Type Inference and Effects

Jonathan Hoag

March 19, 2010

1 Operational Semantics and Polymorphic Type Inference

In his PhD thesis[2], Tofte presents a modification of Milner’s polymorphic type inference system to extend a purely applicative λ-language with imperative features. He introduces the use of imperative type variables such that types in the system now contain information about the store. The potential generalization of let-bound variables is now determined by inference rules that incorporate a distinction between applicative and imperative type variables. Tofte proves the soundness of his type system using operational semantics.

While not the first, Tofte successfully articulates the problems that arise from the generalization of the type of variables that are bound to reference cells. In particular, he identifies the expressions whose syntactic shape lead to the expansion of the store and those that are always safe to generalize over. Tofte’s solution is conservative in the sense that it still types all Hindley-Milner typeable expressions, however it suffers from an inability to compute some polymorphic procedures or data structures.

2 Polymorphic Type Inference and Assignment

In his first publication on closure typing[1], Leroy provides a sound type system that extends the class of let-bound variables that can be generalized and given a polymorphic type. In contrast to Tofte’s solution, the creation of polymorphic references are not recorded and instead the use of a reference cell with two different types is prohibited. To accomplish this, the types of expressions are extended with constraints that maintain information about the store. If a type variable is free in the type of a reference then it is considered a “dangerous” type and is not generalized over.

Leroy’s work, along with many others during this time period, focuses on extending the ability to compute polymorphic procedures. This system establishes the fact that it is only necessary to prohibit the generalization of type variables that appear in the types of references reachable after the binding has been evaluated. Interestingly, types incorporating information about the store and other statically determined effectful information motivates the formulation of many other static analyses involving effects.
3 Simple Imperative Polymorphism

Wright’s paper on simple imperative polymorphism[3] first analyzes the breadth of polymorphic type systems developed for imperative languages and presents a simple solution. By limiting polymorphism to let expressions where the binding is a syntactic value we are assured that their evaluation does not cause any side effects. This restricts the number of Hindley-Milner typeable expressions in the imperative language, however the type of semantically equivalent applicative and imperative procedures are identical. Transformations for existing code from Tofte’s to Wright’s type system are also explored and documented for over 250,000 lines of ML code.

Wright’s type system marks a shift in the perception of the practicality of polymorphic type inference with effects. The ability to assign the same type to equivalent applicative and imperative procedures abstracts the specification of types over the imperativeness of the procedure itself. The key insight was that previous type systems maintained information about the store and essentially “leaked” details of the implementation of imperative procedures into the specification. His simple system, while limiting the class of programs that are typeable, provided and elegant solution to problems that deal with the specification of procedures and maintained the inference of many useful polymorphic types.

References

