1.) Given grammar: S → <A>a<B>b
<A> → <A>b | b
<B> → a<B> | a

a.) baab

S → <A>a<B>b
→ ba<B>b
→ baab

Valid Sentence

b.) bbbab

S → <A>a<B>b
→ <A>ba<B>b X won't work bc of <B>
S → <A>a<B>b
→ ba<B>b X Won't work bc only one 'b' @ start

Invalid

c.) bbaaaaaa

- This is invalid since the sentences in this grammar must end in a 'b'.

d.) bbaab

S → <A>a<B>b
→ <A>ba<B>b
→ bba<B> b
→ bbaab

Valid Sentence

2.) Tokens for grammar: <assign> → <id> = <expr>
<id> → A | B | C
<expr> → <id> + <expr> | <id> * <expr>
| (<expr>) | <id>

Tokens

Identifiers	A, B, C
Operators	=, +, *
Parantheses	(,)

3.) B= B + (C + (A * A))

<assign> → <id> = <expr>

<id> = <expr>

B = <expr>

B= <id> + <expr>

B= B + <expr>

B= B + (<expr>)

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

B=B+(C+<expr>)

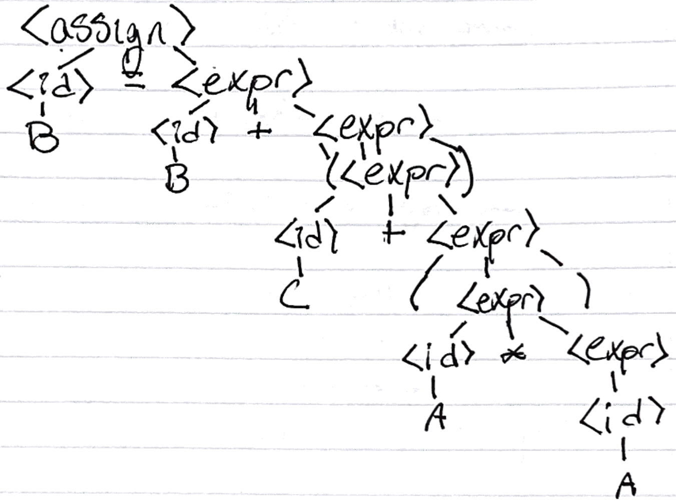
B= B+ (C +(<expr>))

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

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

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

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



4.) Given: S → Aa | Bb

A → Aa | AbC | C

B → S | bb

C → c

- Let's remove indirect recursion: S → Aa | Sb | bbb

A → Aa | AbC | C

C → c

- Now apply rules: S → bS' | ε

S' → AaS' | bbbS'

A → aA' | bCA' | ε

A' → CA'

C → c

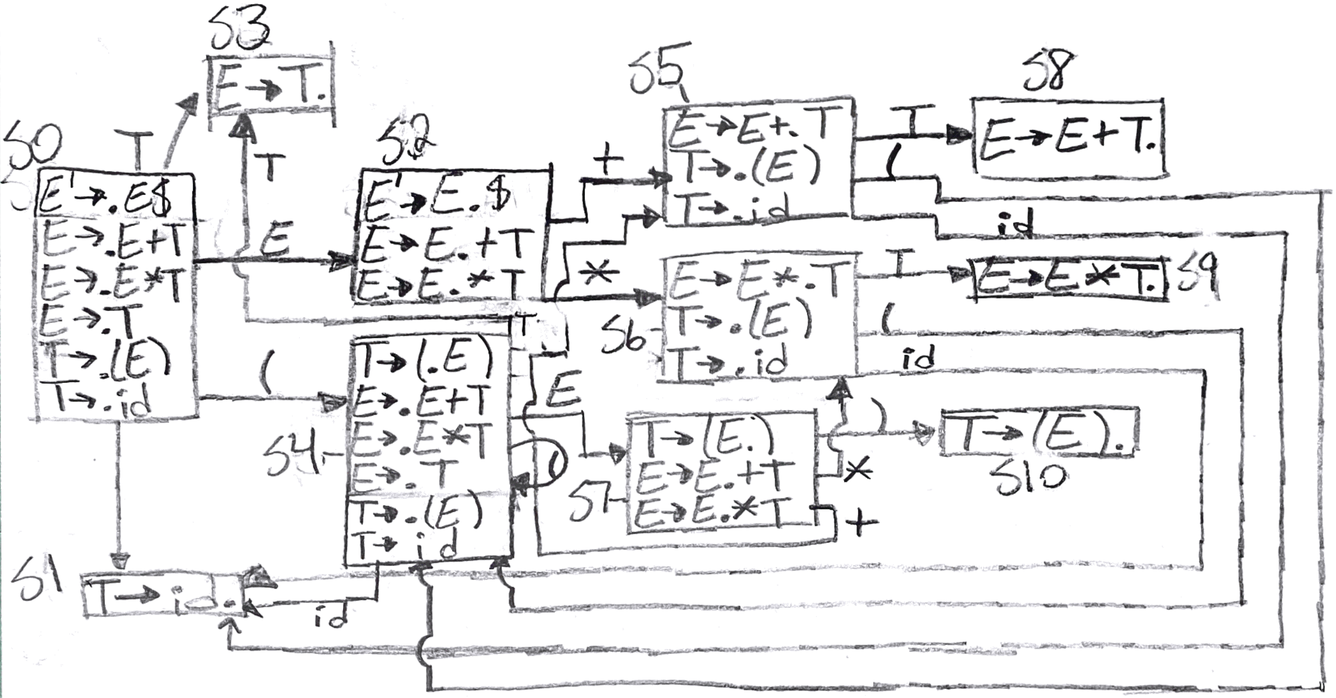
5.) A → aBc | ac | a   B → b | aB

A → a <new> c   B → a <new>

<new> → ε   <new> → ε

<new> → B   <new> → B

6) r1: E→E+T
r2: E→E*T
r3: E→T
r4: T→(E)
r5: T→id



6.) Transition Table

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

6) Parse Table

	Action					GoTo	
	+	*	(	)	id	E	T
0			s4		s1	2	3
1	r5	r5	r5	r5	r5		
2	s5	s6					
3	r3	r3	r3	r3	r3		
4			s4		s1	7	3
5			s4		s1		8
6			s4		s1		9
7	s5	s6		s10			
8	r1	r1	r1	r1	r1		
9	r2	r2	r2	r2	r2		
10	r4	r4	r4	r4	r4		

r1: E→E+T

r2: E→E*T

r3: E→T

r4: T→(E)

r5: T→id

* If given a '$' in State 2, then accept

7.) Botton-Up parse for: (id + id)*id

r1: E→ E+T

r2: E→ E*T

r3: E→ T

r4: T→ (E)

r5: T→ id

Start

input: .(id + id)*id $

Stack: 0

Output:

input: (.id + id)*id $

Stack: 0(4

Output:

input: (id. + id)*id $

Stack: 0(4 id

Output:

input: (id. + id)*id $

Stack: 0(4 id 1

Output:

input: (id. + id)*id $

Stack: 0(4T

Output: 5,

input: (id. + id)*id $

Stack: 0(4T3

Output: 5,

input: (id. + id)*id $

Stack: 0(4E

Output: 5,3,

input: (id. + id)*id $

Stack: 0(4E7

Output: 5,3,

input: (id +. id)*id $

Stack: 0(4E7+

Output: 5,3,

input: (id +. id)*id $

Stack: 0(4E7+5

Output: 5,3

input: (id + id.)*id $

Stack: 0(4E7+5 id

output: 5,3

input: (id + id.)*id $

Stack: 0(4E7+5 id 1

output: 5,3

input: (id + id.)*id $

Stack: 0(4E7+5T

output: 5,3,5

input: (id + id.)*id $

Stack: 0(4E7+5T8

output: 5,3,5

input: (id + id.)*id $

Stack: 0(4E

output: 5,3,5,1

7.) Cont.

input: (id + id.) * id $
Stack: 0(4E7

Output: 5,3,5,1

input: (id + id). * id $
Stack: 0(4E7)

output: 5,3,5,1

input : (id + id). * id $
Stack : 0(4E7) 10

Output : 5,3.5,1

input: (id + id). * id $
Stack : 0T

Output: 5,3,5,4,4

input : (id + id). * id $
Stack : 0T3

Output: 5,3,5,1,4

input: (id + id). * id $
Stack: 0E

Output: 5,3,5,1,4,3

input : (id + id). * id $
Stack: 0E2

output: 5.3,5,1,4,3

input : (id + id) *. id $
Stack: 0E2*

output: 5,3,5,1,4,3

input: (id + id) *. id $
Stack: 0E2*6

output : 5,3,5,1,4,3

input : (id + id) * id. $
Stack : 0E2*6 id

Output: 5,3,5,1,4,3

input : (id + id) * id. $
Stack : 0E2*6 id 1

output:

input : (id + id) * id. $
Stack : 0E2*6T

output: 5,3,5,1,4,3,5

input : (id + id) * id. $
Stack: 0E2*6T9

output: 5,3,5,1,4,3,5

input : (id+id) * id. $
Stack : 0E

output : 5,3,5,1,4,3,5,2

input : (id + id) * id $.
Stack : 0E2$

Output: 5,3,5,1,4,3,5,2

Accept

8.) Rightmost Derivation for (id + id) * id

Output: 5, 3, 5, 1, 4, 3, 5, 2

E → E * T by r2
→ E * id by r5
→ T * id by r3
→ (E) * id by r4
→ (E+T) * id by r1
→ (E+id) * id by r5
→ (T+id) * id by r3
→ (id+id) * id by r5

The output from 7.) shows an equivalent output to the way we produced the rightmost derivation, just in reverse order, which means the output <u>correctly</u> finds all the handles.