Mechanizing the Metatheory of a Language With Linear Resources and Context Effects

Daniel K. Lee
Carnegie Mellon University

Derek Dreyer Andreas Rossberg
Max Planck Institute for Software Systems

We present a mechanized formalization and type safety proof for the variant of Dreyer's RTG language [2] (for *Recursive Type Generativity*) used in Dreyer and Rossberg's work on ML-style mixin modules [3]. This is a core module calculus with the ability to forward declare type variables and then later give them (possibly recursive) definitions. The static semantics of RTG involves tracking the definability of these type variables, which is a linear resource — for soundness reasons, a type variable may only be defined once. Once a type variable is defined, the knowledge of its definition must be added to the context for typechecking subsequent code. The linear nature of definability, combined with the use of context effects to add definitions for existing type variables, means that the mechanized encoding of the metatheory of RTG is not obvious.

Our mechanization uses the Twelf proof system. For the most part, we employ standard techniques for formalizing λ-calculi using HOAS. However, for handling type definability and type definitions, we needed to develop more advanced techniques.

Type definability is a linear resource. In order to ensure that a module defines an abstract type exactly once, we employ a judgment `defonce ([a] M a)`, which identifies when `M a` is a module that defines `a` only once. Within `M a`, the variable `a` can also appear in other non-definition sites, such as type ascriptions, an arbitrary number of times. While our use of the `defonce` judgment is a standard trick, we additionally need to impose well-formedness constraints on the context, which imply that a type variable is not both defined and definable, and that there is no more than one definition for a given type variable. Because Twelf's world checking does not track such restrictions, we cannot use standard HOAS encodings for definability and definitions.

Instead, we use a variation of Crary's explicit context technique [1], in which variables of the object language are represented using meta-level variables, but assumptions about those variables may be more flexibly manipulated. However, whereas Crary uses explicit contexts for typing assumptions, we use explicit contexts only to track information about definability and definitions, and continue to represent typing assumptions in the standard Twelf manner. This application of explicit contexts has a decidedly different flavor from Crary's, since in our setting the contexts appear in the mechanization of the type system itself (not merely as a proof device in the metatheory).

In the process of this development, we discovered a significant technical simplification to Crary's technique. In Crary's presentation, the well-formedness of explicit contexts is enforced by premises in typing rules. We tried to adopt a similar approach initially, but found that the necessary bookkeeping would have required typing assumptions to be included in explicit contexts as well. To avoid this, we removed these premises from the typing rules and included them as hypotheses of our meta-theorems instead. This technique, which is a well-known alternative in paper proofs, enabled fairly painless proofs of a number of important lemmas (such as weakening) that are typically quite painful to prove using explicit contexts in LF.

The type system of our language relies on a notion of type equality that admits β-, η-, and (due to type definitions) δ-equality of type constructors. Consequently, certain inversion properties about type equality, such as injectivity and the inequality of different base types, are not directly provable from our declarative formulation of type equality. These inversions are easily provable on paper with logical relations arguments, but due to Twelf's limitations concerning logical relations arguments, we have left these properties as trusted unproven assumptions. The presence of cyclic type definitions means that type constructors do not necessarily have a canonical form, which rules out the hereditary-substitution-based techniques used to prove such inversions for normalizing type languages [4, 5].

In summary, this work demonstrates how to adapt some of the latest Twelf proof techniques to a language with linear resources and context effects. In future work, we intend to use RTG as a case study for comparing the effectiveness of different proof systems. An intern at MPI-SWS is currently developing a Coq mechanization of the RTG metatheory, and we hope to report on a comparison of the two encodings in the near future.

References

[1] Karl Crary. Explicit Contexts in LF. In *LFMTP*, June 2008.

[2] Derek Dreyer. Recursive type generativity. *Journal of Functional Programming*, 17(4-5):433–471, 2007.

[3] Derek Dreyer and Andreas Rossberg. Mixin' up the ML module system. In *ICFP*, 2008.

[4] Daniel K. Lee, Karl Crary, and Robert Harper. Towards a mechanized metatheory of Standard ML. In *POPL*, 2007.

[5] Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A concurrent logical framework: The propositional fragment. In *Types for Proofs and Programs*, volume 3085 of *Lecture Notes in Computer Science*, 2004.