Programming Language Paradigms Lecture 3

Nicolas Andrew Burnett

August 30, 2021

1 Chapter 1 (cont)

Influence of Language Design

- Major influence is computer architecture. von Neumann influenced imperative languages, used today
- Programmming design methodologies also influence language design
  - top-down design. lead to work on type checking and structured programming
  - data-oriented design methodologies. lead to abstract data-types, object-oriented programming
  - concurrent programming. shifted focus back to procedural-oriented design

Language Categories

- imperative (java, c++, python)
- functional (scheme, haskal)
- logic
- object-oriented

Implementation Methods Compilation

- starts with the source
- sent to lexical analysis (detects keywords, create groups of characters)
- sent to syntax analysis (analyze groups and ensure order is correct)
- sent to intermediate code (turned into low-level code, but not machine code. may also optimize)
- generates machine code (takes intermediate code and converts to machine code, generates executable)

Pure Interpretation

- Runs exactly what it's given. Executes while parsing.

Hybrid

- starts from source
- lexical analysis
- sytax analysis
- intermediate code (considered 'virtual machine code')
- intermediate code is fed into an interpretor.

2 Chapter 3

Chapter 3 will formally define the formation and semantics of a program.

Syntax

- **Alphabet:** a set of symbols
- **String:** a sequence of symbols from some alphabet
- **Language:** a set of strings made from some alphabet
- **Sentence:** a string in a language
- **Lexeme:** description of lowest-level syntactic unit
- **Token:** a category of lexemes (dentifiers, keywords, operators, etc)
  - two ways to formally define languages
  - **Recognition:** check if a string is in the language
  - **Generation:** generate a sentence of a language
- **Grammars:** known as generators, tells how to generate a string

Backus-Naur Form

  - context-free grammar, simply sees circumstances and decides what a symbol is
  - **Terminal Symbol:** a symbol from the alphabet, lexeme, or token
  - **Nonterminal Symbol:** a symbol that can be substituted via a production rule

Describing Lists

  - usually utilizes a form of recursion

Derivations

  - start with start symbol
  - apply rule to substitute a Nonterminal
  - repeat until only terminals are left
  - to derive a pecific string, choose the right rules
  - **Sentential Form:** strings in the derivation
  - **Leftmost Derivation:** always choose the left most nonterminal to substitute
  - **Rightmost Derivation:** always choose the right most nonterminal to substitute

Parse Trees

  - lose order of substitute information (no leftmost or rightmost)
  - gain hierarchical structure information
  - any informational order is lost