# Containers: Compositionality for Tensors

Neil Ghani[1], Pierre Hyvernat[2], and Artjoms Šinkarovs[3]

[1] Kodamai, Glasgow, Scotland. neil@kodamai.com
[2] Université Savoie Mont Blanc, Chambéry, France. pierre.hyvernat@univ-smb.fr
[3] University of Southampton, Southampton, UK. a.sinkarovs@soton.ac.uk

**Abstract**

While tensors are an important computational device, for example in machine learning [3] or physics [4], it seems type theory has had little to say about them. This work in progress is predicated on two principles widely accepted within the Types community: i) compositionality is a key approach enabling the construction of complex structures from simpler, and hence easier to define, structures; and ii) containers have an extremely rich compositional algebra [1]. Given these observations, one might wonder if containers can be used to develop a compositional approach to tensors and that is exactly what this abstract does.

## 1 Introduction

Tensors are actively used in many unrelated areas of computer science and mathematics. There are at least two ways to understand them. On the one hand, they are just multi-dimensional arrays, and this is the angle taken in array languages such as APL. On the other hand, tensors generalise matrices in the following way. If matrix is a normal form for linear functions, where matrix multiplication is function composition, tensors are normal forms for multi-linear maps from sets of vector and co-vector spaces into the underlying field. In both cases, tensors and their operations expose a lot of structure that is interesting to study in the context of type theory.

Despite their importance, tensor algebra tends to be seen as a tool with fairly concrete representation based on the manipulation of lists of natural number-valued indices — structure-preserving tensor operations are thus formalised via list-theoretic computation. While good for fast implementations, this approach suffers from a lack of high-level abstraction common to all modern development in type theory. As a result, proving properties of tensor inside systems such as Agda or Coq is rather cumbersome. We developed a container-based approach to tensors where properties of tensor operations are mapped onto well-known type-theoretic constructions. This is still a work in progress, but the clean and general treatment of, for example, reshaping via container morphisms, makes us confident this approach has something to bring.

**Containers** Containers [1] (also known as polynomial functors [2]) were introduced to study concrete data types. This is a powerful construction as it captures the notion of strictly positive data types, and it is closed under operations such as disjoint sum, product and many others. A container is a pair $(S, P)$ where $S : \mathsf{Set}$ and $P : S \to \mathsf{Set}$. The set $S$ is called the set of shapes and can be thought of as the constructors of a data type. Each constructor/shape $s \in S$ has an arity $P\ s$ which we call the positions of that shape. A container $(S, P)$ is a presentation of the functor $[\![S, P]\!] : \mathsf{Set} \to \mathsf{Set}$ defined by

$$[\![S, P]\!]\ X = (\Sigma s : S) P\ s \to X$$

Elements in $[\![S, P]\!]\ X$ intuitively consist of the choice of a constructor and an assignment of a piece of data (here, $X$) to every position of that constructor. The fundamental theorem of

containers is a classification of the natural transformations between those: a natural transformation $f : [\![S, P]\!] \to [\![Q, R]\!]$ is uniquely given by: i) a function on shapes $u : S \to Q$; and ii) a contravariant re-indexing $t : (\Pi s : S)\ (R\ (u\ s) \to P\ s)$. Such pairs are called container morphisms and form a category $\mathsf{Cont}$ of containers.

Containers are known to support a large number of constructions making them good for compositional modelling. We use the coproduct which is defined as follows. Let $(S, P)$ and $(S', P')$ be containers. Their coproduct has shapes $S + S'$ with positions the cotuple $[P, P']$.

## 2 From Containers to Tensors

We change the usual perspective on containers by writing $(A, I)$ to reflect i) $A : \mathsf{Set}$ will correspond to the set of axes (or dimensions) of a tensor; and ii) $I : A \to \mathsf{Set}$ assigns to a specific axis $a : A$, a set of indexes $I\ a$ on that axis. Consider a traditional two-dimensional $(n \times m)$-matrix which can be given as $A = 2$, $I\ (\mathsf{inl}\ *) = \mathsf{Fin}\ n$ and $I\ (\mathsf{inr}\ *) = \mathsf{Fin}\ m$. With $A : \mathsf{Set}$, there is no order on axes and hence we don't need the complication of re-ordering axes. Note that to get $(n \times m)$ matrices, the usual interpretation of containers needs to be changed:

$$[\![A, I]\!]_\Pi\ X = (\Pi\ A\ I) \to X$$

Note that $[\![-]\!]_\Pi : \mathsf{Cont} \to [\mathsf{Set}, \mathsf{Set}]$ is very different from the usual container interpretation. It is functorial and indeed $[\![-]\!]_\Pi = Y \circ \Pi$ where $Y$ is the Yoneda embedding. While $[\![-]\!]_\Pi$ does not have the full range of compositional operators supported by $[\![-]\!]$, we do have: i) $[\![M]\!]_\Pi \circ [\![N]\!]_\Pi = [\![M + N]\!]_\Pi$; and ii) $[\![M]\!]_\Pi\ X \times [\![N]\!]_\Pi\ Y \to [\![M]\!]_\Pi\ (X \times Y)$. These properties give rise to the $\mathsf{pair}, \mathsf{nest}/\mathsf{unnest}, \mathsf{map}$ combinators:

$$\mathsf{pair} : [\![M]\!]_\Pi\ X \to [\![M]\!]_\Pi\ Y \to [\![M]\!]_\Pi\ (X \times Y)$$

$$\mathsf{nest} : [\![M + M']\!]_\Pi\ X \cong [\![M]\!]_\Pi\ ([\![M']\!]_\Pi\ X) : \mathsf{unnest} \qquad \mathsf{map} : (X \to Y) \to [\![M]\!]_\Pi\ X \to [\![M]\!]_\Pi\ Y$$

Reshaping is an important operation in array languages. Reshaping turns structural changes in shapes into actions on arrays, and it can be used to guide recursive traversals through array elements [5]. For tensors, reshaping can be given by container morphism and their action is defined as follows:

$$\mathsf{reshape} : \mathsf{Cont}(M, M') \to [\![M]\!]_\Pi\ X \to [\![M']\!]_\Pi\ X$$

Tensor contraction is one of the key operation in tensor calculus, and we can define this as follows:

$$\mathsf{matmul} : [\![M_1 + M_2]\!]_\Pi\ X \to [\![M_2 + M_3]\!]_\Pi\ X \to [\![M_1 + M_3]\!]_\Pi\ X$$

under the assumption that $M_2$ is finite and $X$ is a ring. This operation also gives rise to the category of Tensors where objects are containers, morphisms from $s$ to $p$ are $[\![s + p]\!]_\Pi\ X$, and composition is given by $\mathsf{matmul}$. Note that under this interpretation $[\![s + p]\!]_\Pi\ X$ is a tensor of $s$ co-vectors and $p$ (contravariant) vectors. Empty container $\mathbb{0}$ and the singleton container $\mathbb{1}$ give rise to singleton tensors which are usually referred as scalars. Any $[\![s]\!]_\Pi\ X$ can be turned into a "column vector" $[\![s + \mathbb{1}]\!]_\Pi\ X$ or a "row vector" $[\![\mathbb{1} + s]\!]_\Pi\ X$. Here distinction between vectors and co-vectors is not as strong as in the actual tensor calculus, but we have the structure to make it precise. Our work-in-progress formalisation can be found at https://github.com/ashinkarov/2025-types.

# References

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

[2] Nicola Gambino and Martin Hyland. Wellfounded trees and dependent polynomial functors. In Stefano Berardi, Mario Coppo, and Ferruccio Damiani, editors, *Types for Proofs and Programs*, pages 210–225, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[3] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012.

[4] P. Renteln. *Manifolds, Tensors and Forms*. Cambridge University Press, 2014.

[5] Artjoms Šinkarovs, Thomas Koopman, and Sven-Bodo Scholz. Rank-polymorphism for shape-guided blocking. In *Proceedings of the 11th ACM SIGPLAN International Workshop on Functional High-Performance and Numerical Computing*, FHPNC 2023, page 1–14, New York, NY, USA, 2023. Association for Computing Machinery.