Implementing Observational Equality with Normalisation by Evaluation

Matthew Sirman, Meven Lennon-Bertrand, and Neel Krishnaswami

University of Cambridge, UK

Abstract

We report on an experimental implementation of a type theory with an observational equality type, based on Pujet et al.'s $\mathrm{CC}^{\mathrm{obs}}$, extended with a form of inductive and quotient types. It features a normalisation by evaluation function, which is used to implement a bidirectional type checker. We also explore proof assistant features, notably the interaction of strict propositions with meta-variables, and a rudimentary "hole" system.

1 Observational Equality meets NbE

Observational equality In recent years, building on an early proposal by Altenkirch et al. [3, 4], Pujet et al. [10, 9, 11] have developed $\mathrm{CC}^{\mathrm{obs}}$, a dependent type theory featuring a new presentation of equality: *observational equality*. This equality is *definitionally proof-irrelevant*: any two proofs of equality are identified. Moreover, rather than being uniformly defined like the traditional inductive equality, observational equality has a specific behaviour at each type. Together, these aspects lead to an equality close to traditional mathematical one, with a seamless support of quotients, making it very attractive.

Normalisation by Evaluation Pujet et al.'s work come with an extensive meta-theoretic investigation, yet, they do not implement their type theory. We attack this unexplored aspect, by providing an experimental implementation, based on *normalisation by evaluation* (NbE) [1], a modern technique to decide *definitional equality*. To do so, the NbE approach efficiently computes normal forms by instrumenting the evaluation mechanism of the host language, and then compares these normal forms for a simple, structural notion of equality. In particular, abstractions/applications are handled by using functions of the host language.[1]

NbE for $\mathbf{CC}^{\mathbf{obs}}$ In our implementation, we extend standard NbE techniques, as presented by *e.g.* Abel [1] and in Kovács' `elaboration-zoo` [7], to an extension of $\mathrm{CC}^{\mathrm{obs}}$. Our type theory features a sort of definitionally irrelevant propositions Ω [6], an observational equality valued in that sort, and quotient types. We also explore inductive types, as first-class construct equipped with a form of Mendler-style recursion [8]. We did not investigate the meta-theory of this presentation, but believe it would be an interesting avenue for future research.

Our Haskell code is available on GitHub [12]. Despite the standard NbE ideas required some care to adapt, they largely apply to $\mathrm{CC}^{\mathrm{obs}}$, witnessing their robustness.

2 Semantic propositions

Maybe the most important design decision in our implementation is the structure of the semantic domain D^{Ω} in which proofs are evaluated before being quoted back, in the standard NbE fashion. This should reflect the fact that we should never reduce such irrelevant terms.

[1] Technically, we depart from this by replacing functions with closures, but the philosophy still stands.

The simplest strategy is to erase irrelevant terms at evaluation, following Abel et al. [2], which amounts to set D^{Ω} to be a unit type. Unfortunately this means we cannot quote values, and have to present users with incomplete terms. This also has undesirable consequence when solving meta-variables.

A natural alternative is to keep a term during evaluation, *i.e.* to pick $\mathsf{D}^{\Omega} ::= \mathsf{Prop}\ t\ \rho$ – a term t and an environment ρ for its free variables. This is however still problematic, because terms, being represented with de Bruijn *indices*, do not support cheap lifting, a key operation on semantic values in NbE. Similarly, to be able to substitute a semantic value for a variable in such a proof term – an operation needed during evaluation –, we would have to quote the value, making evaluation and quoting mutually defined, which is unsatisfying.

We thus opt for a semantic domain, which uses closures to support substitution, and de Bruijn *levels* for free lifting. However, these values can represent term structure that would be normalised in relevant values, for instance $[\![(\lambda x.t)\ u]\!]^{\Omega} = \mathsf{PApp}(\mathsf{PLam}(\underline{\lambda} t)\rho)\ [\![u]\!]^{\Omega}\rho$, where $[\![\cdot]\!]^{\Omega}$ is evaluation of irrelevant terms, $\underline{\lambda}$ is a closure, and PApp and PLam are both constructors. Compare to relevant terms where the evaluation would give $[\![t]\!]^{\mathcal{U}}(\rho, [\![u]\!]^{\mathcal{U}}\rho)$.

To decide which evaluation function to use between $[\![\cdot]\!]^{\Omega}$ and $[\![\cdot]\!]^{\mathcal{U}}$, we need to sprinkle terms with relevance annotations: this is the case for binders, as done by Pujet et al. [10], but also for application nodes. This is not unsurprising, as with NbE we need to evaluate arguments of applications, while the reduction of Pujet et al. is call-by-name. Fortunately, all these annotations can be inferred during type-checking, so users do not need to put them in.

3 Cast-on-refl in term-directed NbE

An innovation of Pujet et al. [11] is the recovery of the definitional equality $\mathbf{cast}(A, A, e, t) \equiv t$ for any type A, which did not hold in earlier version of observational equality. The idea is to incorporate it not in reduction, but in conversion, i.e. when comparing two terms. We follow suit: our NbE implements β-normalisation only, and we implement cast-on-refl on a by-need basis when comparing semantic values, just like η-expansion. All in all, our algorithm is entirely term-directed.

4 Proof assistant features

We implement contextual meta-variables in the fashion of MATITA [5] and COQ: a meta-variable has a context, and each of its use of comes with a substitution instantiating said context. This contrasts with the more standard use of lambda-lifting [7], proponents of which claim is simpler. Still, we found contextual meta-variables altogether not too complicated, and led to clearer code.

A very useful feature present in AGDA is the ability to incrementally construct terms by filling holes, with the editor presenting information such as the expected type of a hole and the variables in scope. We implement a lightweight version of this mechanism: our syntax contains a primitive `?{t1, …, tn}`, which errors but reports to the user the expected type for this hole, and that of `t1` to `tn`, as exemplified by the following code.

```
let f : ℕ → ℕ =
  λx. S x
in
let x : ℕ =
  S (S (S 0))
in f ?{f, x}
```

```
[error]: Found proof goal.
     ┌─▶ <test-file>@6:6-6:13
   6 │     f ?{f, x}
     •       ┬──────
     •       ╰╴ Expected type [ℕ] at goal.
     •          List of relevant terms and their types:
─────╯          f : ℕ → ℕ, x : ℕ
```

References

[1] Andreas Abel. "Normalization by Evaluation Dependent Types and Impredicativity". PhD thesis. München: Institut für Informatik Ludwig-Maximilians-Universität München, May 2013. URL: https://www.cse.chalmers.se/~abela/habil.pdf.

[2] Andreas Abel, Thierry Coquand, and Miguel Pagano. "A Modular Type-Checking Algorithm for Type Theory with Singleton Types and Proof Irrelevance". In: *Typed Lambda Calculi and Applications*. Ed. by Pierre-Louis Curien. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 5–19.

[3] Thorsten Altenkirch and Conor McBride. "Towards Observational Type Theory". 2006.

[4] Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. "Observational Equality, Now!" In: *Proceedings of the 2007 Workshop on Programming Languages Meets Program Verification*. Freiburg Germany: ACM, Oct. 2007, pp. 57–68. ISBN: 978-1-59593-677-6. DOI: 10.1145/1292597.1292608. URL: https://dl.acm.org/doi/10.1145/1292597.1292608 (visited on 05/08/2023).

[5] Andrea Asperti, Claudio Sacerdoti Coen, Enrico Tassi, and Stefano Zacchiroli. "User Interaction with the Matita Proof Assistant". In: *Journal of Automated Reasoning* 39 (2007), pp. 109–139.

[6] Gaëtan Gilbert, Jesper Cockx, Matthieu Sozeau, and Nicolas Tabareau. "Definitional Proof-Irrelevance without K". In: *Proceedings of the ACM on Programming Languages*. POPL'19 3.POPL (Jan. 2019), pp. 1–28. DOI: 10.1145/329031610.1145/3290316. URL: https://hal.inria.fr/hal-01859964.

[7] András Kovács. *Elaboration-Zoo*. May 2023. URL: https://github.com/AndrasKovacs/elaboration-zoo (visited on 05/14/2023).

[8] Nax Paul Mendler. "Inductive Types and Type Constraints in the Second-Order Lambda Calculus". In: *Annals of Pure and Applied Logic* 51.1-2 (Mar. 1991), pp. 159–172. ISSN: 01680072. DOI: 10.1016/0168-0072(91)90069-X. URL: https://linkinghub.elsevier.com/retrieve/pii/016800729190069X (visited on 05/04/2023).

[9] Loïc Pujet. "Computing with Extensionality Principles in Dependent Type Theory". PhD thesis. Nantes Université, Dec. 2022.

[10] Loïc Pujet and Nicolas Tabareau. "Observational Equality: Now for Good". In: *Proceedings of the ACM on Programming Languages* 6.POPL (Jan. 2022), pp. 1–27. ISSN: 2475-1421. DOI: 10.1145/3498693. URL: https://dl.acm.org/doi/10.1145/3498693 (visited on 04/26/2023).

[11] Loïc Pujet and Nicolas Tabareau. "Impredicative Observational Equality". In: *Proc. ACM Program. Lang.* 7.POPL (Jan. 2023). DOI: 10.1145/3571739.

[12] Matthew Sirman. *observational-type-theory*. May 2023. URL: https://github.com/matthew-sirman/observational-type-theory (visited on 11/29/2023).