Refinement Types, Annotated Bibliography

Kevin Clancy
Northeastern University

T. Freeman and F. Pfenning. Refinement types for ML. In PLDI, 1991.

The first refinement system attempts to solve a classic ML programming conundrum: a case expression is used in a context where the form of the scrutinee is restricted, and so a set of match patterns which is exhaustive in context is consider inexhaustive by the type checker. An extended type definition language for ML allows the programmer to decompose each discriminated union data type into a lattice of refinements. Limited forms of intersection and union types, as well as bounded universal refinement quantifiers, enable precise type checking. A type inference technique based on abstract interpretation is outlined.

Xi, H. Dependent ML. An approach to practical programming with dependent types. Journal of Functional Programming 17.02 (2007): 215-286.

Xi presents the first "dependent" refinement type system. Here a base type is refined using one or more terms of an typed *index language* for expressing computationally solvable constraints. Type checking and subtyping are then defined with respect to a context including an environment ϕ of index variables and a set $\vec{P}$ of constraints on those variables. A refined base type is considered a subtype of another when their indices can be proven equal under the current constraint set $\vec{P}$.

Vazou, N. et al. Refinement types for Haskell. ACM SIGPLAN Notices. Vol. 49. No. 9. ACM, 2014.

Vazou describes a comment-embedded refinement type system for Haskell called Liquid Haskell. In Liquid Haskell, a base type is refined with a boolean program term. Using program terms rather than index terms to refine base types results in a simpler system than DML, but also one that seems to require the sharp distinction between pure and impure code provided by Haskell. Being a lazy language, a typing context in Haskell represents variables bound to arbitrary terms rather than values. Because terms may diverge, using their refinements in the reasoning process leads to inconsistency. Vazou's solution involves a stratified type system, which determines whether a term may diverge, converges to a

value in head-normal Form, or converges to a finite value. Only the refinements of converging terms may be used as assumptions by the constraint solver.

Zeilberger, N. July 2016. Principles of Type Refinement, Chpts 1 - 2.2.3. Retrieved April 4, 2017, from http://noamz.org/oplss16/refinements-notes.pdf

This tutorial describes refinement types, a type systems paradigm in which an existing system without subtyping is conservatively extended for extra precision. Each base type from the underlying system is decomposed into a preordered set of refined base types. Types formed from refined base types and composite type constructors, called *refinement types*, are used to identify subsets of types of the underlying system. A subtyping relation among refinement types then allows the specification of sharper constraints.