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

<S> -> <A> a <B> b. <A> -> b

b a <B> b <B> -> a

b a a b

therefore is valid

* 1. bbbab

<A> a <B> b <A> -> <A> b

<A> b a <B> b <A> -> <A> b

<A> b b a <B> b <A> -> b

b b b a <B> b

bbbab cannot be generated from here and is therefore invalid

* 1. bbaaaaaa

<A> a <B> b

<A> b a <b> b <A> -> <A> b

b b a <b> b <A> -> b

there is no way to get rid of that last b so invalid

* 1. bbaab

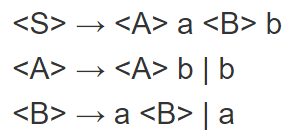
<A> a <B> b

<A> b a <B> b <A> = <A> b

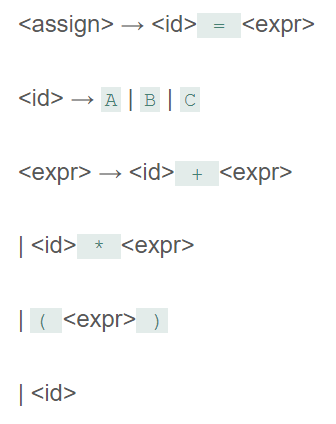
b b a <B> b <A> = b

b b a a b <B> = a

bbaab is generated and is thus valid



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



|  |  |
| --- | --- |
| Token | Lexeme |
| Identity | A, B, C |
| Add\_op | + |
| Mult\_op | \* |
| Open\_paren | ( |
| Close\_paren | ) |
| Equal\_op | = |

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) )

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

B = <expr>

B = <id> + <expr>

B = B + <expr>

B = B + (<expr>)

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

B = B + (C + <expr>)

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

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

B = B + (C + (A \* <expr>))

B = B + (C + (A \* <id>))

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

Diagram

Description automatically generated

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

S -> Aa | Bb

A -> Aa | AbC | C

B -> S | bb

C -> c

S-> Aa | Bb

A-> CA’

A’ -> aA’ | bCA’ | epsilon

B -> S | bb

C -> c

S-> Aa | Bb

A-> CA’

A’ -> aA’ | bCA’ | epsilon

B -> Aa | Bb | bb

C -> c

S-> Aa | Bb

A-> CA’

A’ -> aA’ | bCA’ | epsilon

B-> AaB’ | bbB’

B’-> bB’ | epsilon

C -> c

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

A -> aBc | ac | a

B -> b | aB

A -> aA’

A’ -> Bc | c | epsilon

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

Diagram

Description automatically generatedLetter, calendar

Description automatically generated

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

input : .(id + id) \* id

stack :0

input : (.id + id) \* id

stack: 0 ( 8

input: (id .+ id) \* id

stack: 0 ( 8 id

input: (id. + id) \* id

stack: 0 ( 8 T

output: 5

input: (id. + id) \* id

stack: 0 ( 8 E 9

output: 4

input: (id +. id) \* id

stack: 0 ( 8 E 9 + 4

input : (id + id.) \* id

stack: 0 (8 E 9 + 4 id 5)

input : (id + id.) \* id

stack: 0 (8 E 9 + 4 T 7)

output: 5

input: (id + id.) \* id

stack: 0 (8 E 9

output: 1

input: (id + id). \* id

stack: 0 (8 E 9 ) 10

input: (id + id). \* id

stack: T 8

output: 4

input: (id + id). \* id

stack: E 1

output: 3

input: (id + id) \*. id

stack: E 1 \* 5

input (id + id) \* id.

stack: E 1 \* 5 id 3

input (id + id) \* id.

stack: E 1 \* 6 T 3

output: 5

input (id + id) \* id.

stack: E1

output: 2

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.

A picture containing text, whiteboard, drawing

Description automatically generated