# Observing Definitional Equality

András Kovács
University of Gothenburg & Chalmers University of Technology
Sweden

Suppose we are trying to unify $(t,\, u)$ with $(t',\, u')$ at the $\Sigma$-type $(x : A) \times B\,x$, in an implementation of a dependently typed language. If we get blocked when unifying the first projections, e.g. because it's a problem like $\alpha\,\mathsf{zero} \equiv \beta\,\mathsf{zero}$ for metavariables $\alpha$ and $\beta$, how do we proceed? Here, $t \equiv t'$ is a postponed constraint and we have not yet established that $u$ and $u'$ have the same type. We would like to make as much as possible progress, while preserving soundness and only producing unique solutions.

We can conceptualize the situation as follows. The state of elaboration and unification lives in a two-level type theory, whose object level is the theory we want to elaborate to, and the outer level supports functions and object-level definitional equality, as type formers.[1] This means that we do unification up to the definitional equality of the two-level type theory, and postponed equations are now propositional equalities.

Let's go back to the example. First, assume that unification of $t$ and $t'$ is blocked and returns $p : t \equiv t'$ as a postponed constraint. Now we have $\mathsf{ap}\,B\,p\,:\,B\,t \,\equiv\, B\,t'$, and may try to unify $\mathsf{coe}_{(B\,t)\,(B\,t')}\,u$ and $u'$ (eliding the equality proof in $\mathsf{coe}$). This unification attempt is unlikely to be useful, unless $\mathsf{coe}$ computes in interesting ways. For example, if $B$ is $\lambda\,\_.\,\mathsf{Nat}$ and we're trying to solve $(t,\, \mathsf{zero}) \equiv (t',\, \mathsf{zero})$, $\mathsf{coe}_{\mathsf{Nat}\,\mathsf{Nat}}\,\mathsf{zero}$ may compute to $\mathsf{zero}$ and we can match $\mathsf{zero}$ with $\mathsf{zero}$. Generally, we want $- \equiv -$ to be some kind of *observational equality*.

This idea is not new. Gundry and McBride [3], Gundry [2] and Juan [4] have developed *heterogeneous* unification, where we work with equations of the form $(t : A) \equiv (u : B)$, where $A \equiv B$ is implicitly proven from the context. This is a "dependent equality" lying over a given equality of types. There are rules that decompose equalities between matching type and term formers. If we think of heterogeneous equality as homogeneous equality with an implicit coercion on one side, such rules specify the computation of coercions. This can be viewed as one of the three flavors of proof-irrelevant observational equality in the literature:

1. "John Major" equality as primitive: $(t : A) = (u : B)$ is an existential bundle of a type equality and a term equality [1].
2. Dependent equality as primitive, as we just saw in heterogeneous unification.
3. Homogeneous equality as primitive [5–7].

[1]This is a bit simplified; in practice we probably want a primitive notion of context for contextual metavariables.

Version (2) has some drawbacks when used in an elaborator. In a system without definitional singleton types (like a unit type with an $\eta$-rule), we can check $\eta$-conversion for $\Pi$ and $\Sigma$ in a purely syntax-directed way using homogeneously typed unification; this is how current Rocq works. Even in the presence of singleton types, we may only want to compute types when necessary. With heterogeneous unification, type computation becomes more pervasive: we need to compute types to detect $\eta$-conversion and to check well-typing of metavariable solutions. It seems to be an obvious optimization to support *both* homogeneous and heterogeneous unification, and switch to the latter on demand, but that would necessarily complicate the implementation.

We propose to use (3) instead, with only homogeneous unification and a coercion operation. Previously, heterogeneous unification did not allow coercion to appear in arbitrary positions in terms, only allowing coercions of bound variables, represented as either *twin variables* [2, 3] or *twin contexts* [4]. General coercion is a complication in the core theory of elaboration, but we think that it's possibly a better overall setup, for the following reasons.

First, type computation is compartmentalized in the evaluation of coercions and the rest of the implementation is mostly the same as in the homogeneous case. Second, we can solve more problems with general coercions than with twin variables. For example, we can generally solve $\alpha\,f\,x \equiv \mathsf{coe}_{A\,B}\,(f\,x)$ in our setup but not in the heterogeneous one, since the coe on the right hand side is not applied to a plain bound variable. Overall, we conjecture that this system would be more efficient and slightly stronger in practice than previous heterogeneous systems.

We also note an interesting but possibly dangerous feature that we would like to investigate. It is the meta-level definitional equality expressing that $\mathsf{coe}_{B\,C}\,(\mathsf{coe}_{A\,B}\,t)$ is equal to $\mathsf{coe}_{A\,C}\,t$. This is undecidable, since if we work under an assumption of $A \equiv (A \to A)$, we get a meta-level definitional isomorphism between $A$ and $A \to A$, i.e. we get the untyped lambda calculus. However a) it seems rather difficult to manufacture source syntax that makes the elaborator loop b) we can only get a loop from unsolvable postponed equations, i.e. only invalid source syntax can yield loops c) being able to freely shift coercions between equation sides would make the system significantly stronger than previous heterogeneous unification.

*Implementation.* We are currently working on a prototype. It is not functional but there is a good chance that it will be two months from now.

## References

[1] Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. 2007. Observational equality, now!. In *Proceedings of the 2007 workshop on Programming languages meets program verification*. ACM, 57–68.

[2] Adam Michael Gundry. 2013. *Type inference, Haskell and dependent types*. Ph. D. Dissertation. University of Strathclyde, Glasgow, UK. http://oleg.lib.strath.ac.uk/R/?func=dbin-jump-full&object_id=22728

[3] Adam Michael Gundry and Conor McBride. 2013. A tutorial implementation of dynamic pattern unification. (2013). https://adam.gundry.co.uk/pub/pattern-unify/pattern-unification-2012-07-10.pdf

[4] Víctor López Juan. 2021. *Practical Heterogeneous Unification for Dependent Type Checking*. Ph. D. Dissertation. Chalmers University of Technology. https://research.chalmers.se/publication/527051/file/527051_Fulltext.pdf

[5] Loïc Pujet, Yann Leray, and Nicolas Tabareau. 2025. Observational Equality Meets CIC. *ACM Trans. Program. Lang. Syst.* 47, 2 (2025), 6:1–6:35. https://doi.org/10.1145/3719342

[6] Loïc Pujet and Nicolas Tabareau. 2022. Observational equality: now for good. *Proc. ACM Program. Lang.* 6, POPL (2022), 1–27. https://doi.org/10.1145/3498693

[7] Loïc Pujet and Nicolas Tabareau. 2023. Impredicative Observational Equality. *Proc. ACM Program. Lang.* 7, POPL (2023), 2171–2196. https://doi.org/10.1145/3571739