450 COMPILERS

COMPUTER SCIENCE

News

- Android Studio 2.0
  - For more info …
    - http://android-developers.blogspot.com/2016/04/android-studio-2-0.html
- React v. 15.0
  - For more info …
    - https://facebook.github.io/react/blog/

Administrivia

- All Labs should be graded and returned.
  - Let me know if you submitted a lab and did not receive a grade

Review

- Compilers
  - A program or set of programs that transforms source code written in a programming language (source language) into another computer language (target language)
- Lexicon
  - The vocabulary of a language
    - An analogy is to the dictionary of the English language

Review Continued

- Syntax
  - the set of rules, principles, and processes that govern the structure of sentences in a given language

Compilers

- Lexical Analysis
- Syntax Analysis
- Semantic Analysis
- Intermediate Code Generation
- Code Optimizer
- Code Generation

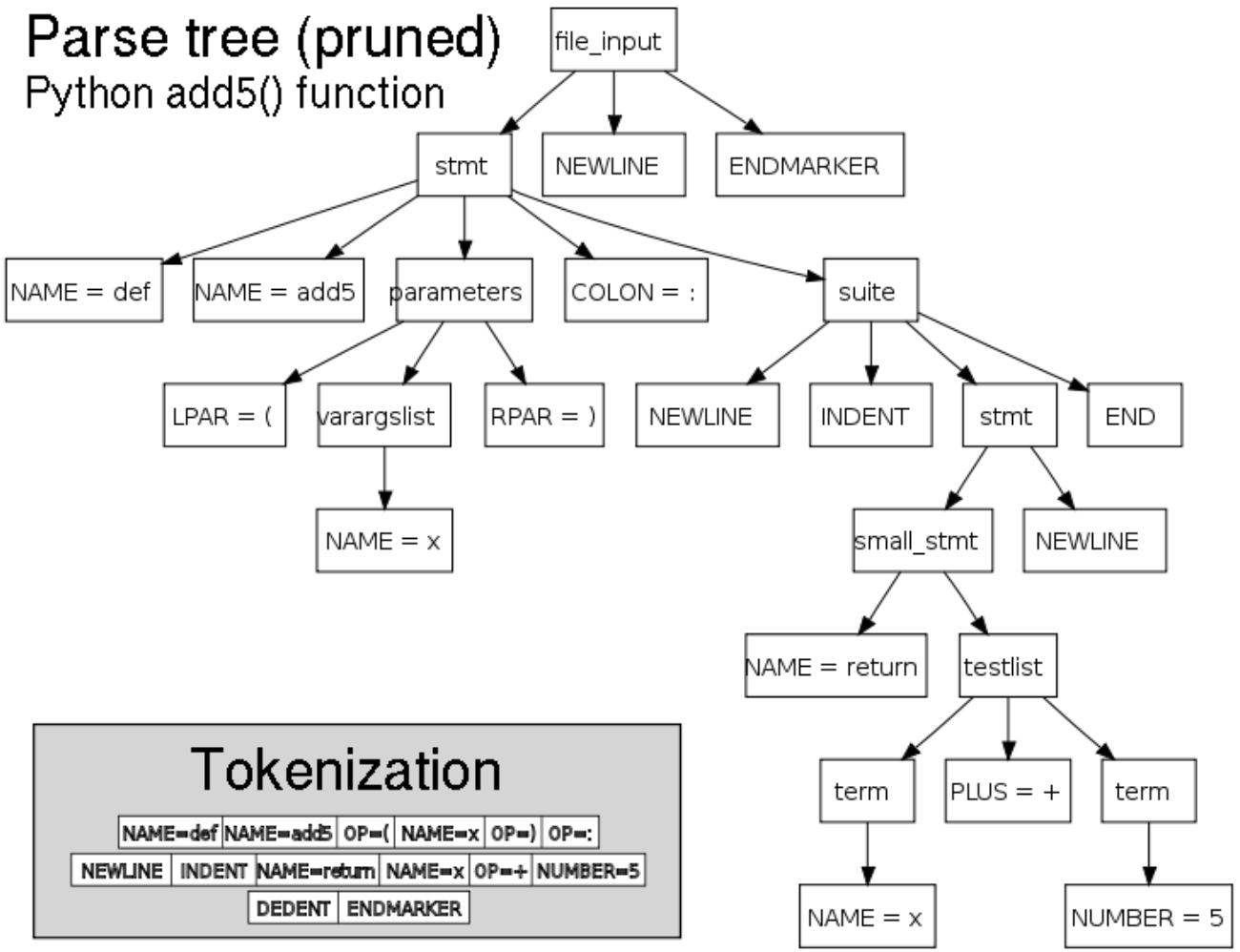
Lexical Analysis

- Also known as linear analysis
- Groups stream into tokens
- Ex. in C/C++
  - position = initial + rate * 60;
- Groups tokens
  - identifier position
  - Assignment symbol =
  - Identifier initial
  - Plus sign +
  - Identifier rate
  - Multiplication sign *
  - Integer 60

Syntax Analysis

- Also known as Hierarchical analysis or parsing
- Grouping tokens into grammatical phrases and are represented as a parse treez

Parse Tree


Parse tree (pruned)
Python add5() function
file_input
stmt
NEWLINE
ENDMARKER
NAME = def
NAME = add5
parameters
COLON = :
suite
LPAR = (
varargslist
RPAR =)
NEWLINE
INDENT
stmt
END
NAME = x
small_stmt
NEWLINE
NAME = return
testlist
term
PLUS = +
term
NAME = x
NUMBER = 5
Tokenization
NAME=def NAME=add5 OP=(NAME=x OP=) OP=:
NEWLINE INDENT NAME=return NAME=x OP=+ NUMBER=5
DEDENT ENDMARKER

Semantic Analysis

- Checks the source program for semantic errors and gathers type information for code generation
- Uses hierarchical structure determined by syntax analysis to identify operators and operands

Context-Free Grammar

- Also known as grammar
- A grammar describes the hierarchical structure of programming language constructs
- Example in C
  - If(exression) statement; else statement;
- To define this a rule
  - stmt -> if(expr) stmt else stmt
- Read as, Statement “can have form of”
- Called a Production

Context-Free Grammar

- Four Components
- 1) A set of tokens, known as terminal symbols
- 2) A set of nonterminals
- 3) A set of productions where each production consists of a nonterminal, called the left side of the production, an arrow, and a sequence of tokens and/or nonterminals, called the right side of the production
- 4) A designation of one of the nonterminals as the start symbol

Terminal vs Non-Terminal

For instance, the following represents an integer (which may be signed) expressed in a variant of Backus–Naur form:

```
<digit> ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
<integer> ::= ['-'] <digit> {<digit>}
```

In this example, the symbols (-,0,1,2,3,4,5,6,7,8,9) are terminal symbols and <digit> and <integer> are nonterminal symbols.

Note: This example supports strings with leading zeroes like "0056" or "0000", as well as negative zero strings like "-0" and "-00000".

Syntax-Directed Translation

- The attributes associated with the constructs needed for a giving programming language
- Syntax-Directed definitions are used for specifying translations for programming language constructs

Syntax-Directed Definitions

- Uses a context-free grammar to specify the syntactic structure of the input
- Each grammar symbol, it associates a set of attributes, and with each production, a set of semantic rules for computing values of the attributes associated with the symbols appearing in that production
- The grammar and the set of semantic rules constitute the syntax directed definitions

Parsing Methods

- Lookahead symbol
  - is the current symbol being scanned; i.e., the leftmost terminal of the input string.
- Top-down Parsing
  - is a preorder (VLR) construction of the parse tree. This corresponds to a leftmost derivation. If you have constructed a parse tree you have performed top-down parsing.

Parsing Methods continued

- Recursive Descent Parsing
  - is a top-down parsing methodology in which the syntax analyzer is based directly on the BNF grammar describing the language (i.e., the syntax of the language). An alternative to recursive descent is to use a parsing table to implement the BNF rules. Both are LL algorithms (the first 'el' means left-to-right scan and the second 'el' means leftmost derivation)

Parsing Methods continued

- Predictive parsing
  - (covered in depth in chapter 4) is a form of recursive descent in which the lookahead symbol unambiguously determines the flow from LHS to RHS of the BNF rules. Without predictive parsing, top-down algorithm may involve backtracing if the incorrect rule is selected.

Translator for Simple Expressions

- An abstract syntax tree (AST) is a parse tree where the interior nodes are programming constructs rather than nonterminals. The interior nodes in a parse tree are nonterminals.

Symbol Table

- Either the scanner or the parser must create an entry in the symbol table for each identifiers in the program. The scanner creates the entry in the symbol table for the current token if it can. The parser must do so in cases where the scanner is not smart enough to so so - such as those that involve scope. Details of symbol table management in later chapters.