Local Rewriting in Dependent Type Theory

Nathaniel Burke

Imperial Computing Final Year Project Presentation

2025

Motivation

- **Dependent Type Theory:** Foundation of many proof assistants and cutting-edge programming languages
  - **Expressive:** Scales to modern mathematics[1] and metamathematics (including the study of type theory itself[2])

---

[1] Escardó and contributors 2025, *TypeTopology*; Buzzard and contributors 2025, *FLT*.

[2] Pujet and Tabareau 2022, *Observational equality: now for good*; Abel, Danielsson, and Eriksson 2023, *A Graded Modal Dependent Type Theory with a Universe and Erasure, Formalized*.

Motivation

- **Dependent Type Theory:** Foundation of many proof assistants and cutting-edge programming languages
  - **Expressive:** Scales to modern mathematics[1] and metamathematics (including the study of type theory itself[2])
- Allows us to precisely specify and verify programs
  - E.g. $\Pi\, x : \mathbb{N}, y : \mathbb{N}.\ x + y = y + x$

[1] Escardó and contributors 2025, *TypeTopology*; Buzzard and contributors 2025, *FLT*.

[2] Pujet and Tabareau 2022, *Observational equality: now for good*; Abel, Danielsson, and Eriksson 2023, *A Graded Modal Dependent Type Theory with a Universe and Erasure, Formalized*.

Motivation

- **Dependent Type Theory:** Foundation of many proof assistants and cutting-edge programming languages
  - **Expressive:** Scales to modern mathematics[1] and metamathematics (including the study of type theory itself[2])
- Allows us to precisely specify and verify programs
  - E.g. $\Pi\, x : \mathbb{N}, y : \mathbb{N}.\ \mathtt{x\ +\ y\ =\ y\ +\ x}$
- **Drawback:** Limited automation, especially with respect to equational reasoning

[1] Escardó and contributors 2025, *TypeTopology*; Buzzard and contributors 2025, *FLT*.

[2] Pujet and Tabareau 2022, *Observational equality: now for good*; Abel, Danielsson, and Eriksson 2023, *A Graded Modal Dependent Type Theory with a Universe and Erasure, Formalized*.

Manual Equational Reasoning in Proof Assistants

Demo...

“Transport Hell”

- *Indexed datatypes* often require equational reasoning mutual with the implementation of operations (*transport*).

[3] Various Contributors 2024, *Relation.Binary.PropositionalEquality.Properties*.

"Transport Hell"

- *Indexed datatypes* often require equational reasoning mutual with the implementation of operations (*transport*).
- Fin n, Vec A n, Tm Γ A etc...

[3] Various Contributors 2024, *Relation.Binary.PropositionalEquality.Properties*.

“Transport Hell”

- *Indexed datatypes* often require equational reasoning mutual with the implementation of operations (*transport*).
- Fin n, Vec A n, Tm Γ A etc...
- When proving laws about these operations, we have to account for these transports, using painful lemmas like[3]:

[3] Various Contributors 2024, *Relation.Binary.PropositionalEquality.Properties*.

"Transport Hell"

- *Indexed datatypes* often require equational reasoning mutual with the implementation of operations (*transport*).
- Fin n, Vec A n, Tm Γ A etc...
- When proving laws about these operations, we have to account for these transports, using painful lemmas like[3]:

$$
\begin{aligned}
\text{transp-application} \;:\; & \Pi\,(B : A \to \mathbf{Type})\,\{C : A \to \mathbf{Type}\} \\
& \{y\}\,(g : \Pi\, x \to B\, x \to C\, x) \\
& (p : x_1 = x_2) \\
\to\; & \mathbf{transp}\; C\; p\; (g\; x_1\; y) \\
=\; & g\; x_2\; (\mathbf{transp}\; B\; p\; y)
\end{aligned}
$$

[3]Various Contributors 2024, *Relation.Binary.PropositionalEquality.Properties*.

Contribution

Demo...

Contribution

Demo...

- A new type theory where the built-in equality (*conversion*) is defined modulo a set of local equations.

Contribution

Demo...

- A new type theory where the built-in equality (*conversion*) is defined modulo a set of local equations.
- To decide conversion, we now rely on techniques from *term rewriting*.

Contribution

Demo...

- A new type theory where the built-in equality (*conversion*) is defined modulo a set of local equations.
- To decide conversion, we now rely on techniques from *term rewriting*.
- Concrete contributions include formal results (proofs!) and a prototype typechecker implementation.

Local Equality Reflection

- **Idea:** Extend contexts with equational assumptions.

Local Equality Reflection

- **Idea:** Extend contexts with equational assumptions.
- Reflecting arbitrary propositional equations is very powerful, but breaks decidability of typechecking.

$$\frac{\Gamma \;:\; \mathrm{Ctx}, \quad A, B \;:\; \mathrm{Ty}\;\Gamma, \quad t_1, t_2 \;:\; \mathrm{Tm}\;\Gamma\;A \\ p \;:\; \mathrm{Tm}\;\Gamma\;(t_1 \;=\; t_2) \\ u \;:\; \mathrm{Tm}\;(\Gamma \rhd t_1 \sim t_2)\;B}{\mathrm{reflect}\;p\;\mathrm{in}\;u \;:\; \mathrm{Tm}\;\Gamma\;B}$$

Local Equality Reflection

- **Idea:** Extend contexts with equational assumptions.
- Reflecting arbitrary propositional equations is very powerful, but breaks decidability of typechecking.

$$\frac{\Gamma : \mathsf{Ctx}, \quad A, B : \mathsf{Ty}\ \Gamma, \quad t_1, t_2 : \mathsf{Tm}\ \Gamma\ A \qquad p : \mathsf{Tm}\ \Gamma\ (t_1 = t_2) \qquad u : \mathsf{Tm}\ (\Gamma \triangleright t_1 \sim t_2)\ B}{\mathsf{reflect}\ p\ \mathsf{in}\ u : \mathsf{Tm}\ \Gamma\ B}$$

- Need to restrict equations somehow...

Local Equality Reflection

- **Idea:** Extend contexts with equational assumptions.
- Reflecting arbitrary propositional equations is very powerful, but breaks decidability of typechecking.

$$\frac{\vdash \Gamma \text{ ctx}, \quad \Gamma \vdash A, B \text{ type}, \quad \Gamma \vdash t_1, t_2 : A \qquad \Gamma \vdash p : (t_1 \ = \ t_2) \qquad (\Gamma \rhd t_1 \sim t_2) \vdash u : B}{\Gamma \vdash \text{reflect } p \text{ in } u : B}$$

- Need to restrict equations somehow...

"Smart Case"

Starting Point: Equations arising from (Boolean) case splits[4]

$$\frac{\Gamma : \mathsf{Ctx}, \quad t : \mathsf{Tm}\ \Gamma\ \mathbb{B}, \quad A : \mathsf{Ty}\ \Gamma \qquad u : \mathsf{Tm}\ (\Gamma \triangleright t \sim \mathsf{tt})\ A \qquad v : \mathsf{Tm}\ (\Gamma \triangleright t \sim \mathsf{ff})\ A}{\mathsf{sif}\ t\ \mathsf{then}\ u\ \mathsf{else}\ v : \mathsf{Tm}\ \Gamma\ A}$$

The scrutinee and pattern are convertible in each branch.

[4] Altenkirch 2011, *The case of the smart case.*

A Substitution Calculus for Contextual Equations ($SC^{\textsc{Bool}}$)

- Mapping from the empty context is trivial.

$$\frac{}{\varepsilon \;:\; \mathsf{Tms}\ \Delta\ \bullet}$$

A Substitution Calculus for Contextual Equations ($\mathrm{SC}^{\textsc{Bool}}$)

- Mapping from the empty context is trivial.

$$\frac{}{\varepsilon \; : \; \mathsf{Tms}\; \Delta \; \bullet}$$

- To map from a context extended with a variable, we need to provide a substitute term.

$$\frac{\delta \; : \; \mathsf{Tms}\; \Delta\; \Gamma, \quad \mathsf{t} \; : \; \mathsf{Tm}\; \Delta\; (\mathsf{A} \; [\; \delta \;]_{\mathsf{Ty}})}{\delta \, , \; \mathsf{t} \; : \; \mathsf{Tms}\; \Delta\; (\Gamma \rhd \mathsf{A})}$$

A Substitution Calculus for Contextual Equations ($\mathrm{SC}^{\textsc{Bool}}$)

- Mapping from the empty context is trivial.

$$\frac{}{\varepsilon \;:\; \mathsf{Tms}\ \Delta\ \bullet}$$

- To map from a context extended with a variable, we need to provide a substitute term.

$$\frac{\delta \;:\; \mathsf{Tms}\ \Delta\ \Gamma, \quad t \;:\; \mathsf{Tm}\ \Delta\ (A\ [\ \delta\ ]_{\mathsf{Ty}})}{\delta\ ,\ t \;:\; \mathsf{Tms}\ \Delta\ (\Gamma \rhd A)}$$

- To map from a context extended with an equation, we need to provide substitute *evidence of convertibility*. (**New!**)

$$\frac{\delta \;:\; \mathsf{Tms}\ \Delta\ \Gamma, \quad p \;:\; t_1\ [\ \delta\ ] \sim_{\mathsf{Tm}} t_2\ [\ \delta\ ]}{\delta\ ,_{\sim}\ p \;:\; \mathsf{Tms}\ \Delta\ (\Gamma \rhd t_1 \sim t_2)}$$

Normalisation by Evaluation (NbE)

- **Aim:** Associate a canonical representative ("normal form") with every equivalence class of terms.

[5] Berger and Schwichtenberg 1991, *An Inverse of the Evaluation Functional for Typed lambda-calculus*.

[6] Altenkirch and Kaposi 2017, *Normalisation by Evaluation for Type Theory, in Type Theory*.

Normalisation by Evaluation (NbE)

- **Aim:** Associate a canonical representative ("normal form") with every equivalence class of terms.
- **Idea:** Construct a model (*evaluation*) and invert it (*quotation*).
  $\mathrm{norm}\ t \ :\equiv\ \mathrm{quote}\ (\mathrm{eval}\ \mathrm{id}^{\mathrm{Env}}\ t)$

[5] Berger and Schwichtenberg 1991, *An Inverse of the Evaluation Functional for Typed lambda-calculus.*

[6] Altenkirch and Kaposi 2017, *Normalisation by Evaluation for Type Theory, in Type Theory.*

Normalisation by Evaluation (NbE)

- **Aim:** Associate a canonical representative ("normal form") with every equivalence class of terms.
- **Idea:** Construct a model (*evaluation*) and invert it (*quotation*).
  $\mathsf{norm}\ t \ \coloneqq\ \mathsf{quote}\ (\mathsf{eval}\ \mathsf{id}^{\mathsf{Env}}\ t)$
- **Soundness:** Conversion is preserved during evaluation and quotation.
  $t \sim_{\mathsf{Tm}} u \ \rightarrow\ \mathsf{norm}\ t \ =\ \mathsf{norm}\ u$

[5] Berger and Schwichtenberg 1991, *An Inverse of the Evaluation Functional for Typed lambda-calculus.*

[6] Altenkirch and Kaposi 2017, *Normalisation by Evaluation for Type Theory, in Type Theory.*

Normalisation by Evaluation (NbE)

- **Aim:** Associate a canonical representative ("normal form") with every equivalence class of terms.
- **Idea:** Construct a model (*evaluation*) and invert it (*quotation*).
  $\mathsf{norm}\ t \ \coloneqq\ \mathsf{quote}\ \ (\mathsf{eval}\ \mathsf{id}^{\mathsf{Env}}\ t)$
- **Soundness:** Conversion is preserved during evaluation and quotation.
  $t \sim_{\mathsf{Tm}} u \ \rightarrow\ \mathsf{norm}\ t \ =\ \mathsf{norm}\ u$
- **Completeness:** Equality of normal forms implies convertibility of original terms (conservative).
  $\mathsf{norm}\ t \ =\ \mathsf{norm}\ u \ \rightarrow\ t \sim_{\mathsf{Tm}} u$

[5] Berger and Schwichtenberg 1991, *An Inverse of the Evaluation Functional for Typed lambda-calculus*.

[6] Altenkirch and Kaposi 2017, *Normalisation by Evaluation for Type Theory, in Type Theory*.

NbE for SC^{Bool}: Inconsistent Contexts

- All types are *propositionally* equal under *propositionally* inconsistent contexts, e.g. $\Gamma \coloneqq p : (\text{tt} = \text{ff})$.
  - Normalisation retained—transport blocks computation

NbE for SC$^{\textsc{Bool}}$: Inconsistent Contexts

- All types are *propositionally* equal under *propositionally* inconsistent contexts, e.g. $\Gamma \coloneqq p : (\mathrm{tt} = \mathrm{ff})$.
  - Normalisation retained—transport blocks computation
- *Definitionally* inconsistent contexts are more dangerous! In $\Gamma \coloneqq b : \mathbb{B}\,,\ b \sim \mathrm{tt}\,,\ b \sim \mathrm{ff}$, "$(\lambda\ x.\ x\ x)\ (\lambda\ x.\ x\ x)$" is typeable.

$$
\begin{aligned}
& A \\
\equiv\ & \text{if tt then } A \text{ else } (A \to A) \\
\equiv\ & \text{if ff then } A \text{ else } (A \to A) \\
\equiv\ & (A \to A)
\end{aligned}
$$

NbE for SC^{Bool}: Inconsistent Contexts

- All types are *propositionally* equal under *propositionally* inconsistent contexts, e.g. $\Gamma \coloneqq p : (tt = ff)$.
  - Normalisation retained—transport blocks computation
- *Definitionally* inconsistent contexts are more dangerous! In $\Gamma \coloneqq b : \mathbb{B}, b \sim tt, b \sim ff$, "$(\lambda\, x.\, x\, x)\ (\lambda\, x.\, x\, x)$" is typeable.

$$\begin{aligned} & A \\ \equiv\ & \text{if } tt \text{ then } A \text{ else } (A \to A) \\ \equiv\ & \text{if } ff \text{ then } A \text{ else } (A \to A) \\ \equiv\ & (A \to A) \end{aligned}$$

- Need to avoid evaluating under (*definitionally*) inconsistent contexts.

NbE for SC$^{\textsc{Bool}}$: Completion

- Deciding context inconsistency is non-trivial!
  - LHSs might be reducible: $(\lambda\ x.\ x)\ b \sim tt\ ,\ \ b \sim ff$
  - LHSs might overlap: $\quad not\ \ b \sim tt\ ,\ \ b \sim tt$

[7] Baader and Nipkow 1998, *Term Rewriting and All That*.

NbE for SC^{Bool}: Completion

- Deciding context inconsistency is non-trivial!
  - LHSs might be reducible: $(\lambda\ x.\ x)\ b \sim tt\ ,\ \ b \sim ff$
  - LHSs might overlap: $not\ \ b \sim tt\ ,\ \ b \sim tt$
- The appropriate technique here is *completion*[7].
  - Aims to transform a set of equations into a *confluent* term rewriting system (TRS).
  - **Confluence:** $t >^* u$ and $t >^* v$ implies existence of a term, w, such that $u >^* w$ and $v >^* w$.
  - Needs a well-founded order...

[7] Baader and Nipkow 1998, *Term Rewriting and All That*.

NbE for $\mathrm{SC}^{\textsc{Bool}}$: Challenges

- **Idea:** Avoid completion by enforcing that LHSs are irreducible and disjoint (syntactic restriction).

NbE for $\mathrm{SC}^{\textsc{Bool}}$: Challenges

- **Idea:** Avoid completion by enforcing that LHSs are irreducible and disjoint (syntactic restriction).
- **Doesn't Work!** Unstable under substitution!
  $\Gamma \ \coloneqq \ x : \mathbb{B}\ ,\ y : \mathbb{B}\ ,\ x \sim tt\ ,\ y \sim ff$

NbE for $\mathrm{SC}^{\textsc{Bool}}$: Challenges

- **Idea:** Avoid completion by enforcing that LHSs are irreducible and disjoint (syntactic restriction).
- **Doesn't Work!** Unstable under substitution!
  $\Gamma \;\coloneqq\; x : \mathbb{B}\,,\; y : \mathbb{B}\,,\; x \sim \mathsf{tt}\,,\; y \sim \mathsf{ff}$
- Stability-under-substitution also rules out a lot of more interesting equations (beyond Booleans).

NbE for $\mathrm{SC}^{\textsc{Bool}}$: Challenges

- **Idea:** Avoid completion by enforcing that LHSs are irreducible and disjoint (syntactic restriction).
- **Doesn't Work!** Unstable under substitution!
  $\Gamma \;\coloneqq\; x : \mathbb{B}\,,\; y : \mathbb{B}\,,\; x \sim \mathsf{tt}\,,\; y \sim \mathsf{ff}$
- Stability-under-substitution also rules out a lot of more interesting equations (beyond Booleans).
- **Further Difficulties:** Evaluation must recurse into the branches of **smart if**.
  - Need to interleave evaluation and completion.
  - Normal forms (also values) are not stable under extending the context with equations.

Recovering Normalisation via Elaboration

Elaborating Case Splits to Top-Level Definitions

Already implemented in Agda[8]

```
cmp : ℕ → ℕ → Cmp
cmp n m with n > m
cmp n m | tt ≔ gt
cmp n m | ff with n < m
cmp n m | ff | tt ≔ lt
cmp n m | ff | ff ≔ eq
```

```
data Cmp : Type where
  gt : Cmp
  eq : Cmp
  lt : Cmp
```

[8] The Agda Team 2024b, *With-Abstraction*; The Agda Team 2024a, *Lambda Abstraction*.

Elaborating Case Splits to Top-Level Definitions

Already implemented in Agda[8]

```
cmp : ℕ → ℕ → Cmp
cmp n m with n > m
cmp n m | tt ≔ gt
cmp n m | ff with n < m
cmp n m | ff | tt ≔ lt
cmp n m | ff | ff ≔ eq
```

```
data Cmp : Type where
  gt : Cmp
  eq : Cmp
  lt : Cmp
```

$\text{cmp-elab}\ n\ m \coloneqq \text{cmp-aux}_1\ n\ m\ (n > m)$

[8] The Agda Team 2024b, *With-Abstraction*; The Agda Team 2024a, *Lambda Abstraction*.

Elaborating Case Splits to Top-Level Definitions

Already implemented in Agda[8]

```
cmp : ℕ → ℕ → Cmp
cmp n m with n > m
cmp n m | tt ≔ gt
cmp n m | ff with n < m
cmp n m | ff | tt ≔ lt
cmp n m | ff | ff ≔ eq
```

```
data Cmp : Type where
  gt : Cmp
  eq : Cmp
  lt : Cmp
```

```
cmp-aux₁ n m tt ≔ gt
cmp-aux₁ n m ff ≔ cmp-aux₂ n m (n < m)

cmp-elab n m ≔ cmp-aux₁ n m (n > m)
```

[8] The Agda Team 2024b, *With-Abstraction*; The Agda Team 2024a, *Lambda Abstraction*.

Elaborating Case Splits to Top-Level Definitions

Already implemented in Agda[8]

```
cmp : ℕ → ℕ → Cmp
cmp n m with n > m
cmp n m | tt ≔ gt
cmp n m | ff with n < m
cmp n m | ff | tt ≔ lt
cmp n m | ff | ff ≔ eq
```

```
data Cmp : Type where
  gt : Cmp
  eq : Cmp
  lt : Cmp
```

```
cmp-aux₂ n m tt ≔ lt
cmp-aux₂ n m ff ≔ eq

cmp-aux₁ n m tt ≔ gt
cmp-aux₁ n m ff ≔ cmp-aux₂ n m (n < m)

cmp-elab n m ≔ cmp-aux₁ n m (n > m)
```

[8] The Agda Team 2024b, *With-Abstraction*; The Agda Team 2024a, *Lambda Abstraction*.

Elaborating Case Splits to Top-Level Definitions (SC^{DEF})

$$\text{f3} : \mathbf{\Pi}\ (\text{f} \ \mathbf{:}\ \mathbb{B} \ \mathbf{\rightarrow}\ \mathbb{B}) \ \mathbf{\rightarrow}\ \text{f}\ \mathbf{tt} \ \mathbf{=}\ \text{f}\ (\text{f}\ (\text{f}\ \mathbf{ff}))$$
$$\text{f3 f} \ \coloneqq\ \text{sif}\ (\text{f}\ \mathbf{tt})\ \text{then}\ \mathbf{refl}\ \text{else}\ (\text{sif}\ (\text{f}\ \mathbf{ff})\ \text{then}\ \mathbf{refl}\ \text{else}\ \mathbf{refl})$$

Becomes...

Elaborating Case Splits to Top-Level Definitions (SC^{DEF})

$\text{f3} : \mathbf{\Pi}\ (\text{f}\ :\ \mathbb{B}\ \rightarrow\ \mathbb{B})\ \rightarrow\ \text{f}\ \mathbf{tt}\ =\ \text{f}\ (\text{f}\ (\text{f}\ \mathbf{ff}))$
$\text{f3 f}\ \coloneqq\ \text{sif}\ (\text{f}\ \mathbf{tt})\ \text{then}\ \mathbf{refl}\ \text{else}\ (\text{sif}\ (\text{f}\ \mathbf{ff})\ \text{then}\ \mathbf{refl}\ \text{else}\ \mathbf{refl})$

Becomes...

$\text{f3-elab}\ :\ \mathbf{\Pi}\ (\text{f}\ :\ \mathbb{B}\ \rightarrow\ \mathbb{B})\ \rightarrow\ \text{f}\ \mathbf{tt}\ =\ \text{f}\ (\text{f}\ (\text{f}\ \mathbf{tt}))$
$\text{f3-elab}\ \coloneqq\ \text{call f3-aux}_1\ \text{f}$

Elaborating Case Splits to Top-Level Definitions (SC^{DEF})

$$\mathrm{f3} : \mathbf{\Pi}\ (\mathrm{f} : \mathbb{B} \rightarrow \mathbb{B}) \rightarrow \mathrm{f}\ \mathbf{tt} = \mathrm{f}\ (\mathrm{f}\ (\mathrm{f}\ \mathbf{ff}))$$
$$\mathrm{f3}\ \mathrm{f} \coloneqq \mathrm{sif}\ (\mathrm{f}\ \mathbf{tt})\ \mathrm{then}\ \mathbf{refl}\ \mathrm{else}\ (\mathrm{sif}\ (\mathrm{f}\ \mathbf{ff})\ \mathrm{then}\ \mathbf{refl}\ \mathrm{else}\ \mathbf{refl})$$

Becomes...

$$\mathrm{f3\text{-}aux}_1 : \mathbf{\Pi}\ (\mathrm{f} : \mathbb{B} \rightarrow \mathbb{B}) \mid \mathrm{f}\ \mathbf{tt}$$
$$\rightarrow \mathrm{f}\ \mathbf{tt} = \mathrm{f}\ (\mathrm{f}\ (\mathrm{f}\ \mathbf{tt}))$$
$$\mathrm{f3\text{-}aux}_1\ \mathrm{f} \mid \mathbf{tt} \coloneqq \mathbf{refl}$$
$$\mathrm{f3\text{-}aux}_1\ \mathrm{f} \mid \mathbf{ff} \coloneqq \mathrm{call}\ \mathrm{f3\text{-}aux}_2\ \mathrm{f}$$

$$\mathrm{f3\text{-}elab} : \mathbf{\Pi}\ (\mathrm{f} : \mathbb{B} \rightarrow \mathbb{B}) \rightarrow \mathrm{f}\ \mathbf{tt} = \mathrm{f}\ (\mathrm{f}\ (\mathrm{f}\ \mathbf{tt}))$$
$$\mathrm{f3\text{-}elab} \coloneqq \mathrm{call}\ \mathrm{f3\text{-}aux}_1\ \mathrm{f}$$

Elaborating Case Splits to Top-Level Definitions ($\mathrm{SC}^{\mathrm{DEF}}$)

$$\mathrm{f3} : \mathbf{\Pi}\ (\mathrm{f} : \mathbb{B} \to \mathbb{B}) \to \mathrm{f}\ \mathbf{tt} = \mathrm{f}\ (\mathrm{f}\ (\mathrm{f}\ \mathbf{ff}))$$
$$\mathrm{f3}\ \mathrm{f} \coloneqq \mathrm{sif}\ (\mathrm{f}\ \mathbf{tt})\ \mathrm{then}\ \mathbf{refl}\ \mathrm{else}\ (\mathrm{sif}\ (\mathrm{f}\ \mathbf{ff})\ \mathrm{then}\ \mathbf{refl}\ \mathrm{else}\ \mathbf{refl})$$

Becomes...

$$\begin{aligned}
\mathrm{f3\text{-}aux}_2 &: \mathbf{\Pi}\ (\mathrm{f} : \mathbb{B} \to \mathbb{B})\ (\mathrm{f}\ \mathbf{tt} \equiv \mathbf{ff})\ \mid\ \mathrm{f}\ \mathbf{ff} \\
&\quad \to \mathrm{f}\ \mathbf{tt} = \mathrm{f}\ (\mathrm{f}\ (\mathrm{f}\ \mathbf{tt}))
\end{aligned}$$
$$\mathrm{f3\text{-}aux}_2\ \mathrm{f}\ \mid\ \mathbf{tt} \coloneqq \mathbf{refl}$$
$$\mathrm{f3\text{-}aux}_2\ \mathrm{f}\ \mid\ \mathbf{ff} \coloneqq \mathbf{refl}$$

$$\begin{aligned}
\mathrm{f3\text{-}aux}_1 &: \mathbf{\Pi}\ (\mathrm{f} : \mathbb{B} \to \mathbb{B})\ \mid\ \mathrm{f}\ \mathbf{tt} \\
&\quad \to \mathrm{f}\ \mathbf{tt} = \mathrm{f}\ (\mathrm{f}\ (\mathrm{f}\ \mathbf{tt}))
\end{aligned}$$
$$\mathrm{f3\text{-}aux}_1\ \mathrm{f}\ \mid\ \mathbf{tt} \coloneqq \mathbf{refl}$$
$$\mathrm{f3\text{-}aux}_1\ \mathrm{f}\ \mid\ \mathbf{ff} \coloneqq \mathrm{call}\ \mathrm{f3\text{-}aux}_2\ \mathrm{f}$$

$$\mathrm{f3\text{-}elab} : \mathbf{\Pi}\ (\mathrm{f} : \mathbb{B} \to \mathbb{B}) \to \mathrm{f}\ \mathbf{tt} = \mathrm{f}\ (\mathrm{f}\ (\mathrm{f}\ \mathbf{tt}))$$
$$\mathrm{f3\text{-}elab} \coloneqq \mathrm{call}\ \mathrm{f3\text{-}aux}_1\ \mathrm{f}$$

Elaborating Case Splits to Top-Level Definitions (SC$^{\text{DEF}}$)

Two new ingredients:

Elaborating Case Splits to Top-Level Definitions (SC^DEF)

Two new ingredients:

- Parameter lists (*telescopes*) of definitions now include convertibility constraints.

Elaborating Case Splits to Top-Level Definitions (SC^{Def})

Two new ingredients:

- Parameter lists (*telescopes*) of definitions now include convertibility constraints.
- Definitions block on neutrals, and reflect appropriate equations.

$$\frac{\begin{gathered}\Xi : \mathrm{Sig} \quad \Gamma : \mathrm{Ctx}\,\Xi \quad A : \mathrm{Ty}\,\Gamma \quad t : \mathrm{Tm}\,\Gamma\,\mathbb{B} \\ u : \mathrm{Tm}\,(\Gamma \rhd t \sim \mathrm{tt})\,A \\ v : \mathrm{Tm}\,(\Gamma \rhd t \sim \mathrm{ff})\,A\end{gathered}}{(\Xi \rhd \Gamma \ \longrightarrow\ A \ \mathrm{sif}\ t\ \mathrm{then}\ u\ \mathrm{else}\ v) : \mathrm{Sig}}$$

Have We Lost Anything?

- Congruence of conversion! Sort of...
  - Distinct case splits are elaborated to different top-level auxiliary definitions.
  - Definitions only reduce after the new equation holds “on-the-nose”.
  - So stuck calls to distinct definitions are never convertible (even if the bodies are).

Have We Lost Anything?

- Congruence of conversion! Sort of...
  - Distinct case splits are elaborated to different top-level auxiliary definitions.
  - Definitions only reduce after the new equation holds "on-the-nose".
  - So stuck calls to distinct definitions are never convertible (even if the bodies are).
- Convertibility of *core terms* is still congruent though!

$$\frac{\mathsf{f}_1 \sim_{\mathsf{SigVar}} \mathsf{f}_2 \quad \delta_1 \sim_{\mathsf{Tms}} \delta_2}{\mathsf{call}\ \mathsf{f}_1\ \delta_1 \sim_{\mathsf{Tm}} \mathsf{call}\ \mathsf{f}_2\ \delta_2}$$

Normalisation is Easy(er)!

- Evaluation can now be defined w.r.t. a single completed TRS.
  - Evaluation halts when it encounters blocked call expressions.

Normalisation is Easy(er)!

- Evaluation can now be defined w.r.t. a single completed TRS.
  - Evaluation halts when it encounters blocked call expressions.
- Unquoting looks up neutral terms in the TRS.

Normalisation is Easy(er)!

- Evaluation can now be defined w.r.t. a single completed TRS.
  - Evaluation halts when it encounters blocked call expressions.
- Unquoting looks up neutral terms in the TRS.
- We still need to obtain the completed TRS in the first place...
  - But, we can now restrict equations however we like!
  - One possible strategy: require that LHSs are disjoint post-normalisation under the prior set of equations.

Conclusion

- Introduced $\mathrm{SC}^{\textsc{Bool}}$: a type theory with contextual equations and **smart if**.
  - Proved soundness (consistency relative to MLTT) by constructing a model

Conclusion

- Introduced $\mathrm{SC}^{\textsc{Bool}}$: a type theory with contextual equations and **smart if**.
    - Proved soundness (consistency relative to MLTT) by constructing a model
- Introduced $\mathrm{SC}^{\textsc{Def}}$: a type theory where equations are reflected at top level definitions.
    - Also proved soundness, and decidability of conversion (via normalisation by evaluation)

Conclusion

- Introduced $\mathrm{SC}^{\textsc{Bool}}$: a type theory with contextual equations and **smart if**.
  - Proved soundness (consistency relative to MLTT) by constructing a model
- Introduced $\mathrm{SC}^{\textsc{Def}}$: a type theory where equations are reflected at top level definitions.
  - Also proved soundness, and decidability of conversion (via normalisation by evaluation)
- Implemented prototype $\mathrm{SC}^{\textsc{Bool}}$ typechecker in Haskell

Conclusion

- Introduced $\mathrm{SC}^{\textsc{Bool}}$: a type theory with contextual equations and **smart if**.
  - Proved soundness (consistency relative to MLTT) by constructing a model
- Introduced $\mathrm{SC}^{\textsc{Def}}$: a type theory where equations are reflected at top level definitions.
  - Also proved soundness, and decidability of conversion (via normalisation by evaluation)
- Implemented prototype $\mathrm{SC}^{\textsc{Bool}}$ typechecker in Haskell
- Formal results are mostly mechanised in Agda

Future Work

- Support a wider class of equations
  - Neutral RHSs and neutral-typed
  - Inductive types (occurs check!)
  - Non-disjoint LHSs (via completion - would need to find a well-founded order)
  - "Equation schemes"

[9] Cockx 2019, *Type Theory Unchained: Extending Agda with User-Defined Rewrite Rules.*

[10] Pujet and Tabareau 2022, *Observational equality: now for good.*

[11] Cohen et al. 2015, *Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom.*

Future Work

- Support a wider class of equations
  - Neutral RHSs and neutral-typed
  - Inductive types (occurs check!)
  - Non-disjoint LHSs (via completion - would need to find a well-founded order)
  - "Equation schemes"
- Implement (as a language extension) in Agda!
  - Investigate interactions with other type system features (e.g. global **REWRITE** rules[9] or observational[10]/cubical[11] equality)

[9] Cockx 2019, *Type Theory Unchained: Extending Agda with User-Defined Rewrite Rules.*

[10] Pujet and Tabareau 2022, *Observational equality: now for good.*

[11] Cohen et al. 2015, *Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom.*

Thank You!

Related Work

- **with**-abstractions/**rewrite**/pattern-matching lambdas[12]
- Tactics[13]
- Global **REWRITE** rules[14]
- "Controlling computation in type theory, locally"[15]
- Strict η for coproducts[16]
- Extension Types[17]
- Coq Modulo Theory (CoqMT)[18]

---

[12] McBride and McKinna 2004, *The view from the left.*
[13] Selsam and Moura 2016, *Congruence Closure in Intensional Type Theory.*
[14] Cockx 2019, *Type Theory Unchained: Extending Agda with User-Defined Rewrite Rules.*
[15] Winterhalter 2025, *Controlling computation in type theory, locally.*
[16] Maillard 2024, *Splitting Booleans with Normalization-by-Evaluation.*
[17] Riehl and Shulman 2017, *A type theory for synthetic ∞-categories.*
[18] Strub 2010, *Coq Modulo Theory.*

Losing Congruence of Conversion

The same phenomenon occurs in Agda:

$$\begin{aligned} \text{not-eq} &\;:\; \text{not}_1\ \text{b} \;\mathbf{=}\; \text{not}_2\ \text{b} \\ \text{not-eq} &\;\coloneqq\; \mathbf{refl} \end{aligned}$$

$$\begin{aligned} \text{not}_1 \;&\coloneqq\; \boldsymbol{\lambda}\ \mathbf{where}\ \mathbf{tt} \to \mathbf{ff} \\ &\qquad\qquad\quad\ \ \mathbf{ff} \to \mathbf{tt} \\ \text{not}_2 \;&\coloneqq\; \boldsymbol{\lambda}\ \mathbf{where}\ \mathbf{tt} \to \mathbf{ff} \\ &\qquad\qquad\quad\ \ \mathbf{ff} \to \mathbf{tt} \end{aligned}$$

Losing Congruence of Conversion

The same phenomenon occurs in Agda:

$$\begin{aligned} \text{not-eq} &: \text{not}_1\ b \ \mathbf{=}\ \text{not}_2\ b \\ \text{not-eq} &\coloneqq \mathbf{refl} \end{aligned}$$

$$\begin{aligned} \text{not}_1 &\coloneqq \boldsymbol{\lambda}\ \textbf{where tt} \to \textbf{ff} \\ &\qquad\qquad\quad \textbf{ff} \to \textbf{tt} \\ \text{not}_2 &\coloneqq \boldsymbol{\lambda}\ \textbf{where tt} \to \textbf{ff} \\ &\qquad\qquad\quad \textbf{ff} \to \textbf{tt} \end{aligned}$$

...:307.7-11: error: [UnequalTerms]
$(\boldsymbol{\lambda}\ \{\textbf{tt} \to \textbf{ff}; \textbf{ff} \to \textbf{tt}\})$ b !=
$(\boldsymbol{\lambda}\ \{\textbf{tt} \to \textbf{ff}; \textbf{ff} \to \textbf{tt}\})$ b of **type** $\mathbb{B}$
Because they are distinct extended lambdas: one is defined at
  ...:298.8-299.30
and the other at
  ...:300.8-301.30,
so they have different internal representations.
when checking that the expression **refl** has **type** $\text{not}_1\ b \ \mathbf{=}\ \text{not}_2\ b$

Losing Congruence of Conversion

Easily circumvented in practice!
The programmer can just repeat the same case split.

$$\begin{aligned}
\text{not-eq} &\ :\ \text{not}_1\ b\ \mathbf{=}\ \text{not}_2\ b \\
\text{not-eq}\ \{b &\ :\equiv\ \mathbf{tt}\}\ :\equiv\ \mathbf{refl} \\
\text{not-eq}\ \{b &\ :\equiv\ \mathbf{ff}\}\ :\equiv\ \mathbf{refl}
\end{aligned}$$

$$\begin{aligned}
\text{not}_1 &\ :\equiv\ \boldsymbol{\lambda}\ \mathbf{where}\ \mathbf{tt} \to \mathbf{ff} \\
&\qquad\qquad\qquad\ \ \mathbf{ff} \to \mathbf{tt} \\
\text{not}_2 &\ :\equiv\ \boldsymbol{\lambda}\ \mathbf{where}\ \mathbf{tt} \to \mathbf{ff} \\
&\qquad\qquad\qquad\ \ \mathbf{ff} \to \mathbf{tt}
\end{aligned}$$

Bibliography I

Abel, Andreas, Nils Anders Danielsson, and Oskar Eriksson (2023). "A Graded Modal Dependent Type Theory with a Universe and Erasure, Formalized". In: *Proc. ACM Program. Lang.* 7.ICFP, pp. 920–954. DOI: 10.1145/3607862.

Altenkirch, Thorsten (2011). "The case of the smart case. How to implement conditional convertibility?" In: *Presented at NII Shonan Seminar 007*. URL: https://shonan.nii.ac.jp/archives/seminar/007/files/2011/09/altenkirch_slides.pdf.

Altenkirch, Thorsten and Ambrus Kaposi (2017). "Normalisation by Evaluation for Type Theory, in Type Theory". In: *Log. Methods Comput. Sci.* 13.4. DOI: 10.23638/LMCS-13(4:1)2017.

Baader, Franz and Tobias Nipkow (1998). *Term Rewriting and All That*. Cambridge university press.

Bibliography II

- Berger, Ulrich and Helmut Schwichtenberg (1991). "An Inverse of the Evaluation Functional for Typed lambda-calculus". In: *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991.* IEEE Computer Society, pp. 203–211. DOI: `10.1109/LICS.1991.151645`.
- Buzzard, Kevin and contributors (2025). *FLT. Ongoing Lean formalisation of the proof of Fermat's Last Theorem.* URL: `https://github.com/ImperialCollegeLondon/FLT`.
- Cockx, Jesper (2019). "Type Theory Unchained: Extending Agda with User-Defined Rewrite Rules". In: *25th International Conference on Types for Proofs and Programs, TYPES 2019, June 11-14, 2019, Oslo, Norway.* Ed. by Marc Bezem and Assia Mahboubi. Vol. 175. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2:1–2:27. DOI: `10.4230/LIPICS.TYPES.2019.2`.

Bibliography III

Cohen, Cyril et al. (2015). "Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom". In: *21st International Conference on Types for Proofs and Programs, TYPES 2015, May 18-21, 2015, Tallinn, Estonia*. Ed. by Tarmo Uustalu. Vol. 69. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 5:1–5:34. DOI: 10.4230/LIPICS.TYPES.2015.5.

Escardó, Martín H. and contributors (2025). *TypeTopology*. Agda development. URL: https://github.com/martinescardo/TypeTopology.

Maillard, Kenji (2024). "Splitting Booleans with Normalization-by-Evaluation". In: *Presented at 30th International Conference on Types for Proofs and Programs TYPES 2024*, p. 121. URL: https://kenji.maillard.blue/Presentations/boolextTypes24.pdf.

McBride, Conor and James McKinna (2004). "The view from the left". In: *J. Funct. Program.* 14.1, pp. 69–111. DOI: 10.1017/S0956796803004829.

Bibliography IV

Pujet, Loïc and Nicolas Tabareau (2022). "Observational equality: now for good". In: *Proc. ACM Program. Lang.* 6.POPL, pp. 1–27. DOI: 10.1145/3498693.

Riehl, Emily and Michael Shulman (2017). "A type theory for synthetic ∞-categories". In: *Higher Structures* 1 (1), pp. 147–224. DOI: 10.21136/hs.2017.06.

Selsam, Daniel and Leonardo de Moura (2016). "Congruence Closure in Intensional Type Theory". In: *Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings*. Ed. by Nicola Olivetti and Ashish Tiwari. Vol. 9706. Lecture Notes in Computer Science. Springer, pp. 99–115. DOI: 10.1007/978-3-319-40229-1_8.

Bibliography V

Strub, Pierre-Yves (2010). “Coq Modulo Theory”. In: *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*. Ed. by Anuj Dawar and Helmut Veith. Vol. 6247. Lecture Notes in Computer Science. Springer, pp. 529–543. DOI: 10.1007/978-3-642-15205-4_40.

The Agda Team (2024a). *Lambda Abstraction. The Agda 2.7.0.1 User Manual*. URL: https://agda.readthedocs.io/en/v2.7.0.1/language/lambda-abstraction.html (visited on 06/10/2025).

— (2024b). *With-Abstraction. The Agda 2.7.0.1 User Manual*. URL: https://agda.readthedocs.io/en/v2.7.0.1/language/with-abstraction.html (visited on 01/20/2025).

Bibliography VI

Various Contributors (2024). *Relation.Binary.PropositionalEquality.Properties. The Agda Standard Library 2.1.1.* URL: https://agda.github.io/agda-stdlib/v2.1.1/Relation.Binary.PropositionalEquality.Properties.html (visited on 06/18/2025).

Winterhalter, Théo (2025). "Controlling computation in type theory, locally". In: *Presented at the EuroProofNet WG6 meeting 2025*. URL: https://theowinterhalter.github.io/res/slides/local-comp-wg6-25.pdf.