Observing Definitional Equality

András Kovács

University of Gothenburg & Chalmers University of Technology

17 January 2026, WITS, Rennes

Postponed equalities

Let's try to solve:

$$(\alpha\,\mathsf{true},\ (t,\ \gamma)) \stackrel{?}{=} (\beta\,\mathsf{true},\ (u,\ \mathsf{true}))$$

at type $(A : \mathsf{Type}) \times (A \times \mathsf{Bool})$, where α, β, γ are metavariables.

Postponed equalities

Let's try to solve:

$$(\alpha\,\mathsf{true},\ (t,\ \gamma)) \stackrel{?}{=} (\beta\,\mathsf{true},\ (u,\ \mathsf{true}))$$

at type $(A : \mathsf{Type}) \times (A \times \mathsf{Bool})$, where α, β, γ are metavariables.

1. We can't solve $\alpha\,\mathsf{true} \stackrel{?}{=} \beta\,\mathsf{true}$ now but it could become uniquely solvable later. We *postpone*

Postponed equalities

Let's try to solve:

$$(\alpha\,\mathsf{true},\ (t,\ \gamma)) \stackrel{?}{=} (\beta\,\mathsf{true},\ (u,\ \mathsf{true}))$$

at type $(A : \mathsf{Type}) \times (A \times \mathsf{Bool})$, where α, β, γ are metavariables.

1. We can't solve $\alpha\,\mathsf{true} \stackrel{?}{=} \beta\,\mathsf{true}$ now but it could become uniquely solvable later. We *postpone*
2. Now $(t,\ \gamma)$ and $(u,\ \mathsf{true})$ have different types. How to proceed?

Postponed equalities

Let's try to solve:

$$(\alpha\,\mathsf{true},\ (t,\ \gamma)) \stackrel{?}{=} (\beta\,\mathsf{true},\ (u,\ \mathsf{true}))$$

at type $(A : \mathsf{Type}) \times (A \times \mathsf{Bool})$, where α, β, γ are metavariables.

1. We can't solve $\alpha\,\mathsf{true} \stackrel{?}{=} \beta\,\mathsf{true}$ now but it could become uniquely solvable later. We *postpone*
2. Now $(t,\ \gamma)$ and $(u,\ \mathsf{true})$ have different types. How to proceed?
3. At least, we'd like to immediately solve $\gamma \stackrel{?}{=} \mathsf{true}$, where the types are the same.

Existing solutions

- Make no progress; block the rest of the unification problem.

[1] `antiUnify` in https://github.com/agda/agda/blob/master/src/full/Agda/TypeChecking/Conversion.hs

Existing solutions

- Make no progress; block the rest of the unification problem.
- Rocq: ignore mismatched types and proceed. Unification is unsound, but the final elaboration output gets re-checked.

[1] `antiUnify` in https://github.com/agda/agda/blob/master/src/full/Agda/TypeChecking/Conversion.hs

Existing solutions

- Make no progress; block the rest of the unification problem.
- Rocq: ignore mismatched types and proceed. Unification is unsound, but the final elaboration output gets re-checked.
- Agda: *anti-unification*. Questionable.[1]

[1] `antiUnify` in https://github.com/agda/agda/blob/master/src/full/Agda/TypeChecking/Conversion.hs

Existing solutions

- Make no progress; block the rest of the unification problem.
- Rocq: ignore mismatched types and proceed. Unification is unsound, but the final elaboration output gets re-checked.
- Agda: *anti-unification*. Questionable.[1]
- Lean & Idris?

[1] `antiUnify` in https://github.com/agda/agda/blob/master/src/full/Agda/TypeChecking/Conversion.hs

Existing solutions

- Make no progress; block the rest of the unification problem.
- Rocq: ignore mismatched types and proceed. Unification is unsound, but the final elaboration output gets re-checked.
- Agda: *anti-unification*. Questionable.[1]
- Lean & Idris?
- In works by Gundry, McBride and López: *heterogeneous unification* [GM13, Gun13, Jua21]. A bit more on this later.

[1] `antiUnify` in https://github.com/agda/agda/blob/master/src/full/Agda/TypeChecking/Conversion.hs

Observational Equality

Let's make the postponed equality *observational*.

$$(\alpha\,\mathsf{true},\ (t,\ \gamma)) \stackrel{?}{=} (\beta\,\mathsf{true},\ (u,\ \mathsf{true})) : (A : \mathsf{Type}) \times (A \times \mathsf{Bool})$$

1. Try to solve $\alpha\,\mathsf{true} = \beta\,\mathsf{true}$.

Observational Equality

Let's make the postponed equality *observational*.

$$(\alpha\,\mathsf{true},\ (t,\ \gamma)) \stackrel{?}{=} (\beta\,\mathsf{true},\ (u,\ \mathsf{true})) : (A : \mathsf{Type}) \times (A \times \mathsf{Bool})$$

1. Try to solve $\alpha\,\mathsf{true} = \beta\,\mathsf{true}$.
2. Postpone: we extend the metacontext with $p : \alpha\,\mathsf{true} = \beta\,\mathsf{true}$.

Observational Equality

Let's make the postponed equality *observational*.

$$(\alpha\,\mathsf{true},\ (t,\ \gamma)) =?\ (\beta\,\mathsf{true},\ (u,\ \mathsf{true})) : (A : \mathsf{Type}) \times (A \times \mathsf{Bool})$$

1. Try to solve $\alpha\,\mathsf{true} = \beta\,\mathsf{true}$.
2. Postpone: we extend the metacontext with $p : \alpha\,\mathsf{true} = \beta\,\mathsf{true}$.
3. Try to solve $\mathsf{coe}_{(\alpha\,\mathsf{true}\times\mathsf{Bool})\ (\beta\,\mathsf{true}\times\mathsf{Bool})}\ (t,\ \gamma) =?\ (u,\ \mathsf{true})$.

Observational Equality

Let's make the postponed equality *observational*.

$$(\alpha\,\mathsf{true},\ (t,\ \gamma)) =?\ (\beta\,\mathsf{true},\ (u,\ \mathsf{true})) : (A : \mathsf{Type}) \times (A \times \mathsf{Bool})$$

1. Try to solve $\alpha\,\mathsf{true} = \beta\,\mathsf{true}$.
2. Postpone: we extend the metacontext with $p : \alpha\,\mathsf{true} = \beta\,\mathsf{true}$.
3. Try to solve $\mathsf{coe}_{(\alpha\,\mathsf{true}\times\mathsf{Bool})\ (\beta\,\mathsf{true}\times\mathsf{Bool})}\ (t,\ \gamma) =?\ (u,\ \mathsf{true})$.
4. Compute to $(\mathsf{coe}_{(\alpha\,\mathsf{true})\ (\beta\,\mathsf{true})}\ t,\ \gamma) =?\ (u,\ \mathsf{true})$.

Observational Equality

Let's make the postponed equality *observational*.

$$(\alpha\,\mathsf{true},\ (t,\ \gamma)) =?\ (\beta\,\mathsf{true},\ (u,\ \mathsf{true})) : (A : \mathsf{Type}) \times (A \times \mathsf{Bool})$$

1. Try to solve $\alpha\,\mathsf{true} = \beta\,\mathsf{true}$.
2. Postpone: we extend the metacontext with $p : \alpha\,\mathsf{true} = \beta\,\mathsf{true}$.
3. Try to solve $\mathsf{coe}_{(\alpha\,\mathsf{true}\times\mathsf{Bool})\,(\beta\,\mathsf{true}\times\mathsf{Bool})}\ (t,\ \gamma) =?\ (u,\ \mathsf{true})$.
4. Compute to $(\mathsf{coe}_{(\alpha\,\mathsf{true})\,(\beta\,\mathsf{true})}\ t,\ \gamma) =?\ (u,\ \mathsf{true})$.
5. $\mathsf{coe}_{(\alpha\,\mathsf{true})\,(\beta\,\mathsf{true})}\ t =?\ u$ may or may not go through, but $\gamma =?\ \mathsf{true}$ produces a solution.

Overview

We use bidirectional elaboration with normalization-by-evaluation:

Presyntax
$\longrightarrow$ Core syntax + values + metavariables + postponed problems ("partial core")
$\longrightarrow$ Core syntax + values ("total core")
$\longrightarrow$ ...

We use a stripped-down version of Pujet and Tabareau's OTT [PT22]

[2] https://github.com/AndrasKovacs/2ltt-impl

Overview

We use bidirectional elaboration with normalization-by-evaluation:

Presyntax
$\longrightarrow$ Core syntax + values + metavariables + postponed problems ("partial core")
$\longrightarrow$ Core syntax + values ("total core")
$\longrightarrow$ ...

We use a stripped-down version of Pujet and Tabareau's OTT [PT22]

We don't have: user-facing observational features, term language for proofs, propositions besides equality.

[2] https://github.com/AndrasKovacs/2ltt-impl

Overview

We use bidirectional elaboration with normalization-by-evaluation:

$$
\begin{aligned}
& \text{Presyntax} \\
\longrightarrow\ & \text{Core syntax} + \text{values} + \text{metavariables} + \text{postponed problems (“partial core”)} \\
\longrightarrow\ & \text{Core syntax} + \text{values (“total core”)} \\
\longrightarrow\ & \ldots
\end{aligned}
$$

We use a stripped-down version of Pujet and Tabareau's OTT [PT22]

We don't have: user-facing observational features, term language for proofs, propositions besides equality.

WIP implementation that's planned to be “production-strength”[2].

[2] https://github.com/AndrasKovacs/2ltt-impl

Issue 1: reflexive coercion

OTT supports the conversion rule $\mathsf{coe}_{A\,A}\,t = t$. How to implement it?

In prior work: coe−refl is not computed during *reduction or evaluation*, it's only computed during *conversion checking* [SLK24, PT22].

[3] https://downloads.haskell.org/ghc/latest/docs/users_guide/exts/implicit_parameters.html

Issue 1: reflexive coercion

OTT supports the conversion rule $\mathsf{coe}_{A\,A}\,t = t$. How to implement it?

In prior work: coe−refl is not computed during *reduction or evaluation*, it's only computed during *conversion checking* [SLK24, PT22].

Instead, I compute eagerly during evaluation.

- Hanging on to coercions feels risky for value/term size.
- In our case, *all coercions must be computed away eventually*.

[3] https://downloads.haskell.org/ghc/latest/docs/users_guide/exts/implicit_parameters.html

Issue 1: reflexive coercion

OTT supports the conversion rule $\mathrm{coe}_{A\,A}\,t = t$. How to implement it?

In prior work: coe−refl is not computed during *reduction or evaluation*, it's only computed during *conversion checking* [SLK24, PT22].

Instead, I compute eagerly during evaluation.

- Hanging on to coercions feels risky for value/term size.
- In our case, *all coercions must be computed away eventually*.

This requires threading a *fresh De Bruijn level* through evaluation, because conversion checking can go under binders.

Haskell trick[3]: functional closures with type `(?lvl :: Int) => Value -> Value`

[3] https://downloads.haskell.org/ghc/latest/docs/users_guide/exts/implicit_parameters.html

Issue 2: η-short coercion

Unlike in prior OTT works, coercion shouldn't η-expand!

η-long coercion for non-dependent functions:

$$\mathsf{coe}_{(A \to B)\,(A' \to B')}\; t = \lambda\, x.\, \mathsf{coe}_{B\,B'}\; (t\, (\mathsf{coe}_{A'\,A}\; x))$$

Issue 2: η-short coercion

Unlike in prior OTT works, coercion shouldn't η-expand!

η-long coercion for non-dependent functions:

$$\mathsf{coe}_{(A \to B)\,(A' \to B')}\; t = \lambda\, x.\, \mathsf{coe}_{B\,B'}\; (t\; (\mathsf{coe}_{A'\,A}\; x))$$

η-short coercion only computes on canonical values:

$$\mathsf{coe}_{(A \to B)\,(A' \to B')}\; (\lambda\, x.\, t) = \lambda\, y.\, \mathsf{coe}_{B\,B'}\; (t[x \mapsto (\mathsf{coe}_{A'\,A}\; y)]$$

Issue 2: η-short coercion

Unlike in prior OTT works, coercion shouldn't η-expand!

η-long coercion for non-dependent functions:

$$\mathrm{coe}_{(A \to B)\,(A' \to B')}\; t = \lambda\, x.\, \mathrm{coe}_{B\,B'}\; (t\, (\mathrm{coe}_{A'\,A}\; x))$$

η-short coercion only computes on canonical values:

$$\mathrm{coe}_{(A \to B)\,(A' \to B')}\; (\lambda\, x.\, t) = \lambda\, y.\, \mathrm{coe}_{B\,B'}\; (t[x \mapsto (\mathrm{coe}_{A'\,A}\; y)]$$

To retain syntax-directed η-conversion, function application must explicitly handle coerced neutral functions:

$$\begin{aligned} (\lambda\, x.\, t)\, u &= t[x \mapsto u] \\ (\mathrm{coe}_{(A \to B)\,(A' \to B')}\; n)\, u &= \mathrm{coe}_{B\,B'}\; (n\, (\mathrm{coe}_{A'\,A}\; u)) \end{aligned}$$

Issue 2: η-short coercion

For pairs:

$$
\begin{aligned}
\mathsf{coe}_{(A\times B)\,(A'\times B')}\,(t,\,u) &= (\mathsf{coe}_{A\,A'}\,t,\,\mathsf{coe}_{B\,B'}\,u)\\
(t,\,u).\mathsf{fst} &= t\\
(\mathsf{coe}_{(A\times B)\,(A'\times B')}\,n).\mathsf{fst} &= \mathsf{coe}_{A\,A'}\,(n.\mathsf{fst})\\
(t,\,u).\mathsf{snd} &= u\\
(\mathsf{coe}_{(A\times B)\,(A'\times B')}\,n).\mathsf{snd} &= \mathsf{coe}_{B\,B'}\,(n.\mathsf{snd})
\end{aligned}
$$

Issue 3: representing neutral coercion

Spine neutral values consist of a **head** which blocks computation, and a **spine** of blocked eliminators. We want immediate access to the reason for blocking!

Issue 3: representing neutral coercion

Spine neutral values consist of a **head** which blocks computation, and a **spine** of blocked eliminators. We want immediate access to the reason for blocking!

In plain NbE-based elaboration, heads can be

- bound variables ("rigid head")
- metavariables ("flexible head")

Weak-head forcing w.r.t. metacontext: unfold solved metas in head position.

How should we represent neutral coercions?

Issue 3: representing neutral coercion

We need at least three different ways to represent coercion.

Issue 3: representing neutral coercion

We need at least three different ways to represent coercion.

1. A rigidly blocked coercion of a canonical value is a **rigid head.**
   - Example: $(\mathsf{coe}_{\mathsf{Bool}\,\mathsf{Nat}}\,\mathsf{true})\,.(\mathsf{NatElim}\;args)$
   - This kind of coercion always signals an error.
   - But we need to compute with errors if we want to report multiple errors to users!

Issue 3: representing neutral coercion

We need at least three different ways to represent coercion.

1. A rigidly blocked coercion of a canonical value is a **rigid head.**
   - Example: $(\mathsf{coe}_{\mathsf{Bool}\,\mathsf{Nat}}\,\mathsf{true})\,.(\mathsf{NatElim}\;\mathit{args})$
   - This kind of coercion always signals an error.
   - But we need to compute with errors if we want to report multiple errors to users!
2. A flexibly blocked coercion of a canonical value is a **flexible head.**
   - Example: $(\mathsf{coe}_{\mathsf{Bool}\,(\alpha\;\mathit{true})}\,\mathsf{true})$ with empty spine.

Issue 3: representing neutral coercion

We need at least three different ways to represent coercion.

1. A rigidly blocked coercion of a canonical value is a **rigid head.**
   - Example: $(\mathsf{coe}_{\mathsf{Bool}\,\mathsf{Nat}}\,\mathsf{true})\,.(\mathsf{NatElim}\;args)$
   - This kind of coercion always signals an error.
   - But we need to compute with errors if we want to report multiple errors to users!
2. A flexibly blocked coercion of a canonical value is a **flexible head.**
   - Example: $(\mathsf{coe}_{\mathsf{Bool}\,(\alpha\;\mathit{true})}\,\mathsf{true})$ with empty spine.
3. Any coercion of a neutral value is a **spine entry**.
   - Example: $\alpha\;t\;u\,.\mathsf{coe}_{A\,B}$.

Issue 4: coercing indexed inductive types

In usual OTT: *fording* with perhaps some non-forded sugar.

Fording doesn't work here.

Issue 4: coercing indexed inductive types

In usual OTT: *forcing* with perhaps some non-forced sugar.

Forcing doesn't work here.

What about $\mathsf{coe}_{(\mathsf{Id}\, A\, t\, t)\, (\mathsf{Id}\, A'\, t'\, t')}\, (\mathsf{refl}\, A\, t) = \mathsf{refl}\, A'\, t'$?

Issue 4: coercing indexed inductive types

In usual OTT: *fording* with perhaps some non-forded sugar.

Fording doesn't work here.

What about $\mathsf{coe}_{(\mathsf{Id}\, A\, t\, t)\, (\mathsf{Id}\, A'\, t'\, t')}\, (\mathsf{refl}\, A\, t) = \mathsf{refl}\, A'\, t'$?

In general, we need some kind of *non-linear matching* to detect when a coercion rule should fire!

- Alternative: don't bother computing these coercions.
- Alternative: *make your users ford.* In the core theory, only support parameterized types, intensional identity and coe computation for refl.

Playing with fire

What about $\mathsf{coe}_{B\,C}\,(\mathsf{coe}_{A\,B}\,t) = \mathsf{coe}_{A\,C}\,t$?

It's undecidable: if the metacontext implies $A = (A \to A)$, coercion yields a definitional isomorphism $A \simeq (A \to A)$, i.e. the untyped lambda calculus.

Playing with fire

What about $\mathrm{coe}_{B\,C}\,(\mathrm{coe}_{A\,B}\,t) = \mathrm{coe}_{A\,C}\,t$?

It's undecidable: if the metacontext implies $A = (A \to A)$, coercion yields a definitional isomorphism $A \simeq (A \to A)$, i.e. the untyped lambda calculus.

But it's so nice in unification!

- $\alpha\,.\mathrm{coe}_{A\,B} \stackrel{?}{=} t$ can be solvable as $\alpha := \mathrm{coe}_{B\,A}\,t$.
- $\alpha\,(x\,.\mathrm{coe}_{A\,B}) \stackrel{?}{=} t$ can be solvable as $\alpha := \lambda\,y.\,t[x \mapsto y\,.\mathrm{coe}_{B\,A}]$.
- We get rid of the asymmetry of coe. AFAIK there's no other way to do it.

We can only loop in inconsistent metacontexts (i.e. starting from invalid source).

How hard is it to make the elaborator loop?

Comparison to heterogeneous unification

Heterogeneous unification restricts OTT: coercions can only appear

1. on the outside of a term in a postponed equation, i.e. in $\mathrm{coe}_{A\,B}\,t = u$.
2. around bound variables (called "twin variables").

Comparison to heterogeneous unification

Heterogeneous unification restricts OTT: coercions can only appear

1. on the outside of a term in a postponed equation, i.e. in $\mathrm{coe}_{A\,B}\,t = u$.
2. around bound variables (called "twin variables").

Heterogeneous unification

- doesn't need coercions in the core,
- but it's weaker
- and we have to use *both* homogeneous and heterogeneous unification if we want to be efficient.

More things & further work

Not mentioned:

- Interaction of OTT with controlled definition unfolding.
- Implementation of constraints & blocking.

Further work: get this to demo-able and testable state.

Adam Gundry and Conor McBride.
A tutorial implementation of dynamic pattern unification.
Unpublished draft, page 194, 2013.

Adam Michael Gundry.
Type inference, Haskell and dependent types.
PhD thesis, University of Strathclyde, Glasgow, UK, 2013.

Víctor López Juan.
Practical Heterogeneous Unification for Dependent Type Checking.
PhD thesis, Chalmers University of Technology, 2021.

Loïc Pujet and Nicolas Tabareau.
Observational equality: now for good.
Proc. ACM Program. Lang., 6(POPL):1–27, 2022.

Matthew Sirman, Meven Lennon-Bertrand, and Neel Krishnaswami.
Implementing a type theory with observational equality, using normalisation by evaluation.
In Rasmus Ejlers Møgelberg and Benno van den Berg, editors, *30th International Conference on Types for Proofs and Programs, TYPES 2024, Copenhagen, Denmark, June 10-14, 2024*, volume 336 of *LIPIcs*, pages 5:1–5:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.