DEFINITIONAL Functoriality for DEPENDENT (Sub)Types

Théo Laurent, Meven Lennon-Bertrand, Kenji Maillard

π³ Seminar — December 11th 2023
DEPENDENT TYPES AND SUBTYPING
Flavours of subtyping – subsumptive

What users™ want:

\[
\frac{\Gamma \vdash_{\text{sub}} t : A \quad \Gamma \vdash_{\text{sub}} A \preceq A'}{\Gamma \vdash_{\text{sub}} t : A'}
\]

Semanticists hate this: forces \( q \Gamma \vdash \text{sub} A \preceq A' \) to be (set) inclusion.

\( J_t K = J t K \in J A K q A' \)

Too strict:

\( A' \preceq A \preceq B \preceq B' \) is not set-theoretic inclusion…
Flavours of subtyping – subsumptive

What users™ want:

\[
\frac{\Gamma \vdash_{\text{sub}} t : A \quad \Gamma \vdash_{\text{sub}} A \preceq A'}{\Gamma \vdash_{\text{sub}} t : A'}
\]

Semanticists hate this: forces \([\Gamma \vdash_{\text{sub}} A \preceq A']\) to be (set) inclusion.

\[
[t] = [t] \\
\cap \quad \cap \\
[A] \quad [A']
\]
What users™ want:

\[
\begin{align*}
\text{SUB} & \quad \frac{\Gamma \vdash_{\text{sub}} t : A \quad \Gamma \vdash_{\text{sub}} A \preceq A'}{\Gamma \vdash_{\text{sub}} t : A'}
\end{align*}
\]

Semanticists hate this: forces \( \left[ \Gamma \vdash_{\text{sub}} A \preceq A' \right] \) to be (set) inclusion.

\[
[t] = [t] \\
\cap \quad \cap \\
[A] \quad [A']
\]

Too strict:

\[
\begin{align*}
A' & \preceq A \\
B' & \preceq B \\
A \to B & \preceq A' \to B'
\end{align*}
\]

is not set-theoretic inclusion...
What semanticists want you to do:

\[
\frac{
\Gamma \vdash \text{coe} \, t : A \\
\Gamma \vdash \text{coe} \, A < A'
}{
\Gamma \vdash \text{coe}_{A,A'} \, t : A'
}
\]

Nicer semantics:
\[r\text{coe}_{A,A'} = \text{"well-chosen functions"}\]

Set-theoretic interpretation for \(r\text{coe}_{A,A'}\):
\[\text{Set-theoretic interpretation for } r\text{coe}_{A,A'}\]

This is the work of a compiler!
What semanticists want you to do:

\[
\frac{\Gamma \vdash \text{coe } t : A \quad \Gamma \vdash \text{coe } A \preceq A'}{\Gamma \vdash \text{coe } \text{coe}_{A,A'} t : A'}
\]

Nicer semantics: \(\left[\text{coe}_{A,A'}\right] = \text{"well-chosen functions"}\)

Set-theoretic interpretation for \(\left[\text{coe}_{A \to B, A' \to B'}\right]\) is the one we want
What semanticists want you to do:

\[ \frac{\Gamma \vdash \text{coe } t : A \quad \Gamma \vdash \text{coe } A \preccurlyeq A'}{\Gamma \vdash \text{coe} \_ \_ A, A' \_ t : A'} \]

Nicer semantics: \( [\text{coe}_{A, A'}] \) = “well-chosen functions”

Set-theoretic interpretation for \( [\text{coe}_{A \rightarrow B, A' \rightarrow B'}] \) is the one we want

This is the work of a compiler!
Let’s just elaborate, then:

\[ \Gamma \vdash_{\text{sub}} t : A \quad \Gamma \vdash_{\text{sub}} A \preceq A' \]

\[ \Gamma \vdash_{\text{sub}} t : A' \]

\[ \sim \sim \]

\[ \tilde{\Gamma} \vdash_{\text{coe}} \tilde{t} : \tilde{A} \quad \tilde{\Gamma} \vdash_{\text{coe}} \tilde{A} \preceq \tilde{A}' \]

\[ \tilde{\Gamma} \vdash_{\text{coe}} \coe_{\tilde{A}, \tilde{A}'} \tilde{t} : \tilde{A}' \]
Let’s just elaborate, then:

\[
\begin{align*}
\Gamma \vdash_{\text{sub}} t : A & \quad \Gamma \vdash_{\text{sub}} A \preceq A' \\
\Rightarrow & \\
\Gamma \vdash_{\text{sub}} t : A' 
\end{align*}
\; \quad \Rightarrow 
\begin{align*}
\tilde{\Gamma} \vdash_{\text{coe}} \tilde{t} : \tilde{A} & \quad \tilde{\Gamma} \vdash_{\text{coe}} \tilde{A} \preceq \tilde{A}' \\
\Rightarrow & \\
\tilde{\Gamma} \vdash_{\text{coe}} \text{coe}_{\tilde{A}, \tilde{A}'} \tilde{t} : \tilde{A}'
\end{align*}
\]

Coherence: \([t] \overset{\text{def}}{=} [\tilde{t}]\) should be unambiguous – all elaborations should have the same semantics.
Let’s just elaborate, then:

\[
\begin{align*}
\Gamma \vdash_{\text{sub}} t : A & \quad \Gamma \vdash_{\text{sub}} A \preceq A' \\
\Gamma \vdash_{\text{sub}} t : A' & \quad \Gamma \vdash_{\text{sub}} A \preceq A' \\
\Gamma \vdash_{\text{coe}} \tilde{t} : \tilde{A} & \quad \Gamma \vdash_{\text{coe}} \tilde{A} \preceq \tilde{A}' \\
\Gamma \vdash_{\text{coe}} \text{coe}_{\tilde{A}, \tilde{A}'} \tilde{t} : \tilde{A}' & \quad \Gamma \vdash_{\text{coe}} \text{coe}_{\tilde{A}, \tilde{A}'} \tilde{t} : \tilde{A}'
\end{align*}
\]

Coherence: \([t] \overset{\text{def}}{=} [\tilde{t}]\) should be unambiguous – all elaborations should have the same semantics.

Actually...

Cannot unify \(t\) and \(\tilde{t}\).
Let’s just elaborate, then:

\[
\Gamma \vdash_{\text{sub}} t : A \\
\Gamma \vdash_{\text{sub}} A \preceq A' \\
\Gamma \vdash_{\text{sub}} t : A'
\]

\[
\tilde{\Gamma} \vdash_{\text{coe}} \tilde{t} : \tilde{A} \\
\tilde{\Gamma} \vdash_{\text{coe}} \tilde{A} \preceq \tilde{A}' \\
\tilde{\Gamma} \vdash_{\text{coe}} \text{coe}_{\tilde{A}, \tilde{A}'} \tilde{t} : \tilde{A}'
\]

Coherence: \([t] \overset{\text{def}}{=} [\tilde{t}]\) should be unambiguous – all elaborations should have the same semantics.

Actually...

Cannot unify \(t\) and \(\tilde{t}\).

Better coherence: we should always have \(\tilde{t} \cong \tilde{t}'\) for two different elaborations of \(t\).
Now we have a goal

\[ |\cdot| \text{ (removes coercions)} \]

\[ \text{users} \leftarrow \text{MLTT}_{\text{sub}} \rightarrow \text{MLTT}_{\text{coe}} \rightarrow \text{models} \]

Provided \( \Gamma \vdash \text{coe} \):
Now we have a goal

Now we have a goal

\[ |\cdot| \]

(removes coercions)

\[ \text{users} \leftarrow \text{MLTT}_{\text{sub}} \rightarrow \text{models} \]

\[ \text{MLTT}_{\text{coe}} \]

? = the compilation we want

Only well-defined if a lot of equations hold

In particular, \(|t| = |u| \Rightarrow \Gamma \vdash_{\text{coe}} t \equiv u : A\) (provided \(\Gamma \vdash_{\text{coe}} t, u : A\))
What is a reasonable computational behaviour for coercions?
What is a reasonable computational behaviour for coercions?

\[
\text{coe}_{\text{List } A, \text{List } A'}[\text{}] \cong [\text{}]
\]

\[
\text{coe}_{\text{List } A, \text{List } A'}(a :: l) \cong (\text{coe}_{A, A'} a) :: (\text{coe}_{\text{List } A, \text{List } A'} l)
\]

\[
(\text{coe}_{A \rightarrow B, A' \rightarrow B'} f) u \cong \text{coe}_{B, B'}(f (\text{coe}_{A', A} u))
\]

⋮
What is a reasonable computational behaviour for coercions?

\[ \text{coe}_{\text{List } A, \text{List } A'}([]) \cong [] \]

\[ \text{coe}_{\text{List } A, \text{List } A'}(a :: l) \cong (\text{coe}_{A, A'} a :: (\text{coe}_{\text{List } A, \text{List } A'} l)) \]

\[ (\text{coe}_{A \rightarrow B, A' \rightarrow B'} f) u \cong \text{coe}_{B, B'}(f (\text{coe}_{A', A} u)) \]

\[ \vdots \]

These are map operations! We better understand them before going further.
DEFINITIONAL FUNCTORIALITY
DEFINITIONAL FUNCTOR LAWS

MLTT

\( \text{map} \)

Each type former \( F \) comes with \( \text{dom}(F) \), \( \text{hom}(F) \) and \( \text{map}_F \) such that

\[
\text{MAP} \quad \Gamma \vdash \text{map}_X, Y : \text{dom}(F) \quad \Gamma \vdash \text{map}_f : \text{hom}_F(X, Y) \quad \Gamma \vdash \text{map}_f \text{map}_F f : F X \rightarrow F Y
\]

\[
\text{MAPID} \quad \Gamma \vdash \text{map}_X : \text{dom}(F) \quad \Gamma \vdash \text{map}_t : F X \quad \Gamma \vdash \text{map}_t \text{map}_F \text{id}_F X t \cong t : F X
\]

\[
\text{MAPCOMP} \quad \Gamma \vdash \text{map}_X, Y, Z : \text{dom}(F) \quad \Gamma \vdash \text{map}_g : \text{hom}_F(X, Y) \quad \Gamma \vdash \text{map}_f : \text{hom}_F(Y, Z) \quad \Gamma \vdash \text{map}_t : F X \quad \Gamma \vdash \text{map}_t \text{map}_F f (\text{map}_F g t) \cong \text{map}_F (f \circ \text{id}_F g) t : F Z
\]

+ congruences, specific laws for each \( F \)

\[
\Gamma \vdash \text{map}(f, g) : \text{hom}_{\Pi}((A, B), (A', B')) \quad \Gamma \vdash \text{map}_h : \Pi x : A. B \quad \Gamma \vdash \text{map}_a : A' \quad \Gamma \vdash \text{map}\Pi (f, g) h a' \cong g(h(f a')) : B'
\]
MLTT map each type former $F$ comes with $\text{dom}(F)$, $\text{hom}(F)$ and $\text{map}_F$ such that
MLTT\text{map} each type former \( F \) comes with \( \text{dom}(F) \), \( \text{hom}(F) \) and \( \text{map}_F \) such that

\[
\begin{align*}
\Gamma \vdash_{\text{map}} X, Y : \text{dom}(F) \\
\Gamma \vdash_{\text{map}} f : \text{hom}_F(X,Y) \\
\Gamma \vdash_{\text{map}} \text{map}_F f : F X \to FY
\end{align*}
\]
DEFINITIONAL FUNCTOR LAWS

\[ \text{map} \] each type former \( F \) comes with \( \text{dom}(F) \), \( \text{hom}(F) \) and \( \text{map}_F \) such that

\[
\begin{align*}
\Gamma \vdash_{\text{map}} X, Y : \text{dom}(F) \\
\Gamma \vdash_{\text{map}} f : \text{hom}_F(X, Y) \\
\Gamma \vdash_{\text{map}} \text{map}_F f : F X \to F Y \\
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash_{\text{map}} X : \text{dom}(F) \\
\Gamma \vdash_{\text{map}} t : F X \\
\Gamma \vdash_{\text{map}} \text{map}_F \text{id}_X^F \ t \equiv t : F X \\
\end{align*}
\]
**DEFINITIONAL FUNCTOR LAWS**

Each type former $F$ comes with $\text{dom}(F)$, $\text{hom}(F)$ and $\text{map}_F$ such that:

**MAP**

$\Gamma \vdash_{\text{map}} X, Y : \text{dom}(F)$

$\Gamma \vdash_{\text{map}} f : \text{hom}_F(X, Y)$

$\Gamma \vdash_{\text{map}} \text{map}_F f : FX \to FY$

**MAPID**

$\Gamma \vdash_{\text{map}} X : \text{dom}(F)$

$\Gamma \vdash_{\text{map}} t : FX$

$\Gamma \vdash_{\text{map}} \text{map}_F \text{id}_X t \equiv t : FX$

**MAPCOMP**

$\Gamma \vdash_{\text{map}} X, Y, Z : \text{dom}(F)$

$\Gamma \vdash_{\text{map}} g : \text{hom}_F(X, Y)$

$\Gamma \vdash_{\text{map}} f : \text{hom}_F(Y, Z)$

$\Gamma \vdash_{\text{map}} t : FX$

$\Gamma \vdash_{\text{map}} \text{map}_F f (\text{map}_F g t) \equiv \text{map}_F(f \circ^F g)t : FZ$
Definitional Functor Laws

Each type former $F$ comes with $\text{dom}(F)$, $\text{hom}(F)$ and $\text{map}_F$ such that

\[
\begin{align*}
\Gamma \vdash_{\text{map}} X, Y : \text{dom}(F) \\
\Gamma \vdash_{\text{map}} f : \text{hom}_F(X, Y)
\end{align*}
\]

\[
\rightarrow \quad \Gamma \vdash_{\text{map}} \text{map}_F f : F X \to FY
\]

\[
\begin{align*}
\Gamma \vdash_{\text{map}} X : \text{dom}(F) \\
\Gamma \vdash_{\text{map}} t : F X
\end{align*}
\]

\[
\rightarrow \quad \Gamma \vdash_{\text{map}} \text{map}_F \text{id}_X t \equiv t : F X
\]

\[
\begin{align*}
\Gamma \vdash_{\text{map}} X, Y, Z : \text{dom}(F) \\
\Gamma \vdash_{\text{map}} g : \text{hom}_F(X, Y) \\
\Gamma \vdash_{\text{map}} f : \text{hom}_F(Y, Z) \\
\Gamma \vdash_{\text{map}} t : F X
\end{align*}
\]

\[
\rightarrow \quad \Gamma \vdash_{\text{map}} \text{map}_F f (\text{map}_F g t) \equiv \text{map}_F(f \circ_F g) t : F Z
\]

+ congruences, specific laws for each $F$

\[
\begin{align*}
\Gamma \vdash_{\text{map}} (f, g) : \text{hom}_\Pi((A, B), (A', B')) \\
\Gamma \vdash_{\text{map}} h : \Pi x : A.B \\
\Gamma \vdash_{\text{map}} a' : A'
\end{align*}
\]

\[
\rightarrow \quad \Gamma \vdash_{\text{map}} \text{map}_\Pi (f, g) h a' \equiv g (h (f a')) : B'[a']
\]
NEW EQUATIONS FOR NEUTRALS

This is *not* vanilla MLTT, where

\[
\text{map}_{\text{List}} f \left( \text{map}_{\text{List}} g x \right) \not\cong \text{map}_{\text{List}} (f \circ g) x
\]
This is *not* vanilla MLTT, where

$$\text{map}_{\text{List}} f \ (\text{map}_{\text{List}} g \ x) \not\cong \text{map}_{\text{List}} (f \circ g) \ x$$

In general, problems with *neutrals* at *positive* types (= without $\eta$)
New Equations for Neutrals

This is not vanilla MLTT, where

$$\text{map}_{\text{List}} f (\text{map}_{\text{List}} g x) \not\equiv \text{map}_{\text{List}} (f \circ g) x$$

In general, problems with neutrals at positive types (= without η)

Can we add these equations in?
**New Equations for Neutral Terms**

A Sound and Complete Decision Procedure, Formalized

Guillaume Allais  
University of Strathclyde  
{guillaume.allais, conor.mcbride}@strath.ac.uk

Pierre Boutillier  
PPS - Paris Diderot  
pierre.boutillier@pps.univ-paris-diderot.fr

**Abstract**

The definitional equality of an intensional type theory is its test of type compatibility. Today’s systems rely on ordinary evaluation semantics to compare expressions in types, frustrating users with type errors arising when evaluation fails to identify two ‘obviously’ equal terms. If only the machine could decide a richer theory! We propose a way to decide theories which supplement evaluation with ’η-rules‘, rearranging the neutral parts of normal forms, and report a successful initial experiment.

We study a simple λ-calculus with primitive fold, map and append operations on lists and develop in Agda a sound and complete decision procedure for an equational theory enriched with monoid, functor and fusion laws.

**Keywords**  
Normalization by Evaluation, Logical Relations, Simply-Typed Lambda Calculus, Map Fusion

1. Introduction

The programmer working in intensional type theory is no stranger to ‘obviously true’ equations she wishes held definitively for her program to typecheck without having to chase down ill-typed terms and brutally coerce them. In this article, we present one way to relax definitional equality, thus accommodating some of her longings. We distinguish three types of fundamental relations between terms.

The first denotes computational rules: it is untyped, oriented and denoted by \(\rightsquigarrow\) in its one step version or \(\Rightarrow\) when the reflexive transitive congruence closure is considered. In Table 1, we introduce a few such rules which correspond to the equations the programmer writes to define functions. They are referred to as \(\delta\) (for definitions) and \(\iota\) (for pattern-matching on inductive data) rules and hold computationally just like the more common \(\beta\)-rule.

The second is the judgmental equality \((\equiv)\): it is typed, tractable

\[
\begin{align*}
\text{map} : (a \rightarrow b) &\rightarrow \text{list} a \rightarrow \text{list} b \\
\text{map} f \text{ [] } &\equiv \text{ [] } \\
\text{map} f (x :: xs) &\equiv f x :: \text{map} f xs \\
(\text{++}) : \text{list} a \rightarrow \text{list} a \rightarrow \text{list} a \\
\text{[] ++ ys} &\equiv ys \\
x :: xs ++ ys &\equiv x :: (xs ++ ys) \\
\text{fold} : (a \rightarrow b \rightarrow b) \rightarrow b \rightarrow \text{list} a \rightarrow b \\
\text{fold} c \text{ n } &\equiv n \\
\text{fold} c \text{ n } (x :: xs) &\equiv c\times (\text{fold} c n xs)
\end{align*}
\]

**Table 1.** \(\delta\)-rules - computational

fied judgmentally. Table 2 shows a kit for building computationally inert neutral terms growing layers of thwarted progress around a variable which we dub the ‘\(\eta\)’, together with some equations on neutral terms which held only propositionally – until now. This paper shows how to extend the judgmental equality with these new \(\eta\)-rules. We gain, for example, that \(\text{map swap} \equiv \text{map swap} \equiv \text{id}\), where \(\text{swap}\) swaps the elements of a pair.

\[
\begin{align*}
\text{swap} : \text{list} a \rightarrow \text{list} b \\
f_1 n f_2 &\equiv \text{map} f f_1 n \\
\text{map} f &\equiv \text{fold} n c \\
\end{align*}
\]

**Table 2.** \(\eta\)-rules - canonicity

\(\equiv\) (= without \(\eta\)”
New Equations for Neutral Terms
A Sound and Complete Decision Procedure, Formalized

Guillaume Allais Conor McBride
University of Strathclyde
{guillaume.allais, conor.mcbride}@strath.ac.uk

Pierre Bouillier
PPS - Paris Diderot
pierre.bouillier@pps.univ-paris-diderot.fr

Abstract
The definitional equality of an intensional type of type compatibility. Today's systems rely on semantics to compare expressions in types, and type errors arising when evaluation fails to identify equal terms. If only the machine could decide a way to decide theories which supply 'η-rules', rearranging the neutral parts of nor a successful initial experiment.

We study a simple λ-calculus with primitive l-pend operations on lists and develop in Agda a decision procedure for an equational theory of functor and fusion laws.

Keywords: Normalization by Evaluation, Log Typed Lambda Calculus, Map Fusion

1. Introduction
The programmer working in intensional type to 'obviously true' equations she wishes held program to typecheck without having to chuse and brutally coerce them. In this article, we present definitional equality, thus accommodating it. We distinguish three types of fundamental relations.

The first denotes computational equalities: it is denoted by \( \equiv \) in its one step version or \( \equiv^\ast \) its \( \ast \) step congruence closure is considered. In The few such rules which correspond to the equat writes to define functions. They are referred to and \( \equiv \) (for pattern-matching on inductive data) putatively just like the more common \( \lambda \)-rules.

The second is the judgmental equality (=)

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 \( \eta \)-equality for \( \Pi \) 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 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;
MARTIN-LÖF À LA COQ

JWW. A. ADJEDJ, K. MAILLARD, P-M. PÉDROT and L. PUJET
Martin-Löf logical framework

\[ \vdash \Gamma \quad \Gamma \vdash A \quad \Gamma \vdash A \equiv B \quad \Gamma \vdash t : A \quad \Gamma \vdash t \equiv u : A \]
Martin-Löf logical framework

\[ ⊢ \Gamma \quad Γ ⊢ A \quad Γ ⊢ A ≅ B \quad Γ ⊢ t : A \quad Γ ⊢ t ≅ u : A \]

\[
\begin{align*}
\text{REFL} & : & Γ ⊢ t : A & \quad & Γ ⊢ t ≅ t : A \\
\text{SYM} & : & Γ ⊢ t ≅ u : A & \quad & Γ ⊢ u ≅ t : A \\
\text{TRANS} & : & Γ ⊢ t ≅ u : A & \quad & Γ ⊢ u ≅ v : A & \quad & Γ ⊢ t ≅ v : A
\end{align*}
\]
Martin-Löf logical framework

\[ \Gamma \vdash \Gamma \vdash A \quad \Gamma \vdash A \equiv B \quad \Gamma \vdash t : A \quad \Gamma \vdash t \equiv u : A \]

+ type formers (Type, Π, Σ, \( x =_{A} y \), W ...)

\[
\begin{align*}
\text{APPCONG} & : \quad \frac{\Gamma \vdash t \equiv t' : \Pi x : A. B \quad \Gamma \vdash u \equiv u' : A}{\Gamma \vdash t \ u \equiv t' \ u' : B[u]}
\end{align*}
\]

\[
\begin{align*}
\text{βFUN} & : \quad \frac{\Gamma \vdash A \quad \Gamma, x : A \vdash B}{\Gamma, x : A \vdash t : B \quad \Gamma \vdash u : A}{\Gamma \vdash (\lambda x : A. t) \ u \equiv t[u] : B[u]}
\end{align*}
\]

\[
\begin{align*}
\text{ηFUN} & : \quad \frac{\Gamma \vdash f : \Pi x : A. B}{\Gamma \vdash f \equiv \lambda x : A. f \ x : \Pi x : A. B}
\end{align*}
\]
Martin-Löf logical framework

\[
\begin{align*}
\Gamma & \vdash A & \Gamma & \vdash A \equiv B & \Gamma & \vdash t : A & \Gamma & \vdash u : A \\
\Gamma & \vdash t \equiv u : A & \Gamma & \vdash u \equiv v : A & \Gamma & \vdash t \equiv v : A \\
\Gamma & \vdash A \equiv B & \Gamma & \vdash t : B \\
\end{align*}
\]

+ type formers \((\text{Type}, \Pi, \Sigma, x =_A y, W \ldots)\)

Derivations are not unique!
Declarative and algorithmic presentations

Declarative typing

Freestanding conversion rule

\[ \Gamma \vdash^{\text{de}} t : A \quad \Gamma \vdash^{\text{de}} A \equiv B \]

\[ \Gamma \vdash^{\text{de}} t : B \]

Algorithmic typing (bidirectional)

Mode-constrained conversion

\[ \Gamma \vdash^{\text{al}} t \triangleright A \quad \Gamma \vdash^{\text{al}} A \equiv B \]

\[ \Gamma \vdash^{\text{al}} t \triangleleft B \]
Declarative and Algorithmic Presentations

### Declarative Typing

**Freestanding** conversion rule

\[
\Gamma \vdash \text{de} \ t : A \quad \Gamma \vdash \text{de} \ A \equiv B \\
\Gamma \vdash \text{de} \ t : B
\]

Conversion mixes *arbitrarily*:
- Computation steps ($\beta$),
- Extensionality steps ($\eta$),
- Congruences,
- Transitivity, symmetry and reflexivity.

### Algorithmic Typing (Bidirectional)

**Mode-constrained conversion**

\[
\Gamma \vdash \text{al} \ t \triangleright A \quad \Gamma \vdash \text{al} \ A \equiv B \\
\Gamma \vdash \text{al} \ t \triangleleft B
\]

**Term/type-directed** conversion alternating:
- Reduction to weak-head normal form,
- Type-directed extensionality rules,
- Congruences.
How can we compare the two presentations of MLTT?
How can we compare the two presentations of MLTT?

Algorithmic $\rightarrow$ Declarative: Admissibility of algorithmic rules
How can we compare the two presentations of MLTT?

Algorithmic → Declarative: Admissibility of algorithmic rules
Declarative → Algorithmic: Need to show that every derivation has a canonical form
How can we compare the two presentations of MLTT?

**Algorithmic → Declarative:** Admissibility of algorithmic rules

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

Gives decidability of conversion/typing... and coherence! (More on this later)
A (proof-relevant) predicate $\Gamma \vdash A$ characterizing types by their weak head normal form.
A (proof-relevant) predicate $\Gamma \vdash A$ characterizing types by their weak head normal form.

For $A :: \Gamma \vdash A$, 3 predicates:

$$\Gamma \vdash A \\ \Gamma \vdash A \equiv B \\ \Gamma \vdash A t : A \\ \Gamma \vdash A t \equiv u : A$$
A (proof-relevant) predicate $\Gamma \vdash A$ characterizing types by their weak head normal form.

For $\mathcal{A} :: \Gamma \vdash A$, 3 predicates:

- $\Gamma \vdash_{\mathcal{A}} A \cong B$
- $\Gamma \vdash_{\mathcal{A}} t : A$
- $\Gamma \vdash_{\mathcal{A}} t \equiv u : A$

Natural numbers:

- $\Gamma \vdash T \rightsquigarrow^* \mathbb{N}$

+ typing side-conditions
A (proof-relevant) predicate $\Gamma \vdash A$ characterizing types by their weak head normal form.

For $\mathcal{A} :: \Gamma \vdash A$, 3 predicates:

\[
\begin{align*}
\Gamma \vdash_{\mathcal{A}} A \equiv B \\
\Gamma \vdash_{\mathcal{A}} t : A \\
\Gamma \vdash_{\mathcal{A}} t \equiv u : A
\end{align*}
\]

Natural numbers:

\[
\Gamma \vdash T \rightsquigarrow^* \mathbb{N} \\
\frac{}{\Gamma \vdash T}
\]

+ typing side-conditions
A (proof-relevant) predicate $\Gamma \vDash A$ characterizing types by their weak head normal form.

For $A :: \Gamma \vDash A$, 3 predicates:

$\Gamma \vdash_A A \equiv B$  $\Gamma \vdash_A t : A$  $\Gamma \vdash_A t \equiv u : A$

Natural numbers:

$\Gamma \vdash t \rightsquigarrow^* 0 : \mathbb{N}$  $\Gamma \vdash t \rightsquigarrow^* S(t') : \mathbb{N}$

$\Gamma \vdash t : T$

$\Gamma \vdash t' : \mathbb{N}$

$\Gamma \vdash t : T$

+ typing side-conditions
A (proof-relevant) predicate $\Gamma \vdash A$ characterizing types by their weak head normal form.

For $A :: \Gamma \vdash A$, 3 predicates:

\[
\begin{align*}
\Gamma \vdash_A A \cong B & \quad \Gamma \vdash_A t : A & \quad \Gamma \vdash_A t \equiv u : A
\end{align*}
\]

Natural numbers:

\[
\begin{align*}
\Gamma \vdash t \rightsquigarrow^* 0 : \mathbb{N} & \quad \Gamma \vdash t' : \mathbb{N} & \quad \Gamma \vdash n \approx n : \mathbb{N}
\end{align*}
\]

+ typing side-conditions
• mutual definition of $\Gamma \vdash A$ and $\Gamma \vdash t : A$
• reducibility at the universe $\Gamma \vdash A : \text{Type}$ is basically $\Gamma \vdash A$...
• mutual definition of $\Gamma \vdash A$ and $\Gamma \vdash t : A$
• reducibility at the universe $\Gamma \vdash A : \text{Type}$ is basically $\Gamma \vdash A$...

(Small) induction-recursion + stratified definitions
Properties of the Logical Relation

- Escape: $\Gamma \vdash A \Rightarrow \Gamma \vdash A$
- Irrelevance
- Equivalence: reflexivity, symmetry, transitivity
- Stability by weakening
- Neutral reflection
- Closure by anti-reduction

Fundamental Lemma:
If $\Gamma \vdash_{de} \Gamma \vdash A$ and $\Gamma \vdash A_{de}$
PROPERTIES OF THE LOGICAL RELATION

• Escape: $\Gamma \Vdash A \Rightarrow \Gamma \vdash A$
• Irrelevance
• Equivalence: reflexivity, symmetry, transitivity
• Stability by weakening
• Neutral reflection
• Closure by anti-reduction

Fundamental lemma: if $\Gamma \vdash^{\text{de}} t : A$ then $A :: \Gamma \Vdash A$ and $\Gamma \vdash_{\mathcal{A}} t : A$
3 LOGICAL RELATIONS IN 1

 declarative
\[ \Gamma \vdash^{\text{de}} A \]

 Soundness

 generic
\[ \Gamma \vdash A \]

 fundamental

 algorithmic
\[ \Gamma \vdash^{\text{al}} A \]

 logical relation
\[ \Gamma \vDash A \]

 escape
ALL FORMALISED!
BACK TO BUSINESS
The whole story extends...
The whole story extends...

To handle new equation for neutrals:

1. map fusion in reduction

\[
\text{neutral } n \\
\text{map} \text{List } f (\text{map} \text{List } g n) \leadsto \text{map} \text{List}(f \circ g)n
\]
The whole story extends...

To handle new equation for neutrals:

1. map fusion in reduction

\[
\text{map}_{\text{List}} f (\text{map}_{\text{List}} g n) \leadsto \text{map}_{\text{List}} (f \circ g) n
\]

2. identity in comparison

\[
\Gamma \vdash_{\text{map}} n \approx n' : \text{List } A \quad \Gamma, x : A \vdash f x \cong x : A
\]

\[
\Gamma \vdash_{\text{map}} \text{List } f n \cong n' : \text{List } A
\]
The whole story extends...

To handle new equation for neutrals:

1. map fusion in reduction
   \[ \text{neutral } n \]
   \[ \text{map}_{\text{List}} f (\text{map}_{\text{List}} g n) \leadsto \text{map}_{\text{List}} (f \circ g) n \]

2. identity in comparison
   \[ \Gamma \vdash \text{map} n \approx n' : \text{List } A \]
   \[ \Gamma, x : A \vdash f x \equiv x : A \]
   \[ \Gamma \vdash \text{map}_{\text{List}} f n \equiv n' : \text{List } A \]

List (\text{&}), W, Id, + ... (\text{&})
The story still extends...
The story still extends...

- extend conversion to subtyping, reducible conversion to reducible subtyping
- $\text{coe}_{A,B} t$ reduces $A, B$, then applies the relevant map
- $\text{coe}_{A,B} \text{coe}_{A',B'} t$ is compacted if $A B A' B'$ are all neutral, or are positive types and $t$ is neutral
- identity in comparison as before
Only the algorithmic system!
Only the algorithmic system!

\[ \| \cdot \| \quad (\text{removes coercions}) \]

users \[\xrightarrow{\text{type preserving elaboration}}\] models

Much easier to show that elaboration preserves \textit{algorithmic} typing. Key lemma: coercions never block redexes.
Only the algorithmic system!

\[ |::| \] (removes coercions)

Much easier to show that elaboration preserves algorithmic typing.
Key lemma: coercions never block redexes.

For “free”: coherence up to conversion.
Wrapping up
• Martin-Löf à la Coq: all meta-theory of MLTT, formalised in Coq (and there’s more I have not told you about).
• Martin-Löf à la CoQ: all meta-theory of MLTT, formalised in CoQ (and there’s more I have not told you about).

• MLTT\textsubscript{map}: normalisation, decidability of type-checking... extending the logical relations.
• Martin-Löf à la CoQ: all meta-theory of MLTT, formalised in CoQ (and there’s more I have not told you about).

• $\text{MLTT}_{\text{map}}$: normalisation, decidability of type-checking... extending the logical relations.

• $\text{MLTT}_{\text{coe}}$: pen and paper, relatively straightforward extension of $\text{MLTT}_{\text{map}}$ (main difference: reduce types in $\text{coe}_{A,B}$).
• Martin-Löf à la CoQ: all meta-theory of MLTT, formalised in CoQ (and there’s more I have not told you about).

• $\text{MLTT}_{\text{map}}$: normalisation, decidability of type-checking… extending the logical relations.

• $\text{MLTT}_{\text{coe}}$: pen and paper, relatively straightforward extension of $\text{MLTT}_{\text{map}}$ (main difference: reduce types in $\text{coe}_{A,B}$).

• $\text{MLTT}_{\text{sub}}$: erasure from $\text{MLTT}_{\text{coe}}$ is type-preserving and invertible; in particular, “syntactic” coherence, up to conversion.
• Martin-Löf à la CoQ: all meta-theory of MLTT, formalised in CoQ (and there’s more I have not told you about).

• $\text{MLTT}_{\text{map}}$: normalisation, decidability of type-checking... extending the logical relations.

• $\text{MLTT}_{\text{coe}}$: pen and paper, relatively straightforward extension of $\text{MLTT}_{\text{map}}$ (main difference: reduce types in $\text{coe}_{A,B}$).

• $\text{MLTT}_{\text{sub}}$: erasure from $\text{MLTT}_{\text{coe}}$ is type-preserving and invertible; in particular, “syntactic” coherence, up to conversion.

• Main tool: bidirectional/algorithmic/canonical derivations.
The results

- Martin-Löf à la Coq: all meta-theory of MLTT, formalised in CoQ (and there’s more I have not told you about).

- $\text{MLTT}_{\text{map}}$: normalisation, decidability of type-checking... extending the logical relations.

- $\text{MLTT}_{\text{coe}}$: pen and paper, relatively straightforward extension of $\text{MLTT}_{\text{map}}$ (main difference: reduce types in $\text{coe}_{A,B}$).

- $\text{MLTT}_{\text{sub}}$: erasure from $\text{MLTT}_{\text{coe}}$ is type-preserving and invertible; in particular, “syntactic” coherence, up to conversion.

- Main tool: bidirectional/algorithmic/canonical derivations.

To interpret subsumptive structural subtyping, you (only) need functoriality equations.
WHAT WE’RE STILL UNHAPPY ABOUT

Martin-Löf à la CoQ

• renamings vs well-typed weakenings
• automation (proofs by reflection?)
• better structure and abstractions (categories?)
• more types!
What we’re still unhappy about

Martin-Löf à la Coq

- renamings vs well-typed weakenings
- automation (proofs by reflection?)
- better structure and abstractions (categories?)
- more types!

Functoriality and subtyping

- No formalisation of $\text{MLTT}_{\text{coe}} / \text{MLTT}_{\text{sub}}$
- No good story for elaborating $\text{MLTT}_{\text{coe}}$ to $\text{MLTT}_{\text{map}}$
- No general class of "good inductive types"
\[ \Gamma \vdash \text{map}_F f : F X \rightarrow F Y \quad \nLeftrightarrow \quad \Gamma \vdash \text{map}_F \text{id}^F_X t \cong t : F X \]

\[ \Gamma \vdash \text{map}_F f \left( \text{map}_F g \, t \right) \cong \text{map}_F (f \circ^F g) \, t : F Z \]