Quotient inductive-inductive types in the setoid model*

Ambrus Kaposi and Zongpu Xie

Eötvös Loránd University, Budapest, Hungary
akaposi@inf.elte.hu and szumixie@gmail.com

Introduction. The setoid model of type theory provides a way to bootstrap functional extensionality [1] and propositional extensionality [3]: the setoid model can be defined in an intensional metatheory with a universe of definitionally proof irrelevant (strict) propositions SProp. It is a strict model in such a metatheory, that is, all equalities of the model (e.g. β and η for function space) hold definitionally. As a result, we obtain a model construction: any model of type theory with SProp can be turned into another model, its "setoidified" version which supports these extra principles. In addition to functional and propositional extensionality, the setoid model justifies propositional truncation[1], quotient types[2] and countable choice[3].

Since Agda supports SProp [9], it is a convenient tool to experiment with the setoid model. It is straightforward to formalise the setoid model as a category with families (CwF [6]) with Π, Σ, unit, empty, Bool, $\mathbb{N}$, Id types, a universe of strict propositions. Extending the setoid model with a (non-univalent) universe of sets is harder, it was shown by Altenkirch et al. [2] that it can be done using a special form of induction-recursion or large induction-induction or an SProp-valued identity type with transport over types.

Until recently we thought [12] that general inductive types and even quotient inductive-inductive types (QIITs, initial algebras of generalised algebraic theories [13, 5]) are unproblematic in the setoid model, provided we have (possibly SProp-sorted) inductive-inductive types in the metatheory. Simon Boulier pointed out that our formalisations of Martin-Löf's identity type[4] and the universal QIIT[5] only provide eliminators in the empty context. They can be salvaged using a method related to the local universes construction [14] which we explain below.

The setoid model. A context or a closed type in this model is a setoid, i.e. a set (we say set instead of (Agda) type to avoid confusion) together with an SProp-valued equivalence relation. A type over a context $\Gamma = (|\Gamma|, \sim_\Gamma)$ is a displayed setoid with a fibration condition $\mathsf{coe}_A : x \sim_\Gamma x' \to |A|\ x \to |A|\ x'$ such that $x \sim_A (\mathsf{coe}_A\, p\, x)$. Substitutions (and terms) are (dependent) functions between the underlying sets which preserve the relations.

Example: Con-Ty. To illustrate the general method, we explain how to construct the following QIIT in the setoid model[6]. It has two sorts, five constructors and one equality constructor.

$$
\begin{aligned}
&\mathsf{Con} : \mathsf{Set} && \mathsf{U} : \mathsf{Ty}\ \gamma \\
&\mathsf{Ty} : \mathsf{Con} \to \mathsf{Set} && \mathsf{El} : \mathsf{Ty}\ (\gamma \rhd \mathsf{U}) \\
&\bullet : \mathsf{Con} && \Sigma : (a : \mathsf{Ty}\ \gamma) \to \mathsf{Ty}\ (\gamma \rhd a) \to \mathsf{Ty}\ \gamma \\
&- \rhd - : (\gamma : \mathsf{Con}) \to \mathsf{Ty}\ \gamma \to \mathsf{Con} && \mathsf{eq} : \gamma \rhd \Sigma\ a\ b = \gamma \rhd a \rhd b
\end{aligned}
$$

*The first author was supported by the ÚNKP-20-5 New National Excellence Program of the Ministry for Innovation and Technology and by the Bolyai Fellowship of the Hungarian Academy of Sciences. The second author was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).

[1] https://bitbucket.org/akaposi/setoid/src/master/agda/Model/Trunc.agda

[2] https://bitbucket.org/akaposi/setoid/src/master/agda/Model/Quotient.agda

[3] https://bitbucket.org/akaposi/setoid/src/master/agda/Model/CountableChoice.agda

[4] https://bitbucket.org/akaposi/qiit/src/master/Setoid/Path.agda

[5] https://bitbucket.org/akaposi/qiit/src/master/Setoid/UniversalQIIT/

[6] https://bitbucket.org/akaposi/qiit/src/master/Setoid/ConTy2.agda

We omitted some arguments, e.g. U implicitly takes a parameter γ. In the setoid model, we need to define a type Con in the empty context, a type Ty over Con, and their elimination principles. We start by defining in Agda an inductive-inductive type (IIT) with these sorts:

$$|\mathsf{Con}| : \mathsf{Set} \qquad\qquad |\mathsf{Ty}| \;: |\mathsf{Con}| \to \mathsf{Set}$$
$$\sim_{\mathsf{Con}} : |\mathsf{Con}| \to |\mathsf{Con}| \to \mathsf{SProp} \qquad\qquad \sim_{\mathsf{Ty}} : \gamma \sim_{\mathsf{Con}} \gamma' \to |\mathsf{Ty}|\,\gamma \to |\mathsf{Ty}|\,\gamma' \to \mathsf{SProp}$$

The constructors of $|\mathsf{Con}|$ are $|\bullet|$ and $|\rhd|$, the constructors of $|\mathsf{Ty}|$ are $|\mathsf{U}|$, $|\mathsf{El}|$ and $|\Sigma|$, while $\sim_{\mathsf{Con}}$ has a constuctor $|\mathsf{eq}|$. In addition, $\sim_{\mathsf{Con}}$ and $\sim_{\mathsf{Ty}}$ have constructors stating that it is an equivalence relation, and they have congruence constructors for each point constructor, e.g. there is $\sim_{\rhd}$: $(p : \gamma \sim_{\mathsf{Con}} \gamma') \to \sim_{\mathsf{Ty}} p\, a\, a' \to (\gamma \rhd a) \sim_{\mathsf{Con}} (\gamma' \rhd a')$. Finally, $|\mathsf{Ty}|$ has a constructor $\mathsf{coe}_{\mathsf{Ty}} : \gamma_0 \sim_{\mathsf{Con}} \gamma_1 \to |\mathsf{Ty}|\,\gamma_0 \to |\mathsf{Ty}|\,\gamma_1$ and $\sim_{\mathsf{Ty}}$ a constructor for $\sim_{\mathsf{Ty}} p\, a\, (\mathsf{coe}_{\mathsf{Ty}}\, p\, a)$. Thus the IIT is the "fibrant equivalence congruence closure" of the constructors.

With the aid of this IIT (note that it has both Set and SProp-sorts) we define the type formation rules and constructors of the Con-Ty QIIT in the setoid model *in the empty context*: the underlying set for Con is $|\mathsf{Con}|$, the relation is $\sim_{\mathsf{Con}}$, the witnesses for the equivalence relation come from the corresponding constructors of $\sim_{\mathsf{Con}}$, and so on. Thus Con becomes a type in the empty context in the setoid model. Ty is a type over the one-element context Con. $\bullet$ is a term in the empty context of type Con, and so on. We added exactly the required structure to the IIT to be able to define the constructors. The eq equality constructor is given by $|\mathsf{eq}|$. Given a Con-Ty algebra in the empty context, we define four functions by recursion-recursion as a first step towards the (non-dependent) elimination principle.

The type formation rules and constructors can easily be lifted from the empty context to an arbitrary context and all the substitution laws hold definitionally. We also need that for any context Γ, we can eliminate into a Con-Ty algebra in Γ. Our setoid model has Π types and K constant types (a context can be turned into a type). With the help of these we can turn a type C in Γ into the type $\Pi(x : \mathsf{K}\,\Gamma).C[x]$ which is in the empty context. This way we turn the algebra in Γ into an algebra in the empty context on which we can apply our previously defined elimination principle. This way we obtain the eliminator in arbitrary contexts. All computation rules of this eliminator are definitional.

We prove uniqueness of the eliminator by induction-induction on $|\mathsf{Con}|$ and $|\mathsf{Ty}|$. The substitution law of the eliminator is proven by another induction-induction on the same sets.

In our formalisation, Con-Ty has an additional infinitary constructor (an infinitary Π type indexed by a code of a setoid in a universe). It seems that with the help of a universe in the setoid model, open QIITs and those with infinitary constructors can be handled as well. Note that in contrast with the unordered infinitely branching tree example in [4], we do not use the (countable) axiom of choice to construct this QIIT.

Arbitrary QIITs Signatures for QIITs can be specified using the theory of QIIT signatures [13] (ToS) which is itself an infinitary QIIT. We formalised[7] that the setoid model supports ToS in the empty context. Based on our experience with Con-Ty, we expect that it is possible to lift ToS from the empty context to arbitrary contexts. If this succeeds, then following [13], we can construct all QIITs from ToS with propositional computation rules. As the construction of [13] is performed in extensional type theory, we use Hofmann's conservativity result [10, 15] to transfer it to the setoid model. This way however we only obtain propositional computation rules. It remains to be proven that all QIITs are supported by the setoid model with definitional computation rules. We plan to do this by induction on QIIT signatures.

We would also like to understand the relationship of this construction to that of higher inductive types (HITs) in cubical models [8, 7] and how they could be extended to HIITs [11].

[7] https://bitbucket.org/akaposi/qiit/src/master/Setoid2/ToS/

References

[1] Thorsten Altenkirch. Extensional equality in intensional type theory. In *14th Symposium on Logic in Computer Science*, pages 412 – 420, 1999.

[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 - 24th International Conference, FOSSACS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings*, volume 12650 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2021.

[3] 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.

[4] Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In Rastislav Bodik and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 18–29. ACM, 2016.

[5] John Cartmell. Generalised algebraic theories and contextual categories. *Annals of Pure and Applied Logic*, 32:209–243, 1986.

[6] Simon Castellan, Pierre Clairambault, and Peter Dybjer. Categories with families: Unityped, simply typed, and dependently typed. *CoRR*, abs/1904.00827, 2019.

[7] Evan Cavallo and Robert Harper. Higher inductive types in cubical computational type theory. *Proc. ACM Program. Lang.*, 3(POPL), January 2019.

[8] Thierry Coquand, Simon Huber, and Anders Mörtberg. On higher inductive types in cubical type theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, page 255–264, New York, NY, USA, 2018. Association for Computing Machinery.

[9] 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.

[10] Martin Hofmann. Conservativity of equality reflection over intensional type theory. In *TYPES 95*, pages 153–164, 1995.

[11] Ambrus Kaposi and András Kovács. Signatures and Induction Principles for Higher Inductive-Inductive Types. *Logical Methods in Computer Science*, Volume 16, Issue 1, February 2020.

[12] Ambrus Kaposi and Zongpu Xie. A model of type theory with quotient inductive-inductive types. In Ugo de' Liguoro and Stefano Berardi, editors, *26th International Conference on Types for Proofs and Programs, TYPES 2020*. University of Turin, 2020.

[13] András Kovács and Ambrus Kaposi. Large and infinitary quotient inductive-inductive types. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 648–661. ACM, 2020.

[14] Peter Lefanu Lumsdaine and Michael A. Warren. The local universes model: An overlooked coherence construction for dependent type theories. *ACM Trans. Comput. Logic*, 16(3), July 2015.

[15] Théo Winterhalter, Matthieu Sozeau, and Nicolas Tabareau. Eliminating reflection from type theory. In Assia Mahboubi and Magnus O. Myreen, editors, *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2019, Cascais, Portugal, January 14-15, 2019*, pages 91–103. ACM, 2019.