Constructing a universe for the setoid model

Thorsten Altenkirch[1] *[0000−0002−6582−5025], Simon Boulier[2]**, Ambrus Kaposi[3][0000−0001−9897−8936] ***, Christian Sattler[4]†, and Filippo Sestini[1][0000−0002−8701−5613]

[1] School of Computer Science, University of Nottingham, UK
{psztxa,psxfs5}@nottingham.ac.uk
[2] Inria, Nantes, Francesimon.boulier@inria.fr
[3] Eötvös Loránd University, Budapest, Hungary akaposi@inf.elte.hu
[4] Chalmers University of Technology, Gothenburg, Sweden sattler@chalmers.se

Abstract. The setoid model is a model of intensional type theory that validates certain extensionality principles, like function extensionality and propositional extensionality, the latter being a limited form of univalence that equates logically equivalent propositions. The appeal of this model construction is that it can be constructed in a small, intensional, type theoretic metatheory, therefore giving a method to boostrap extensionality. The setoid model has been recently adapted into a formal system, namely Setoid Type Theory (SeTT). SeTT is an extension of intensional Martin-Löf type theory with constructs that give full access to the extensionality principles that hold in the setoid model.

Although already a rich theory as currently defined, SeTT currently lacks a way to internalize the notion of type beyond propositions, hence we want to extend SeTT with a universe of setoids. To this aim, we present the construction of a (non-univalent) universe of setoids within the setoid model, first as an inductive-recursive definition, which is then translated to an inductive-inductive definition and finally to an inductive family. These translations from more powerful definition schemas to simpler ones ensure that our construction can still be defined in a relatively small metatheory which includes a proof-irrelevant identity type with a strong transport rule.

Keywords: type theory · function extensionality · univalence · setoid model · induction-recursion · induction-induction

* Supported by USAF grant FA9550-16-1-0029.

** Supported by ERC Starting Grant CoqHoTT 637339.

*** Supported by the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme funding scheme, Project no. ED_18-1-2019-0030 and the Bolyai Fellowship of the Hungarian Academy of Sciences, Project no. BO/00659/19/3.

† Supported by USAF grant FA9550-16-1-0029 and Swedish Research Council grant 2019-03765.

1 Introduction

Intuitionistic type theory is a formal system designed by Per Martin-Löf to be a full-fledged foundation in which to develop constructive mathematics [22,23]. A central aspect of type theory is the coexistence of two notions of equality. On the one hand definitional equality, the computational equality that is built into the formalism. On the other hand "propositional" equality, the internal notion of equality that is actually used to state and prove equational theorems within the system. The precise balance between these two notions is at the center of type theory research; however, it is generally understood that to properly support formalization of mathematics, one should aim for a notion of propositional equality that is as *extensional* as possible.

Two extensionality principles seem particularly desirable, since they arguably constitute the bare minimum for type theory to be comparable to set theory as a foundational system for set-level mathematics, in terms of power and ergonomics. One is function extensionality (or *funext*), according to which functions are equal if point-wise equal. Another is propositional extensionality (or *propext*), that equates all propositions that are logically equivalent.

Type theory with equality reflection, also known as *extensional type theory* (ETT) does support extensional reasoning to some degree, but unfortunately equality reflection makes the problem of type-checking ETT terms computationally unfeasible: it is undecidable.

On the other hand, *intensional type theory* (ITT) has nice computational properties like decidable type checking that can make it more suitable for computer implementation, but as usually defined (for example, in [22]) it severely lacks extensionality. It is known from model constructions that extensional principles like funext are consistent with ITT. Moreover, ITT extended with the principle of *uniqueness of identity proofs* (UIP) and funext is known to be as powerful as ETT [18]. We could recover the expressive power of ETT by adding these principles to ITT as axioms, however destroying some computational properties like canonicity.

What we would like instead is a formulation of ITT that supports extensionality, while retaining its convenient computational behaviour. Unfortunately, canonicity for Martin-Löf's inductively defined identity type says that if two terms are propositionally equal in the empty context, then they are also definitionally equal. This rules out function extensionality. The first step towards a solution is to give up the idea of propositional equality as a single inductive definition given generically for arbitrary types. Instead, equality should be *specific* to each type former in the type theory, or in other words, every type former should be introduced alongside an explanation of what counts as equality for its elements.

This idea of pairing types together with their own equality relation goes back to the notion of *setoid* or *Bishop set*. Setoids provide a quite natural and useful semantic domain in which to interpret type theory. The first setoid model was constructed to justify function extensionality without relying on funext in the metatheory [17]. Moreover, it was shown by Altenkirch [4] that if the model construction is carried out in a type theoretic metatheory with a universe of strict (definitionally proof-irrelevant) propositions, it is possible to define a univalent universe of propositions satisfying propositional extensionality. The setoid model thus satisfies all the extensionality principles that we would like to have in type theory. The question is whether there exists a version of intensional type theory that supports setoid reasoning, and hence the forms of extensionality enabled by it.

This question was revisited and answered in Altenkirch et al. [5]. In this paper, the authors define Setoid Type Theory (SeTT), an extension of intensional Martin-Löf type theory with constructs for setoid reasoning, where funext and propext hold by definition. SeTT is based on the *strict* setoid model of Altenkirch[5], which makes it possible to show consistency via a syntactic translation. This is in contrast with other type theories based on the setoid model, like Observational Type Theory [9,9] and XTT [26], which instead rely on ETT for their justification. A major property of SeTT is thus to illustrate how to bootstrap extensionality, by translation into a small intensional core.

SeTT as defined in [5] is already a rich theory, but its introspection capabilities are currently lacking, as its universes are limited to propositions. We would like to internalise the notion of type in SeTT, thus extending the theory with a universe of setoids. This goal brings up several questions, one of which has to do with the notion of equality with which the universe should come equipped: the universe of setoids is itself a setoid (as any type is) so it certainly cannot be univalent, since setoids lack the necessary structure. Another issue is the way such universe can be justified by the setoid model, and in particular what principles are needed in the metatheory to do so.

Contributions This paper documents our work towards the construction of a universe of setoids inside the setoid model, and tries to answer these and other questions related to the design and implementation of this construction. Our main contribution is the construction of the universe in the model; this is given in steps, first as an inductive-recursive definition, which is then translated to an inductive-inductive definition, and subsequently to an inductive type. As a consequence, we show that we only need to assume indexed W-types and proof-irrelevant identity types in the metatheory (along with some obligatory basic tools like Σ and Π types) to construct the universe.

The universe constructions presented in this paper are, to our knowledge, the first examples of two kinds of data type reductions in an intensional metatheory: the first involving an inductive-recursive type which includes strict propositions, and the second involving an infinitary inductive-inductive type.

Finally, the mathematical contents of this paper have been formalized in the proof-assistant Agda (see [1]).

Structure of the paper We begin by describing the metatheory that we will use throughout the paper, in Section 2. In Section 3, after briefly recalling *categories with families* as an abstract notion of models of type theory, we outline

[5] a *strict* model is one where every equation holds definitionally

Altenkirch's setoid model as given in [5]. We then briefly discuss the rules of Setoid Type Theory in Section 3.2.

In Section 4 we discuss the setoid model and various design choices related to it. We then recall inductive-recursive universes, and the way they can be equivalently defined as a plain inductive definition, in Section 4.1. We then provide, in Section 4.2, a first complete definition of the setoid universe using a special form of induction-recursion. This form of induction-recursion is not known to be reducible to plain inductive types. Then we describe an alternative definition of the universe in Section 4.3, that does not rely on induction-recursion but instead on infinitary induction-induction. This inductive-inductive encoding of the universe is obtained from the inductive-recursive one, inspired by the method of Section 4.1. We end the series of universe constructions with Section 4.4, where we outline a purely inductive definition of the setoid universe, obtained from the inductive-inductive one.

We conclude the paper with a discussion of further work in Section 5.

1.1 Related work

The setoid model was first described in [17] in order to add extensionality principles to Type Theory such as function extensionality and propositional extensionality. A strict variant of the setoid model was given in [4] using a definitionally proof-irrelevant universe of propositions. Recently, support for such a universe was added to the proof-assistants Agda and Coq [16], allowing a full formalization of Altenkirch's setoid model. Setoid Type Theory (SeTT) is a recently developed formal system derived from this model construction [5]. Observational Type Theory (OTT) [9,9] is a syntax for the setoid model differing from SeTT for the use of a different notion of heterogeneous equality. Moreover, the consistency proof for OTT relies on Extensional Type Theory, whereas for SeTT it is obtained via a syntactic translation. XTT [26] is a cubical variant of OTT where the equality type is defined using an interval pretype. XTT's universes support universe induction, whereas it is left open whether the construction presented here supports this principle. Palmgren and Wilander [25] construct a setoid universe using a translation into constructive set theory. In contrast, we target a basic intensional type theory and hence also give a computational semantics.

The principle of propositional extensionality in the setoid model is an instance of Voevodsky's univalence axiom [27]. The cubical set model is a constructive model justifying this axiom [10]. A type theory extracted from this model is Cubical Type Theory [12]. The relationship between the cubical set model and cubical type theory is similar to that between the setoid model and SeTT. Compared to cubical type theories, SeTT has the advantage that the equality type satisfies more definitional equalities. For instance, whereas in cubical type theory equality of functions is isomorphic to pointwise equality, in SeTT the isomorphism is replaced by a definitional equality. SeTT is also a syntactically straightforward extension of Martin-Löf Type Theory, that does not require exotic objects like the interval pretype. In turn, the obvious advantage of cubical type theory is that it is not limited to setoids.

An exceptional aspect of the metatheory used in this paper is the presence of a proof-irrelevant identity type with a strong transport rule allowing to eliminate into arbitrary types. In [2], Abel gives a proof of normalization for the Logical Framework extended with a similar proof-irrelevant equality type. Abel and Coquand show in [3] that the combination of impredicativity with a strong transport rule results in terms that fail to normalize.

2 $\mathsf{MLTT}^{\mathbf{Prop}}$

This section describes $\mathsf{MLTT}^{\mathbf{Prop}}$, our ambient metatheory. We employ Agda notation to write down $\mathsf{MLTT}^{\mathbf{Prop}}$ terms throughout the paper.

One of the main appeals of Altenkirch's setoid model is that it can justify several useful extensionality principles while being defined in a small intensional metatheory. We tried to stay true to this idea when figuring out the necessary metatheoretical tools for the universe construction in this paper. In particular, we wanted to avoid having to assume strong definition schemas that go beyond inductive families.

$\mathsf{MLTT}^{\mathbf{Prop}}$ is thus an intensional type theory in the style of Martin-Löf type theory with $\Pi, \Sigma, \mathsf{Bool}, \mathbf{0}, \mathbf{1}$, and a universe of strict propositions $\mathbf{Prop}$. Σ types are defined negatively by pairing $\_,\_$ and projections π_1, π_2. We also have definitional η rules for Π, Σ, and $\mathbf{1}$ types. Propositions can be lifted to types via a lifting $\mathsf{Lift} : \mathbf{Prop} \to \mathbf{Type}$, with constructor $\mathsf{lift} : \{P : \mathbf{Prop}\} \to P \to \mathsf{Lift}\ P$ and destructor $\mathsf{unlift} : \{P : \mathbf{Prop}\} \to \mathsf{Lift}\ P \to P$.

We also require indexed W-types, both in $\mathbf{Type}$ and $\mathbf{Prop}$; hence we have $W_\square : (S : I \to \mathbf{Type}) \to ((i : I) \to S\ i \to I \to \mathbf{Type}) \to I \to \square$ where $\square \in \{\mathbf{Type}, \mathbf{Prop}\}$. The elimination principle of $W_{\mathbf{Prop}}$ only allows defining functions into $\mathbf{Prop}$s. From $W_{\mathbf{Prop}}$ we can define propositional truncation $\|\_\| : \mathbf{Type} \to \mathbf{Prop}$, with constructor $|\_| : \{A : \mathbf{Type}\} \to A \to \|A\|$ and eliminator $\mathsf{elim}_{\|\_\|} : \{P : \mathbf{Prop}\} \to (A \to P) \to \|A\| \to P$.

In addition to type formers in $\mathbf{Type}$, we will need the propositional versions of $\mathbf{0}$, $\mathbf{1}$, Π, and Σ. The latter three can be defined from their $\mathbf{Type}$ counterparts via truncation. That is, given $P : \mathbf{Prop}$ and $Q : P \to \mathbf{Prop}$:

$$\begin{aligned} \mathbf{1}_{\mathbf{Prop}} &:\equiv \|\mathbf{1}\| \\ \Pi_{\mathbf{Prop}}\ P\ Q &:\equiv \|\Pi\ (\mathsf{Lift}\ P)\ (\mathsf{Lift} \circ Q \circ \mathsf{unlift})\| \\ \Sigma_{\mathbf{Prop}}\ P\ Q &:\equiv \|\Sigma\ (\mathsf{Lift}\ P)\ (\mathsf{Lift} \circ Q \circ \mathsf{unlift})\| \end{aligned}$$

We assume that we have $\mathbf{0}_{\mathbf{Prop}} : \mathbf{Prop}$ together with $\mathsf{exfalso}_{\mathbf{Prop}} : \{A : \mathbf{Type}\} \to \mathbf{0}_{\mathbf{Prop}} \to A$.

Finally, we will assume an identity type in the style of Martin-Löf's inductive identity type. The main difference is that our identity type is a $\mathbf{Prop}$-valued relation. We have a transport combinator transp from which J is derivable.

$$\begin{array}{ll}
\mathsf{Id} & : \{A : \mathbf{Type}\} \to A \to A \to \mathbf{Prop} \\
\mathsf{refl} & : \{A : \mathbf{Type}\}(a : A) \to \mathsf{Id}\ a\ a \\
\mathsf{transp} & : \{A : \mathbf{Type}\}(C : A \to \mathbf{Type})\{a_0\ a_1 : A\} \to \mathsf{Id}\ a_0\ a_1 \to C\ a_0 \to C\ a_1 \\
 & \quad \mathsf{transp}\ C\ \{x\}\ \{x\}\ e\ u \equiv u
\end{array}$$

The transp combinator provides a strong elimination principle allowing to eliminate a strict proposition (the identity type) into arbitrary types. We only use this identity type with such a strong transport in Section 4.4.

2.1 Formalization

A universe of strict propositions has been recently added to the Agda proof assistant [16], making *most* of $\mathsf{MLTT}^{\mathbf{Prop}}$ a subset of Agda, with the exception of the proof-irrelevant identity type. Most of the universe constructions presented here have been formalized and proof-checked using Agda, with the proof-irrelevant identity type and the strong transport rule added via postulates and rewriting. The formalization can be found in [1].

For convenience, we slightly deviate from $\mathsf{MLTT}^{\mathbf{Prop}}$ both in the paper and in the formalization, for instance by relying on pattern matching instead of eliminators, and using primitive versions of **Prop**-valued Π and Σ types instead of deriving them from truncation. We operate under the assumption that everything can be equivalently carried out in $\mathsf{MLTT}^{\mathbf{Prop}}$, although we haven't fully checked all the necessary details.

3 Setoid model

By *setoid model* we mean a class of models of type theory where contexts/closed types are interpreted as setoids, i.e. sets with an equivalence relation, and dependent types are interpreted as dependent/indexed setoids. A setoid model was first given for intensional type theory by M. Hofmann [17], in order to provide a semantics for extensionality principles such as function and propositional extensionality.

Here we consider a similar model construction due to Altenkirch [4]. The peculiarity of this model is that it is presented in a type theoretic and intensional metatheory which includes a strict universe of propositions.

The setoid model thus defined validates function extensionality, a universe of propositions with propositional extensionality, and quotient types. Therefore, it provides a way to bootstrap and "explain" extensionality, since the model construction effectively gives an implementation of various extensionality principles in terms of a small, completely intensional theory.

3.1 Setoid model as a CwF

The setoid model can be framed categorically as a category with families (CwF, [14]) with extra structure for the various type and term formers. The core structure of a CwF can be given as the following signature:

$$
\begin{aligned}
\mathrm{Con} &: \mathbf{Type} \\
\mathrm{Ty} &: (\Gamma : \mathrm{Con}) \to \mathbf{Type} \\
\mathrm{Sub} &: (\Gamma\ \Delta : \mathrm{Con}) \to \mathbf{Type} \\
\mathrm{Tm} &: (\Gamma : \mathrm{Con}) \to \mathrm{Ty}\ \Gamma \to \mathbf{Type}
\end{aligned}
$$

In our presentation of the setoid model, contexts are given by setoids, that is, types together with an equivalence relation. A key point of the model is that the equivalence relation is valued in **Prop** and is thus definitionally proof irrelevant.

$$
\begin{array}{l}
\hfil \Gamma : \mathrm{Con} \\ \hline
|\Gamma| : \mathbf{Type} \\
\Gamma^{\sim} : |\Gamma| \to |\Gamma| \to \mathbf{Prop} \\
\mathsf{refl}\ \Gamma : (\gamma : |\Gamma|) \to \Gamma^{\sim}\ \gamma\ \gamma \\
\mathsf{sym}\ \Gamma : \forall\{\gamma_0\ \gamma_1\} \to \Gamma^{\sim}\ \gamma_0\ \gamma_1 \to \Gamma^{\sim}\ \gamma_1\ \gamma_0 \\
\mathsf{trans}\ \Gamma : \forall\{\gamma_0\ \gamma_1\ \gamma_2\} \to \Gamma^{\sim}\ \gamma_0\ \gamma_1 \to \Gamma^{\sim}\ \gamma_1\ \gamma_2 \to \Gamma^{\sim}\ \gamma_0\ \gamma_2
\end{array}
$$

Types in a context Γ are given by displayed setoids over Γ with a fibration condition given by coe, coh. In the following, we sometimes omit implicit quantifications such as the $\forall\{\gamma_0\ \gamma_1\}$ in the type of $\mathsf{sym}\,\Gamma$.

$$
\begin{array}{l}
\hfil A : \mathrm{Ty}\ \Gamma \\ \hline
|A| : |\Gamma| \to \mathbf{Type} \\
A^{\sim} : \{\gamma_0\ \gamma_1 : |\Gamma|\} \to \Gamma^{\sim}\ \gamma_0\ \gamma_1 \to |A|\gamma_0 \to |A|\gamma_1 \to \mathbf{Prop} \\
\mathsf{refl}^* : \{\gamma : |\Gamma|\}(a : |A|\gamma) \to A^{\sim}\ (\mathsf{refl}\ \Gamma\ \gamma)\ a\ a \\
\mathsf{sym}^* : \forall\{\gamma_0\,\gamma_1\,a_0\,a_1\}\{p : \Gamma^{\sim}\ \gamma_0\ \gamma_1\} \to A^{\sim}\ p\ a_0\ a_1 \to A^{\sim}\ (\mathsf{sym}\ \Gamma\ p)\ a_1\ a_0 \\
\mathsf{trans}^* : A^{\sim}\ p_0\ a_0\ a_1 \to A^{\sim}\ p_1\ a_1\ a_2 \to A^{\sim}\ (\mathsf{trans}\,\Gamma\,p_0\,p_1)\ a_0\ a_2 \\
\mathsf{coe} : \Gamma^{\sim}\ \gamma_0\ \gamma_1 \to |A|\gamma_0 \to |A|\gamma_1 \\
\mathsf{coh} : (p : \Gamma^{\sim}\ \gamma_0\ \gamma_1)(a : |A|\gamma_0) \to A^{\sim}\,p\,a\,(\mathsf{coe}\,A\,p\,a)
\end{array}
$$

This definition of types in the setoid model is different from the one in [4], but it is equivalent to it [11, Section 1.6.1]. The main difference here is in the use of a heterogeneous equivalence relation $A^{\sim}$ in the definition of types.

Substitutions are interpreted as functors between the corresponding setoids, whereas terms of type A in context Γ are sections of the type seen as a setoid fibration $\Gamma.A \to \Gamma$. Note that we only need to include components for the functorial action on objects and morphisms, since the functor laws follow from proof-irrelevance in the metatheory, and thus hold definitionally.

$$
\begin{array}{l}
\hfil \sigma : \mathrm{Sub}\ \Gamma\ \Delta \\ \hline
|\sigma| : |\Gamma| \to |\Delta| \\
\sigma^{\sim} : \Gamma^{\sim}\,\rho_0\,\rho_1 \to \Delta^{\sim}\,(|\sigma|\rho_0)\,(|\sigma|\rho_1)
\end{array}
\qquad
\begin{array}{l}
\hfil t : \mathrm{Tm}\ \Gamma\ A \\ \hline
|t| : (\gamma : |\Gamma|) \to |A|\,\gamma \\
t^{\sim} : (p : \Gamma^{\sim}\ \gamma_0\ \gamma_1) \to A^{\sim}\ p\ (|t|\gamma_0)\ (|t|\gamma_1)
\end{array}
$$

We can show that the setoid model validates the usual basic type formers (Π, Σ, etc.), function extensionality and a universe of strict propositions with propositional extensionality [4]. Note that we don't need identity types or inductive types (W-types) for this.

3.2 Setoid Type Theory

The setoid model presented in the previous section is *strict*, that is, every equation of a CwF holds by definition in the semantics. One advantage of strict models is that they can be turned into *syntactic translations*, in which syntactic objects of the source theory are interpreted as their counterparts in another *target* theory. In the case of the setoid model, this gives rise to a *setoid translation*, where source contexts are interpreted as target contexts together with a target type representing the equivalence relation, and so on.[6]

A setoid translation is used in [5] to justify Setoid Type Theory (SeTT), an extension of Martin-Löf type theory (+ **Prop**) with equality types for contexts and dependent types that reflect the setoid equality of the model.

We recall the rules of SeTT that extend regular MLTT below, but with a variation: whereas the equality types in [5] are stated as elements of SeTT's internal universe of propositions, here we state the context equalities as elements of the external, metatheoretic universe **Prop**. This makes it easier to define models of SeTT as contexts do not need to be partially fibrant. Equality on types is defined as before in [5].

We have a universe of propositions Prop defined as follows:

$$\frac{\Gamma : \mathrm{Con}}{\mathsf{Prop} : \mathrm{Ty}\ \Gamma} \qquad \frac{P : \mathrm{Tm}\ \Gamma\ \mathsf{Prop}}{\underline{P} : \mathrm{Ty}\ \Gamma} \qquad \frac{u : \mathrm{Tm}\ \Gamma\ \underline{P} \qquad v : \mathrm{Tm}\ \Gamma\ \underline{P}}{u \equiv v}$$

Equality type constructors for contexts and dependent types internalize the idea that every context and type comes equipped with a setoid equivalence relation. Note that **Prop** is the universe of the metatheory while Prop is the internal one. As in the model, equality for dependent types is indexed over context equality.

$$\frac{\Gamma : \mathrm{Con} \qquad \rho_0, \rho_1 : \mathrm{Sub}\ \Delta\ \Gamma}{\Gamma^{\sim}\ \rho_0\ \rho_1 : \mathbf{Prop}} \qquad \frac{\begin{array}{c} A : \mathrm{Ty}\ \Gamma \qquad \rho_{01} : \Gamma^{\sim}\ \rho_0\ \rho_1 \\ a_0 : \mathrm{Tm}\ \Delta\ A[\rho_0] \qquad a_1 : \mathrm{Tm}\ \Delta\ A[\rho_1] \end{array}}{A^{\sim}\ \rho_{01}\ a_0\ a_1 : \mathrm{Tm}\ \Delta\ \mathsf{Prop}}$$

We have rules witnessing that these are indeed equivalence relations. We only recall reflexivity:

$$\frac{\rho : \mathrm{Sub}\ \Delta\ \Gamma}{\mathsf{R}\ \rho : \Gamma^{\sim}\ \rho\ \rho} \qquad \frac{A : \mathrm{Ty}\ \Gamma \qquad \rho : \mathrm{Sub}\ \Delta\ \Gamma \qquad a : \mathrm{Tm}\ \Delta\ A[\rho]}{\mathsf{R}\ a : \mathrm{Tm}\ \Gamma\ \underline{A^{\sim}\ (\mathsf{R}\ \rho)\ a\ a}}$$

In addition, we also have rules representing the fact that every construction in SeTT respects setoid equality, so that we can transport along any such equality:

$$\frac{A : \mathrm{Ty}\ \Gamma \qquad \rho_0, \rho_1 : \mathrm{Sub}\ \Delta\ \Gamma \qquad p : \Gamma^{\sim}\ \rho_0\ \rho_1 \qquad a : \mathrm{Tm}\ \Delta\ A[\rho_0]}{\begin{array}{c} \mathsf{coe}_A\ p\ a : \mathrm{Tm}\ \Delta\ A[\rho_1] \\ \mathsf{coh}_A\ p\ a : \mathrm{Tm}\ \Delta\ \underline{A^{\sim}\ p\ a\ (\mathsf{coe}_A\ p\ a)} \end{array}}$$

[6] Semantically, this translation corresponds to a model construction, in particular a functor from the category of models of the target theory to the category of models of what will be Setoid Type Theory. Since the setoid translation is structural in the context component, we can work with models in the style of categories with families rather than contextual categories.

Notably, equality types in SeTT compute definitionally on concrete type formers. In particular, they compute to their obvious intended meaning, so that an equality of pairs is a pair of equalities, an equality of functions is a map of equalities, and so on. From this, we get definitional versions of function and propositional extensionality.

We can easily recover the usual Martin-Löf identity type from setoid equality, with transport implemented via coercion.

$$\frac{A : \mathrm{Ty}\ \Gamma \qquad a_0, a_1 : \mathrm{Tm}\ \Gamma\ A}{\mathsf{Id}_A\ a_0\ a_1 :\equiv A^{\sim}\ (\mathsf{R}\ \Gamma)\ a_0\ a_1 : \mathrm{Tm}\ \Gamma\ \mathsf{Prop}}$$

$$\frac{P : \mathrm{Ty}\ (\Gamma.A) \qquad p : \mathrm{Tm}\ \Gamma\ (\mathsf{Id}\ A\ a_0\ a_1) \qquad t : \mathrm{Tm}\ \Gamma\ P[a_0]}{\mathsf{transp}\ P\ p\ t :\equiv \mathsf{coe}\ P\ (\mathsf{R}\ \mathsf{id}, p)\ t : \mathrm{Tm}\ \Gamma\ P[a_1]}$$

We can also derive Martin-Löf's J eliminator for this homogeneous identity type. The only caveat is that transp and the J eliminator do not compute definitionally on reflexivity.

4 Universe of setoids

As pointed out in the introduction, SeTT is seriously limited by the lack of a universes internalizing the notion of setoid. Our goal is to extend SeTT with a universe of setoids; since SeTT is a direct syntactic reflection of the setoid model, this essentially amounts to showing that a universe of setoids with the necessary structure and equations can be constructed within the setoid model. This opens several questions and possible design choices.

A first fundamental consideration has to do with the very definition of the setoid universe: as any type in the setoid model, this universe must be a setoid and thus come equipped with an equivalence relation. However, unlike the universe of propositions, a universe of setoids cannot be univalent, since this would force it to be a groupoid. The obvious choice is therefore to have a non-univalent universe, and instead define the universe's relation so that it reflects a simple syntactic equality of codes rather than setoid equivalence.

Another question has to do with the metatheoretic tools required to carry out the construction of the universe. In fact, one of the main aspects of the setoid model construction recalled in Section 3 and shown originally in [4] is that it can be carried out in a very small type theoretic metatheory, thus providing a way to reduce extensionality to a small intensional core. We would like to stay faithful to this ideal when constructing this setoid universe.

A known and established method for defining universes in type theory relies on induction-recursion (IR), a definition schema developed by Dybjer [13,15]. Inductive-recursive definitions can be found throughout the literature, from the already mentioned type theoretic universes, including the original formulation à la Tarski by Martin-Löf [23], to metamathematical tools like computability predicates.

Although universe constructions in type theory—including our own setoid universe—are naturally presented as inductive-recursive definitions, they may not necessarily require a metatheory with induction-recursion. In fact, it is possible to reduce some instances of induction-recursion to plain induction (more specifically, inductive families), including some universe definitions. We recall this reduction in Section 4.1.

Other design choices on the setoid universe are less essential, but still require careful consideration. For instance, one question is whether the setoid universe should support universe induction, thus exposing the inductive structure of the codes. Such elimination principle is known to be inconsistent with univalence, although this is not an issue in our case; nevertheless it's not immediately clear if the elimination principle can be justified by the semantics, that is, if our encoding of the setoid universe in the model allows to define such a universe eliminator. The question arises because our final encoding of the setoid universe only supports a weak form of elimination, for reasons that are explained in Section 4.4. Although not currently needed, a stronger eliminator might be necessary to justify universe induction. This problem should not arise in the other encodings of the setoid universe (as given in Section 4.2 and Section 4.3).

Another design choice has to do with how the setoid universe relates to the other universes. One could provide a code for Prop in the setoid universe. Moreover, the setoid universes could form a hierarchy, possibly cumulative.

Yet another choice is whether to have two separate sorts, one for propositions and one for sets (with propositions convertible to sets) or a single sort of types (sets), with propositions given by elements of a universe of proposition, which is a (large) type. We have chosen to present the second option to fit with the standard notion of (unisorted) CwF. However, this has downsides: to even talk about propositions, we need to have a notion of large types. The first option is more symmetric: we can have parallel hierarchies for propositions and sets.

4.1 Inductive-recursive universes

An inductive-recursive universe is given by a type of codes $\mathsf{U} : \mathbf{Type}$, and a family $\mathsf{El} : \mathsf{U} \to \mathbf{Type}$ that assigns, to each code corresponding to some type, the meta-theoretic type of its elements. The resulting definition is inductive-recursive because the inductive type of codes is defined simultaneously with the recursive function El.

As an example, consider the following definition of a small universe with bool and Π.

$$\begin{array}{ll}
\mathsf{data}\ \mathsf{U}\ : \mathbf{Type} & \mathsf{El}\ : \mathsf{U} \to \mathbf{Type} \\
\quad \mathsf{bool} : \mathsf{U} & \mathsf{El}\ \mathsf{bool} :\equiv \mathbb{2} \\
\quad \mathsf{pi} : (A : \mathsf{U}) \to (\mathsf{El}\ A \to \mathsf{U}) \to \mathsf{U} & \mathsf{El}\ (\mathsf{pi}\ A\ B) :\equiv (a : \mathsf{El}\ A) \to \mathsf{El}\ (B\ a)
\end{array}$$

Induction-recursion is arguably a nice and natural way to define internal universes in type theory, however it is not always strictly required. We can translate basic instances of induction-recursion into inductive families using the equivalence of I-indexed families of types and types over I (that is, $\mathbf{Type}_i \to I$) [21].

In our case, we can encode U as an inductive type inU that *carves out* all types in **Type** that are in the image of El. In other words, inU is a predicate that holds for any type that would have been obtained via El in the inductive-recursive definition. As El is indexed by the type of codes, the definition of inU quite expectedly reflects the inductive structure of codes.

$$\begin{aligned}
&\mathsf{data\ inU} : \mathbf{Type} \to \mathbf{Type}_1\\
&\quad \mathsf{inBool} : \mathsf{in\text{-}U}\ \mathbb{2}\\
&\quad \mathsf{inPi} \quad : \mathsf{inU}\ A \to ((a : A) \to \mathsf{inU}\ (B\ a)) \to \mathsf{inU}\ ((a : A) \to (B\ a))
\end{aligned}$$

U and El can be given by $\mathsf{U} :\equiv \Sigma\ (A : \mathbf{Type})\ (\mathsf{in\text{-}U}\ A)$ and $\mathsf{El} :\equiv \pi_1$.

Note that this construction gives rise to a universe in $\mathbf{Type}_1$, rather than **Type**, since the definition of U quantifies over all possible types in **Type**. Hence this kind of construction requires a metatheory with at least one universe.

4.2 Inductive-recursive setoid universe

In this section we give a first definition of the setoid universe, as a direct generalization of the simple inductive-recursive definition just shown. We only consider a very small universe with bool type $\mathbb{2}$ and Π for simplicity; a more realistic universe that includes more type formers can be found in the Agda formalization.

To construct the universe of setoids in the setoid model, we first of all need to define a type $\mathbb{U} : \mathrm{Ty}\ \Gamma$ for every $\Gamma : \mathrm{Con}$, and for every $A : \mathrm{Tm}\ \Gamma\ \mathbb{U}$ a type $\mathbb{E}l\ A : \mathrm{Ty}\ \Gamma$. Recalling Section 3, these are essentially record types made of several components. Since $\mathbb{U}$ is a closed type, it requires the same data of a setoid; in particular, we need a type of codes together with an equivalence relation reflecting equality of codes, in addition to proofs that these are indeed equivalence relations:

$$\begin{aligned}
&\mathsf{data}\ \mathcal{U} : \mathbf{Type}_1\\
&\_\sim_{\mathcal{U}}\_ : \mathcal{U} \to \mathcal{U} \to \mathbf{Prop}_1
\end{aligned}
\qquad
\begin{aligned}
&\mathsf{refl}_{\mathcal{U}} : (A : \mathcal{U}) \to A \sim_{\mathcal{U}} A\\
&\mathsf{sym}_{\mathcal{U}} : A \sim_{\mathcal{U}} B \to B \sim_{\mathcal{U}} A\\
&\mathsf{trans}_{\mathcal{U}} : A \sim_{\mathcal{U}} B \to B \sim_{\mathcal{U}} C \to A \sim_{\mathcal{U}} C
\end{aligned}$$

$\mathbb{E}l$ is given by a family of setoids indexed over the universe, that is, a way to assign to each code in the universe a carrier set and an equivalence relation.

$$\begin{aligned}
&\mathsf{El} : \mathcal{U} \to \mathbf{Type}\\
&\_\vdash\_\sim_{\mathsf{El}}\_ : \{a\ a' : \mathcal{U}\} \to a \sim_{\mathcal{U}} a' \to \mathsf{El}\ a \to \mathsf{El}\ a' \to \mathbf{Prop}
\end{aligned}$$

Note that $\_\vdash\_\sim_{\mathsf{El}}\_$ is indexed over equality on the universe, because El is a displayed setoid over $\mathcal{U}$, hence in particular it must respect the setoid equality of $\mathcal{U}$. We also require data and proofs that make sure we get setoids out of El:

$$\begin{aligned}
&\mathsf{refl}_{\mathsf{El}} : (A : \mathcal{U})(x : \mathsf{El}\ A) \to \mathsf{refl}_{\mathcal{U}}\ A \vdash x \sim_{\mathsf{El}} x \\
&\mathsf{sym}_{\mathsf{El}} : p \vdash x \sim_{\mathsf{El}} x' \to \mathsf{sym}_{\mathcal{U}}\ p \vdash x' \sim_{\mathsf{El}} x \\
&\mathsf{trans}_{\mathsf{El}} : p \vdash x \sim_{\mathsf{El}} x' \to q \vdash x' \sim_{\mathsf{El}} x'' \to \mathsf{trans}_{\mathcal{U}}\ p\ q \vdash x \sim_{\mathsf{El}} x'' \\
&\mathsf{coe}_{\mathsf{El}} : A \sim_{\mathcal{U}} B \to \mathsf{El}\ A \to \mathsf{El}\ B \\
&\mathsf{coh}_{\mathsf{El}} : (p : A \sim_{\mathcal{U}} A')\ (x : \mathsf{El}\ A) \to p \vdash x \sim_{\mathsf{El}} \mathsf{coe}_{\mathsf{El}}\ p\ x
\end{aligned}$$

We give an inductive definition of $\mathcal{U}$, mutually with a recursive definition of the 4 functions $\_ \sim_{\mathcal{U}} \_$, $\mathsf{refl}_{\mathcal{U}}$, El and $\_ \vdash \_ \sim_{\mathsf{El}} \_$. The other functions are then recursively defined:

- $\mathsf{refl}_{\mathsf{El}}$ alone,
- $\mathsf{sym}_{\mathcal{U}}$ and $\mathsf{sym}_{\mathsf{El}}$ mutually,
- $\mathsf{trans}_{\mathcal{U}}$, $\mathsf{trans}_{\mathsf{El}}$, $\mathsf{coe}_{\mathsf{El}}$ and $\mathsf{coh}_{\mathsf{El}}$ mutually.

The whole construction is quite long, below we only show the more interesting definitions of $\mathcal{U}$ and El:

$$\begin{array}{ll}
\mathsf{data}\ \mathcal{U} : \mathbf{Type}_1 & \mathsf{El}\ \mathsf{bool} :\equiv \mathbb{2} \\
\quad \mathsf{bool} : \mathcal{U} & \mathsf{El}\ (\mathsf{pi}\ A\ B\ h) :\equiv \\
\quad \mathsf{pi} : (A : \mathcal{U})(B : \mathsf{El}\ A \to \mathcal{U}) & \quad \Sigma\ (f : (a : \mathsf{El}\ A) \to \mathsf{El}\ (B\ a)) \\
\qquad \to (\{x\ x' : \mathsf{El}\ A\} \to \mathsf{refl}_{\mathcal{U}}\ A \vdash x \sim_{\mathsf{El}} x' & \qquad (\forall \{x\ x'\}(p : \mathsf{refl}_{\mathcal{U}}\ A \vdash x \sim_{\mathsf{El}} x') \\
\qquad\quad \to B\ x \sim_{\mathcal{U}} B\ x') \to \mathcal{U} & \qquad\quad \to h\ p \vdash f\ x \sim_{\mathsf{El}} f\ x')
\end{array}$$

Note that in the definition of $\mathcal{U}$ we require that the family $B : \mathsf{El}\ A \to \mathcal{U}$ be a setoid morphism, respecting the setoid equalities involved. This choice is crucial for the definition of El to go through, in particular since we eliminate the code for Π types into the setoid of functions that map equal elements to equal results. To state this mapping property we need to compare elements in different types, coming from applying f to different arguments x and x'. We know that x and x' are equal, but to conclude $B\ x \sim_{\mathcal{U}} B\ x'$ we need to know that B respects setoid equality. This is exactly what we get from our definition of $\mathcal{U}$.

We can now give a full definition of the setoid universe, and of $\mathbb{E}l\ A$ for any $A : \mathrm{Tm}\ \Gamma\ \mathbb{U}$:

$$\begin{array}{ll}
|\mathbb{U}| :\equiv \lambda\,\gamma.\,\mathcal{U} & |\mathbb{E}l\ A| :\equiv \lambda\,\gamma.\,\mathsf{El}\ (|A|\,\gamma) \\
\mathbb{U}^{\sim} :\equiv \lambda\,p\,x\,y.\,x \sim_{\mathcal{U}} y & (\mathbb{E}l\ A)^{\sim} :\equiv \lambda\,p\,x\,y.\,A^{\sim}\,p \vdash x \sim_{\mathsf{El}} y \\
\mathsf{refl}\ \mathbb{U} :\equiv \mathsf{refl}_{\mathcal{U}} & \mathsf{refl}\ (\mathbb{E}l\ A) :\equiv \mathsf{refl}_{\mathsf{El}} \\
\ldots & \ldots \\
\mathsf{coe}\ \mathbb{U} :\equiv \lambda\,p\,a.\,a & \mathsf{coe}\ (\mathbb{E}l\ A) :\equiv \lambda\,p.\,\mathsf{coe}_{\mathsf{El}}\,(A^{\sim}\,p) \\
\mathsf{coh}\ \mathbb{U} :\equiv \lambda\,p.\,\mathsf{refl}_{\mathcal{U}} & \mathsf{coh}\ (\mathbb{E}l\ A) :\equiv \lambda\,p.\,\mathsf{coh}_{\mathsf{El}}\,(A^{\sim}\,p)
\end{array}$$

We can show that $\mathbb{U}$ is closed under Π types and booleans, and satisfies $\mathbb{E}l\,(\mathsf{pi}\,A\,B) \equiv \Pi\,(\mathbb{E}l\,A)\,(\mathbb{E}l\,B)$ and $\mathbb{E}l\,\mathsf{bool} = \mathsf{Bool}$. The universe can be closed under more constructions if more codes are added to $\mathcal{U}$.

This gives a complete definition of a universe of setoids, which is, however, inductive-recursive. Moreover, the kind of recursion involved in this definition is particularly complex, and not obviously reducible to well-understood notions of induction-recursion like the one described in [15]. In any case, we would like to avoid extending the metatheory with any form of induction-recursion in order to keep the metatheory as small and essential as possible.

In the next section we transform our current inductive-recursive definition to one that doesn't use induction-recursion. The way this is done is inspired by the well-known trick to eliminate induction-recursion described in Section 4.1, but modified in a novel way to account for the presence of **Prop**-valued types. To our knowledge, this is the first time this reduction method is applied to an inductive-recursive type of this kind.

4.3 Inductive-inductive setoid universe

We will follow the method outlined in Section 4.1. In addition to inU for defining U, we also introduce a family inU∼ of binary relations between types in the universe, from which we then define $\_ \sim_{\mathcal{U}} \_$.

```
data inU : $\mathbf{Type} \to \mathbf{Type}_1$
    bool : inU $\mathbb{2}$
    $\pi : \forall\{A\ A_\sim\ B\ B_\sim\}(a : \mathsf{inU}\ A)(b : (x : A) \to \mathsf{inU}\ (B\ x))$
      $\to \mathsf{inU}{\sim}\ a\ a\ A_\sim \to (\forall\{x_0\ x_1\}(x_{01} : A_\sim\ x_0\ x_1) \to \mathsf{inU}{\sim}\ (b\ x_0)\ (b\ x_1)\ (B_\sim\ x_{01}))$
      $\to \mathsf{inU}\ (\Sigma\ (f : (x : A) \to B\ x)$
                 $((x_0\ x_1 : A)(x_{01} : A_\sim\ x_0\ x_1) \to B_\sim\ x_{01}\ (f\ x_0)\ (f\ x_1)))$
data inU∼ : $\{A\ A' : \mathbf{Type}\} \to \mathsf{inU}\ A \to \mathsf{inU}\ A' \to (A \to A' \to \mathbf{Prop}) \to \mathbf{Type}_1$
    $\mathsf{bool}_\sim : \mathsf{inU}{\sim}\ \mathsf{bool}\ \mathsf{bool}\ (\lambda x_0\ x_1\ .\ x_0 \overset{?}{=}_{\mathbb{2}} x_1)$
    $\pi_\sim : \forall\{A_0\ A_1\ A_{0\sim}\ A_{1\sim}\ A_{01\sim}\ B_0\ B_1\ B_{0\sim}\ B_{1\sim}\ B_{01\sim}\}\{a_0 : \mathsf{inU}\ A_0\}\{a_1 : \mathsf{inU}\ A_1\}$
          $\{b_0 : (x_0 : A_0) \to \mathsf{inU}\ (B_0\ x_0)\}\{b_1 : (x_1 : A_1) \to \mathsf{inU}\ (B_1\ x_1)\}$
          $\{a_{0\sim} : \mathsf{inU}{\sim}\ a_0\ a_0\ A_{0\sim}\}\{a_{1\sim} : \mathsf{inU}{\sim}\ a_1\ a_1\ A_{1\sim}\}$
          $\{b_{0\sim} : \forall\{x_0\ x_1\}(x_{01} : A_{0\sim}\ x_0\ x_1) \to \mathsf{inU}{\sim}\ (b_0\ x_0)\ (b_0\ x_1)\ (B_{0\sim}\ x_{01})\}$
          $\{b_{1\sim} : \forall\{x_0\ x_1\}(x_{01} : A_{1\sim}\ x_0\ x_1) \to \mathsf{inU}{\sim}\ (b_1\ x_0)\ (b_1\ x_1)\ (B_{1\sim}\ x_{01})\}$
       $\to \mathsf{inU}{\sim}\ a_0\ a_1\ A_{01\sim}$
       $\to (\forall\{x_0\ x_1\}(x_{01} : A_{01\sim}\ x_0\ x_1) \to \mathsf{inU}{\sim}\ (b_0\ x_0)\ (b_1\ x_1)\ (B_{01\sim}\ x_{01}))$
       $\to \mathsf{inU}{\sim}\ (\pi\ a_0\ a_{0\sim}\ b_0\ b_{0\sim})\ (\pi\ a_1\ a_{1\sim}\ b_1\ b_{1\sim})$
                 $(\lambda f_0\ f_1\ .\ \forall(x_0\ x_1) \to A_{01\sim}\ x_0\ x_1 \to B_{01\sim}\ x_{01}\ (\pi_1\ f_0\ x_0)\ (\pi_1\ f_1\ x_1))$
```

Just as the role of inU is, as before, to classify all types that are image of El, in the same way inU∼ $a\ a'$ classifies all relations of type $A \to A' \to \mathbf{Prop}$ that are image of $\_ \vdash \_ \sim_{\mathsf{El}} \_$, given proofs $a : \mathsf{inU}\ A, a' : \mathsf{inU}\ A'$. In particular, this definition of inU∼ states that the appropriate equivalence for boolean elements is the obvious syntactic equality $\_ \overset{?}{=}_{\mathbb{2}} \_$, whereas functions are to be compared pointwise. Note that inU appears in the sort of $\mathsf{inU}\!\sim$. Since these types are mutually defined, they form an instance of *induction-induction*, a schema that allows the definition of a type mutually with other types that contain the first one in their signature [24].[7]

As in the universe example in Section 4.1, we now define $\mathcal{U}$ as a Σ type, and El as the corresponding first projection.

$$\begin{aligned} &\mathcal{U} : \mathbf{Type}_1 && \mathsf{El} : \mathcal{U} \to \mathbf{Type} \\ &\mathcal{U} :\equiv \Sigma\ (X : \mathbf{Type})\ (\mathsf{inU}\ X) && \mathsf{El} :\equiv \pi_1 \end{aligned}$$

What is left now is to define the setoid equality relation on the universe, as well as the setoid equality relation on $\mathsf{El}\ A$ for any A in $\mathcal{U}$. Two codes A, B in the universe $\mathcal{U}$ are equal when there exists a setoid equivalence relation on their respective sets $\mathsf{El}\ A$ and $\mathsf{El}\ B$. Intuitively, since elements of a setoid are only ever compared to elements of the same setoid, this should only be possible if A and B are codes for the same setoid, that is, if $A \sim_{\mathcal{U}} B$. Existence and well-formedness of such relations is expressed via the type $\mathsf{inU}\!\sim$ just defined, hence we would expect $A \sim_{\mathcal{U}} B$ to be defined as follows:

$$(A, a) \sim_{\mathcal{U}} (B, b) :\equiv \Sigma\ (R : A \to B \to \mathbf{Prop})\ (\mathsf{inU}\!\sim\ a\ b\ R)$$

Unfortunately this definition only manages to capture the idea, but does not actually typecheck. In fact, $\_ \sim_{\mathcal{U}} \_$ should be $\mathbf{Prop}_1$-valued relation, so $A \sim_{\mathcal{U}} B$ should be a proposition. However, the Σ type shown above clearly isn't, since it quantifies over a type of relations, which is not a proposition. One possible solution is actually quite simple, and it just involves truncating the Σ type above to force it to be in $\mathbf{Prop}_1$.

$$\begin{aligned} &\_ \sim_{\mathcal{U}} \_ : \mathcal{U} \to \mathcal{U} \to \mathbf{Prop}_1 \\ &(A, a) \sim_{\mathcal{U}} (B, b) = \parallel \Sigma\ (R : A \to B \to \mathbf{Prop})\ (\mathsf{inU}\!\sim\ a\ b\ R) \parallel \end{aligned}$$

We are now left to define the indexed equivalence relation on El:

$$\begin{aligned} &\_ \vdash \_ \sim_{\mathsf{El}} \_ : \{A\ B : \mathsf{U}\} \to A\ \sim_{\mathcal{U}}\ B \to \mathsf{El}\ A \to \mathsf{El}\ B \to \mathbf{Prop} \\ &p \vdash x \sim_{\mathsf{El}} y :\equiv ? \end{aligned}$$

In the definition above, p has type $\parallel \Sigma\ (R : \mathsf{El}\,A \to \mathsf{El}\,B \to \mathbf{Prop})\ (...) \parallel$. If the type wasn't propositionally truncated, we could define $p \vdash x \sim_{\mathsf{El}} y$ by extracting the relation out of the first component of p, and apply it to x, y. That is, $p \vdash x \sim_{\mathsf{El}} y :\equiv \pi_1\, p\, x\, y$. This would make the definition of $\_ \sim_{\mathcal{U}} \_$ and $\_ \vdash \_ \sim_{\mathsf{El}} \_$ in line with how we defined $\mathcal{U}$ and El.

However, this doesn't work in our case, since the type of p *is* propositionally truncated, hence it cannot be eliminated to construct a proof-relevant object. Fortunately, we can work around this limitation by defining $p \vdash x \sim_{\mathsf{El}} y$ by induction on the codes $A\ B : \mathcal{U}$, in a way that ends up being logically equivalent

[7] The main example of induction-induction is the intrinsic definition of a dependent type theory in type theory [6].

to the proposition we would have obtained by $\pi_1\ p\ x\ y$ if there were no truncation. More precisely, we need to construct proofs that for any concrete R and inR, the types $|(R, \mathsf{inR})| \vdash x \sim_{\mathsf{El}} y$ and $R\ x\ y$ are logically equivalent. These in turn need to be defined mutually with $\_ \vdash \_ \sim_{\mathsf{El}} \_$. We direct the interested reader to the Agda formalization for the full details of these definitions, as they are quite involved.

The full definition of the universe is concluded with the remaining definitions, like $\mathsf{refl}_{\mathcal{U}}, \mathsf{refl}_{\mathsf{El}}$, etc., which can be adapted from their IR counterparts more or less straightforwardly. The final result does not use induction-recursion, but it's nevertheless an instance of infinitary induction-induction. The ability to define arbitrary, infinitary inductive-inductive types clashes, again, with our objective of keeping the metatheory as small and simple as possible. The next step is therefore to reduce this inductive-inductive universe to one that does not require (infinitary) induction-induction.

4.4 Inductive setoid universe

This section outlines a way to encode the inductive-inductive universe of setoids from the previous section without assuming arbitrary inductive-inductive definitions in the metatheory.

Before turning our attention to the setoid universe, we recall the known, systematic method to reduce finitary inductive-inductive types to inductive families.

Reducing finitary induction-induction It is known that finitary inductive-inductive definitions can be reduced to inductive families [8,7,20]. To illustrate the idea, let us consider a well-known example of a finitary inductive-inductive type, the intrinsic encoding of type theory in type theory itself. Actually, we only consider the type of contexts $\mathsf{Con} : \mathbf{Type}$ and the type of types $\mathsf{Ty} : \mathsf{Con} \to \mathbf{Type}$; since the latter is indexed over the former, this is already an example of induction-induction.

Contexts in Con are formed out of empty contexts $\bullet$ and context extension $\_,\_$. Types in Ty are either the base type ι or Π types.

$$\begin{array}{ll}
\bullet\ \ : \mathsf{Con} & \iota\ \ : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\ \Gamma \\
\_,\_ : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\ \Gamma \to \mathsf{Con} & \Pi : \{\Gamma : \mathsf{Con}\}(A : \mathsf{Ty}\ \Gamma) \to \mathsf{Ty}\ (\Gamma, A) \to \mathsf{Ty}\ \Gamma
\end{array}$$

The general method to eliminate induction-induction is to split the original inductive-inductive types into a type of codes and associated well-formedness predicates. In our Con/Ty example, these would be respectively given by codes $\mathsf{Con}_0, \mathsf{Ty}_0 : \mathbf{Type}$ and predicates $\mathsf{Con}_1 : \mathsf{Con}_0 \to \mathbf{Type}, \mathsf{Ty}_1 : \mathsf{Con}_0 \to \mathsf{Ty}_0 \to \mathbf{Type}$.

The definition of the codes and predicate types follows that of the original inductive-inductive type, and can be derived systematically from it. More importantly, they can be defined without induction-induction, since although Con_0 and Ty_0 are defined mutually, their sorts are not indexed.

$$\begin{aligned}
\bullet_0 &: \mathsf{Con}_0 \\
\_,_0\_ &: \mathsf{Con}_0 \to \mathsf{Ty}_0 \to \mathsf{Con}_0 \\
\iota_0 &: \mathsf{Con}_0 \to \mathsf{Ty}_0 \\
\Pi_0 &: \mathsf{Con}_0 \to \mathsf{Ty}_0 \to \mathsf{Ty}_0 \to \mathsf{Ty}_0
\end{aligned}$$

$$\begin{aligned}
\bullet_1 &: \mathsf{Con}_1\ \bullet_0 \\
\_,_1\_ &: \forall\{\Gamma_0\ A_0\} \to \mathsf{Con}_1\ \Gamma_0 \to \mathsf{Ty}_1\ \Gamma_0\ A_0 \\
&\quad \to \mathsf{Con}_1\ (\Gamma_0 ,_0 A_0) \\
\iota_1 &: \forall\{\Gamma_0\} \to \mathsf{Con}_1\ \Gamma_0 \to \mathsf{Ty}_1\ \Gamma_0\ (\iota_0\ \Gamma_0) \\
\Pi_1 &: \forall\{\Gamma_0\ A_0\ B_0\} \to \mathsf{Con}_1\ \Gamma_0 \\
&\quad \to \mathsf{Ty}_1\ \Gamma_0\ A_0 \to \mathsf{Ty}_1\ (\Gamma_0 ,_0 A_0)\ B_0 \\
&\quad \to \mathsf{Ty}_1\ \Gamma_0\ (\Pi_0\ \Gamma_0\ A_0\ B_0)
\end{aligned}$$

We can recover the original inductive-inductive type as $\mathsf{Con} :\equiv \Sigma\ (\Gamma_0 : \mathsf{Con}_0)\ (\mathsf{Con}_1\ \Gamma_0)$ and $\mathsf{Ty}\ \Gamma :\equiv \Sigma\ (A_0 : \mathsf{Ty}_0)\ (\mathsf{Ty}_1\ (\pi_1\ \Gamma)\ A_0)$. Recovering the constructors is straightforward:

$$\begin{aligned}
\bullet &:\equiv (\bullet_0, \bullet_1) \\
(\Gamma_0, \Gamma_1), (A_0, A_1) &:\equiv ((\Gamma_0 ,_0 A_0), (\Gamma_1 ,_1 A_1)) \\
\iota\ (\Gamma_0, \Gamma_1) &:\equiv (\iota_0\ \Gamma_0, \iota_1\ \Gamma_1) \\
\Pi\ \{\Gamma_0, \Gamma_1\}(A_0, A_1)(B_0, B_1) &:\equiv (\Pi_0\ \Gamma_0\ A_0\ B_0, \Pi_1\ \Gamma_1\ A_1\ B_1)
\end{aligned}$$

Finally, we can define eliminators/induction principles for Con and Ty as just defined, by induction on the well-typing predicates.

For any inductive-inductive type $A : \mathbf{Type}, B : A \to \mathbf{Type}$, the most general form of (dependent) eliminator is given by motives $F : A \to \mathbf{Type}$ and $G : (a : A) \to (b : Ba) \to F\ a \to \mathbf{Type}$, and the functions $\mathsf{elim}_A : (a : A) \to F\ a, \mathsf{elim}_B : (a : A) \to (b : B\ a) \to G\ a\ b\ (\mathsf{elim}_A\ a)$. A simpler notion of eliminator does not require the second component of the motive to mention the first. In particular, we have $F : A \to \mathbf{Type}$, $G : (a : A) \to B\ a \to \mathbf{Type}$, and eliminators $\mathsf{elim}_A : (a : A) \to F\ a, \mathsf{elim}_B : (a : A)(b : B\ a) \to G\ a\ b$. Note that this elimination principle—that we call *simple elimination* after [24]—is no longer recursive-recursive, since elim_A is not mentioned in the signature of elim_B.

Let us go back to our Con/Ty example. Given any dependent algebra of Con and Ty with motives $C : \mathsf{Con} \to \mathbf{Type}$ and $T : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\ \Gamma \to C\ \Gamma \to \mathbf{Type}$, the general eliminators have the following signatures:

$$\begin{aligned}
\mathsf{elim}_{\mathsf{Con}} &: (\Gamma : \mathsf{Con}) \to C\ \Gamma \\
\mathsf{elim}_{\mathsf{Ty}} &: \{\Gamma : \mathsf{Con}\}(A : \mathsf{Ty}\ \Gamma) \to T\ \Gamma\ A\ (\mathsf{elim}_{\mathsf{Con}}\ \Gamma)
\end{aligned}$$

These can be derived from our encoding of Con and Ty via untyped codes and well-typing predicates. The way to do it is to first define the graph of the eliminators in the form of inductively-generated relations:

$$\begin{aligned}
\mathsf{data\ R\text{-}Con} &: (\Gamma : \mathsf{Con}) \to C\ \Gamma \to \mathbf{Type} \\
\mathsf{data\ R\text{-}Ty} &: \{\Gamma : \mathsf{Con}\}(A : \mathsf{Ty}\ \Gamma)(\gamma : C\ \Gamma) \to T\ \Gamma\ A\ \gamma \to \mathbf{Type}
\end{aligned}$$

The next step is to prove that these relations are functional, by induction on the untyped codes Con_0 and Ty_0 [20]. From this result, defining the eliminators is immediate.

Reducing the setoid universe The reduction described in the previous section works generically for an arbitrary finitary inductive-inductive type, thus giving a systematic way to reduce finitary inductive-inductive definitions to inductive families. However, it is not clear whether this method extends to *infinitary* induction-induction, of which the setoid universe defined in Section 4.3 is an instance. Of course, the absence of a general reduction method does not mean that we can't reduce particular concrete instances of infinitary induction-induction, which is exactly what we hope for our universe construction.

The obvious challenge in successfully completing this reduction is to avoid the need for extensionality in the metatheory. In fact, consider the simple infinitary inductive-inductive type obtained from the previous Con/Ty example by replacing the finitary constructor Π with an infinitary one: $\Pi : \{\Gamma : \mathsf{Con}\} \to (\mathbb{N} \to \mathsf{Ty}\ \Gamma) \to \mathsf{Ty}\ \Gamma$. Already with this simple example, we run into problems as soon as we try to define the eliminator. One issue is that the definition of the eliminator relies on a proof that the well-typing predicates $\mathsf{inU}_1, \mathsf{inU}{\sim}_1$ are propositional, that is, any two of their elements are equal. Without further assumptions this proof can only be done by induction, and requires function extensionality since these predicates include higher-order constructors.

One way to get around this is to define the well-typing predicates as **Prop**-valued families, rather than in **Type**:

$$\begin{aligned}
&\mathsf{data}\ \mathsf{inU}_0 &&: \mathbf{Type} \to \mathbf{Type}_1\\
&\mathsf{data}\ \mathsf{inU}{\sim}_0 &&: \{A\ A' : \mathbf{Type}\} \to (A \to A' \to \mathbf{Prop}) \to \mathbf{Type}_1\\
&\mathsf{data}\ \mathsf{inU}_1 &&: (A : \mathbf{Type}) \to \mathsf{inU}_0\ A \to \mathbf{Prop}_1\\
&\mathsf{data}\ \mathsf{inU}{\sim}_1 &&: \{A\ A' : \mathbf{Type}\} \to (R : A \to A' \to \mathbf{Prop}) \to \mathsf{inU}{\sim}_0\ R \to \mathbf{Prop}_1
\end{aligned}$$

Using **Prop** avoids the issue of proving propositionality altogether, since the predicates are now propositional by definition. However, it introduces a different issue: inU_1 and $\mathsf{inU}{\sim}_1$ give rise to equational constraints on their indices, in the form of proofs of the **Prop**-valued identity type. The definition of the eliminators for inU and $\mathsf{inU}{\sim}$ relies on the ability to transport along these proofs, hence the need to extend our metatheory with a primitive, strong form of transport for Id.[8]

Having **Prop** and a strong transport principle does help to some extent. However, we would still need extensionality to derive the general eliminators for inU and $\mathsf{inU}{\sim}$. In fact, as explained in the previous section, to derive the general recursive-recursive eliminators we need to prove that the corresponding graph relations are functional, which can't be done without funext.

Luckily, it would seem the *simple* elimination principle is sufficient for our purposes: careful inspection of the code suggests that all functions described in Section 4.3 can be defined using simple mutual pattern matching without recursion-recursion. The simple eliminator can be defined by pattern matching

[8] Note that this issue cannot be solved by expressing the equational constraints with an identity type in **Type**, since the well-typing predicates force it to necessarily be in **Prop**.

on the untyped codes, and does not require extensionality or any extra principles beyond strong transport. Note that producing a full formalization that uses exclusively simple eliminators appears to be extremely time-consuming; we therefore leave this task to future work.

Once the inductive encoding of the inductive-inductive universe is done, the setoid universe can be defined just as in Section 4.3.

5 Conclusions and further work

We have described the construction of a universe of setoids in the setoid model of type theory; this is given in several steps, first as an inductive-recursive definition, then as an inductive-inductive definition, and finally as an inductive type. Every encoding is obtained from the previous by adapting known data type transformation methods in a novel way that accounts for the peculiarities of our construction. Most of the mathematical contents of this paper have been formalized in the proof-assistant Agda (see [1]). We aim to complete the formalization in the near future, in particular to obtain a machine-checked proof that our inductive construction of the setoid universe does not rely on recursion-recursion.

In contrast to the inductive-recursive and inductive-inductive versions of the universe, the inductive definition relies on a metatheory with a strong transport rule. As future work, we would like to prove normalization for this metatheory since previous work in this respect [3] seems to suggest that is represents a non-trivial addition.

Another question regards the relationship between SeTT [5] and XTT [26]. Both systems are syntactic representations of the setoid model with similar design choices, like definitional proof-irrelevance. We would like to know whether their respective notions of models are equivalent, that is, if we can obtain an XTT model from a SeTT model, and vice versa. Since XTT universes support universe induction, for one direction we would need to extend our own universe with the same principle (see discussion in Section 3 and the previous paragraph). Thus a related question is whether our encodings of the setoid universe can support universe induction. A further question is whether this mapping of models is functorial.

Groupoids can be regarded as generalized setoids. In the future we would like to design a type theory internalizing the groupoid model of type theory [19], in the same way that SeTT represents a syntax for the setoid model. A further question is whether such "groupoid type theory" can be justified, similarly to SeTT, via a syntactic translation, perhaps with SeTT itself as the target theory.

References

1. Agda formalization of the setoid universe. `https://bitbucket.org/taltenkirch/setoid-univ/src/master/`.
2. Andreas Abel. Extensional normalization in the logical framework with proof irrelevant equality. In Olivier Danvy, editor, *Workshop on Normalization by Evaluation, affiliated to LiCS 2009, Los Angeles, 15 August 2009*, 2009.

3. Andreas Abel and Thierry Coquand. Failure of normalization in impredicative type theory with proof-irrelevant propositional equality, 2019. `arXiv:1911.08174`.
4. Thorsten Altenkirch. Extensional equality in intensional type theory. In *Proceedings of the Fourteenth Annual IEEE Symposium on Logic in Computer Science (LICS 1999)*, pages 412–420. IEEE Computer Society Press, July 1999.
5. Thorsten Altenkirch, Simon Boulier, Ambrus Kaposi, and Nicolas Tabareau. Setoid type theory—a syntactic translation. In Graham Hutton, editor, *Mathematics of Program Construction*, pages 155–196, Cham, 2019. Springer International Publishing.
6. Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. *SIGPLAN Not.*, 51(1):18–29, January 2016. `doi:10.1145/2914770.2837638`.
7. Thorsten Altenkirch, Ambrus Kaposi, András Kovács, and Jakob von Raumer. Reducing inductive-inductive types to indexed inductive types. In José Espírito Santo and Luís Pinto, editors, *24th International Conference on Types for Proofs and Programs, TYPES 2018*. University of Minho, 2018.
8. Thorsten Altenkirch, Ambrus Kaposi, András Kovács, and Jakob von Raumer. Constructing inductive-inductive types via type erasure. In Marc Bezem, editor, *25th International Conference on Types for Proofs and Programs, TYPES 2019*. Centre for Advanced Study at the Norwegian Academy of Science and Letters, 2019.
9. Thorsten Altenkirch, Conor McBride, and Wouter Swierstra. Observational equality, now! In *PLPV '07: Proceedings of the 2007 workshop on Programming languages meets program verification*, pages 57–68, New York, NY, USA, 2007. ACM. `doi:http://doi.acm.org/10.1145/1292597.1292608`.
10. Marc Bezem, Thierry Coquand, and Simon Huber. A Model of Type Theory in Cubical Sets. In Ralph Matthes and Aleksy Schubert, editors, *19th International Conference on Types for Proofs and Programs (TYPES 2013)*, volume 26 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 107–128, Dagstuhl, Germany, 2014. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: `http://drops.dagstuhl.de/opus/volltexte/2014/4628`, `doi:10.4230/LIPIcs.TYPES.2013.107`.
11. Simon Boulier. *Extending Type Theory with Syntactical Models*. PhD thesis, IMT Atlantique, 2018.
12. Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. In Tarmo Uustalu, editor, *21st International Conference on Types for Proofs and Programs (TYPES 2015)*, volume 69 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:34, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: `http://drops.dagstuhl.de/opus/volltexte/2018/8475`, `doi:10.4230/LIPIcs.TYPES.2015.5`.
13. Peter Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *Journal of Symbolic Logic*, 65, 06 2003. `doi:10.2307/2586554`.
14. Peter Dybjer. Internal type theory. 06 2003. `doi:10.1007/3-540-61780-9_66`.
15. Peter Dybjer and Anton Setzer. A finite axiomatization of inductive-recursive definitions. In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications*, pages 129–146, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.
16. Gaëtan Gilbert, Jesper Cockx, Matthieu Sozeau, and Nicolas Tabareau. Definitional Proof-Irrelevance without K. *Proceedings of the ACM on Programming Languages*, pages 1–28, January 2019. URL: `https://hal.inria.fr/hal-01859964`, `doi:10.1145/329031610.1145/3290316`.

17. Martin Hofmann. *Extensional concepts in intensional type theory.* PhD thesis, University of Edinburgh, 1995.
18. Martin Hofmann. Conservativity of equality reflection over intensional type theory. In Stefano Berardi and Mario Coppo, editors, *Types for Proofs and Programs*, pages 153–164, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.
19. Martin Hofmann and Thomas Streicher. The groupoid interpretation of type theory. In *Twenty-five years of constructive type theory (Venice, 1995)*, volume 36 of *Oxford Logic Guides*, pages 83–111. Oxford Univ. Press, New York, 1998.
20. Ambrus Kaposi, András Kovács, and Lafont Ambroise. For finitary induction-induction, induction is enough. *Submitted to TYPES 2019 post-proceedings*, 2019.
21. Lorenzo Malatesta, Thorsten Altenkirch, Neil Ghani, Peter Hancock, and Conor McBride. Small induction recursion, indexed containers and dependent polynomials are equivalent, 2013. TLCA 2013.
22. Per Martin-Löf. An intuitionistic theory of types: Predicative part. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium '73*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73 – 118. Elsevier, 1975. URL: `http://www.sciencedirect.com/science/article/pii/S0049237X08719451`, `doi:https://doi.org/10.1016/S0049-237X(08)71945-1`.
23. Per Martin-Löf. *Intuitionistic type theory*, volume 1 of *Studies in proof theory.* Bibliopolis, 1984.
24. Fredrik Nordvall Forsberg. *Inductive-inductive definitions.* PhD thesis, Swansea University, 2013.
25. Erik Palmgren and Olov Wilander. Constructing categories and setoids of setoids in type theory. *arXiv e-prints*, page arXiv:1408.1364, August 2014. `arXiv:1408.1364`.
26. Jonathan Sterling, Carlo Angiuli, and Daniel Gratzer. Cubical syntax for reflection-free extensional equality. In Herman Geuvers, editor, *Proceedings of the 4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*, volume 131 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:25. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: `http://drops.dagstuhl.de/opus/volltexte/2019/10538`, `arXiv:1904.08562`, `doi:10.4230/LIPIcs.FSCD.2019.31`.
27. The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics.* `https://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.