The University of Nottingham

Containing Families

An introduction to Containers

Peter Morris

w/ Thorsten Altenkirch

pwm@cs.nott.ac.uk

University of Nottingham

A Problem. . .

The University of Nottingham

- Find a theory of datatypes that supports reusable libraries of code
  - Boilerplate
  - Generics — Haskell deriving

A Problem. . .

The University of Nottingham

- Find a theory of datatypes that supports reusable libraries of code
  - Boilerplate
  - Generics — Haskell deriving

- Some types might support
  - Equality
  - Traversability

A Problem. . .

The University of Nottingham

- Find a theory of datatypes that supports reusable libraries of code
  - Boilerplate
  - Generics — Haskell deriving
- Some types might support
  - Equality
  - Traversability
- Should be intuitive to use

An opportunity. . .

The University of Nottingham

- Epigram a new dependently typed functional programming language

An opportunity. . .

The University of Nottingham

- Epigram a new dependently typed functional programming language
- It needs to have reusable (generic) libraries
  - Map for Lists
  - Map for Vectors
  - Map for Telescopes

An opportunity. . .

The University of Nottingham

- Epigram a new dependently typed functional programming language
- It needs to have reusable (generic) libraries
  - Map for Lists
  - Map for Vectors
  - Map for Telescopes
- Its notion of datatype is up for grabs

An opportunity. . .

The University of Nottingham

- Epigram a new dependently typed functional programming language
- It needs to have reusable (generic) libraries
  - Map for Lists
  - Map for Vectors
  - Map for Telescopes
- Its notion of datatype is up for grabs
- Vectors:

$$\underline{\text{data}}\ \frac{n : \mathsf{Nat}\ ;\ A : \star}{\mathsf{Vec}\ n\ A : \star}\ \underline{\text{where}}\ \frac{}{\mathsf{nil} : \mathsf{Vec}\ 0\ A}\ ;\ \frac{a : A\ ;\ as : \mathsf{Vec}\ n\ A}{\mathsf{cons}\ a\ as : \mathsf{Vec}\ (1{+}\ n)\ A}$$

Positivity (Luo)

The University of Nottingham

- In the current version we use Luo's syntactic definition of strictly positive inductive families. This doesn't allow us to have:

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

Positivity (Luo)

The University of Nottingham

- In the current version we use Luo's syntactic definition of strictly positive inductive families. This doesn't allow us to have:

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

- Why would we want this anyway?

Positivity (Luo)

The University of Nottingham

- In the current version we use Luo's syntactic definition of strictly positive inductive families. This doesn't allow us to have:

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

- Why would we want this anyway?
- The type we are defining can only appear to the right of arrows:

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

But hang on. . .

- What about definitions like the following:

$$\frac{ts : \mathsf{List}\ (\mathsf{RoseTree}\ A)}{\mathsf{node}\ ts : \mathsf{RoseTree}\ A}$$

But hang on. . .

- What about definitions like the following:

$$\frac{ts : \mathsf{List}\ (\mathsf{RoseTree}\ A)}{\mathsf{node}\ ts : \mathsf{RoseTree}\ A}$$

- This is forbidden; the syntactic test can't know that List preserves positivity

But hang on. . .

The University of Nottingham

- What about definitions like the following:

$$\frac{ts : \mathsf{List}\ (\mathsf{RoseTree}\ A)}{\mathsf{node}\ ts : \mathsf{RoseTree}\ A}$$

- This is forbidden; the syntactic test can't know that List preserves positivity

- We can have an encoding of Rose Trees, but they won't be the ones we want!
  - The definition above is the most concise and intuitive
  - We want to use the real List functions to map across sub-nodes

What are containers?

The University of Nottingham

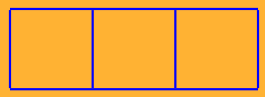
Containers have a set of Shapes. $S : \star \ldots$

What are containers?

The University of Nottingham

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

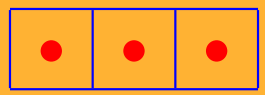
For example:
Lists have a shape very similar to the natural numbers. . .



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 (say 3) the set of positions has exactly that many elements

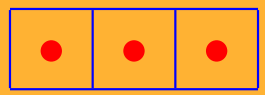


What are containers?

The University of Nottingham

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 (say 3) the set of positions has exactly that many elements



Trees: The shape of Binary trees with data at the leaves is a Binary Tree with no room for data, Positions are then paths to leaves...

More Containers

The University of Nottingham

- Closed under:

$$\mu, +, \times, K \rightarrow$$

  ... Strictly Positive Types

More Containers

The University of Nottingham

- Closed under:

$$\mu, +, \times, K \rightarrow$$

  ... Strictly Positive Types

- Elements of Container types — pick a shape, and for all the positions give a piece of payload

More Containers

The University of Nottingham

- Closed under:

$$\mu, +, \times, K \rightarrow$$

  ... Strictly Positive Types

- Elements of Container types — pick a shape, and for all the positions give a piece of payload

- Or define codes for types ($\mu, +, \times \ldots$) and for each code a type of elements
  - Generic Equality, Map, Differentiation...
  - See previous BCTCS talk

Indexed Containers

- For each shape, an input sort from $I : \star$

Indexed Containers

- For each shape, an input sort from $I : \star$
- For each position an output sort from $O : \forall s : S \Rightarrow \star$

Indexed Containers

- For each shape, an input sort from $I : \star$
- For each position an output sort from $O : \forall s : S \Rightarrow \star$
- For example, Vectors are indexed by the natural numbers, we take lists and copy the shape across to the input sort

Indexed Containers

The University of Nottingham

- For each shape, an input sort from $I : \star$
- For each position an output sort from $O : \forall s : S \Rightarrow \star$
- For example, Vectors are indexed by the natural numbers, we take lists and copy the shape across to the input sort

$$\frac{I, O : \star}{\mathsf{IC}\ I\ O : \star}\ \underline{\text{where}}\ \frac{\begin{array}{c} S : O \to \star \\ P : \forall o : O; s : S\ o \Rightarrow I \to \star \end{array}}{S \triangleleft P : \mathsf{IC}\ I\ O}$$

Roses again

The University of Nottingham

- We can compose containers
  Given a container, with input index type J and output O, and a sub-container, with inputs I and outputs J, their composition is indexed by I and O.

Roses again

The University of Nottingham

- We can compose containers
  Given a container, with input index type J and output O, and a sub-container, with inputs I and outputs J, their composition is indexed by I and O.

- Since List is a container, and ICs are closed under fixpoints, we can have the previous definition of RoseTree.

Roses again

- We can compose containers
  Given a container, with input index type J and output O, and a sub-container, with inputs I and outputs J, their composition is indexed by I and O.

- Since List is a container, and ICs are closed under fixpoints, we can have the previous definition of RoseTree.

- More generally, we have a compositonal *semantic* notion of strict positivity for datatypes.

ICs and Generics

The University of Nottingham

- We define an Epigram datatype which gives a syntax for ICs:

$$\mu, \Pi, \Sigma, \Delta, K \rightarrow \ldots$$

ICs and Generics

The University of Nottingham

- We define an Epigram datatype which gives a syntax for ICs:

$$\mu, \Pi, \Sigma, \Delta, K \rightarrow \ldots$$

- . . . along with an interpretation, that given a code builds a type that is isomorphic to the type the code represents. This is also an Epigram datatype.

ICs and Generics

The University of Nottingham

- We define an Epigram datatype which gives a syntax for ICs:

$$\mu, \Pi, \Sigma, \Delta, K \rightarrow \ldots$$

- . . . along with an interpretation, that given a code builds a type that is isomorphic to the type the code represents. This is also an Epigram datatype.

- If we write programs parametrised by these codes and interpretations we have generic programs.
  - Generic Equality, Map, Modalities?, Differentiation
  - Composition + generics = reusable libraries for Epigram

SPF

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

$$\underline{\text{where}}\ \frac{}{\text{`Z'} : \mathsf{SPF}\ (\vec{I} :: O)\ O} \quad \frac{T : \mathsf{SPF}\ \vec{I}\ O}{\text{`wk'}\ T : \mathsf{SPF}\ (\vec{I} :: I)\ O}$$

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

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

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

Epigram eats itself

The University of Nottingham

- Our syntax can represent all strictly positive inductive families.

Epigram eats itself

The University of Nottingham

- Our syntax can represent all strictly positive inductive families.
- It is *itself* a strictly positive inductive family.

Epigram eats itself

The University of Nottingham

- Our syntax can represent all strictly positive inductive families.
- It is *itself* a strictly positive inductive family.
- We can give the code for the code, modulo universe sizes. Epigram in Epigram?

Epigram eats itself

The University of Nottingham

- Our syntax can represent all strictly positive inductive families.
- It is *itself* a strictly positive inductive family.
- We can give the code for the code, modulo universe sizes. Epigram in Epigram?
- Strong theory of containers (Abbott, Altenkirch, Ghani, Hancock, McBride) can be used to justify these constructs.

Future directions

The University of Nottingham

- SPFs are a large universe of types
  - The larger the universe, the less structure, the fewer generic programs it can support
  - Our equality already uses Regular Families (ω-continuous functors) because we need to know there are only finitely many positions
  - Traversability is one thing that active container research is focusing on

Future directions

The University of Nottingham

- SPFs are a large universe of types
  - The larger the universe, the less structure, the fewer generic programs it can support
  - Our equality already uses Regular Families (ω-continuous functors) because we need to know there are only finitely many positions
  - Traversability is one thing that active container research is focusing on

- Work in progress
  - Connection to high level syntax
  - How to make writing generic programs as easy as possible within the language
  - Actually implement this stuff in Epigram2

The End

The University of Nottingham

- More information:
  - Containers: http://sneezy.cs.nott.ac.uk/containers
  - Epigram: http://www.e-pig.org

- Thanks for your attention!