Canonicity of the Mahlo Universe

Master's thesis in computer science and engineering

Ondřej Kubánek

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

Canonicity of the Mahlo Universe

Ondřej Kubánek

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Canonicity of the Mahlo Universe
Ondřej Kubánek

Supervisor: András Kovács, Department of Computer Science and Engineering
Examiner: Thierry Coquand, Department of Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2025

Canonicity of the Mahlo Universe
Ondřej Kubánek
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The thesis presents a proof of canonicity of the external Mahlo universe using the gluing technique. The used metatheory is an extensional type theory extended with indexed induction-recursion to express the reducibility predicate for the external Mahlo universe. This is motivated by the fact that the Mahlo universe can express certain inductive-recursive definitions and so understanding the metatheory for the external Mahlo universe should help with metatheory for induction-recursion.

Keywords: type theory, metatheory, gluing, canonicity, mahlo universe, induction-recursion

Acknowledgements

I would like to thank my advisor András Kovács for his patience and enthusiasm and my family and friends for their help and support.

Ondřej Kubánek, Gothenburg, 2025-06-06

Contents

List of Figures

List of Tables

1

Introduction

This chapter introduces the concepts necessary to understand what an external Mahlo universe is and why it is interesting.

1.1 Dependent type theory

Dependent type theory (DTT) [1] is one of possible foundations of mathematics. One can also think of it as a convenient programming language for proofs, since it is often used in proof assistants. As with any other type system for programming languages, the main notions are types, terms, contexts and substitutions. The difference between DTT and other type systems is that types can depend on terms. Further in the text, type theory will always mean DTT and sometimes will be abbreviated to TT.

In the example below we define a function that returns the unit type if a natural number is zero and the empty type otherwise. If we get a term of this type for some n, the n must be equal to zero. The example is written in the proof assistant Agda [2]. In the thesis, any code with syntax highlighting is Agda and any without it is not Agda.

```
iszero : Nat → Set
iszero 0 = ⊤
iszero (suc _) = ⊥

zeroiszero : (n : Nat) → iszero n → n ≡ 0
zeroiszero 0 zz = refl
zeroiszero (suc _) ()
```

In the following subsections basic rules for DTT will be introduced. For a more thorough introduction refer to Hofmann [3].

1.1.1 Contexts

$$\frac{}{\cdot \text{ Context}}\ \textsc{EmptyCtx} \qquad \frac{\Gamma \text{ Context} \quad \Gamma \vdash A \text{ Type}}{\Gamma, x : A \text{ Context}}\ \textsc{ExtendCtx}$$

To form contexts, either we take the empty context or we extend a context with a type in that context. Note that the type depends on the context. In a simply typed system this would not be the case. In the rest of the rules all mentioned contexts will be assumed valid. Some types are also assumed valid as long as it is obvious where they come from.

1.1.2 Working with contexts

$$\frac{\Gamma \vdash A \text{ Type}}{\Gamma, x : A \vdash x : A}\ \textsc{Var} \qquad \frac{\Gamma, \Delta \vdash a : A \quad \Gamma \vdash B \text{ Type}}{\Gamma, x : B, \Delta \vdash a : A}\ \textsc{Weakening}$$

$$\frac{\Gamma \vdash t : T \qquad \Gamma, x : T, \Delta \vdash u : U}{\Gamma, \Delta \vdash u[t/x] : U[t/x]}\ \textsc{Subst-Tm}$$

$$\frac{\Gamma \vdash t : T \qquad \Gamma, x : T, \Delta \vdash U \text{ Type}}{\Gamma, \Delta \vdash U[t/x] \text{ Type}}\ \textsc{Subst-Ty}$$

The variable rule says that a variable from the context can be used as a valid term of its type.

The weakening rule says that if something holds in some context then adding something to that context does not change that. In other words, it says that extra assumptions do not interfere with a proof.

The substitution rules allow substituting a free variable in a term or a type. We use capture-free substitution.

1.1.3 Dependent function type

$$\frac{\Gamma \vdash A \text{ Type} \quad \Gamma, x : A \vdash B \text{ Type}}{\Gamma \vdash \Pi\ A\ B \text{ Type}}\ \textsc{Pi-Form} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : \Pi\ A\ B}\ \textsc{Pi-Intro}$$

$$\frac{\Gamma \vdash f : \Pi\ A\ B \quad \Gamma \vdash a : A}{\Gamma \vdash f\, a : B[a/x]}\ \textsc{Pi-Elim}$$

Dependent function types or Pi types allow us to truly use dependent types, since the return type of such a function depends on the term we pass it.

The introduction rule says that given a term in an extended context we can make a function term by binding the variable. Similarly, the elimination rule says that we can apply the function to a term by substituting the bound variable by the term.

One can think of the dependent function type as the $\forall$ operator from classical logic. We can see this in the introduction rule since to have a proof that $\Pi\ A\ B$ one has to form a function term which depends on a variable and thus cannot use some special structure of some concrete $x : A$.

1.1.4 Universes

$$\frac{}{\Gamma \vdash \mathrm{U}\ \mathrm{Type}}\ \textsc{U-Form} \qquad \frac{\Gamma \vdash a\ \mathrm{Type}}{\Gamma \vdash a : \mathrm{U}}\ \textsc{U-Intro} \qquad \frac{\Gamma \vdash a : \mathrm{U}}{\Gamma \vdash a\ \mathrm{Type}}\ \textsc{U-Elim}$$

Universes allow us to quantify over types. The introduction and elimination rules allow U to be a type of U which causes the type theory to be inconsistent [4, 5]. The usual solution is to have a hierarchy of universes. We shall call a function into a universe a *type family*.

The universes in the rules are à la Russell, since the terms of the universe are the actual types. In the thesis we only consider such universes. However, there are numerous other notions of universes. A notable example are universes à la Tarski which contain codes of a type instead of the types themselves.

In set theory simple universes correspond to some inaccessible cardinals since they are closed under function types.

1.1.5 Natural numbers

$$\frac{}{\Gamma \vdash \mathbb{N}\ \mathrm{Type}}\ \textsc{Nat-Form} \qquad \frac{}{\Gamma \vdash 0 : \mathbb{N}}\ \textsc{Zero-Intro}$$

$$\frac{\Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathsf{succ}(n) : \mathbb{N}}\ \textsc{Succ-Intro}$$

$$\frac{\begin{array}{c}\Gamma \vdash C : \mathbb{N} \to \mathsf{U} \\ \Gamma \vdash c_0 : C(0) \qquad \Gamma \vdash c_s : \Pi(n : \mathbb{N}).\, C(n) \to C(\mathsf{succ}(n)) \qquad \Gamma \vdash n : \mathbb{N}\end{array}}{\Gamma \vdash \mathsf{ind}_{\mathbb{N}}(C, c_0, c_s, n) : C(n)}\ \textsc{Nat-Elim}$$

$$\frac{\begin{array}{c}\Gamma \vdash C : \mathbb{N} \to \mathsf{U} \\ \Gamma \vdash c_0 : C(0) \qquad \Gamma \vdash c_s : \Pi(n : \mathbb{N}).C(n) \to C(\mathsf{succ}(n))\end{array}}{\Gamma \vdash \mathsf{ind}_{\mathbb{N}}(C, c_0, c_s, 0) \doteq z}\ \textsc{Nat-}\beta\textsc{-zero}$$

$$\frac{\begin{array}{c}\Gamma \vdash C : \mathbb{N} \to \mathsf{U} \\ \Gamma \vdash c_0 : C(0) \qquad \Gamma \vdash c_s : \Pi(n : \mathbb{N}).C(n) \to C(\mathsf{succ}(n)) \qquad \Gamma \vdash n : \mathbb{N}\end{array}}{\Gamma \vdash \mathsf{ind}_{\mathbb{N}}(C, c_0, c_s, \mathsf{succ}(n)) \doteq c_s(\mathsf{ind}_{\mathbb{N}}(C, c_0, c_s, n))}\ \textsc{Nat-}\beta\textsc{-succ}$$

Natural numbers are an example of an inductive type. The most interesting rule here is the elimination rule. It is precisely the induction principle for natural numbers. Given a predicate C, a proof that the base case holds for C and a proof that the induction step holds for C the predicate must hold for any n. The zero β computation rule says that the induction principle computes to the base case at zero.

Note, that unlike classical mathematics type theory has two notions of equality. The first one is definitional equality, which corresponds to the computational behaviour of type theory. For example, the Nat-β-zero rule introduces one such equality. Note that we use $\doteq$ for definitional equality. We will introduce the other notion in the next subsection.

1.1.6 Identity type

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : A}{\Gamma \vdash \mathsf{Id}_A(a,b) : \mathrm{Type}}\ \textsc{Id-Form} \qquad\qquad \frac{\Gamma \vdash a : A}{\Gamma \vdash \mathsf{refl}_A(a) : \mathsf{Id}_A(a,a)}\ \textsc{Id-Intro}$$

$$\frac{\begin{array}{c}\Gamma, x : A, y : A, p : \mathsf{Id}_A(x,y) \vdash C(x,y,p)\ \mathrm{Type}\\ \Gamma, x : A \vdash d : C(x,x,\mathsf{refl}_A(x))\\ \Gamma \vdash a : A \qquad \Gamma \vdash b : A \qquad \Gamma \vdash p : \mathsf{Id}_A(a,b)\end{array}}{\Gamma \vdash \mathsf{J}(d,a,b,p) : C(a,b,p)}\ \textsc{Id-Elim}$$

$$\frac{\begin{array}{c}\Gamma, x : A, y : A, p : \mathsf{Id}_A(x,y) \vdash C(x,y,p)\ \mathrm{Type}\\ \Gamma, x : A \vdash d : C(x,x,\mathsf{refl}_A(x))\\ \Gamma \vdash t : A \qquad \Gamma \vdash \mathsf{J}(d,t,t,\mathsf{refl}_A(t)) : C(t,t,\mathsf{refl}_A(t))\end{array}}{\Gamma \vdash \mathsf{J}(d,t,t,\mathsf{refl}_A(t)) \doteq d(t) : C(t,t,\mathsf{refl}_A(t))}\ \textsc{Id-}\beta$$

The second type of equality in type theory is the identity type. Unlike definitional equality one has to prove that two terms are the same by constructing term of their identity type. The only constructor for the equality type is refl, which witnesses the equality of a term with itself. Terms of an identity type can be eliminated by the Id-Elim rule (also known as J-rule), which states that to prove a property depending on an equality it is enough to prove it for refl.

1.1.6.1 Extensional type theory

One of initial questions about DTT was whether two terms of an identity type are equal. This was disproven by Hofmann and Streicher [6]. If we want such a property, we have to state it as an additional axiom. One type theory that assumes this axiom is *extensional type theory* (ETT). ETT assumes two additional axioms, the just mentioned *uniqueness of identity proofs* (UIP) and the *equality reflection axiom.*

$$\frac{\Gamma \vdash p : \mathsf{Id}_A(a,b) \qquad \Gamma \vdash q : \mathsf{Id}_A(a,b)}{\Gamma \vdash \mathsf{UIP}(p,q) : \mathsf{Id}_{\mathsf{Id}_A(a,b)}(p,q)}\ \mathrm{UIP} \qquad\qquad \frac{\Gamma \vdash p : \mathsf{Id}_A(a,b)}{\Gamma \vdash a \doteq b}\ \textsc{Id-Reflection}$$

Uniqueness of identity proofs states that every equality proof is refl and equality reflection states that provable identity implies definitional equality. Id-Reflection causes definitional equality to be undecidable since one can assume the rules of SKI-calculus as equalities in a context which makes checking whether two terms of such a type equivalent to the halting problem [7]. This means that extensional type theory can be hardly used in proof assistants but it is still consistent since the set-theoretic model does not distinguish between the two equalities anyway. In this sense using extensional type theory is similar to set theory and so it is more useful for paper proofs, which will also be our use for it.

1.2 Induction-recursion

IR is a way to allow a type theory to have more inductive types. In particular we can define a datatype and a function acting on the datatype at the same time (mutually). The function can act on the datatype even in negative positions which is crucial when modeling universes.

It was first used by Per Martin-Löf to specify the Tarski universe and later axiomatized by Dybjer [8]. Its most common use case is to prove metatheoretical properties of TTs [9]. Having IR in one universe allows us to define a Nat-indexed hierarchy within the universe. Hence, IR is a strong feature in a type theory, and we need even stronger assumptions in the metatheory to prove properties about such a type theory.

Below we have a type theory with a universe hierarchy indexed by natural numbers [10]. The module Step represents individual universe levels of the TT where the type family ensures that the smaller universe has a code in the bigger universe and that the elements of the smaller universe also have codes in the bigger universe. The operators $_1\_$ and $_2\_$ are just the corresponding projections of the dependent pair type. There are also some basic type formers such as dependent functions and products and natural numbers. Using Step we instantiate the `UEl` function which represents for each `n` the `n`-th universe `U n` and its interpretation function into the Agda universe `El`.

```
Fam : Set₁
Fam = Σ Set λ A → A → Set

module Step (AB : Fam) where
  A = ₁ AB
  B = ₂ AB

  mutual
    data U : Set where
      A'     : U
      B'     : A → U
      ℕ' 0'  : U
      Π' Σ'  : (A : U) → (El A → U) → U

    El : U → Set
    El A'         = A
    El (B' x)     = B x
    El ℕ'         = ℕ
    El 0'         = ⊥
    El (Π' A B) = (a : El A) → El (B a)
    El (Σ' A B) = Σ (El A) (El ∘ B)
  UEl : Fam
  UEl = U , El

UEl : ℕ → Fam
UEl zero      = Step.UEl (⊥ , λ ())
UEl (suc n) = Step.UEl (UEl n)

U : ℕ → Set
U n = ₁ (UEl n)

El : ∀ {n} → U n → Set
El {n} = ₂ (UEl n )
```

1.3 Mahlo universe

The external Mahlo universe is defined as a universe which for every mapping between two type families contains a subuniverse closed under that mapping. It was first discussed by Setzer [11]. Further in the text we will use "Mahlo universe" to refer to external Mahlo universes.

Note that an elimination principle for types in a Mahlo universe yields an inconsistency [12]. One can think of this as a positivity issue since it would mean that we would have a code in *Set* for any function *Set* → *Set* but this has a negative occurrence of *Set*.

The definition we will be using is derived from Figure 1.1 from Doel [13], which in turns takes its definition from Setzer [14]. In the definition, `Ma` specifies Mahlo subuniverses, and the `U+` and `El+` parameters together yield a function between type families. Note, that the definition implements the Mahlo universe using induction-recursion, so if a universe supports induction-recursion, it is a Mahlo universe.

```
module Doel
    (U+ : (A : Set) → (B : A → Set) → Set)
    (El+ : (A : Set) → (B : A → Set) → U+ A B → Set) where

  data Ma : Set
  El : Ma → Set

  data Ma where
    ℕ' 0' : Ma
    Π'  : (a : Ma) → (b : El a → Ma) → Ma
    U+' : (a : Ma) → (b : El a → Ma) → Ma
    El+' : (a : Ma) → (b : El a → Ma) → U+ (El a) (El ∘ b) → Ma

  El ℕ'            = ℕ
  El 0'            = ⊥
  El (Π' a b)      = (x : El a) → El (b x)
  El (U+' a b)     = U+ (El a) (El ∘ b)
  El (El+' a b x) = El+ (El a) (El ∘ b) x

  MaElim : (P : Ma → Set)
    → (n      : P ℕ')
    → (z      : P 0')
    → (pi     : ∀ a → P a → ∀ b → (∀ x → P (b x)) → P (Π' a b))
    → (u+     : ∀ a → P a → ∀ b → (∀ x → P (b x)) → P (U+' a b))
    → (el+    : (a : Ma) → P a → (b : El a → Ma) → (∀ x → P (b x))
                  → (x : U+ (El a) (El ∘ b)) → P (El+' a b x))
    → (u : Ma) → P u
  MaElim P n z pi u+ el+ ℕ'           = n
  MaElim P n z pi u+ el+ 0'           = z
  MaElim P n z pi u+ el+ (Π' a b)     =
    pi a (MaElim P n z pi u+ el+ a) b (MaElim P n z pi u+ el+ ∘ b)
  MaElim P n z pi u+ el+ (U+' a b)    =
    u+ a (MaElim P n z pi u+ el+ a) b (MaElim P n z pi u+ el+ ∘ b)
  MaElim P n z pi u+ el+ (El+' a b x) =
    el+ a (MaElim P n z pi u+ el+ a) b (MaElim P n z pi u+ el+ ∘ b) x
```

Figure 1.1: Agda definition of Mahlo subuniverses from Doel

In our definition below, we make some changes compared to Setzer's version. Our version has two advantages. Firstly, it has fewer rules. Secondly, instead of fixing some base type formers, we parameterize the subuniverses with a base type family, given by `U0` and `El0`. This can be used to specify base types as needed. The eliminator for the Mahlo universe is derived as the Dybjer-Setzer eliminator for the IR definition of the Mahlo subuniverse [15].

```
module Mahlo
    (U0  : Set)
    (El0 : U0 → Set)
    (U+ : (A : Set) → (B : A → Set) → Set)
    (El+ : (A : Set) → (B : A → Set) → U+ A B → Set) where

  data Ma : Set
  El : Ma → Set

  data Ma where
    El0' : U0 → Ma
    El+' : (a : Ma) → (b : El a → Ma) → U+ (El a) (El ∘ b) → Ma

  El (El0' u)       = El0 u
  El (El+' a b x) = El+ (El a) (El ∘ b) x

  MaElim : (P : Ma → Set)
    → (el0 : ∀ x → P (El0' x))
    → (el+ : (a : Ma) → P a → (b : El a → Ma) → (∀ x → P (b x))
                          → (x : U+ (El a) (El ∘ b)) → P (El+' a b x))
    → (u : Ma) → P u
  MaElim P el0 el+ (El0' x)       = el0 x
  MaElim P el0 el+ (El+' a b x) = el+ a (MaElim P el0 el+ a)
                                          b (MaElim P el0 el+ ∘ b) x
```

We prove that our definition is at least as strong as Setzer's in the following translation. Constructors of `U` which are not dependent are recovered from `El0'` and those with type dependencies are recovered from `El+'`. Finally, the type family mapping can be recovered from `El+`.

```
module Translation
  (U+ : (A : Set) → (B : A → Set) → Set)
  (El+ : (A : Set) → (B : A → Set) → U+ A B → Set) where

  data U0* : Set where
    ℕ'* 0'* : U0*

  El0* : U0* → Set
  El0* ℕ'* = ℕ
  El0* 0'* = ⊥
```

```
data U+* (A : Set) (B : A → Set) : Set where
  Π'*  : U+* A B
  U'+* : U+* A B
  El+'* : U+ A B → U+* A B

El+* : (A : Set) → (B : A → Set) → U+* A B → Set
El+* A B Π'*        = ∀ a → B a
El+* A B U'+*       = U+ A B
El+* A B (El+'* x) = El+ A B x

module M = Mahlo U0* El0* U+* El+*

Ma : Set
Ma = M.Ma

El : Ma → Set
El = M.El

ℕ' : Ma
ℕ' = M.El0' ℕ'*

0' : Ma
0' = M.El0' 0'*

Π' : (a : Ma) → (b : El a → Ma) → Ma
Π' a b = M.El+' a b Π'*

U+' : (a : Ma) → (b : El a → Ma) → Ma
U+' a b = M.El+' a b U'+*

El+' : (a : Ma) → (b : El a → Ma) → U+ (El a) (El ∘ b) → Ma
El+' a b x = M.El+' a b (El+'* x)

Elℕ'   : El ℕ'                       ≡ ℕ
El0'   : El 0'                       ≡ ⊥
ElΠ'   : ∀ {a b} → El (Π' a b)       ≡ ((x : El a) → El (b x))
ElU+'  : ∀ {a b} → El (U+' a b)      ≡ U+ (El a) (El ∘ b)
ElEl+' : ∀ {a b x} → El (El+' a b x) ≡ El+ (El a) (El ∘ b) x
Elℕ'   = refl
El0'   = refl
ElΠ'   = refl
ElU+'  = refl
ElEl+' = refl
```

```
MaElim : (P   : Ma → Set)
       → (n   : P ℕ')
       → (z   : P 0')
       → (pi  : ∀ a → P a → ∀ b → (∀ x → P (b x)) → P (Π' a b))
       → (u+  : ∀ a → P a → ∀ b → (∀ x → P (b x)) → P (U+' a b))
       → (el+ : (a : Ma) → P a → (b : El a → Ma) → (∀ x → P (b x))
                          → (x : U+ (El a) (El ∘ b)) → P (El+' a b x))
       → (u : Ma) → P u
MaElim P n z pi u+ el+ = M.MaElim
  P
  (λ { ℕ'* → n ; 0'* → z})
  (λ { a pa b pb Π'*      → pi a pa b pb ;
       a pa b pb U'+* → u+ a pa b pb ;
       a pa b pb (El+'* x) → el+ a pa b pb x})

MaElimℕ'   : ∀ {P n z pi u+ el+} → MaElim P n z pi u+ el+ ℕ' ≡ n
MaElim0'   : ∀ {P n z pi u+ el+} → MaElim P n z pi u+ el+ 0' ≡ z
MaElimΠ'   : ∀ {P n z pi u+ el+ a b}
  → MaElim P n z pi u+ el+ (Π' a b)  ≡
      pi a (MaElim P n z pi u+ el+ a) b (MaElim P n z pi u+ el+ ∘ b)
MaElimU+' : ∀ {P n z pi u+ el+ a b }
  → MaElim P n z pi u+ el+ (U+' a b) ≡
      u+ a (MaElim P n z pi u+ el+ a) b (MaElim P n z pi u+ el+ ∘ b)
MaElimEl+' : ∀ {P n z pi u+ el+ a b x}
  → MaElim P n z pi u+ el+ (El+' a b x) ≡
      el+ a (MaElim P n z pi u+ el+ a)
          b (MaElim P n z pi u+ el+ ∘ b) x
MaElimℕ'   = refl
MaElim0'   = refl
MaElimΠ'   = refl
MaElimU+'  = refl
MaElimEl+' = refl
```

For example, using our definition of Mahlo we can define some IR definitions but it is not clear what the relationship between Mahlo and IR is. Using this definition of Mahlo, we can define the same Nat-indexed hierarchy of universes in Figure 1.2 as in Figure 1.2. We again have a `Step` module for each universe level and we interpret the family exactly the same way. `El0` and `El+` in this case provide us interpretations of the type formers in the Agda universe.

```
module Example where
  Fam : Set₁
  Fam = Σ Set λ A → A → Set


  module Step (AB : Fam) where
    A = ₁ AB
    B = ₂ AB

    data U0 : Set where
      A'          : U0
      B'          : A → U0
      0' 1' 2' ℕ' : U0

    El0 : U0 → Set
    El0 A'     = A
    El0 (B' x) = B x
    El0 0'     = ⊥
    El0 1'     = ⊤
    El0 2'     = Bool
    El0 ℕ'     = ℕ

    data U+ (A : Set)(B : A → Set) : Set where
      Π' : U+ A B
      Σ' : U+ A B

    El+ : (A : Set) → (B : A → Set) → U+ A B → Set
    El+ A B Π' = (x : A) → B x
    El+ A B Σ' = Σ A B

    UEl : Fam
    UEl = Mahlo.Ma U0 El0 U+ El+ ,
          Mahlo.El U0 El0 U+ El+

  UEl : ℕ → Fam
  UEl zero    = Step.UEl (⊥ , λ ())
  UEl (suc n) = Step.UEl (UEl n)

  U : ℕ → Set
  U n = ₁ (UEl n)

  El : ∀ {n} → U n → Set
  El {n} = ₂ (UEl n)
```

Figure 1.2: Type theory with a Nat-indexed hierarchy of universes, using the Mahloness of Set

2 Metatheory

We cannot construct an internal model of a type theory in itself due to the Gödel's Second incompleteness theorem [16]. So to prove something about a type theory we need a stronger metatheory (foundations) to express its model. In this chapter, we will describe the metatheory and the basic setup for formalizing the object theory and proving properties about it.

2.1 DTT models

Since we will be doing denotational semantics, we have to introduce mathematical structures that have similar properties as DTT. We will use categories with families for the specification of models and consider the syntax as an initial model, given as a quotient inductive-inductive type.

2.1.1 Generalized algebraic theories

One way to present dependent type theory is to use *Generalized algebraic theories* (GAT) introduced by Cartmell [17]. GATs are a generalization of algebraic theories from universal algebra.

Algebraic theories [18] are defined by constants, operators and equalities. The example of an algebraic theory below is the SKI combinator calculus where the combinators are the constants, the only operation is composition and the equalities specify how the combinators compute.

```
Tm       : Set
S, K, I : Tm
_∘_      : Tm → Tm → Tm

S ∘ x ∘ y ∘ z = x ∘ z ∘ (y ∘ z)
K ∘ x ∘ y = y
I ∘ x = x
```

Generalized algebraic theories are an extension of algebraic theories that allows multiple dependent sorts, dependently typed operations and equations. The example of a GAT below is the theory of categories, morphisms can be given as a family of sets indexed by source and target objects, and well-typed composition of morphisms can be stated.

```
O        : Set
Hom(_,_) : O → O → Set

id       : Hom(x,x)

_∘_      : Hom(y,z) → Hom(x,y) → Hom(x,z)

id ∘ x = x
x ∘ id = x
(x ∘ y) ∘ z = x ∘ (y ∘ z)
```

2.1.2 Categories with families

Introduced by Dybjer [19], *categories with families* (CwFs) are a specification of dependent type theory. We chose CwFs instead of other specifications [20, 21] mainly because our canonicity proof is based on the paper Gluing for Type Theory [22].

In Figure 2.1 we can see four sorts that is contexts, substitutions, types and terms. Mirroring the DTT rules, contexts can be formed as either empty or extended by a type in the context. Substitutions are similar, but instead of being extended by a type they are extended by a term. CwFs specify how should terms and types be substituted but concrete term and type formers depend on the specific TT. Note that the theory presented by CwFs is already well-typed.

There are two important things to know about CwFs. Firstly, substituting is contravariant, since this allows us to extend substitutions with a term substituted by the original substitution. Secondly, `p` represents weakening and `q` represents the de Bruijn index 0. So if we weaken `q` twice then it is pointing to the third variable in the context counting from the right.

```
swap : Sub (•,A,B) (•,B,A)
swap = ε, q, q[p]

(•,Bool,Nat)[swap] = (•,Nat,Bool)

diag : Sub (•,A) (•,A,A)
diag = ε, q, q

(•,Bool,Bool)[swap] = (•,Bool)
```

In this example we have two different substitutions. The `swap` substitution takes a context with two variable and swaps them and the `diag` substitution maps two variables of the same type to the same variable.

```
Con   : Set
Ty    : Con → Set
Tm    : (Γ : Con) → Ty Γ → Set
Sub   : Con → Con → Set

●     : Con
_,_   : (Γ : Con) → Ty Γ → Con

_[_]  : Ty Δ → Sub Γ Δ → Ty Γ

ε     : Sub Γ ●
_,_   : {A : Ty Δ} → (σ : Sub Γ Δ) → Tm Γ (A[σ]) → Sub Γ (Δ, A)

εη    : (σ : Sub Γ ●) → σ = ε
,∘    : (σ, t) ∘ δ = (σ ∘ δ, t[δ])

id    : Sub Γ Γ
[id]  : {A : Ty Γ} → A[id] = A

_∘_   : Sub Δ Σ → Sub Γ Δ → Sub Γ Σ
ass   : σ ∘ (δ ∘ γ) = (σ ∘ δ) ∘ γ
idl   : id ∘ σ      = σ
idr   : σ ∘ id      = σ

[∘]   : A[σ ∘ δ] = A[σ][δ]

_[_]  : Tm Δ A → (σ : Sub Γ Δ) → Tm Γ (A[σ])


p     : Sub (Γ, A) Γ
p∘    : p ∘ (σ, t) = σ

q     : Tm (Γ, A) (A[p])
q[]   : q[σ, t] = t

,η    : (p, q) = id
```

Figure 2.1: Basic rules of CwF as a GAT

2.1.3 Quotient inductive-inductive types

Quotient inductive-inductive types (QIITs) [23, 24] allow us to define multiple sorts where later ones can be indexed over the previous ones and also allow for equations on the type. This internalizes GATs into type theory. In particular this allows us to define syntax of type theories quotiented by conversion as we will see in Section 2.2.2.2. Below we have an example of a type theory which specifies contexts, the unit type, its term and the fact that all its terms are exactly the unit.

```
data Context : Set where
    •   : Context
    _,_ : (Γ : Context) → Ty Γ → Context

data Ty : Context → Set where
    1 : {Γ : Context} → Ty Γ

dat Tm : (Γ : Context) → Ty Γ → Set where
    () : {Γ : Context} → Tm Γ 1
    eq : {Γ : Context} (a : Tm Γ 1) → a ≡ ()
```

This allows us to have a category of CwFs with an initial object internally to a TT with QIITs.

2.2 Proof techniques

In this section we will introduce some properties one would like to prove about TT. We also give an overview of proof techniques used for this purpose. Finally, we will look at gluing, which is the proof technique we will use for the Mahlo universe.

2.2.1 Metatheoretical properties

Canonicity and normalization are two notable desirable properties of a type theory.

Canonicity means that closed terms are definitionally equal to a term that's only built from contructors. For example, a canonical term of type Nat is a finite application of successor to zero. In particular, the empty type has no constructors, so canonicity implies consistency.

Normalization means that all terms are convertible to a normal form. This usually means either values or a spine of constructors stuck on a variable (neutral term). Normalization is a common method to show that conversion is decidable, which is required for decidable type checking.

2.2.2 Proof technique overview

The technique we will use is categorical gluing [22, 25]. Gluing is an intrinsic technique quotiented by conversion, based on the logical predicate technique. The goal of this subsection is to explain most of the terms involved in this technique.

2.2.2.1 Extrinsic vs. intrinsic

Extrinsic methods start with untyped syntax. Untyped terms are then constrained by typing and well-formedness relations. This has the benefit of being simpler since the dependent typing is not that heavy and one has a notion of ill-typed terms, which makes it easier for proof assistants [9, 26, 27].

Intrinsic methods start with well-typed syntax and so do not need to care about ill-typed terms. This allows for shorter proofs with pen and paper, but the large amount of type indexing makes machine-checked formalization harder. CwFs are an example of an intrinsic specification of the sorts and substitutions of a type theory [19, 28, 29].

Using extrinsic methods would be somewhat difficult for the Mahlo universe. We cannot use IR to represent a logical relation for the theory like [9], since that would give us an elimination principle for the Mahlo universe, which would be inconsistent. The proof would also be large.

2.2.2.2 Syntax quotiented by conversion

Quotienting is broadly speaking taking equivalence classes of elements instead of elements themselves based on some structure respecting equivalence. For example, in groups one gets quotients for each normal subgroup.

When we talk about conversion in TT it means definitional equality since terms that are definitionally equal are of the same type and so taking quotient of TT by the equivalence classes defined by the computation rules is a good notion. This in particular means that substitutions compute definitionally in ETT or set theory. GATs and QIITs fall into syntax that is intrinsic and quotiented by conversion [30].

The cost of this is that quotients do not behave well in proof assistants and you will either get into transport hell or will have to use even more theory on top [31–33].

2.2.2.3 Logical predicate

Logical predicate is a proof technique used for normalization arguments in various type theories. First used by Tait [34], the technique defines inductively a predicate on the syntax of the type theory.

There are two main ideas of logical predicates. Firstly, function types must preserve the logical predicate, that is, for any normalizing input their output must be also normalizing. Secondly, even if we are only proving canonicity, we still have to handle arbitrary contexts by closing them with closing substitutions. This is due to the fact that to handle the function application case we must have a notion of valid closing substitutions.

2.2.3 Gluing

Our definition of gluing closely follows Gluing for Type Theory [22]. In particular, we will use gluing instantiated with the global sections functor, since we are only proving canonicity.

Gluing takes two models and a weak morphism between them and results in a displayed model over the source model. In our case, the source model is the syntax, the weak morphism is the global sections functor, the target model is the standard model and we get canonicity as the unique section from the displayed model into the syntax. In the following subsection we will explain what this means and what is the connection to canonicity.

2.2.3.1 Standard model

The *standard model* interprets every object-theoretic construction with an analogous matatheoretical one.

For example, if we work in Set theory as metatheory, the standard model of a type theory would interpret types as (families of) sets, functions as functions and universes as a suitable set-theoretic notion of universe. On the other hand, if we work in a type theory, the standard model interprets type formers with the same type formers in the metatheory.

2.2.3.2 Displayed model

A *displayed model* over a model `M` gives a predicate for each sort of a CwF such that each predicate depends on the same sort from `M` and on the predicate on its indexes and it has to be preserved by all functions and respect all equations.

For example, given a type in `M` in context Γ and the predicate for contexts instantiated with Γ we define a predicate for the corresponding `M` type.

A *section* of a displayed model is a CwF-morphism from `M` into the displayed model. In particular, if `M` is the syntax for any displayed model over it the section is unique by initiality of syntax. Having a section for a displayed model means that the elements of `M` satisfy the predicates over them.

Another example is a displayed model for natural numbers. The first part is a definition of some model of natural numbers. Secondly, we define the ^{D} versions of all of the components of the model are the corresponding components of the displayed model. Lastly, the section ^{S} for natural numbers witnesses that the predicate the zero is actually true only for zero and similarly for the successor function.

```
Nat   : Set
z     : Nat
succ  : Nat → Nat
ind   : (C : Nat → Set) → C z →
         ({n : Nat} → C n → C (succ n))
         → (n : Nat) → C n

Nat^D  : Nat → Set
z^D    : Nat^D z
succ^D : (n : Nat) → N^D n → N^D (succ n)
```

```
ind^D   : (C : (n : Nat) → Nat^D n → Set) → C z z^D
          → ({n : Nat} → {n^D : Nat^D n}
             → C n n^D → C (succ n) (succ^D n))
          →  (n : Nat) → (n^D : Nat^D n) → C n n^D

Nat^S   : (n : Nat) → Nat^D s
z^S     : Nat^S z = z^D
s^S     : (n : Nat) → Nat^S (succ n) = succ^D n (Nat^S n)
```

Notice that the displayed model and the section are precisely the induction principle and the computation rules for natural numbers like in Subsection 1.1.5. For any algebra the condition that every displayed algebra has a section is precisely the induction principle.

2.2.3.3 Global sections functor

Weak morphism is a CwF-morphism that does not preserve contexts up to equality, but up to isomorphism. The global sections functor is one such morphism. While there are more general notions of global sections functors, we will only explain the one which targets the standard model.

The global sections functor maps a context to the set of closed substitutions that target the context. Substitutions are sent into functions that get the closing substitution from their source context and transform it into a closing substitution of the target context by composition with the substitution itself. Types are sent to a function that takes the closing substitution for its context and sends it to terms of the type closed by that substitution. Lastly, terms are sent to functions that get a closing substitution and produces the term closed by that substitution (into the image of its type).

```
(Γ : Con)       → Sub • Γ
(γ : Sub Γ Δ)  → λ δ . γ ∘ δ
(A : Ty Γ)      → λ δ . Tm • A[δ]
(t : Tm Γ A)   → λ δ . t[δ]
```

Note that the global sections functor is actually a weak morphism since empty context does not get sent to the empty context (which is the singleton set), but to the set of substitutions between empty contexts, which is merely isomorphic by the CwF equation $\varepsilon\eta$.

2.2.3.4 Gluing for canonicity

Gluing defines a displayed model, where for contexts and types, all the arguments from the source model are mapped by the weak morphism, and substitutions and terms preserve the predicate. For example, since in our case the weak morphism is the global sections functor `F`, instead of `(Γ : Con) → Set` we get `F Γ → Set` which computes to `Sub • Γ → Set`.

Unfolding the definition for all the sorts leads to the following definitions. Contexts are interpreted as predicates over closing substitutions. Substitutions respect that predicate. Types for a closed term define a predicate for when a term of the type is a value. Lastly, terms must respect the predicates. For booleans we say that a term is a value if it is `true` or `false` and it is obvious that the term true fulfills the predicate.

$$\begin{aligned}
&\texttt{Con}^p\ \Gamma && = \texttt{Sub}\ \bullet\ \Gamma \to \texttt{Set}\\
&\texttt{Sub}^p\ \Gamma\ \Delta\ \sigma && = (\gamma\ :\ \texttt{Sub}\ \bullet\ \Gamma) \to \Gamma^p\ \gamma \to \Delta^p\ (\sigma\ \circ\ \gamma)\\
&\texttt{Ty}^p\ \Gamma\ \texttt{A} && = (\gamma\ :\ \texttt{Sub}\ \bullet\ \Gamma) \to \Gamma^p\ \gamma \to \texttt{Tm}\ \bullet\ (\texttt{A}[\gamma]) \to \texttt{Set}\\
&\texttt{Tm}^p\ \Gamma\ \texttt{A}\ \texttt{t} && = (\gamma\ :\ \texttt{Sub}\ \bullet\ \Gamma)(\gamma^p\ :\ \Gamma^p\ \gamma) \to \texttt{A}^p\ \gamma\ \gamma^p\ (\texttt{t}[\gamma])
\end{aligned}$$

$$\begin{aligned}
&\texttt{Bool}^p\ \_\ \_\ \texttt{t} && = (\texttt{t} \equiv \texttt{true}) + (\texttt{t} \equiv \texttt{false})\\
&\texttt{true}^p\ \_\ \_ && = \texttt{inl refl}
\end{aligned}$$

The induction principle then yields a section of the displayed model, which implies that all elements of the syntax satisfy the canonicity predicates.

In the rest of the thesis, the syntax we will be using is that the canonicity predicate of a element of syntax `t` is $\texttt{t}^p$ like in the example above and since this is defined inductively all the subterms of `t` satisfy the canonicity predicate.

3

Canonicity of the Mahlo universe

In this chapter we present our proof of canonicity of the Mahlo universe.

3.1 Metatheory

We work in an extensional type theory with QIITs, we have $\omega + 1$ universes with indexed IR [35] to express the canonicity predicate for the Mahlo universe. For readability of the proof, the universes are Russell-style with cumulative subtyping and universe polymorphism.

Below we have rules for cumulative subtyping specifying a preorder. The subtyping respects type formers and a smaller universe is a subtype of larges universes. The most important part is that we can implicitly cast types in smaller universes to bigger universes.

$$\frac{}{\Gamma \vdash A \leq A} \qquad \frac{\Gamma \vdash A \leq B \quad \Gamma \vdash B \leq C}{\Gamma \vdash A \leq C} \qquad \frac{\Gamma \vdash A \leq A' \quad \Gamma \vdash t : A}{\Gamma \vdash t : A'}$$

$$\frac{i \leq j}{\Gamma \vdash (Ui) \leq (Uj)} \qquad \frac{\Gamma, x : A \vdash B \leq B'}{\Gamma \vdash (x : A) \to B \leq (x : A) \to B'}$$

3.2 Syntax

Our syntax has a Nat-indexed hierarchy of Russell-style universes, where each universe is a Mahlo universe. This could be fairly easily extended with more type formers since gluing is modular with respect to them.

We use Agda-like notation which can be translated to the GAT of CwFs. Further we use some syntactic sugar. Specifically p^N for iterated composition of weakenings, $q^N = q[p^N]$ for an iteratively weakened variable. $t \mathbin{\$\$} u = (app\ t)[id,\ u]$ for function application, note that this is left-associative so we can write $f \mathbin{\$\$} x \mathbin{\$\$} y$ for $(f(x))(y))$. $\sigma\!\uparrow$ is used for weakening under a binder $(\sigma \circ p, q)$. We use module syntax and implicit arguments but this desugars to explicit arguments and extra function arguments corresponing to module parameters.

```
Con : Set
Sub : Con → Con → Set
Ty  : Con → ∀ n → Set
Tm  : (Γ : Con) → ∀ {n} → Ty Γ n → Set

Π   : {Γ : Con} {i j : MetaNat} → (A : Ty Γ i) → Ty (Γ, A) j
                                                → Ty Γ (i ⊔ j)
app : (Tm Γ (Π A B)) → Tm (Γ,A) B
lam : Tm (Γ,A) B → (Tm Γ (Π A B))

U   : {Γ : Con} (n : MetaNat) → Ty Γ (suc n)
ElU : {Γ : Con} {n : MetaNat} → Tm Γ (U n) → Ty Γ n
c   : {Γ : Con} {n : MetaNat} → Ty Γ n → Tm Γ (U n)
Tm Γ (U n) = Ty Γ n
UEl t      = t
c t        = t
```

The syntax specifies the sorts, function types and Russell universes, which will become Mahlo once we add the subuniverses. Our Russell universes are specified as Coquand-style universes with extra equations. Since our syntax is a CwF the other rules of CwFs still apply, see Figure 2.1.

3.3 Syntax for the Mahlo universe

This section contains the syntax for Mahlo subuniverses using CwFs.

```
module Ma
  {i   : MetaNat}
  (U0  : Tm Γ (U i))
  (El0 : Tm Γ (U0 → (U i)))
  (U+  : Tm Γ ((U i) → (q → (U i)) → (U i)))
  (El+ : Tm Γ ((U i) → (q → (U i)) → U+ $$ q¹ $$ q → (U i)))
  where

  Ma   : Tm Γ (U i)
  (Ma U0 El0 U+ El+)[γ] ≡ Ma U0[γ] El0[γ] U+[γ] El+[γ]

  El0' : Tm Γ U0 → Tm Γ Ma
  (El0' u)[γ] ≡ (El0' u[γ])

  El+' : (A : Tm Γ Ma) → (B : Tm Γ (El $$ A → Ma))
         → Tm Γ (U+ $$ (El $$ A) $$ (lam (El[p] $$ (B[p] $$ q))))
         → Tm Γ Ma
  (El+' A B C)[γ] ≡ (El+' A[γ] B[γ] C[γ])
```

```
El   : Tm Γ (Ma → (U i))
El $$ (El0' x)     ≡ El0 $$ x
El $$ (El+' A B C) ≡ El+ $$ (El $$ A)
                         $$ (lam (El[p] $$ (B[p] $$ q))) $$ C

maElim : {j : MetaNat}(P : Tm Γ (Ma → (U j))
      → Tm Γ (U0 → P $$ (El0' $$ q))
      → Tm Γ (Ma → P $$ q → (El $$ q¹ → Ma)
        → (lam (P[p] $$ (q¹ $$ q)))
        → U+ $$ (El $$ q³) $$ lam (El[p] $$ (q² $$ q))
        → P $$ (El+' $$ q⁴ $$ q² $$ q)) →
         (A : Tm Γ Ma) → Tm Γ (P A)
maElim P el0 el+ (El0' x)     = el0 $$ x
maElim P el0 el+ (El+' a b x) =
  el+ $$ a $$
    (maElim P el0 el+ a)
    $$ b $$
    (lam (maElim P[p] el0[p] el+[p] (b[p] $$ q)))
    $$ x

(maElim P el0 el+ t)[γ] ≡
        (maElim P[γ] el0[γ] el+[γ] t[γ])
```

This follows the Agda definition 1.3 but also the Dybjer-Setzer eliminator for IR [15].

3.4 Canonicity model without the Mahlo universe

In this section, we describe the canonicity model for our syntax, excluding the Mahlo subuniverses. Any element of syntax with the ^{p} suffix is the canonicity predicate for the element. The proof itself is fairly standard [33]. Substitutions and contexts are valid if they are empty or they are an extension of a valid context by a valid term or type respectively. Function types preserve the predicate like in any other proof by logical predicate. A term `A` with type `U` is interpreted as a predicate over closed terms of type `A`. Lastly the `q` and `p` behave like a first and second projections from the predicate of the context which again respects the CwF structure. Equations for this part of the syntax are omitted.

In the following, instead of directly specifying the components of the displayed model, we specify the components of its section, which looks like a "recursive" definition, and which is more readable.

```
(Γ : Con)ᵖ       : Sub • Γ → Set ω

(σ : Sub Γ Δ)ᵖ : (γ : Sub • Γ) → Γᵖ γ → Δᵖ (σ ∘ γ)

(A : Ty n Γ)ᵖ  : (γ : Sub • Γ) → Γᵖ γ → Tm • A[γ] → Set n
```

```
(t : Tm $\Gamma$ A)$^p$  : ($\gamma$ : Sub $\bullet$ $\Gamma$) ($\gamma^p$ : $\Gamma^p$ $\gamma$) $\to$ A$^p$ $\gamma$ $\gamma^p$ t[$\gamma$]

$\bullet^p$ : Con$^p$ $\bullet$
$\bullet^p$ _ = $\top$

($\Gamma$ , a)$^p$ : Con$^p$ ($\Gamma$ , a)
($\Gamma$ , A)$^p$ ($\gamma$ , a) = ($\gamma^p$ : $\Gamma^p$ $\gamma$) $\times$ (A$^p$ $\gamma$ $\gamma^p$ a)

$\varepsilon^p$ $\gamma$ $\gamma^p$ = $\top$

($\delta$ , a)$^p$ $\gamma$ $\gamma^p$ = $\delta^p$ $\gamma$ $\gamma^p$ , a$^p$ ($\delta$ $\circ$ $\gamma$) ($\delta^p$ $\gamma$ $\gamma^p$)

id$^p$ $\gamma$ $\gamma^p$ = $\gamma^p$

p$^p$ ($\gamma$, $\alpha$) ($\gamma^p$, $\alpha^p$) = $\gamma^p$
q$^p$ ($\gamma$, $\alpha$) ($\gamma^p$, $\alpha^p$) = $\alpha^p$

($\delta$ $\circ$ $\sigma$)$^p$ $\gamma$ $\gamma^p$ = $\delta^p$ ($\sigma^p$ $\gamma$ $\gamma^p$) ($\sigma$ $\circ$ $\gamma$)

(A[$\delta$])$^p$ $\gamma$ $\gamma^p$ a = A$^p$ ($\delta$ $\circ$ $\gamma$) ($\delta^p$ $\gamma$ $\gamma^p$) a

(t[$\delta$])$^p$ $\gamma$ $\gamma^p$ = t$^p$ ($\delta$ $\circ$ $\gamma$) ($\delta^p$ $\gamma$ $\gamma^p$)

(U i)$^p$ $\gamma$ $\gamma^p$ a   = Tm $\bullet$ (El a) $\to$ Set i
(El a)$^p$ $\gamma$ $\gamma^p$ t = a$^p$ $\gamma$ $\gamma^p$ t
(c a)$^p$ $\gamma$ $\gamma^p$     = $\lambda$ t $\to$ a$^p$ $\gamma$ $\gamma^p$ t

(t : Tm $\Gamma$ (U j)) $\to$ (t[$\delta$]Ty)$^p$ $\gamma$ $\gamma^p$ = (t[$\delta$]Tm)$^p$ $\gamma$ $\gamma^p$

($\Pi$ A B)$^p$ : $\forall$ $\gamma$ $\gamma^p$ $\to$ Tm $\bullet$ ($\Pi$ A[$\gamma$] B[$\gamma\uparrow$]) $\to$ Set (i $\sqcup$ j)
($\Pi$ A B)$^p$ $\gamma$ $\gamma^p$ f = (a : Tm $\bullet$ A[$\gamma$]) (a$^p$ : A$^p$ a)
                         $\to$  B$^p$ ($\gamma$, a) ($\gamma^p$, a$^p$) (f $$ a)

(lam f)$^p$ $\gamma$ $\gamma^p$ a a$^p$       = f$^p$ ($\gamma$, a) ($\gamma^p$, a$^p$)
(app f)$^p$ ($\gamma$, a) ($\gamma$, a)$^p$ = f$^p$ $\gamma$ $\gamma^p$ a a$^p$
```

3.5 Predicate for the Mahlo universe

Here we define the canonicity predicate for the Mahlo universe. Since the Mahlo universe is a strong assumption we need to use indexed IR to define it. The predicate expresses when a term of the Mahlo subuniverse is canonical. We also define the Dybjer-Setzer eliminator for our predicate [35].

Note that `Ma`^{p} `U0 U0`^{p}`...` is not the same as `(Ma U0 El0 ...)`^{p}, since the first one is the metatheoretical predicate used to instantiate the gluing model and the other is the thing being instantiated.

```
module {i : MetaNat}
  (U0    : Tm • (U i))
  (U0^p  : Tm • U0 → Set i)
  (El0   : Tm • (U0 → (U i)))
  (El0^p : (u : Tm • U0) (u^p : U0^p u) → Tm • (El0 $$ u) → Set i)
  (U+    : Tm • ((A : (U i)) → (A → (U i)) → (U i)))
  (U+^p  : (A : Tm • (U i)) (A^p : Tm • A → Set i)
           (B : Tm • (A → (U i)))
           (B^p  : (a : Tm • A)(a^p : A^p a) → Tm • (B $$ a) → Set i)
           → Tm • (U+ $$ A $$ B) → Set i)
  (El+   : Tm • ({A B} → U+ $$ A $$ B → (U i))
  (El+^p : (A : Tm • U) (A^p : Tm • A → Set i) (B : Tm • (A → (U i)))
            (B^p : (a : Tm • A)(a^p : A^p a) → Tm • (B $$ a) → Set i)
            (u+ : Tm • (U+ $$ A $$ B)) → (u+^p : U+^p A A^p B B^p u+)
                                         → Tm • (El+ $$ u+) → Set i)

data Ma^p : Tm • (Ma U0 El0 U+ El+) → Set i where
   El0'^p : (u : Tm • U0) → (U0^p u) → Ma^p (El0' $$ u)
   El+'^p : (A : Tm • Ma) → (A^p : Ma^p A) → (B : Tm • (El $$ A → Ma))
          → (B^p : Tm • (El $$ A) → El^p A A^p a → Ma^p (B $$ a))
          → (C : Tm • (U+ $$ (El $$ A) $$
                             (lam (El[p] $$ (B[p] $$ q)))))
          → U+^p (El $$ A) (El^p A A^p) (lam (El[p] $$ (B[p] $$ q)))
                          (λ a a^p → El^p (B $$ a) (B^p a a^p)) C
          → Ma^p (El+' $$ A $$ B $$ C)

El^p : (u : Tm • (Ma U0 El0 U+ El+)) → (u^p : Ma^p u) → Tm • (El $$ u)
                                                            → Set i
El^p m (El0'^p u u^p) a              = El0^p u u^p a
El^p m (El+'^p A A^p B B^p u+ u+^p) t =
              El+^p (El $$ A) (El^p A A^p) (lam (El[p] $$ (B[p] $$ q)))
             (λ a a^p → El^p (B $$ a) (B^p a a^p)) u+ u+^p t
```

```
Ma^pElim :
  {j : MetaNat}
  → (P : Tm • ((Ma U0 El0 U+ El+) → U j))
  → (P^p : (m : Tm • (Ma U0 El0 U+ El+))
    → Ma^p m → Set j)
  → Tm • (U0 → P $$ (El0' $$ q))
  → ((u : Tm • U0) → (u^p : U0^p u)
    → P^p (El0' u) (El0'^p u u^p))
  → Tm • (Ma → P $$ q → (El $$ q^1 → Ma)
    → (lam (P[p] $$ (q^1 $$ q)))
    → U+ $$ (El $$ q^3) $$ (lam (El[p] $$ q^2 $$ q))
    → P $$ (El+' $$ q^4 $$ q^2 $$ q) →
  → ((A : Tm • Ma) → (A^p : Ma^p A)
    → (PA  : Tm • (P $$ A))
    → (PA^p : P^p A A^p (P $$ A))
    → (B : Tm • (El $$ A → Ma))
    → (B^p : (a : Tm • (El $$ A))
      → El^p A A^p a → Ma^p (B $$ a))
    → (PB  : Tm • (lam (P[p] $$ (q^1 $$ q)))
    → (PB^p : (a : Tm • (El $$ A))
      → (a^p : El^p A A^p a)
      → P^p (B $$ a) (B^p a a^p) (P $$ (B $$ a)))
    → (C : Tm • (U+ $$ (El $$ A) $$ (lam (El[p] $$ (B[p] $$ q))))
    → (C^p : U+^p (El $$ A) (El^p A A^p) (lam (El[p] $$ (B[p] $$ q)))
                    (λ a a^p → El^p (B $$ a) (B^p a a^p)) C)
    → P^p (El+' $$ A $$ B $$ C) (El+'^p A A^p B B^p C C^p)) →
  → (m : Tm • (Ma U0 El0 U+ El+)) → (m^p : Ma^p m)
  → P^p m m^p
Ma^pElim P P^p pel0 pel0^p pel+ pel+^p m (El0'^p u u^p) = pel0^p u u^p
Ma^pElim P P^p pel0 pel0^p pel+ pel+^p m (El+'^p A A^p B B^p C C^p) =
  pel+^p A A^p (maElim P pel0 pel+ A)
    (Ma^pElim P P^p pel0 pel0^p pel+ pel+^p A A^p) B B^p
    (λ a → maElim P pel0 pel+ a)
    (λ a a^p → Ma^pElim P P^p pel0 pel0^p pel+ pel+^p
              (B $$ a) (B^p a a^p))
    C C^p
```

3.6 Canonicity model for the Mahlo universe

Lastly, we finish the proof by interpreting the syntactic Ma rules using the metatheoretic `Ma`^{p}. Since we set up the predicate similarly to the syntax of the Mahlo universe, the interpretation itself is pretty easy. Note that the proof that the gluing model respects substitution equations, equations defining `El` and equations defining `maElim`.These equations are in Appendix A since they are not very enlightening.

```
(Ma U0 El0 U+ El+)^p : (γ : Sub • Γ) → (γ^p : Γ^p γ)
              → Tm • ((Ma U0 El0 U+ El+)[γ]) → Set i
(Ma U0 El0 U+ El+)^p γ γ^p t = Ma^p U0[γ] (U0^p γ γ^p) El0[γ] (El0^p γ γ^p)
                                 U+[γ] (U+^p γ γ^p) El+[γ] (El+^p γ γ^p) t

(El0' u)^p : (γ : Sub • Γ) (γ^p : Γ^p γ)
                                → (Ma U0 El0 U+ El+)^p γ γ^p (El0' u[γ])
(El0' u)^p γ γ^p = El0'^p u[γ] (u^p γ γ^p)

(El+' A B C)^p : (γ : Sub • Γ) (γ^p : Γ^p γ)
              → (Ma U0 El0 U+ El+)^p γ γ^p (U+' {A[γ]} {B[γ]} C[γ])
(El+' A B C)^p γ γ^p = El+'^p A[γ] (A^p γ γ^p)
                         (λ a -> (app B)[γ,a])
                         (λ a a^p → (B^p γ γ^p a a^p) C[γ] (C^p γ γ^p)

(El U0 El0 U+ El+ u)^p : (γ : Sub • Γ) → (γ^p : Γ^p γ)
                           → Tm • ((El u)[γ]) → Set i
(El U0 El0 U+ El+ u)^p γ γ^p t = El^p U0[γ] (U0^p γ γ^p)
                                   El0[γ] (El0^p γ γ^p)
                                   U+[γ] (U+^p γ γ^p)
                                   El+[γ] (El+^p γ γ^p)
                                   u[γ] (u^p γ γ^p) t

(maElim P el0 el+ m)^p : (γ : Sub • Γ) (γ^p : Γ^p γ)
                                  → P^p γ γ^p m[γ] (m^p γ γ^p)
(maElim P el0 el+ m)^p γ γ^p = Ma^pElim P[γ] (P^p γ γ^p)
                    el0[γ]
                    (λ u u^p → el0^p γ γ^p u u^p)
                    el+[γ]
                    (λ A A^p PA PA^p B B^p PB PB^p C C^p →
                      el+^p γ γ^p A A^p PA PA^p B B^p PB PB^p C C^p)
                    m[γ] (m^p γ γ^p)
```

3.7 Canonicity

To see how this proves canonicity, we take a look at what happens for terms in empty context. As seen below, since `Sub • •` is the empty substitution and a predicate for it is just the unit, the section for `t` tells us that it reduces to what we defined as values for the type `A`. This is precisely the statement of canonicity.

```
(t : Tm Γ A)^p : (ε : Sub • •)(ε^p : •^p •) → A^p ε ε^p (t[ε])


(t : Tm Γ A)^p ε tt : A^p ε tt t
```

4
Conclusion

Our contribution is the proof of canonicity of the external Mahlo universe. Previously most of the work on Mahlo universe was on justifying its consistency and determining its proof-theoretic strength [11, 14, 36] and building predicative semantics of Mahlo [37].

We hope that this work will increase the understanding of metatheoretic properties for strong induction principles and eventually will lead to a normalization proof for Mahlo and proofs of metatheoretic properties of induction-recursion. One current problem with these is that there is currently no presheaf model for induction-recursion and in order to prove normalization of Mahlo by gluing one would be needed.

Bibliography

[1] P. Martin-Löf, "Intuitionistic type theory: Notes by giovanni sambin of a series of lectures given in padova, june 1980," 2021.

[2] U. Norell, "Dependently typed programming in agda," in *Proceedings of the 4th international workshop on Types in language design and implementation*, 2009, pp. 1–2.

[3] M. Hofmann and M. Hofmann, "Syntax and semantics of dependent types," *Extensional Constructs in Intensional Type Theory*, pp. 13–54, 1997.

[4] J.-Y. Girard, "Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur," 1972.

[5] A. J. Hurkens, "A simplification of girard's paradox," in *Typed Lambda Calculi and Applications: Second International Conference on Typed Lambda Calculi and Applications, TLCA'95 Edinburgh, United Kingdom, April 10–12, 1995 Proceedings 2*, Springer, 1995, pp. 266–278.

[6] M. Hofmann and T. Streicher, "The groupoid interpretation of type theory," *Twenty-five years of constructive type theory (Venice, 1995)*, vol. 36, pp. 83–111, 1998.

[7] M. Hofmann, "Extensional concepts in intensional type theory," 1995.

[8] P. Dybjer, "A general formulation of simultaneous inductive-recursive definitions in type theory," *The journal of symbolic logic*, vol. 65, no. 2, pp. 525–549, 2000.

[9] A. Abel, J. Öhman, and A. Vezzosi, "Decidability of conversion for type theory in type theory," *Proceedings of the ACM on Programming Languages*, vol. 2, no. POPL, pp. 1–29, 2017.

[10] C. McBride, "Datatypes of datatypes," *Summer School on Generic and Effectful Programming, St Annes College, Oxford, Lecture Notes*, 2015.

[11] A. Setzer, "A model for a type theory with mahlo universe," *Draft. Available via http://www. math.uu.se/ setzer/articles/uppermahlo. dvi.gz*, 1996.

[12] E. Palmgren, "On universes in type theory^1," *Twenty five years of constructive type theory*, vol. 36, p. 191, 1998.

[13] D. Doel. "Various universe constructions in agda," Accessed: Nov. 30, 2023. [Online]. Available: `https://web.archive.org/web/20231130165646/https://hub.darcs.net/dolio/universes/`.

[14] A. Setzer, "Universes in type theory part i–inaccessibles and mahlo," in *Logic Colloquium*, vol. 4, 2005, pp. 123–156.

[15] P. Dybjer and A. Setzer, "A finite axiomatization of inductive-recursive definitions," in *Typed Lambda Calculi and Applications: 4th International Confer-*

ence, TLCA99 LAquila, Italy, April 7–9, 1999 Proceedings 4, Springer, 1999, pp. 129–146.

[16] K. Gödel, “Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i,” *Monatshefte für mathematik und physik*, vol. 38, pp. 173–198, 1931.

[17] J. Cartmell, “Generalised algebraic theories and contextual categories,” *Annals of pure and applied logic*, vol. 32, pp. 209–243, 1986.

[18] A. N. Whitehead, *A treatise on universal algebra: with applications.* The University Press, 1898, vol. 1.

[19] P. Dybjer, “Internal type theory,” in *International Workshop on Types for Proofs and Programs*, Springer, 1995, pp. 120–134.

[20] S. Awodey, “Natural models of homotopy type theory,” *Mathematical Structures in Computer Science*, vol. 28, no. 2, pp. 241–286, 2018.

[21] T. Uemura, “A general framework for the semantics of type theory,” *Mathematical Structures in Computer Science*, vol. 33, no. 3, pp. 134–179, 2023.

[22] A. Kaposi, S. Huber, and C. Sattler, “Gluing for type theory,” in *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, pp. 25–1.

[23] T. Altenkirch, P. Capriotti, G. Dijkstra, N. Kraus, and F. Nordvall Forsberg, “Quotient inductive-inductive types,” in *International Conference on Foundations of Software Science and Computation Structures*, Springer International Publishing Cham, 2018, pp. 293–310.

[24] A. Kaposi, A. Kovács, and T. Altenkirch, “Constructing quotient inductive-inductive types,” *Proceedings of the ACM on Programming Languages*, vol. 3, no. POPL, pp. 1–24, 2019.

[25] M. Artin, A. Grothendieck, and J. L. Verdier, *Théorie des topos et cohomologie etale des schémas: Tome 3*. Springer-Verlag, 1973.

[26] M. Carneiro, “Lean4lean: Towards a verified typechecker for lean, in lean,” *arXiv preprint arXiv:2403.14064*, 2024.

[27] A. Adjedj, M. Lennon-Bertrand, K. Maillard, P.-M. Pédrot, and L. Pujet, “Martin-löf à la coq,” in *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs*, 2024, pp. 230–245.

[28] N. A. Danielsson, “A formalisation of a dependently typed language as an inductive-recursive family,” in *International Workshop on Types for Proofs and Programs*, Springer, 2006, pp. 93–109.

[29] J. Chapman, “Type theory should eat itself,” *Electronic notes in theoretical computer science*, vol. 228, pp. 21–36, 2009.

[30] T. Altenkirch and A. Kaposi, “Type theory in type theory using quotient inductive types,” *ACM SIGPLAN Notices*, vol. 51, no. 1, pp. 18–29, 2016.

[31] P.-M. Pédrot, “Russian constructivism in a prefascist theory,” in *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, 2020, pp. 782–794.

[32] K. Ambrus and P. Loïc, “Type theory in type teory using a strictified syntax,” 2025, Preprint at `https://web.archive.org/web/20250317004303/https://pujet.fr/pdf/strictification_preprint.pdf`. Accessed: Mar. 17, 2025.

[33] A. Kaposi, A. Kovács, and N. Kraus, "Shallow embedding of type theory is morally correct," in *International Conference on Mathematics of Program Construction*, Springer, 2019, pp. 329–365.

[34] W. W. Tait, "Intensional interpretations of functionals of finite type i," *The journal of symbolic logic*, vol. 32, no. 2, pp. 198–212, 1967.

[35] P. Dybjer and A. Setzer, "Indexed induction–recursion," *The Journal of Logic and Algebraic Programming*, vol. 66, no. 1, pp. 1–49, 2006.

[36] A. Setzer, "Extending martin-löf type theory by one mahlo-universe," *Archive for Mathematical Logic*, vol. 39, no. 3, pp. 155–181, 2000.

[37] P. Dybjer and A. Setzer, "The extended predicative mahlo universe in martin-löf type theory," *Journal of Logic and Computation*, exad022, 2023.

A

Equations for the Canonicity model

A.0.1 `El` equations for the Mahlo canonicity model

Here we prove that the El predicate for the Mahlo universe respects the equations of El of Mahlo in syntax.

```
ElEl0^p : (γ : Sub • Γ) (γ^p : Γ^p γ) → (u : Tm Γ U0)
            → (el0 : Tm Γ (El0 $ u)) →
  (El El0 U+ El+ $$ (El0' u))^p γ γ^p el0[γ]
  ≡
  (El0 u)^p γ γ^p el0[γ]

  El^p ... El0[γ] (El0^p γ γ^p) ... (El0' u[γ]) ((El0' u)^p γ γ^p) el0[γ]
  ≡
  (El0 u)^p γ γ^p el0[γ]

  El0^p γ γ^p u[γ] (u^p γ γ^p) el0[γ]
  ≡
  El0^p γ γ^p u[γ] (u^p γ γ^p) el0[γ]

ElEl+^p : (γ : Sub • Γ) (γ^p : Γ^p γ) → (A : Tm • Ma)
          → (B : Tm • El A → Ma)
          → (C : Tm • (U+ $$ (El A) $$
                                (lam (El[p] $$ (B[p] $$ q)))))
          → (el+ : Tm Γ (El {U+} {El+} (El+ $$ C))) →
  (El {U+} {El+} $$ (El+ A B C))^p  γ γ^p el+[γ]
  ≡
  (El+ (El $$ A) (lam (El[p] $$ B[p] $$ q)) (El C))^p γ γ^p el+[γ]

  El+^p (El $$ A[γ]) (El^p A[γ] (A^p γ γ^p))
     ((lam (El[p] $$ (B[p] $$ q)))[γ])
     (λ a a^p → El^p ((B $$ a)[γ]) (B^p γ γ^p a a^p))
     C[γ] (C^p γ γ^p) el+[γ]
  ≡
  El+^p (El $$ A[γ]) (El^p A[γ] (A^p γ γ^p))
```

```
                  ((lam (El[p] $$ (B[p] $$ q)))[γ])
      (λ a a^p → El^p ((app B)[γ,a]) (B^p γ γ^p a a^p))
      C[γ] (C^p γ γ^p) el+[γ]
```

A.0.2 `maElim` equations for the Mahlo canonicity model

Checking that the β rules for eliminator of `Ma` are respected by the canonicity model.

```
El0maElim^p : (γ : Sub • Γ) (γ^p : Γ^p γ) →
        {j : MetaNat}(P : Tm Γ (Ma → (U j))
        → (el0 : Tm Γ (U0 → P $$ (El0' $$ q)))
        → (el+ : Tm Γ (Ma → P $$ q → (El $$ q^1 → Ma)
          → (lam (P[p] $$ (q^1 $$ q)))
          → U+ $$ (El $$ q^3) $$ lam (El[p] $$ (q^2 $$ q))
          → P $$ (El+' $$ q^4 $$ q^2 $$ q))) →
        (x : Tm Γ U0) →
  (maElim P el0 el+ (El0' x))^p γ γ^p
  ≡
  (el0 x)^p γ γ^p

  Ma^pElim P[γ] (P^p γ γ^p)
      el0[γ] (λ u u^p → el0^p γ γ^p u u^p) el+[γ]
      (λ A A^p PA PA^p B B^p PB PB^p C C^p →
        el+^p γ γ^p A A^p PA PA^p B B^p PB PB^p C C^p)
      (El0' x[γ]) (El0'^p x[γ] (x^p γ γ^p))
  ≡
  el0^p γ γ^p x[γ] (x^p γ γ^p)

  (λ u u^p → el0^p γ γ^p u u^p) x[γ] (x^p γ γ^p)
  ≡
  el0^p γ γ^p x[γ] (x^p γ γ^p)

  el0^p γ γ^p x[γ] (x^p γ γ^p)
  ≡
  el0^p γ γ^p x[γ] (x^p γ γ^p)

El+maElim^p : (γ : Sub • Γ) (γ^p : Γ^p γ) →
        {j : MetaNat}(P : Tm Γ (Ma → (U j))
        → (el0 : Tm Γ (U0 → P $$ (El0' $$ q)))
        → (el+ : Tm Γ (Ma → P $$ q → (El $$ q^1 → Ma)
          → (lam (P[p] $$ (q^1 $$ q)))
          → U+ $$ (El $$ q^3) $$ (lam (El[p] $$ (q^2 $$ q)))
          → P $$ (El+'  → (A^p : Ma^p A)$$ q^4 $$ q^2 $$ q)))
          → (a : Tm • Ma) → (b : Tm • (El $$ A → Ma))
          → (x : Tm • (U+ $$ (El $$ A) $$
                                (lam (El[p] $$ (B[p] $$ q)))))
  (maElim P el0 el+ (El+' a b x))^p γ γ^p
```

```
≡
(el+ $$ a $$
  (maElim P el0 el+ a) $$ b $$
  (lam (maElim P[p] el0[p] el+[p] (b[p] $$ q)))
  $$ x)^p γ γ^p

Ma^pElim P[γ] (P^p γ γ^p)
    el0[γ] (λ u u^p → el0^p γ γ^p u u^p) el+[γ]
    (λ A A^p PA PA^p B B^p PB PB^p C C^p →
      el+^p γ γ^p A A^p PA PA^p B B^p PB PB^p C C^p)
    (El+' a[γ] b[γ] x[γ])
    (El+'^p a[γ] (a^p γ γ^p)
      b[γ]
      (λ A A^p → b^p γ γ^p A A^p) x[γ] (x^p γ γ^p))
≡
el+^p γ γ^p a[γ] (a^p γ γ^p)
  (maElim P el0 el+ a)[γ]
  ((maElim P el0 el+ a)^p γ γ^p)
  b[γ] (λ A A^p → b^p γ γ^p A A^p)
  (λ A → maElim P[γ] el0[γ] el+[γ] (b[γ] $$ A))
  (λ A A^p → (maElim P el0 el+ (b $$ a))^p γ γ^p a a^p)
  x[γ] (x γ γ^p)

(λ A A^p PA PA^p B B^p PB PB^p C C^p →
      el+^p γ γ^p A A^p PA PA^p B B^p PB PB^p C C^p)
     a[γ] (a^p γ γ^p) (maElim P[γ] el0[γ] el+[γ] a[γ])
    (Ma^pElim P[γ] (P^p γ γ^p) el0[γ] (λ u u^p → el0^p γ γ^p u u^p) el+[γ]
      (λ A A^p PA PA^p B B^p PB PB^p C C^p →
        el+^p γ γ^p A A^p PA PA^p B B^p PB PB^p C C^p) a[γ] (a^p γ γ^p))
    b[γ] (λ A A^p → b^p γ γ^p A A^p)
    (λ A → maElim P[γ] el0[γ] el+[γ] (B[γ] $$ A))
    (λ A A^p →
    (Ma^pElim P[γ] (P^p γ γ^p) el0[γ] (λ u u^p → el0^p γ γ^p u u^p) el+[γ]
      (λ A A^p PA PA^p B B^p PB PB^p C C^p →
        el+^p γ γ^p A A^p PA PA^p B B^p PB PB^p C C^p)
        ((app b[γ]) $$ A) (b^p γ γ^p A A^p)))
    x[γ] (x γ γ^p)
≡
el+^p γ γ^p a[γ] (a^p γ γ^p)
  (maElim P el0 el+ a)[γ]
  ((maElim P el0 el+ a)^p γ γ^p)
  b[γ] (λ A A^p → b^p γ γ^p A A^p)
  (λ A → maElim P[γ] el0[γ] el+[γ] (b[γ] $$ A))
  (λ A A^p → (maElim P el0 el+ (b $$ a))^p γ γ^p a a^p)
  x[γ] (x γ γ^p)
```

```
 el+^p γ γ^p a[γ] (a^p γ γ^p) (maElim P[γ] el0[γ] el+[γ] a[γ])
     (Ma^pElim P[γ] (P^p γ γ^p) el0[γ] (λ u u^p → el0^p γ γ^p u u^p) el+[γ]
       (λ A A^p PA PA^p B B^p PB PB^p C C^p →
         el+^p γ γ^p A A^p PA PA^p B B^p PB PB^p C C^p) a[γ] (a^p γ γ^p))
     b[γ] (λ A A^p → b^p γ γ^p A A^p)
     (λ A → maElim P[γ] el0[γ] el+[γ] (B[γ] $$ A))
     (λ A A^p →
     (Ma^pElim P[γ] (P^p γ γ^p) el0[γ] (λ u u^p → el0^p γ γ^p u u^p) el+[γ]
       (λ A A^p PA PA^p B B^p PB PB^p C C^p →
         el+^p γ γ^p A A^p PA PA^p B B^p PB PB^p C C^p)
         ((app b[γ]) $$ A) (b^p γ γ^p A A^p)))
     x[γ] (x γ γ^p)
 ≡
 el+^p γ γ^p a[γ] (a^p γ γ^p) (maElim P[γ] el0[γ] el+[γ] a[γ])
     (Ma^pElim P[γ] (P^p γ γ^p) el0[γ] (λ u u^p → el0^p γ γ^p u u^p)
       el+[γ] (λ A A^p PA PA^p B B^p PB PB^p C C^p →
                    el+^p γ γ^p A A^p PA PA^p B B^p PB PB^p C C^p)
       a[γ] (a^p γ γ^p))
     b[γ] (λ A A^p → b^p γ γ^p A A^p)
     (λ A → maElim P[γ] el0[γ] el+[γ] (B[γ] $$ A))
     (λ A A^p →
       (Ma^pElim P[γ] (P^p γ γ^p) el0[γ] (λ u u^p → el0^p γ γ^p u u^p)
       el+[γ] (λ A A^p PA PA^p B B^p PB PB^p C C^p →
                  el+^p γ γ^p A A^p PA PA^p B B^p PB PB^p C C^p)
         ((app b[γ]) $$ A) (b^p γ γ^p A A^p)))
     x[γ] (x γ γ^p)
```

A.0.3 Substitution equations for the Mahlo canonicity model

Proof that the predicates for Mahlo respect substitution.

```
subEl0^p : (γ : Sub • Γ) (γ^p : Γ^p γ) → (δ : Sub Γ Δ)
                                        → (u : Tm Δ U0) →
 ((El0' u)[δ])^p γ γ^p
 ≡
 (El0' u[δ])^p γ γ^p

 (El0' u)^p (δ ∘ γ) (δ^p γ γ^p)
 ≡
 (El0' u[δ])^p γ γ^p

 El0'^p u[δ ∘ γ] (u^p (δ ∘ γ) (δ^p γ γ^p))
 ≡
 El0'^p u[δ][γ] (u[δ]^p γ γ^p)
 El0'^p u[δ ∘ γ] (u^p (δ ∘ γ) (δ^p γ γ^p))
 ≡
 El0'^p u[δ ∘ γ] (u^p (δ ∘ γ) (δ^p γ γ^p))

subEl+^p : (γ : Sub • Γ) (γ^p : Γ^p γ) → (δ : Sub Γ Δ)
        → (A : Tm Δ Ma)
        → (B : Tm Δ (El $$ A) → Ma)
        → (C : Tm • (U+ $$ (El $$ A)
                    $$ (lam (El[p] $$ (B[p] $$ q))))) →
  ((El+' $$ A $$ B $$ C)[δ])^p γ γ^p
  ≡
  (El+' $$ A[δ] $$ B[δ] C[δ])^p γ γ^p

  (El+' $$ A $$ B $$ C)^p (δ ∘ γ) (δ^p γ γ^p)
  ≡
  (El+' $$ A[δ] $$ B[δ] C[δ])^p γ γ^p

  El+'^p A[δ ∘ γ] (A^p (δ ∘ γ) (δ^p γ γ^p)) B[δ ∘ γ]
    (λ a a^p → (B^p (δ ∘ γ) (δ^p γ γ^p) a a^p)
    C[δ ∘ γ] (C^p (δ ∘ γ) (δ^p γ γ^p))
  ≡
  El+'^p A[δ][γ] (A[δ]^p γ γ^p) B[δ][γ]
      (λ a a^p → (B[δ]^p γ γ^p a a^p) C[δ][γ] (C[δ]^p γ γ^p)

  El+'^p A[δ ∘ γ] (A^p (δ ∘ γ) (δ^p γ γ^p)) B[δ ∘ γ]
    (λ a a^p → (B^p (δ ∘ γ) (δ^p γ γ^p) a a^p)
    C[δ ∘ γ] (C^p (δ ∘ γ) (δ^p γ γ^p))
  ≡
  El+'^p A[δ ∘ γ] (A^p (δ ∘ γ) (δ^p γ γ^p)) B[δ ∘ γ]
      (λ a a^p → (B^p (δ ∘ γ) (δ^p γ γ^p) a a^p) C[δ ∘ γ]
      (C^p (δ ∘ γ) (δ^p γ γ^p))

submaElim^p : (γ : Sub • Γ) (γ^p : Γ^p γ) → (δ : Sub Γ Δ) →
        {j : MetaNat}(P : Tm Γ (Ma → (U j))
        → (el0 : Tm Γ (U0 → P $$ (El0' $$ q)))
        → (el+ : Tm Γ (Ma → P $$ q → (El $$ q^1 → Ma)
          → (lam (P[p] $$ (q^1 $$ q)))
          → U+ $$ (El $$ q^3) $$ lam (El[p] $$ (q^2 $$ q))
          → P $$ (El+' $$ q^4 $$ q^2 $$ q))) →
        (t : Tm Γ Ma) →
 ((maElim P el0 el+ t)[δ])^p γ γ^p
 ≡
 (maElim P[δ] el0[δ] el+[δ] t[δ])^p γ γ^p

 (maElim P el0 el+ t)^p (δ ∘ γ) (δ^p γ γ^p)
 ≡
```

```
(maElim P[δ] el0[δ] el+[δ] t[δ])^p γ γ^p

Ma^pElim P[δ ∘ γ] (P^p (δ ∘ γ) (δ^p γ γ^p))
        el0[δ ∘ γ] (λ u u^p → el0^p (δ ∘ γ) (δ^p γ γ^p) u u^p)
        el+[δ ∘ γ] (λ A A^p PA PA^p B B^p PB PB^p C C^p →
        el+^p (δ ∘ γ) (δ^p γ γ^p) A A^p PA PA^p B B^p PB PB^p C C^p)
        m[δ ∘ γ] (m^p (δ ∘ γ) (δ^p γ γ^p))
≡
Ma^pElim P[δ][γ] ((P[δ])^p γ γ^p)
        el0[δ][γ] (λ u u^p → (el0[δ])^p γ γ^p u u^p)
        el+[δ][γ]
        (λ A A^p PA PA^p B B^p PB PB^p C C^p →
        (el+[δ])^p γ γ^p A A^p PA PA^p B B^p PB PB^p C C^p)
        m[δ][γ] ((m[δ])^p γ γ^p)

Ma^pElim P[δ ∘ γ] (P^p (δ ∘ γ) (δ^p γ γ^p))
        el0[δ ∘ γ] (λ u u^p → el0^p (δ ∘ γ) (δ^p γ γ^p) u u^p)
        el+[δ ∘ γ] (λ A A^p PA PA^p B B^p PB PB^p C C^p →
        el+^p (δ ∘ γ) (δ^p γ γ^p) A A^p PA PA^p B B^p PB PB^p C C^p)
        m[δ ∘ γ] (m^p (δ ∘ γ) (δ^p γ γ^p))
≡
Ma^pElim P[δ ∘ γ] (P^p (δ ∘ γ) (δ^p γ γ^p))
        el0[δ ∘ γ] (λ u u^p → el0^p (δ ∘ γ) (δ^p γ γ^p) u u^p)
        el+[δ ∘ γ] (λ A A^p PA PA^p B B^p PB PB^p C C^p →
        el+^p (δ ∘ γ) (δ^p γ γ^p) A A^p PA PA^p B B^p PB PB^p C C^p)
        m[δ ∘ γ] (m^p (δ ∘ γ) (δ^p γ γ^p))
```