A container model of type theory*

Thorsten Altenkirch[1] and Ambrus Kaposi[1][2]

[1] University of Nottingham, Nottingham, United Kingdom
[2] Eötvös Loránd University, Budapest, Hungary

Introduction

We present a model of type theory where types are interpreted as containers [1] aka polynomial functors. The motivation for this construction is to provide a container semantics for inductive-inductive types and quotient inductive-inductive types.

Consider the core of our usual example of an inductive-inductive type:

```
data Con : Set
data Ty : Con → Set
_,_ : (Γ : Con) → Ty Γ → Con
Π : (Γ : Con) (A : Ty Γ) (B : Ty (Γ , A)) → Ty Γ
```

Since the constructor Π uses the previously defined constructor _,_ in its domain there is no hope of a functorial semantics where an inductive type is the initial algebra of a container. Instead we have to interpret the domain of a constructor as a functor L from the category of algebras induced by the previous constructors to Set, and the codomain a functor from the category of elements of L to Set. The semantics of a constructor with regard to a fixed algebra X is given by (x : L X) → R (X , x). This is explained in detail in [2] where L is an arbitrary functor and types are interpreted using the usual presheaf semantics of type theory.

Such a functorial semantics is too generous because there are many functors which do not have initial algebras. Hence we want to use containers to model strict positivity semantically. As a first step we show that containers do form a model of basic type theory, i.e. a category with families.

Tamara von Glehn also presents a model of type theory using polynomial functors [7] using comprehension categories as notion of model. The same model was presented by Atkey [4] and by Kovács [6] using categories with families (CwFs). This model has the same contexts and substitutions as ours but different types and terms (see below).

In this abstract when we write Set we mean Agda's universe of types and we assume uniqueness of identity proofs (see also the Discussion).

The model

A container (or polynomial functor) is given by a set of shapes S : Set and a family of positions P : S → Set. This gives rise to a functor S ◁ P : Set → Set which on objects is given by (S ◁ P) X = Σ s : S . P s → X. Given containers S ◁ P and T ◁ Q a morphism is given by a function on shapes f : S → T and a family of functions on positions g : (s : S) → Q (f s) → P s — note the change of direction. This gives rise to a natural transformation f ◁ g : (X : Set) → (S ◁ P) X → (T ◁ Q) X given by

*Supported by USAF, Airforce office for scientific research, award FA9550-16-1-0029.

$(\mathsf{f} \lhd \mathsf{g})\,\mathsf{X}\,(\mathsf{s}\;,\,\mathsf{p}) = (\mathsf{f}\,\mathsf{s}, \lambda\,\mathsf{s} \to \mathsf{p} \circ \mathsf{g}\,\mathsf{s})$. Using the Yoneda lemma we can show that every natural transformation between containers arises this way (i.e. the evaluation functor from the category of containers to the functor category is full and faithful).

We can generalize set-containers to containers over an arbitrary category $\mathbb{C}$, i.e. covariant functors $\mathbb{C} \to \mathsf{Set}$ using $\mathsf{S} : \mathsf{Set}$ and $\mathsf{P} : \mathsf{S} \to \mathbb{C}$ and $(\mathsf{S} \lhd \mathsf{P})\,\mathsf{X} = \Sigma\,\mathsf{s} : \mathsf{S}\,.\,\mathbb{C}\,(\mathsf{P}\,\mathsf{s}, \mathsf{X})$ and morphisms are given by $\mathsf{f} : \mathsf{S} \to \mathsf{T}$ and $\mathsf{g} : (\mathsf{s} : \mathsf{S}) \to \mathbb{C}\,(\mathsf{Q}\,(\mathsf{f}\,\mathsf{s})\;,\,\mathsf{P}\,\mathsf{s})$ with the same definition of the natural transformation.

We define a category with families (CwF) which are the algebras of an intrinsic presentation of Type Theory as given in [3]. The objects corresponding to contexts are set containers and the morphisms are container morphisms. We write Con for this category. Below we sketch some aspects of the construction, for details please check our (incomplete) Agda formalisation [5].

To interpret types we define a functor $\mathsf{Ty} : \mathsf{Con} \to \mathsf{Set}_1$ on objects: given $\Gamma : \mathsf{Con}$, an $\mathsf{A} : \mathsf{Ty}\,\Gamma$ is given by a container $\mathsf{A} : \int \Gamma \to \mathsf{Set}$. Here $\int \Gamma$ is the category of elements of Γ with objects $\Sigma\,\mathsf{X} : \mathsf{Set}\,.\,\Gamma\,\mathsf{X}$ and morphisms $\int \Gamma\,((\mathsf{X}\;,\,\mathsf{x})\;,\,(\mathsf{Y}\;,\,\mathsf{y})$ are given by a function $\mathsf{f} : \mathsf{X} \to \mathsf{Y}$ such that $\Gamma\,\mathsf{f}\,\mathsf{x} = \mathsf{y}$.

In contrast, a type in von Glehn's model [7] over a context $\Gamma = \mathsf{S} \lhd \mathsf{P}$ is a container $\mathsf{A} : \int (\mathsf{Const}\;\mathsf{S}) \to \mathsf{Set}$ where $\mathsf{Const}\;\mathsf{S}$ is the constant S presheaf. Hence types there are dependent only on shapes, but not positions.

The interpretation of terms is given by $\mathsf{Tm} : \int \mathsf{Ty} \to \mathsf{Set}$. On an object of $\int \mathsf{Ty}$, i.e. a $\Gamma : \mathsf{Con}$ and $\mathsf{A} : \int \Gamma \to \mathsf{Set}$ a term is given by a *dependent natural transformation* $(\mathsf{X} : \mathsf{Set})\,(\mathsf{x} : \Gamma\,\mathsf{X}) \to \mathsf{A}\,(\mathsf{X}\;,\,\mathsf{x})$. Assuming that $\Gamma = \mathsf{S}_\Gamma \lhd \mathsf{P}_\Gamma$ and $\mathsf{A} = \mathsf{S}_\mathsf{A} \lhd \mathsf{P}_\mathsf{A}$ using the *dependent Yoneda lemma* we can show that this corresponds to $\mathsf{f} : \mathsf{S}_\Gamma \to \mathsf{S}_\mathsf{A}$ and $\mathsf{g} : (\mathsf{s} : \mathsf{S}_\mathsf{A}) \to \int \Gamma\,(\mathsf{P}_\mathsf{A}\,\mathsf{s}\;,\,(\mathsf{P}_\Gamma\,(\mathsf{P}^s_\mathsf{A}\,\mathsf{s})\;,\,\mathsf{P}^s_\mathsf{A}\,\mathsf{s}\;,\,\mathsf{id}))$. This can be further simplified using dependent types — see our Agda code.

Given $\mathsf{A} : \mathsf{Ty}\,\Gamma$ we write $\mathsf{P}^X_\mathsf{A} : \mathsf{S}_\mathsf{A} \to \mathsf{Set}$, $\mathsf{P}^s_\mathsf{A} : \mathsf{S}_\mathsf{A} \to \mathsf{S}_\Gamma$ and $\mathsf{P}^g_\mathsf{A} : (\mathsf{s} : \mathsf{S}_\mathsf{A}) \to \mathsf{P}_\Gamma\,(\mathsf{P}^s_\mathsf{A}\,\mathsf{s}) \to \mathsf{P}^X_\mathsf{A}\,\mathsf{s}$ for the projections of $\mathsf{P}_\mathsf{A} : \mathsf{S}_\mathsf{A} \to \int \Gamma$.

The empty context is given by the terminal object in Con which is $1 \lhd 0$. Assuming a $\Gamma : \mathsf{Con}$ and $\mathsf{A} : \mathsf{Ty}\,\Gamma$ we construct the context extension $\Gamma\;,\,\mathsf{A}$ as $\mathsf{S}_\mathsf{A} \lhd \mathsf{P}^X_\mathsf{A}$. We can verify the universal property — see the Agda code.

The definition of type substitution requires pushouts which can be defined using a quotient inductive type (QIT). That is given $\mathsf{f} \lhd \mathsf{g} : \mathsf{Con}\,(\Delta\;,\,\Gamma)$ and $\mathsf{A} : \mathsf{Ty}\,\Gamma$ we construct $\mathsf{A}[\mathsf{f} \lhd \mathsf{g}] = \mathsf{S} \lhd \mathsf{P} : \mathsf{Ty}\,\Gamma$. We obtain S as the pullback of f and P^s_A. Given $\mathsf{s} : \mathsf{S}$, $\mathsf{P}\,\mathsf{s}$ is the pushout of $\mathsf{g}\,\mathsf{s}$ and $\mathsf{P}^g_\mathsf{A}\,\mathsf{s}$ (this only type-checks after transporting along the equations).

Discussion

For our application we need to construct the model wrt to an already constructed category of algebras instead of Set. Hence we need to verify that pushouts exists. We need a constructive variant of locally presentable categories here, which have the required colimits.

For our application we only need a basic CwF structure but we can also interpret Σ-types and we expect that we can interpret Π-types in our model, the latter giving rise to higher-order abstract syntax.

One issue with our construction is that types and contexts are not h-sets hence we need to address the coherence issues. We believe that this fits very well with a generalisation of CwFs where types can be groupoids (or 1-types) which we are also investigating (coherent CwFs).

References

[1] Michael Gordon Abbott. *Categories of containers*. PhD thesis, University of Leicester, 2003.

[2] Thorsten Altenkirch, Paolo Capriotti, Gabe Dijkstra, Nicolai Kraus, and Fredrik Nordvall Forsberg. Quotient inductive-inductive types. In *International Conference on Foundations of Software Science and Computation Structures*, pages 293–310. Springer, Cham, 2018.

[3] Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '16, pages 18–29, New York, NY, USA, 2016. ACM.

[4] Robert Atkey. Interpreting dependent types with containers. Talk given at the MSP101 seminar of the University of Strathclyde. Slides available at https://bentnib.org/docs/tt-in-containers.pdf and formalisation at https://gist.github.com/bobatkey/0d1f04057939905d35699f1b1c323736., June 2020.

[5] Ambrus Kaposi and Thorsten Altenkirch. Agda formalisation of the container model of type theory, available at https://bitbucket.org/akaposi/qiitcont, 2021.

[6] András Kovács. The container model of type theory. Talk given at the type theory seminar of Eötvös Loránd University. Formalisation available at https://bitbucket.org/akaposi/tipuselmelet/src/master/notes/seminar20200623_container.agda., June 2020.

[7] Tamara von Glehn. *Polynomials and models of type theory*. PhD thesis, University of Cambridge, 2014.