An Agda Formalisation of Modalities and Erasure in a Dependently Typed Language

Oskar Eriksson and Andreas Abel

Department of Computer Science and Engineering, Chalmers and Gothenburg University,
{oskeri, abela}@chalmers.se

Modal types, like their counterparts in logic, allow modifiers to be attached to types. Given different sets of modifiers and different rules for the treatment of such modifiers by the type system, this allows types to be given a range of different interpretations. These interpretations can range from being quantitative in nature [6, 4], expressing, for instance, linearity or erasure [10, 12], to a variety of other interpretations such as data privacy [2]. Often, modal type systems are designed with specific interpretations in mind and the modifiers and rules are chosen based on this interpretation. An example of this is McBride’s modality for erasure and linearity [9]. Others treat modal types in a more general sense and use a more general set of rules to allow different interpretations to be used in the same system.

There are different approaches to generalizing modalities [11, 8]. We have made a formalisation\(^1\) in Agda of one such system for a dependently typed language [7], based on the ideas presented by Abel [1, 2] and Bernardy [2]. A modality is a ring-like structure whose elements are used to annotate terms in order to achieve the desired interpretation. By varying the structure, one can achieve a wide range of different interpretations but the common algebraic properties of the structures are used to define a type system that can check that annotations have been put down correctly.

The Agda formalisation (ca. 26000 lines of code) builds on a formalisation of decidability of type conversion by Abel et al. [3] (ca. 15000 lines of code) with the following novelties:

1. We adapt the syntax and typing judgements with modality annotations.
2. We introduce a new typing judgement for checking the validity of annotations.
3. As a case study, we instantiate it to the erasure modality with extraction to an untyped language.

Most closely related to our work is the Agda formalization by Wood [13] of a simply typed version of our calculus. In comparison to Atkey [5], our calculus also features a weak and a strong Σ-type, but we omit it in the following for lack of space.

The typing judgement for modality annotations relates modality contexts (assigning a modality element to each free variable) with terms. It is defined as follows, making use of the ring-like structure of the modality elements with its operations lifted to act pointwise on contexts \(\gamma, \delta\), a module for this ring.

\[
\begin{align*}
0 \triangleright U & \quad 0 \triangleright N \\
\gamma \triangleright t & \quad \delta \triangleright u \\
\gamma + p\delta \triangleright t^p u \\
\gamma \triangleright F & \quad \delta \triangleright q \triangleright G \\
\gamma + \delta \triangleright \Pi_p^q F G \\
e_i \triangleright x_i & \\
\gamma \triangleright \lambda^p t & \\
\gamma, p \triangleright t & \quad \gamma \triangleright t & \quad \gamma \triangleright t & \quad \delta \leq \gamma
\end{align*}
\]

\(^1\)Available at https://fhkfy.github.io/modalities_and_erasure/Logrel-MLTT.html
The erasure case study considers a modality with two elements, 0 and ω which are used to annotate computationally irrelevant and relevant terms respectively. For instance, λt\(^0\) t represents a function whose argument is not used during computation whereas λt\(^ω\) t represents a function whose argument is. The point of using these erasure annotations is, of course, to achieve a kind of optimisation in which terms that have been marked as erasable can be removed. For this, we use an erasure function \(\bullet\) which translates terms into the untyped lambda calculus while removing erasable terms. Most notably \((t\,^0\, u)\,^\bullet = t\,^\xi\), where the erasable function argument \(u\) is replaced with \(\xi\), representing an undefined value.

If sound, applying this erasure function should not affect the result of fully evaluating a closed term. The proof of soundness makes use of two logical relations. The first, reducibility is a reducible type relation for terms of some type \(F\). For types, it is defined inductively as below. The Π-type case makes use of the reducibility of terms is designed such that a term belongs to the relation iff it reduces to canonical form.

\[
\begin{align*}
\langle U \rangle & \quad \models_\ell U, \ell' < \ell \\
\langle N \rangle & \quad \frac{} {A \to^* N} \quad \models_\ell A \\
\langle \Pi \rangle & \quad \frac{A \to^* \Pi^p F G}{\models_\ell F} \quad \frac{\forall a. (\models_\ell a : F/F) \Rightarrow (\models_\ell G[a])}{\models_\ell A}
\end{align*}
\]

The fundamental lemma for this relation is that \(\models A\) and \(\models t : A\) imply \(\models_\ell A\) and \(\models_\ell t : A/\mathcal{A}\). This relation, in a more general form not restricted to closed terms, was part already in the formalisation by Abel et al. \[3\].

The second relation relates closed terms in the source language with closed terms in the target language, \(t \, \hat{\circ} \, v : A/\mathcal{A}\) and is also defined by recursion on \(\mathcal{A}\), a proof that \(A\) is reducible:

- If \(\mathcal{A} = \langle U \rangle\) then \(t \, \hat{\circ} \, v : A/\mathcal{A}\) holds iff \(\models t : U\).
- If \(\mathcal{A} = \langle N \rangle\) then \(t \, \hat{\circ} \, v : A/\mathcal{A}\) holds iff either
  - \(t \to^* \text{zero} : N\) and \(v \to^* \text{zero}\), or
  - \(t \to^* \text{suc} t' : N\) and \(v \to^* \text{suc} v'\) and \(t' \, \hat{\circ} \, v' : A/\mathcal{A}\).
- If \(\mathcal{A} = \langle \Pi \rangle\), then \(A \to^* \Pi^p F G\) holds and there are derivations \(\mathcal{F} : \models_\ell F\) and \(\mathcal{G} : \forall a. \models_\ell a : F/F \Rightarrow \models_\ell G[a]\). We then define \(t \, \hat{\circ} \, v : A/\mathcal{A}\) to hold iff either
  - \(p = 0\) and \(t\,^0\, a \, \hat{\circ} \, v : G[a]/\mathcal{G}(a)\) for all \(\models_\ell a : F/F\), or
  - \(p = \omega\) and \(t\,^\omega\, a \, \hat{\circ} \, v : G[a]/\mathcal{G}(a)\) for all \(\models_\ell a : F/F\) and \(a \, \hat{\circ} \, w : F/F\).

Especially noteworthy is the case for Π-types where, non-erased function terms, \(t\) and \(v\) are related if they are related when applied to related arguments. For erasable functions, however, the argument on the target language side is replaced with \(\xi\), mirroring the definition of the extraction function.

The fundamental lemma for this relation is that \(\models t : A\) and \(\triangleright t\) imply \(t \, \hat{\circ} \, t^\bullet : A/\mathcal{A}\). Using this property, the extraction function can be shown to be sound. In particular, all terms of type \(N\) represent the same natural number before and after extraction.
References


