A model of type theory supporting quotient inductive-inductive types[1]

Ambrus Kaposi and Szumi Xie

Eötvös Loránd University

akaposi@inf.elte.hu and szumixie@gmail.com

Quotient inductive-inductive types

Natural numbers, lists, binary trees, the syntax of terms of a programming language are all examples of inductive types. In general, an inductive type is a collection of trees of finite depth with a fixed branching structure. Inductive-inductive types (IITs) [11] generalise inductive types by allowing multiple mutually defined sorts where the later sorts can be indexed over the previous ones. Quotient inductive-inductive types (QIITs) [9] in addition allow equality constructors. They are more general than simple quotients as the elements of the type are generated *at the same time* as the quotienting. This allows the constructive definition of Cauchy real numbers in type theory without using the axiom of choice [12]. Other examples are the partiality monad [3], the intrinsic syntax of programming languages [4]. In general, for any generalised algebraic theory [5], the initial object in the category of its models is a QIIT [9].

A simple example of a QIIT is a subset of the intrinsic syntax of type theory [4] with two sorts, five operators and one equation:

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

There is an ordinary sort of contexts Con, a sort of types Ty indexed over contexts. That is, for every $\gamma : \mathsf{Con}$ we have that $\mathsf{Ty}\,\gamma$ is a sort. There are two operators producing contexts, empty context $\bullet$ and context extension $- \rhd -$, the latter refers to types. This is why Ty cannot be defined separately after Con, they have to be given mutually, at the same time. There are three different ways to form types corresponding to the three operators

[1]The first author was by supported by the "Application Domain Specific Highly Reliable IT Solutions" project which has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme TKP2020-NKA-06 (National Challenges Subprogramme) funding scheme. The second author was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).

producing Tys. Finally, the equation `eq` says that extending a context γ with a Σ type made of a and b is the same as extending it first with a and then with b.

A model of type theory supports a particular QIIT if (i) there is an algebra called the constructors: this means that for each sort there is a type, for each operator there are corresponding terms and for each equation there is a term of the corresponding identity type; (ii) for any other algebra there is a unique algebra homomorphism from the constructor to this algebra. An algebra homomorphism is given by terms (functions) for each sort that respect the operations up to definitional equality of the model. If they only respect the operations up to the identity type of the model, we only have propositional (weak) computation rules for the QIIT. A formal description of these notions is given in [9].

The universal QIIT

If a model of extensional type theory (ETT) supports a particular QIIT called *the universal QIIT* (UQIIT), then it supports all QIITs [9]. Hofmann's conservativity theorem [8] says that if a type can be expressed in intensional type theory with function extensionality and uniqueness of identity proofs (ITT+funext+UIP) and there is a term of this type in ETT, then there is also a term of this type in ITT+funext+UIP. As for any QIIT Ω the type

$$\text{“the universal QIIT exists} \Rightarrow \Omega \text{ exists”}$$

can be expressed in ITT+funext+UIP, we can transfer the proof of [9] given in ETT to any model of ITT+funext+UIP. However as this type expresses the computation rules in “Ω exists” using the identity type of the model instead of definitional equality, we only obtain weak computation rules. Thus we have that if a model of ITT+funext+UIP supports the UQIIT with propositional computation rules, then it supports all QIITs with propositional computation rules.

The setoid model

The setoid model of type theory [1] justifies funext and UIP and our goal is to show that it also supports the UQIIT, thus all QIITs with propositional computation rules.

We formalised in Agda the setoid model as a category with families (CwF [6]) with extra structure. We formalised what it means that this model supports the UQIIT. More precisely, we formalised what it means for a strict model to support the UQIIT. We typechecked our formalisation with two strict models: the setoid model and the standard (set) model. This involves saying what types and terms a UQIIT algebra consists of, what a UQIIT homomorphism between two such algebras is (with definitional computation rules) and what it means that two parallel homomorphisms are equal (up to the model's identity type). We used a variant of the UQIIT which supports infinitary operations [10]. The formalisation is available at the URL `https://bitbucket.org/akaposi/qiit`.

In the setoid model a context is given by a set (a type in the metatheory or target model) and an equivalence relation. A type is a family of sets together with a family of equivalence relations indexed over those of the context and fibration conditions: if there are two elements of the context that are related, then we can transport between types at each element (coercion), and this transport preserves the relation of the type (coherence). A term is a function between the sets which respects the relation. In the setoid model, the identity type is not given by an inductive type once and for all for all types, but by a separate relation for each type. This relation is inductively generated for inductive types and is coinductively generated for coinductive types. Specifically, the identity type of two functions (the function type is a coinductive type) says that they are pointwise equal (pointwise equality is another function, hence a coinductive type). From the reflexive property of the equivalence relation we obtain the usual reflexivity constructor of the identity type. From the fact that all terms respect equality and the coercion operation of types, we obtain the usual eliminator (transport or J) of the identity type. From the definition of the relation for function space, we obtain function extensionality and from the fact that the equivalence relations are proof irrelevant (target Prop [7] instead of Set), we obtain uniqueness of identity proofs. Thus setoids model ITT+funext+UIP. Moreover, the setoid model is strict, that is, all equalities are definitional in type theory as metatheory. This provides a *model construction* (syntactic translation) [2]: from any model of ITT with Prop, we obtain a "setoidified" version of the model which satisfies funext and UIP.

QIITs in the setoid model

In the setoid model, we can define the constructors of the UQIIT. This is given by an IIT with twice as many sorts as the UQIIT: there is a "point sort" for each sort of the QIIT and an additional propositional sort for its equality. The point sorts have a constructor for each operation and the indexed point sorts in addition have coercion constructors. The equality sorts are the free fibrant equivalence congruence relations over the equality constructors: all of them have constructors for reflexivity, symmetry, transitivity, one congruence constructor for each operator, one constructor for each equation, and for the indexed sorts, there is a coherence constructor. For example, we can define our example QIIT with Con and Ty in the setoid model using the following IIT (sometimes we denote implicit arguments by curly braces, e.g. in the type of $\sim_{\mathsf{U}}$).

$\vert\mathsf{Con}\vert$	$: \mathsf{Set}$	point sorts
$\vert\mathsf{Ty}\vert$	$: \vert\mathsf{Con}\vert \to \mathsf{Set}$	⋮
$- \sim_{\mathsf{Con}} -$	$: \vert\mathsf{Con}\vert \to \vert\mathsf{Con}\vert \to \mathsf{SProp}$	equality sorts
$\sim_{\mathsf{Ty}}$	$: \gamma \sim_{\mathsf{Con}} \gamma' \to \vert\mathsf{Ty}\vert\, \gamma \to \vert\mathsf{Ty}\vert\, \gamma' \to \mathsf{SProp}$	⋮
$\vert \bullet \vert$	$: \vert\mathsf{Con}\vert$	point constructor for
$-\vert\rhd\vert-$	$: (\gamma : \vert\mathsf{Con}\vert) \to \vert\mathsf{Ty}\vert\, \gamma \to \vert\mathsf{Con}\vert$	each operator
$\vert\mathsf{U}\vert$	$: \vert\mathsf{Ty}\vert\, \gamma$	⋮
$\vert\mathsf{El}\vert$	$: \vert\mathsf{Ty}\vert\, (\gamma \,\vert\rhd\vert\, \vert\mathsf{U}\vert)$	
$\vert\Sigma\vert$	$: (a : \vert\mathsf{Ty}\vert\, \gamma) \to \vert\mathsf{Ty}\vert\, (\gamma \,\vert\rhd\vert\, a) \to \vert\mathsf{Ty}\vert\, \gamma$	
$\sim_\bullet$	$: \vert \bullet \vert \sim_{\mathsf{Con}} \vert \bullet \vert$	congruence for
$- \sim_\rhd -$	$: (\bar\gamma : \gamma \sim_{\mathsf{Con}} \gamma') \to \sim_{\mathsf{Ty}} \bar\gamma\ \alpha\ \alpha' \to (\gamma \,\vert\rhd\vert\, \alpha) \sim_{\mathsf{Con}} (\gamma' \,\vert\rhd\vert\, \alpha')$	each operator
$\sim_{\mathsf{U}}$	$: \{\bar\gamma : \gamma \sim_{\mathsf{Con}} \gamma'\} \to \sim_{\mathsf{Ty}} \bar\gamma\ (\vert\mathsf{U}\vert\, \{\gamma\})\, (\vert\mathsf{U}\vert\, \{\gamma'\})$	⋮
$\sim_{\mathsf{El}}$	$: \{\bar\gamma : \gamma \sim_{\mathsf{Con}} \gamma'\} \to \sim_{\mathsf{Ty}} \bar\gamma\ (\vert\mathsf{El}\vert\, \{\gamma\})\, (\vert\mathsf{El}\vert\, \{\gamma'\})$	
$\sim_\Sigma$	$: (\bar a : \sim_{\mathsf{Ty}} \bar\gamma\ a\ a') \to \sim_{\mathsf{Ty}} (\bar\gamma \sim_\rhd \bar a)\ b\ b' \to$ $\sim_{\mathsf{Ty}} \bar\gamma\ (\vert\Sigma\vert\ a\ b)\ (\vert\Sigma\vert\ a'\ b')$	

$\lvert\mathsf{eq}\rvert$	$: \gamma \,\lvert\rhd\rvert\, \lvert\Sigma\rvert\; a\; b \sim_{\mathsf{Con}} \gamma \,\lvert\rhd\rvert\, a \,\lvert\rhd\rvert\, b$	equality constructor
$\mathsf{refl}_{\mathsf{Con}}$	$: (\gamma : \lvert\mathsf{Con}\rvert) \to \gamma \sim_{\mathsf{Con}} \gamma$	equivalence relations
$\mathsf{sym}_{\mathsf{Con}}$	$: \gamma \sim_{\mathsf{Con}} \gamma' \to \gamma' \sim_{\mathsf{Con}} \gamma$	$\vdots$
$\mathsf{trans}_{\mathsf{Con}}$	$: \gamma \sim_{\mathsf{Con}} \gamma' \to \gamma' \sim_{\mathsf{Con}} \gamma'' \to \gamma \sim_{\mathsf{Con}} \gamma''$	
$\mathsf{refl}_{\mathsf{Ty}}$	$: (a : \lvert\mathsf{Ty}\rvert\; \gamma) \to \sim_{\mathsf{Ty}} (\mathsf{refl}_{\mathsf{Con}}\; \gamma)\; a\; a$	
$\mathsf{sym}_{\mathsf{Ty}}$	$: \sim_{\mathsf{Ty}} \bar{\gamma}\; a\; a' \to \sim_{\mathsf{Ty}} (\mathsf{sym}_{\mathsf{Con}}\; \bar{\gamma})\; a'\; a$	
$\mathsf{trans}_{\mathsf{Ty}}$	$: \sim_{\mathsf{Ty}} \bar{\gamma}\; a\; a' \to \sim_{\mathsf{Ty}} \bar{\gamma}'\; a'\; a'' \to \sim_{\mathsf{Ty}} (\mathsf{trans}_{\mathsf{Con}}\; \bar{\gamma}\; \bar{\gamma}')\; a\; a''$	
$\mathsf{coe}_{\mathsf{Ty}}$	$: \gamma \sim_{\mathsf{Con}} \gamma' \to \lvert\mathsf{Ty}\rvert\; \gamma \to \lvert\mathsf{Ty}\rvert\; \gamma'$	fibration conditions
$\mathsf{coh}_{\mathsf{Ty}}$	$: (\bar{\gamma} : \gamma \sim_{\mathsf{Con}} \gamma')(\alpha : \lvert\mathsf{Ty}\rvert\; \gamma) \to \sim_{\mathsf{Ty}} \bar{\gamma}\; \alpha\; (\mathsf{coe}_{\mathsf{Ty}}\; \bar{\gamma}\; \alpha)$	$\vdots$

We define the IIT for the UQIIT completely analogously. Then the constructor UQIIT algebra is given exactly by the components of this IIT.

The recursor takes as input a UQIIT algebra in the empty context and returns a homomorphism from the constructors to this algebra. The computation rules are definitional. An algebra in an arbitrary context can be turned into an algebra in the empty context $\cdot$. For example, for a type $\Gamma \vdash C$, we turn it into $\cdot \vdash \Pi(x : \mathsf{K}\ \Gamma).C[x]$, that is to a dependent function type where the domain is constant $(\mathsf{K})\ \Gamma$. Thus, when we want to use the recursor in a context Γ, we convert the algebra into an algebra in the empty context, and then we apply the recursor. The computation rules of this lifted recursor are still definitional. Uniqueness of the recursor and the fact that the recursor is stable under substitution are proved by induction on the IIT.

Further work

The notion of UQIIT algebra, morphism, the implementation IIT and the recursor in the empty context are all defined in Agda. The lifting of the recursor to arbitrary context is still a work in progress.

An alternative of our construction would be to directly reduce a QIIT to an IIT in the setoid model by induction on the QIIT signature. This way we would avoid going through the UQIIT and ETT and we expect that we would get definitional computation rules.

References

[1] Thorsten Altenkirch. Extensional equality in intensional type theory. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 412–420. IEEE Computer Society, 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] John Cartmell. Generalised algebraic theories and contextual categories. *Annals of Pure and Applied Logic*, 32:209–243, 1986.

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

[7] Gaëtan Gilbert, Jesper Cockx, Matthieu Sozeau, and Nicolas Tabareau. Definitional proof-irrelevance without K. *Proc. ACM Program. Lang.*, 3(POPL):3:1–3:28, 2019.

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

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

[10] 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.

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

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