Abstract

The setoid model is a model of intensional type theory that validates extensionality principles such as function extensionality and proposition extensionality, and perhaps most importantly, that can be used to extend intensional type theory with quotient types. However, the manipulation of said quotients is rendered difficult by the the lack of a principle called unique choice in existing accounts of the setoid model. In this work, we propose a new construction for a universe of setoids which does not require the equalities to be strict propositions, and we use it to obtain two models for the Calculus of Inductive Constructions: a predicative setoid model which validates the principle of unique choice, and an impredicative setoid model which validates the axiom of choice for decidable relations with a countable codomain. Both models can be formulated in the Calculus of Inductive Constructions without axioms, and they preserve all of its computation rules – except for the η-rule for functions, which is weakened to a propositional equality.

2012 ACM Subject Classification Theory of computation $\rightarrow$ Type theory

Keywords and phrases type theory, quotient types, unique choice, setoid model, impredicativity, induction-recursion

Digital Object Identifier 10.4230/LIPIcs...

Weak Choice Principles for Setoids

Loïc Pujet
Stockholm University

1 Introduction

Writing formal proofs in dependent type theory is a delicate art, even for the most seasoned users of interactive proof assistants: the challenge often lies not so much in working out the proof itself, but in finding the right definition to build upon. This is especially true in *intensional* type theory (ITT) [16], where the equality judgment only ever identifies definitions that are $\beta\eta$-convertible. Now, this whole intensionality business may seem unreasonably restrictive compared to the extensional nature of informal mathematics, but it is in fact a necessary condition for the decidability of the typing relation, which is important for both foundational and practical reasons.

But while the intensional nature of the equality judgment serves an important purpose, the same cannot be said of the *propositional* equality. And yet, in ITT the latter is usually defined as an inductive family that inherits the coarseness of the equality judgment, resulting in a notion of equality that is quite difficult to work with. As a result users commonly postulate extensionality axioms for the propositional equality, such as the principle of function extensionality, which states that two functions are equal if their images are equal, the principle of proposition extensionality, which states that logically equivalent propositions are equal or the principle of uniqueness of identity proofs (UIP). Unfortunately, axioms break some computational properties of the system: adding function extensionality will result in closed terms of type $\mathbb{N}$ that do not evaluate to an actual integer.

An alternative strategy is to work with types that are equipped with an equivalence relation, which are known as *setoids*. Doing so allows the user to specify the meaning of equality for each type, but it comes at the cost of having to prove manually that every function preserves the setoid equality. This should be the work of a compiler, not of the user! What we really want is a way to reason in type theory with extensionality principles, and then automatically elaborate our reasoning into plain intensional type theory by inserting setoid reasoning when necessary. This is precisely what Hofmann achieved with his setoid model [15], which interprets all types as setoids and validates the principles of function extensionality and proposition extensionality, and allows the definition of quotient types. Unfortunately, Hofmann's setoid model only handles a fragment of type theory. In particular it does not support universes, which makes it impossible to even prove that $0 \neq 1$.

Hofmann's pioneering work subsequently inspired many developments around setoids and their internal type theories [1, 3, 2, 18, 17]. A particularly noteworthy innovation was introduced by Altenkirch [1], who define the setoid equalities as *strict* propositions, *i.e.,* propositions whose inhabitants are all judgmentally equal. This strong version of UIP greatly simplifies the construction of the setoid model, which allowed Altenkirch, Boulier, Kaposi, Sattler and Sestini to equip it with a proper universe of setoids [2].

In counterpart, the use of strict propositions seriously weakens the quotient types that are provided by the model: now, if the user wants to form the quotient of a type by a proof-relevant relation, they must first use a truncation operation to turn it into a proposition. But propositional truncation is a lossy operation, and in general, it is not possible to recover an inhabitant of the original type from a proof of its truncated version without invoking some form of the axiom of choice. Still, depending on the exact nature of the truncation operation, it may be possible to invert it in some special cases. In this paper, our aim is to provide a setoid model with such "weak choice principles", to make quotient types more usable.

Weak Choice Principles

In particular, we are interested in two choice principles that may be stated in any type theory that supports a sort of propositions `Prop` and a notion of propositional truncation $\|\_\|$. The first principle is a strong form of *unique choice*, which states that we may construct a function from any relation `R : A → B → Prop` which is propositionally a functional relation:

```
∀ (a : A), ‖ Σ (b : B), (R a b) × (∀ c, R a c → c = b) ‖
  → Σ (f : A → B), ∀ (a : A), R a (f a).
```

In dependent type theory, this statement happens to be equivalent to the simpler principle of truncation elimination for “homotopy propositions”, *i.e.*, for types whose inhabitants are all propositionally equal.

```
isHProp A → ‖ A ‖ → A                                         (1)
```

Our second principle is the axiom of choice for any *decidable* relation `R : A → ℕ → Prop`:

```
∀ a b, (R a b) + ¬ (R a b)
  → ∀ (a : A), ‖ Σ (b : ℕ), R a b ‖ → Σ (f : A → ℕ), ∀ (a : A), R a (f a).
```

This is equivalent to truncation elimination for Σ^0_1 types, *i.e.*, types that are equivalent to `Σ (n : ℕ). P n` for some decidable predicate `P`.

```
isSigma01 A → ‖ A ‖ → A                                       (2)
```

Principle (2) is a consequence of unique choice (as we may uniquely specify the solution by taking the smallest one), but unlike unique choice, it is actually provable in the type theory of Coq, as a consequence of the large elimination rule for the accessibility predicate:

```
Inductive Acc {A : Type} {R : A → A → Prop} (a : A) : Prop :=
| acc : (∀ (b : A), R b a → Acc b) → Acc a.
```

This large elimination rule is not available in Coq if we replace `Prop` with the sort of strict propositions `SProp`, or otherwise the typechecking would become undecidable [12]. As a consequence, neither unique choice nor large elimination for the accessibity predicate is supported by the setoid models which are based on strict propositions.

Contributions

We construct a new universe of setoids which does not require the equalities to be strict propositions. We then use our universe of setoids to construct two extensions of Hofmann's setoid model: an *impredicative* model which uses `Prop`-valued equalities and is formulated in CIC, and a *predicative* model which uses `Type`-valued equalities and can be formulated in the “predicative CIC”, *i.e.*, without using the impredicative features of the theory. The predicative model supports universes, Π-types, Σ-types, inductive types and quotient types, along with with extensionality principles and the principle of unique choice. The impredicative model adds a sort of propositions with impredicative quantification and large elimination of the accessibility predicate, but it does not validate the principle of unique choice. Both models preserve all the computation rules of CIC, except for the η-rule for function types which is relegated to a propositional equality. Because our models are constructed in plain intensional type theory, they show that extensionality principles, quotients and weak choice principles do not add any consistency strength to intensional type theory.

The construction of the impredicative model has been fully formalised in Coq, and links to the relevant parts of the formalisation will be provided throughout the text. The complete proof is available at the following (anonymised) URL: `https://anonymous.4open.science/r/supplementary_material-12B1/supplementary_material.v`

1.1 Related work

The setoid model was originally conceived by Hofmann in order to extend intensional type theory with extensionality principles and quotient types [14]. However, this first iteration did not support universes or large elimination. Building on this work, Altenkirch used a type theory extended with a sort of definitionally proof-irrelevant propositions to give a stricter version of the setoid model, and equipped it with a universe of simple types [1]. This already provides a limited form of large elimination, and in particular this is enough to prove that $0 \neq 1$. Later down the line, Altenkirch *et al.* studied several methods to construct a universe that supports dependent type formers, using further extensions of the ambient type theory such as induction-recursion, induction-induction or a strengthened J eliminator for the equality [2]. Compared to their work, our universe only needs plain indexed inductive types, and we do not require our sort of proposition to be strict, which unfortunately weakens the η-rule to a propositional equality.

Meanwhile, Altenkirch, McBride and Swierstra explored a different way to mix extensionality and intensionality with Observational Type Theory (OTT), an intensional type theory whose propositional equality is defined on a type by type basis [3]. This strategy allows them to implement quotient types and extensionality principles, while preserving the good computational properties of their system. Subsequently, the community explored several points in the design space of observational type theories: Sterling, Angiuli and Gratzer's XTT [18] uses a cubical syntax to implement extensionality principles, and implements universe induction, but lacks unique choice. Pujet and Tabareau's $\mathsf{CIC}^{\mathrm{obs}}$ adds an impredicative sort of propositions and a scheme of inductive definitions to bring their system closer to CIC, but they are not able to implement large elimination of the propositional accessibility predicate [17]. Our work fills these gaps – we are able to handle unique choice in our predicative model, and large elimination of accessibility in our impredicative model.

Cubical Type Theories [7, 19, 4] provide yet another way to compute with extensionality principles, trading UIP for the more sophisticated principle of univalence. Additionally, they support higher inductive types [6], which can be used to obtain quotient types with unique choice. However, defining an impredicative version of Cubical Type Theory that preserves all of its computational properties is still an open problem.

2 Preliminary Definitions

Our model construction takes place in an idealised version of the type theory behind the Coq proof assistant, the Calculus of Inductive Constructions [8]. Accordingly, we will present our model using the syntax of Coq.

Our version of the CIC is equipped with two sorts: the universe hierarchy $\{\texttt{Type}_i\}_{i\in\mathbb{N}}$, where $\texttt{Type}_i : \texttt{Type}_{i+1}$, and the sort of propositions `Prop`, which has type $\texttt{Type}_0$. We will also need dependent products, which we write $\forall$ `(x : A). B`, or `A` $\to$ `B` when `B` does not depend on `A`. As usual, the typing rule for dependent products is split into two cases: on the one hand, if the codomain `B` is in $\texttt{Type}_i$, then the dependent product is in a universe that is larger or equal than the universes of `A` and `B` (taking the convention that `Prop` is smaller than $\texttt{Type}_i$ for all i). On the other hand, if `B` is in `Prop`, then the dependent product is also in `Prop` regardless of the type of `A`. In other words, the sort of propositions `Prop` is *impredicative*. We also ask for primitive Σ-types with η-expansion, which we write Σ (`x : A`). `B`, or `A` × `B` when `B` does not depend on `A`. This datatype can be implemented in Coq using a record type with primitive projections.

Finally, we allow the definition of indexed inductive types in both $\texttt{Type}_i$ and `Prop`, using a definition scheme that is similar to the one implemented in Coq [9]. Large elimination, the possibility to define arbitrarily large objects by induction, is permitted for inductive types in $\texttt{Type}_i$, and for inductive types in `Prop` that satisfy the subsingleton criterion[1]. Examples of indexed inductive types include the type of booleans `Bool`, the type of natural numbers ℕ, the inductive identity type `Id`, the type of well-founded trees `W` (`x : A`). `B`, and the accessibility predicate `Acc`. They can also be used to define universe lifting operators, which we may silently insert to simulate cumulativity.

Predicative CIC

As mentioned in the introduction, we are actually going to construct two different models: an impredicative model which puts the setoid equalities in `Prop`, and a predicative model which uses the `Type` hierarchy instead. One important point about the predicative model is that it can be constructed in CIC without relying on impredicativity at all, *i.e.,* without using the impredicative sort `Prop`. The resulting theory, which we call preCIC (short for predicative CIC), is roughly equivalent to Martin-Löf's intensional type theory [16] extended with Coq's scheme for inductive definitions and the η-rule for Σ-types.

Since the constructions of our two models will be very similar, we want to factor out the redundant parts. To this end, we introduce a placeholder sort $\texttt{Sort}_i$, which means $\texttt{Type}_i$ for the predicative model, and `Prop` for the impredicative model. We will be careful to only use $\texttt{Sort}_i$ in places where both the typing rules for $\texttt{Type}_i$ and the typing rules for `Prop` apply, and we will revert to explicit sorts when the difference starts to matter.

2.1 Setoids

A setoid is a type that is equipped with an equivalence relation. This relation, the setoid equality, should be valued in `Prop` for the impredicative model and in `Type` for the predicative model. Thus, a setoid of universe level i may be summarised by the following record type:

```
Record Setoidᵢ : Typeᵢ₊₁ := {
 A      :  Typeᵢ ;
 eq     :  A → A → Sortᵢ ;
 refl   :  ∀ a, eq a a ;
 sym    :  ∀ a b, eq a b → eq b a ;
 trans  :  ∀ a b c, eq a b → eq b c → eq a c
}.
```

Then, setoid morphisms are defined as functions that preserve the setoid equality:

```
Record SetoidFunᵢ (X Y : Setoidᵢ) : Typeᵢ := {
 f     :  X.A → Y.A ;
 f_eq  :  ∀ a b, X.eq a b → Y.eq (f a) (f b)
}.
```

[1] An inductive type is a *subsingleton* if it has at most one constructor, and all the types of the constructor arguments are propositions.

Remark that we do not ask for preservation of reflexivity, symmetry or transitivity. Finally, if our goal is to interpret dependent type theory, we need a notion of indexed setoid family:

```
Record SetoidFam_{i,j} (X : Setoid_i) : Type_max(i,j) := {
 P       :  X.A → Type_j ;
 heq     :  ∀ a b, P a → P b → Sort_j ;
 hrefl   :  ∀ a p, heq a a p p ;
 hsym    :  ∀ a b p q, heq a b p q → heq b a q p ;
 htrans  :  ∀ a b c p q r, X.eq a b → heq a b p q → heq b c q r → heq a c p r ;
 coe     :  ∀ a b, X.eq a b → P a → P b ;
 coh     :  ∀ a b p e, heq a p b (coe a b e p)
}.
```

Our setoid families are equipped with a heterogeneous equality modeled after the observational equality of Altenkirch, McBride and Swierstra [3], and a coercion operator which preserves the heterogeneous equality.

Hofmann and his successors directly use the category of setoids as a model of type theory, by giving it the structure of a *category with families* (CwF, [11]). They explain how to interpret contexts as arbitrary setoids, substitutions as setoid morphisms, types as dependent setoids, *etc.*, and after that they discuss the addition of a universe. In this paper, we do it the other way around: we start by constructing a universal setoid that is closed under all the basic type formers of dependent type theory, and then we use it to define a model.

3 The Universal Setoid

3.1 Inductive-Recursive Recipe

How do we construct our universal setoid? The most natural approach would be to equip the type of all small setoids with a setoid equality, but there does not seem to exist any good candidate. In particular, we cannot use arbitrary isomorphisms as equalities, because the type of isomorphisms between two setoids is not a proposition, and truncating it would destroy too much information to get a usable notion of equality.

Instead, we follow the approach of Altenkirch *et al.* [2] and define our universe as the smallest setoid that is closed under all the type formers of our theory. In other words, we are building a setoid of *codes*, which contains a code for the setoid of natural numbers, codes for dependent products of setoids (given a code for the domain and a function from the domain into the codes for the codomain), *etc.* The setoid equality between codes is recursively generated: for instance, the codes for `A` → `B` and `C` → `D` are equal if and only if the codes for `A` and `C` are equal and the the codes for `B` and `D` are equal.

The easiest way to define this structure is *induction-recursion* [10], which allows us to simultaneously define the carrier of the universal setoid as an inductive type $\texttt{U}_0$ along with three recursive functions $\texttt{eqU}_0$, $\texttt{El}_0$ and $\texttt{eq}_0$ that respectively represent the setoid equality on the universe, the universal family of small setoids and its heterogeneous equality (Figure 1). Unfortunately our metatheory does not support induction-recursion, so we need to find a way to encode this definition with plain indexed inductive types. The canonical method is Hancock *et al.*'s small induction-recursion [13], but it does not apply to our definition: not only do the recursive functions $\texttt{eqU}_0$ and $\texttt{eq}_0$ have two arguments of type $\texttt{U}_0$ instead of one, but the return type of $\texttt{eqU}_0$ is not even small for the predicative model.

```
(* Underlying type of the universal setoid *)
Inductive U₀ : Type₁ :=
| cℕ : U₀
| cΣ : ∀ (A : U₀) (P : El₀ A → U₀) (Pe : ∀ a a', eq₀ A A a a' → eqU₀ (P a) (P a')), U₀
| cΠ : ∀ (A : U₀) (P : El₀ A → U₀) (Pe : ∀ a a', eq₀ A A a a' → eqU₀ (P a) (P a')), U₀.

(* Setoid equality on U₀ *)
Fixpoint eqU₀ (A B : U₀) : Sort₁ :=
 match A, B with
 | cℕ, cℕ ⇒ True
 | cΣ A P Pe, cΣ B Q Qe ⇒ (eqU₀ A B) × (∀ a b, eq₀ A B a b → eqU₀ (P a) (Q b))
 | cΠ A P Pe, cΠ B Q Qe ⇒ (eqU₀ A B) × (∀ a b, eq₀ A B a b → eqU₀ (P a) (Q b))
 | _, _ ⇒ False
 end.

(* Universal family of setoids over U₀ *)
Fixpoint El₀ (A : U₀) : Type₀ :=
 match A with
 | cℕ ⇒ ℕ
 | cΣ A P Pe ⇒ Σ (a : El₀ A), El₀ (P a)
 | cΠ A P Pe ⇒ Σ (f : ∀ (a : El₀ A), El₀ (P a))
               , (∀ a a', eq₀ A A a a' → eq₀ (P a) (P a') (f a) (f a'))
 end.

(* Heterogeneous setoid equality on the family *)
Fixpoint eq₀ (A B : U₀) (x : El₀ A) (y : El₀ B) : Sort₀ :=
 match A, B with
 | cℕ, cℕ ⇒ ℕeq x y  (* inductively defined equality on ℕ *)
 | cΣ A P Pe, cΣ B Q Qe ⇒ (eq₀ A B x.1 y.1) × (eq₀ (P x.1) (Q y.1) x.2 y.2)
 | cΠ A P Pe, cΠ B Q Qe ⇒ ∀ a b, eq₀ A B a b → eq₀ (P a) (Q b) (x a) (y b)
 | _, _ ⇒ False
 end.
```

Figure 1 Inductive-recursive definition of a universal setoid with three type formers.

3.2 A Plan in Two Steps

To get around this issue, we start by defining an overapproximation $\texttt{preU}_0$ for our setoid universe, with a constructor for dependent products that is parametrised by arbitrary equality relations on `A` and `P` and does not enforce `P` to be a setoid morphism (Figure 2). This way, the definition of $\texttt{preU}_0$ is not mutual with the definition of $\texttt{eq}_0$ and $\texttt{eqU}_0$ anymore, and it fits in the framework of small induction-recursion. It may thus be expressed with an ordinary inductive type, as we did in the Coq definition. As an aside, remark that there is no need to parametrise the constructor for dependent sums in a similar way, because dependent sums of setoids do not involve any sort of equality preservation condition. Now, we define the equality relations $\texttt{eq}_0$ and $\texttt{eqU}_0$ on the overapproximated universe, using the same definitions from Figure 1. Thanks to our little maneuver, these functions are not mutually defined with $\texttt{preU}_0$, and thus they can be given a simple recursive definition.

```
Inductive preU₀ : Type₁ :=
| preℕ : preU₀
| preΣ : ∀ (A : preU₀) (P : El₀ A → preU₀), preU₀
| preΠ : ∀ (A : preU₀) (Aeq : El₀ A → El₀ A → Sort₀)
           (P : El₀ A → preU₀) (Peq : ∀ a₀ a1, El₀ (P a₀) → El₀ (P a₁) → Sort₀), preU₀.

Fixpoint El₀ (A : preU₀) : Type₀ :=
  match A with
  | preℕ ⇒ ℕ
  | preΣ A P ⇒ Σ (a : El₀ A), El₀ (P a)
  | preΠ A Aeq P Peq ⇒ Σ (f : ∀ (a : El₀ A), El₀ (P a))
                          , (∀ a a', Aeq a a' → Peq (f a) (f a'))
  end.
```

Figure 2 Small inductive-recursive definition of an overapproximated universe.

```
Inductive extU₀ : preU₀ → Type₁ :=
| extℕ : extU₀ cℕ
| extΣ : ∀ (A : preU₀) (Ae : extU₀ A)
           (P : El₀ A → preU₀) (Pe : ∀ a, extU₀ (P a))
           (Pext : ∀ a₀ a₁, eq₀ A A a₀ a₁ → eqU₀ (P a₀) (P a₁))
         , extU₀ preΣ A P.
| extΠ : ∀ (A : preU₀) (Ae : extU₀ A)
           (P : El₀ A → preU₀) (Pe : ∀ a, extU₀ (P a))
           (Pext : ∀ a₀ a₁, eq₀ A A a₀ a₁ → eqU₀ (P a₀) (P a₁))
         , extU₀ (preΠ A (eq₀ A A) P (fun a₀ a₁ ⇒ eq₀ (P a₀) (P a₁))).

Definition U₀ := Σ (A : preU₀), extU₀ A.
```

Figure 3 Inductive predicate that carves out the well-formed elements of $\texttt{preU}_0$.

Next, we define an inductive predicate $\texttt{extU}_0$ that carves out the codes from $\texttt{preU}_0$ which have a counterpart in the inductive-recursive definition from Figure 1. More specifically, it ensures that that the dependent codomains that appear in $\texttt{pre}_\Pi$ and $\texttt{pre}_\Sigma$ are proper setoid morphisms from the domain into the universe, and that the codes for dependent products have been parametrised with the equality relations defined by $\texttt{eq}_0$. The definition of $\texttt{extU}_0$ is given in Figure 3. Finally, we can put everything together: the carrier type of our universal setoid is defined as $\texttt{U}_0 := \Sigma\ (\texttt{A} : \texttt{preU}_0).\ \texttt{extU}_0\ \texttt{A}$, its setoid equality is given by $\texttt{eqU}_0$ on the first component, the universal dependent family on the universe is provided by $\texttt{El}_0$, and the heterogeneous equality on that family is given by $\texttt{eq}_0$.

This roundabout encoding is actually faithful to the original inductive-recursive definition, because we can derive the same induction principle with its computation rules (Coq definition), and the three functions $\texttt{El}_0$, $\texttt{eqU}_0$ and $\texttt{eq}_0$ compute on type formers.

3.3 Transitivity and Coercion

In order to complete the definition of our universal setoid, we need to show that the equality relation $\texttt{eqU}_0$ is an equivalence relation on $\texttt{U}_0$, and that the relation $\texttt{eq}_0$ is a heterogeneous equality equipped with a coercion operator. Reflexivity and symmetry are a straightforward application of the induction principle for $\texttt{U}_0$ (Coq definition), but transitivity and the coercion operator require a bit more work. All in all, we want the following four operators:

$$\begin{array}{lll}
\texttt{transU}_0 & : & \forall\ (\texttt{A B C} : \texttt{U}_0),\ \texttt{eqU}_0\ \texttt{A B} \to \texttt{eqU}_0\ \texttt{B C} \to \texttt{eqU}_0\ \texttt{A C} \\
\texttt{trans} & : & \forall\ (\texttt{A B C} : \texttt{U}_0)\ (\texttt{e} : \texttt{eqU}_0\ \texttt{A B})\ (\texttt{a} : \texttt{El}_0\ \texttt{A})\ (\texttt{b} : \texttt{El}_0\ \texttt{B})\ (\texttt{c} : \texttt{El}_0\ \texttt{C}), \\
 & & \quad \texttt{eq}_0\ \texttt{A B a b} \to \texttt{eq}_0\ \texttt{B C b c} \to \texttt{eq}_0\ \texttt{A C a c} \\
\texttt{cast} & : & \forall\ (\texttt{A B} : \texttt{U}_0)\ (\texttt{e} : \texttt{eqU}_0\ \texttt{A B})\ (\texttt{a} : \texttt{El}_0\ \texttt{A}),\ \texttt{El}_0\ \texttt{B} \\
\texttt{casteq} & : & \forall\ (\texttt{A B} : \texttt{U}_0)\ (\texttt{e} : \texttt{eqU}_0\ \texttt{A B})\ (\texttt{a} : \texttt{El}_0\ \texttt{A}),\ \texttt{eq}_0\ \texttt{A B a}\ (\texttt{cast A B e a})
\end{array}$$

Consider how the `trans` operator should compute on dependent products. If `f` : Π `A P` is equal to `g` : Π `B Q`, which is itself equal to `h` : Π `C R`, we want to produce a proof of equality between `f` and `h`. To this end, we need to explain how to go from a proof of `eq0 A C a c` to a proof of `eq0 (P a) (R c) (f a) (h c)`, but our hypotheses only provide functoriality laws for equalities that involve inhabitants of `B`. Thus, our only choice is to use `cast` and `casteq` to construct an element `b` : `B` that is equal to `a` and `c`, and then use our functoriality hypotheses and a recursive call to transitivity on the codomain. But on the other hand, the definition of `cast` and `casteq` for dependent products involves an application of the `trans` operator on the domain and the codomain.

In the end, `trans` `cast` and `casteq` can be defined mutually by induction on $\texttt{U}_0$. In the Coq definition, we are forced to duplicate each of these operators (with a version that goes forward and a version that goes backward), or otherwise the contravariance introduced by the dependent products would throw off the syntactic termination checker. Once these three operators are defined, $\texttt{transU}_0$ can be obtained with a straightforward induction on $\texttt{U}_0$.

3.4 The Universe Hierarchy

We designed $\texttt{U}_0$ as a universe of small setoids, *i.e.*, setoids whose carrier type is in $\texttt{Type}_0$ and whose equality relation is in $\texttt{Sort}_0$. The next step is the definition of $\texttt{U}_1$, a similar universe for setoids whose underlying type is in $\texttt{Type}_1$ and whose equality is in $\texttt{Sort}_1$. In particular $\texttt{U}_1$ should contain a code for $\texttt{U}_0$, an embedding of all the codes from $\texttt{U}_0$, and it should be closed under dependent products and dependent sums. To achieve this, we use the same recipe as for $\texttt{U}_0$, but with a slightly different set of constructors:

```
Inductive preU₁ : Type₂ :=
| pre_emb : ∀ (A : U₀), preU₁
| pre_U   : preU₁
| pre_Σ   : (* same as in U₀ *)
| pre_Π   : (* same as in U₀ *)
```

```
Fixpoint El₁ (A : preU₁) : Type₁ :=
  match A with
  | pre_emb A ⇒ El₀ A
  | pre_U ⇒ U₀
  | pre_Σ ⇒ (* same as in U₀ *)
  | pre_Π ⇒ (* same as in U₀ *)
  end.
```

We follow with a definition of $\texttt{eqU}_1$ and $\texttt{eq}_1$, which are respectively the setoid equality on $\texttt{U}_1$ and the heterogeneous equality on the universal family over $\texttt{U}_1$. On the constructors $\texttt{pre}_\texttt{U}$ and $\texttt{pre}_\texttt{emb}$, we simple reuse the previously defined equalities. The definition of reflexivity, symmetry, transitivity and the coercion operator are similarly straightforward, reusing the previously defined operators where they come in handy.

$$\begin{array}{lll@{\qquad}lll}
\texttt{eqU}_1\ \texttt{c}_\texttt{U}\ \texttt{c}_\texttt{U} & := & \texttt{True} & \texttt{eq}_1\ \texttt{c}_\texttt{U}\ \texttt{c}_\texttt{U}\ \texttt{A B} & := & \texttt{eqU}_0\ \texttt{A B} \\
\texttt{eqU}_1\ (\texttt{c}_\texttt{emb}\ \texttt{A})\ (\texttt{c}_\texttt{emb}\ \texttt{B}) & := & \texttt{eqU}_0\ \texttt{A B} & \texttt{eq}_1\ (\texttt{c}_\texttt{emb}\ \texttt{A})\ (\texttt{c}_\texttt{emb}\ \texttt{B})\ \texttt{a b} & := & \texttt{eq}_0\ \texttt{A B a b}
\end{array}$$

This construction yields a functioning universe hierarchy of any finite length with a notion of explicit cumulativity provided by the embedding constructor $\texttt{pre}_\texttt{emb}$. Another approach would be to remove that constructor and define the universe embedding as a setoid morphism from $\mathtt{U}_0$ to $\mathtt{U}_1$ instead, but the definition of this function does not seem to be within the reach of a straightforward application of our induction principle.

3.5 Propositions

In parallel of the universe hierarchy, it makes sense to have a sort for propositions, since they play such an important role in setoid models. Here, the story is bound to be different between the impredicative model and the predicative model, so we treat them separately. First, in the impredicative case, we should have one setoid of propositions that sits at the bottom of the universe hierarchy, so we add a new constructor $\mathtt{c}_\Omega$ to the universe $\mathtt{U}_0$, along with the following definitions:

$$\begin{array}{lll}
\mathtt{c}_\Omega & : & \mathtt{U}_0 \\
\mathtt{El}_0\ \mathtt{c}_\Omega & := & \mathtt{Prop} \\
\mathtt{eqU}_0\ \mathtt{c}_\Omega\ \mathtt{c}_\Omega & := & \mathtt{True} \\
\mathtt{eq}_0\ \mathtt{c}_\Omega\ \mathtt{c}_\Omega\ \mathtt{A}\ \mathtt{B} & := & \mathtt{A} \leftrightarrow \mathtt{B}
\end{array}$$

We shall also add a constructor $\mathsf{c}_{\mathsf{emb}\Omega}$ that simulates the cumulativity of $\mathtt{Prop}$ into $\mathtt{Type}$:

$$\begin{array}{lll}
\mathsf{c}_{\mathsf{emb}\Omega} & : & \mathtt{Prop} \to \mathtt{U}_0 \\
\mathtt{El}_0\ (\mathsf{c}_{\mathsf{emb}\Omega}\ \mathtt{A}) & := & \mathtt{A} \\
\mathtt{eqU}_0\ (\mathsf{c}_{\mathsf{emb}\Omega}\ \mathtt{A})\ (\mathsf{c}_{\mathsf{emb}\Omega}\ \mathtt{B}) & := & \mathtt{A} \leftrightarrow \mathtt{B} \\
\mathtt{eq}_0\ (\mathsf{c}_{\mathsf{emb}\Omega}\ \mathtt{A})\ (\mathsf{c}_{\mathsf{emb}\Omega}\ \mathtt{B})\ \mathtt{a}\ \mathtt{b} & := & \mathtt{True}
\end{array}$$

On the other hand, in the predicative model, we should have an entire hierarchy of propositional universes that mirrors the $\mathtt{Type}$ hierarchy. Thus, we add the constructor $\mathtt{c}_\Omega$ to every universe *except for the bottom one*, and we define our setoids of propositions as $\mathtt{Type}_i$ quotiented by logical equivalence:

$$\begin{array}{lll}
\mathtt{c}_\Omega & : & \mathtt{U}_{i+1} \\
\mathtt{El}_{i+1}\ \mathtt{c}_\Omega & := & \mathtt{Type}_i \\
\mathtt{eqU}_{i+1}\ \mathtt{c}_\Omega\ \mathtt{c}_\Omega & := & \mathtt{True} \\
\mathtt{eq}_{i+1}\ \mathtt{c}_\Omega\ \mathtt{c}_\Omega\ \mathtt{A}\ \mathtt{B} & := & \mathtt{A} \leftrightarrow \mathtt{B}
\end{array}$$

We also add a constructor $\mathsf{c}_{\mathsf{emb}\Omega} : \mathtt{Type}_i \to \mathtt{U}_i$ to embed propositions into the universes, with the same definitions as above for the setoid equalities.

4 Designing a Model from the Universe Hierarchy

We now have all of the necessary ingredients to build our model. Our plan is to interpret contexts as inhabitants of an universe $\mathtt{U}_i$ and substitutions between two contexts as setoid morphisms. Then, given a context $\Gamma : \mathtt{U}_i$, a dependent type of level j over Γ will be a setoid morphism from $\mathtt{El}_i\ \Gamma$ to $\mathtt{U}_j$, and a term of that type will be an inhabitant of the corresponding dependent product of setoids.

If we organise this data as a category with families (CwF, [11]), we can show that the resulting CwF supports dependent products and dependent sums along with their η-rules, natural numbers with their dependent eliminator, a sort of propositions, *etc.* In other words, we can get a fully fledged model of dependent type theory. However, such a presentation would hide an important aspect of our model construction: because the judgmental equalities of intensional type theory are interpreted as judgmental equalities of the metatheory, our model *preserves computation*. In order to properly account for this aspect, we will instead present our model as a syntactic translation in the sense of Boulier *et al.* [5].

We use two interpretation functions $[\![\_]\!]$ and $\{\!\{\_\}\!\}$ that are defined by recursion on untyped syntax, with the intended meaning being that if the judgment $\Gamma \vdash \mathtt{t} : \mathtt{A}$ is derivable in CIC (for the impredicative model) or in predicative CIC (for the predicative model), then the following two judgments are derivable in CIC (resp. preCIC):

$$\begin{array}{lll} [\![\Gamma]\!] & \vdash [\![\mathtt{t}]\!]_\Gamma & : \mathtt{El}_i\ [\![\mathtt{A}]\!]_\Gamma \\ \{\!\{\Gamma\}\!\} & \vdash \{\!\{\mathtt{t}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e} & : \mathtt{eq}_i\ [\![\mathtt{A}]\!]_{\Gamma_0}\ [\![\mathtt{A}]\!]_{\Gamma_1}\ [\![\mathtt{t}]\!]_{\Gamma_0}\ [\![\mathtt{t}]\!]_{\Gamma_1} \end{array}$$

The first judgment corresponds to a dependent function from the setoid interpretation of Γ to the setoid interpretation of $\mathtt{A}$, while the second judgment ensures that this function is a setoid morphism which sends equal inputs to equal outputs. Accordingly, the translated context $\{\!\{\Gamma\}\!\}$ in the second judgment consists of two instances Γ_0 and Γ_1 of $[\![\Gamma]\!]$ and a list of equalities Γ_e between the two instances; the translated term $\{\!\{\mathtt{t}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e}$ is a proof of equality between $[\![\mathtt{t}]\!]_{\Gamma_0}$, the translation of $\mathtt{t}$ whose variables have been instanciated with variables from Γ_0, and $[\![\mathtt{t}]\!]_{\Gamma_1}$, whose variables have been instanciated with variables from Γ_1.

Contexts and Variables

The interpretation of contexts is defined by recursion on their length:

$$\begin{array}{lcl} [\![\varepsilon]\!] & = & \varepsilon \\ \{\!\{\varepsilon\}\!\} & = & \varepsilon \\ [\![\Gamma, \mathtt{x} : \mathtt{A}]\!] & = & [\![\Gamma]\!],\ \mathtt{x} : \mathtt{El}_i\ [\![\mathtt{A}]\!]_\Gamma \\ \{\!\{\Gamma, \mathtt{x} : \mathtt{A}\}\!\} & = & [\![\Gamma]\!],\ \mathtt{x}_0 : \mathtt{El}_i\ [\![\mathtt{A}]\!]_{\Gamma_0},\ \mathtt{x}_1 : \mathtt{El}_i : [\![\mathtt{A}]\!]_{\Gamma_1},\ \mathtt{x}_e : \mathtt{eq}_i\ [\![\mathtt{A}]\!]_{\Gamma_0}\ [\![\mathtt{A}]\!]_{\Gamma_1}\ [\![\mathtt{t}]\!]_{\Gamma_0}\ [\![\mathtt{t}]\!]_{\Gamma_1} \end{array}$$

where i is the universe level of the type $\mathtt{A}$. Writing these universe levels everywhere gets a bit cumbersome, so from now on we will omit them and simply write $\mathtt{El}$ or $\mathtt{eq}$. Now, we can describe the interpretation of the terms of our theory, starting with variables:

$$\begin{array}{lcl} [\![\mathtt{x}]\!]_\Gamma & = & \Gamma(\mathtt{x}) \\ \{\!\{\mathtt{x}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e} & = & \Gamma_e(\mathtt{x}) \end{array}$$

Dependent Sums

$$\begin{array}{lcl} [\![\Sigma\ (\mathsf{a} : \mathtt{A}), \mathtt{B}]\!]_\Gamma & = & \mathsf{c}_\Sigma\ [\![\mathtt{A}]\!]_\Gamma\ (\mathtt{fun}\ \mathsf{a} \Rightarrow [\![\mathtt{B}]\!]_{\Gamma,\mathsf{a}})\ (\mathtt{fun}\ \mathsf{a}_0\ \mathsf{a}_1\ \mathsf{a}_e \Rightarrow \{\!\{\mathtt{B}\}\!\}_{\Gamma,\Gamma,\mathsf{refl},\mathsf{a}_0,\mathsf{a}_1,\mathsf{a}_e}) \\ \{\!\{\Sigma\ (\mathsf{a} : \mathtt{A}), \mathtt{B}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e} & = & (\{\!\{\mathtt{A}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e}\ ;\ \mathtt{fun}\ \mathsf{a}_0\ \mathsf{a}_1\ \mathsf{a}_e \Rightarrow \{\!\{\mathtt{B}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e,\mathsf{a}_0,\mathsf{a}_1,\mathsf{a}_e}) \\ [\![(\mathtt{t}\ ;\ \mathtt{u})]\!]_\Gamma & = & ([\![\mathtt{t}]\!]_\Gamma\ ;\ [\![\mathtt{u}]\!]_\Gamma) \\ \{\!\{(\mathtt{t}\ ;\ \mathtt{u})\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e} & = & (\{\!\{\mathtt{t}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e}\ ;\ \{\!\{\mathtt{u}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e}) \\ [\![\mathtt{t}.1]\!]_\Gamma & = & ([\![\mathtt{t}]\!]_\Gamma).1 \\ \{\!\{\mathtt{t}.1\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e} & = & (\{\!\{\mathtt{t}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e}).1 \\ [\![\mathtt{t}.2]\!]_\Gamma & = & ([\![\mathtt{t}]\!]_\Gamma).2 \\ \{\!\{\mathtt{t}.2\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e} & = & (\{\!\{\mathtt{t}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e}).2 \end{array}$$

Dependent sums are directly translated as metatheoretical dependent sums. This implies that both the β-reduction rules and the η-expansion rule are preserved by the translation.

Dependent Products

$$\begin{aligned}
\llbracket \forall\ (\mathtt{a} : \mathtt{A}),\ \mathtt{B} \rrbracket_\Gamma &= \mathtt{c}_\Pi\ \llbracket \mathtt{A} \rrbracket_\Gamma\ (\mathtt{fun}\ \mathtt{a} \Rightarrow \llbracket \mathtt{B} \rrbracket_{\Gamma,\mathtt{a}})\ (\mathtt{fun}\ \mathtt{a}_0\ \mathtt{a}_1\ \mathtt{a}_e \Rightarrow \{\!\{\mathtt{B}\}\!\}_{\Gamma,\Gamma,\mathsf{refl},\mathtt{a}_0,\mathtt{a}_1,\mathtt{a}_e})\\
\{\!\{\forall\ (\mathtt{a} : \mathtt{A}),\ \mathtt{B}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e} &= (\{\!\{\mathtt{A}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e}\ ;\ \mathtt{fun}\ \mathtt{a}_0\ \mathtt{a}_1\ \mathtt{a}_e \Rightarrow \{\!\{\mathtt{B}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e,\mathtt{a}_0,\mathtt{a}_1,\mathtt{a}_e})\\[1ex]
\llbracket \mathtt{fun}\ (\mathtt{a} : \mathtt{A}) \Rightarrow \mathtt{t} \rrbracket_\Gamma &= (\mathtt{fun}\ (\mathtt{a} : \mathtt{El}\ \llbracket \mathtt{A} \rrbracket_\Gamma) \Rightarrow \llbracket \mathtt{t} \rrbracket_{\Gamma,\mathtt{a}}\ ;\ \mathtt{fun}\ \mathtt{a}_0\ \mathtt{a}_1\ \mathtt{a}_e \Rightarrow \{\!\{\mathtt{t}\}\!\}_{\Gamma,\Gamma,\mathsf{refl},\mathtt{a}_0,\mathtt{a}_1,\mathtt{a}_e})\\
\{\!\{\mathtt{fun}\ (\mathtt{a} : \mathtt{A}) \Rightarrow \mathtt{t}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e} &= \mathtt{fun}\ \mathtt{a}_0\ \mathtt{a}_1\ \mathtt{a}_e \Rightarrow \{\!\{\mathtt{t}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e,\mathtt{a}_0,\mathtt{a}_1,\mathtt{a}_e}\\[1ex]
\llbracket \mathtt{t}\ \mathtt{u} \rrbracket_\Gamma &= (\llbracket \mathtt{t} \rrbracket_\Gamma).1\ \ \llbracket \mathtt{u} \rrbracket_\Gamma\\
\{\!\{\mathtt{t}\ \mathtt{u}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e} &= \{\!\{\mathtt{t}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e}\ \ \llbracket \mathtt{u} \rrbracket_{\Gamma_0}\ \ \llbracket \mathtt{u} \rrbracket_{\Gamma_1}\ \ \{\!\{\mathtt{u}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e}
\end{aligned}$$

The translation preserves β-reduction, but unfortunately the same cannot be said of the judgmental η-expansion rule. After applying $\llbracket \_ \rrbracket$ to both sides of $\mathtt{f} = (\mathtt{fun}\ \mathtt{x} \Rightarrow \mathtt{f}\ \mathtt{x})$, we are left with an equation that boils down to

$$(\llbracket \mathtt{f} \rrbracket.1\ ;\ \llbracket \mathtt{f} \rrbracket.2) \overset{?}{=} (\llbracket \mathtt{f} \rrbracket.1\ ;\ \{\!\{\mathtt{f}\}\!\})$$

which does not have any good reason to hold judgmentally in the absence of judgmental proof irrelevance. However, remark that the setoid equality on dependent products equates functions with their η-expanded forms, which means that the η-rule at least holds up to a propositional equality.

Natural Numbers

$$\begin{aligned}
\llbracket \mathbb{N} \rrbracket_\Gamma &= \mathtt{c}_\mathbb{N}\\
\{\!\{\mathbb{N}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e} &= *\quad \texttt{(* The unique inhabitant of True *)}\\[1ex]
\llbracket \mathtt{zero} \rrbracket_\Gamma &= \mathtt{zero}\\
\{\!\{\mathtt{zero}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e} &= \mathbb{N}\mathtt{eq\_zero}\\[1ex]
\llbracket \mathtt{suc}\ \mathtt{n} \rrbracket_\Gamma &= \mathtt{suc}\ \llbracket \mathtt{n} \rrbracket_\Gamma\\
\{\!\{\mathtt{suc}\ \mathtt{n}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e} &= \mathbb{N}\mathtt{eq\_suc}\ \{\!\{\mathtt{n}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e}
\end{aligned}$$

The interpretation of the natural numbers uses the metatheoretical natural numbers quite transparently. As a consequence, the translation of the recursor can be derived from the metatheoretical recursor:

$$\llbracket \mathbb{N}\mathtt{\_rec}\ \mathtt{P}\ \mathtt{p_z}\ \mathtt{p}_s\mathtt{uc}\ \mathtt{n} \rrbracket_\Gamma = \mathbb{N}\mathtt{\_rec}\ (\mathtt{fun}\ \mathtt{n} \Rightarrow \mathtt{El}\ (\llbracket \mathtt{P} \rrbracket_\Gamma.1\ \mathtt{n}))\ \llbracket \mathtt{p_z} \rrbracket_\Gamma\ (\mathtt{fun}\ \mathtt{m}\ \mathtt{p} \Rightarrow (\llbracket \mathtt{p}_s\mathtt{uc} \rrbracket_\Gamma.1\ \mathtt{m}).1\ \mathtt{p})\ \llbracket \mathtt{n} \rrbracket_\Gamma$$

Likewise, $\{\!\{\mathbb{N}\mathtt{\_rec}\ \mathtt{P}\ \mathtt{p_z}\ \mathtt{p}_s\mathtt{uc}\ \mathtt{n}\}\!\}$ is derived from the metatheoretical recursor for $\mathbb{N}\mathtt{eq}$. We do not reproduce the full term here, but the interested reader can consult the Coq definition.

Universes and Propositions

The universe $\mathtt{Type}_i$ is interpreted as the code $\mathtt{c}_\mathtt{U}$ in the setoid $\mathtt{U}_{i+1}$. In the impredicative model, we also interpret the universe of propositions $\mathtt{Prop}$ as the code $\mathtt{c}_\Omega$:

$$\begin{aligned}
\llbracket \mathtt{Type}_i \rrbracket_\Gamma &= \mathtt{c}_\mathtt{U} & \qquad \llbracket \mathtt{Prop} \rrbracket_\Gamma &= \mathtt{c}_\Omega\\
\{\!\{\mathtt{Type}_i\}\!\}_\Gamma &= * & \qquad \{\!\{\mathtt{Prop}\}\!\}_\Gamma &= *
\end{aligned}$$

Naturally, that second part does not apply to the predicative model.

Reifying the Setoid Equality

Having worked so hard to setup our model in the category of setoids, it is time to reap the reward: a flexible notion of equality that validates function extensionality and proposition extensionality. In order to account for this extra data, we extend the source theory of our model with axioms that reify the behaviour of the setoid equality. More formally, define the type theory $\mathrm{CIC}_{\mathsf{ext}}$ as the fragment of CIC that is supported by our impredicative model (*i.e.,* dependent products without η, dependent sums, natural numbers, universes and the impredicative sort of propositions), plus the following axioms:

```
eq      : ∀ {A B : Type_i}, A → B → Prop
refl    : ∀ {A : Type_i} (a : A), eq a a
sym     : ∀ {A B : Type_i} (a : A) (b : B), eq a b → eq b a
trans   : ∀ {A B C : Type_i} (e : eq A B) (a : A) (b : B) (c : C),
            eq a b → eq b c → eq a c
ap      : ∀ {A P} (f : ∀ (a : A), P a) (a b : A), eq a b → eq (f a) (f b)
cast    : ∀ {A B : Type_i} (e : eq A B), A → B
casteq  : ∀ {A B : Type_i} (e : eq A B) (a : A), eq a (cast e a)
irr     : ∀ {P Q : Prop} (p : P) (q : Q), eq p q
propext : ∀ (P Q : Prop), eq P Q ↔ (P ↔ Q)
funext  : ∀ {A B} {P : A → Type_i} {Q : B → Type_i} (f : ∀ (a : A), P a) (g : ∀ (b : B), Q b),
            (∀ a b, eq a b → eq (f a) (g b)) → eq f g
pi_inj  : ∀ {A B} {P : A → Type_i} {Q : B → Type_i},
            eq (∀ (a : A), P a) (∀ (b : B), Q b) → (eq A B) × (eq P Q)
sig_inj : ∀ {A B} {P : A → Type_i} {Q : B → Type_i},
            eq (Σ (a : A), P a) (Σ (b : B), Q b) → (eq A B) × (eq P Q)
```

Similarly, we define the type theory $\mathrm{preCIC}_{\mathsf{ext}}$ as the fragment of preCIC that is supported by the predicative model, extended with a hierarchy of propositional universes $\{\mathtt{Prop}_i\}_{i\in\mathbb{N}}$ such that $\mathtt{Prop}_i : \mathtt{Type}_{i+1}$, and all of the above axioms for the setoid equality (after replacing $\mathtt{Prop}$ with $\mathtt{Prop}_i$ in their types). Finally, we arrive at the following two theorems:

▶ **Theorem 1** (Soundness of the Impredicative Model). *If the judgment* $\Gamma \vdash \mathtt{t} : \mathtt{A}$ *is derivable in* $\mathrm{CIC}_{\mathsf{ext}}$*, then the following two judgments are derivable in* CIC*:*

$$\begin{aligned} [\![\Gamma]\!] &\vdash [\![\mathtt{t}]\!]_\Gamma &&: \mathtt{El}_i\ [\![\mathtt{A}]\!]_\Gamma \\ \{\!\{\Gamma\}\!\} &\vdash \{\!\{\mathtt{t}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e} &&: \mathsf{eq}_i\ [\![\mathtt{A}]\!]_{\Gamma_0}\ [\![\mathtt{A}]\!]_{\Gamma_1}\ [\![\mathtt{t}]\!]_{\Gamma_0}\ [\![\mathtt{t}]\!]_{\Gamma_1} \end{aligned}$$

▶ **Theorem 2** (Soundness of the Predicative Model). *If the judgment* $\Gamma \vdash \mathtt{t} : \mathtt{A}$ *is derivable in* $\mathrm{preCIC}_{\mathsf{ext}}$*, then the following two judgments are derivable in* preCIC*:*

$$\begin{aligned} [\![\Gamma]\!] &\vdash [\![\mathtt{t}]\!]_\Gamma &&: \mathtt{El}_i\ [\![\mathtt{A}]\!]_\Gamma \\ \{\!\{\Gamma\}\!\} &\vdash \{\!\{\mathtt{t}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e} &&: \mathsf{eq}_i\ [\![\mathtt{A}]\!]_{\Gamma_0}\ [\![\mathtt{A}]\!]_{\Gamma_1}\ [\![\mathtt{t}]\!]_{\Gamma_0}\ [\![\mathtt{t}]\!]_{\Gamma_1} \end{aligned}$$

Additionally, since the natural numbers are interpreted as the type of natural numbers of CIC, our model satisfies the *canonicity* property for natural numbers: any number that can be defined in the empty context evaluates to an actual natural number made of zero and successors.

5 Quotients and Unique Choice

Setoids were originally studied by Hofmann as a model for quotients. Thus, it should come as no surprise that our universe of setoids can be equipped with a notion of quotient types. We extend the definition of $\mathtt{U}_i$ with a new constructor:

$$\begin{array}{lcl}
\mathtt{cQ} & : & \forall\ (\mathtt{A} : \mathtt{U}_i)\ (\mathtt{R} : \mathtt{A} \to \mathtt{A} \to \mathtt{Sort}_i),\ \mathtt{U}_0 \\
\mathtt{El}_i\ (\mathtt{cQ\ A\ R\ Re}) & := & \mathtt{El}_i\ \mathtt{A} \\
\mathtt{eq}_i\ (\mathtt{cQ\ A\ R\ Re})\ (\mathtt{cQ\ B\ S\ Se})\ \mathtt{a\ b} & := & \Sigma\ (\mathtt{a}_0 : \mathtt{A})\ (\mathtt{b}_0 : \mathtt{B}),\ (\mathtt{R}{*}\ \mathtt{a}\ \mathtt{a}_0) \times (\mathtt{eq}_i\ \mathtt{A\ B}\ \mathtt{a}_0\ \mathtt{b}_0) \times (\mathtt{S}{*}\ \mathtt{b}_0\ \mathtt{b}) \\
\mathtt{eqU}_i\ (\mathtt{cQ\ A\ R\ Re})\ (\mathtt{cQ\ B\ S\ Se}) & := & (\mathtt{eqU}_i\ \mathtt{A\ B}) \times (\forall\ \mathtt{a}_0\ \mathtt{b}_0\ (\mathtt{e}_0 : \mathtt{eq}_i\ \mathtt{A\ B}\ \mathtt{a}_0\ \mathtt{b}_0) \\
 & & \qquad\qquad \mathtt{a}_1\ \mathtt{b}_1\ (\mathtt{e}_1 : \mathtt{eq}_i\ \mathtt{A\ B}\ \mathtt{a}_1\ \mathtt{b}_1), \\
 & & \qquad\qquad \mathtt{R}\ \mathtt{a}_0\ \mathtt{a}_1 \leftrightarrow \mathtt{S}\ \mathtt{b}_0\ \mathtt{b}_1)
\end{array}$$

Remark that our constructor for quotients does not require R to be an equivalence relation. Instead, the definition of the equality between two inhabitants of a quotient types uses the reflexive, symmetric and transitive closures of the relations R and S, which we denote with an asterisk. Next, we reify the quotient types in the syntax of our source theories $\mathrm{CIC}_{\mathsf{ext}}$ and $\mathrm{preCIC}_{\mathsf{ext}}$ with the following primitives:

$$\begin{array}{lcl}
\mathtt{Q} & : & \forall\ (\mathtt{A} : \mathtt{Type}_i)\ (\mathtt{R} : \mathtt{A} \to \mathtt{A} \to \mathtt{Prop}_i),\ \mathtt{Type}_i \\
\mathtt{quo} & : & \forall\ \{\mathtt{A}\}\ \{\mathtt{R}\},\ \mathtt{A} \to \mathtt{Q\ A\ R} \\
\mathtt{Q\_eq} & : & \forall\ \{\mathtt{A}\}\ \{\mathtt{R}\}\ \{\mathtt{a\ b} : \mathtt{A}\},\ \mathtt{R\ a\ b} \to \mathtt{eq}_0\ (\mathtt{Q\ A\ R})\ (\mathtt{Q\ A\ R})\ (\mathtt{quo\ a})\ (\mathtt{quo\ b}) \\
\mathtt{Q\_rec} & : & \forall\ \{\mathtt{A}\}\ \{\mathtt{R}\}\ (\mathtt{P} : \mathtt{Q\ A\ R} \to \mathtt{Type}_j)\ (\mathtt{p}_{\mathsf{quo}} : \forall\ \mathtt{a},\ \mathtt{P}\ (\mathtt{quo\ a})) \\
 & & (\mathtt{p}_e : \forall\ \mathtt{a\ b},\ \mathtt{R\ a\ b} \to \mathtt{eq}_0\ (\mathtt{P}\ (\mathtt{quo\ a}))\ (\mathtt{P}\ (\mathtt{quo\ b}))\ (\mathtt{p}_{\mathsf{quo}}\ \mathtt{a})\ (\mathtt{p}_{\mathsf{quo}}\ \mathtt{b})) \\
 & & (\mathtt{x} : \mathtt{Q\ A\ R}),\ \mathtt{P\ x}
\end{array}$$

and the computation rule $\mathtt{Q\_rec}\ \mathtt{P}\ \mathtt{p}_{\mathsf{quo}}\ \mathtt{p}_e\ (\mathtt{quo\ a}) = \mathtt{p}_{\mathsf{quo}}\ \mathtt{a}$. The reader who is interested in the detailed implementation of these primitives is invited to consult the Coq development.

5.1 Propositional Truncation and Unique Choice

We would like to draw the reader's attention to the fact that a type may only be quotiented by a propositional relation. This means that if we want to take the quotient of a type A by a relation that is valued in Type, we need to find a way to approximate this relation with a proposition. This is precisely the role of a *truncation operator*, which may be axiomatised in the source theory as follows:

$$\begin{array}{lcl}
\|\_\| & : & \mathtt{Type}_i \to \mathtt{Prop}_i \\
|\_| & : & \forall\ (\mathtt{A} : \mathtt{Type}_i),\ \mathtt{A} \to \|\,\mathtt{A}\,\| \\
\mathtt{unbox} & : & \forall\ (\mathtt{A} : \mathtt{Type}_i)\ (\mathtt{P} : \mathtt{Prop}_i),\ (\mathtt{A} \to \mathtt{P}) \to \|\,\mathtt{A}\,\| \to \mathtt{P}
\end{array}$$

In the predicative model, we can find a very simple interpretation for truncation: since $[\![\,\mathtt{Prop}_i\,]\!]$ is defined as $\mathtt{Type}_i$, we interpret the truncation operator as $[\![\,\|\,\mathtt{A}\,\|\,]\!] := \mathtt{El}_i\ [\![\,\mathtt{A}\,]\!]$. With this definition, the truncation operation forgets the setoid structure, but it does not affect the underlying type. This allows us to add an operator that evades truncation when A is a homotopy proposition:

$$\mathtt{evade} : \forall\ (\mathtt{A} : \mathtt{Type}_i),\ (\forall\ (\mathtt{a\ b} : \mathtt{A}),\ \mathtt{eq}_i\ \mathtt{A\ A\ a\ b}) \to \|\,\mathtt{A}\,\| \to \mathtt{A}$$

As we explained in the introduction, evade is equivalent to the principle of unique choice. This has important consequence for our source type theory: functions become equivalent to functional relations, and quotient types become *effective* in the sense of Hofmann [14]. Unfortunately, this trick does not work for the impredicative model, since we cannot evade the universe of propositions so easily.

6 The Accessibility Predicate

We cannot get unique choice in our impredicative model, but we have another ace up our sleeve: recall the inductive definition of the accessibility predicate in CIC

```
Inductive Acc {A : Type} {R : A → A → Prop} (a : A) : Prop :=
| acc : (∀ (b : A), R b a → Acc b) → Acc a.
```

Accessibility provides a type-theoretic way to capture the notion of well-foundedness: an element of `A` is accessible when all of its predecessors for the relation `R` are themselves accessible. A key point about this definition is that it fulfills the syntactic criterion for being a subsingleton inductive type[2], which means that large elimination is allowed for accessibility proofs. In other words, we can use accessibility to build inhabitants of arbitrarily large types by induction over a relation that is only *propositionally* well-founded.

6.1 Escaping the World of Propositions

Large elimination of the accessibility predicate provides a notion of truncation elimination for Σ^0_1 types, *i.e.*, types that are equivalent to Σ (n : ℕ). P n for some decidable predicate `P`.

▶ **Theorem 3.** *The following statement is provable in CIC:*

```
∀ (P : ℕ → Prop) (decP : ∀ n, (P n) + (¬ P n)), ‖Σ (n : ℕ). P n‖ → Σ (n : ℕ). P n
```

Proof. If (`P 0`) is true, then we are done. Otherwise, we can show that 0 is accessible for the relation $>$ on the type of integers that do not satisfy `P`. Indeed, accessibility is a proposition, so in order to prove it we may unbox the truncated hypothesis and reason by induction on the integer that satisfies `P`. Then, we show that there exists some `n` such that (`P n`) by induction on the accessibility proof, using the decidability hypothesis to go from `n` to `n+1`. ◀

Although theorem 3 is weaker than unique choice, it lets us extract computational information from the world of propositions: given a Turing machine and a proof that it halts on every input, we can extract the function computed by this machine as a term of type ℕ → ℕ. For instance, theorem 3 can be combined with impredicativity to define a function that computes the normal form of any well-typed term of System F.

6.2 Accessibility in the World of Setoids

We extend the source theory of the impredicative model with the following primitives:

```
Acc      :  ∀ (A : Type) (R : A → A → Prop), A → Prop
acc      :  ∀ {A} {R} (a : A), (∀ (b : A), R b a → Acc A R b) → Acc A R a
acc_rec  :  ∀ {A} {R} (P : A → Type_i),
              (∀ a, (∀ b, R b a → Acc A R b) → (∀ b, R b a → P b) → P a)
              → ∀ (a : A), Acc A R a → P a
```

and the computation rule `acc_rec P IH a (acc a Ha) = IH a Ha (fun b Hb ⇒ acc_rec P IH b Hb)`. The predicate `Acc` is directly interpreted as the metatheoretical accessibility predicate, but

[2] Recall that an inductive type is a syntactic subsingleton when it has only one constructor, and all of the arguments of this constructor are propositions.

the proof that it preserves setoid equalities is a bit more involved, as it follows from an well-founded induction over an accessibility proof (Coq definition).

$$\begin{aligned}
[\![\texttt{Acc A R a}]\!]_\Gamma &= \texttt{Acc (El } [\![\texttt{A}]\!]_\Gamma\texttt{) (fun a b} \Rightarrow \texttt{(}[\![\texttt{R}]\!]_\Gamma\texttt{.1 a).1 b) } [\![\texttt{a}]\!]_\Gamma \\
\{\!\{\texttt{Acc A R a}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e} &= \texttt{(* ... *)}
\end{aligned}$$

$$\begin{aligned}
[\![\texttt{acc a Ha}]\!]_\Gamma &= \texttt{acc } [\![\texttt{a}]\!]_\Gamma\ [\![\texttt{Ha}]\!]_\Gamma \\
\{\!\{\texttt{acc a Ha}\}\!\}_{\Gamma_0,\Gamma_1,\Gamma_e} &= *
\end{aligned}$$

The interpretation of the induction principle presents a more serious issue: it seems that it is necessary to define $[\![\texttt{acc\_rec}]\!]$ mutually with $\{\!\{\texttt{acc\_rec}\}\!\}$, but the first function appears twice in the return type of the second one, which would result in a "doubly recursive-recursive" definition – which are not allowed in Coq. Even more concerning, the usual trick to get rid of simple recursive-recursive definitions with a Σ-type does not seem to work in this tricky situation. Thankfully, we can sidestep the issue by defining an inductive type that encodes the graph of $[\![\texttt{acc\_rec}]\!]$, and then showing by induction that $\{\!\{\texttt{acc\_rec}\}\!\}$ holds for all the values in the graph. Finally, we use induction to show that every value has an image in the graph, thereby defining our two functions. The interested reader can find the details in the Coq development.

7 Conclusion and Future Work

In this paper, we constructed two setoid models: a model of preCIC which supports extensionality principles, quotients and the principle of unique choice, and a model of CIC which supports extensionality principles, quotients and large elimination of the accessibility predicate. Since our models can be formulated in (pre)CIC and preserve the canonicity of the natural numbers, we obtain a proof that any theorem about integers that can be proved with extensionality principles and weak choice principles can also be proved without them. Nevertheless, there are a few unsatisfactory points left:

- Firstly, our models only validate a propositional η-rule for functions. We believe that this issue could be fixed by ensuring that setoid morphisms preserve reflexivity proofs up to a judgmental equality, but this constraint seems difficult to express in pure CIC.
- Unlike CIC, the equality type of our model is not equipped with a J eliminator that computes on proofs by reflexivity. It seems that it would be possible to adapt the construction of identity types of Pujet and Tabareau [17] to recover a proper J eliminator, at the cost of losing the computation rules of the equality type former.
- Finally, and perhaps most importantly, the fact that we have two models instead of one. We would like to enjoy both impredicativity and unique choice in the same model! However, we suspect that the resulting theory would be logically stronger than CIC, and thus that it is impossible to model it in CIC without axioms.

References

1 Thorsten Altenkirch. Extensional equality in intensional type theory. In *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*, pages 412–420, 1999. doi:10.1109/LICS.1999.782636.

2 Thorsten Altenkirch, Simon Boulier, Ambrus Kaposi, Christian Sattler, and Filippo Sestini. Constructing a universe for the setoid model. In Stefan Kiefer and Christine Tasson, editors, *Foundations of Software Science and Computation Structures*, pages 1–21, Cham, 2021. Springer International Publishing.

3 Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. Observational equality, now! In *Proceedings of the Workshop on Programming Languages meets Program Verification (PLPV 2007)*, pages 57–68, 2007. doi:10.1145/1292597.1292608.

4 Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Robert Harper, Kuen-Bang Hou (Favonia), and Daniel R. Licata. Syntax and models of cartesian cubical type theory. *Mathematical Structures in Computer Science*, 31(4):424–468, 2021. doi:10.1017/S0960129521000347.

5 Simon Boulier, Pierre-Marie Pédrot, and Nicolas Tabareau. The next 700 syntactical models of type theory. In *Proceedings of Certified Programs and Proofs*, pages 182–194. ACM, 2017.

6 Evan Cavallo and Robert Harper. Higher inductive types in cubical computational type theory. *Proc. ACM Program. Lang.*, 3(POPL), jan 2019. doi:10.1145/3290314.

7 Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical Type Theory: a constructive interpretation of the univalence axiom. In *21st International Conference on Types for Proofs and Programs*, number 69 in 21st International Conference on Types for Proofs and Programs, page 262, Tallinn, Estonia, May 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: https://hal.inria.fr/hal-01378906, doi:10.4230/LIPIcs.TYPES.2015.5.

8 The Coq Development Team. *The Coq proof assistant reference manual.* 2016. Version 8.6. URL: http://coq.inria.fr.

9 Thierry Coquand and Christine Paulin. Inductively defined types. In *Proceedings of the International Conference on Computer Logic*, COLOG '88, page 50–66, Berlin, Heidelberg, 1988. Springer-Verlag.

10 Peter Dybjer. *Inductive Sets and Families in Martin-Löf's Type Theory and Their Set-Theoretic Semantics*, page 280–306. Cambridge University Press, USA, 1991.

11 Peter Dybjer. Internal type theory. In Stefano Berardi and Mario Coppo, editors, *Types for Proofs and Programs*, pages 120–134, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

12 Gaëtan Gilbert, Jesper Cockx, Matthieu Sozeau, and Nicolas Tabareau. Definitional Proof-Irrelevance without K. *Proceedings of the ACM on Programming Languages*, 3:1–28, January 2019. URL: https://hal.inria.fr/hal-01859964, doi:10.1145/3290316.

13 Peter Hancock, Conor McBride, Neil Ghani, Lorenzo Malatesta, and Thorsten Altenkirch. Small induction recursion. In Masahito Hasegawa, editor, *Typed Lambda Calculi and Applications*, pages 156–172, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

14 Martin Hofmann. *Extensional concepts in intensional type theory.* PhD thesis, University of Edinburgh, 1995.

15 Martin Hofmann. A simple model for quotient types. In Mariangiola Dezani-Ciancaglini and Gordon Plotkin, editors, *Typed Lambda Calculi and Applications*, pages 216–234, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

16 Per Martin-Löf. An intuitionistic theory of types, 1971. Unpublished manuscript.

17 Loïc Pujet and Nicolas Tabareau. Observational Equality: Now For Good. *Proceedings of the ACM on Programming Languages*, 6(POPL):1–29, January 2022. URL: https://hal.inria.fr/hal-03367052, doi:10.1145/3498693.

18 Jonathan Sterling, Carlo Angiuli, and Daniel Gratzer. A Cubical Language for Bishop Sets. *Logical Methods in Computer Science*, 18, March 2022. arXiv:2003.01491, doi:10.46298/lmcs-18(1:43)2022.

19 Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. Cubical agda: A dependently typed programming language with univalence and higher inductive types. *Proc. ACM Program. Lang.*, 3(ICFP), July 2019. doi:10.1145/3341691.