Refinement Types

Christos Dimoulas

1 Refinement Types for ML

Freeman and Pfenning propose an extension to ML’s type language that allows programmers to define subtypes of datatypes and thus express invariants of their programs that are beyond the expressive power of the ML type language. Programmers can use a special form of recursive datatype definitions to specify a refinement of an existing datatype by imposing restrictions to the arguments of its constructors. The authors demonstrate how the specifications of refinement types can be turned into a lattice that on its turn becomes the basis for sound refinement type inference.

Evaluation

The idea is innovative but it turns out to be impractical. The refinement types of functions grow exponentially when the number of refinement type declarations increase. The brute-force treatment of polymorphic functions also leads to huge refinement types. The use of abstract interpretation for inferring refinement types in combination with the size of function types turns the approach infeasible. To support the last argument, the longest program that type-checked in a reasonable time is 50 lines long. Finally, the system of the paper is so restricted that it fails to infer a refinement type even for the motivating example of the paper.

2 Practical Refinement-type Checking
**Content**  Davies continues the programme of Pfenning for the development of a practical refinement type system for ML. He abandons refinement type inference and tries out refinement type checking. Programmers have to provide refinement types for every function they define. The author shows that his bi-directional refinement type checker is sound for mini-ML. He, also, investigates extensions of his system that can lift his result to full SML and validates his approach through experiments.

**Evaluation**  The result is not practical enough. The author successfully handles mutation and pattern matching but other issues remain with parametric datatypes, exceptions, polymorphism and layered pattern matching. It is still unclear if refinement types, even after requiring function annotations, can be seen as a conservative addition to ML. Finally, the experimental part is weak and the analysis of the empirical results is inconvincing.

### 3 Refinement Types for Logical Frameworks

@inproceedings{p:refine-logic-frameworks,
author = Frank Pfenning,
title = Refinement Types for Logical Frameworks,
book = Workshop on Types for Proofs and Programs,
pages = 285--299,
year = 1993}

**Content**  LF lacks sub-typing, which poses severe problems concerning the encoding of some deductive systems. Pfenning proposes the introduction of refinement types to gain some controlled form of sub-typing. He shows that type checking remains decidable even after the addition of refinement types and that the extension leads to a more natural encoding of deductive systems.

**Evaluation**  The system has not been implemented and thus no empirical results are provided on its efficiency or its limitations. More specifically, it is unclear if the restrictions imposed on the use of refinement types severely damage the usefulness of the system. Finally, it remains open whether and how refinement types affect the adequacy proofs for encodings.