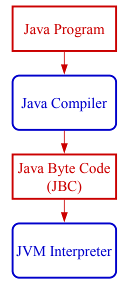
Week 2
Lexical Analysis and Parsing

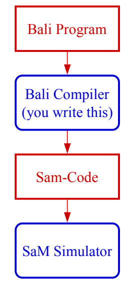
Paul Chew
CS 212 – Spring 2004

Recall

- Compiling Java



- Compiling Bali



Compilers

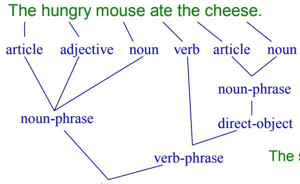
- Basically, a compiler
  - translates one language (e.g., Java)
  - into another (e.g., JBC: Java Byte Code)
- Why do this?
  - Idea is to translate a language that is easy for humans to understand into one that is easy for a computer to understand
  - This idea was initially controversial!
- Typical compiler phases
  - Lexical analysis
    - Breaking input into *tokens*
  - Parsing
    - Understanding program's structure
  - Code Generation
    - Creating code in a *simpler* language (e.g., JBC)
  - Optimization
    - Making the code more efficient (e.g., precomputing constant expressions, avoid recomputing)

Parts of a *Language*

- Human language
  - alphabet → words → sentences → paragraphs → chapters → book
- Computer language
  - alphabet → tokens → statements → program
- Both types of language have
  - Syntax
    - Structural rules
  - Semantics
    - Meaning

Syntax

- Remember diagramming sentences? This was syntax!
  - sentence → noun-phrase verb-phrase
  - noun-phrase → article [adjective] noun
  - verb-phrase → verb direct-object
  - direct-object → noun-phrase
- The hungry mouse ate the cheese.



The shiny elbow drank the automobile.

Syntax vs. Semantics

- Syntax = structure
  Semantics = meaning
- Legal syntax does <u>not</u> imply valid meaning
- Examples of semantic rules for a programming language
  - Variables must be declared before use
  - Division by zero causes an error
  - The then-clause is executed only if the if-expression is True
- It's relatively easy to define valid syntax (especially if we get to invent the language)
- It's harder to specify semantics
- How can we specify semantics?
  - Formally, using logic (*axiomatic semantics*)
  - Informally, using explanations in English
  - By reference to a canonical implementation

Compiling Overview

- Compiling a program
  - Lexical analysis
    - Break program into tokens
  - Parsing
    - Analyze token arrangement
    - Discover structure
  - Code generation
    - Create code
- For a computer language, each phase can be completed before the next one begins
- Understanding a sentence
  - Lexical analysis
    - Break sentence into words
  - Parsing
    - Analyze word arrangement
    - Discover structure
  - Understanding
    - Understand the sentence
- For human language, there is feedback between parsing and understanding

Lexical Analysis

- Goal: divide program into *tokens*
- Tokens
  - Individual units or words of a language
  - Smallest element in a language that conveys meaning
  - Examples: operators, names, strings, keywords, numbers
- Tokens can be specified using regular expressions

a^* = repeat *a* zero or more times

a^+ = repeat *a* one or more times

[*abc*] = choose one of *a*, *b*, or *c*

? = matches any one character

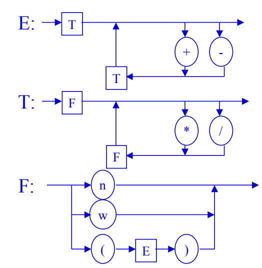
- Examples
  - operator = [+ - * /]
  - integer = $[0123456789]^+$
- For Bali, we give you the *lexical analyzer* (or *tokenizer*)

Building a Tokenizer

- For tokens, can tell what to do next by checking a few characters (usually 1 character) ahead
  - Example: If it starts with a letter, it's a word; the word ends when you reach a non-alphanumeric character
  - Example: If it starts with a digit, it's a number; if you reach a decimal point, it's a floating point number,…
- Java has a class java.io.StreamTokenizer
  - Can recognize identifiers, numbers, quoted strings, and various comment styles
  - Strangely, it can't recognize a number in scientific notation (6.02E23)
- Early computer languages were not parsed based on tokens

Specifying Syntax

- How do we specify syntax?
  - Can use a *grammar*
  - Can use a *syntax chart*
- Example grammar (anything in single-quotes is a token; n and w represent a number token and a word token, respectively; parentheses are used for grouping; | indicates choice; brackets indicate optional)
  - E → T [('+' | '-') E]
  - T → F [('*' | '/') T]
  - F → n | w | '(' E ')'
- Example syntax charts (anything in a rounded box is a token)



Grammars

- The rules in a grammar are called *productions*
- Syntax rules can be specified using a *Context Free Grammar*
  - All productions are of the form V → w
  - V is a single *nonterminal* (i.e., it's not a token)
  - w is word made from *terminals* (i.e., tokens) and nonterminals
- In simple examples, uppercase is used for nonterminals, lowercase for terminals
- Example (ε represents the empty string):

  A → ε

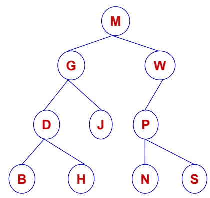
  A → aAb
- A grammar defines a *language*
  - Language of example: all strings of the form a^nb^n for $n \geq 0$
- CS 381 for more detail

Building a Parse Tree

- Grammars can be used in two ways
  - A grammar defines a language
  - A grammar can be used to parse a *sentence* (thus, checking if the sentence is *in* the language)
  - For us,
    - We will give you the grammar for Bali
    - The *sentence* is a Bali program
- You can show a sentence is in a language by building a *parse tree* (much like diagramming a sentence)
- Example: Show that 8+x/5 is a valid Expression (E) by building a parse tree
  - E → T [('+' | '-') E]
  - T → F [('*' | '/') T]
  - F → n | w | '(' E ')'

Tree Terminology

- M is the *root* of this *tree*
- G is the *root* of the left *subtree* of M
- B, H, J, N, and S are *leaves*
- P is the *parent* of N
- M and G are *ancestors* of D
- P, N, and S are *descendents* of W
- A collection of trees is called a *??*



Syntactic Ambiguity

- Sometimes a sentence has more than one parse tree

  $S \rightarrow A \mid aaB$

  $A \rightarrow \varepsilon \mid aAb$

  $B \rightarrow \varepsilon \mid aB \mid bB$

  - The string aabb can be parsed in two ways
- This kind of ambiguity sometimes shows up in programming languages

if E1 then if E2 then S1 else S2

- This ambiguity actually affects the program's meaning
- How do we resolve this?
  - Provide an extra non-grammar rule (e.g., the *else* goes with the closest *if*)
  - Modify the grammar (e.g., an if-statement must end with a '*fi*')
  - Other methods (e.g., Python uses amount of indentation)
- We try to avoid syntactic ambiguity in Bali