

# Syntax Analysis

Chapter 4, Part I  
Context Free Grammar

CSS810 Spring 2008 Chapter4-1 1

## The Structure of a Compiler

```

    graph TD
      LA[Lexical Analyzer] --> SA[Syntax Analyzer]
      SA --> SEM[Semantic Analyzer]
      SEM --> ICG[Intermediate Code Generator]
      ICG --> MICO[Machine-Independent Code Optimizer]
      MICO --> CG[Code Generator]
      CG --> MDCO[Machine-dependent Code Optimizer]
  
```

CSS810 Spring 2008 Chapter4-1 2

## The Functionality of the Parser

- Input:** sequence of tokens from lexer
- Output:** parse tree of the program

```

    graph LR
      SP[source program] --> LA[Lexical Analyzer]
      LA -- token yylex() --> P[Parser]
      P -- parse tree --> LS[Later Stages]
      P --> SE[Syntax Errors]
  
```

CSS810 Spring 2008 Chapter4-1 3

## Parser

- Checks the stream of words and their parts of speech (produced by the scanner) for grammatical correctness
- Determines if the input is syntactically well formed
- Guides checking at deeper levels than syntax
- Builds an IR representation of the code

CSS810 Spring 2008 Chapter4-1 4

## Example

- Source input  
`if x = y then 1 else 2 fi`
- Parser input  
`IF ID = ID THEN INT ELSE INT FI`
- Parser output

```

    graph TD
      A[IF-THEN-ELSE] --> B[=]
      A --> C[INT]
      A --> D[INT]
      B --> E[ID]
      B --> F[ID]
  
```

CSS810 Spring 2008 Chapter4-1 5

## Comparison with Lexical Analysis

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

CSS810 Spring 2008 Chapter4-1 6

### The Study of Parsing

The process of discovering a derivation for some sentence

- Need a mathematical model of syntax — a grammar  $G$
- Need an algorithm for testing membership in  $L(G)$
- Need to keep in mind that our goal is building parsers, not studying the mathematics of arbitrary languages

CSS810 Spring 2008 Chapter4-1 7

### Roadmap for our study of parsing

1. Context-free grammars and derivations
2. Top-down parsing
  - Generated LL(1) parsers
3. Bottom-up parsing
  - Generated LR(1) parsers

CSS810 Spring 2008 Chapter4-1 8

### Context Free Grammars

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

$X \rightarrow \epsilon$  , or  
 $X \rightarrow Y_1 Y_2 \dots Y_n$  where  $Y_i \in N \cup T$   
 $X \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_k$

CSS810 Spring 2008 Chapter4-1 9

### 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

CSS810 Spring 2008 Chapter4-1 10

### Examples of CFGs (cont.)

Simple arithmetic expressions:

$$\begin{array}{l}
 E \rightarrow E * E \\
 \quad \mid E + E \\
 \quad \mid (E) \\
 \quad \mid id
 \end{array}$$

CSS810 Spring 2008 Chapter4-1 11

### The Language of a CFG

Read productions as replacement rules:

$X \rightarrow Y_1 \dots Y_n$   
 Means  $X$  can be replaced by  $Y_1 \dots Y_n$

$X \rightarrow \epsilon$   
 Means  $X$  can be erased (replaced with empty string)

CSS810 Spring 2008 Chapter4-1 12

### Key Idea

1. Begin with a string consisting of the start symbol "S"
2. Replace any non-terminal  $X$  in the string by a right-hand side of some production
 
$$X \rightarrow Y_1 \dots Y_n$$
3. Repeat (2) until there are no non-terminals in the string

CSS810 Spring 2008
Chapter4-1
13

### The Language of a CFG (Cont.)

More formally, write

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

if there is a production

$$X_i \rightarrow Y_1 \dots Y_m$$

CSS810 Spring 2008
Chapter4-1
14

### The Language of a CFG (Cont.)

Write

$$X_1 \dots X_n \rightarrow^* Y_1 \dots Y_m$$

if

$$X_1 \dots X_n \rightarrow \dots \rightarrow Y_1 \dots Y_m$$

in 0 or more steps

CSS810 Spring 2008
Chapter4-1
15

### The Language of a CFG

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

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

CSS810 Spring 2008
Chapter4-1
16

### Terminals

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

CSS810 Spring 2008
Chapter4-1
17

### Arithmetic Example

Simple arithmetic expressions:

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

Some elements of the language:

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

CSS810 Spring 2008
Chapter4-1
18

### Notes

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

CS5810 Spring 2008

Chapter4-1

19

### More Notes

- Form of the grammar is important
  - Many grammars generate the same language
  - Tools are sensitive to the grammar

CS5810 Spring 2008

Chapter4-1

20

### Derivations and Parse Trees

A *derivation* is a sequence of productions

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

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  $X$

CS5810 Spring 2008

Chapter4-1

21

### Derivation Example

- Grammar

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

- String

$$id * id + id$$

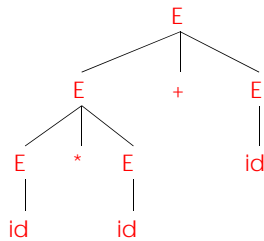
CS5810 Spring 2008

Chapter4-1

22

### Derivation Example (Cont.)

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



CS5810 Spring 2008

Chapter4-1

23

### Derivation in Detail (1)



CS5810 Spring 2008

Chapter4-1

24

### Derivation in Detail (2)

E

→ E+E

CS5810 Spring 2008Chapter4-125

### Derivation in Detail (3)

E

→ E+E

→ E \* E + E

CS5810 Spring 2008Chapter4-126

### Derivation in Detail (4)

E

→ E+E

→ E \* E + E

→ id \* E + E

CS5810 Spring 2008Chapter4-127

### Derivation in Detail (5)

E

→ E+E

→ E \* E + E

→ id \* E + E

→ id \* id + E

CS5810 Spring 2008Chapter4-128

### Derivation in Detail (6)

E

→ E+E

→ E \* E + E

→ id \* E + E

→ id \* id + E

→ id \* id + id

CS5810 Spring 2008Chapter4-129

### 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

CS5810 Spring 2008Chapter4-130

### 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

$$E$$

$$\rightarrow E+E$$

$$\rightarrow E+id$$

$$\rightarrow E * E + id$$

$$\rightarrow E * id + id$$

$$\rightarrow id * id + id$$

CS5810 Spring 2008 Chapter4-1 31

### Right-most Derivation in Detail (1)

$$E$$

CS5810 Spring 2008 Chapter4-1 32

### Right-most Derivation in Detail (2)

$$E$$

$$\rightarrow E+E$$

CS5810 Spring 2008 Chapter4-1 33

### Right-most Derivation in Detail (3)

$$E$$

$$\rightarrow E+E$$

$$\rightarrow E+id$$

CS5810 Spring 2008 Chapter4-1 34

### Right-most Derivation in Detail (4)

$$E$$

$$\rightarrow E+E$$

$$\rightarrow E+id$$

$$\rightarrow E * E + id$$

CS5810 Spring 2008 Chapter4-1 35

### Right-most Derivation in Detail (5)

$$E$$

$$\rightarrow E+E$$

$$\rightarrow E+id$$

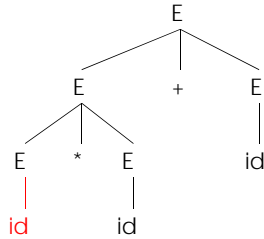
$$\rightarrow E * E + id$$

$$\rightarrow E * id + id$$

CS5810 Spring 2008 Chapter4-1 36

### Right-most Derivation in Detail (6)

- E
- E+E
- E+id
- E \* E + id
- E \* id + id
- id \* id + id



CS5810 Spring 2008

Chapter4-1

37

### 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

CS5810 Spring 2008

Chapter4-1

38

### 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

CS5810 Spring 2008

Chapter4-1

39

### Ambiguity

- A grammar that produces more than one parse tree for some sentence is ambiguous
  - id + id \* id

CS5810 Spring 2008

Chapter4-1

40

### CFG v.s. RE

- Every regular language is a context-free language
  - For each state  $i$  of the NFA, create a nonterminal  $A_i$
  - If the state  $i$  has a transition to  $j$  on input  $a$ , add the production  $A_i \rightarrow aA_j$ . If  $i$  goes to  $j$  on input  $\epsilon$ , add  $A_i \rightarrow A_j$
  - If  $i$  is an accepting state, add  $A_i \rightarrow \epsilon$
  - If  $i$  is the start state, make  $A_i$  the start symbol
  - E.x.  $(a|b)^*abb$

CS5810 Spring 2008

Chapter4-1

41

### CFG v.s. RE

- A context-free language is not necessary a regular language is
  - Finite automaton can't remember # of times it has visited a particular state
  - E.g., language of balanced parentheses is not regular:

$$\{c^i \mid i \geq 0\}$$

CS5810 Spring 2008

Chapter4-1

42

### Examples

Strings of balanced parentheses

Two grammars:  $\{( )^i \mid i \geq 0\}$

$$S \rightarrow (S) \quad \text{OR} \quad S \rightarrow (S)$$

$$S \rightarrow \epsilon \quad \quad \quad | \quad \epsilon$$

CSS810 Spring 2008 Chapter 4-1 43

### Intro to Top-Down Parsing

- Terminals are seen in order of appearance in the token stream:
 
$$t_1 \ t_2 \ t_3 \ t_4 \ t_5$$
- The parse tree is constructed
  - From the top
  - From left to right

2/6/2008 Chapter 4-2, CSS810 Spring 2008 44

### Recursive Descent Parsing

- Consider the grammar
 
$$E \rightarrow T + E \mid T$$

$$T \rightarrow ( E ) \mid \text{int} \mid \text{int} * T$$
- Token stream is:  $\text{int} * \text{int}$
- Start with top-level non-terminal  $E$
- Try the rules for  $E$  in order

2/6/2008 Chapter 4-2, CSS810 Spring 2008 45

### Recursive Descent Parsing. Example (Cont.)

- Try  $E \rightarrow T + E$
- Then try a rule for  $T \rightarrow ( E )$ 
  - But  $($  does not match input token  $\text{int}$
- Try  $T \rightarrow \text{int}$ . Token matches.
  - But  $+$  after  $T$  does not match input token  $*$
- Try  $T \rightarrow \text{int} * T$ 
  - This will match but  $+$  after  $T$  will be unmatched
- Have exhausted the choices for  $T$ 
  - Backtrack to choice for  $E$

2/6/2008 Chapter 4-2, CSS810 Spring 2008 46

### Recursive Descent Parsing. Example (Cont.)

- Try  $E \rightarrow T$
- Follow same steps as before for  $T$ 
  - And succeed with  $T \rightarrow \text{int} * T$  and  $T \rightarrow \text{int}$
  - With the following parse tree

2/6/2008 Chapter 4-2, CSS810 Spring 2008 47

### Recursive Descent Parsing. Notes.

- Easy to implement by hand
  - An example implementation is provided as a supplement "Recursive Descent Parsing"
- But does not always work ...

2/6/2008 Chapter 4-2, CSS810 Spring 2008 48

## Recursive-Descent Parsing

- Parsing: given a string of tokens  $t_1 t_2 \dots t_n$ , find its parse tree
- Recursive-descent parsing: Try all the productions exhaustively
  - At a given moment the fringe of the parse tree is:  $t_1 t_2 \dots t_k A \dots$
  - Try all the productions for A: if  $A \rightarrow BC$  is a production, the new fringe is  $t_1 t_2 \dots t_k B C \dots$
  - Backtrack when the fringe doesn't match the string
  - Stop when there are no more non-terminals

2/6/2008

Chapter 4-2, CSS810 Spring 2008

49

## When Recursive Descent Does Not Work

- Consider a production  $S \rightarrow S a$ :
  - In the process of parsing  $S$  we try the above rule
  - What goes wrong?
- A left-recursive grammar has a non-terminal  $S$ 

$$S \rightarrow^+ S \alpha \text{ for some } \alpha$$
- Recursive descent does not work in such cases
  - It goes into a dead loop

2/6/2008

Chapter 4-2, CSS810 Spring 2008

50

## Elimination of Left Recursion

- Consider the left-recursive grammar
 
$$S \rightarrow S \alpha \mid \beta$$
- $S$  generates all strings starting with a  $\beta$  and followed by a number of  $\alpha$
- Can rewrite using right-recursion
 
$$S \rightarrow \beta S'$$

$$S' \rightarrow \alpha S' \mid \epsilon$$

2/6/2008

Chapter 4-2, CSS810 Spring 2008

51

## Elimination of Left-Recursion. Example

- Consider the grammar
 
$$S \rightarrow 1 \mid S 0 \quad (\beta = 1 \text{ and } \alpha = 0)$$
- can be rewritten as
 
$$S \rightarrow 1 S'$$

$$S' \rightarrow 0 S' \mid \epsilon$$

2/6/2008

Chapter 4-2, CSS810 Spring 2008

52

## More Elimination of Left-Recursion

- In general
 
$$S \rightarrow S \alpha_1 \mid \dots \mid S \alpha_n \mid \beta_1 \mid \dots \mid \beta_m$$
- All strings derived from  $S$  start with one of  $\beta_1, \dots, \beta_m$  and continue with several instances of  $\alpha_1, \dots, \alpha_n$
- Rewrite as
 
$$S \rightarrow \beta_1 S' \mid \dots \mid \beta_m S'$$

$$S' \rightarrow \alpha_1 S' \mid \dots \mid \alpha_n S' \mid \epsilon$$

2/6/2008

Chapter 4-2, CSS810 Spring 2008

53

## General Left Recursion

- The grammar
 
$$S \rightarrow A \alpha \mid \delta$$

$$A \rightarrow S \beta$$
 is also left-recursive because
 
$$S \rightarrow^+ S \beta \alpha$$
- This left-recursion can also be eliminated
- See book, Section 4.3 for general algorithm

2/6/2008

Chapter 4-2, CSS810 Spring 2008

54

## Summary of Recursive Descent

- Simple and general parsing strategy
  - Left-recursion must be eliminated first
  - ... but that can be done automatically
- Unpopular because of backtracking
  - Thought to be too inefficient
- In practice, backtracking is eliminated by restricting the grammar