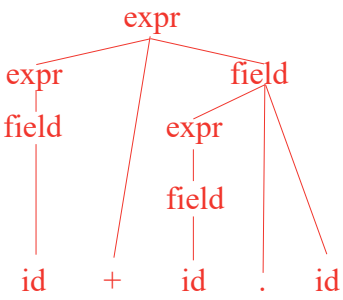
CSE 401 Section 2 — Grammars and Ambiguity Solutions

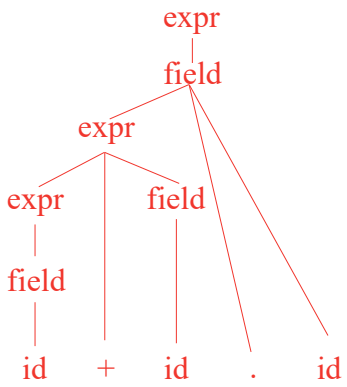
1. Consider the following syntax for expressions involving addition and field selection:

expr ::= *expr* + *field*
expr ::= *field*
field ::= *expr* . id
field ::= id

a) Show that this grammar is ambiguous.

Here are two derivations of id+id.id:





b) Give an unambiguous context-free grammar that fixes the problem(s) with the grammar in part (a) and generates expressions with `id`, field selection, and addition. As in Java, field selection should have higher precedence than addition and both field selection and addition should be left-associative (i.e. `a+b+c` means `(a+b)+c`).

The problem is in the first rule for *field*, which creates an ambiguous precedence

expr ::= *expr* + *field*
expr ::= *field*
field ::= *field* . id
field ::= id

2. The following grammar is ambiguous:

A ::= *B* b *C*
B ::= b | ε
C ::= b | ε

To demonstrate this ambiguity we can use pairs of derivations. Here are five different pairs. For each pair of derivations, circle OK if the pair correctly proves that the grammar is ambiguous. Circle WRONG if the pair does *not* give a correct proof. You do not need to explain your answers.

(Note: Whitespace in the grammar rules and derivations is used only for clarity. It is not part of the grammar or of the language generated by it.)

(a) OK WRONG

(Mix of left/rightmost derivations; also b b b has unique leftmost and unique rightmost derivations)

A => *B* b *C* => b b *C* => b b b
A => *B* b *C* => *B* b b => b b b

(b) OK WRONG

(Two different leftmost derivations of b b)

A => *B* b *C* => b b *C* => b b
A => *B* b *C* => b *C* => b b

(c) OK WRONG

(Different derivations: one leftmost, one rightmost)

A => *B* b *C* => b b *C* => b b
A => *B* b *C* => *B* b b => b b

(d) OK WRONG

(Two different strings, not two derivations of same string)

A => *B* b *C* => b b *C* => b b
A => *B* b *C* => b b *C* => b b b

(e) OK WRONG

(Two different rightmost derivations of b b)

A => *B* b *C* => *B* b => b b
A => *B* b *C* => *B* b b => b b

3. The following grammar is ambiguous. (As before, whitespace is used only for clarity; it is not part of the grammar or the language generated by it.)

P ::= ! Q | Q && Q | Q
Q ::= P | id

Give a grammar that generates exactly the same language as the one generated by this grammar but that is not ambiguous. You may resolve the ambiguities however you want – there is no requirement for any particular operator precedence or associativity in the resulting grammar.

This solution disambiguates ! and && by putting them in different productions, and also forces the binary operator && to be left-associative:

P ::= P && Q | Q
Q ::= !Q | id

Other unambiguous grammars that generated all of the strings produced by the original grammar also received full credit, regardless of how they fixed the problem.