Generic grouping and sorting

RALF HINZE

Institut für Informatik III, Universität Bonn
Römerstraße 164, 53117 Bonn, Germany
Email: ralf@informatik.uni-bonn.de
Homepage: http://www.informatik.uni-bonn.de/~ralf

October, 2005

(Pick up the slides at .../~ralf/talks.html#T45.)

Equivalence relation over a type

Capture equivalence relations using a generalized algebraic data type (GADT):

$$
\begin{array}{l}
\textbf{data}\ \mathit{Equiv} :: \star \to \star\ \textbf{where} \\
\quad \begin{array}{ll}
\mathit{Char} & :: \mathit{Equiv}\ \mathit{Char} \\
\mathit{IgnoreCase} & :: \mathit{Equiv}\ \mathit{Char} \\
\mathit{Unit} & :: \mathit{Equiv}\ () \\
\mathit{Sum} & :: \mathit{Equiv}\ \tau_1 \to \mathit{Equiv}\ \tau_2 \to \mathit{Equiv}\ (\tau_1 + \tau_2) \\
\mathit{Pair} & :: \mathit{Equiv}\ \tau_1 \to \mathit{Equiv}\ \tau_2 \to \mathit{Equiv}\ (\tau_1 \times \tau_2) \\
\mathit{List} & :: \mathit{Equiv}\ \tau \to \mathit{Equiv}\ [\tau] \\
\mathit{Bag} & :: \mathit{Equiv}\ \tau \to \mathit{Equiv}\ [\tau]
\end{array}
\end{array}
$$

☞ *Set* omitted.

Overview

$$
\begin{aligned}
&\mathit{related} :: (\mathit{Equiv}\ \tau) \to \tau \to \tau \to \mathit{Bool} \\
&\mathit{sort} \quad :: (\mathit{Equiv}\ \tau) \to [\tau] \to [\tau] \\
&\mathit{group} \ :: (\mathit{Equiv}\ \tau) \to [(\tau, \nu)] \to [(\tau, [\nu])]
\end{aligned}
$$

Are two elements related?

$$
\begin{array}{lllll}
related :: (Equiv\ \tau) \rightarrow & \tau \rightarrow & \tau & \rightarrow Bool & \\
related\ \ (Char) & x & y & = x == y & \\
related\ \ (IgnoreCase) & x & y & = toUpper\ x == toUpper\ y & \\
related\ \ (Unit) & x & y & = True & \\
related\ \ (Sum\ r_1\ r_2) & (Inl\ x_1) & (Inl\ y_1) & = related\ (r_1)\ x_1\ y_1 & \\
related\ \ (Sum\ r_1\ r_2) & (Inl\ x_1) & (Inr\ y_2) & = False & \\
related\ \ (Sum\ r_1\ r_2) & (Inr\ x_2) & (Inl\ y_1) & = False & \\
related\ \ (Sum\ r_1\ r_2) & (Inr\ x_2) & (Inr\ y_2) & = related\ (r_2)\ x_2\ y_2 & \\
related\ \ (Pair\ r_1\ r_2) & (x_1, x_2) & (y_1, y_2) & & \\
\quad = related\ (r_1)\ x_1\ y_1 \wedge related\ (r_2)\ x_2\ y_2 & & & & \\
related\ \ (Bag\ r) & xs & ys & & \\
\quad = related\ (List\ r)\ (sort\ (r)\ xs)\ (sort\ (r)\ ys) & & & &
\end{array}
$$

☞ The *List* case can be done generically (not shown). The *IgnoreCase* and the *Bag* case are done via normalization.

Generic sorting

$$
\begin{array}{lll}
sort :: (Equiv\ \tau) \rightarrow [\tau] \rightarrow [\tau] & & \\
sort\ \ (Char) & xs & = sortChar\ xs \\
sort\ \ (IgnoreCase) & xs & = sort\ (Char)\ [toUpper\ x \mid x \leftarrow xs] \\
sort\ \ (Unit) & xs & = xs \\
sort\ \ (Sum\ r_1\ r_2) & xs & = [Inl\ y_1 \mid y_1 \leftarrow sort\ (r_1)\ [x_1 \mid Inl\ x_1 \leftarrow xs]] \\
 & & +\!\!+\ [Inr\ y_2 \mid y_2 \leftarrow sort\ (r_2)\ [x_2 \mid Inr\ x_2 \leftarrow xs]] \\
sort\ \ (Pair\ r_1\ r_2) & xs & = [(x_1, y_2) \mid (x_1, ys_2) \leftarrow group\ (r_1)\ xs \\
 & & \qquad\qquad , y_2 \leftarrow sort\ (r_2)\ ys_2] \\
sort\ \ (Bag\ r) & xs & = sort\ (List\ r)\ [sort\ (r)\ x \mid x \leftarrow xs]
\end{array}
$$

Generic grouping

$$
\begin{array}{l}
group :: (Equiv\ \tau) \rightarrow [(\tau, \nu)] \rightarrow [(\tau, [\nu])] \\
group\ \ (Char) \qquad\quad xs = groupChar\ xs \\
group\ \ (IgnoreCase)\ xs = group\ (Char)\ [(toUpper\ x, v) \mid (x, v) \leftarrow xs] \\
group\ \ (Unit) \qquad\quad xs = make\ ((), [v \mid ((), v) \leftarrow xs]) \\
group\ \ (Sum\ r_1\ r_2) \quad xs \\
\quad = [(Inl\ y_1, vs) \mid (y_1, vs) \leftarrow group\ (r_1)\ [(x_1, v) \mid (Inl\ x_1, v) \leftarrow xs]] \\
\quad +\!\!+ [(Inr\ y_2, vs) \mid (y_2, vs) \leftarrow group\ (r_2)\ [(x_2, v) \mid (Inr\ x_2, v) \leftarrow xs]] \\
group\ \ (Pair\ r_1\ r_2) \quad xs \\
\quad = [\ ((a_1, a_2), vs) \\
\qquad \mid (a_1, ys) \leftarrow group\ (r_1)\ [(a_1, (a_2, v)) \mid ((a_1, a_2), v) \leftarrow xs] \\
\qquad , (a_2, vs) \leftarrow group\ (r_2)\ ys] \\
group\ \ (Bag\ r) \qquad\quad xs = group\ (List\ r)\ [(sort\ (r)\ x, v) \mid (x, v) \leftarrow xs]
\end{array}
$$

Generic grouping — continued

$$
\begin{array}{lll}
make :: (\tau, [\nu]) \rightarrow [(\tau, [\nu])] \\
make & (a, [\,]) & = [\,] \\
make & (a, xs) & = [(a, xs)]
\end{array}
$$

Dealing with arbitrary data types

The top-level structure of a list:

$$
\begin{array}{lll}
fromList :: [\tau] & \rightarrow () + \tau \times [\tau] \\
fromList\ \ [\,] & = Inl\ () \\
fromList\ \ (x : xs) & = Inr\ (x, xs) \\
toList :: () + \tau \times [\tau] & \rightarrow [\tau] \\
toList\ \ (Inl\ ()) & = [\,] \\
toList\ \ (Inr\ (x, xs)) & = x : xs
\end{array}
$$

An equivalence relation for the top-level structure:

$$
\begin{array}{lll}
list :: Equiv\ \tau & \rightarrow Equiv\ (() + \tau \times [\tau]) \\
list\ \ r & = Sum\ Unit\ (Pair\ r\ (List\ r))
\end{array}
$$

The missing piece for *related*:

$$
related\ (List\ r)\ xs\ ys = related\ (list\ r)\ (fromList\ xs)\ (fromList\ ys)
$$