A model of type theory with quotient inductive-inductive types *

Ambrus Kaposi and Zongpu Xie

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

QIITs. Quotient inductive-inductive types (QIITs) are a general class of inductive types that allow multiple sorts indexed over each other (inductive-inductive [10]) and support equality constructors (quotient). An example is given in the left column of Figure 1. This QIIT has two sorts, Con and Ty, five point constructors $\bullet$, ..., Σ, and an equality constructor eq. It comes with two elimination principles (one for each sort, we don't list them here) which enforce that every function from Con preserves the equality eq. Con–Ty can be extended to the full syntax of type theory [4]. Other examples of QIITs include the real numbers [11] and the partiality monad [3]. Kaposi et al. [8] gave a general definition of QIITs and showed that all finitary QIITs can be constructed from a single QIIT called the universal QIIT. However they did not show that this universal QIIT exists.

In this talk we show that there is a model of type theory which supports the universal QIIT, namely the setoid model [1]. The setting of [8] is extensional type theory, hence by the conservativity result of Hofmann [7] the construction can be transferred to a model of type theory with function extensionality and uniqueness of identity proofs (UIP). As these hold in the setoid model, we conclude that all finitary QIITs can be defined in the setoid model.

The contents of this talk were formalised in Agda, we provide links to specific parts below.

Con	$:\mathsf{Set}$	Con^s	$:\mathsf{Ty}\ \bullet$
Ty	$:\mathsf{Con}\to\mathsf{Set}$	Ty^s	$:\mathsf{Ty}\ (\bullet\triangleright\mathsf{Con}^\mathsf{s})$
$\bullet$	$:\mathsf{Con}$	$\bullet^\mathsf{s}$	$:\mathsf{Tm}\ \bullet\ \mathsf{Con}^\mathsf{s}$
$-\triangleright-$	$:(\Gamma:\mathsf{Con})\to\mathsf{Ty}\ \Gamma\to\mathsf{Con}$	$\triangleright^\mathsf{s}$	$:\mathsf{Tm}\ (\bullet\triangleright\mathsf{Con}^\mathsf{s}\triangleright\mathsf{Ty}^\mathsf{s})\ (\mathsf{Con}^\mathsf{s}\,[\epsilon])$
U	$:\mathsf{Ty}\ \Gamma$	U^s	$:\mathsf{Tm}\ (\bullet\triangleright\mathsf{Con}^\mathsf{s})\ \mathsf{Ty}^\mathsf{s}$
El	$:\mathsf{Ty}\ (\Gamma\triangleright\mathsf{U})$	El^s	$:\mathsf{Tm}\ (\bullet\triangleright\mathsf{Con}^\mathsf{s})\ (\mathsf{Ty}^\mathsf{s}\,[\epsilon,\triangleright^\mathsf{s}\,[\mathsf{id},\mathsf{U}^\mathsf{s}]])$
Σ	$:(A:\mathsf{Ty}\ \Gamma)\to\mathsf{Ty}\ (\Gamma\triangleright A)\to\mathsf{Ty}\ \Gamma$	Σ^s	$:\mathsf{Tm}\ (\bullet\triangleright\mathsf{Con}^\mathsf{s}\triangleright\mathsf{Ty}^\mathsf{s}\triangleright\mathsf{Ty}^\mathsf{s}\,[\epsilon,\triangleright^\mathsf{s}])\ (\mathsf{Ty}^\mathsf{s}\,[\mathsf{wk}^2])$
eq	$:\Gamma\triangleright\Sigma\ A\ B=\Gamma\triangleright A\triangleright B$	eq^s	$:\mathsf{Tm}\ (\bullet\triangleright\mathsf{Con}^\mathsf{s}\triangleright\mathsf{Ty}^\mathsf{s}\triangleright\mathsf{Ty}^\mathsf{s}\,[\epsilon,\triangleright^\mathsf{s}])$
			$(\mathsf{El}\ (\mathsf{Id}\ (\mathsf{Con}^\mathsf{s}\,[\epsilon])\ (\triangleright^\mathsf{s}\,[\mathsf{wk}^2,\Sigma^\mathsf{s}])\ (\triangleright^\mathsf{s}\,[(\epsilon,\triangleright^\mathsf{s})^\uparrow])))$

Figure 1: Constructors of the QIIT Con–Ty, a fragment of the well-typed syntax of type theory. Note that Con, Ty on the left become Con^s, Ty^s on the right and Ty, Tm on the right are those of a model of type theory.

Specification of QIITs in a model. We use categories with families (CwFs [5]) as the notion of model of type theory. That is, a model is a category given by objects Con, morphisms Sub, families Ty, Tm, substitution is written $-[-]$, empty context $\bullet$, context extension $\triangleright$.

*The first author was supported by the ÚNKP-19-4 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).

Specifying what it means for a model to support a QIIT is straightforward but tedious. For the constructors of Con–Ty this is done on the right hand side of Figure 1. Here Ty and Tm refer to types and terms of the model, not sorts of the QIIT. The elimination principle can be specified in a similar way. The difficulty of working inside a model is that we have to write out all arguments explicitly (e.g. Γ for U), we need weakenings (e.g. in the type of Σ^{s}) and CwF-combinators instead of variables names. For modularity and efficiency reasons we don't use function space of the model to list parameters of constructors, instead we add them to the context. E.g. $\triangleright^{\mathsf{s}}$ is not a function with two arguments, but a term in a context of length two.

The universal QIIT is a syntax for a small type theory describing signatures of QIITs. It is (roughly) an extension of our Con–Ty example. We specified the constructors and elimination principles of the universal QIIT. These specifications import the setoid model, but they also work with the standard (set) model or any other strict model. Importing the standard model and normalising the types helps to make sure that the internal specification (right hand side of Figure 1) corresponds to the external one (left hand side of the figure).

The setoid model. In the setoid model [1], an element of Con is a setoid, that is, a set with a (Prop-valued) equivalence relation. The idea is that each type comes with its own identity type encoded in this relation. For example, the relation for function space says that the two functions are extensionally equal. This way the setoid model supports function extensionality, propositional extensionality and quotients. We formalised the setoid model in Agda without using any axioms or postulates, we also did not rely on UIP in Agda. The only special feature that we used was the definitionally proof irrelevant universe of propositions Prop [6]. The setoid model is strict [2], that is, all the equalities are definitional in Agda.

Implementation of QIITs in the setoid model. We defined an IIT with four sorts which we call the *implementation IIT* for Con–Ty. There are the two Set-sorts for Con and Ty and two Prop-sorts for their equality relations. The constructor eq is a constructor of the equality relation for Con. The Set-sort for Ty includes an additional coercion constructor, the Prop-sorts include constructors expressing that they are equivalence relations, congruence rules and that coercion respects the relation. With the help of the implementation IIT, we defined the constructors for Con–Ty. The elimination principle is defined simply by pattern matching (eliminating from the implementation IIT). We defined both the recursion principle with uniqueness and the dependent elimination principle for Con–Ty. All the computation rules hold definitionally.

The above method also works for the universal QIIT, we defined the recursor and formalised uniqueness. We haven't managed to typecheck uniqueness yet due to performance problems.

Further work. We would like to extend the universal QIIT with more type formers to allow non-closed QIITs (metatheoretic Π), infinitary constructors, equalities as inputs of constructors, sort equalities. We formalised that the setoid model supports arbitrary branching trees where the order of subtrees do not matter. This is an infinitary QIT which seems not to be definable from quotients without using the axiom of choice [4]. In fact, some QITs are known not to be constructible without the axiom of choice [9, Section 9].

We showed that finitary QIITs exist in the setoid model, but can they be defined in the set model (using only quotients)? There is a weak morphism of models from the setoid model to the set model, we plan to investigate what this morphism maps the universal QIIT to.

As the setoid model is given in an intensional metatheory, it provides a computational interpretation of the QIITs we defined. It remains to be checked what happens if we replay the construction of other QIITs from the universal QIIT [8]. Would we still get definitional computation rules?

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, 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.

[3] Thorsten Altenkirch, Nils Anders Danielsson, and Nicolai Kraus. Partiality, revisited. In *Proceedings of the 20th International Conference on Foundations of Software Science and Computation Structures - Volume 10203*, page 534–549, Berlin, Heidelberg, 2017. Springer-Verlag.

[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] Peter Dybjer. Internal type theory. In *Lecture Notes in Computer Science*, pages 120–134. Springer, 1996.

[6] 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.

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

[8] Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. Constructing quotient inductive-inductive types. *Proc. ACM Program. Lang.*, 3(POPL):2:1–2:24, January 2019.

[9] Peter LeFanu Lumsdaine and Mike Shulman. Semantics of higher inductive types, 2012. Note.

[10] Fredrik Nordvall Forsberg. *Inductive-inductive definitions*. PhD thesis, Swansea University, 2013.

[11] The Univalent Foundations Program. Homotopy type theory: Univalent foundations of mathematics. Technical report, Institute for Advanced Study, 2013.