CSE 504: Compiler Design

Introduction

Pradipta De
pradipta.de@sunykorea.ac.kr
Today’s Topic

• Overview of Compilers
  – Anatomy of a compiler
Course Textbook

• Engineering a Compiler
  – Keith Cooper and Linda Torczon

• References for a specific class will be shared during lecture
Purpose of a Compiler

- Translates high level language
  - To machine instructions
  - To other high level language

- Can I compile natural language, like English?
  - Natural language grammar can be ambiguous
  - Compiler can translate language with an unambiguous grammar
Compiler vs. Interpreter

**Compiler:**
- a program that translates an executable program in one language into an executable program in another language
- Example: C, C++

**Interpreter:**
- a program that reads an executable program and produces the results of running that program
- Example: Perl, PHP

**Compiled and Interpreted:**
- Java
  - Java source compiled to bytecode, and bytecode is interpreted (can be compiled also called JIT compilation)
- Python

Interpretation or compilation are implementation techniques, not attributes of a language.
Two-pass Compiler Design

- **Front-end**
  - Maps legal source code into Intermediate Representation (IR)
- **Back-end**
  - Maps IR into target machine code

- Front-end techniques are well established with good solutions \([O(n) \text{ or } O(n\log n)]\)
- Back-end problems are more challenging with some problems being NP Complete
Three-pass Compiler Design

- **Optimizer**: Typically a middle stage which analyzes and transforms the IR to improve it
- **Optimizer can use multiple passes over IR**
- **Helps in generating target program with one or more objectives**
  - Faster execution time
  - Low energy consumption
  - Reduce page faults
Anatomy of a Compiler
Anatomy of a Compiler: Front-End

• Scanner:
  – Check syntax of the program
  – Generates tokens (words and its type) from the character streams
  – Tokens: numbers, identifiers, +, -

• Parser:
  – Checks if the input stream follows the grammar rules
    ⇒ verifies if a sentence is valid
  – Helps in checking the semantics done after parsing (Type Checking stage)
Anatomy of a Compiler: Back-end

- Instruction Selection
  - Chooses target machine instructions to generate executable code
  - There can be multiple choice of instructions to accomplish the same task

- Register Allocation
  - Target machine has limited set of registers → how to use the registers during execution

- Instruction Scheduling
  - Execution time of different operations can vary → actual instruction execution may need to consider machine architecture
Summary

• Overview of different parts of a typical compiler
  – Front-end, Optimizer, Back-end
  – Scanner, Parser, Instruction Selection, Register Allocation, Instruction Scheduling