Generating Higher Identity Proofs in Homotopy Type Theory

Thibaut Benjamin

27 September 2025

Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles

Dealing with identity types in Rocq

Example: transitivity lemma

$$x \overset{p}{=\!=} y \overset{q}{=\!=} z \quad \longrightarrow \quad x \overset{\mathrm{trans}(p,q)}{=\!=\!=\!=} z$$

Example: transitivity lemma

$$x \overset{p}{=\!=\!=} y \overset{q}{=\!=\!=} z \longrightarrow x \overset{\mathrm{trans}(p,q)}{=\!=\!=\!=\!=} z$$

```
Lemma trans : forall {A : Type} {x y z : A}
                     (p : x = y) (q : y = z), x = z.
Proof.
  intros.
  induction q.
  induction p.
  reflexivity.
Defined.
```

A direct definition of the transitivity proof term

$$x \overset{p}{=\!=} y \overset{q}{=\!=} z \quad \longrightarrow \quad x \overset{\mathrm{trans}(p,q)}{=\!=\!=\!=} z$$

A direct definition of the transitivity proof term

$$x \overset{p}{=\!=} y \overset{q}{=\!=} z \longrightarrow x \overset{\mathrm{trans}(p,q)}{=\!=\!=\!=} z$$

```
Definition trans : forall {A : Type} {x y z : A}
                          (p : x = y) (q : y = z), x = z :=
  fun A z y z p q ⇒
    match q with
    | eq_refl ⇒
      match p with
      | eq_refl ⇒ eq_refl
      end
    end.
```

A more involved example: "whiskering"

A more involved example: “whiskering”


```
Definition whisk : forall {A : Type} {x y z : A}
                          {p q : x = y} (a : p = q)
                          (r : y = z), trans p r = trans q r :=
  fun A z y z p q a r ⇒
    match r with
    | eq_refl ⇒
      match a with
      | eq_refl ⇒
        match p with
        |eq_refl ⇒ eq_refl
        end
      end
    end.
```

Streamlining these arguments

- These arguments are purely structural
  No transport, no interaction of identity with other structure

Streamlining these arguments

- These arguments are purely structural
  No transport, no interaction of identity with other structure

- The proof strategies are very similar:
  Pattern match every equality against `eq_refl`, and the proof is `eq_refl`

Streamlining these arguments

- These arguments are purely structural
  No transport, no interaction of identity with other structure
- The proof strategies are very similar:
  Pattern match every equality against `eq_refl`, and the proof is `eq_refl`
- Yet, they are not trivial
  See the Eckmann-Hilton example at the end of the talk

Streamlining these arguments

- These arguments are purely structural
  No transport, no interaction of identity with other structure
- The proof strategies are very similar:
  Pattern match every equality against `eq_refl`, and the proof is `eq_refl`
- Yet, they are not trivial
  See the Eckmann-Hilton example at the end of the talk
- Aim of the talk: streamline as much of these arguments as possible

Identity types, weak ω-groupoids and Catt

Weak ω-groupoids and identity types

- Weak ω-groupoids are the algebraic structure that describe identity types.
  They describe exaclty the arguments we want to streamline

Weak ω-groupoids and identity types

- Weak ω-groupoids are the algebraic structure that describe identity types.
  They describe exaclty the arguments we want to streamline
- They are a flavour of higher-dimensional categories.

Weak ω-groupoids and identity types

- Weak ω-groupoids are the algebraic structure that describe identity types.
  They describe exaclty the arguments we want to streamline

- They are a flavour of higher-dimensional categories.

- In this talk, we present the language Catt, a DSL to work with weak ω-categories.
  We will not introduce formally the theory of weak ω-groupoids

Weak ω-groupoids and identity types

- Weak ω-groupoids are the algebraic structure that describe identity types.
  They describe exaclty the arguments we want to streamline
- They are a flavour of higher-dimensional categories.
- In this talk, we present the language Catt, a DSL to work with weak ω-categories.
  We will not introduce formally the theory of weak ω-groupoids
- Weak ω-groupoids are a particular case of weak ω-categories.
  We use a language for the latter, for historical reasons, but we will ignore the difference in this talk

Pasting diagrams

Pasting diagrams are equality schemes that can be completely pattern-matched away against `eq_refl`[1]

[1] not all of them

Pasting diagrams

Pasting diagrams are equality schemes that can be completely pattern-matched away against `eq_refl`[1]

Examples:

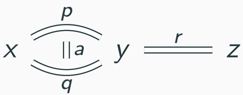
$$x \overset{p}{=\!=} y \overset{q}{=\!=} z \qquad \rightsquigarrow \qquad (x(p)y(q)z)$$

[1] not all of them

Pasting diagrams are equality schemes that can be completely pattern-matched away against `eq_refl`[1]

Examples:

$$x \overset{p}{=\!=\!=} y \overset{q}{=\!=\!=} z \qquad \rightsquigarrow \qquad (x(p)y(q)z)$$



$$\rightsquigarrow \qquad (x(p(a)q)y(r)z)$$

[1] not all of them

Obstructions to being a pasting diagram

Not all equality schemes can be completely pattern-matched away. The obstructions are topological

Obstructions to being a pasting diagram

Not all equality schemes can be completely pattern-matched away. The obstructions are topological

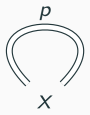
Examples:

Pattern-matching the loop onto `eq_refl` requires the axiom K

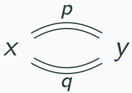
Obstructions to being a pasting diagram

Not all equality schemes can be completely pattern-matched away. The obstructions are topological

Examples:



Pattern-matching the loop onto `eq_refl` requires the axiom K



Pattern-matching q onto `eq_refl` creates loop

The language Catt[2]

- types : $\star$, and $a \sim b$, where a, b are terms of the same type
  Intuition:$a \sim b$ are abstract equalities

[2] terms and conditions: groupoids vs categories

The language Catt[2]

- types : $\star$, and $a \sim b$, where a, b are terms of the same type
  Intuition:$a \sim b$ are abstract equalities
- terms : generated by variables and $\mathrm{coh}(\Gamma, A)$ where Γ is a pasting diagram and A is a type in Γ
  Intuition: In a pasting diagram, every pair of terms of the same type are equal, by pattern-matching the entire diagram away

[2] terms and conditions: groupoids vs categories

The language Catt[2]

- types : $\star$, and $a \sim b$, where a, b are terms of the same type
  Intuition:$a \sim b$ are abstract equalities
- terms : generated by variables and $\mathrm{coh}(\Gamma, A)$ where Γ is a pasting diagram and A is a type in Γ
  Intuition: In a pasting diagram, every pair of terms of the same type are equal, by pattern-matching the entire diagram away
- In the implemented language, additionnal conditions are put on the type A in a coherence, to model categories.
  I hope to implement the version presented here soon

[2]terms and conditions: groupoids vs categories

Live demo

Generating identity proof from Catt terms

Principle

- Use terms in Catt to generate identity proofs in Rocq

Principle

- Use terms in Catt to generate identity proofs in Rocq
- Boilerplate inductive mechanism for the structure of the theory

Principle

- Use terms in Catt to generate identity proofs in Rocq
- Boilerplate inductive mechanism for the structure of the theory
- Base case: $\mathrm{coh}(\Gamma, A)$ is handled by pattern-matching away all equalities described by Γ and the proof is then `eq_refl`.

Principle

- Use terms in Catt to generate identity proofs in Rocq
- Boilerplate inductive mechanism for the structure of the theory
- Base case: $\mathsf{coh}(\Gamma, A)$ is handled by pattern-matching away all equalities described by Γ and the proof is then `eq_refl`.
- Implicit arguments are managed automatically

Live demo

Building complex proofs

Proof automation in Catt

- Catt is a small and simple language, with a direct focus
  Contrarily to Rocq, which is wide and general purpose

Proof automation in Catt

- Catt is a small and simple language, with a direct focus
  Contrarily to Rocq, which is wide and general purpose

- We can algorithmically manipulate Catt as a language to generate proof-terms.
  Several meta-operations are already implemented
  Suspension, opposites, functoriality, naturality...

Proof automation in Catt

- Catt is a small and simple language, with a direct focus
  Contrarily to Rocq, which is wide and general purpose

- We can algorithmically manipulate Catt as a language to generate proof-terms. Several meta-operations are already implemented
  Suspension, opposites, functoriality, naturality...

- Our export to Rocq allows to leverage these to build proofs on the structure of identity

The Eckmann-Hilton argument

- The Eckmann-Hilton argument is an important argument in topology
  It is tightly connected to homotopy theory, and in Rocq, is proven by purely structural manipulation on identity types

The Eckmann-Hilton argument

- The Eckmann-Hilton argument is an important argument in topology
  It is tightly connected to homotopy theory, and in Rocq, is proven by purely structural manipulation on identity types
- It also provides a refutation axiom K
  It allows one to construct an equality between a term and itself, which in some models is not the reflexivity

The Eckmann-Hilton argument: live demo

Thank you!