Sharing and Unsharing in Hindley Milner

Didier Rémy

Master internship, 2020

Subject:	Sharing and unsharing in Hindley Milner
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. For the core language, the type system is an extension of simple types with toplevel polymorphism; although its worst-case complexity is exponential, its type inference is quite efficient in practice, provided types are *shared* during unification, typically using using union-find algorithms to represent equivalence classes—and type generalization is performed locally.

However, many important extensions of the ML type system that have been proposed play with type sharing in significant ways.

The simplest form of sharing comes from *type abbreviations*: they allow the introduction of type synonyms, which should be expanded on demand to respect type equalities, but memorized to preserve efficiency and retain type abbreviations whenever possible when presenting types to the user. This has been recently studied (Morel, 2019) and should constitute a solid basis to study other form of type sharing, which is the objective of the internship.

- *Object types* make a crucial use of type abbreviations: it is a key for readability that the inferred types retain as much as possible the name of the class they belong to; object types are recursive by nature, and their recursive definitions are carefully unfolded so that the open view of the object type can be directly derived from its graph representation (Rémy and Vouillon, 1998; Vouillon, 2000).

- *First-class polymorphism* relies on the ML generalization mechanism to tell whether polymorphic types are known, now independent of the typing context and can be instantiated—or just being inferred and cannot be instantiated yet. To gain in expressiveness, type representations should be unshared as much as possible at generalization sites (Garrigue and Rémy, 1999)—by contrast with object types.

- GADTs introduce type equations that behaves much as type abbreviations, but with local scoping, which makes type representations ambiguous when exiting the scope of equations. Ambivalent types, introduced to separate true ambiguities from accidental ones, rely on type sharing (Garrigue and Rémy, 2013).

- The language of modules may introduce equalities between an abstract type and its implementation, which may itself be concrete or abstract. This generate type equivalences that are key to the typeckecking of modules. Recursive modules, raises the double-vision problem that requires an additional form of equality where internally one module knows altogether: its internal concrete view, its identity seen through the over module, and its fully opaque external view, which should all be equated.

Each of these extensions has been precisely defined and formally studied, but in separate works; they have also been implemented in the OCaml type-checker, but not in a modern constraint-based applicative implementation of type inference (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 sharing and unsharing in the context of Hindley Milner, so as to give it a proper formal status, cover all difference usages, and provide a reference constraint-based modular implementation, possibly in applicative style.

Type inference will return an explicitly typed term. Notice however, that sharing is also a problem with explicit typing, since the size of type annotations may grow in the square of the size of the untyped-program (Jay and Peyton Jones, 2008). Hence, we should also care about sharing in the explicitly typed language, for instance by introducing a let-binding notation in types. Although not a piority, this could also be explored.

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.

Barry Jay and Simon Peyton Jones. Scrap your type applications. In *Mathematics of Program Construction (MPC'08)*, July 2008. URL `https://www.microsoft.com/en-us/research/publication/scrap-your-type-applications/`.

Carine Morel. Type inference and modular elaboration with constraints for ML extended with type abbreviations. Master's thesis, Université Paris Diderot-Paris 7, September 2019. URL `https://hal.inria.fr/hal-02361707`.

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`.

Didier Rémy and Jérôme Vouillon. Objective ML: An effective object-oriented extension to ML. *Theory And Practice of Object Systems*, 4(1):27–50, 1998. A preliminary version appeared in the proceedings of the 24th ACM Conference on Principles of Programming Languages, 1997.

Jérôme Vouillon. *Conception et realisation d'une extension du langage ml avec des objets*. PhD thesis, Université Paris Diderot, 2000. URL `http://www.theses.fr/2000PA077234`.