Delta Lenses over Inductive Types

⌜Hugo Pacheco⌟[1], Alcino Cunha[1] and Zhenjiang Hu[2]

[1]HASLab / INESC TEC & Universidade do Minho, Braga, Portugal

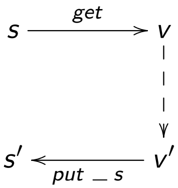
[2]National Institute of Informatics, Tokyo, Japan

BX 2012

Tallinn - March 25th 2012

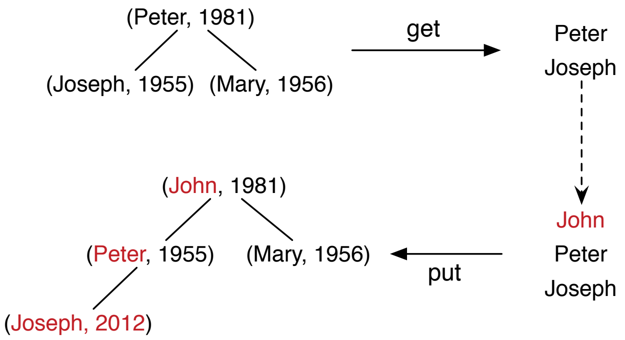
State-based Lenses

- traditional framework
- updates are represented as states



$$Lens\ S\ V$$
$$get : S \rightarrow V$$
$$put : V \rightarrow S \rightarrow S$$

State-based Lens Example



Lens GenTree People

$$\textbf{data}\ \mathit{GenTree} = \mathit{Empty} \mid \mathit{Node}\ \mathit{Person}\ \mathit{GenTree}\ \mathit{GenTree}$$

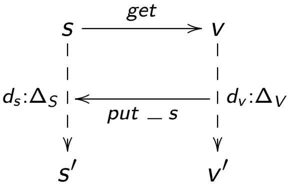
$$\textbf{type}\ \mathit{Person} = (\mathit{Name}, \mathit{Birth})$$

$$\textbf{type}\ \mathit{LeftAsc} = [\mathit{Name}]$$

- no update information, positional behavior
- birth years? Mary?

Delta-based Lenses

- abstract framework
- deltas model "change"

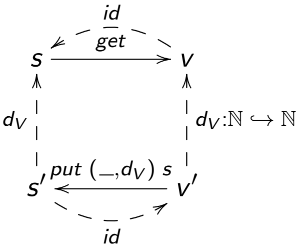


$$DLens\ S\ V$$

$$get : S \to V$$

$$put : \Delta_V \to S \to \Delta_S$$

- instantiate delta lenses (strings)
- types with a notion of position



$$MLens\ S\ V$$

$$get : S \to V$$

$$put : V \times (\mathbb{N} \hookrightarrow \mathbb{N}) \to S \to S$$

Positions/Deltas over Inductive Types

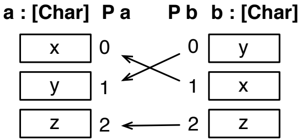
- Polymorphic inductive types can be seen as containers

$$\begin{aligned} \mathit{shape} &: T\ A \to T\ 1 \\ \mathit{data} &: \forall v : T\ A.\ (P\ v \to A) \end{aligned}$$

- $P\ v$ is a dependent type (defined over the structure)
- For lists $[A]$, $\mathit{shape}\ l = \mathit{length}\ l$, $P\ l \cong \{0 \mathinner{..} \mathit{length}\ l - 1\}$

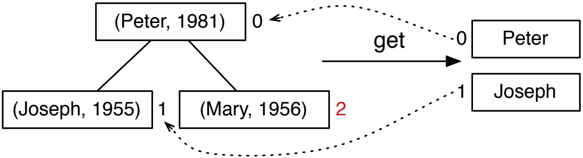
$$\mathbf{data}\ [a] = \mathit{Nil} \mid \mathit{Cons}\ a\ [a]$$

- $P\ v \cong \{0 \mathinner{..} n - 1\}$, n number of placeholders in v
- A delta $b \Delta a$ is a partial function $P\ b \hookrightarrow P\ a$



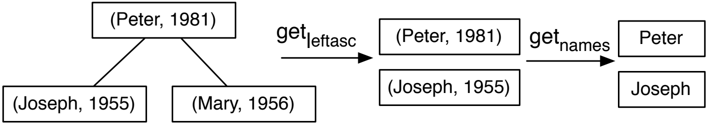
Decomposing the Example

- matching lenses cannot drop positions



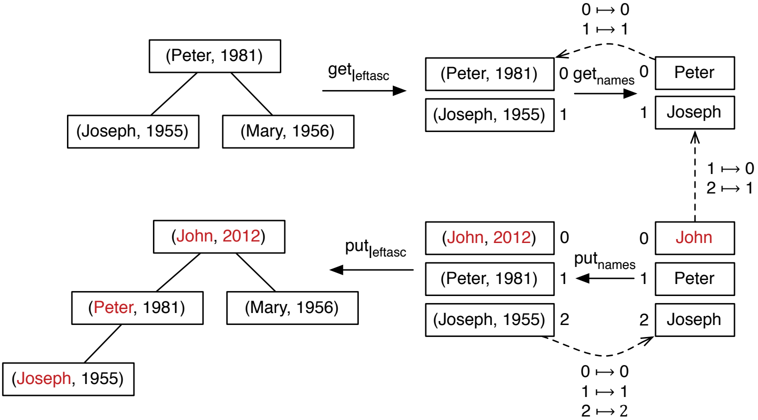
type *GenTree* = *Tree Person* **type** *LeftAsc* = [ ] *Name*
data *Tree a* = *Empty* | *Node a* (*Tree a*) (*Tree a*)

- decompose the example (reshaping + mapping)



leftasc : *Lens GenTree* [*Person*] *names* : *MLens* ([ ] *Person*) ([ ] *Name*)

Matching Lens Example

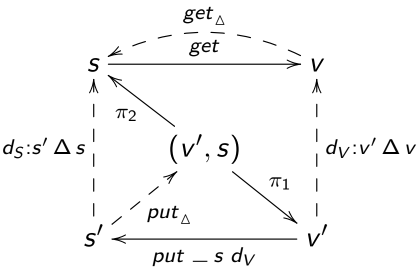


- data alignment, positional shape behavior
- Mary?

Motivation

- so far...
  - delta lenses good for end-users
  - implementations require traceability
  - matching lenses have trivial traceability
  - half of the problem: data alignment
- our goal...
  - framework of horizontal delta lenses with traceability
  - language of horizontal delta lenses
  - horizontal = trace
  - full problem: data and shape alignment

- instantiate delta lenses (polymorphic inductive types)
- additional traceability (horizontal) deltas



$$P\ (a, b) = P\ a + P\ b$$
$$d_S = (get_\triangle \circ d_V \nabla id) \circ put_\triangle$$

$HDLens\ (S\ A)\ (V\ B)$

$get : S\ A \rightarrow V\ B$

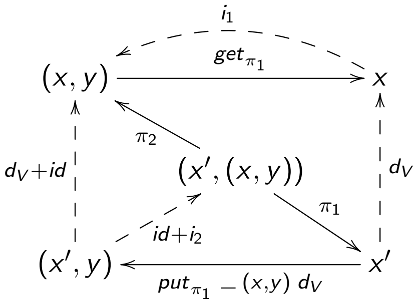
$put : \forall v' : V\ B, s : S\ A.\ v' \Delta\, get\ s \rightarrow S\ A$

$get_\triangle : \forall s : S\ A.\ get\ s \Delta\, s$

$put_\triangle : \forall v' : V\ B, s : S\ A, d_V : v' \Delta\, get\ s.\ put\ (v', s)\ d_V \Delta\, (v', s)$

Horizontal Delta Lenses Language (Projection)

- drop the second element of a product
- PF horizontal deltas



$$\pi_1 : HDLens\ ((F \otimes G)\ A)\ (F\ A)$$

Shape Alignment VS Data Alignment

- composition, products, sums, etc
- for mapping lenses, that preserve the shape
  - new source shape = new view shape
  - new source data = new view data + lost source data retrieved by the data associations
- for reshaping lenses, that preserve the data
  - new source shape $\neq$ new view shape
  - avoid disrupting the original shape (eg, relationship between people in a genealogical tree)
  - identify where modifications occur in the view shape
  - propagate to where they shall occur in the source shape

What are Shape Updates?

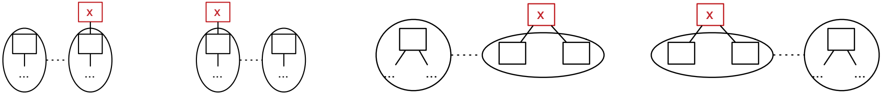
- polymorphic type arguments = data locations

$$\textbf{data}\ [a] = \mathit{Nil} \mid \mathit{Cons}\ a\ [a]$$
$$\textbf{data}\ \mathit{Tree}\ a = \mathit{Empty} \mid \mathit{Node}\ a\ (\mathit{Tree}\ a)\ (\mathit{Tree}\ a)$$

- "head" of type constructors = shape locations

$$\textbf{data}\ [a] = \mathit{Nil} \mid \mathit{Cons}\ a\ [a]$$
$$\textbf{data}\ \mathit{Tree}\ a = \mathit{Empty} \mid \mathit{Node}\ a\ (\mathit{Tree}\ a)\ (\mathit{Tree}\ a)$$

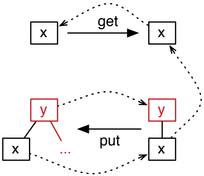
- insertions / deletions on lists and trees



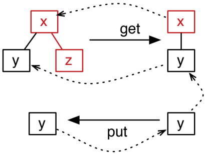
- propagating view (shape) updates to source (shape) updates
- requires knowing the behavior of the lens

$$
\begin{aligned}
&get_{leftasc} : Tree\ Person \rightarrow [Person] \\
&get_{leftasc}\ Empty = Nil \\
&get_{leftasc}\ (Node\ p\ l\ r) = Cons\ p\ (get_{leftasc}\ l)
\end{aligned}
$$

- insertion

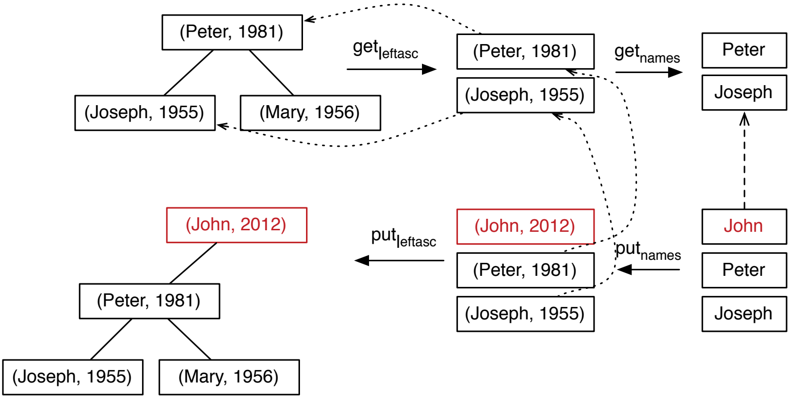


- deletion



- alignment-aware folds $(\!|\cdot|\!)$ and unfolds $[\![\cdot]\!]$

Horizontal Delta Lens Example



$$leftasc : HDLens\ (Tree\ Person)\ [Person] \qquad names : HDLens\ [Person]\ [Name]$$
$$leftasc = (\!|in \circ (id + id \times \pi_1)|\!) \qquad names = map\ \pi_1$$

- data alignment, shape alignment

Conclusions

- \+ instantiate delta lenses for inductive types
- \+ formalize positions and deltas (dependent types)
- \+ framework of horizontal delta lenses
- \+ delta lens combinators (composition, products, sums)
- \+ data alignment (mapping)
- \+ shape alignment (folds and unfolds)
- \- more operations
- \- domain-specific language

Demos: Haskell++

- other examples (filtering, concatenation, ...)
- `cabal install pointless-lenses`