Liquid Types Revisited

Mário Pereira, Sandra Alves, and Mário Florido

University of Porto, Department of Computer Science & LIACC,
R. do Campo Alegre 823, 4150-180, Porto, Portugal

Refinement types [2] state complex program's invariants, by simplying augmenting type systems with logical predicates. A refinement type of the form $\{\nu : B \mid \phi\}$ stands for the set of values from basic type B restricted to the filtering predicate (refinement) ϕ. However, the use of arbitrary boolean terms as refinement expressions leads to undecidable type systems, both for type checking and inference.

Liquid Types [4] (*Logically Qualified Data Types*) present a system capable of automatically infer refinement types, by means of two main restrictions to a general refinement type system: every refinement predicate is a conjunction of expressions exclusively taken from a global, user-supplied set of logical qualifiers (simple predicates over program variables, the value variable ν and the variable placeholder $\star$); and a conservative (ence decidable) notion of subtyping.

The Liquid Types system is defined as an extension to the Damas-Milner type system, with the term language extended with an `if-then-else` constructor and constants. A key idea behind this system is that the refinement type of every term is a refinement of the corresponding ML type.

We propose a refinement type system based on Liquid Types, with the addition of intersection types [1]. Our intersections are at the refinement expressions level only, i.e. for the type $\sigma \cap \tau$ both σ and τ are of the same form, solely differing in the refinement predicates. As an example, the identity function $id = \lambda x.x$ (considered to act only over integer values) could be typed within our system as $(x : \{\nu : int \mid \nu \geq 0\} \rightarrow \{\nu : int \mid \nu \geq 0\}) \cap (x : \{\nu : int \mid \nu \leq 0\} \rightarrow \{\nu : int \mid \nu \leq 0\})$. Our use of intersections for refinement types draws some inspiration from [3].

With our type system we are able to derive more precise types than in the original system, leading to a detailed description of programs' behaviour.

References

1. Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *The journal of symbolic logic*, 48(4):931–940, 1983.
2. E. Denney. Refinement types for specification. In David Gries and Willem-Paul Roever, editors, *Programming Concepts and Methods PROCOMET '98*, IFIP — The International Federation for Information Processing. Springer US, 1998.
3. Tim Freeman and Frank Pfenning. Refinement types for ML. In *Proceedings of the ACM SIGPLAN 1991 Conference on Programming Language Design and Implementation*, PLDI '91, pages 268–277, New York, NY, USA, 1991. ACM.
4. Patrick M. Rondon, Ming Kawaguci, and Ranjit Jhala. Liquid types. In *Proceedings of the 2008 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '08, pages 159–169, New York, NY, USA, 2008. ACM.