Containers

Constructing Strictly Positive Types

Felix Rech
Advisor: Steven Schäfer

June 30, 2016

Topic

Construction of types

Nested inductive and coinductive types

Based on simple primitives

Type Equivalence

Definition (Equivalence)

Two types A and B are equivalent ($A \cong B$) iff there is an isomorphism from A to B.

Axiom (Univalence)

Equivalence is equivalent to equality between two types

Proposition

$$Univalence \to Funext$$

Inductive Types

$$F\ X :\equiv 1 + X$$

$\mathbb{N}$ is a fixed point of F:

$$\mathbb{N} \cong 1 + \mathbb{N}$$

$\mathbb{N}$ is the least fixed point of F:

$$\mathbb{N} \cong \mu 1 + X$$

Least fixed points are exactly the inductive types.

Inductive Types

What about $F\ X :\equiv X \to 0$?
Assume $X \cong (X \to 0)$

- If X is empty, then $X \to 0$ is inhabited.
- If X is inhabited, then $X \to 0$ is empty.

$\Rightarrow$ F has no least fixed point.

Solution: restriction to strictly positive types

Strictly Positive Type Expressions

$$e_I ::= x \mid k \mid e_I + e'_I \mid e_I \times e'_I \mid k \to e_I \mid \mu e_{\text{Option } I} \mid \nu e_{\text{Option } I}$$
$$x : I, k : \text{Type}$$

- Parametrized by the type of free variables I
- They serve as a specification for types, that depends on an environment $\Gamma : I \to \text{Type}$.
- Types for x, k, $e_I + e'_I$, $e_I \times e'_I$ and $k \to e_I$ are specified explicitly.
- $\mu e_{\text{Option } I}$ and $\nu e_{\text{Option } I}$ require more work.

μ-Expressions

Idea: $\mu e_{\mathrm{Option}\ I}$ describes the least fixed point of $e_{\mathrm{Option}\ I}$
Problem: $e_{\mathrm{Option}\ I}$ might contain more than one free variable.

More precisely:

- Fix an expression $e_{\mathrm{Option}\ I}$ and an environment $\Gamma : I \to \mathrm{Type}$.
- Assume $F : \mathrm{Type} \to \mathrm{Type}$ is a function such that $F\ Y$ corresponds to $e_{\mathrm{Option}\ I}$ in the environment $\Gamma; ; Y$ for all Y.
- If X is the least fixed point of F, then X corresponds to $\mu e_{\mathrm{Option}\ I}$.

The specification for greatest fixed points works in the same way.

Containers

Definition (Unary Container)

A unary container consists of:

- A type of shapes S (*constructors*)
- A function $P : S \to \mathrm{Type}$
  Assigns a type of positions to every shape (*arities*)

Notation: $S \blacktriangleright P$

Definition (Unary Container Function)

$$(\!|\cdot|\!) : \mathrm{UContainer} \to (Type \to Type)$$
$$(\!|S \blacktriangleright P|\!)\ X :\equiv \sum_{s:S} P\ s \to X$$

Example

$$\mathrm{List}\ X \cong (\!|\mathbb{N} \blacktriangleright (\lambda n \Rightarrow \mathrm{Fin}\ \ n)|\!)\ X$$

Containers

Definition (Container)

A container for an index type I consists of:

- A type of shapes S (*constructors*)
- A function $P : I \to S \to \mathrm{Type}$
  Assigns a type of positions to every shape and index (*arities*)

Notation: $S \triangleright P$

Definition (Container Function)

$$\llbracket \cdot \rrbracket : \mathrm{Container}\, I \to ((I \to Type) \to Type)$$

$$\llbracket S \triangleright P \rrbracket\ \Gamma :\equiv \sum_{s:S} \prod_{i:I} P\ i\ s \to \Gamma\ i$$

Main Result

Theorem

Every strictly positive expression corresponds to a container.

We define this container by recursion.

Product Container

- By recursion we have containers $S_1 \rhd P_1$ and $S_2 \rhd P_2$ corresponding to e_I and e'_I.
- We need a container c with $[\![c]\!]\ \Gamma \cong [\![S_1 \rhd P_1]\!]\ \Gamma \times [\![S_2 \rhd P_2]\!]\ \Gamma$ for all environments Γ.

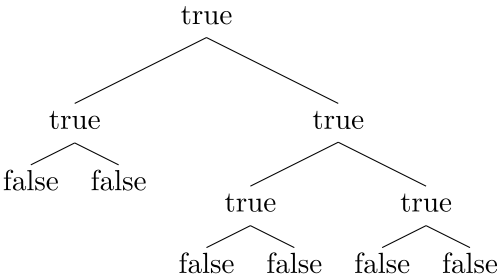
$$
\begin{aligned}
&[\![S_1 \rhd P_1]\!]\ \Gamma \times [\![S_2 \rhd P_2]\!]\ \Gamma \\
&\qquad \equiv (\sum_{s:S_1} \prod_i P_1\ i\ s \to \Gamma\ i) \times (\sum_{s:S_2} \prod_i P_2\ i\ s \to \Gamma\ i) \\
&\qquad \cong \sum_{(s_1,s_2):S_1 \times S_2} (\prod_i P_1\ i\ s_1 \to \Gamma\ i) \times (\prod_i P_2\ i\ s_2 \to \Gamma\ i) \\
&\qquad \cong \sum_{(s_1,s_2):S_1 \times S_2} \prod_i (P_1\ i\ s_1 \to \Gamma\ i) \times (P_2\ i\ s_2 \to \Gamma\ i) \\
&\qquad \cong \sum_{(s_1,s_2):S_1 \times S_2} \prod_i (P_1\ i\ s_1 + P_2\ i\ s_2) \to \Gamma\ i \\
&\qquad \equiv [\![S_1 \times S_2 \rhd \lambda\ i\ s \Rightarrow P_1\ i\ s_1 + P_2\ i\ s_2]\!]\ \Gamma
\end{aligned}
$$

W-Types (Well-Founded Trees)

```
Inductive W A (B : A -> Type) :=
sup (label : A) (subtrees : B label -> W A B) : W A B.
```

Example

$$BTree \cong W\ \mathrm{Bool}\ (\lambda b \Rightarrow \mathit{if}\ b\ \mathit{then}\ \mathrm{Bool}\ \mathit{else}\ 0)$$



Lemma

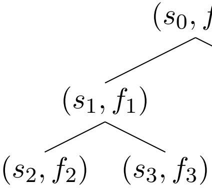
$W\ A\ B$ the least fixed point for $\llparenthesis A \blacktriangleright B \rrparenthesis$.

μ-Containers

By recursion we have a container $S \rhd P$ corresponding to $e_{\mathrm{Option}\,I}$.
For every environment Γ we need a least fixed point for

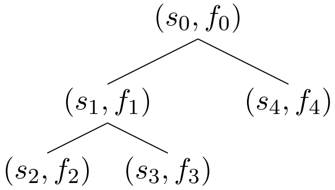
$$\begin{aligned}
&\lambda X \Rightarrow [\![S \rhd P]\!]\ (\Gamma; ; X) \\
&\quad = (\!|\sum_{s:S} \prod_{i:I} P\ (\mathrm{some}\ \ i)\ s \to \Gamma\ i \blacktriangleright P\ \ \mathrm{none} \circ \mathrm{fst} |\!)
\end{aligned}$$

Representation as W-type:



μ-Containers

$$W \left(\sum_{s:S} \prod_{i:I} P\ (\mathrm{some}\ \ i)\ s \to \Gamma\ i \right)\ (P\ \ \mathrm{none} \circ \mathrm{fst})$$



Shapes:

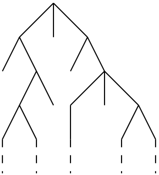
$$S_\mu :\equiv W\ S\ (P\ \ \mathrm{none})$$

Positions:

$$P_\mu\ i\ (\mathrm{sup}\ \ r\ s) :\equiv P\ (\mathrm{some}\ \ i)\ r + \sum_{p:P\ \ \mathrm{none}\ \ r} P_\mu\ i\ (s\ p)$$

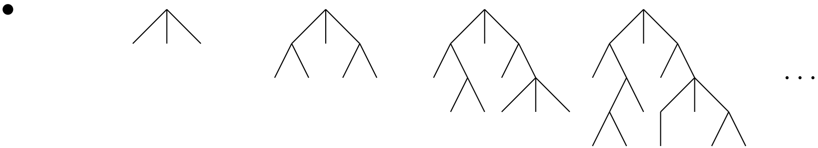
M-Types (Non-Wellfounded Trees)

We want the greatest fixed point of $F :\equiv (\!| A \blacktriangleright B |\!)$.



Representation as a sequence of finite trees:

$$1 \xleftarrow{\pi_0} F\,1 \xleftarrow{\pi_1} F^2\,1 \xleftarrow{\pi_2} F^3\,1 \xleftarrow{\pi_3} F^4\,1 \dashleftarrow \cdots$$



$$\sum_{x:\prod_n F^n 1} \prod_{n:\mathbb{N}} \pi_n\, x_{n+1} = x_n$$

Conclusion

- We wanted to construct nested inductive an coinductive types.
- For the construction of fixed points we introduced:
  - strictly positive type expressions
  - their representation as containers
- Every strictly positive type expression corresponds to a container.
- M-types can be constructed from inductive types.

Conclusion

What we used:

- dependent functions
- dependent pairs
- sums
- equalities
- W-types

What we didn't use:

- mutual inductive definitions
- coinductive definitions

References

Michael Abbott, Thorsten Altenkirch, and Neil Ghani. "Containers: constructing strictly positive types". In: *Theoretical Computer Science* 342.1 (2005), pp. 3–27.

Benedikt Ahrens, Paolo Capriotti, and Régis Spadotti. "Non-wellfounded trees in homotopy type theory". In: *arXiv preprint arXiv:1504.02949* (2015).