1. Introduction (Types and Programming Languages)

Kwanghoon Choi

Software Languages and Systems Laboratory
Chonnam National University

Week 1

1.1 Background: Software Engineering

To make software correctly work and to make it smoothly evolve are very difficult and very expensive.

Two main approaches to tame software monsters are

- empirical software engineering based on *experiences*
- *formal methods* based on *mathematics and logics*

In this class, we are interested in formal methods so called *type systems*.

1.1 Background: Formal Methods

A broad range of *formal methods* in software engineering

Some powerful frameworks at one end of the spectrum

- Hoare logic, Algebraic specification languages, Modal logics, and Denotational semantics

Pros: They can express very general correctness properties

Cons:

- But they are often cumbersome to use and demand a good deal of sophistication on the part of programmers

1.1 Background: Formal Methods (cont.)

A broad range of *formal methods* in software engineering

Lightweight formal methods at the other end of the spectrum

- Model checkers, Run-time monitoring, *Type systems*, etc.

Cons:

- Much more modest power than the previous powerful frameworks

Pros:

- But modest enough that automatic checkers can be built into compilers
- and thus be applied even by programmers unfamiliar with the underlying theories

1.1 Types in Computer Science

In this class, a type system is

- a tractable syntactic method
- for proving the absence of certain program behaviors
- by classifying phrases according to the kinds of values they compute

$\Rightarrow$ Type systems as tools for reasoning about programs.

$\Rightarrow$ Type systems as calculating a kind of *static approximation* to the certain run-time behaviors of the terms in a program.

But type systems (or *type theory*) refers to a much broader field of study in logic, mathematics, and philosophy, dating back to the early 1900s as ways of avoiding the logical paradoxes, such as Russell's.

1.2 What Type Systems Are Good For

Detecting errors

Abstraction

Documentation

Language safety

Efficiency

Further applications : Automated theorem proving such as with Coq, Isabelle, Agda

1.3 Type Systems and Language Design

Programming language design go hand in hand with type-system design

The type system itself is the foundation of the design and the organizing principle in typed programming languages

Even in untyped programming languages, they are structured in a way for which a type system can be written.

In summary, there are no programming languages without type systems!

1.4 Capsule History

The earliest type systems to make very simple distinctions between integer and floating point representations of numbers in Fortran (1950s)

This classification was extended to structured data (arrays, records, etc.) and to higher-order functions (1960s)

A number of even richer concepts (parametric polymorphism, abstract data types, module systems, and subtyping) were introduced (1970s).

Computer scientists began to be aware of the connections between the type systems found in PLs and those studied in mathematical logic!

1.5 Related Reading

Many useful, interesting, and mind-blowing textbooks about types and programming languages