CSE 504: Compiler Design

Semantic Analysis
Type System

Pradipta De
pradipta.de@sunykorea.ac.kr
Current Topic

- Semantic Analysis
  - What is the use of this phase?
- Type Systems
- Attribute Grammar
- Adhoc Syntax directed translation
Semantic Analysis

- Final phase of the front-end of a compiler
- Must catch any error in the program structure before the machine code is generated
  - Last chance to catch program error

- Lexical analysis and Parsing catches syntax error, but there are structures beyond syntax that cannot be expressed using Context Free Grammar
  - Semantic Analysis step attempts to catch remaining errors of structure after parsing
Possible errors

- Use of undeclared variables
  - What if the language does not require declarations?
- Duplicate declaration of variables
- Mismatch in the type of expressions and variables
  - $x = y$, and $x = z$ are different if $x$, $y$ are integers, but $z$ is a string
- Which declaration of a variable to use when it is used in an expression (Scope rules)
- Inheritance Relationships
- Function argument mismatch
  - How many parameters does the function expect
  - Does the arguments match in type

Discovering correct answer to these questions or catching such errors require contextual knowledge in the source code ➔ semantic analysis phase discovers meaning beyond syntax
Why not CFG?

- CFG deals with syntactic categories, not the specific values or words
  - Grammar specifies the position where a variable name can occur ➔ parser validates if a word is allowed in a specific position

- Grammar cannot match one instance of a variable name to another
What is a type?
- Abstract category that specifies properties held in common by all its members
  - Integers, boolean, character, string, list

Types can be specified by rules
- Rules specify the collection of properties for each category

The set of types in a programming language, along with the rules that use types to specify program behavior, are collectively called a type system

Type System is used for – safety, expressiveness, and runtime efficiency
- Avoid type errors
- Is it possible to catch all type related errors?
Type Inference

• The process of determining a type for each name and each expression in the code
• Must check types of operands of each operator against the rules that the language allow
  – May allow to convert value from one representation to another
  – In Java, c = a + b, where a is int, and b, c are double
    ➔ a is converted to double

• What if the language does not require type declaration before use of the variable?
  – Do runtime type inference and checking
Types of Type Checking

• Static Type Checking
  – Analyze program at compile time to ensure absence of errors

• Dynamic Type Checking
  – Check operations at runtime to avoid errors

• No Type Checking
  – As it says, assume the program does not have type errors
Expressiveness

• Type systems allow a language to specify behavior more precisely than CFG

• Operator Overloading
  – An operator has different meaning based on the type of its arguments
Type Checking

• Analyze the program to assign a type to each named variable, and each expression

• Two separate activities
  – Type inference
  – Type error detection
Different Types

• Typical programming languages support
  – Set of base types or built-in types
  – Rules for constructing new types from base types
  – Method to determine if two types are equivalent or compatible
  – Rules for inferring types of each expression

• Base Types
  – Numbers, characters, Booleans

• Compound Types
  – Arrays, Strings, Enumeration, Structures, Pointers
Type Equivalence

• Name equivalence: Two types are equivalent iff they have the same name

• Structural equivalence: two types are equivalent iff they have the same structure

```
struct Tree {
    struct Tree *left;
    struct Tree *right;
    int value
}

struct STree {
    struct STree *left;
    struct STree *right;
    int value
}
```
Inference Rules

- Inference Rules specify for each operator the mapping between the operand types and the result type
  - Simple rules ➔ left-hand side of an assignment must have same type as right hand side
  - Implicit conversions
    - If LHS is less precise than RHS, then type error (can override with a cast operation)

- How to assign type to constants?
  - Constant’s form implies type
    - 2 is integer, but 2.0 is floating point number
  - Infer type of constant based on usage
    - \( \text{sin}(2) \) implies that 2 is a floating point number
Infer Type for Expressions

• Function calls add complexity while determining type of expressions

• Must ensure that each actual parameter is type compatible to formal parameters
  – Programmer can define a “Type signature”, like function prototypes in C

• A more complex scenario: parametric polymorphism
  – Result type is a function of the argument types

• Choices for type inference
  – Entire program is presented for compilation as a unit
  – Requires a mandatory function prototype
  – Defer type checking until link time or runtime
  – Embed the compiler in a program development environment such that requisite information is available for type inferencing