Martin-Löf à la Coq
and other tales of formalized type theories

Kenji Maillard

Gallinette Project-Team, Inria, Nantes, France

Séminaire LIMD, Chambéry, November 2024

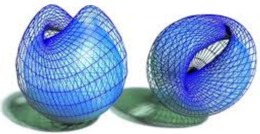
Proof Assistants ?

A software/programming language to:

- Specify theorems,
- Write proofs,
- Check mechanically the correctness of theses proofs.

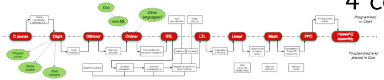
Proof Assistants at Work

OS Microkernel



Sphere eversion

4 color theoreom



CompCert

Marton's "Polynomial Freiman-Ruzsa conjecture"

Challenges: Correctness, Expressivity, Usability, Scalability, Inter-operability, Efficiency

Demo ?

Martin-Löf Type Theory and its Implementations

Martin-Löf logical framework + type formers $(\mathrm{Type}, \Pi, \Sigma, \mathrm{Id}\, A\, x\, y, \ldots)$

$$\Gamma \vdash \qquad \Gamma \vdash A \qquad \Gamma \vdash A \equiv B$$

$$\Gamma \vdash t : A \qquad \Gamma \vdash t \equiv u : A$$

- Idealized metatheory of proofs assistants based on dependent types.
- Practical implementation $\sim$ algorithms deciding each judgements

Martin-Löf Type Theory and its Implementations

Martin-Löf logical framework + type formers $(\mathrm{Type}, \Pi, \Sigma, \mathrm{Id}\, A\, x\, y, \ldots)$

$$\Gamma \vdash \qquad \Gamma \vdash A \qquad \Gamma \vdash A \equiv B$$

$$\Gamma \vdash t : A \qquad \Gamma \vdash t \equiv u : A$$

$$\textsc{App}\ \frac{\Gamma \vdash t : (x : A) \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\ u : B[x := u]} \qquad \textsc{Lam}\ \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : (x : A) \to B}$$

$$\textsc{Conv}\ \frac{\Gamma \vdash t : A \qquad \Gamma \vdash A \cong B}{\Gamma \vdash t : B} \qquad \textsc{Beta}\ \frac{\Gamma, x : A \vdash t : B \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda x : A.t)\ u \cong t[x := u] : B[x := u]}$$

Formalized Metatheory of Type Theory: Why ?

Logical aspects

- Relative consistency
- Normalization/Canonicity
- Proof-theoretical bounds

Certification aspects

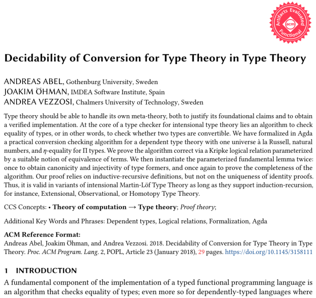
- Correctness, completeness and totality of the implemented algorithms

Formalized Metatheory of Type Theory: State of the art

MetaCoq

Normalization oracle

Logical relations for MLTT



Decidability of Conversion for Type Theory in Type Theory

ANDREAS ABEL, Gothenburg University, Sweden
JOAKIM ÖHMAN, IMDEA Software Institute, Spain
ANDREA VEZZOSI, Chalmers University of Technology, Sweden

Type theory should be able to handle its own meta-theory, both to justify its foundational claims and to obtain a verified implementation. At the core of a type checker for intensional type theory lies an algorithm to check equality of types, or in other words, to check whether two types are convertible. We have formalized in Agda a practical conversion checking algorithm for a dependent type theory with one universe à la Russell, natural numbers, and η-equality for Π types. We prove the algorithm correct via a Kripke logical relation parameterized by a suitable notion of equivalence of terms. We then instantiate the parameterized fundamental lemma twice: once to obtain canonicity and injectivity of type formers, and once again to prove the completeness of the algorithm. Our proof relies on inductive-recursive definitions, but not on the uniqueness of identity proofs. Thus, it is valid in variants of intensional Martin-Löf Type Theory as long as they support induction-recursion, for instance, Extensional, Observational, or Homotopy Type Theory.

CCS Concepts: • **Theory of computation → Type theory**; *Proof theory*;

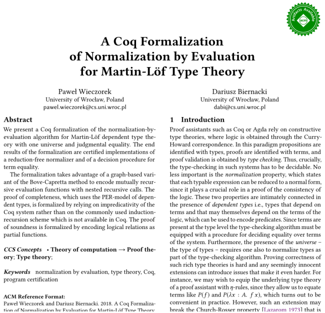
Additional Key Words and Phrases: Dependent types, Logical relations, Formalization, Agda

ACM Reference Format:
Andreas Abel, Joakim Öhman, and Andrea Vezzosi. 2018. Decidability of Conversion for Type Theory in Type Theory. *Proc. ACM Program. Lang.* 2, POPL, Article 23 (January 2018), 29 pages. https://doi.org/10.1145/3158111

1 INTRODUCTION

A fundamental component of the implementation of a typed functional programming language is an algorithm that checks equality of types; even more so for dependently-typed languages where

Rely on Induction-Recursion



A Coq Formalization of Normalization by Evaluation for Martin-Löf Type Theory

Paweł Wieczorek
University of Wrocław, Poland
pawel.wieczorek@cs.uni.wroc.pl

Dariusz Biernacki
University of Wrocław, Poland
dabi@cs.uni.wroc.pl

Abstract

We present a Coq formalization of the normalization-by-evaluation algorithm for Martin-Löf dependent type theory with one universe and judgmental equality. The end results of the formalization are certified implementations of a reduction-free normalizer and of a decision procedure for term equality.

The formalization takes advantage of a graph-based variant of the Bove-Capretta method to encode mutually recursive evaluation functions with nested recursive calls. The proof of completeness, which uses the PER-model of dependent types, is formalized by relying on impredicativity of the Coq system rather than on the commonly used induction-recursion scheme which is not available in Coq. The proof of soundness is formalized by encoding logical relations as partial functions.

CCS Concepts • **Theory of computation → Proof theory**; **Type theory**;

Keywords normalization by evaluation, type theory, Coq, program certification

ACM Reference Format:
Paweł Wieczorek and Dariusz Biernacki. 2018. A Coq Formalization of Normalization by Evaluation for Martin-Löf Type Theory

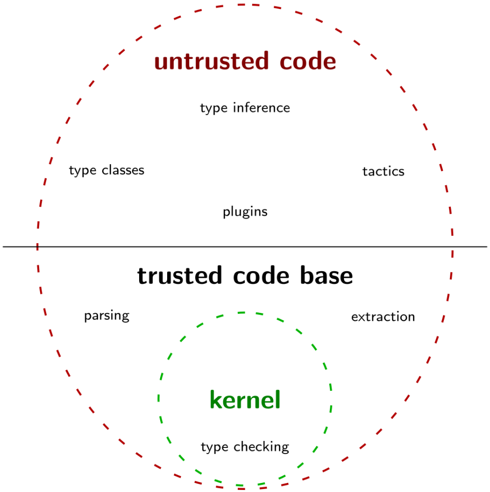
1 Introduction

Proof assistants such as Coq or Agda rely on constructive type theories, where logic is obtained through the Curry-Howard correspondence. In this paradigm propositions are identified with types, proofs are identified with terms, and proof validation is obtained by *type checking*. Thus, crucially, the type-checking in such systems has to be decidable. No less important is the *normalization* property, which states that each typable expression can be reduced to a normal form, since it plays a crucial role in a proof of the consistency of the logic. These two properties are intimately connected in the presence of *dependent types* i.e., types that depend on terms and that may themselves depend on the terms of the logic, which can be used to encode predicates. Since terms are present at the type level the type-checking algorithm must be equipped with a procedure for deciding equality over terms of the system. Furthermore, the presence of the *universe* – the type of types – requires one also to normalize types as part of the type-checking algorithm. Proving correctness of such rich type theories is hard and any seemingly innocent extensions can introduce issues that make it even harder. For instance, we may wish to equip the underlying type theory of a proof assistant with η-rules, since they allow us to equate terms like $P(f)$ and $P(\lambda x : A.\ f\,x)$, which turns out to be convenient in practice. However, such an extension may break the Church-Rosser property [Lazarom 1973] that is

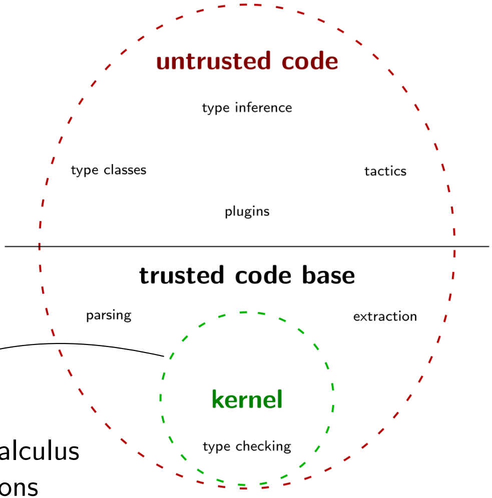
Rely on Impredicativity

MetaCoq

The implementation of Coq (Idealised)



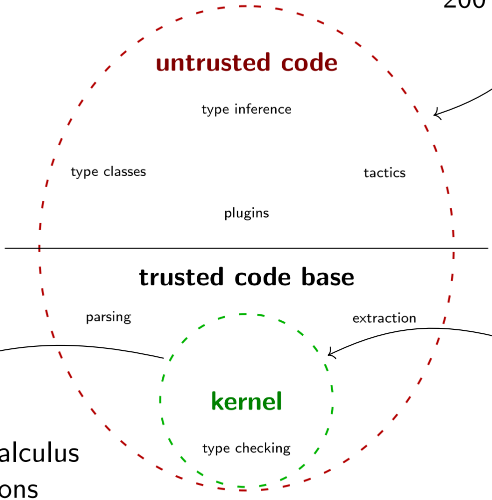
The implementation of Coq (Idealised)



PCUIC
Polymorphic Cumulative calculus
of Inductive Constructions

The implementation of Coq (Idealised)

200 kLoc of Ocaml

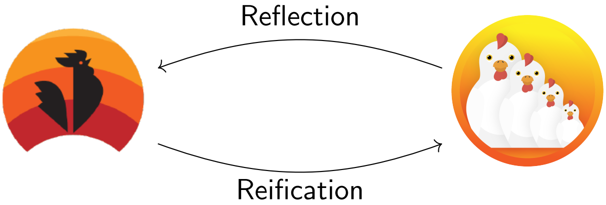


18 kLoc
≈1 critical bug/year

PCUIC
Polymorphic Cumulative calculus
of Inductive Constructions

Representing Coq in Coq: Template-Coq

Representing Coq in Coq: Template-Coq



Representing Coq in Coq: Template-Coq

Reflection

Reification

```
Inductive term :=
| tRel (n : nat)
| tVar (i : ident) (* For free variables (e.g. in a goal) *)
| tEvar (n : nat) (l : list term)
| tSort (u : Universe.t)
| tProd (na : aname) (A B : term)
| tLambda (na : aname) (A t : term)
| tLetIn (na : aname) (b B t : term) (* let na := b : B in t *)
| tApp (u v : term)
| tConst (k : kername) (ui : Instance.t)
| tInd (ind : inductive) (ui : Instance.t)
| tConstruct (ind : inductive) (n : nat) (ui : Instance.t)
| tCase (indn : case_info) (p : predicate term) (c : term) (brs : list (branch term))
| tProj (p : projection) (c : term)
| tFix (mfix : mfixpoint term) (idx : nat)
| tCoFix (mfix : mfixpoint term) (idx : nat)
| tPrim (prim : prim_val term).
```

```
Inductive typing (Σ : global_env_ext) (Γ : context) : term -> term -> Type :=
| type_Rel : forall n decl,
    wf_local Σ Γ ->
    nth_error Γ n = Some decl ->
    Σ ;;; Γ |- tRel n : lift0 (S n) decl.(decl_type)

| type_Sort : forall s,
    wf_local Σ Γ ->
    wf_universe Σ s ->
    Σ ;;; Γ |- tSort s : tSort (Universe.super s)

| type_Prod : forall na A B s1 s2,
    Σ ;;; Γ |- A : tSort s1 ->
    Σ ;;; Γ ,, vass na A |- B : tSort s2 ->
    Σ ;;; Γ |- tProd na A B : tSort (Universe.sort_of_product s1 s2)
```

Metatheorems and Results from MetaCoq

Metatheorems

- Confluence of reduction
- Injectivity of type formers
- Subject reduction
- Principality wrt. cumulativity

Products

- Verified type-checker and conversion checker
- That can be extracted to Ocaml
- Verified extraction to ocaml/malfunction

Metatheorems and Results from MetaCoq

Metatheorems

- Confluence of reduction
- Injectivity of type formers
- Subject reduction
- Principality wrt. cumulativity

Assumes normalization as an oracle !

```
(* AXIOM Postulate existence of a guard condition checker *)

Inductive FixCoFix : Type := Fix | CoFix.

Class GuardChecker :=
{ (* guard check for both fixpoints (Fix) and cofixpoints (CoFix)  *)
  guard : FixCoFix -> global_env_ext -> context -> mfixpoint term -> Prop ;
}.

Axiom guard_checking : GuardChecker.
```

Products

- Verified type-checker and conversion checker
- That can be extracted to Ocaml
- Verified extraction to ocaml/malfunction

Martin-Löf à la Coq: Mechanized Logical Relation in Coq for MLTT

j.w.w. A. Adjedj, M. Lennon-Bertrand, P.-M. Pédrot, L. Pujet

Main goal & Hauptsatz

Informally: Normalization of Coq in Coq

Main goal & Hauptsatz

Informally: Normalization of Coq in Coq

Main goal & Hauptsatz

Informally: Normalization of Coq in Coq

Theorem : Typing and conversion are decidable for MLTT
wrt the theory of Coq .

MLTT with Π, Σ, $\mathbb{0}$, $\mathbb{1}$, $\mathbb{N}$, List

The theory of Coq: PCUIC

Main goal & Hauptsatz

Informally: Normalization of Coq in Coq

Theorem : Typing and conversion are decidable for MLTT with 1 universe wrt the theory of Coq with $1 + 5$ universes.

MLTT with Π, Σ, $\mathbb{0}$, $\mathbb{1}$, $\mathbb{N}$, List

The theory of Coq: PCUIC

Main goal & Hauptsatz

Informally: Normalization of Coq in Coq

Theorem : Typing and conversion are decidable for MLTT with 1 universe wrt the theory of Coq with $1 + 5$ universes.

MLTT with Π, Σ, $\mathbb{0}$, $\mathbb{1}$, $\mathbb{N}$, List

The theory of Coq: PCUIC

Current gap: indexed inductive types and a hierarchy of universes.

Towards decidability

Declarative typing

- Free standing conversion rule

$$\frac{\Gamma \vdash_{\mathsf{de}} t : A \qquad \Gamma \vdash_{\mathsf{de}} A \cong B}{\Gamma \vdash_{\mathsf{de}} t : B}$$

- Conversion mixes arbitrary uses of congruence, computation (β), extensionality and transitivity steps.

Towards decidability

Declarative typing

- Free standing conversion rule

$$\frac{\Gamma \vdash_{\mathsf{de}} t : A \qquad \Gamma \vdash_{\mathsf{de}} A \cong B}{\Gamma \vdash_{\mathsf{de}} t : B}$$

- Conversion mixes arbitrary uses of congruence, computation (β), extensionality and transitivity steps.

Algorithmic typing (bidirectional)

- Conversion constrained to phase changes

$$\frac{\Gamma \vdash_{\mathsf{al}} t \rhd A \qquad \Gamma \vdash_{\mathsf{al}} A \cong B}{\Gamma \vdash_{\mathsf{al}} t \lhd B}$$

- Conversion guided by the terms: alternating weak-head reduction and syntax directed congruences/extensionality rules

How can we compare the two presentations of MLTT?

Bridging the gap between declarative and algorithmic

How can we compare the two presentations of MLTT?

Algorithmic $\rightarrow$ Declarative: Admissibility of algorithmic rules

Bridging the gap between declarative and algorithmic

How can we compare the two presentations of MLTT?

Algorithmic $\to$ **Declarative:** Admissibility of algorithmic rules ✓

Bridging the gap between declarative and algorithmic

How can we compare the two presentations of MLTT?

Algorithmic $\to$ **Declarative:** Admissibility of algorithmic rules ✓

Declarative $\to$ **Algorithmic:** Need to show that every derivation has a canonical form

Logical Relations, formally in Coq

Key Idea:

- Attach to every type a notion of reduction to a canonical form $\Gamma \Vdash A$
- Use witnesses $[A] : \Gamma \Vdash A$ to define a similar notion $\Gamma \Vdash t : A/[A]$ for terms of type A
- Show that the definition enjoy many stability properties

Fundamental lemma: Mutual induction on the judgements, using all the derived properties

$$\begin{array}{rcl} \Gamma \vdash_{\mathsf{de}} A & \Longrightarrow & \Gamma \Vdash A \\ \Gamma \vdash_{\mathsf{de}} t : A & \Longrightarrow & [A] : \Gamma \Vdash A \quad \wedge \quad \Gamma \Vdash t : A/[A] \end{array}$$

Formal development: $\sim$ 10k loc (4k specs/6k proofs); overall development $\sim$ 25k loc

Beyond MLTT

Extending MLTT: Why would we care ?

Add new proof principles:

- Uniqueness of identity proofs (`UIP`)
- Function extensionality (`funext`)
- Negation of `funext`
- Quotients
- Univalence principle
- Markov principle
- Parametricity
- Church Thesis

Account for existing programming features:

- Subtyping
- Exceptions
- Read access to a global environment
- Dynamic type
- Non-determinism (?)
- Probabilistic choices (?)

Functorial structure on List

The type former $\mathrm{List} : \mathrm{Type} \to \mathrm{Type}$ can be endowed with a

$$\texttt{map} \;:\; (A \to B) \to \mathrm{List}\, A \to \mathrm{List}\, B$$

$$\begin{aligned} \texttt{map}\; f\; [] &= [] \\ \texttt{map}\; f\; (hd :: tl) &= f\, hd :: \texttt{map}\, f\, tl \end{aligned}$$

Functorial structure on List

The type former $\mathrm{List} : \mathrm{Type} \to \mathrm{Type}$ can be endowed with a

$$\texttt{map} \;:\; (A \to B) \to \mathrm{List}\,A \to \mathrm{List}\,B$$

$$\begin{aligned}
\texttt{map}\;\; f\;\; [] &= [] \\
\texttt{map}\;\; f\;\; (hd :: tl) &= f\,hd :: \texttt{map}\, f\, tl
\end{aligned}$$

Functor laws:

$$\texttt{map}\ \mathrm{id} \equiv \mathrm{id} \qquad\qquad \texttt{map}\ f \circ \texttt{map}\ g \equiv \texttt{map}\ (f \circ g)$$

Functorial structure on List

The type former $\mathrm{List} : \mathrm{Type} \to \mathrm{Type}$ can be endowed with a

$$\mathtt{map} \;:\; (A \to B) \to \mathrm{List}\,A \to \mathrm{List}\,B$$

$$\begin{array}{lllll} \mathtt{map} & f & [] & = & [] \\ \mathtt{map} & f & (hd :: tl) & = & f\,hd :: \mathtt{map}\,f\,tl \end{array}$$

Functor laws:

$$\mathtt{map}\ \mathrm{id} \equiv \mathrm{id} \qquad\qquad \mathtt{map}\ f \circ \mathtt{map}\ g \equiv \mathtt{map}\ (f \circ g)$$

Not validated on neutrals !

$$A : \mathrm{Type}, l : \mathrm{List}\,A \nvdash \mathtt{map}\,\mathrm{id}_A\,l \equiv l : \mathrm{List}\,A$$

Functor laws for lists, Mechanized !

- Extend MLTT with functor laws on List

Functor laws for lists, Mechanized !

- Extend MLTT with functor laws on List
- Add reduction rules for map composition

$$\texttt{map}\ f\ (\texttt{map}\ g\ l) \rightsquigarrow \texttt{map}\ (f \circ g)\ l \quad \text{for neutral } l$$

Functor laws for lists, Mechanized !

- Extend MLTT with functor laws on List
- Add reduction rules for map composition

$$\texttt{map}\ f\ (\texttt{map}\ g\ l) \rightsquigarrow \texttt{map}\ (f \circ g)\ l \quad \text{for neutral } l$$

- Extend the logical relation with equations on neutrals

$$\frac{\Gamma \vdash f \equiv \mathrm{id}_A : A \to A \qquad \Gamma \vdash l \equiv l' : \mathrm{List}\, A \qquad l, l' \text{ neutrals}}{\Gamma \vdash \texttt{map}\ f\ l \equiv l' : \mathrm{List}\, A}$$

Functor laws for lists, Mechanized !

- Extend MLTT with functor laws on List
- Add reduction rules for map composition

$$\texttt{map}\ f\ (\texttt{map}\ g\ l) \rightsquigarrow \texttt{map}\ (f \circ g)\ l \quad \text{for neutral } l$$

- Extend the logical relation with equations on neutrals

$$\frac{\Gamma \vdash f \equiv \mathrm{id}_A : A \to A \qquad \Gamma \vdash l \equiv l' : \mathrm{List}\, A \qquad l, l' \text{ neutrals}}{\Gamma \vdash \texttt{map}\ f\ l \equiv l' : \mathrm{List}\, A}$$

- Mechanized: Consistency, Canonicity, Decidability of conversion and type-checking

Functor laws for lists, Mechanized !

- Extend MLTT with functor laws on List
- Add reduction rules for map composition

$$\texttt{map}\ f\ (\texttt{map}\ g\ l) \rightsquigarrow \texttt{map}\ (f \circ g)\ l \quad \text{for neutral } l$$

- Extend the logical relation with equations on neutrals

$$\frac{\Gamma \vdash f \equiv \mathrm{id}_A : A \to A \qquad \Gamma \vdash l \equiv l' : \mathrm{List}\, A \qquad l, l' \text{ neutrals}}{\Gamma \vdash \texttt{map}\ f\ l \equiv l' : \mathrm{List}\, A}$$

- Mechanized: Consistency, Canonicity, Decidability of conversion and type-checking

A Functorial Type Theory $\mathrm{MLTT}_{\mathrm{map}}$

$$\mathrm{MLTT} + \texttt{map} \text{ for } \mathrm{List}, \Pi, \Sigma, \texttt{W}, \texttt{Id} + \text{functor laws}$$

Application: Structural Coercive and Subsumptive subtyping

$$\textsc{Sub}\quad \frac{\Gamma \vdash_{\mathrm{sub}} t : A \qquad \Gamma \vdash_{\mathrm{sub}} A \preccurlyeq}{\Gamma \vdash_{\mathrm{sub}} t : A'}$$

$$\textsc{Coe}\quad \frac{\Gamma \vdash_{\mathrm{coe}} t : A \qquad \Gamma \vdash_{\mathrm{coe}} A \preccurlyeq A'}{\Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{A,A'}\ t : A'}$$

Application: Structural Coercive and Subsumptive subtyping

$$\frac{\Gamma \vdash_{\mathrm{sub}} t : A \qquad \Gamma \vdash_{\mathrm{sub}} A \preccurlyeq}{\Gamma \vdash_{\mathrm{sub}} t : A'}\ \textsc{Sub} \qquad\qquad \frac{\Gamma \vdash_{\mathrm{coe}} t : A \qquad \Gamma \vdash_{\mathrm{coe}} A \preccurlyeq A'}{\Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{A,A'}\ t : A'}\ \textsc{Coe}$$

Structural coercions:

$$\mathrm{coe}_{\mathrm{List}\ A, \mathrm{List}\ B}\ l \rightsquigarrow \mathtt{map}\ \mathrm{coe}_{A,B}\ l$$

Application: Structural Coercive and Subsumptive subtyping

$$\textsc{Sub}\ \frac{\Gamma \vdash_{\mathrm{sub}} t : A \qquad \Gamma \vdash_{\mathrm{sub}} A \preccurlyeq}{\Gamma \vdash_{\mathrm{sub}} t : A'} \qquad \textsc{Coe}\ \frac{\Gamma \vdash_{\mathrm{coe}} t : A \qquad \Gamma \vdash_{\mathrm{coe}} A \preccurlyeq A'}{\Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{A,A'}\ t : A'}$$

Structural coercions:

$$\mathrm{coe}_{\mathrm{List}\ A, \mathrm{List}\ B}\ l \rightsquigarrow \mathtt{map}\ \mathrm{coe}_{A,B}\ l$$

Equations validated by $\mathrm{MLTT}_{\mathrm{sub}}$:

$$\Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{A,A}\ t \equiv t : A \qquad \Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{B,C}\ (\mathrm{coe}_{A,B}\ t) \equiv \mathrm{coe}_{A,C}\ t : C$$

Application: Structural Coercive and Subsumptive subtyping

$$\textsc{Sub}\ \frac{\Gamma \vdash_{\mathrm{sub}} t : A \qquad \Gamma \vdash_{\mathrm{sub}} A \preccurlyeq}{\Gamma \vdash_{\mathrm{sub}} t : A'} \qquad\qquad \textsc{Coe}\ \frac{\Gamma \vdash_{\mathrm{coe}} t : A \qquad \Gamma \vdash_{\mathrm{coe}} A \preccurlyeq A'}{\Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{A,A'}\ t : A'}$$

Structural coercions:

$$\mathrm{coe}_{\mathrm{List}\ A, \mathrm{List}\ B}\ l \rightsquigarrow \texttt{map}\ \mathrm{coe}_{A,B}\ l$$

Equations validated by $\mathsf{MLTT}_{\mathrm{sub}}$:

$$\Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{A,A}\ t \equiv t : A \qquad\qquad \Gamma \vdash_{\mathrm{coe}} \mathrm{coe}_{B,C}\ (\mathrm{coe}_{A,B}\ t) \equiv \mathrm{coe}_{A,C}\ t : C$$

$$\begin{tikzcd}
 & \mathsf{MLTT} \arrow[dl, hook] \arrow[d, hook] \arrow[dr, hook] & \\
\mathsf{MLTT}_{\mathrm{sub}} & \mathsf{MLTT}_{\mathrm{coe}} \arrow[l, two heads, "|\cdot|", "\sim"'] \arrow[r, hook, "[\![\cdot]\!]"] & \mathsf{MLTT}_{\mathrm{map}}
\end{tikzcd}$$

Current limitations and future steps

Logrel-Coq

- Currently, only one universe
- A branch with List and functor laws
- Missing some (co)inductives: $\mathbb{W}$, $\mathbb{Id}$, $\mathbb{M}$ wanted !

  Add a scheme for cumulative indexed-inductives ?
- Some performance issues to tackle.

Current limitations and future steps

Logrel-Coq

- Currently, only one universe
- A branch with List and functor laws
- Missing some (co)inductives: $\mathbb{W}$, $\mathbb{Id}$, $\mathbb{M}$ wanted !
  Add a scheme for cumulative indexed-inductives ?
- Some performance issues to tackle.

A playground for experimentations on type theories and their normalization

Validate Coq's guard condition, add extensionality principle for booleans, ...

Models from the literature

Reflexive graphs model: external parametricity [Atkey et al.]

Types equipped with a reflexive relation

Setoid model: `UIP`, `funext` [Altenkirch et al.]

Types equipped with an irrelevant equivalence relation

Exceptional model: Exceptions [Pédrot et al.]

Pointed types

Reader model: Reading and setting a global cell [Boulier et al.]

Presheaves on a set of states

Formalizing Logical Relations for MLTT

A logical relation for iterated whnf

A (proof-relevant) predicate

$$\Gamma \Vdash A$$

characterizing types by their
weak head normal form.

A logical relation for iterated whnf

A (proof-relevant) predicate

$$\Gamma \Vdash A$$

characterizing types by their **weak head normal form**.

For $[A] : \Gamma \Vdash A$, 3 predicates:

$$\Gamma \Vdash_{[A]} A \cong B$$

$$\Gamma \Vdash_{[A]} t : A$$

$$\Gamma \Vdash_{[A]} t \cong u : A$$

A logical relation for iterated whnf

A (proof-relevant) predicate

$$\Gamma \Vdash A$$

characterizing types by their **weak head normal form**.

For $[A] : \Gamma \Vdash A$, 3 predicates:

$$\Gamma \Vdash_{[A]} A \cong B$$

$$\Gamma \Vdash_{[A]} t : A$$

$$\Gamma \Vdash_{[A]} t \cong u : A$$

Using **small-induction recursion** [Hancock et al.] in Coq.

```
Inductive LR@{i j k} {l : TypeLevel} (rec : forall l', l' << l -> RedRel@{i j})
: RedRel@{j k} :=
  | LRU {Γ A} (H : [Γ ||-U<l> A]) :
      LR rec Γ A
      (fun B   => [Γ ||-U≅ B ])
      (fun t   => [ rec | Γ ||-U t     : A | H ])
      (fun t u => [ rec | Γ ||-U t ≅ u : A | H ])
  | LRne {Γ A} (neA : [ Γ ||-ne A ]) :
      LR rec Γ A
      (fun B   =>  [ Γ ||-ne A ≅ B     | neA])
      (fun t   =>  [ Γ ||-ne t     : A | neA])
      (fun t u =>  [ Γ ||-ne t ≅ u : A | neA])
  | LRPi {Γ : context} {A : term} (ΠA : PiRedTy@{j} Γ A) (ΠAad : PiRedTyAdequat
    LR rec Γ A
      (fun B   => [ Γ ||-Π A ≅ B     | ΠA ])
      (fun t   => [ Γ ||-Π t     : A | ΠA ])
      (fun t u => [ Γ ||-Π t ≅ u : A | ΠA ])
  | LRNat {Γ A} (NA : [Γ ||-Nat A]) :
    LR rec Γ A (NatRedTyEq NA) (NatRedTm NA) (NatRedTmEq NA)
  | LREmpty {Γ A} (NA : [Γ ||-Empty A]) :
    LR rec Γ A (EmptyRedTyEq NA) (EmptyRedTm NA) (EmptyRedTmEq NA)
  | LRSig {Γ : context} {A : term} (ΣA : SigRedTy@{j} Γ A) (ΣAad : SigRedTyAdeq
    LR rec Γ A (SigRedTyEq ΣA) (SigRedTm ΣA) (SigRedTmEq ΣA)
  | LRList {Γ : context} {A : term}
      (LA : ListRedTyPack@{j} Γ A) (LAAd : ListRedTyAdequate@{j k} (LR rec) LA)
    LR rec Γ A
      (ListRedTyEq@{j} Γ A LA)
      (ListRedTm@{j} Γ A LA)
      (ListRedTmEq@{j} Γ A LA).
```

Properties of the logical relation

- Escape: if $\Gamma \Vdash_{[A]} t : A$ then $\Gamma \vdash t : A$
- Irrelevance (including universe level)
- Equivalence: reflexivity, symmetry, transitivity
- Neutral reflection
- Closure by anti-reduction

Properties of the logical relation

- Escape: if $\Gamma \Vdash_{[A]} t : A$ then $\Gamma \vdash t : A$
- Irrelevance (including universe level)
- Equivalence: reflexivity, symmetry, transitivity
- Neutral reflection
- Closure by anti-reduction

Fundamental lemma: if $\Gamma \vdash_{\mathsf{de}} t : A$ then $[A] : \Gamma \Vdash A$ and $\Gamma \Vdash_{[A]} t : A$

Properties of the logical relation

- Escape: if $\Gamma \Vdash_{[A]} t : A$ then $\Gamma \vdash t : A$
- Irrelevance (including universe level)
- Equivalence: reflexivity, symmetry, transitivity
- Neutral reflection
- Closure by anti-reduction

Fundamental lemma: if $\Gamma \vdash_{\mathsf{de}} t : A$ then $[A] : \Gamma \Vdash A$ and $\Gamma \Vdash_{[A]} t : A$

Corollary: if $\Gamma \vdash_{\mathsf{de}} t : A$ then $\Gamma \vdash t : A$

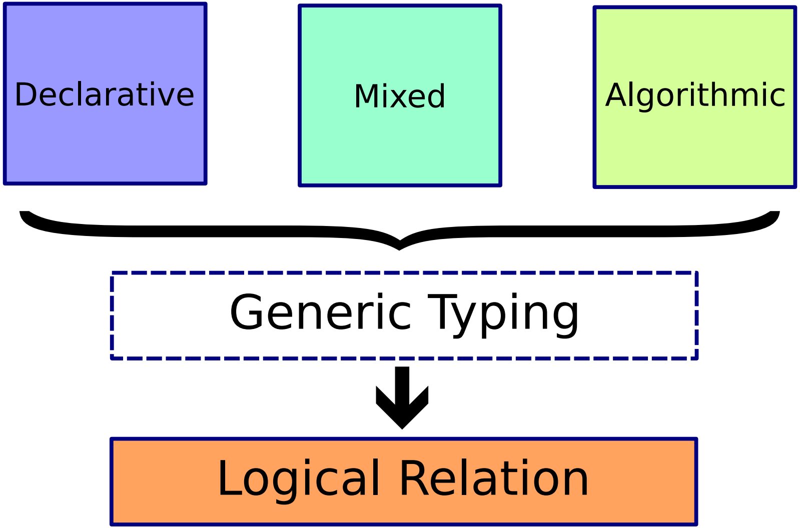
Properties of the logical relation

- Escape: if $\Gamma \Vdash_{[A]} t : A$ then $\Gamma \vdash_{\mathrm{gen}} t : A$
- Irrelevance (including universe level)
- Equivalence: reflexivity, symmetry, transitivity
- Neutral reflection
- Closure by anti-reduction

Fundamental lemma: if $\Gamma \vdash_{\mathsf{de}} t : A$ then $[A] : \Gamma \Vdash A$ and $\Gamma \Vdash_{[A]} t : A$

Corollary: if $\Gamma \vdash_{\mathsf{de}} t : A$ then $\Gamma \vdash_{\mathrm{gen}} t : A$

3 logical relations in 1



Engineering aspects

Code: 20k loc (9k spec; 11k proofs)

The formalization rely on

- `autosubst2` for generating renaming, substitution and their lemmas
- `Equations`
- `partialfun` (T. Winterhalter) for defining the typechecking algorithm and reasoning on it

Tactics:

- for discharging typing goals (`eauto` with typing lemmas)
- in order to dispatch the many forms of irrelevance
- for instantiating the logical relation with valid substitutions