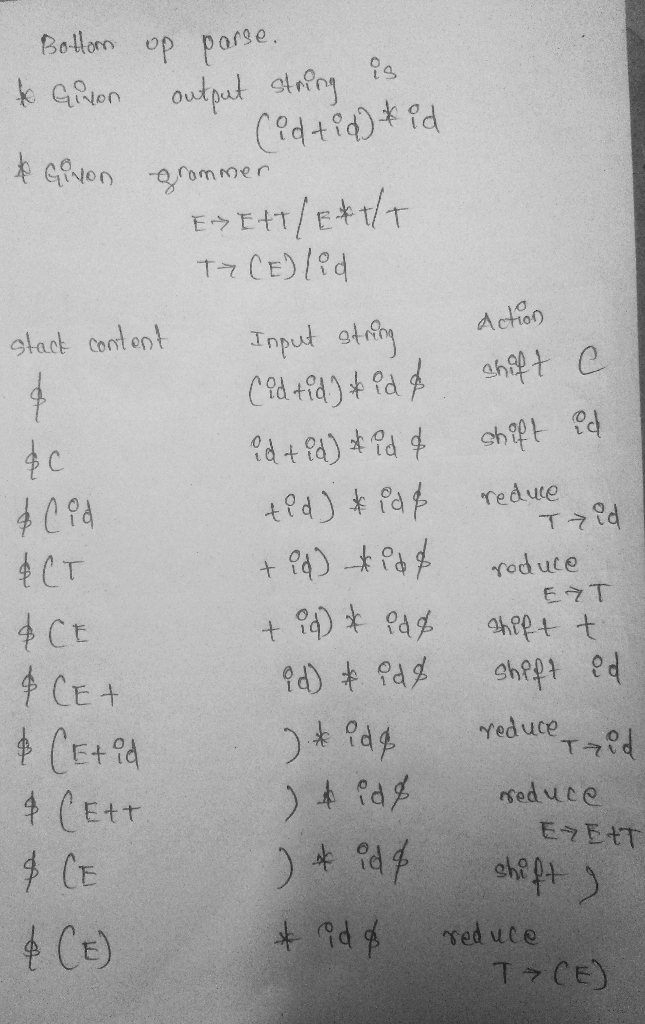
E -> E + T | E \* T | T

T -> ( E ) | id

Commor
*
Given grammer is
EXEXT/EXT/T
T> CE) lid
Oure aum is to construct LR(o) parse
table for the guten grammer,
for constrE>EXT/E*T/T
& These numbering
e These no
Tz (E)/ id
I are used for
reduce steps in
the parse table ?I 5
lo
€ > EXT.
E Et.T
TY.CE)
T>.id
I 6
E*.T
E - E
(E.
EE
EXE.IT
EYEXT EE.
E.E&TI
Ez.T NE
(E- I.
I
->Ez E*T.)
-
Тя
KT.
Ilo
13Go to
RELI
action
t* coid
53
55 56
To
I
Accept
14
I4
85 85
Is
I6

1. (20 pts) Show a complete bottom-up parse, including the parse stack contents, input string, and action for the string below using the parse table you created in step 6. Think about how I went through this in class.

(id + id) \* id



1. (10 pts) Show a rightmost derivation for the string above, and show how the bottom-up parse you completed in step 7 correctly finds all of the handles for the input string above.

