Section 2: Grammars & Ambiguity

Anand, Gavin, Yukai

Adapted from Autumn 2020

Announcements

- Due Tonight at 11PM: HW1
  - Office hours on Zoom (Canvas tab)
- Due Th 4/15 11PM: scanner part of project

April				
Monday	Tuesday	Wednesday	Thursday	Friday
05 14:30-15:20 Lecture zoom link *Scanners (concl.); Grammars and ambiguity (start) (3.1-3.3)* slides	06 23:00 project partner info due	07 14:30-15:20 Lecture zoom link *Grammars and ambiguity*	08 Section *Project infrastructure, scanners, grammars* 23:00 hw1 due (Regular exps)	09 14:30-15:20 Lecture zoom link *Ambiguity (concl.); LR (bottom-up) parsing (3.4)* slides
12 14:30-15:20 Lecture zoom link *LR parsing (concl.); LR table construction (start) (3.5)*	13	14 14:30-15:20 Lecture zoom link *LR table construction (concl.)*	15 Section *LR parser construction* 23:00 Project: scanner due	16 14:30-15:20 Lecture zoom link *LR conflicts, first / follow, SLR*

Demo

- Git refresher
- Walkthrough of starter code

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 – Version Control

- When in doubt, Google it and copy-paste the StackOverflow answer with the most upvotes
  - Make sure to check the replies for words of caution
  - Be careful of using the 'force' option
  - Don't hesitate to copy-paste a backup somewhere before messing around with weird git commands
- We're here to help
- Consult the official git documentation at https://git-scm.com/doc

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!

Code Walkthrough!

Grammar Worksheet!

Answers

Problem 1a

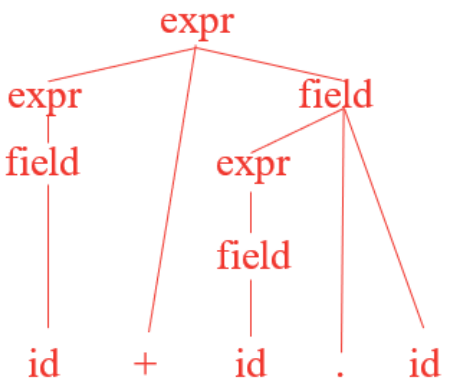
1) Consider the following syntax for expressions involving addition and field selection:

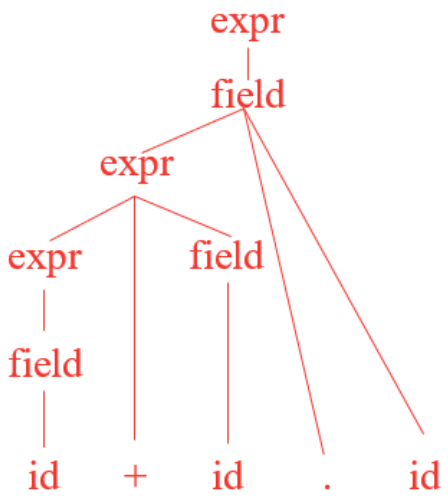
expr ::= *expr* + *field*
expr ::= *field*
field ::= *expr* . id
field ::= id

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

2a)

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

$$A ::= B\ b\ C$$
$$B ::= b \mid \varepsilon$$
$$C ::= b \mid \varepsilon$$

Problem 2a answer

2a)

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

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

A ::= *B* b *C*

B ::= b | ε

C ::= b | ε

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

Problem 2b

2b)

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

$$A ::= B\ b\ C$$
$$B ::= b \mid \varepsilon$$
$$C ::= b \mid \varepsilon$$

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)

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

$$A ::= B\ b\ C$$
$$B ::= b \mid \varepsilon$$
$$C ::= b \mid \varepsilon$$

Problem 2c answer

2c)

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

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

$A ::= B\ b\ C$

$B ::= b \mid \varepsilon$

$C ::= b \mid \varepsilon$

Wrong: Different derivations: one leftmost, one rightmost

Problem 2d

2d)

$$A \Rightarrow B\ \mathrm{b}\ C \Rightarrow \mathrm{b}\ \mathrm{b}\ C \Rightarrow \mathrm{b}\ \mathrm{b}$$
$$A \Rightarrow B\ \mathrm{b}\ C \Rightarrow \mathrm{b}\ \mathrm{b}\ C \Rightarrow \mathrm{b}\ \mathrm{b}\ \mathrm{b}$$

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

Problem 2d answer

2d)

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

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

$A ::= B\ b\ C$

$B ::= b \mid \varepsilon$

$C ::= b \mid \varepsilon$

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

Problem 2e

2e)

$$A \Rightarrow B\ \mathrm{b}\ C \Rightarrow B\ \mathrm{b} \Rightarrow \mathrm{b}\ \mathrm{b}$$
$$A \Rightarrow B\ \mathrm{b}\ C \Rightarrow B\ \mathrm{b}\ \mathrm{b} \Rightarrow \mathrm{b}\ \mathrm{b}$$

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

Problem 2e answer

2e)

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

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

$A ::= B\ b\ C$

$B ::= b \mid \varepsilon$

$C ::= b \mid \varepsilon$

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 | 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.

Problem 3 answer

3) Original grammar:

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

Q ::= P | 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 | 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.