Abstract Syntax Via Containers

Neil Ghani

Dept of Maths and Computer Science
University of Leicester

(Also Michael Abbott, Thorsten Altenkirch, Tarmo Uustalu)

An Overview of the Talk

- **Abstract Syntax:** Variable binding via initial algebra
  - A semantics in functor categories
  - Examples are nests, lambda terms and explicit substitutions
- **Containers:** A semantic approach to concrete data structures
  - Fundamental idea of data stored at locations in memory
  - Strictly positive types (lists, trees etc) are containers
- **Ambition:** Do algorithms for lists, trees etc generalise smoothly to nests, lambda terms and explicit substitutions etc?
  - Yes if these examples are containers

Abstract Syntax

- **Examples:** Nests, Lambda terms and Explicit Substitutions are canonical examples of constrained datatypes, everything, and abstract machines

$$\begin{aligned} N(X) &= X + N(X \times X) \\ L(X) &= X + L(X) \times L(X) + L(X+1) \\ E(X) &= X + E(EX) \\ &= X + \int \mathcal{C}(n, EX) \times E(n) \end{aligned}$$

- **Solutions:** Rewrite and solve in functor categories

$$\begin{aligned} N &= \mathsf{Id} + N \circ \Delta \\ L &= \mathsf{Id} + \Delta \circ L + L \circ \delta \\ E &= \mathsf{Id} + E \circ E \end{aligned}$$

where $\Delta(X) = X \times X$ and $\delta(X) = X + 1$

A Few Pesky Details

- **Key Idea:** Solve $F = \Phi(F)$ where $\Phi : [\mathcal{C}, \mathcal{C}] \to [\mathcal{C}, \mathcal{C}]$
  - If $\mathcal{C}$ has w-colimits, so has $[\mathcal{C}, \mathcal{C}]$
  - If Φ is built from $K_F, +, \times, \circ$, then Φ preserves ω-colimits
  - Size matters - use lfp categories. Others use variants of Sets
- **Theorem:** Such Φ have an initial algebra. Tarmo showed that such structures are often monadic.
- **Reflection:** This is great because we love initial algebra semantics. But
  - Are these examples concrete data structures
  - Is lfp-ness the right approach

Containers

- **Lists:** Given by a length and a mapping of cells to data

$$\texttt{List}(X) = \Sigma n{:}\mathbb{N}.\, \{0, \ldots, n-1\} \longrightarrow X$$

- **Pairs:** One shape containing two positions for data

$$X \times X = \Sigma s{:}1.\, 2 \longrightarrow X$$

- **Definition:** A container is a dependent type $s : S \vdash P_s$ `type`
  - A presentation of $T_{S,P}(X) = \Sigma s{:}S.\, P_s \longrightarrow X$
  - Formalises the idea of data stored at memory locations
  - Containers are more general than polynomial functors.

Some Results about Containers

- **Lemma 1:** Strictly positive types are containers.

$$\tau := X \mid \tau + \tau \mid \tau \times \tau \mid K_A \mid \mu X.\tau \mid \nu X.\tau \mid \tau \circ \tau$$

- **Lemma 2:** Semantics require LCCC + W-types. That is, W-types calculates shapes and positions.

- **Lemma 3:** W-types = least fixed points of containers

  – A semantic definition of W-types

- **Lemma 4:** Containers can be differentiated etc (McBride, Hancock)

Abstract Syntax Via Containers

- If $NX = X + N(X \times X)$ satisfies $N = T_{S,P}$ then

$$\begin{array}{lcl} S & = & 1 + S \\ P(0) & = & 1 \\ P(n+1) & = & P(n) + P(n) \end{array}$$

- If $LX = X + LX \times LX + L(X + 1)$ satisfies $L = T_{S,P}$ then

$$\begin{array}{lcl} S & = & 1 + S \times S + \Sigma s : S.P(s) \rightarrow \texttt{ Bool} \\ P(*) & = & 1 \\ P(s_1, s_2) & = & P(s_1) + P(s_2) \\ P(s, f) & = & \Sigma t : P(s).\texttt{if } f(t) \texttt{ then 1 else 0} \end{array}$$

- If $EX = X + E(EX)$ satisfies $E = T_{S,P}$ then

$$\begin{array}{lcl} S & = & 1 + \Sigma s : S.P(s) \rightarrow S \\ P(*) & = & 1 \\ P(s, f) & = & \Sigma p : P(s).P(fp) \end{array}$$

- **Motivation:** If so algorithms such as zippers, differentiation, resource analysis may generalise

- **Question:** A grammar of higher order datatypes?

  - Let n denote $[\mathcal{C}^n, \mathcal{C}]$ and $\overline{m} \to n = m_1 \times \cdots m_k \to n$

$$\frac{}{X_i : \overline{m} \to m_i} \qquad \frac{F : n}{K_F : \overline{m} \to n}$$

$$\frac{\phi : \overline{m}, n \to n}{\mu\phi, \nu\phi : \overline{m} \to n} \qquad \frac{\phi : \overline{m} \to n \quad \psi_i : \overline{k} \to m_i}{\phi(\psi_1, \ldots, \psi_r) : \overline{k} \to n}$$

    Can represent E, L, N and define $[\![\phi]\!]$ as a functor.

  - **Problem:** But we wanted containers, not functors!

Container Transformers

- Let G_n be the category of n-ary containers
  - Any $\phi : n \to n$ is a container transformer $[\![\phi]\!] : G_n \to G_n$
  - ϕ-preserves filtered colimits (of a certain size)
- **But:** G_n doesnt have ω-colimits
  - But G_n has enough ω-colimits - cartesian chains!
  - So $\mu\phi : n$ is a container as required
- **Future Work:** Replace size conditions, lfp-conditions with W-types. EPSRC grant for RA and PhD student to
  - Dependent containers, resource containers, generic navigation algorithms on containers ...