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

Abstract

We present a new type system combining refinement types ideas and the expressiveness of intersection type discipline. The use of such features makes it possible to derive very precise types, using the types language itself as a detailed description for programs' functional behaviour. We have been able to prove several interesting properties for our system (including subject reduction) and started the development of an inference algorithm, which was proved sound.

1 Motivation

Refinement types [2, 4] state complex program invariants, by 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. A subtyping relation exists for refinement types, which will generate implication conditions (much like a VCGen in the context of program verification):

$$\frac{\Gamma; \nu : B \vdash \phi \Rightarrow \psi}{\Gamma \vdash \{\nu : B \mid \phi\} <: \{\nu : B \mid \psi\}}$$

One idea behind the use of such type systems is to perform type-checking using SMTs (Satisfability Modulo Theories), discharging conditions as the above $\phi \Rightarrow \psi$. However, the use of arbitrary boolean terms as refinement expressions leads to undecidable type systems, both for type checking and inference.

Liquid Types [5, 6] (*Logically Qualified Data Types*) present a system capable of automatically infer refinement types, by means of two main restrictions to a system: every refinement predicate is a conjunction of expressions exclusively taken from a global, user-supplied set (denoted $\mathbb{Q}$) of logical qualifiers (simple predicates over program variables, the value variable ν and the variable placeholder $\star$); and a conservative (hence 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.

Despite the interest of Liquid Types, some situations arise where the inference procedure infers poorly accurate types. For example, considering $\mathbb{Q} = \{\nu \geq 0, \nu \leq 0\}$ and the term $\text{neg} \equiv \lambda x. -x$, Liquid Types system infers $\text{neg} :: x : \{\nu : int \mid 0 \leq \nu \wedge 0 \geq \nu\} \rightarrow \{\nu : int \mid 0 \leq \nu \wedge 0 \geq \nu\}$ (the syntax $x : \tau \rightarrow \sigma$ is here preferred over the usual $\Pi(x : \tau).\sigma$ for functional dependent types). This type cannot, at all, be taken as a precise description of the neg function's behaviour.

2 Intersection-refinement types

We propose a refinement type system 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. We introduce a new rule to form intersections on types (our typing relation is denoted by $\vdash^{\cap}$):

$$\textsc{Intersect}\quad \frac{\Gamma \vdash^{\cap} M : \sigma \qquad \Gamma \vdash^{\cap} M : \tau}{\Gamma \vdash^{\cap} M : \sigma \cap \tau}$$

As an example, the neg function could be typed within our system as

$$(x : \{\nu : int \mid \nu \geq 0\} \to \{\nu : int \mid \nu \leq 0\}) \cap (x : \{\nu : int \mid \nu \leq 0\} \to \{\nu : int \mid \nu \geq 0\})$$

Our use of intersections for refinement types draws some inspiration from [3].

We use a standard call-by-value small step operational semantics to define the evaluation relation, denoted $\rightsquigarrow$. Based on this relation, we were able to prove the following result:

Theorem 1 (Subject reduction)**.** *If* $\Gamma \vdash^{\cap} M : \sigma$ *and* $M \rightsquigarrow N$ *then* $\Gamma \vdash^{\cap} N : \sigma$.

Based on the Liquid Types restrictions, we conceived an algorithm to infer appropriate refined types with intersections. We use the set $\mathbb{Q}$ to restrict the possible refinements of types and to guarantee that the algorithm terminates. We define the inference algorithm in terms of a program M, a typing context Γ and $\mathbb{Q}$ as $\texttt{Infer}(\Gamma, M, \mathbb{Q}) = \sigma$, where σ is an intersection-refined type. The following result holds:

Theorem 2 (Soundness)**.** *If* $\texttt{Infer}(\Gamma, M, \mathbb{Q}) = \sigma$ *then* $\Gamma \vdash^{\cap} M : \sigma$.

With the use of intersections we are able to derive more precise types than in a classical refinement type system. These types can thus be taken as detailed descriptions of programs' behaviour.

We are currently investigating completeness properties of our inference algorithm. We take particular interest in realizing if our algorithm infers *most-general types* by means of subtyping. In other words, we would like to prove that if $\texttt{Infer}(\Gamma, M, \mathbb{Q}) = \sigma$ and $\Gamma \vdash^{\cap} M : \sigma'$, for some σ', then $\sigma <: \sigma'$.

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] Ewen Denney. Refinement types for specification. In *Programming Concepts and Methods PROCOMET'98*, pages 148–166. Springer, 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] Kenneth Knowles and Cormac Flanagan. Hybrid type checking. *ACM Trans. Program. Lang. Syst.*, 32(2):6:1–6:34, February 2010.

[5] 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.

[6] Niki Vazou, Patrick M. Rondon, and Ranjit Jhala. Abstract refinement types. In *Proceedings of the 22Nd European Conference on Programming Languages and Systems*, ESOP'13, pages 209–228, Berlin, Heidelberg, 2013. Springer-Verlag.