Specifying QIITs using Containers

Thorsten Altenkirch[1] and Stefania Damato[1]

[1] University of Nottingham, Nottingham, UK
{thorsten.altenkirch, stefania.damato}@nottingham.ac.uk

We present ongoing work on providing semantics and syntax for QIITs via generalised containers. Our aim is to contribute towards the long-term goal of providing a rigorous theoretical foundation for higher inductive types in HoTT.

What are QIITs? *Induction-induction* allows us to simultaneuosly define a type `A : Type` with a family `B : A → Type` over `A`, where the constructors of `A` can refer to `B`. This means we can refer to a family (e.g. a predicate) over `A` when defining `A` itself. *Quotient inductive types* are inductive types that not only admit point constructors, but also path constructors, or equalities. Combining these two together, we get *quotient inductive-inductive types* (QIITs). QIITs can also be viewed as set-truncated higher inductive types with induction-induction. The QIIT of the syntax of a very basic type theory is given below as an example.

```
data Con : Set        data Ty : Con → Set

data Con where
  ◇ : Con
  _,_ : (Γ : Con) → Ty Γ → Con
  eq : (Γ : Con) (A : Ty Γ) (B : Ty (Γ , A)) → (Γ , A) , B ≡ Γ , σ Γ A B

data Ty where
  ι : (Γ : Con) → Ty Γ
  σ : (Γ : Con) (A : Ty Γ) (B : Ty (Γ , A)) → Ty Γ
```

Induction-induction allows us to refer to σ when defining `Con`, while quotienting allows us to write `eq` as a constructor of `Con`.

Semantics of QIITs Simple inductive types and inductive families can be described semantically as the initial algebras of container functors and indexed container functors respectively [1, 3]. A similar semantic description for QIITs has not yet been established. The first obstacle we face is that due to the high dependency allowed between constructors of different sorts, we have no way of expressing QIITs using endofunctors. To overcome this first obstacle, the usual functorial semantics of inductive types can be generalised to QIITs, by starting off with a category of the sorts of a QIIT, and adding one constructor at a time, where the n^{th} constructor is represented by a pair of functors L_n (the left hand side, or the arguments) and R_n (the right hand side, or the target type, which can either be a sort or an equality). At the end of this process, once all the constructors are added, we end up with a category of 'dependent dialgebras', whose initial object corresponds to the QIIT [2].

This process tells us that *if* a QIIT specification has an initial algebra, then we can describe it in a specific way. However, it doesn't tell us *which* QIIT specifications have initial algebras. This is precisely the problem we aim to tackle, namely, we want to provide a canonical way to represent QIIT specifications that admit an initial algebra, i.e. the strictly positive ones. The way this was achieved for simple inductive types and inductive families was using containers. We take inspiration from this and aim to 'containerify' the semantics given in [2] to obtain semantics for strictly positive QIITs. Although our investigation is in its early stages and most of what follows is conjectural, all of our examples so far have corroborated the results below.

Containerification Our approach involves applying restrictions to the pair of functors L_n and R_n in order to only allow QIIT specifications that are guaranteed to have an initial algebra. One such restriction is ensuring L_n and R_n are container functors. Since these functors will have types $L_n \colon \mathbf{A}_n \to \mathbf{Set}$ and $R_n \colon \int L_n \to \mathbf{Set}$,[1] and since the containers developed so far can only represent endofunctors, we require a more general version of containers. A *generalised container* [4] over an arbitrary category $\mathbf{C}$ is given by a set of shapes $S : \mathsf{Set}$ and a family of positions over the shapes $P \colon S \to |\mathbf{C}|$, written $S \triangleleft P$. This gives rise to a functor $[\![S \triangleleft P]\!] \colon \mathbf{C} \to \mathbf{Set}$, which on objects $X : |\mathbf{C}|$ is defined as $[\![S \triangleleft P]\!]\, X \coloneqq \sum(s : S)(\mathbf{C}(P\, s, X))$.

The first restriction we identified is for L_n and R_n to be generalised container functors. In this case, we would have $S_{L,n} : \mathsf{Set}$ and $P_{L,n} \colon S_{L,n} \to |\mathbf{A_n}|$ and be able to write $L_n \colon \mathbf{A}_n \to \mathbf{Set}$ as

$$L_n\, X \;\cong\; [\![S_{L,n} \triangleleft P_{L,n}]\!]\, X \;=\; \sum(s : S_{L,n})(\mathbf{A_n}(P_{L,n}\, s, X)).$$

We would also have $S_{R,n} : \mathsf{Set}$ and $P_{R,n} \colon S_{R,n} \to |\int L_n|$ with components $P^X_{R,n}, P^s_{R,n}$, and $P^f_{R,n}$. Further restrictions we identified for strict positivity are (i) $S_{R,n} = S_{L,n}$ and (ii) $P^s_{R,n}\, t = t$, which allow us to write $R_n \colon \int L_n \to \mathbf{Set}$ just using the two parameters $P^X_{R,n}$ and $P^f_{R,n}$:

$$\begin{aligned} R_n\,(X,(s,f)) &\cong [\![(t : S_{R,n}) \triangleleft (P^X_{R,n}\, t, (P^s_{R,n}\, t, P^f_{R,n}\, t))]\!]\,(X,(s,f)) \\ &\cong \sum(h : \mathbf{A_n}(P^X_{R,n}\, s, X))(h \circ P^f_{R,n}\, s = f). \end{aligned}$$

Assuming the existence of QIITs in the metatheory, this scheme is general enough to encode both point and path constructors.

A syntax for QIITs The 'containerified' semantics described above also gives rise to a syntax for QIITs. A specification of a QIIT consists of a list of constructors, each of which is specified by the parameters $S_{L,n}, P_{L,n}, P^X_{R,n}$ and $P^f_{R,n}$. We expect this syntax to be a refinement of the theory of signatures presented in [5]. We hope this alternative syntax facilitates a formal reduction from inductive-inductive types to inductive families, and helps us identify a so-called QW-type, which would be a succinct type to which all strictly positive QIITs could be reduced.

[1] $\mathbf{A_n}$ is the category of algebras to which we are adding a constructor, and $\int F$ is the category of elements of F.

References

[1] M. Abbott, T. Altenkirch, and N. Ghani. Containers: Constructing strictly positive types. *Theoretical Computer Science*, 342(1):3–27, 2005. Applied Semantics: Selected Topics.

[2] T. Altenkirch, P. Capriotti, G. Dijkstra, N. Kraus, and F. Nordvall Forsberg. Quotient inductive-inductive types. In C. Baier and U. Dal Lago, editors, *FoSSACS*, pages 293–310. Springer, 2018.

[3] T. Altenkirch, N. Ghani, P. Hancock, C. McBride, and P. Morris. Indexed containers. *Journal of Functional Programming*, 25:e5, 2015.

[4] T. Altenkirch and A. Kaposi. A container model of type theory. In *TYPES 2021*, 2021.

[5] A. Kaposi, A. Kovács, and T. Altenkirch. Constructing quotient inductive-inductive types. *Proc. ACM Program. Lang.*, 3(POPL), 2019.