Breaking records: structural subtyping as a language design principle

A pre-abstract for FABRIC

Dynamically typed languages like Python or JavaScript are in widespread use. Their main advantage is flexibility, as the same code can be freely reused for different types, with the programmer unconstrained by a static type checker. Despite the usefulness of static type systems – like those in the ML family – dynamic and static disciplines have been at odds for decades.

The design of statically typed languages primarily focuses on *nominal types* with limited subtyping – simplifying problems like type inference and program optimisation. Meanwhile, dynamically typed languages essentially rely on *structural types* with implicit subtyping (often termed *duck typing*). Thus, the real static-dynamic divide is even deeper than it might first seem: it has also been co-mingled with nominal-structural types and presence of nuanced subtyping.

Nevertheless, there have been many recent developments to bring static type systems to dynamically typed languages – either through optional typing (like in Python), or typed dialects (like TypeScript for JavaScript). In practice, these attempts suffer from a confused relationship between nominal and structural types. TypeScript lacks proper support for nominal types – crucial for some abstractions – while Python's typing discipline often prefers them over structural types (in contrast to programs in practice). These existing systems have also failed to satisfactorily type workloads common in dynamic languages, such as arrays and dataframes. In this work we show we can address these by embracing structural types. We conclude there is a need to not only reconcile the predictability of static languages with the flexibility of dynamic ones, but also to give particular consideration to both nominal and structural types.

Thanks to the recent development of theory of *algebraic subtyping* it has become possible to bring ML-style type inference to languages with static typing and structural subtyping. It is a milestone bridging the design gap between the static and dynamic disciplines, enabling annotation-free checking of more expressive programs.

I advocate for a language design with a focus on structural subtyping, while acknowledging the crucial role of nominal types. I present the design of the statically typed FABRIC language and its prototype compiler targeting the modern WEBASSEMBLY standard. FABRIC features expressive structural types for arrays, and user-extensible inference rules for checkable *properties*. The compiler emits efficient representations for structural record and variant types. FABRIC's design approach enjoys the safety and performance of static languages, while also limiting the loss in expressivity of dynamic languages and preserving their extensibility.

Jakub Bachurski <jkb55>