The essence of type-theoretic elaboration *

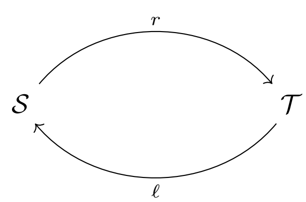
Anja Petković Komel

TU Wien
anja.komel@tuwien.ac.at

When using a type theory in a proof assistant, the syntax can quickly become too verbose to handle. Terms annotated with full typing information are easily amenable to algorithmic processing and have good meta-theoretic properties, but more economic terms that omit typing information are much more usable in practice.

One common solution to this problem is to design two type theories: a fully annotated type theory $\mathcal{S}$ that resides in the kernel of the proof assistant and an economic one $\mathcal{T}$ for the users input. The latter version is then translated to the former via an *elaborator* i.e., the missing information is somehow recovered, usually during or in parallel with type-checking. We can see this process in practice, for example with Agda's [Agd21] or Coq's [Coq21] inferred implicit arguments, termination checking [Abe98] (where evidence of termination is added), or universe polymorphism [ST14] (where explicit universe levels are calculated and constraints checked).

The type-theoretic account of an *elaboration map* can be summarized in the following diagram, which we call the "essence" of elaboration:



We start with the economic type theory $\mathcal{T}$ (a finitary type theory as defined by Haselwarter and Bauer in [HB21]). The fully-annotated type theory to which we elaborate is a standard type theory $\mathcal{S}$ [HB21], in which all specific object rules are symbol rules that faithfully record all the premises, and thus make the theory a good candidate for the kernel. Of course we want our economic version $\mathcal{T}$ to be conservative over $\mathcal{S}$, namely that for every derivable type in $\mathcal{S}$, if we can provide a term of said type in $\mathcal{T}$, there is also a term of the original type.

Next there is a "forgetful" type-theoretic transformation $r\colon \mathcal{S} \to \mathcal{T}$, called the *retrogression transformation*, which erases the annotations, but is still conservative. It is a transformation, that works syntactically on type-theoretic judgements, while preserving their derivability. The interesting part is in the other direction, the so called *elaboration map* ℓ, which acts as a section to the retrogression transformation. But since the economic syntax does not provide sufficient information, the elaboration map takes entire derivations in the economic type theory $\mathcal{T}$ and maps them to judgements in the standard type theory $\mathcal{S}$.

This definition of elaboration map enjoys two important meta-theoretic properties: every finitary type theory has an elaboration map to a standard type theory and it satisfies a universal property, making it unique up-to judgemental equality.

A relationship between the algorithmic content of the elaboration map and type-checking of $\mathcal{T}$ can be described via the *elaborator*: an algorithm, that takes a (not necessarily derivable) judgement J in $\mathcal{T}$ and outputs a derivable judgement J' in $\mathcal{S}$ such that $r_*(J') = J$ if such J' exists, or reports there is none. An elaborator for $\mathcal{T}$ exists if and only if $\mathcal{T}$ has decidable type-checking and equality-checking.

*Joint work with Andrej Bauer.

References

[Abe98] Andreas Abel. foetus - termination checker for simple functional programs. https://www.cse.chalmers.se/~abela/foetus/, 1998.

[Agd21] The Agda proof assistant. https://wiki.portal.chalmers.se/agda/, 2021.

[Coq21] The Coq proof assistant, version 2021.02.2. https://coq.inria.fr/, 2021.

[HB21] Philipp G. Haselwarter and Andrej Bauer. Finitary type theories with and without contexts. Available at https://arxiv.org/abs/2112.00539, 2021.

[ST14] Matthieu Sozeau and Nicolas Tabareau. Universe polymorphism in coq. In Gerwin Klein and Ruben Gamboa, editors, *Interactive Theorem Proving*, pages 499–514, Cham, 2014. Springer International Publishing.