Constructing Strictly Positive Types

A Joint venture of the Nottingham Container Consortium and the Epigram Team

Peter Morris
w/ Thorsten Altenkirch
pwm@cs.nott.ac.uk

University of Nottingham

First up

- This talk is about dependently typed programming, specifically the datatypes we use to do it

First up

- This talk is about dependently typed programming, specifically the datatypes we use to do it
- What we have:

$$\underline{\text{data}}\ \frac{A : \star \quad n : \mathsf{Nat}}{\mathsf{Vec}\ A\ n : \star}$$

$$\underline{\text{where}}\ \frac{}{\varepsilon : \mathsf{Vec}\ A\ 0}\ ;\ \frac{a : A \quad as : \mathsf{Vec}\ A\ n}{a{::}as : \mathsf{Vec}\ A\ (1+\ n)}$$

First up

- This talk is about dependently typed programming, specifically the datatypes we use to do it
- What we have:

$$\underline{\text{data}} \ \frac{A : \star \quad n : \mathsf{Nat}}{\mathsf{Vec}\ A\ n : \star}$$

$$\underline{\text{where}} \ \frac{}{\varepsilon : \mathsf{Vec}\ A\ 0} \ ; \ \frac{a : A \quad as : \mathsf{Vec}\ A\ n}{a :: as : \mathsf{Vec}\ A\ (1+\ n)}$$

- \+ Pattern matching, Structural Recursion. . .

What else?

- We really want to add more support for the programmer
  - Reusable libraries of code to use with a range of datatypes
  - Datatype Generics (Vectors, Lists, Telescopes. . .)
  - Remove Boilerplate

What else?

- We really want to add more support for the programmer
  - Reusable libraries of code to use with a range of datatypes
  - Datatype Generics (Vectors, Lists, Telescopes. . .)
  - Remove Boilerplate
- We'd also like to be able to express the theory of our datatypes in the language
  - To better define what datatypes ARE
  - To explain the generation of $\Leftarrow$ <u>rec</u> and $\Leftarrow$ <u>case</u> gadgets
  - To build Epigram in Epigram?

data currently

Data declarations must conform to Luo's syntactic test for strict positivity

data currently

Data declarations must conform to Luo's syntactic test for strict positivity

$$\ldots \frac{a \, : \, A \quad f \, : \, B\ a \to \mathsf{W}_A\ B}{\mathsf{sup}\ a\ f \, : \, \mathsf{W}_A\ B}$$

Is OK

data currently

Data declarations must conform to Luo's syntactic test for strict positivity

$$\ldots \frac{a : A \quad f : B\ a \to \mathsf{W}_A\ B}{\mathsf{sup}\ a\ f : \mathsf{W}_A\ B}$$

Is OK

$$\ldots \frac{f : (\mathsf{X} \to \mathsf{Bool}) \to \mathsf{Bool}}{\mathsf{c}\ f : \mathsf{X}}$$

Is not (Negative)

data currently

Data declarations must conform to Luo's syntactic test for strict positivity

$$\ldots\frac{a : A \quad f : B\ a \to \mathsf{W}_A\ B}{\mathsf{sup}\ a\ f : \mathsf{W}_A\ B}$$ Is OK

$$\ldots\frac{f : (\mathsf{X} \to \mathsf{Bool}) \to \mathsf{Bool}}{\mathsf{c}\ f : \mathsf{X}}$$ Is not (Negative)

$$\ldots\frac{ts : \mathsf{List}\ (\mathsf{RoseTree}\ A)}{\mathsf{node}\ ts : \mathsf{RoseTree}\ a}$$ Is also rejected...

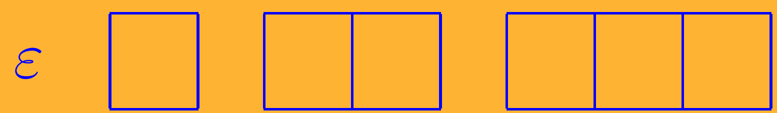
What are containers?

Containers have a set of Shapes. $S : \star \ldots$

What are containers?

Containers have a set of Shapes. $S : \star \ldots$

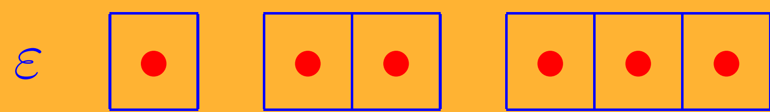
For example:
Lists have a shape very similar to the natural numbers...



What are containers?

Containers have a set of Shapes. $S : \star \ldots$
and for each shape, some set of Positions where data goes
$P : \forall s : S \Rightarrow \star$

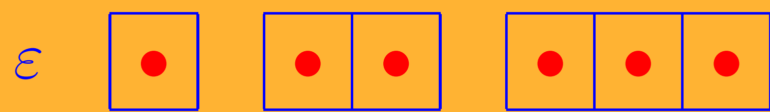
For example:
Lists have a shape very similar to the natural numbers...
and given a shape the set of positions has exactly that many elements



What are containers?

Containers have a set of Shapes. $S : \star \ldots$
and for each shape, some set of Positions where data goes
$P : \forall s : S \Rightarrow \star$

For example:
Lists have a shape very similar to the natural numbers...
and given a shape the set of positions has exactly that many elements



Closed under: $\mu, +, \times, K \rightarrow \ldots$

Indexed Containers

Are given by:

- A Type of Shapes

$$S \;:\; \star$$

$$\frac{I, O \;:\; \star}{\mathsf{IC}\; I\;\; O \;:\; \star} \;\; \underline{\text{where}} \;\; \frac{}{(\;\; |S \lhd \quad | \;\;) \;:\; \mathsf{IC}\; I\;\; O}$$

Indexed Containers

Are given by:

- A Type of Shapes
- For each shape an Output index...

$$S : \star$$
$$q : S \to O$$

$$\frac{I, O : \star}{\mathsf{IC}\ I\ O : \star} \quad \underline{\text{where}} \quad \frac{}{(q | S \lhd \ \ | \ \ ) : \mathsf{IC}\ I\ O}$$

Indexed Containers

The University of Nottingham

Are given by:

- A Type of Shapes
- For each shape an Output index...
- and a Type of Positions

$$\frac{I, O : \star}{\mathsf{IC}\ I\ O : \star} \ \underline{\text{where}} \ \frac{\begin{array}{l} S : \star \\ q : S \to O \\ P : S \to \star \end{array}}{(q|S \lhd P|\ ) : \mathsf{IC}\ I\ O}$$

Indexed Containers

The University of Nottingham

Are given by:

- A Type of Shapes
- For each shape an Output index...
- and a Type of Positions
- And for each position an Input index

$$\frac{I, O \;:\; \star}{\mathsf{IC}\; I\; O \;:\; \star} \quad \underline{\text{where}} \quad \frac{\begin{array}{l} S \;:\; \star \\ q \;:\; S \to O \\ P \;:\; S \to \star \\ r \;:\; \forall s \;:\; S \Rightarrow P\; s \to I \end{array}}{(q|S \lhd P|r) \;:\; \mathsf{IC}\; I\; O}$$

Extension

The Extension of an Indexed Container $(q|S \lhd P|r)$: $\mathsf{IC}\ I\ O$ gives rise to a functor:

$$[\![q|S \lhd P|r]\!] : (I \to \star) \to (O \to \star)$$

Extension

The Extension of an Indexed Container $(q|S\lhd P|r) \,:\, \mathsf{IC}\; I\; O$ gives rise to a functor:

$$[\![q|S\lhd P|r]\!] \,:\, (I \longrightarrow \star) \longrightarrow (O \longrightarrow \star)$$

which is given by:

$$\begin{aligned}&[\![q|S\lhd P|r]\!]\; X\; o \Rightarrow \\ &\quad \exists s \,:\, S \Rightarrow (o = q\; s) \times (\forall p \,:\, P\; s \Rightarrow X\; (r\; s\; p))\end{aligned}$$

What we have done

- We have a Universe of Indexed Containers which contains codes for all Strictly Positive Families.

What we have done

- We have a Universe of Indexed Containers which contains codes for all Strictly Positive Families.
- The codes and interpretation are both Epigram datatypes.

What we have done

- We have a Universe of Indexed Containers which contains codes for all Strictly Positive Families.
- The codes and interpretation are both Epigram datatypes.
- This gives us a semantic, compositional notion of SPFs *in Epigram*

What we have done

- We have a Universe of Indexed Containers which contains codes for all Strictly Positive Families.
- The codes and interpretation are both Epigram datatypes.
- This gives us a semantic, compositional notion of SPFs *in Epigram*
- Functions in this universe are generic programs.

What we have done

- We have a Universe of Indexed Containers which contains codes for all Strictly Positive Families.
- The codes and interpretation are both Epigram datatypes.
- This gives us a semantic, compositional notion of SPFs *in Epigram*
- Functions in this universe are generic programs.
- Conclusion? Defining what we really mean by ‘datatype’ can give us Generic programming for *all* Epigram datatypes

SPF

$$\underline{\text{data}}\ \frac{\vec{I} : \mathsf{Vec}\ \star\ n \quad O : \star}{\mathsf{SPF}\ \vec{I}\ O : \star}\ \underline{\text{where}}$$

$$\frac{}{\text{‘Z’} : \mathsf{SPF}\ (\vec{I} :: O)\ O} \qquad \frac{T : \mathsf{SPF}\ \vec{I}\ O}{\text{‘wk’}\ T : \mathsf{SPF}\ (\vec{I} :: I)\ O}$$

$$\frac{f : \forall t : \mathsf{Fin}\ n \Rightarrow \mathsf{SPF}\ \vec{I}\ O}{\text{‘Tag’}\ f : \mathsf{SPF}\ \vec{I}\ (O \times \mathsf{Fin}\ n)} \qquad \frac{}{\text{‘0’}, \text{‘1’} : \mathsf{SPF}\ \vec{I}\ O}$$

$$\frac{f : O \to O' \quad T : \mathsf{SPF}\ \vec{I}\ O}{\text{‘}\Sigma\text{’}O\ f\ T : \mathsf{SPF}\ \vec{I}\ O'} \qquad \frac{f : O' \to O \quad T : \mathsf{SPF}\ \vec{I}\ O}{\text{‘}\Delta\text{’}O\ f\ T : \mathsf{SPF}\ \vec{I}\ O'}$$

$$\frac{f : O \to O' \quad T : \mathsf{SPF}\ \vec{I}\ O}{\text{‘}\Pi\text{’}O\ f\ T : \mathsf{SPF}\ \vec{I}\ O'} \qquad \frac{T : \mathsf{SPF}\ (\vec{I} :: O)\ O}{\text{‘}\mu\text{’}\ T : \mathsf{SPF}\ \vec{I}\ O}$$

El

$$\underline{\text{data}} \quad \frac{T : \mathsf{F}\,\vec{I}\,O \quad \vec{T} : \mathsf{Tel}\,\vec{I} \quad o : O}{[\![T]\!]\,\vec{T}\,o : \star} \quad \underline{\text{where}}$$

$$\frac{v : [\![T]\!]\,\vec{T}\,\vec{X}\,o}{\mathsf{top}\,v : [\![\text{‘Z’}]\!]\,(\vec{T} :: T)\,o} \qquad \frac{v : [\![T]\!]\,\vec{T}\,\vec{X}\,o}{\mathsf{pop}\,v : [\![\text{‘wk’}\,T]\!]\,(\vec{T} :: T)\,o}$$

$$\frac{v : [\![f\ t]\!]\,\vec{T}\,o}{\mathsf{tag}\,v : [\![\text{‘Tag’}f]\!]\,\vec{T}\,(o; t)} \qquad \frac{}{\mathsf{void} : [\![\text{‘1’}]\!]\,\vec{T}\,o}$$

$$\frac{v : [\![T]\!]\,\vec{T}\,o}{\sigma_o\,v : [\![\text{‘}\Sigma\text{’}f\ T]\!]\,\vec{T}\,(f\ o)} \qquad \frac{v : [\![T]\!]\,\vec{T}\,(f\ o)}{\delta\,v : [\![\text{‘}\Delta\text{’}f\ T]\!]\,\vec{T}\,o}$$

$$\frac{\vec{v} : \forall o : O;\ p : (f\ o) = o'[\![T]\!]\,\vec{T}\,o}{\pi\,\vec{v} : [\![\text{‘}\Pi\text{’}f\ T]\!]\,\vec{T}\,o'} \qquad \frac{v : [\![T]\!]\,(\vec{T} :: (\text{‘}\mu\text{’}\ T))\,o}{\mathsf{in}\,v : [\![\text{‘}\mu\text{’}\ T]\!]\,\vec{T}\,o}$$