Revisiting Containers in Cubical Agda

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 a type-theoretic literature review of the state of the art on containers, as well as a Cubical Agda formalisation of generalised containers.

Strict positivity An *inductive type* X is a type given by a list of constructors, each specifying a way to form an element of X. Defining types inductively is a central notion in Martin-Löf type theory, with examples including the natural numbers $\mathbb{N}$, lists, finite sets, and many more. In this setting, we usually want to be able to make sense of our inductive definitions predicatively, with elements of the type being generated 'in stages'. The condition we would like to impose on our definitions is that they are *strictly positive*. This roughly means that the constructors of X only allow X to appear in input types that are arrows if it appears to the right. So we allow constructors like $\mathtt{c}\colon(\mathbb{N} \to \mathtt{X}) \to \mathtt{X}$, but not $\mathtt{d}\colon(\mathtt{X} \to \mathbb{N}) \to \mathtt{X}$ or $\mathtt{e}\colon((\mathtt{X} \to \mathbb{N}) \to \mathbb{N}) \to \mathtt{X}$. In general, we want to avoid definitions that are not strictly positive, as they can lead to inconsistencies under certain assumptions (like classical logic), so we would like a semantic description of strict positivity in order for our systems to only admit such types. Containers help us do exactly this.

What are containers? A (ordinary) *container* $S \triangleleft P$ is a set of shapes $S : \mathsf{Set}$ and a family of positions over those shapes $P\colon S \to \mathsf{Set}$. Every strictly positive type can be thought of as a well-founded tree whose nodes are labelled by elements s of S, and where node s has $P\,s$ many subtrees. E.g. the `List` data type is given as a container by $(n : \mathbb{N}) \triangleleft (\mathtt{Fin}\,n)$. The shape of a list is a natural number $n : \mathbb{N}$ representing its length, and given a length n, the data of a list is stored at the positions, which are the elements of a finite set of size n, $\mathtt{Fin}\,n$.

To every container $S \triangleleft P$, we associate a functor $[\![S \triangleleft P]\!]\colon \underline{\mathbf{Set}} \to \underline{\mathbf{Set}}$ defined as follows.

- On objects $X : \mathsf{Set}$, we have $[\![S \triangleleft P]\!]\,X \coloneqq \sum(s : S)(P\,s \to X)$.
- On morphisms $f\colon X \to Y$, we have $[\![S \triangleleft P]\!]\,f\,(s, g) \coloneqq (s, f \circ g)$.

This functor reflects the idea that strictly positive types are simply memory locations in which data can be stored. E.g. the container functor $[\![(n : \mathbb{N}) \triangleleft (\mathtt{Fin}\,n)]\!]$ allows us to represent concrete lists. The list of `Char`s ['r', 'e', 'd'] is represented as $(3, (0 \mapsto \text{'}r\text{'}; 1 \mapsto \text{'}e\text{'}; 2 \mapsto \text{'}d\text{'}) : \sum (n : \mathbb{N})\,(\mathtt{Fin}\,n \to \mathtt{Char})$. Containers are also known in the literature as polynomial functors [9, 10]. W-types are the initial algebras of container functors.

Our contribution Over the years, containers have been studied extensively [1, 7, 5]. Some of the key developments on containers are presented using a heavily category-theoretic approach—in particular, they are presented as constructions in the internal language of locally Cartesian closed categories (LCCCs) with disjoint coproducts and W-types (also called Martin-Löf categories). We felt that adapting these results using a more type-theoretic approach would be beneficial for a few reasons. Firstly, using LCCCs to describe models of dependent type theory is too restrictive and not entirely precise (e.g. setoids are not an LCCC [11] but still model dependent type theory). Secondly, we wanted a more accessible presentation of containers for programmers and computer scientists, who might have less of a thorough background in category theory.

To this end, we present ongoing work on a review paper offering a comprehensive and updated type-theoretic view of the state of the art on containers [4]. The paper presents all the established results on (ordinary) containers, discusses other kinds of containers, introduces generalised containers [6], and does so in the language of type theory. To supplement this study, we formalised several results on containers in Cubical Agda [8]. We have two proofs in Cubical Agda of the central result that the container extension functor $[\![\_]\!]$ mapping containers to functors is full and faithful. This was proven for the case of generalised containers, which generalise ordinary containers in that they are parameterised by an arbitrary category $\underline{\mathbf{C}}$ and give rise to functors of type $\underline{\mathbf{C}} \to \underline{\mathbf{Set}}$. One follows the proof given in [1], and the other is a new proof that makes use of the Yoneda lemma. While these two proofs are fully formalised, the review paper as well as a formalisation of additional results is work in progress.

One of the consequences of $[\![\_]\!]$ being full and faithful is that we obtain a characterisation of natural transformations between container functors (i.e. polymorphic functions on strictly positive types) as container morphisms. A container morphism $(S \lhd P) \to (T \lhd Q)$ is a pair $u : S \to T$ and $f : (s : S) \to Q\,(u\,s) \to P\,s$, e.g. container morphisms between lists are given by a pair $u : \mathbb{N} \to \mathbb{N}$ and $f : (n : \mathbb{N}) \to \texttt{Fin}\ \ (u\,n) \to \texttt{Fin}\ \ n$. This result tells us that *any* polymorphic function on lists (such as tail and reverse) can be represented as such a pair, supporting the claim that containers are a canonical way of representing strictly positive types.

Our formalisation makes use of the category theoretic definitions available in the Cubical library. The Cubical mode of Agda avoids us having to postulate functional extensionality, facilitates the use of heterogenous equality, and allows for future generalisations related to higher inductive types (discussed below).

Future work Our survey of containers was primarily motivated by our current interest in applying them to obtain semantics for quotient inductive-inductive types (QIITs). Our end goal is to provide a canonical way to represent QIIT specifications that admit an initial algebra, i.e. the strictly positive ones. Our approach is to 'containerify' the semantics given in [2] to obtain a semantics for strictly positive QIITs. More details on this can be found at [3].

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 and S. Damato. Specifying QIITS using containers. Abstract submitted to HoTT/UF, 2023. Available at https://stefaniatadama.com/talks/abstract_hott_uf_2023.pdf.

[4] T. Altenkirch and S. Damato. A type-theoretic survey of containers, 2023. In preparation.

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

[6] T. Altenkirch and A. Kaposi. A container model of type theory. TYPES abstract, 2021.

[7] T. Altenkirch, P. Levy, and S. Staton. Higher-order containers. In F. Ferreira, B. Löwe, E. Mayordomo, and L. Mendes Gomes, editors, *Programs, Proofs, Processes*, pages 11–20, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[8] Stefania Damato. A formalisation of generalised containers in Cubical Agda. Formalisation, 2023. Available at https://github.com/stefaniatadama/TYPES-23.

[9] N. Gambino and M. Hyland. Wellfounded trees and dependent polynomial functors. In S. Berardi, M. Coppo, and F. Damiani, editors, *Types for Proofs and Programs*, pages 210–225, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[10] J. Kock. Notes on polynomial functors, October 2009. Available at https://mat.uab.cat/~kock/cat/polynomial.pdf.

[11] N. Kraus and T. Altenkirch. Setoids are not an LCCC. Unpublished note, 2012. Available at https://www.cs.nott.ac.uk/~psznk/docs/setoids.pdf.