Generic Views

Stefan Holdermans
sholderm@students.cs.uu.nl

Utrecht University
Institute of Information and Computing Sciences

January 5, 2005

Views

- *Views: A Way for Pattern Matching to Cohabit With Data Abstaction* (Wadler, 1987)
- *Views: An Extension to Haskell Pattern Matching* (Burton et al., 1996)

Built-in Integers

$$\textbf{data}\ \text{Integer} = \ldots \mid -1 \mid 0 \mid 1 \mid \ldots$$

Peano Naturals

$$\textbf{data}\ \text{Nat} = \mathit{Zero} \mid \mathit{Succ}\ \text{Nat}$$

Views

- *Views: A Way for Pattern Matching to Cohabit With Data Abstaction* (Wadler, 1987)
- *Views: An Extension to Haskell Pattern Matching* (Burton et al., 1996)

Lists

$$\begin{aligned}\textbf{data}\ \mathsf{List}\ (\mathsf{a} :: \star) = {} & \mathit{Nil} \\ \mid {} & \mathit{Cons}\ \mathsf{a}\ (\mathsf{List}\ \mathsf{a})\end{aligned}$$

Backwards Lists

$$\begin{aligned}\textbf{data}\ \mathsf{ListB}\ (\mathsf{a} :: \star) = {} & \mathit{Lin} \\ \mid {} & \mathit{Snoc}\ (\mathsf{ListB}\ \mathsf{a})\ \mathsf{a}\end{aligned}$$

Generic Views

- Generic views generalize Wadler's views
- Data types to structural representations
- In Generic Haskell, data types are viewed as sums of products

Structure Types

$$
\begin{aligned}
&\textbf{data}\ \text{Unit} && = \mathit{Unit} \\
&\textbf{data}\ \text{Sum}\ (\text{a} :: \star)\ (\text{b} :: \star) && = \mathit{Inl}\ \text{a} \mid \mathit{Inr}\ \text{b} \\
&\textbf{data}\ \text{Prod}\ (\text{a} :: \star)\ (\text{b} :: \star) && = \text{a} \times \text{b}
\end{aligned}
$$

Generic Views

Lists

$$\begin{aligned}\textbf{data}\ \mathsf{List}\ (\mathsf{a} :: \star) = {} & \mathit{Nil} \\ \mid {} & \mathit{Cons}\ \mathsf{a}\ (\mathsf{List}\ \mathsf{a})\end{aligned}$$

Structural Representation

$$\textbf{type}\ \textbf{Str}(\mathsf{List})\ (\mathsf{a} :: \star) = \mathsf{Sum}\ \mathsf{Unit}\ (\mathsf{Prod}\ \mathsf{a}\ (\mathsf{List}\ \mathsf{a}))$$

Alternative Views

- Let's consider some alternative views
- A balanced view makes compression more efficient
- A list-like view allows for more intuitive traversal code
- A fixed-point view gives access to recursive calls in data types, increasing expressiveness

Compression

Generic Compression

$$
\begin{array}{lll}
encode\ \langle \mathsf{a} :: \star \rangle & & :: (encode\ \langle \mathsf{a} \rangle) \Rightarrow \mathsf{a} \to [\mathsf{Bit}] \\
encode\ \langle \mathsf{Unit} \rangle & Unit & = [\,] \\
encode\ \langle \mathsf{Sum}\ \alpha\ \beta \rangle & (Inl\ a) & = \mathtt{0} : encode\ \langle \alpha \rangle\ a \\
encode\ \langle \mathsf{Sum}\ \alpha\ \beta \rangle & (Inr\ b) & = \mathtt{1} : encode\ \langle \beta \rangle\ b \\
encode\ \langle \mathsf{Prod}\ \alpha\ \beta \rangle & (a \times b) & = encode\ \langle \alpha \rangle\ a +\!\!+ encode\ \langle \beta \rangle\ b
\end{array}
$$

Bits

$$\mathbf{data}\ \mathsf{Bit} = \mathtt{0} \mid \mathtt{1}$$

Emit a single bit for every choice between two constructors

Days of the Week

Days

$$\mathbf{data}\ \mathrm{Day} = \mathit{Monday} \mid \mathit{Tuesday} \mid \mathit{Wednesday} \mid \mathit{Thursday} \mid \mathit{Friday} \mid \mathit{Saturday} \mid \mathit{Sunday}$$

Structural Representation

$$\begin{aligned}\mathbf{type}\ \mathbf{Str}(\mathrm{Day}) = {} & \mathrm{Sum}\ \mathrm{Unit} \\ & (\mathrm{Sum}\ \mathrm{Unit} \\ & \quad (\mathrm{Sum}\ \mathrm{Unit} \\ & \quad\quad (\mathrm{Sum}\ \mathrm{Unit} \\ & \quad\quad\quad (\mathrm{Sum}\ \mathrm{Unit} \\ & \quad\quad\quad\quad (\mathrm{Sum}\ \mathrm{Unit}\ \mathrm{Unit})))))\end{aligned}$$

Generic Compression Is Inefficient

Generic vs. Handwritten

Monday	`0`
Tuesday	`10`
Wednesday	`110`
Thursday	`1110`
Friday	`11110`
Saturday	`111110`
Sunday	`111111`

Generic Compression Is Inefficient

Generic vs. Handwritten

	generic	handwritten
Monday	`0`	`00`
Tuesday	`10`	`010`
Wednesday	`110`	`011`
Thursday	`1110`	`100`
Friday	`11110`	`101`
Saturday	`111110`	`110`
Sunday	`111111`	`111`

Generic Compression Is Inefficient

Generic vs. Handwritten

	generic	handwritten
Monday	`0`	`00`
Tuesday	`10`	`010`
Wednesday	`110`	`011`
Thursday	`1110`	`100`
Friday	`11110`	`101`
Saturday	`111110`	`110`
Sunday	`111111`	`111`

Problem

Compared to a handwritten encoding, the generic encoding is rather inefficient

Balanced Representation

Days

$$\begin{aligned}\textbf{data}\ \text{Day} = {} & \mathit{Monday} \mid \mathit{Tuesday} \mid \mathit{Wednesday} \mid \mathit{Thursday} \\ & \mid \mathit{Friday} \mid \mathit{Saturday} \mid \mathit{Sunday}\end{aligned}$$

Structural Representation

$$\begin{aligned}\textbf{type}\ \textbf{Str}(\text{Day}) = {} & \text{Sum}\ (\text{Sum}\ \text{Unit}\ (\text{Sum Unit Unit})) \\ & (\text{Sum}\ (\text{Sum Unit Unit})\ (\text{Sum Unit Unit}))\end{aligned}$$

Missing Products

Example

$$\mathbf{data}\ \text{Example}\ (\text{a} :: \star)\ (\text{b} :: \star) = A \mid B\ \text{a} \mid C\ \text{a}\ \text{b}$$

Structural Representation

$$\mathbf{type}\ \mathbf{Str}(\text{Example})\ (\text{a} :: \star)\ (\text{b} :: \star) = \\ \text{Sum Unit}\ (\text{Sum a}\ (\text{Prod a b}))$$

Missing Products

Example

$$\mathbf{data}\ \text{Example}\ (\text{a} :: \star)\ (\text{b} :: \star) = A \mid B\ \text{a} \mid C\ \text{a}\ \text{b}$$

Structural Representation

$$\mathbf{type}\ \mathbf{Str}(\text{Example})\ (\text{a} :: \star)\ (\text{b} :: \star) = \text{Sum Unit}\ (\text{Sum a}\ (\text{Prod a b}))$$

Problem

- Sum structures are only created if there are two or more constructors
- Product structures are only created if there are two or more fields

Empty Sums

Structure Types

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Zero} \\
&\textbf{data}\ \mathsf{Unit} &&= \mathit{Unit} \\
&\textbf{data}\ \mathsf{Sum}\ (\mathsf{a} :: \star)\ (\mathsf{b} :: \star) &&= \mathit{Inl}\ \mathsf{a} \mid \mathit{Inr}\ \mathsf{b} \\
&\textbf{data}\ \mathsf{Prod}\ (\mathsf{a} :: \star)\ (\mathsf{b} :: \star) &&= \mathsf{a} \times \mathsf{b}
\end{aligned}
$$

List-like Products

Example

$$\textbf{data}\ \text{Example}\ (\text{a} :: \star)\ (\text{b} :: \star) = A \mid B\ \text{a} \mid C\ \text{a}\ \text{b}$$

Structural Representation

$$\begin{aligned}&\textbf{type}\ \textbf{Str}(\text{Example})\ (\text{a} :: \star)\ (\text{b} :: \star) = \\ &\quad \text{Sum Unit} \\ &\qquad (\text{Sum}\ (\text{Prod a Unit}) \\ &\qquad\quad (\text{Sum}\ (\text{Prod a}\ (\text{Prod b Unit}))\ \text{Zero}))\end{aligned}$$

- Zero and Unit act like Nil
- Sum and Prod act like $Cons$

Collecting Subterms

Terms

$$
\begin{aligned}
\textbf{data}\ \text{Term} &= \mathit{Var}\ \text{Variable} \\
&\mid \mathit{Abs}\ \text{Variable Term} \\
&\mid \mathit{App}\ \text{Term Term} \\
\textbf{type}\ \text{Variable} &= \text{String}
\end{aligned}
$$

Subterms

$$
\begin{aligned}
&\mathit{subterms} && :: \text{Term} \to [\text{Term}] \\
&\mathit{subterms}\ (\mathit{Var}\ x) && = [\,] \\
&\mathit{subterms}\ (\mathit{Abs}\ x\ t) && = [t] \\
&\mathit{subterms}\ (\mathit{App}\ t\ u) && = [t, u]
\end{aligned}
$$

Collecting Subtrees

External Binary Search Trees

$$\begin{aligned}\textbf{data}\ \mathsf{Tree}\ (\mathsf{a} :: \star)\ (\mathsf{b} :: \star) = {} & \mathit{Tip}\ \mathsf{a} \\ \mid {} & \mathit{Node}\ (\mathsf{Tree\ a\ b})\ \mathsf{b}\ (\mathsf{Tree\ a\ b})\end{aligned}$$

Subtrees

$$\begin{aligned}&\mathit{subtrees} :: \forall(\mathsf{a} :: \star)\ (\mathsf{b} :: \star)\ .\ \mathsf{Tree\ a\ b} \to [\mathsf{Tree\ a\ b}] \\ &\mathit{subtrees}\ (\mathit{Tip}\ a) = [\,] \\ &\mathit{subtrees}\ (\mathit{Node}\ l\ b\ r) = [l, r]\end{aligned}$$

No Access to Recursive Calls

Generic Children

$$children\ \langle \mathsf{a} :: \star \rangle :: \mathsf{a} \to [\mathsf{a}]$$

No Access to Recursive Calls

Generic Children

$$children\ \langle \mathsf{a} :: \star \rangle :: \mathsf{a} \to [\mathsf{a}]$$

Problem

The standard view does not provide access to recursive calls in data types

Work-around

Type-level Fixed-point Operator

$$\mathbf{data}\ \mathsf{Fix}\ (\mathsf{f} :: \star \to \star) = \mathit{In}\ (\mathsf{f}\ (\mathsf{Fix}\ \mathsf{f}))$$

Define recursive data types in terms of Fix

Work-around

Terms

$$
\begin{aligned}
\textbf{data}\ \text{Term} &= \mathit{Var}\ \text{Variable} \\
&\mid \mathit{Abs}\ \text{Variable Term} \\
&\mid \mathit{App}\ \text{Term Term} \\
\textbf{type}\ \text{Variable} &= \text{String}
\end{aligned}
$$

Implicit Recursion

$$
\begin{aligned}
\textbf{data}\ \text{TermBase}\ (\text{r} :: \star) &= \mathit{VarBase}\ \text{Variable} \\
&\mid \mathit{AbsBase}\ \text{Variable r} \\
&\mid \mathit{AppBase}\ \text{r r} \\
\textbf{type}\ \text{Term}' &= \text{Fix TermBase}
\end{aligned}
$$

Work-around

External Binary Search Trees

$$\begin{aligned}\textbf{data}\ \mathsf{Tree}\ (\mathsf{a} :: \star)\ (\mathsf{b} :: \star) = {} & \mathit{Tip}\ \mathsf{a} \\ \mid {} & \mathit{Node}\ (\mathsf{Tree\ a\ b})\ \mathsf{b}\ (\mathsf{Tree\ a\ b})\end{aligned}$$

Implicit Recursion

$$\begin{aligned}\textbf{data}\ \mathsf{TreeBase}\ (\mathsf{a} :: \star)\ (\mathsf{b} :: \star)\ (\mathsf{r} :: \star) = {} & \mathit{TipBase}\ \mathsf{a} \\ \mid {} & \mathit{NodeBase}\ \mathsf{r\ b\ r} \\ \textbf{type}\ \mathsf{Tree}'\ (\mathsf{a} :: \star)\ (\mathsf{b} :: \star) = {} & \mathsf{Fix}\ (\mathsf{TreeBase\ a\ b})\end{aligned}$$

Work-around

Generic Collecting

$$
\begin{array}{lll}
\mathit{collect}\ \langle \mathsf{a} :: \star \mid \mathsf{c} :: \star \rangle & :: & (\mathit{collect}\ \langle \mathsf{a} \mid \mathsf{c} \rangle) \Rightarrow \mathsf{a} \to [\mathsf{c}] \\
\mathit{collect}\ \langle \mathsf{Unit} \rangle \quad \mathit{Unit} & = & [\,] \\
\mathit{collect}\ \langle \mathsf{Sum}\ \alpha\ \beta \rangle \quad (\mathit{Inl}\ a) & = & \mathit{collect}\ \langle \alpha \rangle\ a \\
\mathit{collect}\ \langle \mathsf{Sum}\ \alpha\ \beta \rangle \quad (\mathit{Inr}\ b) & = & \mathit{collect}\ \langle \beta \rangle\ b \\
\mathit{collect}\ \langle \mathsf{Prod}\ \alpha\ \beta \rangle \quad (a \times b) & = & \mathit{collect}\ \langle \alpha \rangle\ a +\!\!+ \mathit{collect}\ \langle \beta \rangle\ b \\
\mathit{collect}\ \langle \mathsf{Char} \rangle \quad c & = & [\,]
\end{array}
$$

Generic Children

$$
\begin{array}{lll}
\mathit{children}\ \langle \mathsf{a} :: \star \rangle & :: & (\mathit{collect}\ \langle \mathsf{a} \mid \mathsf{a} \rangle) \Rightarrow \mathsf{a} \to [\mathsf{a}] \\
\mathit{children}\ \langle \mathsf{Fix}\ \phi \rangle\ (\mathit{In}\ r) & = & \mathbf{let}\ \mathit{collect}\ \langle \alpha \rangle\ a = [a] \\
 & & \mathbf{in}\ \ \mathit{collect}\ \langle \phi\ \alpha \rangle\ r
\end{array}
$$

Fixed-point Encoding

Terms

$$\begin{aligned}\textbf{data}\ \text{Term} &= \mathit{Var}\ \text{Variable}\\ &\mid \mathit{Abs}\ \text{Variable Term}\\ &\mid \mathit{App}\ \text{Term Term}\\ \textbf{type}\ \text{Variable} &= \text{String}\end{aligned}$$

Structural Representation

$$\begin{aligned}\textbf{type}\ \textbf{Str}(\text{Term}) &= \text{Fix}\ (\textbf{Ptr}(\text{Term}))\\ \textbf{data}\ \textbf{Ptr}(\text{Term})\ (\text{r} :: \star) &= \textbf{Ptr}(\mathit{Var})\ \text{Variable}\\ &\mid \textbf{Ptr}(\mathit{Abs})\ \text{Variable r}\\ &\mid \textbf{Ptr}(\mathit{App})\ \text{r r}\end{aligned}$$

Implementing Views

- Only two aspects of the machinery need to be adapted
- Generation of structural representations
- Generation of embedding-projection pairs

Isomorphisms

Embedding-projection Pairs

$$\mathbf{data}\ \mathsf{EP}\ (\mathsf{a} :: \star)\ (\mathsf{b} :: \star) = EP\ \{from :: \mathsf{a} \to \mathsf{b},\ to :: \mathsf{b} \to \mathsf{a}\}$$

Embedding-projection pairs witness isomorphisms between data types and stuctural representations

Requirement

$$to\ ep \cdot from\ ep \equiv id$$

for $ep :: \mathsf{EP\ T}\ (\mathbf{Str}(\mathsf{T}))$

Embedding-projection Pairs for the Balanced View

Lists

$$\begin{aligned}\textbf{data}\ \mathsf{Day} = {} & \mathit{Monday} \mid \mathit{Tuesday} \mid \mathit{Wednesday} \mid \mathit{Thursday} \\ & \mid \mathit{Friday} \mid \mathit{Saturday} \mid \mathit{Sunday}\end{aligned}$$

Structural Representation

$$\begin{aligned}\textbf{type}\ \mathbf{Str}(\mathsf{Day}) = {} & \mathsf{Sum}\ (\mathsf{Sum}\ \mathsf{Unit}\ (\mathsf{Sum}\ \mathsf{Unit}\ \mathsf{Unit})) \\ & (\mathsf{Sum}\ (\mathsf{Sum}\ \mathsf{Unit}\ \mathsf{Unit})\ (\mathsf{Sum}\ \mathsf{Unit}\ \mathsf{Unit}))\end{aligned}$$

Embedding-projection Pairs for the Balanced View

From

$$
\begin{aligned}
&\mathbf{from}(\mathsf{Day}) && :: \mathsf{Day} \to \mathbf{Str}(\mathsf{Day}) \\
&\mathbf{from}(\mathsf{Day})\ \mathit{Monday} && = \mathit{Inl}\ (\mathit{Inl}\ \mathit{Unit}) \\
&\vdots && \vdots\ \vdots \\
&\mathbf{from}(\mathsf{Day})\ \mathit{Sunday} && = \mathit{Inr}\ (\mathit{Inr}\ (\mathit{Inr}\ \mathit{Unit}))
\end{aligned}
$$

To

$$
\begin{aligned}
&\mathbf{to}(\mathsf{Day}) && :: \mathbf{Str}(\mathsf{Day}) \to \mathsf{Day} \\
&\mathbf{to}(\mathsf{Day})\ (\mathit{Inl}\ (\mathit{Inl}\ \mathit{Unit})) && = \mathit{Monday} \\
&\vdots && \vdots\ \vdots \\
&\mathbf{to}(\mathsf{Day})\ (\mathit{Inr}\ (\mathit{Inr}\ (\mathit{Inr}\ \mathit{Unit}))) && = \mathit{Sunday}
\end{aligned}
$$

Embedding-projection Pairs for the List-like View

Example

$$\mathbf{data}\ \text{Example}\ (\text{a} :: \star)\ (\text{b} :: \star) = A \mid B\ \text{a} \mid C\ \text{a}\ \text{b}$$

Structural Representation

$$\begin{aligned}&\mathbf{type}\ \mathbf{Str}(\text{Example})\ (\text{a} :: \star)\ (\text{b} :: \star) = \\ &\quad \text{Sum Unit} \\ &\qquad (\text{Sum}\ (\text{Prod a Unit}) \\ &\qquad\quad (\text{Sum}\ (\text{Prod a}\ (\text{Prod b Unit}))\ \text{Zero}))\end{aligned}$$

Embedding-projection Pairs for the List-like View

From

$$
\begin{aligned}
&\mathbf{from}(\text{Example}) && :: \forall(\text{a} :: \star)\ (\text{b} :: \star)\ .\\
&&& \quad \text{Example a b}\\
&&& \quad \rightarrow \mathbf{Str}(\text{Example})\ \text{a b}\\
&\mathbf{from}(\text{Example})\ A && = \mathit{Inl}\ \mathit{Unit}\\
&\mathbf{from}(\text{Example})\ (B\ a) && = \mathit{Inr}\ (\mathit{Inl}\ (a \times \mathit{Unit}))\\
&\mathbf{from}(\text{Example})\ (C\ a\ b) && = \mathit{Inr}\ (\mathit{Inr}\ (a \times (b \times \mathit{Unit})))
\end{aligned}
$$

Embedding-projection Pairs for the List-like View

To

$$
\begin{aligned}
&\mathbf{to}(\text{Example}) && :: \\
&\quad \forall(\text{a} :: \star)\ (\text{b} :: \star)\ .\ \mathbf{Str}(\text{Example})\ \text{a}\ \text{b} && \rightarrow \text{Example a b} \\
&\mathbf{to}(\text{Example})\ (\mathit{Inl}\ \mathit{Unit}) && = A \\
&\mathbf{to}(\text{Example})\ (\mathit{Inr}\ (\mathit{Inl}\ (a \times \mathit{Unit}))) && = B\ a \\
&\mathbf{to}(\text{Example})\ (\mathit{Inr}\ (\mathit{Inr}\ (a \times (b \times \mathit{Unit})))) && = C\ a\ b
\end{aligned}
$$

Embedding-projection Pairs for the Fixed-point View

Terms

$$\begin{aligned}\textbf{data}\ \text{Term} &= \mathit{Var}\ \text{Variable}\\ &\mid \mathit{Abs}\ \text{Variable Term}\\ &\mid \mathit{App}\ \text{Term Term}\\ \textbf{type}\ \text{Variable} &= \text{String}\end{aligned}$$

Structural Representation

$$\begin{aligned}\textbf{type}\ \textbf{Str}(\text{Term}) &= \text{Fix}\ (\textbf{Ptr}(\text{Term}))\\ \textbf{data}\ \textbf{Ptr}(\text{Term})\ (\text{r} :: \star) &= \textbf{Ptr}(\mathit{Var})\ \text{Variable}\\ &\mid \textbf{Ptr}(\mathit{Abs})\ \text{Variable r}\\ &\mid \textbf{Ptr}(\mathit{App})\ \text{r r}\end{aligned}$$

Embedding-projection Pairs for the Fixed-point View

From

$$
\begin{aligned}
&\mathbf{from}(\mathsf{Term}) && :: \mathsf{Term} \to \mathbf{Str}(\mathsf{Term}) \\
&\mathbf{from}(\mathsf{Term})\ (\mathit{Var}\ x) && = \mathit{In}\ (\mathit{VarBase}\ x) \\
&\mathbf{from}(\mathsf{Term})\ (\mathit{Abs}\ x\ t) && = \mathit{In}\ (\mathit{AbsBase}\ x\ (\mathbf{from}(\mathsf{Term})\ t)) \\
&\mathbf{from}(\mathsf{Term})\ (\mathit{App}\ t\ u) && = \\
&\qquad \mathit{In}\ (\mathit{AppBase}\ (\mathbf{from}(\mathsf{Term})\ t)\ (\mathbf{from}(\mathsf{Term})\ u))
\end{aligned}
$$

Embedding-projection Pairs for the Fixed-point View

To

$$
\begin{aligned}
&\mathbf{to}(\text{Term}) && :: \mathbf{Str}(\text{Term}) \to \text{Term} \\
&\mathbf{to}(\text{Term})\ (In\ (VarBase\ x)) && = Var\ x \\
&\mathbf{to}(\text{Term})\ (In\ (AbsBase\ x\ t)) && = Abs\ x\ (\mathbf{to}(\text{Term})\ t) \\
&\mathbf{to}(\text{Term})\ (In\ (AppBase\ t\ u)) && = App\ (\mathbf{to}(\text{Term})\ t)\ (\mathbf{to}(\text{Term})\ u)
\end{aligned}
$$

Encoding Recursion Requires Recursion

Problem

The fixed-point view does not allow embedding-projection pairs for types with a higher-order kind

Possible Solutions

- Restrict the view domain
- Introduce additional dependencies on *bimap*
- Choose other structural representations

Alternative Recursion Operator

Recursion

$$\mathbf{data}\ \mathrm{Rec}\ (\mathrm{f} :: \star \to \star)\ (\mathrm{r} :: \star) = \mathit{Rec}\ (\mathrm{f}\ \mathrm{r})\ (\mathrm{EP}\ \mathrm{r}\ (\mathrm{Rec}\ \mathrm{f}\ \mathrm{r}))$$

Used instead of Fix to model recursion in data types

Other Views

- Variations with (list-like) sums and products
- Based on types of kind $\star \longrightarrow \star$
- Constructors and labels
- ‘Scrap Your Boilerplate’
- XML
- . . .

Conclusion and Future Work

- Views make generic programming in Haskell more efficient, more intuitive, and more expressive
- Views allow us to implement other idioms for generic programming in Generic Haskell

- Applications
- User-defined views
- Most general view