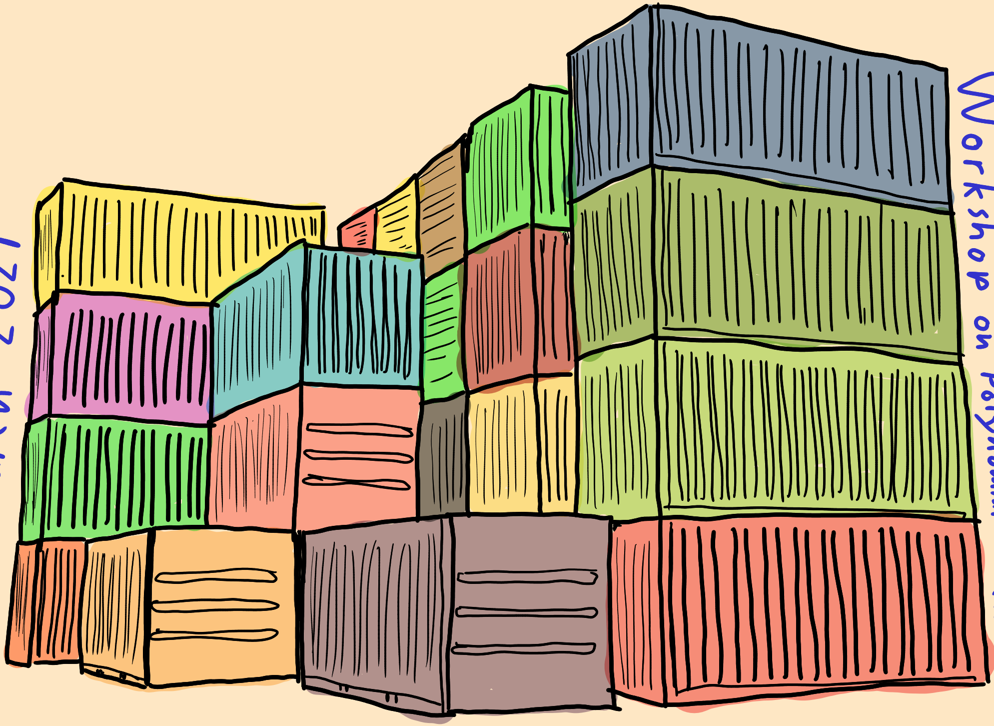
Quantitative polynomial functors

Fredrik Nordvall Forsberg, University of Strathclyde

19th March 2021



Workshop on Polynomial Functors

joint work in progress with Georgi Nakov

Linear logic

In "ordinary" logic:

$$x : A, f : A \to B \vdash (x, f(x)) : A \times B$$

In the "real" world:

$$x : \mathrm{Apple}, \mathrm{oven} : \mathrm{Apple} \multimap \mathrm{Pie} \nvdash \mathrm{Apple} \otimes \mathrm{Pie}$$

Captured by Linear logic [Girard 1987].

In Computer Science, useful for I/O, communication channels, memory management, ...

Dependent type theory

A foundation for constructive mathematics [Martin-Löf 1972].

A functional programming language [Martin-Löf 1982].

Key point: type system expressive enough to encode logical propositions, e.g.

$$\mathsf{head} : \mathsf{List}\ A\ (n+1) \longrightarrow A$$

$$\mathsf{sort} : \mathsf{List}\ A\ n \longrightarrow \mathsf{SortedList}\ A\ n$$

Good for functional correctness, but what about resource safety etc?

Combining linear and dependent types

A difficult problem!

Is

$$\mathsf{refl} : (x : A) \to x =_A x$$

linear?

Is

$$\mathsf{divide} : (n : \mathbb{N}) \to (m : \mathbb{N}) \to m > 0 \to \mathbb{N}$$

linear?

Is

$$\mathsf{sort} : (xs : \mathsf{List}\ A\ n) \to \left(\left(xs' : \mathsf{SortedList}\ A\ n\right) \times \left(\mathsf{Permutation}\ xs\ (U(xs'))\right)\right)$$

linear?

Linear dependent types

Early attempts generalising linear/non-linear logic [Krishnaswami et al 2015, Vákár 2015], splitting context into a linear and a non-linear region:

$$\underbrace{\Gamma}_{\text{non-linear}} ; \underbrace{\Delta}_{\text{linear}} \vdash a : A$$

Importantly, types may only depend on non-linear terms, i.e. are only formed when $\Delta = \emptyset$.

$\Rightarrow$ Question how to count occurrences in types goes away. ✓

$\Rightarrow$ But, cannot prove any properties of linear terms. ?

Quantitative Type Theory

Core idea: It is still possible to contemplate consumed things. [McBride 2016]

Rather than erasing things from the context, we record their usage, e.g. $R = \mathbb{N}$, $R = \{0, 1, \omega\}$

$y \overset{0}{:} \text{Poison},\ x \overset{2}{:} \text{Apple},\ \text{oven} \overset{1}{:} \text{Apple} \multimap \text{Pie} \vdash (x, \text{oven}(x)) : \text{Apple} \otimes \text{Pie}$

In general: annotations from a rig/semiring $(R, +, *, 0, 1)$.

$$\frac{\Gamma \vdash f : (x \overset{\rho}{:} A) \to B[x] \qquad \Gamma' \vdash a : A}{\Gamma + \rho\Gamma' \vdash f\,a : B[a]}$$

Importantly: forming types does not consume resources, i.e. it happens in contexts of the form $0 * \Gamma$.

That is: occurances in types are free, and we can still prove properties of linear things.

Categorical semantics [Atkey 2018]

Quantitative extension of Categories with families [Dybjer 1995].

$\langle Ty, Tm \rangle : \mathbb{C}^{op} \to Fam(Set)$

$\mathbb{C}$ category — contexts and substitutions

$Ty : \mathbb{C}^{op} \to Set$ — types and subst. actions

$Tm : (\Gamma \in \mathbb{C}^{op}) \to Ty(\Gamma) \to Set$ — terms and — " —

$\_ \cdot \_ : (\Gamma \in \mathbb{C}) \to Ty(\Gamma) \to \mathbb{C}$ — context extension (with universal property)

In addition:

$\mathbb{L}$ category — resourced contexts and subs

$U : \mathbb{L} \to \mathbb{C}$ — underlying context

$\rho(-) : \mathbb{L} \to \mathbb{L}$ for each $\rho \in R$ — scaling

$(+) : \mathbb{L} \times_U \mathbb{L} \to \mathbb{L}$ — context addition

$RTm : (\Gamma \in \mathbb{L}^{op}) \to Ty(U\Gamma) \to Set$ — resourced terms (over Tm)

$\_ \cdot^{\rho} \_ : (\Gamma \in \mathbb{L}) \to Ty(U\Gamma) \to \mathbb{L}$ for each $\rho \in R$

resourced context extension (over $\_ \cdot \_$)

Thm (Atkey) Sound and complete semantics for QTT

Concrete models

SK(I) λ-calc

BCI linear λ-calc

1. Take $\mathbb{C}$ any CwF, $\mathbb{L} = \mathbb{C}$, $U = id$

2. Fix an R-linear combinatory algebra $(\underbrace{\mathcal{A}}_{\text{"programs"}}, \underbrace{(\cdot)}_{\text{"application" } (\cdot): \mathcal{A}\times\mathcal{A}\to\mathcal{A}}, \underbrace{!_p}_{\text{"duplication" } !_p: \mathcal{A}\to\mathcal{A}}, \underbrace{B, C, I, K, W, D, \delta, F}_{\text{combinators, e.g. } B\cdot x\cdot y\cdot z = x\cdot(y\cdot z),\ K\cdot x\cdot !_0 y = x})$.

An assembly $X = (|X|, \Vdash_X)$ is a set $|X|$ and a relation $\Vdash_X \subseteq \mathcal{A} \times |X|$.

A morphism $X \to Y$ is a function $f : |X| \to |Y|$ s.t. there exists $a_f \in \mathcal{A}$ realising f: if $a \Vdash_X x$ then $a_f \cdot a \Vdash_Y f(x)$.

graphs of linear functions on $\mathbb{N}$

Take $\mathbb{L} = Asm(\mathcal{A})$, $\mathbb{C} = Set$, $U = |-|$. Concretely can take $\mathcal{A} = P(\omega)_{lin}$ [Hoshino 2007].

3. Relational realisability models: $\mathbb{L} = ReflGraph(Asm(\mathcal{A}))$, $\mathbb{C} = ReflGraph(Set)$

Some type formers

$$!_\rho A = (x \overset{\rho}{:} A) \otimes I$$

- $(x \overset{\rho}{:} A) \to P[x]$ — dep. functions using argument ρ times
- $(x \overset{\rho}{:} A) \otimes P[x]$ — dep. pairs with ρ copies of first component
- I — monoidal unit (type)
  $[\![I]\!] = (\{*\}, x \Vdash * \Leftrightarrow x = \{I\})$
- $\top$ — terminal type
  $[\![\top]\!] = (\{*\}, x \Vdash * \Leftrightarrow \text{True})$
- $A \oplus B$ — additive disjunction (coproduct)
- $A \mathbin{\&} B$ — additive conjunction ("pick one — your choice")

Data types?

How to add the trees, lists, natural numbers etc that we all know and love?

If done ad-hoc, how do we know elimination principle is right?

Instead, let's take a principled approach and consider initial algebras of polynomial functors.

Quantitative containers

$$\underline{F(X)} = (s \overset{1}{:} S) \otimes (P(s) \multimap X)$$

Functor on cat. of ~~closed~~ types and linear functions:

over fixed context $\Delta = 0\Delta$.

Objects T s.t. $\Delta \vdash T$ type

Morphisms $(\underline{\Gamma}, f)$ s.t. $\Gamma \vdash f : T \multimap T'$

$0\Gamma = \Delta$

$id = (\underline{\Delta}, \lambda x.x)$

$(\Gamma, f) \circ (\Gamma', f') = (\underline{\Gamma + \Gamma'}, f \circ f')$

Prop $f : X \multimap Y \implies F(f) : F(X) \multimap F(Y)$

$F(\underline{\Gamma}, f) = (\underline{\Gamma}, \ldots)$

Hence we can consider category of F-algebras.

Initial F-algebras

Can construct initial algebras for finitary polynomial functors in $\mathrm{Asm}(P(\omega))$:

- Underlying set constructed using initial algebras in metatheory.

  E.g. $F(X) = I \oplus X \quad |\mu F| = \mathbb{N}$

- By induction on elements in metatheory, define realisers $r(t) \in \omega$

  E.g. $r(0) = 0 \qquad r(n+1) = \langle 1, r(n) \rangle$

  Define $x \Vdash_{\mu F} t \iff x = \{r(t)\}$.

- Check that constructors and mediating map $\mu F \to X$ are realised, again by meta-induction.

$$F(X) = (s \overset{1}{:} S) \otimes (P(s) \multimap X)$$

$$c : F(W) \to W$$

Induction

What about the elimination principle?

$$\frac{Q : W \to \mathrm{Type} \quad M : (s \overset{1}{:} S)(h \overset{0}{:} P(s) \multimap W)(ih \overset{1}{:} (y : P(s)) \to Q(h\,y)) \to Q[c(s,h)]}{\mathrm{elim}(Q, M) : (x \overset{1}{:} W) \to Q[x]}$$

Get it from initiality [Hermida & Jacobs 1998, Awodey, Gambino & Sojakova 2017]:

- Use c, M to make $(y \overset{0}{:} W) \otimes Q[y]$ into F-algebra; get $\mathrm{fold} : W \multimap (y : W) \otimes Q$.
- Compose with $\mathrm{snd} : (x : (y \overset{0}{:} W) \otimes Q) \multimap Q[\mathrm{fst}\,x]$ to get $(x : W) \multimap Q[\overbrace{\mathrm{fst}\,(\mathrm{fold}\,x)}^{\mathrm{id}\ =}]$.
- Prove $\mathrm{fst} : (y : W) \otimes Q \to W$ is F-alg morphism $\Rightarrow$ so is $\mathrm{fst} \circ \mathrm{fold} : W \to W$.
  By uniqueness $\mathrm{fst} \circ \mathrm{fold} = \mathrm{id}$ $\Rightarrow$ $(x : W) \multimap Q[x]$ ✓.

$$\begin{tikzcd}
FW \arrow[r] \arrow[d] & W \arrow[d, "\mathrm{fold}"] \arrow[dd, bend left=60, "\mathrm{id}"] \\
F(\ldots) \arrow[r] \arrow[d] \arrow[dr, phantom, "\circlearrowleft"] & (y : W) \otimes Q \arrow[d, "\mathrm{fst}"] \\
FW \arrow[r] & W
\end{tikzcd}$$

Thm (W, c) is initial iff it satisfies induction.

The lack of normality

Polynomial functors traditionally inductively generated by

$$\mathrm{Id} \mid \mathrm{Const}_A \mid (+) \mid (\times) \mid A \to -$$

Thm [Abbott et al 2005] Every such polynomial functor has a container normal form $F(X) \cong (s : S_F) \times (P_F(s) \to X)$

E.g. $F(X) = \underset{X \,\&\, X \,\&\, X}{1 + X \times X \times X} \cong (b:2) \times \left(\left(\begin{array}{l} \text{if } b \text{ then } 0 \\ \text{else } 3 \end{array}\right) \to X\right)$

This breaks down in QTT setting.

E.g. $(0 \multimap X) \cong \top \not\cong I$

$(2 \multimap X) \cong X \,\&\, X \not\cong X \otimes X$

We can do it by hand

Instead of computing the CNF and deriving its induction principle, we can formulate it directly.

Main step is to inductively compute the predicate lifting $\hat{F} : (Q : X \to Type) \to (F X \to Type)$, which encodes the I.H. for the elim. principle.

available, since we are contemplating a type

E.g. $\widehat{F \otimes G}(Q, z) = \hat{F}(Q, \mathrm{fst}\, z) \otimes \hat{G}(Q, \mathrm{snd}\, z)$

$\mathrm{step} : (y : F W) \to (ih : \hat{F}(P, y)) \to P(c\, y)$

Derivation of elim. from initiality works the same, mutatis mutandis.

Summary and outlook

Thank you!

QTT combining linear and dependent types

Initial algebras of polynomial functors as principled data types for QTT

Constructing μF for non-finitary F in concrete models ?

Linear functors, and relationship with derivatives of containers ?

External "semantic" description of quantitative polynomial functors ?

Extending permutation-preservation [Atkey and Wood 2018, Abel & Bernardy 2020] to arbitrary containers ?

References

Abbott, M., Altenkirch, T. and Ghani, N., 2003. Categories of containers. In FoSSaCS '03 (pp. 23-38). Springer.

Abbott, M., Altenkirch, T. and Ghani, N., 2005. Containers: Constructing strictly positive types. Theoretical Computer Science, 342(1), pp.3-27.

Abel, A. and Bernardy, J.-P., 2020. A unified view of modalities in type systems. Proc. ACM Program. Lang. 4, ICFP, Article 90.

Atkey, R., 2018. Syntax and semantics of quantitative type theory. In LICS'18 (pp. 56-65).

Atkey, R. and Wood, J., 2018. Context constrained computation. In TyDe'18.

Awodey, S., Gambino, N. and Sojakova, K., 2017. Homotopy-Initial Algebras in Type Theory. J. ACM 63, 6, Article 51.

Dybjer, P., 1995. Internal type theory. In TYPES '95 (pp. 120-134). Springer.

Hermida, C. and Jacobs, B., 1998. Structural induction and coinduction in a fibrational setting. Information and computation, 145(2), pp.107-152.

Hishino, N., 2007. Linear Realizability. In CSL '07 (pp. 420-434).

Girard, J-Y., 1987. Linear logic, Theoretical Computer Science 50:1.

Krishnaswami, N.R., Pradic, P. and Benton, N., 2015. Integrating linear and dependent types. In POPL '15.

Martin-Löf, P., 1972. An intuitionistic theory of types. Republished in Twenty-five years of constructive type theory.

Martin-Löf, P., 1982. Constructive mathematics and computer programming. In Studies in Logic and the Foundations of Mathematics (Vol. 104, pp. 153-175). Elsevier.

McBride, C., 2016. I got plenty o'nuttin'. In A List of Successes That Can Change the World (pp. 207-233). Springer.

Vákár, M., 2015. A categorical semantics for linear logical frameworks. In FoSSACS '15 (pp. 102-116). Springer.