Calculating with Lenses

Optimising Bidirectional Transformations

⌜Hugo Pacheco⌟ Alcino Cunha

DI-CCTC, Universidade do Minho, Portugal

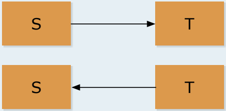
PEPM'11

Austin - January 25th 2011

Bidirectional Transformations

Bidirectional transformations (naive approach)

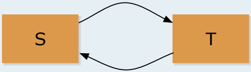
- two separate unidirectional transformations



- manual design: expensive, error-prone, maintenance problem

Bidirectional languages

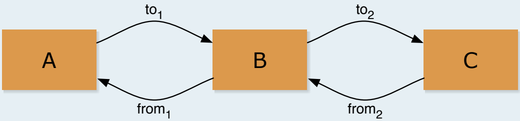
- derive both from the same specification

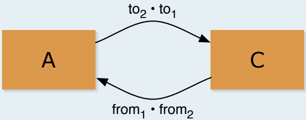


- combinatorial design: clean semantics, compositional

Motivation - Calculation

- compositionality = cluttering

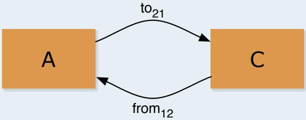




- manual optimisation: unreasonable, impossible?

Goal

- how to optimise bidirectional transformations? a calculus



A point-free design

- An application domain (Trees)

$$\textbf{data}\ Maybe\ a = Nothing \mid Just\ a$$
$$\textbf{data}\ [a] = [\,] \mid a : [a]$$

- A set of combinators

$$\begin{aligned} id &: A \to A \\ \circ &: (B \to C) \to (A \to B) \to (A \to C) \\ \pi_1 &: A \times B \to A \\ i_1 &: A \to A + B \\ \triangle &: (A \to B) \to (A \to C) \to (A \to B \times C) \end{aligned}$$

- An algebraic calculus

$$f \circ (g \circ h) = (f \circ g) \circ h \qquad \circ\text{-}\textsc{Assoc}$$
$$\pi_1 \circ (f \triangle g) = f \wedge \pi_2 \circ (f \triangle g) = g \qquad \times\text{-}\textsc{Cancel}$$

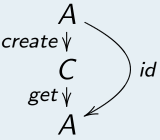
Lenses

$$
\begin{aligned}
get &: C \to A \\
create &: A \to C \\
put &: A \times C \to C
\end{aligned}
$$

$\trianglerighteq$

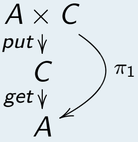
Proving well-behavedness by calculation

- CreateGet



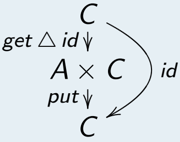
$get \circ create = id$

- PutGet



$get \circ put = \pi_1$

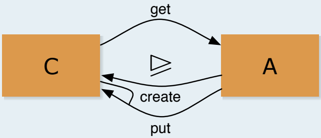
- GetPut



$put \circ (get \bigtriangleup id) = id$

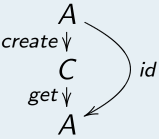
Lenses

$$\begin{aligned} get &: C \to A \\ create &: A \to C \\ put &: A \times C \to C \end{aligned}$$



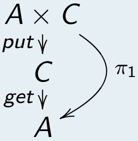
Proving well-behavedness by calculation

- CreateGet



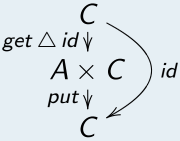
$get \circ create = id$

- PutGet



$get \circ put = \pi_1$

- GetPut



$put \circ (get \triangle id) = id$

A lens point-free design

- An application domain (Lenses over trees)

$$\textbf{data}\ c \trianglerighteq a = Lens\ \{ get :: c \to a,\ create :: a \to c,\ put :: (a, c) \to c \}$$

- A set of combinators

$$\begin{aligned} id &: A \trianglerighteq A \\ \circ &: (B \trianglerighteq C) \to (A \trianglerighteq B) \to (A \trianglerighteq C) \\ \pi_1 &: A \times B \trianglerighteq A \\ \times &: (A \trianglerighteq C) \to (B \trianglerighteq D) \to (A \times B \trianglerighteq C \times D) \end{aligned}$$

- An algebraic calculus

$$f = g \Leftrightarrow \begin{cases} get_f = get_g \\ create_f = create_g \\ put_f = put_g \end{cases}$$

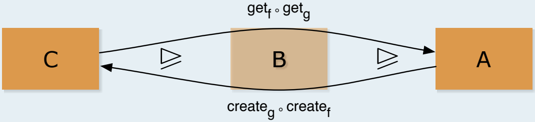
  - which algebraic laws can be lifted to lenses?

Composition as a lens

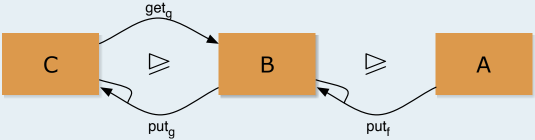
Lens composition

$$\forall f : B \trianglerighteq A,\ g : C \trianglerighteq B.\ \ f \circ g : C \trianglerighteq A$$

$get = get_f \circ get_g$ $\qquad$ $create = create_g \circ create_f$



$$put = put_g \circ (put_f \circ (id \times get_g) \triangle \pi_2) : A \times C \to C$$



$$id \circ f = f = f \circ id \qquad \text{ID-NAT}$$

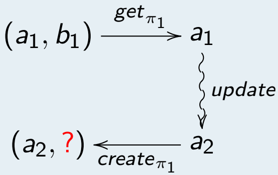
$$f \circ (g \circ h) = (f \circ g) \circ h \qquad \circ\text{-ASSOC}$$

Projection

$$\forall f : A \rightarrow B.\ {\pi_1}^f : A \times B \trianglerighteq A$$

$$\begin{aligned}
get &: A \times B \rightarrow A \\
get &= \pi_1 \\
create &: A \rightarrow A \times B \\
create &= id \triangle f \\
put &: A \times (A \times B) \rightarrow A \times B \\
put &= id \times \pi_2
\end{aligned}$$

- Choice of create



More combinators

$$\forall f : A \trianglerighteq C, g : B \trianglerighteq D.\ f \times g : A \times B \trianglerighteq C \times D$$

$$swap : A \times B \trianglerighteq B \times A$$

$$assoc : A \times (B \times C) \trianglerighteq (A \times B) \times C$$

"Conditional" choice

$$\forall p : C \to 2, f : A \trianglerighteq C, g : B \trianglerighteq C.\ (f \nabla g)^p : A + B \trianglerighteq C$$

$$\begin{array}{lcl}
get & : & A + B \to C \\
get & = & get_f \nabla get_g \\
create & : & C \to A + B \\
create & = & (create_f + create_g) \circ p\,? \\
put & : & C \times (A + B) \to A + B \\
put & = & (put_f + put_g) \circ distr
\end{array}$$

- Choice of create

$$\begin{array}{c}
C \\
\downarrow p? \\
C + C \\
\downarrow create_f + create_g \\
A + B
\end{array}$$

More combinators

$$\forall f : A \trianglerighteq C, g : B \trianglerighteq D.\ f + g : A + B \trianglerighteq C + D$$

$$coswap : A + B \trianglerighteq B + A$$

$$coassoc : A + (B + C) \trianglerighteq (A + B) + C$$

Some lens laws

Products

$$id \times id = id \qquad \times\text{-Functor-Id}$$

$$(f \times g) \circ (h \times i) = f \circ h \times g \circ i \qquad \times\text{-Functor-Comp}$$

$${\pi_1}^{h} \circ (f \times g) = f \circ {\pi_1}^{create_g \circ h \circ get_f} \qquad \pi_1\text{-Nat}$$

$$swap \circ (f \times g) = (g \times f) \circ swap \qquad swap\text{-Nat}$$

$${\pi_1}^{f} \circ swap = {\pi_2}^{f} \qquad swap\text{-Cancel}$$

Sums

$$(id + id) = id \qquad +\text{-Functor-Id}$$

$$(f + g) \circ (h + i) = f \circ h + g \circ i \qquad +\text{-Functor-Comp}$$

$$f \circ (g \nabla h)^{p} = (f \circ g \nabla f \circ h)^{p \circ create_f} \qquad +\text{-Fusion}$$

$$(f \nabla g)^{p} \circ (h + i) = (f \circ h \nabla g \circ i)^{p} \qquad +\text{-Absor}$$

$$(f \nabla g)^{p} \circ coswap = (g \nabla f)^{coswap \circ p} \qquad coswap\text{-Cancel}$$

Recursive lenses

Examples v1

$$\begin{aligned} &\mathit{length} : [A] \trianglerighteq \mathbb{N} \\ &\mathit{get}\ [\,] = 0 \\ &\mathit{get}\ (x : xs) = (\mathit{get}\ xs) + 1 \end{aligned}$$

$$\begin{aligned} &\mathit{map} : (A \trianglerighteq B) \to ([A] \trianglerighteq [B]) \\ &\mathit{get}\ f\ [\,] = [\,] \\ &\mathit{get}\ f\ (x : xs) = \mathit{get}_f\ x : \mathit{get}\ xs \end{aligned}$$

Fixpoints & recursion patterns

$$\begin{array}{ll} [A] \simeq \mu L_A & L_A\ [A] \simeq 1 + A \times [A] \\ \mathit{in}_F : F\ \mu F \trianglerighteq \mu F & \forall f : F\ A \trianglerighteq A.\ (\!|f|\!)_F : \mu F \trianglerighteq A \\ \mathit{out}_F : \mu F \trianglerighteq F\ \mu F & \forall f : A \trianglerighteq F\ A.\ [\!(f)\!]_F : A \trianglerighteq \mu F \end{array}$$

Examples v2

- point-free definitions: "lensification" for free!

$$\begin{aligned} \mathit{length} &= (\!|\mathit{in}_N \circ (\mathit{id} + \pi_2)|\!)_L \\ \mathit{map}\ f &= (\!|\mathit{in}_L \circ (\mathit{id} + f \times \mathit{id})|\!)_L \end{aligned}$$

Recursive lens laws

$$f = (\![g]\!)_F \Leftrightarrow f \circ in_F = g \circ F\ f \qquad (\![\cdot]\!)\text{-UNIQ}$$

$$f \circ (\![g]\!)_F = (\![h]\!)_F \Leftarrow f \circ g = h \circ F\ f \qquad (\![\cdot]\!)\text{-FUSION}$$

$$(\![g]\!) \circ map\ f = (\![g \circ (id + f \times id)]\!) \qquad (\![\cdot]\!)\text{-MAP-FUSION}$$

$$map\ id = id \qquad map\text{-ID}$$

$$map\ f \circ map\ g = map\ (f \circ g) \qquad map\text{-FUSION}$$

$length^v \circ map\ f$
$= \{$length-DEF; $(\![\cdot]\!)$-MAP-FUSION$\}$
$(\![in_N \circ (id + \pi_2{}^{v \circ !}) \circ (id + f \times id)]\!)_L$
$= \{$+-FUNCTOR-COMP; π_2-NAT$\}$

...

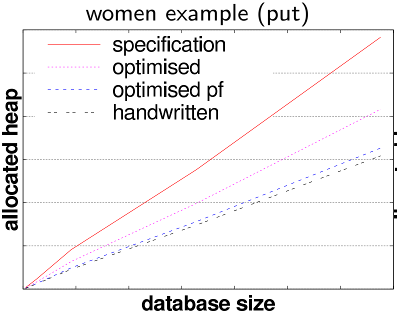
$(\![in_N \circ (id + \pi_2{}^{\underline{create_f\ v} \circ !})]\!)_L$
$= \{$length-DEF$\}$
$length^{create_f\ v}$

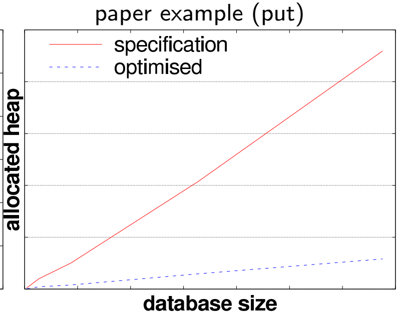
Optimisation

- algebraic laws + term rewriting ⇒ simplification tool
- from a simple example...

$$\textbf{type}\ Person = (Name, Gender)\ \textbf{data}\ Gender = M \mid F$$
$$women : [Person] \trianglerighteq \mathbb{N}$$
$$women = length \circ filter\_r \circ map\ (out_G \circ \pi_{Gender})$$

- ...to complex transformation scenarios





Conclusions

Results

- + Bidirectional language from standard point-free combinators
- + Clear bidirectionalisation: straightforward proofs
- + Support for recursive lenses
- + Algebraic lens calculus
- + Automated optimisation tool

Hugo Pacheco and Alcino Cunha
Generic Point-free Lenses.
Mathematics of Program Construction, 2010.

Demos: Haskell++

- `http://hackage.haskell.org` ⇒ `pointless-lenses`
- `http://hackage.haskell.org` ⇒ `pointless-rewrite`

Future work

- Expressiveness

$$
\begin{aligned}
NonLens ::= {} & i_1 : A \to A + B \mid i_2 : B \to A + B \\
\mid {} & \underline{\cdot} : 1 \to B \quad \mid \; ? : (A \to 2) \to (A \to A + A) \\
\mid {} & \cdot \triangle \cdot : (A \to B) \to (A \to C) \to (A \to B \times C)
\end{aligned}
$$

- partial lenses $\Rightarrow$ partial laws + ill-behaved composition
- idea: go relational... regain totality

- Alignment

- non-determinism
- multiple *puts*
- user's choice?
- idea: tweak recursion?