



CSCI 742 - Compiler Construction

Lecture 8

Introduction to Syntax Analysis

Instructor: Hossein Hojjat

February 2, 2017

Compiler Phases

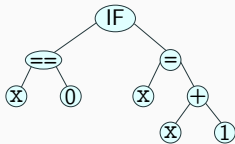
Source Code
(concrete syntax)

```
if (x == 0) x = x + 1;
```

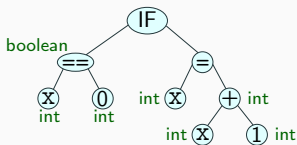
Token Stream

```
if ( x == 0 ) x = x + 1 ;
```

Abstract Syntax Tree
(AST)

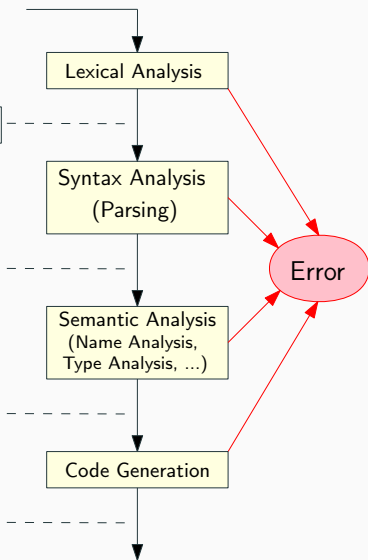


Attributed AST



Machine Code

```
16: iload_2
17: ifne 24
20: iload_2
21: iconst_1
22: iadd
23: istore_2
24: ...
```

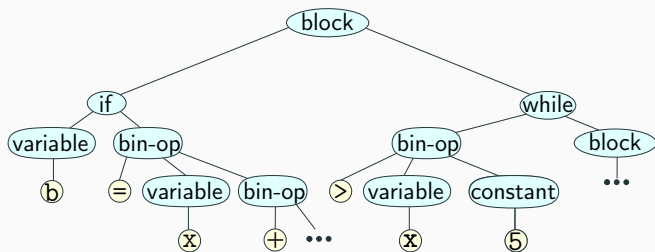


Syntax Analysis: Example

Source Code
(token stream)

```
{  
    if(b) x = x + y;  
    while (x > 5) {  
        System.out.println(x);  
        x = x - 1;  
    }  
}
```

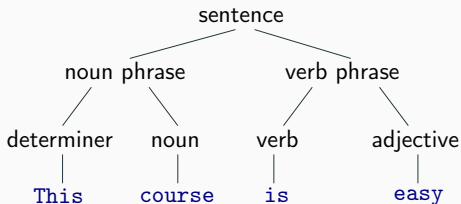
Abstract
Syntax Tree
(AST)



Abstract syntax tree removes extra syntax (e.g. parenthesis)

Syntax Analysis Analogy

Analogy: for natural languages recognize whether a sentence is grammatically well-formed



Syntax Analysis Scope

- Parsing only checks syntax correctness
- Several important inspections are deferred until later phases
 - e.g. semantic analysis is responsible for type checking

Program with correct syntax:

```
int x = true;      // type not agree
int y;            // variable not initialized
x = (y < z);      // variable not declared
```

Overview of Syntax Analysis

- **Input:** Stream of tokens
- **Output:** Abstract Syntax Tree (AST)

What we need for syntax analysis:

- Expressive description technique: describe the syntax
- Acceptor mechanism: determine if input token stream satisfies the syntax description

For lexical analysis:

- Regular expressions describe tokens
- Finite Automata is acceptor for regular expressions

Specifying Language Syntax

- First problem:
 - how to describe language syntax precisely and conveniently
- Regular expressions can describe tokens expressively
- Regular expressions are
 - easy to implement
 - efficient by converting to DFA
- Why not use regular expressions (on tokens) to specify programming language syntax?

Limits of REs

- Programming languages are not regular:
cannot be described by regular expressions
- Consider nested constructs (blocks, expressions, statements)
- Example: language of balanced parentheses is not regular
() (()) ()() (()()((()())))
(() ()) (()())
- Problem: acceptor needs to keep track of number of parentheses seen so far (unbounded counting)
- Automaton has finite memory, cannot count

Limits of REs

- Programming languages are not regular:
cannot be described by regular expressions
- Consider nested constructs (blocks, expressions, statements)
- Example: language of balanced parentheses is not regular
() (()) ()() (())()((()))
(())() (())
- Problem: acceptor needs to keep track of number of parentheses seen so far (unbounded counting)
- Automaton has finite memory, cannot count

- **Question:** How can we show that a language is non-regular?
- **Answer:** Pumping Lemma
(refer to Computer Science Theory course)

Context-free Grammars

- We use context-free grammars instead of finite state automata
- A specification of the balanced-parenthesis language using context-free grammar

$$S \rightarrow (S)$$

$$S \rightarrow SS$$

$$S \rightarrow \epsilon$$

- If a grammar accepts a string, there is a **derivation** of that string using the rules of the grammar

$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow (())S \Rightarrow (())(S) \Rightarrow (())()$$

A context-free grammar is a 4-tuple $G = (T, N, S, R)$ where

- T : token or ϵ
- N : Non-terminal symbols: syntactic variables
- S : Start symbol: special non-terminal
- R : Production rule of the form LHS \rightarrow RHS
 - LHS: single non-terminal
 - RHS: a string of terminals and non-terminals

- Vertical bar is shorthand for multiple production rules
- We abbreviate

$$S \rightarrow p$$

$$S \rightarrow q$$

- as $S \rightarrow p \mid q$

- Production rule specifies how non-terminals can be expanded
- A derivation in G starts from the starting symbol S
- Each step replaces a non-terminal with one of its right hand sides
- Language $L(G)$ of a grammar G :
set of all strings of terminals derived from the start symbol

Exercise

Give context-free grammars that generate the following languages under $\Sigma = \{0, 1\}$

1. all strings that contain at least three 1s

2. all strings with odd length and the middle symbol 0

Exercise

Give context-free grammars that generate the following languages under $\Sigma = \{0, 1\}$

1. all strings that contain at least three 1s

$$S \rightarrow X1X1X1X$$

$$X \rightarrow 0X \mid 1X \mid \epsilon$$

2. all strings with odd length and the middle symbol 0

Exercise

Give context-free grammars that generate the following languages under $\Sigma = \{0, 1\}$

1. all strings that contain at least three 1s

$$S \rightarrow X1X1X1X$$

$$X \rightarrow 0X \mid 1X \mid \epsilon$$

2. all strings with odd length and the middle symbol 0

$$S \rightarrow 0S0 \mid 0S1 \mid 1S0 \mid 1S1 \mid 0$$

RE is a subset of CFG

- Inductively build a production rule for each regular expression operator

ϵ	$S \rightarrow \epsilon$
a	$S \rightarrow a$
R_1R_2	$S \rightarrow S_1S_2$
$R_1 R_2$	$S \rightarrow S_1 S_2$
R_1^*	$S \rightarrow S_1S \epsilon$

where

- G_1 : grammar for R_1 , with start symbol S_1
- G_2 : grammar for R_2 , with start symbol S_2

Derivation Example

- Grammar:

$$S \rightarrow E + S \mid E$$

$$E \rightarrow \text{number} \mid (S)$$

- Derive: $(1 + 2) + 3$

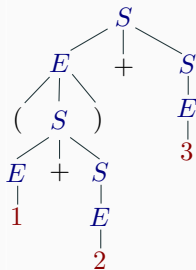
$$S \Rightarrow E + S \Rightarrow (S) + S \Rightarrow (E + S) + S$$

$$\Rightarrow (1 + S) + S \Rightarrow (1 + E) + S \Rightarrow (1 + 2) + S$$

$$\Rightarrow (1 + 2) + E \Rightarrow (1 + 2) + 3$$

Derivation \Rightarrow Parse Tree

- Parse Tree: tree representation of derivation
- Leaves of tree are terminals
- Internal nodes: non-terminals
- No information on order of derivation steps



Derivation

$$\begin{aligned} S &\Rightarrow E + S \Rightarrow (S) + S \Rightarrow (E + S) + S \\ &\Rightarrow (1 + S) + S \Rightarrow (1 + E) + S \Rightarrow (1 + 2) + S \\ &\Rightarrow (1 + 2) + E \Rightarrow (1 + 2) + 3 \end{aligned}$$

Another Derivation

$$\begin{aligned} S &\Rightarrow E + S \Rightarrow (S) + S \Rightarrow (E + S) + S \\ &\Rightarrow (E + E) + S \Rightarrow (E + E) + E \Rightarrow (1 + E) + E \\ &\Rightarrow (1 + 2) + E \Rightarrow (1 + 2) + 3 \end{aligned}$$

Example

Consider the grammar $G = (\{a, b\}, \{S, P, Q\}, S, R)$ where R is:

$$S \rightarrow PQ$$

$$P \rightarrow a \mid aP$$

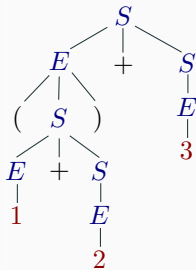
$$Q \rightarrow \epsilon \mid aQb$$

Show a derivation tree for

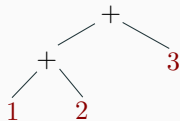
aaaabb

Show at least two derivations that correspond to that tree.

Parse Tree vs. AST



Parse Tree (Concrete Syntax)



Abstract Syntax Tree
Discards nonessential information