CSE 401/M501 – Compilers

Section 2: Project Infrastructure, Ambiguity

Kory Watson, Aaron Johnston, Miya Natsuhara, Sam Wolfson

Autumn 2019

Agenda

- Quick refresher on `git` version control
  - See handouts/references on website for more
- Walkthrough of the starter code
  - How the project pieces fit together
- Practice with ambiguity of formal grammars
  - Determining when a grammar is ambiguous
  - Fixing ambiguity

Git Review – SSH Keys

- An SSH key lets a `git` server remember a specific client computer
- If `git` asks for a password to push or pull, you need to setup an SSH key
- Typically just need to do the following:
  - `ssh-keygen -t rsa -C "you@cs.washington.edu" -b 4096`
  - Copy `~/.ssh/id_rsa.pub` into your GitLab account
- Full setup and troubleshooting instructions: https://gitlab.cs.washington.edu/help/ssh/README

Git Review – Version Control

- The "official" repo (a.k.a., the **remote**) lives on the CSE GitLab server
- **Cloning** a repo gives you a private, local *copy*
- **Committing** saves *local* changes into the *local* repo's revision history
- **Push** to send *local* commits to *remote* repo
- **Pull** to bring *remote* commits to *local* repo
- Beware of **merge conflicts** – pull frequently

Git Review – The 401 Repository

- Each project pair is given a repository in which to work and collaborate
  - The repository starts out with a tiny demo compiler to show how the tools work together
- You will submit each phase of the project using a tag in the repository (see each project phase spec for exact tag)
- To get started, simply clone your repo locally and get started!

MiniJava Project – Walkthrough

Together, we're going to do the following:

1. Clone the repository
2. Try out the demo scanner
3. Get to know the CUP/JFlex infrastructure
4. Run a main program as in the scanner phase
5. Try making some changes to lexical spec.

DEMO

Ambiguity of a Formal Grammar

- Recall from lecture:
  - A formal grammar is *ambiguous* when a sentence in the language has multiple leftmost (or rightmost) derivations (*i.e.*, multiple parse trees).
- Now some exercises selected from a past exam…

Ambiguity – 4.a (15wi midterm)

Question 4. Context-free grammars (14 points) Consider the following syntax for expressions involving addition and field selection:

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

(a) (8 points) Show that this grammar is ambiguous.

Ambiguity – 4.b (15wi midterm)

Question 4. Context-free grammars (14 points) Consider the following syntax for expressions involving addition and field selection:

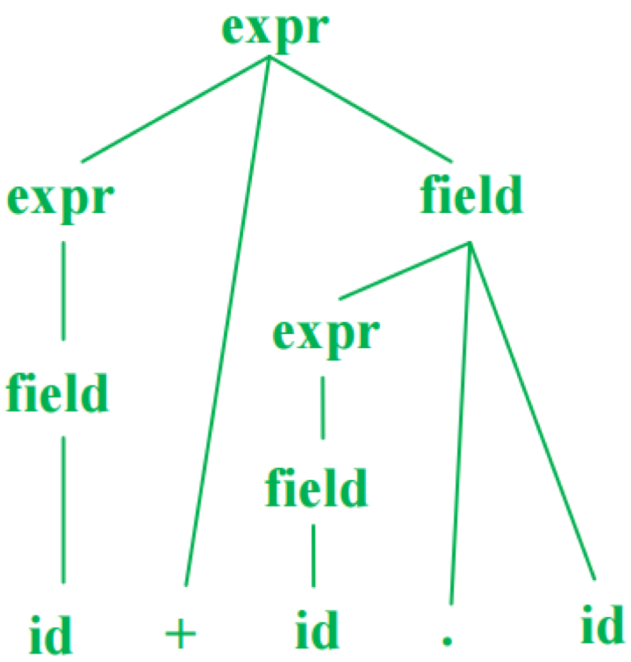
expr ::= *expr* + *field*
expr ::= *field*
field ::= *expr* `.` `id`
field ::= `id`

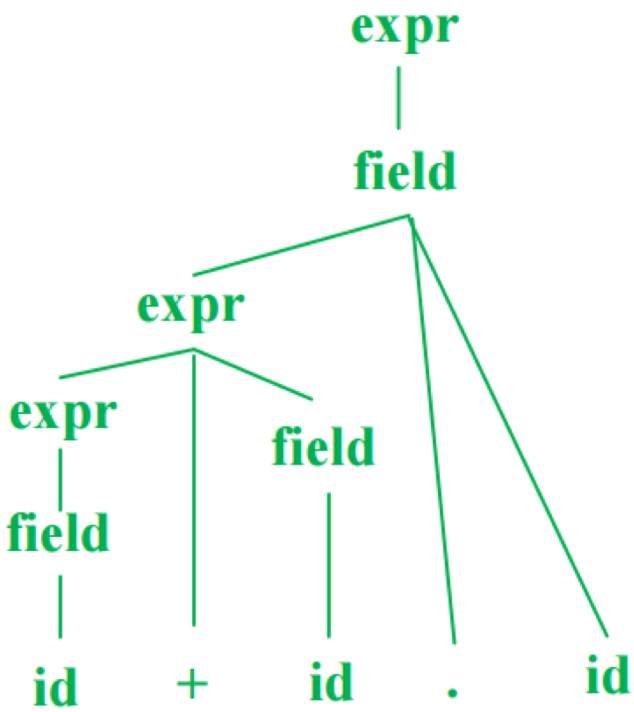
(b) (6 points) Give an unambiguous context-free grammar that fixes the problem(s) with the grammar in part (a) and generates expressions with `id`, field selection and addition. As in Java, field selection should have higher precedence than addition and both field selection and addition should be left-associative (i.e., a+b+c means (a+b)+c).

Ambiguity – 4.a solution (example)

(a) (8 points) Show that this grammar is ambiguous.

Here are two derivations of id+id.id:





Ambiguity – 4.b solution (example)

(b) (6 points) 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. Here is a reasonably simple fix.

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