CSE 401 - LL Semantics, Semantics, Type Checking, & Vtables

1. Edit the following Grammars to make them LL(1). Then walk through the top down parse for the string given in the parenthesis.

Grammar 1 (“azx”)

0. S ::= a B | a w
1. B ::= C x | y
2. C ::= ε | z

Grammar 2 (“ax”)

0. S ::= a B
1. B ::= C x | y
2. C ::= ε | x

Grammar 3 (“azx”)

0. S ::= S B | a | w
1. B ::= C x | y
2. C ::= ε | z

Grammar 4 (“azx”)

0. S ::= B w | a B
1. B ::= S | z x

2. Suppose we have the following global scope:

```
class Bar { boolean field; public int method(int i, int j); }
class Foo extends Bar { int val; public boolean whoop(int x); }
```

Now, consider the following hypothetical method definition for `Bar.method`:

```
public int method(int i, int j) {
  int r;
  boolean b;
  Foo o;
  if (this.field) {
    o = this;
    b = o.whoop(i + j);
    r = o.val;
  } else {
    r = i * j + 3;
  }
  return r;
}
```

a. What variables (locals, parameters, etc.) are defined in the *local* scope in the `method` body?

b. When we execute this method body, a runtime error could result. Explain how something could go wrong by giving values of the parameters and/or variables involved that would cause a runtime error.

c. The method body also has type errors. Can you describe which type check(s) the compiler could use to deduce this fact?

d. Does *every* possible execution of this method produce a runtime error? Can you describe any that happen to be statically correct? (Again, possible runtime values for parameters/variables would suffice.)

e. Suppose that we replaced the use of **`this.`**`field` in the method body to call a boolean method that always returns false. How would this change your answers to the previous questions?

Appendix—Canonical LL(1) Problems and their Solutions:

FIRST Conflict:
Both productions of A have α in their FIRST sets
0. A ::= αβ | αγ

Solution:
Factor out the prefix (α)
0. A ::= α Tail
1. Tail ::= β | γ

FIRST FOLLOW Conflict:
B is nullable, α in FIRST & FOLLOW
0. A ::= B α
1. B ::= α | ε

Solution:
Substitute B into A
0. A ::= αα | α
Factor out the prefix (α)
0. A ::= α Tail
1. Tail ::= α | ε

Left Recursion:
Special FIRST conflict: β in FIRST for both productions
0. A ::= A α | β

Solution:
Create recursive tail from suffix of recursive production
1. Tail ::= α Tail
Append Tail to non recursive productions
0. A ::= β Tail
1. Tail ::= α Tail
Add empty string (ε) as a rhs for the tail production
0. A ::= β Tail
1. Tail ::= α Tail | ε

Indirect Left Recursion:
Recursively alternates between A & B
0. A ::= B β
1. B ::= A | α

Solution:
Substitute B into A
0. A ::= A β | α β
Solve like normal Left Recursion
0. A ::= α β Tail
1. Tail ::= β Tail | ε