Quantitative Polynomial Functors

Georgi Nakov
Department of Computer and Information Sciences, University of Strathclyde, UK

Fredrik Nordvall Forsberg
Department of Computer and Information Sciences, University of Strathclyde, UK

Abstract

We investigate containers and polynomial functors in Quantitative Type Theory, and give initial algebra semantics of inductive data types in the presence of linearity. We show that reasoning by induction is supported, and equivalent to initiality, also in the linear setting.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Linear logic

Keywords and phrases quantitative type theory, polynomial functors, inductive data types

Digital Object Identifier 10.4230/LIPIcs.CALCO.2021.22

Category Early Ideas

1 Quantitative Type Theory

Quantitative Type Theory (QTT) [3, 12] combines linear and dependent types, allowing for tracking and reasoning about resource usage of programs. Such a combination is non-trivial as there is no obvious answer how to treat terms occurring in type formation. Previous attempts include the Linear Logical Framework [6], and the work of Krishnaswami, Pradic and Benton [11] and Vákár [14], based on Linear/Non-Linear logic [5], in which the context is split into intuitionistic and linear parts, and each type is allowed to depend on only one of the two. QTT differs by maintaining a single context, where each variable is annotated with resource information (for a similar approach, see Orchard et al. [13], Fu et al. [8] and Abel and Bernardy [2]). For example, consider the judgement (where $\mathsf{Fin} : \mathbb{N} \to \mathsf{Type}$ is a type family with $\mathsf{Fin}(n)$ consisting of natural numbers smaller than n):

$$n \overset{0}{:} \mathbb{N}, x \overset{2}{:} \mathsf{Fin}(n) \vdash x + x \overset{1}{:} \mathsf{Fin}(2n)$$

The variables on the left of the turnstile form the context of the judgement. Each one is annotated with a quantity, taken from a fixed semiring R (here, $R = (\mathbb{N}, +, *)$). These quantities denote how many the times the variables must be used in the term on the right. Hence here we see that n is used 0 times, since it only occurs in types, and x is used twice. A context can be pointwise scaled by an element $\pi \in R$, and two contexts with the same "underlying non-resource-annotated context" can be added pointwise:

$$\pi(\Gamma, x \overset{\rho}{:} S) = \pi\Gamma, x \overset{\pi\rho}{:} S$$
$$(\Gamma_1, x \overset{\rho_1}{:} S) + (\Gamma_2, x \overset{\rho_2}{:} S) = (\Gamma_1 + \Gamma_2), x \overset{\rho_1+\rho_2}{:} S, \text{ if } 0\Gamma_1 = 0\Gamma_2.$$

While arbitrary elements from R may occur as annotations in the context, only the 0 and 1 are valid annotations for the conclusion, in order to retain admissibility of substitution [3, § 2.3]. Such a restriction effectively splits the theory in two fragments. Terms in the so-called

$$\frac{0\Gamma \vdash X : \mathsf{Type} \quad 0\Gamma, x \overset{0}{:} X \vdash Y : \mathsf{Type}}{0\Gamma \vdash (x \overset{\rho}{:} X) \to Y : \mathsf{Type}} \qquad \frac{\Gamma, x \overset{\rho}{:} X \vdash t : Y}{\Gamma \vdash \lambda x.\, t : (x \overset{\rho}{:} X) \to Y}$$

$$\frac{\Gamma_1 \vdash f : (x \overset{\rho}{:} X) \to Y \quad \Gamma_2 \vdash t : S \quad 0\Gamma_1 = 0\Gamma_2}{\Gamma_1 + \pi\Gamma_2 \vdash f(t) : Y[t/x]}$$

$$\frac{0\Gamma \vdash X : \mathsf{Type} \quad 0\Gamma, x \overset{0}{:} X \vdash Y : \mathsf{Type}}{0\Gamma \vdash (x \overset{\rho}{:} X) \otimes Y : \mathsf{Type}} \qquad \frac{\Gamma_1 \vdash s : X \quad \Gamma_2 \vdash t : Y[s/x] \quad 0\Gamma_1 = 0\Gamma_2}{\rho\Gamma_1 + \Gamma_2 \vdash (s,t) : (x \overset{\rho}{:} X) \otimes Y}$$

$$\frac{0\Gamma_1, z \overset{0}{:} (x \overset{\rho}{:} X) \otimes Y \vdash U : \mathsf{Type} \quad \Gamma_1 \vdash w : (x \overset{\rho}{:} X) \otimes Y \quad \Gamma_2, x \overset{\rho}{:} X, y \overset{1}{:} Y \vdash u : U[(x,y)/z] \quad 0\Gamma_1 = 0\Gamma_2}{\Gamma_1 + \Gamma_2 \vdash \ \text{let } (x,y) = s \text{ in } t : U[w/z]}$$

Figure 1 Typing rules for the dependent function and dependent tensor types.

0-fragment bear no computational content, while the inhabitants of the 1-fragment are computationally relevant[1]. The core insight is that contemplating variables in types is always possible, even for already consumed ones, and thus type formation happens in the 0-fragment.

The typing rules in the system now carry resource tracking duty, which is especially evident in types with binders – see Figure 1. Function type now records how many times its arguments are used; e.g. a function $f : (x \overset{\rho}{:} X) \to Y$ must be supplied with ρ many copies of x. (If the type family Y does not depend on X, we will use the simplified notation $f : X \xrightarrow{\rho} Y$ for $f : (x \overset{\rho}{:} X) \to Y$.) In a similar vein, the second component of the dependent pair $t : (x \overset{\rho}{:} A) \otimes B$ requires ρ many copies of the first component.

2 Quantitative Containers

We want to extend QTT with data types to program with and reason about, but an ad-hoc approach of writing out introduction and elimination rules for each data type is cumbersome and error-prone. A principled solution to adding inductive data types in a traditional setting is provided by the theory of polynomial functors [9] and containers [1]. We build the syntactic category of closed types and linear functions in QTT and define polynomial functors on it. We can then systematically derive appropriate elimination rules for their initial algebras.

2.1 Quantitative container functors on the category of (closed) types and linear functions

In traditional type theory, a container functor is a functor on types of the form $F_{S,P}(X) = (s : S) \times (P[s] \to X)$, where $(x : A) \times B[x]$ is the dependent pair type. We think of the parameter S as a type of *shapes*, and the family $P : S \to \mathsf{Type}$ as a type of *positions* for each shape. An element of $F_{S,P}(X)$ is describing how to fill the container with "payloads" from X: a choice of a shape $s : S$, and an element of X for every position in $P[s]$. The following is the obvious transfer of this idea to a quantitative setting:

[1] From now on, we omit the annotation on terms in the 1-fragment on the right of the turnstile in typing judgements – that is, $\Gamma \vdash t : A$ tacitly means $\Gamma \vdash t \overset{1}{:} A$.

▶ **Proposition 1.** *Let $\mathcal{C}$ be the category of closed types and linear functions: its objects are types $\vdash X$, and morphisms are functions $\vdash f : X \xrightarrow{1} Y$. For fixed $S : \mathsf{Type}$ and $P : S \xrightarrow{0} \mathsf{Type}$, the mapping $F_{S,P}(X) = (s \overset{1}{:} S) \otimes (P[s] \xrightarrow{1} X)$ is a functor $\mathcal{C} \to \mathcal{C}$.*

We call any functor isomorphic to one of the form $F_{S,P}$ a *quantitative container*. The above proposition can be generalised to types and functions over an arbitrary, fixed context of the shape $\Gamma = 0\Gamma$, i.e. where all variables are annotated with 0.

2.2 Induction principles and initial algebras

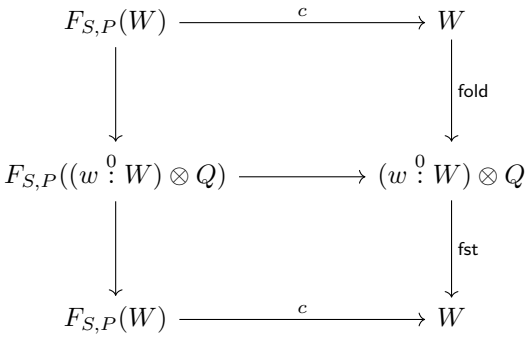
Recall that an F-algebra for an endofunctor $F : \mathcal{C} \to \mathcal{C}$ is a pair $(A, a : F(A) \to A)$, where A is an object of $\mathcal{C}$ and a is a $\mathcal{C}$-morphism. A morphism between F-algebras (A, a) and (B, b) is a map $f : A \to B$ in $\mathcal{C}$, such that $f \circ a = b \circ F(f)$. Inductive data types are initial algebras: the algebra map corresponds to the introduction rule of the type, and the mediating map corresponds to the elimination rule. A priori, this only gives *non-dependent* elimination rules (also known as recursion principles), but by exploiting the uniqueness of the mediating map, we can also derive *dependent* elimination rules (induction principles), and vice versa:

▶ **Theorem 2.** *Let $\mathbf{W} := (W, c : F_{S,P}(W) \xrightarrow{1} W)$ be an $F_{S,P}$-algebra. $\mathbf{W}$ is initial if and only if the following induction principle holds:*

$$\frac{w \overset{0}{:} W \vdash Q : \mathsf{Type} \quad \vdash M : (s \overset{1}{:} S) \to (h \overset{0}{:} P[s] \xrightarrow{1} W) \to ((p \overset{1}{:} P[s]) \to Q(h(p))) \xrightarrow{1} Q(c(s,h))}{\vdash \mathsf{elim}(Q, M) : (x \overset{1}{:} W) \to Q[x]}$$

Proof (sketch). The proof is based on the standard construction (see e.g. Awodey, Gambino and Sojakova [4] or Hermida and Jacobs [9]), but accounting for linearity.

Assuming that $\mathbf{W}$ is initial and the premises of the elimination rule, we build an $F_{S,P}$-algebra on the dependent tensor type $(w \overset{0}{:} W) \otimes Q$, and get a unique mediating morphism $\mathsf{fold} : W \xrightarrow{1} (w \overset{0}{:} W) \otimes Q$ by initiality. We compose with the second projection $\mathsf{snd} : (x \overset{1}{:} (w \overset{0}{:} W) \otimes Q) \to Q[\mathsf{fst}(x)]$ to get a map $(x \overset{1}{:} W) \to Q[\mathsf{fst}(\mathsf{fold}(x))]$. Note that the use of second projection is admissible due to the annotation of the first component $w \overset{0}{:} W$ – we are free to dispose of w, since it is used 0 times. To show that $\mathsf{snd} \circ \mathsf{fold}$ has the right type, we need to show that $Q[\mathsf{fst}(\mathsf{fold}(x))] = Q[x]$ for every $x : W$, but as this is a type equality, unrestricted use of terms is permissible. The map $\mathsf{fst} : (w \overset{0}{:} W) \otimes Q \xrightarrow{1} W$ is an $F_{S,P}$-algebra morphism, and thus the composite $\mathsf{fst} \circ \mathsf{fold} : W \xrightarrow{1} W$ is also one:



Thus $\mathsf{fst} \circ \mathsf{fold} = \mathsf{id}$ holds by uniqueness of the mediating morphism out of W.

The converse direction follows analogously. ◀

3 Quantitative polynomial functors

However, there is a caveat – most quantitative versions of standard data types are not quantitative containers in the above sense. Consider, for example, the natural numbers, the initial algebra of the polynomial functor $F(X) = \mathbf{1} + X$, or binary trees, the initial algebra of $G(X) = A + X \times X$. Their representations in "container normal form" crucially depend on isomorphisms $(\mathbf{0} \to X) \cong \mathbf{1}$ and $(\mathbf{Bool} \to X) \cong X \times X$, respectively, but their QTT counterparts do not hold: $(\mathbf{0} \xrightarrow{1} X) \ncong \mathbf{I}$ (where $\mathbf{I}$ is the monoidal unit) and $(\mathbf{Bool} \xrightarrow{1} X) \ncong X \otimes X$. Thus we resort to generating the class of *quantitative polynomial functors* inductively by the following grammar:

$$F, G ::= \mathsf{Id} \mid \mathsf{Const}_A \mid F \underline{\otimes} G \mid F \underline{\oplus} G \mid F \,\underline{\&}\, G \mid A \underline{\xrightarrow{1}} X \tag{1}$$

Theorem 2 still holds for this class of functors, with the induction principle reformulated using an appropriately defined predicate lifting $\widehat{F}_X : (Q : X \to \mathsf{Type}) \to (F(X) \to \mathsf{Type})$ for the type of induction hypothesis:

▶ **Theorem 3.** *Let F be a quantitative polynomial functor and $\mathbf{W} := (W, c : F(W) \xrightarrow{1} W)$ an F-algebra. $\mathbf{W}$ is initial if and only if the following induction principle holds:*

$$\frac{w \overset{0}{:} W \vdash Q : \mathsf{Type} \quad \vdash M : ((w \overset{0}{:} F(W)) \otimes \widehat{F}_W(Q, w)) \xrightarrow{1} Q(c(w))}{\vdash \mathsf{elim}(Q, M) : (x \overset{1}{:} W) \to Q[x]}$$

Our proof is a variation on the proof of Theorem 2, using a distributive lemma for the dependent tensor and the predicate lifting:

▶ **Lemma 4.** *For a quantitative polynomial functor F, we have*

$$F((w \overset{0}{:} W) \otimes Q) \cong (w' \overset{0}{:} F(W)) \otimes \widehat{F}_W(Q, w').$$

4 Existence of initial algebras

Categorical models of QTT build upon Categories with families (CwF) [7], a standard framework for models of dependent type theories. A Quantitative CwF [3] consists of an ordinary CwF $\mathcal{C}$, a category of resourced contexts and substitutions $\mathcal{L}$, and data to interpret resourced counterparts of context extensions and terms. A faithful functor $U : \mathcal{L} \to \mathcal{C}$ relates the computationally relevant 1-fragment in $\mathcal{L}$ to the 0-fragment in $\mathcal{C}$.

A concrete model is given by choosing $\mathcal{C} = \mathcal{S}et$ and $\mathcal{L}$ to be the category of assemblies [15] using linear realisability [10]. We can prove existence of initial algebras of finitary quantitative polynomial functors (i.e., generated without $A \underline{\xrightarrow{1}} X$) in the model by reusing the initial algebras in $\mathcal{S}et$, using their universal properties to construct the realisers. Unfortunately, this method does not extend to non-finitary polynomial functors (i.e. those generated using $A \underline{\xrightarrow{1}} X$ in the grammar (1)), since the case for the type constructor $A \underline{\xrightarrow{1}} X$ gives a realiser $r(a)$ for every $a : A$, but not a realisable function. In the future, we hope to overcome this shortcoming.

References

1 Michael Abbott, Thorsten Altenkirch, and Neil Ghani. Categories of containers. *Lecture Notes in Computer Science*, 2620:23–38, 2003. doi:10.1007/3-540-36576-1_2.

2 Andreas Abel and Jean-Philippe Bernardy. A unified view of modalities in type systems. *Proceedings of the ACM on Programming Languages*, 4(ICFP):1–28, 2020. doi:10.1145/3408972.

3 Robert Atkey. Syntax and Semantics of Quantitative Type Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science - LICS '18*, pages 56–65, New York, New York, USA, 2018. ACM Press. doi:10.1145/3209108.3209189.

4 Steve Awodey, Nicola Gambino, and Kristina Sojakova. Homotopy-initial algebras in type theory. *Journal of the ACM*, 63(6), 2017.

5 P. N. Benton. A mixed linear and non-linear logic: Proofs, terms and models. In *Lecture Notes in Computer Science*, volume 933, pages 121–135. Springer, Berlin, Heidelberg, 1995. doi:10.1007/BFb0022251.

6 Iliano Cervesato and Frank Pfenning. A Linear Logical Framework. *Information and Computation*, 179(1):19–75, 2002. doi:10.1006/inco.2001.2951.

7 Peter Dybjer. Internal type theory. In *Lecture Notes in Computer Science*, volume 1158 LNCS, pages 120–134. Springer, Berlin, Heidelberg, 1996. doi:10.1007/3-540-61780-9_66.

8 Peng Fu, Kohei Kishida, and Peter Selinger. Linear Dependent Type Theory for Quantum Programming Languages: Extended Abstract. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2020: Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 440–453. Association for Computing Machinery, 2020. doi:10.1145/3373718.3394765.

9 Claudio Hermida and Bart Jacobs. Structural Induction and Coinduction in a Fibrational Setting. *Information and Computation*, 145(2):107–152, 1998. doi:10.1006/inco.1998.2725.

10 Naohiko Hoshino. Linear Realizability. In *Computer Science Logic*, volume 4646 LNCS, pages 420–434. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. doi:10.1007/978-3-540-74915-8_32.

11 Neelakantan R. Krishnaswami, Pierre Pradic, and Nick Benton. Integrating Linear and Dependent Types. *ACM SIGPLAN Notices*, 50(1):17–30, 2015. doi:10.1145/2775051.2676969.

12 Conor McBride. I Got Plenty o' Nuttin'. In *Lecture Notes in Computer Science*, volume 9600, pages 207–233. Springer Verlag, 2016. doi:10.1007/978-3-319-30936-1_12.

13 Dominic Orchard, Vilem-Benjamin Liepelt, and Harley Eades III. Quantitative program reasoning with graded modal types. *Proceedings of the ACM on Programming Languages*, 3(ICFP):110:1–110:30, 2019. doi:10.1145/3341714.

14 Matthijs Vákár. *In search of effectful dependent types*. PhD thesis, University of Oxford, 2017.

15 Jaap van Oosten. *Realizability: an introduction to its categorical side*. Elsevier, 2008.