

Introduction to Parsing

Lecture 5

Prof. Aiken CS 143 Lecture 5

1

Outline

- Regular languages revisited
- Parser overview
- Context-free grammars (CFG's)
- Derivations
- Ambiguity

Prof. Aiken CS 143 Lecture 5

2

Languages and Automata

- Formal languages are very important in CS
 - Especially in programming languages
- Regular languages
 - The weakest formal languages widely used
 - Many applications
- We will also study context-free languages, tree languages

Prof. Aiken CS 143 Lecture 5

3

Beyond Regular Languages

- Many languages are not regular
- Strings of balanced parentheses are not regular:

$$\{()^i \mid i \geq 0\}$$

Prof. Aiken CS 143 Lecture 5

4

What Can Regular Languages Express?

- Languages requiring counting modulo a fixed integer
- Intuition: A finite automaton that runs long enough must repeat states
- Finite automaton can't remember # of times it has visited a particular state

Prof. Aiken CS 143 Lecture 5

5

The Functionality of the Parser

- Input: sequence of tokens from lexer
- Output: parse tree of the program
(But some parsers never produce a parse tree . . .)

Prof. Aiken CS 143 Lecture 5

6

Example

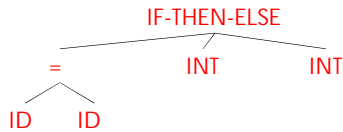
- Cool

if x = y then 1 else 2 fi

- Parser input

IF ID = ID THEN INT ELSE INT FI

- Parser output



Prof. Aiken CS 143 Lecture 5

7

Comparison with Lexical Analysis

Phase	Input	Output
Lexer	String of characters	String of tokens
Parser	String of tokens	Parse tree

Prof. Aiken CS 143 Lecture 5

8

The Role of the Parser

- Not all strings of tokens are programs . . .
- . . . Parser must distinguish between valid and invalid strings of tokens
- We need
 - A language for describing valid strings of tokens
 - A method for distinguishing valid from invalid strings of tokens

Prof. Aiken CS 143 Lecture 5

9

Context-Free Grammars

- Programming language constructs have recursive structure
- An **EXPR** is
 - if EXPR then EXPR else EXPR fi
 - while EXPR loop EXPR pool
 - ...
- Context-free grammars are a natural notation for this recursive structure

Prof. Aiken CS 143 Lecture 5

10

CFGs (Cont.)

- A CFG consists of
 - A set of *terminals* T
 - A set of *non-terminals* N
 - A *start symbol* S (a non-terminal)
 - A set of *productions*

$$X \rightarrow Y_1 Y_2 \cdots Y_n$$

where $X \in N$ and $Y_i \in T \cup N \cup \{\epsilon\}$

Prof. Aiken CS 143 Lecture 5

11

Notational Conventions

- In these lecture notes
 - Non-terminals are written upper-case
 - Terminals are written lower-case
 - The start symbol is the left-hand side of the first production

Prof. Aiken CS 143 Lecture 5

12

Examples of CFGs

A fragment of Cool:

```
EXPR → if EXPR then EXPR else EXPR fi
      | while EXPR loop EXPR pool
      | id
```

Prof. Aiken CS 143 Lecture 5

13

Examples of CFGs (cont.)

Simple arithmetic expressions:

```
E → E * E
   | E + E
   | (E)
   | id
```

Prof. Aiken CS 143 Lecture 5

14

The Language of a CFG

Read productions as rules:

$$X \rightarrow Y_1 \cdots Y_n$$

Means X can be replaced by $Y_1 \cdots Y_n$

Prof. Aiken CS 143 Lecture 5

15

Key Idea

1. Begin with a string consisting of the start symbol " S "
2. Replace any non-terminal X in the string by the right-hand side of some production

$$X \rightarrow Y_1 \cdots Y_n$$

3. Repeat (2) until there are no non-terminals in the string

Prof. Aiken CS 143 Lecture 5

16

The Language of a CFG (Cont.)

More formally, write

$$X_1 \cdots X_i \cdots X_n \rightarrow X_1 \cdots X_{i-1} Y_1 \cdots Y_m X_{i+1} \cdots X_n$$

if there is a production

$$X_i \rightarrow Y_1 \cdots Y_m$$

Prof. Aiken CS 143 Lecture 5

17

The Language of a CFG (Cont.)

Write

$$X_1 \cdots X_n \xrightarrow{*} Y_1 \cdots Y_m$$

if

$$X_1 \cdots X_n \rightarrow \cdots \rightarrow \cdots \rightarrow Y_1 \cdots Y_m$$

in 0 or more steps

Prof. Aiken CS 143 Lecture 5

18

The Language of a CFG

Let G be a context-free grammar with start symbol S . Then the language of G is:

$$\{a_1 \dots a_n \mid S \xrightarrow{*} a_1 \dots a_n \text{ and every } a_i \text{ is a terminal}\}$$

Prof. Aiken CS 143 Lecture 5

19

Terminals

- Terminals are so-called because there are no rules for replacing them
- Once generated, terminals are permanent
- Terminals ought to be tokens of the language

Prof. Aiken CS 143 Lecture 5

20

Examples

$L(G)$ is the language of CFG G

Strings of balanced parentheses $\{()^i \mid i \geq 0\}$

Two grammars:

$$\begin{array}{l} S \rightarrow (S) \\ S \rightarrow \varepsilon \end{array} \quad \text{OR} \quad \begin{array}{l} S \rightarrow (S) \\ \quad \mid \quad \varepsilon \end{array}$$

Prof. Aiken CS 143 Lecture 5

21

Cool Example

A fragment of COOL:

```
EXPR → if EXPR then EXPR else EXPR fi
      | while EXPR loop EXPR pool
      | id
```

Prof. Aiken CS 143 Lecture 5

22

Cool Example (Cont.)

Some elements of the language

id

if id then id else id fi

while id loop id pool

if while id loop id pool then id else id

if if id then id else id fi then id else id fi

Prof. Aiken CS 143 Lecture 5

23

Arithmetic Example

Simple arithmetic expressions:

$$E \rightarrow E + E \mid E * E \mid (E) \mid \text{id}$$

Some elements of the language:

id		id + id
(id)		id * id
(id) * id		id * (id)

Prof. Aiken CS 143 Lecture 5

24

Notes

The idea of a CFG is a big step. But:

- Membership in a language is “yes” or “no”; also need parse tree of the input
- Must handle errors gracefully
- Need an implementation of CFG's (e.g., bison)

Prof. Aiken CS 143 Lecture 5

25

More Notes

- Form of the grammar is important
 - Many grammars generate the same language
 - Tools are sensitive to the grammar
- Note: Tools for regular languages (e.g., flex) are sensitive to the form of the regular expression, but this is rarely a problem in practice

Prof. Aiken CS 143 Lecture 5

26

Derivations and Parse Trees

A *derivation* is a sequence of productions

$$S \rightarrow \dots \rightarrow \dots \rightarrow \dots$$

A derivation can be drawn as a tree

- Start symbol is the tree's root
- For a production $X \rightarrow Y_1 \dots Y_n$ add children $Y_1 \dots Y_n$ to node X

Prof. Aiken CS 143 Lecture 5

27

Derivation Example

- Grammar

$$E \rightarrow E+E \mid E * E \mid (E) \mid id$$

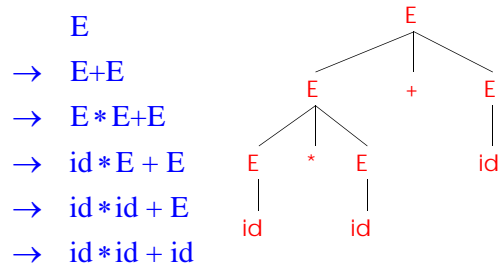
- String

$$id * id + id$$

Prof. Aiken CS 143 Lecture 5

28

Derivation Example (Cont.)



Prof. Aiken CS 143 Lecture 5

29

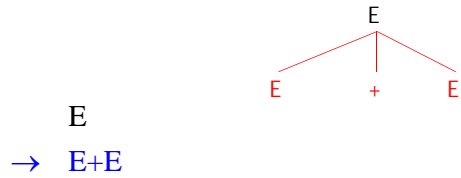
Derivation in Detail (1)



Prof. Aiken CS 143 Lecture 5

30

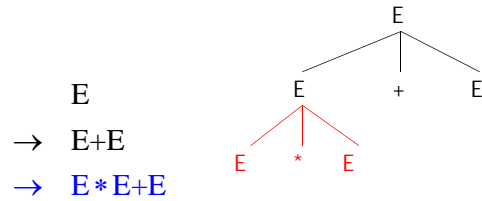
Derivation in Detail (2)



Prof. Aiken CS 143 Lecture 5

31

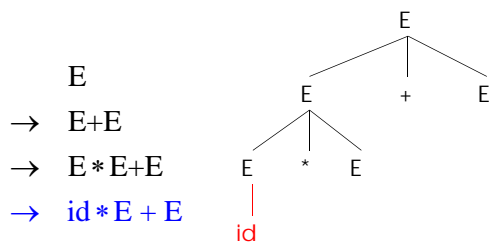
Derivation in Detail (3)



Prof. Aiken CS 143 Lecture 5

32

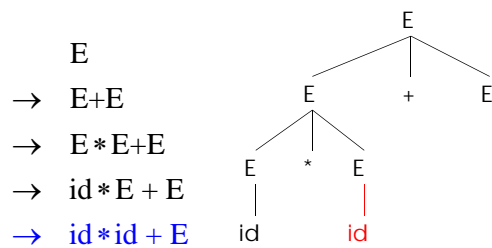
Derivation in Detail (4)



Prof. Aiken CS 143 Lecture 5

33

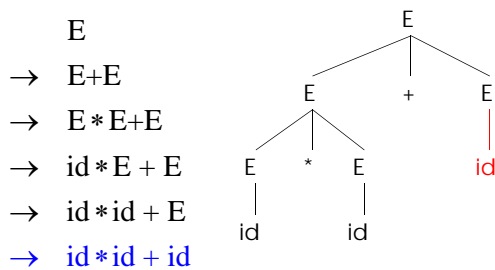
Derivation in Detail (5)



Prof. Aiken CS 143 Lecture 5

34

Derivation in Detail (6)



Prof. Aiken CS 143 Lecture 5

35

Notes on Derivations

- A parse tree has
 - Terminals at the leaves
 - Non-terminals at the interior nodes
- An in-order traversal of the leaves is the original input
- The parse tree shows the association of operations, the input string does not

Prof. Aiken CS 143 Lecture 5

36

Left-most and Right-most Derivations

- The example is a *left-most* derivation
 - At each step, replace the left-most non-terminal
- There is an equivalent notion of a *right-most* derivation

E
 $\rightarrow E+E$
 $\rightarrow E+id$
 $\rightarrow E * E + id$
 $\rightarrow E * id + id$
 $\rightarrow id * id + id$

Prof. Aiken CS 143 Lecture 5

37

Right-most Derivation in Detail (1)

E
 E

Prof. Aiken CS 143 Lecture 5

38

Right-most Derivation in Detail (2)

E
 $\rightarrow E+E$

E
 $+$
 E

Prof. Aiken CS 143 Lecture 5

39

Right-most Derivation in Detail (3)

E
 $\rightarrow E+E$
 $\rightarrow E+id$

E
 $+$
 E
 id

Prof. Aiken CS 143 Lecture 5

40

Right-most Derivation in Detail (4)

E
 $\rightarrow E+E$
 $\rightarrow E+id$
 $\rightarrow E * E + id$

E
 $*$
 E
 id

Prof. Aiken CS 143 Lecture 5

41

Right-most Derivation in Detail (5)

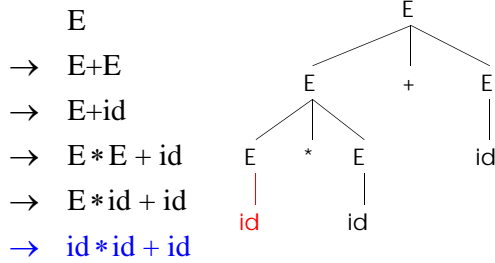
E
 $\rightarrow E+E$
 $\rightarrow E+id$
 $\rightarrow E * E + id$
 $\rightarrow E * id + id$

E
 $*$
 E
 id
 id

Prof. Aiken CS 143 Lecture 5

42

Right-most Derivation in Detail (6)



Prof. Aiken CS 143 Lecture 5

43

Derivations and Parse Trees

- Note that right-most and left-most derivations have the same parse tree
- The difference is the order in which branches are added

Prof. Aiken CS 143 Lecture 5

44

Summary of Derivations

- We are not just interested in whether $s \in L(G)$
 - We need a parse tree for s
- A derivation defines a parse tree
 - But one parse tree may have many derivations
- Left-most and right-most derivations are important in parser implementation

Prof. Aiken CS 143 Lecture 5

45

Ambiguity

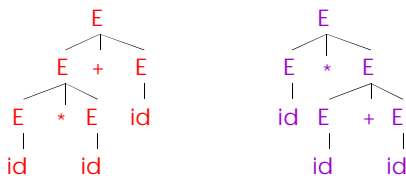
- Grammar $E \rightarrow E + E \mid E * E \mid (E) \mid id$
- String $id * id + id$

Prof. Aiken CS 143 Lecture 5

46

Ambiguity (Cont.)

This string has two parse trees



Prof. Aiken CS 143 Lecture 5

47

Ambiguity (Cont.)

- A grammar is *ambiguous* if it has more than one parse tree for some string
 - Equivalently, there is more than one right-most or left-most derivation for some string
- Ambiguity is BAD
 - Leaves meaning of some programs ill-defined

Prof. Aiken CS 143 Lecture 5

48

Dealing with Ambiguity

- There are several ways to handle ambiguity
- Most direct method is to rewrite grammar unambiguously

$$E \rightarrow E' + E \mid E'$$

$$E' \rightarrow \text{id} * E' \mid \text{id} \mid (E) * E' \mid (E)$$

- Enforces precedence of $*$ over $+$

Prof. Aiken CS 143 Lecture 5

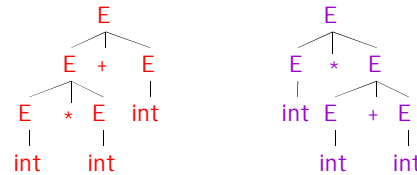
49

Ambiguity in Arithmetic Expressions

- Recall the grammar

$$E \rightarrow E + E \mid E * E \mid (E) \mid \text{int}$$

- The string $\text{int} * \text{int} + \text{int}$ has two parse trees:



Prof. Aiken CS 143 Lecture 5

50

Ambiguity: The Dangling Else

- Consider the grammar

$$E \rightarrow \text{if } E \text{ then } E$$

$$\quad \mid \text{if } E \text{ then } E \text{ else } E$$

$$\quad \mid \text{OTHER}$$

- This grammar is also ambiguous

Prof. Aiken CS 143 Lecture 5

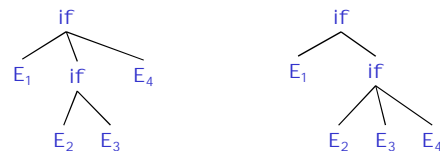
51

The Dangling Else: Example

- The expression

$$\text{if } E_1 \text{ then if } E_2 \text{ then } E_3 \text{ else } E_4$$

has two parse trees



- Typically we want the second form

Prof. Aiken CS 143 Lecture 5

52

The Dangling Else: A Fix

- else matches the closest unmatched then
- We can describe this in the grammar

$$E \rightarrow \text{MIF} \quad /* \text{ all then are matched } */$$

$$\quad \mid \text{UIF} \quad /* \text{ some then is unmatched } */$$

$$\text{MIF} \rightarrow \text{if } E \text{ then MIF else MIF}$$

$$\quad \mid \text{OTHER}$$

$$\text{UIF} \rightarrow \text{if } E \text{ then } E$$

$$\quad \mid \text{if } E \text{ then MIF else UIF}$$

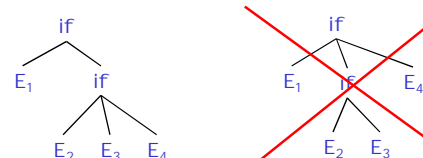
- Describes the same set of strings

Prof. Aiken CS 143 Lecture 5

53

The Dangling Else: Example Revisited

- The expression $\text{if } E_1 \text{ then if } E_2 \text{ then } E_3 \text{ else } E_4$



- A valid parse tree (for a UIF)

- Not valid because the then expression is not a MIF

Prof. Aiken CS 143 Lecture 5

54

Ambiguity

- No general techniques for handling ambiguity
- Impossible to convert automatically an ambiguous grammar to an unambiguous one
- Used with care, ambiguity can simplify the grammar
 - Sometimes allows more natural definitions
 - We need disambiguation mechanisms

Prof. Aiken CS 143 Lecture 5

55

Precedence and Associativity Declarations

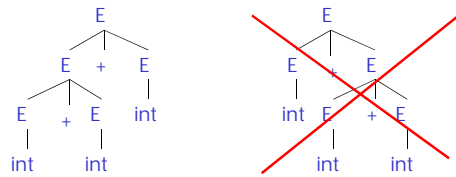
- Instead of rewriting the grammar
 - Use the more natural (ambiguous) grammar
 - Along with disambiguating declarations
- Most tools allow precedence and associativity declarations to disambiguate grammars
- Examples ...

Prof. Aiken CS 143 Lecture 5

56

Associativity Declarations

- Consider the grammar $E \rightarrow E + E \mid \text{int}$
- Ambiguous: two parse trees of $\text{int} + \text{int} + \text{int}$



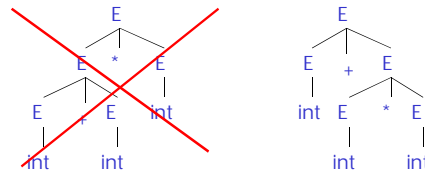
- Left associativity declaration: `%left +`

Prof. Aiken CS 143 Lecture 5

57

Precedence Declarations

- Consider the grammar $E \rightarrow E + E \mid E * E \mid \text{int}$
- And the string $\text{int} + \text{int} * \text{int}$



- Precedence declarations: `%left +`
`%left *`

Prof. Aiken CS 143 Lecture 5

58