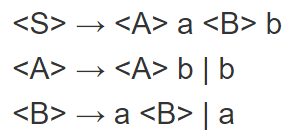
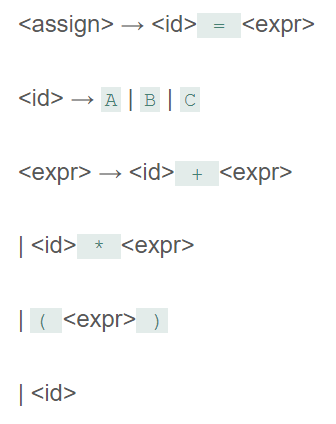
1. (10) Given the grammar below, identify which sentences are in the language (which are valid sentence).
   1. baab – Valid sentence
   2. bbbab – NOT valid sentence
   3. bbaaaaaa – NOT valid sentence
   4. bbaab – Valid sentence



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 | Lexemes |
| <id> | A, B, C |
| <equalOp> | = |
| <addOp> | + |
| <multOp> | \* |
| <lParen> | ( |
| <rParen> | ) |

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 + (<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))

Parse tree:

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

S -> Aa | Bb

A -> Aa | AbC | C

B -> S | bb

C -> c

S -> AaS’ | bbbS’

S -> bS’ | epsilon

A -> CA’

A’ -> aA’ | bCA’ | 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 -> aC

B -> b | aB

C -> 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

Rules:

R0: S’ -> E$

R1: E -> E + T

R2: E -> E \* T

R3: E -> T

R4: T -> (E)

R5: T -> id

Closures:

Transition Table:

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

Parse Table:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Action** | | | | | | **Go To** | |
|  | id | \* | + | ( | ) | $ | E | T |
| **0** | S2 |  |  | S4 |  |  | 1 | 3 |
| **1** |  | S5 | S6 |  |  | acc |  |  |
| **2** | R5 | R5 | R5 | R5 | R5 |  |  |  |
| **3** | R3 | R3 | R3 | R3 | R3 |  |  |  |
| **4** | S2 |  |  | S4 |  |  | 9 | 3 |
| **5** | S2 |  |  | S4 |  |  |  | 8 |
| **6** | S2 |  |  | S4 |  |  |  | 7 |
| **7** | R1 | R1 | R1 | R1 | R1 |  |  |  |
| **8** | R2 | R2 | R2 | R2 | R2 |  |  |  |
| **9** |  | S5 | S6 |  | S10 |  |  |  |
| **10** | 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

|  |  |  |
| --- | --- | --- |
| **STACK** | **INPUT** | **ACTION** |
| 0 | .(id + id) \* id $ | Shift 4 |
| 0 ( 4 | (.id + id) \* id $ | Shift 2 |
| 0 ( 4 id 2 | (id. + id) \* id $ | Reduce by T🡪id (R5) |
| 0 ( 4 T 3 | (id. + id) \* id $ | Reduce by E🡪T (R3) |
| 0 ( 4 E 9 | (id. + id) \* id $ | Shift 6 |
| 0 ( 4 E 9 + 6 | (id +. id) \* id $ | Shift 2 |
| 0 ( 4 E 9 + 6 id 2 | (id + id.) \* id $ | Reduce by T🡪id (R5) |
| 0 ( 4 E 9 + 6 T | (id + id.) \* id $ | Reduce by E🡪E + T (R1) |
| 0 ( 4 E 9 | (id + id.) \* id $ | Shift 10 |
| 0 ( 4 E 9 ) 10 | (id + id). \* id $ | Reduce by T🡪(E) (R4) |
| 0 T 3 | (id + id). \* id $ | Reduce by E🡪T (R3) |
| 0 E 1 | (id + id). \* id $ | Shift 5 |
| 0 E 1 \* 5 | (id + id) \*. id $ | Shift 2 |
| 0 E 1 \* 5 id 2 | (id + id) \* id. $ | Reduce by T🡪id (R5) |
| 0 E 1 \* 5 T 8 | (id + id) \* id. $ | Reduce by E🡪E \* T (R2) |
| 0 E 1 | (id + id) \* id. $ | Accept |
| 0 E 1 | (id + id) \* id $. | --------- |

Output: 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.