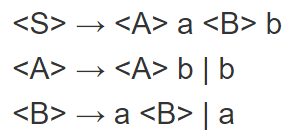
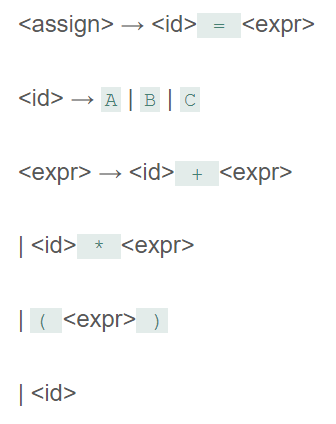
1. (10) Given the grammar below, identify which sentences are in the language (which are valid sentence).
   1. baab
   2. bbbab
   3. bbaaaaaa
   4. bbaab



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 | Lexemes |
| var | A,B,C |
| add\_op | + |
| equal\_op | = |
| multi\_op | \* |
| left\_par | ( |
| right\_par | ) |

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))

A close up of a whiteboard

Description automatically generated

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

S -> Aa | Bb

A -> Aa | AbC | C | Sb

B -> bb

C -> c

S -> Aa | Bb

A -> CA’ | SbA’

A’ -> aA’| bCA’ | ε

B -> 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

A -> ε |c C’

B -> b | aB

C’ -> ε|B

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

r0 S’ -> E$

r1 E -> E + T

r2 E -> E \* T

r3 E -> T

r4 T -> ( E )

r5 T -> id

A picture containing map, text, sky

Description automatically generated

Transition Table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Item | + | \* | ( | ) | id | E | T |
| 0 |  |  | 8 |  | 1 | 2 | 3 |
| 1 |  |  |  |  |  |  |  |
| 2 | 4 | 5 |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |
| 4 |  |  |  |  | 1 |  | 7 |
| 5 |  |  | 8 |  | 1 |  | 6 |
| 6 |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |
| 8 |  |  | 8 |  | 1 | 9 | 3 |
| 9 | 4 | 5 |  | 10 |  |  |  |
| 10 |  |  |  |  |  |  |  |

Parser Table

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| State | Action | | | | | | Goto | |
|  | + | \* | id | ( | ) | $ | E | T |
| 0 |  |  | S1 | S8 |  |  | 2 | 3 |
| 1 | R5 | R5 | R5 | R5 | R5 | R5 |  |  |
| 2 | S4 | S5 |  |  |  | acc |  |  |
| 3 | R3 | R3 | R3 | R3 | R3 | R3 |  |  |
| 4 |  |  | S1 |  |  |  |  | 7 |
| 5 |  |  | S1 | S8 |  |  |  | 6 |
| 6 | R2 | R2 | R2 | R2 | R2 | R2 |  |  |
| 7 | R1 | R1 | R1 | R1 | R1 |  |  |  |
| 8 |  |  | S1 | S8 |  |  | 9 | 3 |
| 9 | S4 | S5 |  |  | S10 |  |  |  |
| 10 | R4 | R4 | R4 | R4 | R4 | R4 |  |  |

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 | stack | output |
| .( id + id) \* id$ | 0 |  |
| (. id + id) \* id$ | 0 ( 8 |  |
| ( id. + id) \* id$ | 0 ( 8 id 1 |  |
| ( id. + id) \* id$ | 0 ( 8 T | 5 |
| ( id. + id) \* id$ | 0 ( 8 T 3 | 5 |
| ( id. + id) \* id$ | 0 ( 8 E | 5, 3 |
| ( id. + id) \* id$ | 0 ( 8 E 9 | 5, 3 |
| ( id +. id) \* id$ | 0 ( 8 E 9 + | 5, 3 |
| ( id +. id) \* id$ | 0 ( 8 E 9 + 4 | 5, 3 |
| ( id + id.) \* id$ | 0 ( 8 E 9 + 4 id | 5, 3 |
| ( id + id.) \* id$ | 0 ( 8 E 9 + 4 id 1 | 5, 3 |
| ( id + id.) \* id$ | 0 ( 8 E 9 + 4 T | 5, 3, 5 |
| ( id + id.) \* id$ | 0 ( 8 E 9 + 4 T 7 | 5, 3, 5 |
| ( id + id.) \* id$ | 0 ( 8 E | 5, 3, 5, 1 |
| ( id + id.) \* id$ | 0 ( 8 E 9 | 5, 3, 5, 1 |
| ( id + id). \* id$ | 0 ( 8 E 9 ) | 5, 3, 5, 1 |
| ( id + id). \* id$ | 0 ( 8 E 9 ) 10 | 5, 3, 5, 1 |
| ( id + id). \* id$ | 0 T | 5, 3, 5, 1, 4 |
| ( id + id). \* id$ | 0 T 3 | 5, 3, 5, 1, 4 |
| ( id + id). \* id$ | 0 E | 5, 3, 5, 1, 4, 3 |
| ( id + id). \* id$ | 0 E 2 | 5, 3, 5, 1, 4, 3 |
| ( id + id) \*.id$ | 0 E 2 \* | 5, 3, 5, 1, 4, 3 |
| ( id + id) \*.id$ | 0 E 2 \* 5 | 5, 3, 5, 1, 4, 3 |
| ( id + id) \* id.$ | 0 E 2 \* 5 id | 5, 3, 5, 1, 4, 3 |
| ( id + id) \* id.$ | 0 E 2 \* 5 id 1 | 5, 3, 5, 1, 4, 3 |
| ( id + id) \* id.$ | 0 E 2 \* 5 T | 5, 3, 5, 1, 4, 3, 5 |
| ( id + id) \* id.$ | 0 E 2 \* 5 T 6 | 5, 3, 5, 1, 4, 3, 5 |
| ( id + id) \* id.$ | 0 E | 5, 3, 5, 1, 4, 3, 5, 2 |
| ( id + id) \* id.$ | 0 E 2 | 5, 3, 5, 1, 4, 3, 5, 2 |
| ( id + id) \* id$. | 0 E 2 $ | 5, 3, 5, 1, 4, 3, 5, 2 |
| ( id + id) \* id$. | accept | 5, 3, 5, 1, 4, 3, 5, 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.

|  |  |
| --- | --- |
| Step for rightmost derivation | Rule from bottom-up |
| E | R0 |
| E \* T | R2 |
| E \* id | R5 |
| T \* id | R3 |
| (E) \* id | R4 |
| (E + T) \* id | R1 |
| (E + id) \* id | R5 |
| (T + id) \* id | R3 |
| (id + id) \* id | R5 |