CS 160 Compilers

Lecture 12: Type Checking II

Yu Feng

Fall 2021

Outline

- We will talk about Patina

Motivation

- When writing programs, everything is great as long as the program works.

- Unfortunately, this is usually not the case

- Programs crash, don't compute what we want them to compute, etc.

- This is arguably the biggest problem software faces today

Software correctness

- Problem: Rice's th  ny non-trivial property about a Turing machine is ble

- This means that we can never give an algorithm, that for all programs can decide if this program has an error on some inputs.

- What can we do?

Big idea

- Big Idea: Just because we cannot prove something about the original program does not mean we cannot prove something about an *abstraction* of the program.

- Strategy: In addition to the operational semantics, we will also define *abstract semantics* that will overapproximate the states a program is in.

- Example: In λ^+, the operational semantics compute a concrete integer or list, while our abstract semantics only compute the if the result is of kind integer or list.

Abstraction

- Of course, any abstraction will be less precise than the program
- One popular abstraction: types

- Let's assume we have types Int and List
- Example: let x = 10 in x

- Operational semantics yield concrete value 10

- Abstract semantics that only differentiate the kind (or type) of the expression yield: Integer

Abstraction

- But we don't just want any abstraction, we need abstractions that *overapproximate* the result of the concrete program

- Recall the example: let x = 10 in x

- Abstract value *Integer* overapproximates 10 since 10 is a kind of integer

- On the other hand, abstract value *Zero* does not overapproximate 10.

Soundness

- The reason we only [illegible]t sound abstract semantics is the following:

- Theorem: If some abstract semantics are sound and an expression is of abstract value x, then its concrete value y is always part of the abstract value x.

- Why is this useful?

- This means that if a program has no error in the abstract semantics, it is guaranteed not to have an error in the concrete semantics.

- ASTREE tools: http://www.astree.ens.fr/

Types

- In this class, we will look at one kind of abstraction: types
- This means abstract values are the types in the language

- What is a type? An abstract value representing an (usually) infinite set of concrete values

- Question: For proving what kind of properties are types as abstract values useful?

- Answer: To avoid run-time type errors!

Inference rules

$$\frac{\text{Hypothesis 1} \quad \ldots \quad \text{Hypothesis N}}{\text{Conclusion}}$$

- This means“given hypothesis1,…N, the conclusion is provable”

$$\frac{\text{Mid-term grade} >= 70 \quad \ldots \quad \text{Final grade} >= 140}{\vdash \text{Final grade: A}}$$

TODOs by next lecture

- HW3 is out today