Short Guide to Papers

Ben Greenman *October 3, 2019*

Here's a brief summary of the technical contributions in the papers on migratory typing that I've worked on. Click on the venue of a paper to open a PDF.

POPL '16 Introduces a method to evaluate the performance of a migratory typing system; specifically, the method is for programmers who want to add types to an arbitrary set of modules in their program. Evaluates Typed Racket on a suite of benchmark programs.

JFP '19 (major revision of POPL '16) Compares the performance of three implementations of migratory typing and introduces a sampling variant of the evaluation method. Evaluates Typed Racket on an extended suite of benchmark programs; the new programs use object-oriented designs.

PEPM '18 Evaluates Transient Reticulated Python; notes major differences from Typed Racket. On one hand, the worst observed overhead is within one order of magnitude. On the other hand, the fully-typed version of a program is typically the slowest. Typed Racket suffers more in the worst case but fares better when fully-typed.

ICFP '18 Presents a systematic, two-part investigation of the design space of migratory typing systems. The first part models different designs as different semantics for a common mixed-typed language, compares their type soundness guarantees, and present additional examples that are not explained by type soundness. The second part compares the performance of natural, erasure, and transient migratory typing as implementations of the Typed Racket surface language; this evaluation confirms the conjectures from PEPM '19.

OOPSLA '19 Continues the theoretical investigation of ICFP '18 with a formal account of differences that were not explained by type soundness. Type soundness captures shallow behavioral properties; it says little about systems that compose typed and untyped components. The paper adapts complete monitoring (from prior research on higher-order contracts) to test whether a language protects all channels of communication between typed and untyped code.

OOPSLA '18 Uses the POPL '16 method to measure the benefit of an optimization that folds stacks of higher-order contracts into a single tree-shaped contract. The collapsing method was implemented by Daniel Feltey and invented by Michael Greenberg. Performance is still poor in many benchmarks after collapsing; the optimization is irrelevant in some benchmarks, and needs to be implemented for additional higher-order contracts to support others.

DLS '18 Uses the ICFP '18 model to survey developers' preference regarding three different semantics for migratory typing (natural, transient, and erasure). Respondents preferred a semantics that enforces all types. The survey did not ask about performance.