Section 2:
Grammars & Ambiguity & Project Overview

CSE 401/M501

Adapted from Spring 2021

Announcements

- Due ***Tonight*** at **11:59PM**: HW1
- Due Thursday 4/11 at 11:59PM: scanner part of project
  - You'll be using git/CSE GitLab for project
  - Remember to **git tag** your submission

Monday	Tuesday	Wednesday	Thursday	Friday
01 12:00-13:00 OH (Connor) CSE2 150 14:30-15:20 Lecture CSE2 G10 *Scanners (concl.); Grammars and ambiguity (start) (3.1-3.2)* slides 15:30-16:30 OH (Aragorn) CSE2 151	02 12:00-13:00 OH (Eric) CSE2 153 15:30-16:30 OH (Edward) CSE2 151 23:59 project partner info due	03 11:30-12:30 OH (Aragorn) CSE2 150 14:30-15:20 Lecture CSE2 G10 *Grammars and ambiguity (concl.); LR (bottom-up) parsing (start) (3.4)* 15:30-16:30 OH (Richard) CSE2 151	04 Section *Project infrastructure, scanners, grammars* 10:30-11:30 OH (Richard) CSE2 151 16:30-18:00 OH (Edward) CSE2 150 23:59 hw1 due (Regular exps)	05 12:00-13:00 OH (Connor) CSE2 151 14:30-15:20 Lecture CSE2 G10 *LR parsing (cont.)* 15:30-16:30 OH (Eric) CSE2 153
08 12:00-13:00 OH (Connor) CSE2 150 14:30-15:20 Lecture CSE2 G10 *LR parsing (concl.); LR table construction (3.5)* 15:30-16:30 OH (Aragorn) CSE2 151	09 12:00-13:00 OH (Eric) CSE2 153 15:30-16:30 OH (Edward) CSE2 151	10 11:30-12:30 OH (Aragorn) CSE2 150 14:30-15:20 Lecture CSE2 G10 *LR conflicts, first/follow (no new slides)* 15:30-16:30 OH (Richard) CSE2 151	11 Section *LR parser construction* 10:30-11:30 OH (Richard) CSE2 151 16:30-18:00 OH (Edward) CSE2 150 23:59 Project: scanner due	12 12:00-13:00 OH (Connor) CSE2 151 14:30-15:20 Lecture CSE2 G10 *ASTs & visitors* 15:30-16:30 OH (Eric) CSE2 153

Agenda

- ~~Git Review~~
- Walkthrough of starter code
- Grammar/Ambiguity Practice

Code Walkthrough!

Summary: Project Structure

- Use ant to clean/compile/test…
- See README.txt for full folder description
    - src: your MiniJava compiler code
        - DemoParser.java and DemoScanner.java: example usages for you
        - MiniJava.java: the main compiler file, you will create this file and build on it for each lab
        - Scanner/minijava.jflex: Scanner code
        - Parser/minijava.cup: Parser code
        - Note: don't push build files; run `ant clean`
    - test: tests you will write
        - junit: JUnit tests for minijava
        - resources: your minijava programs and expected output
    - SamplePrograms: example programs for you

Summary: to support a new token

- src/Parser/minijava.cup
  - Add a new terminal for the symbol
- src/Scanner/minijava.jflex
  - Add a new regex rule to return the new symbol on match
  - If you want the raw value
    - Add a new case in `symbolToString`
    - Use `yytext()` to get the raw value

To avoid the common mistakes…

- Implement MiniJava, break the demo code/tests if needed
  - Read input from the specified file (**NOT** `System.in`), print output to `System.out`
  - Print errors to `System.err`
  - Use `System.exit` with status 1 after processing **entire file** if errors; status 0 if none
  - Do not generate `/* comment */` tokens
- Write and run (a lot of) JUnit tests
  - …and double check with the MiniJava grammar
- Do **NOT** modify or commit the generated files
  - Run `ant clean` before commit

Optional Testing Framework

- Framework by Apollo Zhu (22au)
- Simplifies the test code for MiniJava:

```
private void runScannerTestCase(String testCaseName) {
    try {
        new MiniJavaTestBuilder()
                .assertSystemOutMatchesContentsOf(
                        Path.of(TEST_FILES_LOCATION,
                                testCaseName + TEST_FILES_EXPECTED_EXTENSION))
                .testCompiler("-S", TEST_FILES_LOCATION + testCaseName + TEST_FILES_INPUT_EXTENSION);
    } catch (IOException e) {
        fail(e.getMessage());
    }
}
```

- Allows for testing error output and exit codes too
- Check out the website for more details on how to use this tool!

Grammar Worksheet!

Answers

Problem 1a

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

expr ::= *expr* + *field*

expr ::= *field*

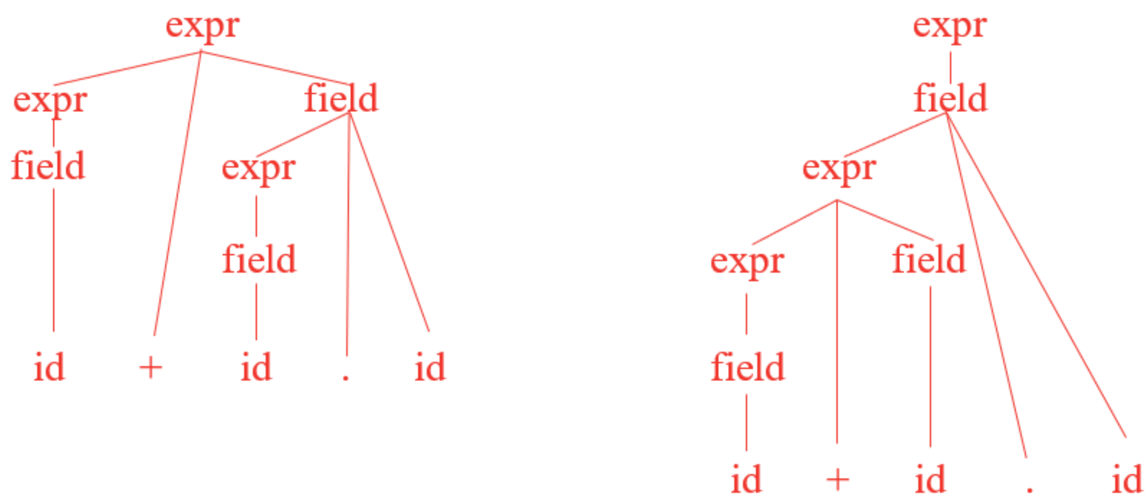
field ::= *expr* . id

field ::= id

a) Show that this grammar is ambiguous.

Problem 1a solution

Here are two derivations of id+id.id:



Problem 1b

1b) 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`).

expr ::= *expr* + *field*

expr ::= *field*

field ::= *expr* . id

field ::= id

Problem 1b answer

1b) 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

Problem 2

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.)

Problem 2a

$A ::= B \text{ b } C$

$B ::= \text{b} \mid \varepsilon$

$C ::= \text{b} \mid \varepsilon$

2a)

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

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

Problem 2a answer

$A ::= B \text{ b } C$
$B ::= \text{b} \mid \varepsilon$
$C ::= \text{b} \mid \varepsilon$

2a)

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

Wrong: Mix of left/rightmost derivations; also b b b has unique leftmost and unique rightmost derivations

Problem 2b

2b)

$$A \Rightarrow B\ b\ C \Rightarrow b\ b\ C \Rightarrow b\ b$$

$$A \Rightarrow B\ b\ C \Rightarrow b\ C \Rightarrow b\ b$$

A ::= B b C

B ::= b | ε

C ::= b | ε

Problem 2b answer

2b)

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

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

Ok: Two different leftmost derivations of b b

A ::= *B* b *C*

B ::= b | ε

C ::= b | ε

Problem 2c

2c)

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

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

A ::= B b C

B ::= b | ε

C :: = b | ε

Problem 2c answer

2c)

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

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

A ::= B b C

B ::= b | ε

C ::= b | ε

Wrong: Different derivations: one leftmost, one rightmost

Problem 2d

2d)

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

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

A ::= *B* b *C*

B ::= b | ε

C :: = b | ε

Problem 2d answer

2d)

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

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

A ::= *B* b *C*

B ::= b | ε

C ::= b | ε

Wrong: Two different strings, not two derivations of same string

Problem 2e

$A ::= B \text{ b } C$
$B ::= \text{b} \mid \varepsilon$
$C ::= \text{b} \mid \varepsilon$

2e)

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

Problem 2e answer

$A ::= B \text{ b } C$
$B ::= \text{b} \mid \varepsilon$
$C ::= \text{b} \mid \varepsilon$

2e)

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

Ok: Two different rightmost derivations of b b

Problem 3

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.)

$$P ::= \; !\, Q \mid Q \;\&\&\; Q \mid Q$$
$$Q ::= P \mid \text{id}$$

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.

Problem 3 answer

3) Original grammar:

$$P \ ::= \ !\,Q \ | \ Q \,\&\&\, Q \ | \ Q$$
$$Q \ ::= \ P \ | \ \text{id}$$

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

$$P \ ::= \ P \,\&\&\, Q \ | \ Q$$
$$Q \ ::= \ !Q \ | \ \text{id}$$

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.