CS 350: Programming Language Design

Lecture 7

Language implementation systems analyze source code. Regardless of the specific implementation approach.

Nearly all syntax analysis is based on a formal description of the syntax of the source language (BNF)

Advantages of using BNF to describe

Implementation methods

* Compilation
  + Translate high-level program into machine code
  + Slow translation, fast execution
  + Several phases
    - Lexical analysis, converts source characters into lexical units
    - Syntax analysis, transforms lexical units into parse trees which represent the syntactic structure of programs
    - Semantics analysis, generate intermediate code
    - Code generation, machine code is generated
* Pure Interpretation
  + No translation
    - Interpreter is a VM with fetch-decode-execute cycle
  + Easier implementation of programs
    - Run time errors can easily and immediately be displayed
  + Slower execution (10 to 100 times)
    - Due to statement decoding
  + Now rare for traditional high-level languages
    - Comeback for JS and Python
* Hybrid Implementation
  + A compromise between interpretation and compilation
  + A high-level language program is translated to an intermediate language that allows easy interpretation
  + Faster than pure interpretation
  + Example: Java JIT

Syntax Analysis

* Consists of two components
  + A low-level lexical analyzer
  + A high-level syntax analyzer or parser
* Reasons to separate the two analyzers
  + Simplicity, simple processes can be used for lexical analysis, this cannot be achieved with combined analyzers
  + Efficiency – separation allows optimization of the lexical analyzer, you could not optimize both to greatest affect when combined since there are differences in optimization
  + Portability, parts of the lexical analyzer may not be portable when combined with the syntax analyzer
* Lexical Analysis
  + A lexical analyzer is a pattern matcher for character strings
    - Front-end of parser
  + Identifies substrings of the program that belong together “lexemes”
    - Lexemes match a character pattern which is associated with a category of lexemes, which are called by a token.
  + May
    - Skip comments
    - Skip blanks
    - Inset lexemes for idenitfiers and literals into a symbol table
    - Detects syntactic errors in lexemes, like bad literals or identifiers
* The Parsing problem
  + Goals of parsing
    - Produce a parse tree
    - Find all syntax