Containing Families

An introduction to Containers

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

University of Nottingham
A Problem...

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

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

- Some types might support
  - Equality
  - Traversability
A Problem. . .

- 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...

- Epigram a new dependently typed functional programming language
An opportunity...

- **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...

- 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...

- **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:

\[
\begin{align*}
\text{data} & \quad \frac{n : \text{Nat}}{\text{Vec} \, n \, A} \quad \frac{A : \star}{\star} \\
\text{where} & \quad \frac{\text{nil} : \text{Vec} \, 0 \, A}{\star} \\
& \quad \frac{\text{cons} \, a \, as : \text{Vec} \, (1 + n) \, A}{\star}
\end{align*}
\]
In the current version we use Luo’s syntactic definition of strictly positive inductive families. This doesn’t allow us to have:

\[ f : (X \rightarrow \text{Bool}) \rightarrow \text{Bool} \]

\[ c\ f : X \]
Positivity (Luo)

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

\[
f : (X \rightarrow \text{Bool}) \rightarrow \text{Bool}
\]

\[
c f : X
\]

- Why would we want this anyway?
In the current version we use Luo’s syntactic definition of strictly positive inductive families. This doesn’t allow us to have:

\[ f : (X \to \text{Bool}) \to \text{Bool} \]

\[ \text{c} \ f : X \]

Why would we want this anyway?

The type we are defining can only appear to the right of arrows:

\[ f : \text{Bool} \to X \]

\[ \text{c} \ f : X \]
But hang on...

- What about definitions like the following:

\[
\begin{align*}
    ts & : \text{List} \ (\text{RoseTree} \ A) \\
    \text{node} \ ts & : \text{RoseTree} \ A
\end{align*}
\]
But hang on...

- What about definitions like the following:

\[
\text{node } ts : \text{ RoseTree } A
\]

\[
\text{List } (\text{RoseTree } A)
\]

- This is forbidden; the syntactic test can't know that \text{List} preserves positivity
But hang on... 

• What about definitions like the following:

\[
ts : \text{List (RoseTree } A) \\
\text{node } ts : \text{RoseTree } A
\]

• This is forbidden; the syntactic test can’t know that \text{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 \text{List} functions to map across sub-nodes
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\ldots and given a shape (say 3) 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\ldots 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\ldots
More Containers

- Closed under:

\[ \mu, +, \times, K \rightarrow \]

\[ \ldots \text{Strictly Positive Types} \]
More Containers

• 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

- Closed under: 

\[ \mu, +, \times, K \rightarrow \]

\[ \ldots \text{ Strictly Positive Types} \]

- 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\ldots
  - 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

- 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

$$
\frac{I, O : \star}{IC I O : \star} \quad \text{where} \quad \frac{S : O \rightarrow \star}{P : \forall o : O ; s : S o \Rightarrow I \rightarrow \star} \quad \frac{S \triangleleft P : IC I O}{S \triangleleft P : IC I O}
$$
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 $O$. 
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.
• 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

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

\[ \mu, \Pi, \Sigma, \Delta, K \rightarrow \ldots \]
ICs and Generics

- 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

- 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
data \( \vec{I} : \text{Vec} \star n O : \star \)

\[
\begin{align*}
\text{SPF} \quad \vec{I} O : \star
\end{align*}
\]

where

\[
\begin{align*}
\text{‘Z’ : SPF (} \vec{I} : O \text{) } O
\end{align*}
\]

\[
\begin{align*}
\text{‘wk’ } T : \text{SPF (} \vec{I} : I \text{) } O
\end{align*}
\]

\[
\begin{align*}
f : \forall t : \text{Fin } n \Rightarrow \text{SPF } \vec{I} O
\end{align*}
\]

\[
\begin{align*}
\text{‘Tag’ } f : \text{SPF } \vec{I} (O \times \text{Fin } n)
\end{align*}
\]

\[
\begin{align*}
0, 1 : \text{SPF } \vec{I} O
\end{align*}
\]

\[
\begin{align*}
f : O \rightarrow O' T : \text{SPF } \vec{I} O
\end{align*}
\]

\[
\begin{align*}
\text{‘Σ’ } O f T : \text{SPF } \vec{I} O'
\end{align*}
\]

\[
\begin{align*}
\text{‘Δ’ } O f T : \text{SPF } \vec{I} O'
\end{align*}
\]

\[
\begin{align*}
f : O \rightarrow O' T : \text{SPF } \vec{I} O
\end{align*}
\]

\[
\begin{align*}
\text{‘Π’ } O f T : \text{SPF } \vec{I} O'
\end{align*}
\]

\[
\begin{align*}
\mu T : \text{SPF (} \vec{I} : O \text{) } O
\end{align*}
\]
Epigram eats itself

- Our syntax can represent all strictly positive inductive families.
Epigram eats itself

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

- 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

- 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

- 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 ($\omega$-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

- 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 ($\omega$-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

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

- Thanks for your attention!