

A Hitchhiker's Guide to the Universes

(Universes for Generic Programs and Proofs)

Marcin Benke

Peter Dybjer

Patrik Jansson

Chalmers Technical University

Main goals:

- extend generic programming to functional languages with dependent types
- create generic proofs of properties of generic functions

Generic functional programming

- **Basic idea:** define generic functions by induction on the definition of a data type
- **Examples:**
 - generic Boolean equality: SML (built-in) and Haskell (derivable class)
 - generic **map** combinators, and generic **iteration** and **recursion** over inductive datatypes
- **Benefits:**
 - highly reusable and adaptive definitions
 - well suited for building libraries of programs, theorems **and proofs**

Related work:

Generic programming:

- '85 Böhm & Berarducci (universal algebra)
- '91 Backhouse et al. (Squiggol),
- '93 Bird et al. (generic functional programming),
- '95 Jay (shape polymorphism),
- '97 Jansson & Jeuring (polytypic programming)
- '02 Hinze & Jeuring (Generic Haskell).

Generic programming and dependent types:

- '99 Dybjer & Setzer (generic dependent type theory: IIR)
- '99 Pfeifer & Rueß (first polytypic proof)
- '02 Altenkirch & McBride; Norell (Generic Haskell in Type Theory)

Dependent types

Examples:

- $\text{Vect } n$ — vectors (lists) of length n
- data structures with invariants: ordered lists, balanced trees, AVL-trees, red-black-trees, etc.
- In general: we can express more or less arbitrary properties of programs and data structures.
- Universe of codes for datatypes — the natural setting for generic programming

The ideas presented in this talk have been implemented and tested using the *Alfa* proof editor.

Universes

- A universe consists of
 - a set of codes for datatypes: $\text{Sig} : \text{Set}$
 - a decoding function: $\text{T} : (\Sigma : \text{Sig}) \rightarrow \text{Set}$
- **Example:** $\text{Sig} = \text{Bool}$ and $\text{T} = \text{Tr}$

$\text{Tr} : \text{Bool} \rightarrow \text{Set}$

$\text{Tr } \mathbf{False} = \text{Void}$

$\text{Tr } \mathbf{True} = \text{Unit}$

A universe for single-sorted algebras

Consider the class of term algebras \mathbb{T}_Σ for a one-sorted signature Σ .

Such a signature is a list of arities of the operations.

Examples are:

- the empty type with $\Sigma = []$,
- the booleans with $\Sigma = [0, 0]$, lists of booleans with $\Sigma = [0, 1, 1]$,
- the natural numbers with $\Sigma = [0, 1]$,
- and binary trees without information in the nodes with $\Sigma = [0, 2]$.

This universe is described by the set of signatures $\text{Sig} = [\text{Nat}]$, and the decoding function $\mathbb{T} : \text{Sig} \rightarrow \text{Set}$, which maps a signature to (the carrier of) its term algebra.

Generic programs and proofs

We define generic functions, e.g.

$$\text{size} \quad : \quad (\Sigma : \text{Sig}) \rightarrow \mathbb{T}_\Sigma \rightarrow \text{Nat}$$

$$\text{eq} \quad : \quad (\Sigma : \text{Sig}) \rightarrow \mathbb{T}_\Sigma \rightarrow \mathbb{T}_\Sigma \rightarrow \text{Bool}$$

and proofs

$$\text{reflexivity} \quad : \quad (\Sigma : \text{Sig}) \rightarrow (x : \mathbb{T}_\Sigma) \rightarrow \text{Tr}(\text{eq}_\Sigma x x)$$

$$\begin{aligned} \text{substitutivity} \quad : \quad & (\Sigma : \text{Sig}) \rightarrow (x, y : \mathbb{T}_\Sigma) \rightarrow \text{Tr}(\text{eq}_\Sigma x y) \rightarrow \\ & (P : \mathbb{T}_\Sigma \rightarrow \text{Set}) \rightarrow (P x) \rightarrow (P y) \end{aligned}$$

To do this we introduce a generic iterator and recursor for \mathbb{T}_Σ .

Functions and proofs are defined by applying the iterator (or the recursor) to a step-function which is in turn defined by induction on Σ .

More Universes

By varying the definition of Sig , we can create generic programs and proofs in various settings:

- Iterated induction: $\text{Sig} = [\text{Arity}]$ and
 $\text{Arity} = \mathbf{data\ Zero} \mid \mathbf{Rec\ Arity} \mid \mathbf{NonRec\ Arity\ Arity}$
- Parameterized algebraic types:
 $\text{Arity} = \mathbf{data\ Zero} \mid \mathbf{Rec\ Arity} \mid \mathbf{NonRec\ Arity\ Arity} \mid \mathbf{Par\ Arity}$
- ... with n parameters: $\text{Arity}(n : \text{Nat}) = \mathbf{data\ Zero} \mid \mathbf{Rec\ Arity} \mid$
 $\mathbf{NonRec\ Arity\ Arity} \mid \mathbf{Par\ (Fin\ } n) \text{ Arity}$
- Generalized induction: $\text{Sig} = [\text{Set}]$
- Generalized iterated induction: $\text{Sig} = [\text{Sig}]$
- Inductive families indexed by I :
 $\text{Sig}_I = \mathbf{data\ Nil} \mid \mathbf{NonRec\ (A : Set)(A \rightarrow Sig_I)} \mid \mathbf{Rec\ } I \text{ Sig}_I$

Formal theory

We work in versions of Martin-Löf type theory, each consisting of:

- Rules for the logical framework (as in Martin-Löf)
- Rules for the universe Sig
- Generic formation, introduction, elimination and equality rules for \mathbb{T}_Σ .

Conclusions

- Dependent types are the natural setting for generic programming.
- This setting allows generic proofs as well as generic functions.
- Our approach works for a wide range of universes, including inductive families.
- All of these have been implemented and tested using the *Alfa* proof editor.