Mechanizing the Metatheory of Standard ML

Daniel K. Lee Karl Crary Robert Harper
Carnegie Mellon University

Abstract

A rigorous definition of a programming language consists of a precise specification of its static and dynamic semantics. One benefit of a rigorous definition of a language is that it permits a proof of type safety, in the sense that the static semantics rules out classes of dynamic errors in programs. Although many safety proofs exist for various core calculi, none exist for any full-scale programming language, due to the daunting complexity of such languages. For the languages we actually use, we settle for much less: at most, a general agreement that the language's core aspects have been studied carefully, and that any errors the might exist in the language as a whole must be minor. Thus it is unsurprising that numerous minor errors have been uncovered in supposedly type-safe languages.

The goal of this work is to give a fully rigorous, machine-checked proof of type safety for Standard ML. Although the proof must deal with the full complexity of the language, machine checking gives us confidence that we have done so correctly, confidence that it would be nearly impossible to earn for a pencil-and-paper proof. Since parts of the Definition of Standard ML are given too informally for rigorous treatment, we base our effort on *elaborative semantics,* in the sense of Harper and Stone.

This elaborative semantics consists of three parts: (1) a relation elaborating Standard ML into an explicitly typed internal language, (2) a static semantics for the internal language, and (3) a small-step operational semantics for the internal language. Given this semantics, the type safety proof of Standard ML is expressed in two parts: (1) a proof that elaboration results in well-typed internal programs, and (2) a type safety proof for the internal language.

In this paper we describe our formalization of our internal language and its safety proof in Twelf. By necessity, the internal language simultaneously supports all the programming mechanisms needed to implement Standard ML (*e.g.,* translucent modules, abstraction, polymorphism, higher kinds, references, exceptions, recursive types, and recursive functions). Our successful formalization of the proof involved a careful interplay between the precise formulations of the various mechanisms, and required the invention of new representation and proof techniques of general interest. We utilize an algorithmic formulation of type equality for which certain crucial properties (such as inversion principles for type equality) are easily established. Verifying the equivalence of the algorithmic and declarative formulations of type equality was a significant portion of the overall proof.

In our ongoing and future work we are formalizing in Twelf the elaboration of Standard ML into the internal language, and proving that elaboration results in well-typed internal programs.