Comparing Libraries for Generic Programming in Haskell

Patrik Jansson

Chalmers University of Technology, Sweden

Sept. 2007 (Kyoto)
Comparing Libraries for Generic Programming in Haskell

- Generic Programming
  - Definition (from Jazayeri 1998)
  
  Generic programming is a sub-discipline of computer science that deals with finding abstract representations of efficient algorithms, data structures, and other software concepts, and with their systematic organization. The goal of generic programming is to express algorithms and data structures in a broadly adaptable, interoperable form that allows their direct use in software construction.
Comparing Libraries for Generic Programming in Haskell

- Generic Programming
- Libraries — not tools
  Excluding Generic Haskell, preprocessors etc.

Patrik Jansson
CompGenLib
Comparing Libraries for Generic Programming in Haskell

- Generic Programming
- Libraries — not tools
  Excluding Generic Haskell, preprocessors etc.
- in Haskell (+ implemented extensions)
  excluding C++ STL, open datatypes, ...
Generic Programming
Libraries — not tools
Excluding Generic Haskell, preprocessors etc.
in Haskell (+ implemented extensions)
excluding C++ STL, open datatypes, . . .

Too narrow scope?
Comparing Libraries for Generic Programming in Haskell

- Generic Programming
- Libraries — not tools
  Excluding Generic Haskell, preprocessors etc.
- in Haskell (+ implemented extensions)
  excluding C++ STL, open datatypes, . . .

Too narrow scope? No! Haskell scores best of: C++, SML, Ocaml, Haskell, Eiffel, Java, C#, Cecil in "An extended comparative study of language support for generic programming" [GJL+07]
Haskell libraries for Generic Programming

2000: DTC [HPJ00] Derivable Type Classes, Hinze + Peyton Jones
Haskell libraries for Generic Programming

2000: DTC [HPJ00] Derivable Type Classes
2002: LIGD [CH02] A lightweight implementation of generics and dynamics, Cheney + Hinze
Haskell libraries for Generic Programming

2000: DTC [HPJ00] Derivable Type Classes
2003: SYB1 [LPJ03] Scrap Your Boilerplate: a practical approach to generic programming, Lämmel + Peyton Jones;
many Haskell libraries for Generic Programming

2000: DTC [HPJ00] Derivable Type Classes
2003: SYB1 [LPJ03]; PolyP [NJ04] Polytypic Programming in Haskell, Norell + Jansson
many Haskell libraries for Generic Programming

2000: DTC [HPJ00] Derivable Type Classes
2003: SYB1 [LPJ03]; PolyP [NJ04]
2004: SYB2 [LPJ04]; GM [Hin04], [Hin06]
many Haskell libraries for Generic Programming

2000: DTC [HPJ00] Derivable Type Classes
2003: SYB1 [LPJ03]; PolyP [NJ04]
2004: SYB2 [LPJ04]; GM [Hin04], [Hin06]
2005: SYB3 [LPJ05]
Too many Haskell libraries for Generic Programming!

2000: DTC [HPJ00] Derivable Type Classes
2003: SYB1 [LPJ03]; PolyP [NJ04]
2004: SYB2 [LPJ04]; GM [Hin04], [Hin06]
2005: SYB3 [LPJ05]
2006: SYB rel. [HLO06]; Compos [BR06]; RepLib [Wei06]; RecLib [RE06]; SYB rev. [HL06]; Smash [Kis]; EMGM [dSOHL07]
Too many Haskell libraries for Generic Programming!

2000: DTC [HPJ00] Derivable Type Classes
2003: SYB1 [LPJ03]; PolyP [NJ04]
2004: SYB2 [LPJ04]; GM [Hin04], [Hin06]
2005: SYB3 [LPJ05]
2006: SYB rel. [HLO06]; Compos [BR06]; RepLib [Wei06]; RecLib [RE06]; SYB rev. [HL06]; Smash [Kis]; EMGM [dSOHL07]
2007: GP NOW [HL07]; Uniplate [MR07]
Too many Haskell libraries for Generic Programming!

2000: DTC [HPJ00] Derivable Type Classes
2003: SYB1 [LPJ03]; PolyP [NJ04]
2004: SYB2 [LPJ04]; GM [Hin04], [Hin06]
2005: SYB3 [LPJ05]
2006: SYB rel. [HLO06]; Compos [BR06]; RepLib [Wei06]; RecLib [RE06]; SYB rev. [HL06]; Smash [Kis]; EMGM [dSOHL07]
2007: GP NOW [HL07]; Uniplate [MR07]
Comparison criteria

- Implementation techniques
- Expressiveness
- Conciseness / Readability
- Efficiency / Portability / Stability
### Implementation techniques

- Representation of overloaded functions
  - Generic view of datatypes
    - type reflection
      - type class
        - cast
      - LIGD
        - DTC, GM, EMGM

<table>
<thead>
<tr>
<th>sum-of-prod</th>
<th>type reflection</th>
<th>type class</th>
<th>cast</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LIGD</td>
<td>DTC, GM, EMGM</td>
<td></td>
</tr>
</tbody>
</table>
## Implementation techniques

- Representation of overloaded functions
- Generic view of datatypes

<table>
<thead>
<tr>
<th>sum-of-prod</th>
<th>type reflection</th>
<th>type class</th>
<th>cast</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LIGD</td>
<td>DTC, GM, EMGM</td>
<td></td>
</tr>
<tr>
<td>fixed point</td>
<td>SYB reload.</td>
<td>PolyP</td>
<td></td>
</tr>
</tbody>
</table>

(Table based on [HL07, page 54].)
### Implementation techniques

- Representation of overloaded functions
- Generic view of datatypes

<table>
<thead>
<tr>
<th></th>
<th>type reflection</th>
<th>type class</th>
<th>cast</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum-of-prod</td>
<td>LIGD</td>
<td>DTC, GM, EMGM</td>
<td></td>
</tr>
<tr>
<td>fixed point</td>
<td>SYB reload.</td>
<td>PolyP</td>
<td>SYB1,2</td>
</tr>
<tr>
<td>spine</td>
<td>SYB rel, rev</td>
<td>SYB3</td>
<td>SYB1,2</td>
</tr>
</tbody>
</table>

(Table based on [HL07, page 54].)
### Implementation techniques

<table>
<thead>
<tr>
<th>Representation of overloaded functions</th>
<th>Generic view of datatypes</th>
</tr>
</thead>
<tbody>
<tr>
<td>➔</td>
<td>➡</td>
</tr>
<tr>
<td>type reflection</td>
<td>type class</td>
</tr>
<tr>
<td>sum-of-prod</td>
<td>cast</td>
</tr>
<tr>
<td>fixed point</td>
<td></td>
</tr>
<tr>
<td>spine</td>
<td></td>
</tr>
<tr>
<td>([a], [a] → a)</td>
<td></td>
</tr>
<tr>
<td>LIGD</td>
<td>DTC, GM, EMGM</td>
</tr>
<tr>
<td>SYB reload.</td>
<td>PolyP</td>
</tr>
<tr>
<td>SYB rel, rev</td>
<td>SYB3</td>
</tr>
<tr>
<td></td>
<td>SYB1,2</td>
</tr>
<tr>
<td></td>
<td>Uniplate</td>
</tr>
</tbody>
</table>

- Table based on [HL07, page 54].

Patrik Jansson  
CompGenLib
### Implementation techniques

- **Representation of overloaded functions**
- **Generic view of datatypes**

<table>
<thead>
<tr>
<th>Type</th>
<th>Representation</th>
<th>Type Class</th>
<th>Cast</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum-of-prod</td>
<td>type reflection</td>
<td>DTC, GM, EMGM</td>
<td></td>
</tr>
<tr>
<td>fixed point</td>
<td></td>
<td>SYB reload.</td>
<td>PolyP</td>
</tr>
<tr>
<td>spine</td>
<td></td>
<td>SYB rel, rev</td>
<td>SYB3</td>
</tr>
<tr>
<td>([a], [a] → a)</td>
<td></td>
<td></td>
<td>SYB1,2</td>
</tr>
<tr>
<td>Type List</td>
<td>RepLib</td>
<td></td>
<td>Smash</td>
</tr>
</tbody>
</table>

(Table based on [HL07, page 54].)
A generic function is a family of functions depending on a type (representation). This dependency can be

- **Explicit**:  
  \[ \text{eq} :: \text{Rep} \ a \to a \to a \to \text{Bool}, \ 
  \text{test} = \text{eq} \ r\text{Tree} \ t1 \ t2 \]

- **Implicit**:  
  \[ \text{eq} :: \text{Rep} \ a \Rightarrow a \to a \to \text{Bool}, \ 
  \text{test} = \text{eq} \ t1 \ t2 \]

The type representation can be
- **Concrete/first order**: abstract syntax
- **Abstract/higher order**: functions
Expressiveness

- **Universe:**
  - How big is the set of types a Gen. Fun. can be used on?
  - Can different sub-universes be specified?
- **Which generic functions can be defined?**
  - **Traversal:** $\text{Rep } a \rightarrow a \rightarrow a$
  - **Evaluator / Catamorphism / Consumer:** $\text{Rep } a \rightarrow a \rightarrow T$
  - **Parser / Anamorphism / Producer:** $\text{Rep } a \rightarrow T \rightarrow a$
Conciseness / readability

- How much code is required
  - for each new datatype?
  - for each new function?
  - for each application of a generic function?

- How intuitive are the types?
  
  **LIGD:** $Rep \ a \rightarrow \ a \rightarrow \ a \rightarrow \ Bool$

  **EMGM:** $Rep\ Geq\ a \Rightarrow \ a \rightarrow \ a \rightarrow \ Bool$

  **SYB:** $(Data\ a1,\ Data\ a2) \Rightarrow \ a1 \rightarrow \ a2 \rightarrow \ Bool$
Efficient / Portable / Stable

- Is the generic function compiled to efficient code?
- Is the library portable across Haskell implementations?
- Is the library supported?

A preliminary benchmark indicates that EMGM and Uniplate are fastest, followed by RepLib, SYB rev/rel, LIGD and with the traditional SYB lacking far behind.
### EMGM: Infrastructure

```haskell
data Unit = Unit

data \(\alpha :+ : \beta = \text{Inl} \ \alpha \mid \text{Inr} \ \beta\)

data \(\alpha :*: \beta = \alpha :* : \beta\)

data Iso \(\alpha \ \beta = \text{Iso}\{ \text{from} :: \alpha \to \beta, \text{to} :: \beta \to \alpha \}\)

class Generic g where
  unit :: g Unit
  char :: g Char
  plus :: g a \to g b \to g (a :+ : b)
  prod :: g a \to g b \to g (a :* : b)
  view :: Iso b a \to g a \to g b

class Generic g \Rightarrow Rep g a where
  rep :: g a
```

Patrik Jansson

CompGenLib
newtype Geq a = Geq{geq' :: a → a → Bool}
instance Generic Geq where
  unit = Geq
    (λx y → case (x, y) of (Unit, Unit) → True)
  plus a b = Geq
    (λx y → case (x, y) of (Inl x, Inl y) → geq' a x y
    (Inr x, Inr y) → geq' b x y
    _ → False)
  prod a b = Geq
    (λ(x ::* y) (v ::* w) → geq' a x v ∧ geq' b y w)
  char = Geq (≡)
  view iso _ a = Geq
    (λx y → geq' a (from iso x) (from iso y))
class Uniplate a where
    uniplate :: a → ([a], [a] → a)
children :: Uniplate a ⇒ a → [a]
children = fst ∘ uniplate
universe :: Uniplate a ⇒ a → [a]
universe x = x : concatMap universe (children x)
transform :: Uniplate a ⇒ (a → a) → a → a
transform f x = f $ combine $ map (transform f) cs
    where (cs, combine) = uniplate x
Uniplate: Generic Equality
Uniplate: Generic Equality

Cannot be expressed.
Haskell is a good language for generic programming
Software reuse is difficult (everyone “rolls their own”)
Easy to use, portable and efficient: Uniplate, EMGM
(work in progress)
Haskell libraries for Generic Programming

- Generics for the Masses: LIGD and GM are superseeded by EMGM [dSOHL07]
- Scrap your boilerplate: SYB1 [LPJ03]; SYB2 [LPJ04]; SYB3 [LPJ05] RecLib [RE06]; SYB rel. [HLO06]; SYB rev. [HL06];
- Compos [BR06]; Uniplate [MR07]


Ronald Garcia, Jaakko Jarvi, Andrew Lumsdaine, Jeremy Siek, and Jeremiah Willcock.
An extended comparative study of language support for generic programming.

Ralf Hinze.
Generics for the masses.

Ralf Hinze.
Generics for the masses.


Ralf Hinze and Andres Löh. Generic programming, now!

Ralf Hinze, Andres Löh, and Bruno C. d. S. Oliveira.

“Scrap Your Boilerplate” reloaded.


TLDI’03.


Ralf Lämmel and Simon Peyton Jones. Scrap your boilerplate with class: extensible generic functions.


Stephanie Weirich. RepLib: a library for derivable type classes.