Name: R. Eland Anthony

CS403/503 Programming Languages
Spring 2021
Assignment #1

1. Problem 3 of Chapter 3 on Page 157

Answer:

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

<id> -> A | B | C

<expr> -> <expr> * <term>

| <term>

<term> -> <factor> + <term>

| <factor>

<factor> -> (<expr>)

| <id>

2. Problem 8 of Chapter 3 on Page 158

Answer:

To prove whether the following grammar is ambiguous, one may compare valid leftmost parse trees for the given grammar.If there exists two or more unique trees the reflected grammar is ambiguous;this operation extends to rightmost parse trees as well.

Two unique leftmost derivation parse trees for the grammar:

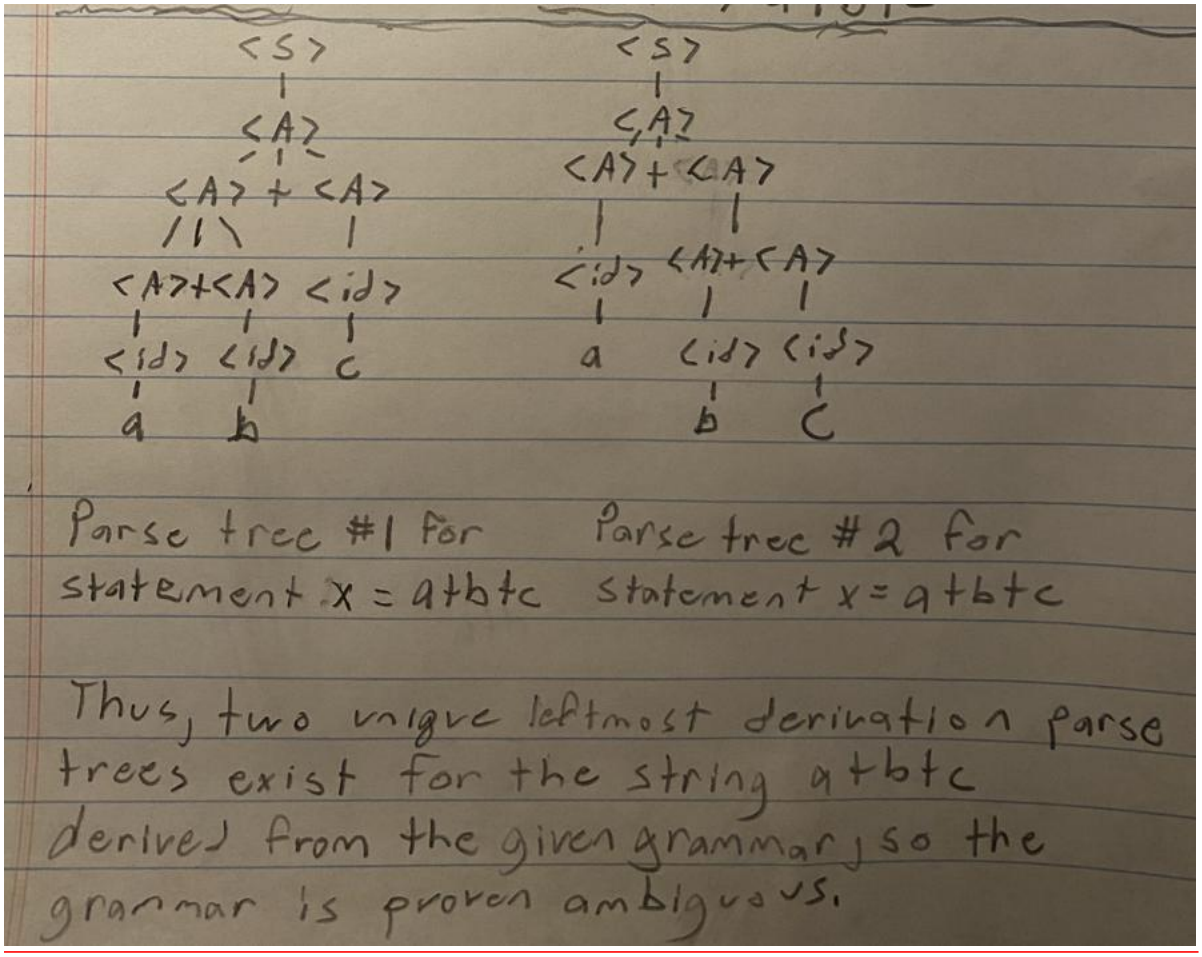
<S> -> <A>

<A> -> <A> + <A> | <id>

<id> -> a | b | c

Are illustrated in the embedded photo below:

Name: R. Eland Anthony



3. Problem 20 of Chapter 3 on Page 159

Answer:

1) syntax rule: <assign> -> <id> = <expr>

semantic rule: <expr>.expected_type <- <id>.actual_type

2) syntax rule: <id> -> A | B | C

semantic rule: <id>.actual_type <- look-up(<id>.string)

3) syntax rule: <expr> -> <id> + <expr>

Name: R. Eland Anthony

semantic rule: <expr>.actual_type <-

if (<id>.actual_type = int) and (<expr>.actual_type = int)

  then int

  else real

end if

predicate: <expr>.actual_type == <expr>.expected_type

4) syntax rule: <expr> -> <id> * <expr>

semantic rule: <expr>.actual_type <-

if (<id>.actual_type = int) and (<expr>.actual_type = int)

  then int

  else real

end if

predicate: <expr>.actual_type == <expr>.expected_type

5) syntax rule: <expr>[1] -> (<expr>[2])

semantic rule: <expr>[1].actual_type <- <expr>[2].actual_type

predicate: <expr>[1].actual_type == <expr>[2].expected_type

6) syntax rule: <expr> -> <id>

semantic rule: <expr>.actual_type <- <id>.actual_type

predicate: <expr>.actual_type == <expr>.expected_type

Name: R. Eland Anthony

4. Problem 8 of Chapter 4 on Page 194

Answer:

Stack Contents	Input String	Action
0	(id + id) * id $	Shift 5
0(4	id + id) * id $	Shift 5
0(4id5	+ id) * id $	Reduce 6(use GOTO[4, F])
0(4F3	+ id) * id $	Reduce 4(use GOTO[4, T])
0(4T2	+ id) * id $	Reduce 2(use GOTO[4, E])
0(4E8	+ id) * id $	Shift 6
0(4E+6	id) * id $	Shift 5
0(4E+6id5	) * id $	Reduce 6(use GOTO[6, F])
0(4E+6F3	) * id $	Reduce 4(use GOTO[6, T])
0(4E+6T9	) * id $	Reduce 1(use GOTO[4, E])
0(4E8	) * id $	Shift 11
0(4E8)11	* id $	Reduce 5(use GOTO[0, (E)])
0F3	* id $	Reduce 4(use GOTO[6, T])
0T2	* id $	Shift 7
0T*7	id $	Shift 5
0T*7id5	$	Reduce 6(use GOTO[7, F])
0T*7F10	$	Reduce 3(use GOTO[0, T])
0T2	$	Reduce 2(use GOTO[0, E])
0E1	$	Accept