Retargeting Gradual Typing

Ross Tate

Cornell University, Ithaca, NY, USA
ross@cs.cornell.edu

Abstract

Gradual typing is often motivated by efforts to add types to massive untyped code bases. A major challenge here is the fact that these code bases were not written with types in mind, yet the goal is to add types to them without requiring any significant changes in their implementation. Thus, critical to this application is the notion that gradual typing is being added onto a preexisting system.

But gradual typing also has applications in education, prototyping, and scripting. It allows programmers to ignore types while they are learning programmatic reasoning, while they are experimenting with new designs, or while they are interacting with external systems. At the same time, gradual typing allows these programmers to utilize APIs with types that provide navigable documentation, that concisely describe interfaces, and that enable IDEs to provide assistance. In these applications, programmers are working with types even when they are not writing types. By targeting just these applications, we can lift a major burden from gradual typing. Rather than being added to something that already exists, here gradual typing can be integrated into the software-development process, into the core language design, and into the run-time environment, with each component designed to support gradual typing from conception.

This retargeting provides significant flexibility, enabling designers to tradeoff various capabilities of gradual typing. For example, a designer might choose to require some minor annotation burden in untyped programs for, say, a hundred-fold improvement in run-time performance. For the past half decade I have been exploring gradual typing behind the scenes in both academia and industry, and I will be presenting my experiences with these design tradeoffs so far.

1998 ACM Subject Classification D.3.1 [Programming Languages]: Formal Definitions and Theory – Semantics; D.3.2 [Programming Languages]: Language Classifications – Object-oriented languages; D.3.4 [Programming Languages]: Processors – Run-time environments; F.3.3 [Programming Languages]: Studies of Program Constructs – Type structure

Keywords and phrases Design, Efficiency, Gradual Typing, Nominal Types

Digital Object Identifier 10.4230/LIPIcs.ECOOP.2017.3

Category Invited Talk