Discrete Polymorphism with Dynamic Types

Pedro Ângelo Mário Florido

Faculdade de Ciências da Universidade do Porto

Gradual typing [3, 1] enables dynamic and static typing in the same language, by extending an existing type system with a dynamic (?) type and consistency ($\sim$) relation that compares gradual types. We can choose between disciplines by inserting a dynamic type annotation in dynamically typed parts of the program, dellaying type checking to the evaluation following an operational semantics which explicitly checks types. It is also important to note that polymorphism, both Hindley-Milner and parametric are supported in gradual typing.

The intersection type system [2] is a counterpart to the Hindley-Milner system, regarding polymorphism. By allowing expressions to be typed with a range of different types, the system provides discrete polymorphism. However, this system is more expressive than the Hindley-Milner system, due to being able to type more expressions, self application for example. Intersection types are of the form $T_1 \cap \ldots \cap T_n$, and the type system for intersection types extends the simply typed lambda calculus to deal with intersections.

In this work we extend the intersection type system with gradual typing, resulting in a system that contains all the expressive power of the intersection type system, and all the advantages of gradual typing. In our system, the type ? may be used in the type connective $\cap$, which allows a single expression to be typed with dynamic and static types simultaneously. For example, the expression $(\lambda x : Int \cap ? \ . \ x + x)\ 1$ has type Int, however, the left expression in the application types with an intersection type: $\lambda x : Int \cap ? \ . \ x + x$ types with $Int \cap ? \rightarrow Int$ and 1 types with Int. This type system adheres to some correctness criteria presented in [1], such as conservative extension and monotonicity w.r.t. precision.

References

[1] Matteo Cimini and Jeremy G Siek. The gradualizer: a methodology and algorithm for generating gradual type systems. *ACM SIGPLAN Notices*, 51(1):443–455, 2016.

[2] Mario Coppo, Mariangiola Dezani-Ciancaglini, et al. An extension of the basic functionality theory for the λ-calculus. *Notre Dame journal of formal logic*, 21(4):685–693, 1980.

[3] Jeremy G Siek, Michael M Vitousek, Matteo Cimini, and John Tang Boyland. Refined criteria for gradual typing. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 32. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.