Propagation of type annotations in Hindler-Milner based type-systems

Didier Rémy

Master internship, 2020

Subject:	Propagation of type annotations in Hindler-Milner based type-systems
Supervisor:	Didier Rémy
Location:	Centre Inria de Paris

Context

Hindley Milner refers to the ML type system which is at the basis of modern languages with type inference such as Haskell or OCaml. Type inference for ML is based on first-order unification, which can be performed in arbitrary order, with the addition of prenex polymorphism introduced by let-bindings which impose a synchronization between typechecking (the polymorphic part) of the bound expression and typechecking its body.

However, several extensions of ML that have been proposed departs from Hindley-Milner and rely on the propagation of type information that is present in the source during the type inference process, or even on types of previously inferred phrases:

- *First-class polymorphism* relies on the type generalization mechanism to tell whether polytypes are known, independent of the typing context and can be instantiated—or just being inferred, still in the typing context, and cannot be instantiated yet (Garrigue and Rémy, 1999).

  Type annotations can be used to turn an *inferred*, frozen polytype into a *known*, instantiable polytype.

  Types annotations need not be written exactly at the node that really needs them: placed above, they will be automatically passed to subexpressions as their expected type during the generation of typing constraints.

- In OCaml, optional labeled arguments also use source type annotations (and previously inferred types) to help detect missing optional arguments.

- Overloading of records labels, which has been recently introduced, uses contextual type information for disambiguation.
- GADTs introduces type equalities, treated as type abbreviations, with limited scope, which are a source of ambiguity, since equivalent forms within the scope of an equality may become incompatible when exiting its scope (Garrigue and Rémy, 2013).

  Type annotations are again used to resolve such ambiguities.

Each of these extensions has been defined independently, more or less precisely, sometimes formally studied, but in separate works; they have also been implemented in the OCaml type-checker, which is not yet using a modern constraint-based approach (Pottier, 2014).

Internship description

Motivated by the plan to redesign the implementation of the OCaml type-checker, the goal of this internship is to revisit propagation of type information in the context of Hindley Milner, so as to give it a proper formal status and provide a reference constraint-based implementation (Pottier and Rémy, 2005; Pottier, 2014).

The solution should cover all scenarios described above (even though all underlying features need not be present), with as much control as possible to the way type information can be propagated.

Other forms of type propagation could also be considered. For example, the Haskell language now uses bidirectional type checking that is limited but in ad-hoc ways so that only source program annotations are being used and, more recently, so that propagation does not depend on the order of arguments in multi-argument applications. This later extension actually uses a local form of shape inference, proposed earlier (Pottier and Régis-Gianas, 2006).

The propagation of existing type annotations may also be used to propagate previously inferred types such as the types of imported modules or types of previous toplevel phrases. Interestingly, such a mechanism may perhaps also allow a simpler treatment of first-class polymorphism.

References

Jacques Garrigue and Didier Rémy. Extending ML with semi-explicit higher-order polymorphism. *Information and Computation*, 155(1/2):134–169, 1999. URL `http://www.springerlink.com/content/m303472288241339/`. A preliminary version appeared in TACS'97.

Jacques Garrigue and Didier Rémy. Ambivalent Types for Principal Type Inference with GADTs. In *11th Asian Symposium on Programming Languages and Systems*, Melbourne, Australia, December 2013.

François Pottier. Hindley-Milner elaboration in applicative style. In *ACM SIGPLAN International Conference on Functional Programming (ICFP)*, September 2014. doi: http://dx.doi.org/10.1145/2628136.2628145. URL http://gallium.inria.fr/~fpottier/publis/fpottier-elaboration.pdf.

François Pottier and Yann Régis-Gianas. Stratified type inference for generalized algebraic data types. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 232–244, Charleston, South Carolina, January 2006. doi: http://doi.acm.org/10.1145/1111037.1111058. URL http://gallium.inria.fr/fpottier/publis/pottier-regis-gianas-popl06.pdf.

François Pottier and Didier Rémy. The essence of ML type inference. In Benjamin C. Pierce, editor, *Advanced Topics in Types and Programming Languages*, chapter 10, pages 389–489. MIT Press, 2005. URL http://cristal.inria.fr/attapl/.