S3D Software and Societal Systems Department

Polarized Subtyping

Thesis Proposal $\downarrow\sigma^{-}$
PhD in Software Engineering $\uparrow\tau^{+}$

Zeeshan Lakhani

August 16th, 2024

1 PM - 4 PM

GHC 6501

Committee Members

Prof. Frank Pfenning (chair),
Prof. Jan Hoffmann,
Prof. Jonathan Aldrich,
Prof. Ronald Garcia (UBC)

Abstract

Choices made in programming language and type system design involve tradeoffs between safety, simplicity, extensibility, and usability. Extensible features centered on type structure, like subtyping or mixing evaluation regime (lazy/eager) in functional programming, are often avoided or aborted due to the complexity of efficient type inference or the need for additional syntactic layers. Mainstream languages like TypeScript forsake soundness to capture typing scenarios for *all* of JavaScript and its now lengthy history of existent programs.

In this thesis, we introduce **Polarized Subtyping**, a flexible structural subtyping system grounded in the call-by-push-value paradigm, separating the language of types into two layers: a positive layer characterized by inductively defined, eagerly evaluated, observable values and a negative layer characterized by coinductively defined, lazily evaluated, possibly infinite computations---with adjoint modalities (or shifts) mediating between them. We extend the underlying call-by-push-value calculus with a decidable equirecursive subtyping variant that is both structural and semantic, forming a higher-order type system combining unfettered recursion, variant and lazy records, consequential property types like intersections, unions, and type difference, and (eventually) parametric polymorphism with subtyping and the interaction with effects at the heart of all these constructs.

Our approach moves beyond the traditional confines of syntactic type soundness by championing a semantic characterization of typing with step-indexing to capture the observation depth of recursive computations, from which we can immediately derive a form of semantic subtyping. This approach offers advantages in understanding complex type system features and maintaining behavioral safety when encountering varying evaluation regimes, nontermination, and computational effects.

Being explicit about values and computations while making subtyping first-class in our system opens up new possibilities for reasoning about the properties of functional programs and how type structure affects subtyping relations and enables compilation optimizations. Furthermore, to make the system practical, we present Polite, a reference implementation that demonstrates the feasibility of our approach. Polite gives us a platform to experiment with various type system features and optimizations in the future.