Designing Modular Type Inferences with Subtyping and Principal Typings

Axel Simon

The Hindley-Milner type system forms the basis of most languages that provide type inference. These inferences are often extensions of the Damas-Milner type inference algorithm $\mathcal{W}$. This algorithm is known to produce principal types but not principal typings, that is to say, it is incomplete for polymorphic recursive functions but able to type check these if a type signature is provided. Henglein resolved this deficiency using semi-unification, thereby creating a type inference that delivers principal typings.

While Henglein's work is widely known, only few have noticed that his inference rules are incorrect with respect to the scoping of type variables, creating bogus equality relations. We thus turn to an invited paper by Cousot in which abstract interpretation is used to derive a type inference algorithm by abstracting the denotational semantics into a type semantics. We observe that this paper is wrong in that it neglects equality relations between type variables and thus derives a type system that is correct by construction but not that of Hindley and Milner. We sketch a fix and conclude that the Hindley-Milner type system is inherently complex due to it being flow insensitive but relational. Finally, we identify what constitutes a modular type inference that infers principal typings and propose a flow-sensitive, complete, modular type inference that is polymorphic and allows for sub-typing.