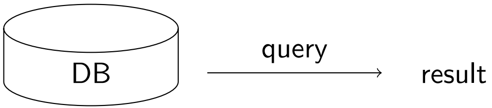
Generalizing Lenses

A New Foundation for Bidirectional Programming

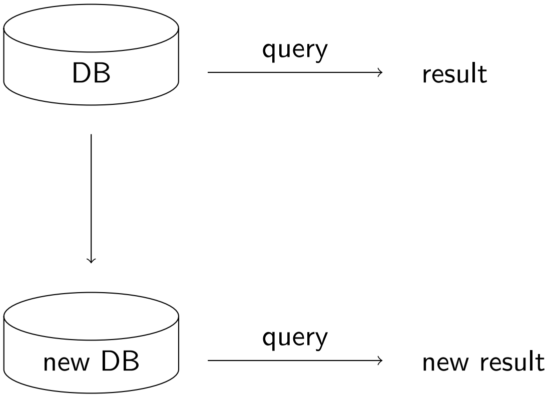
Daniel Wagner

June 13, 2014

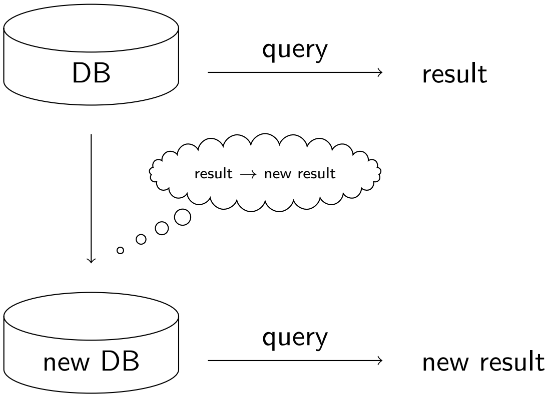
History: a problem with databases



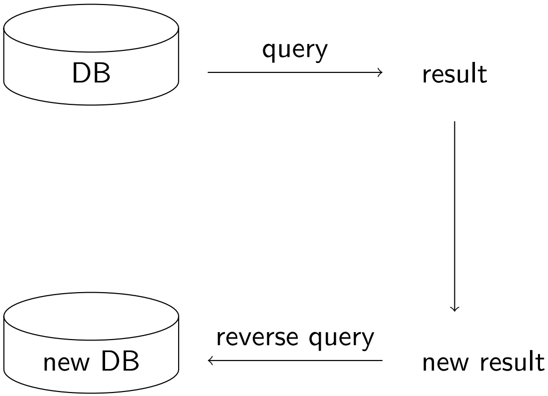
History: a problem with databases



History: a problem with databases



History: a problem with databases



Bidirectional programming

Many other settings with similar problems, like

- parsing (in-memory structures $\leftrightarrow$ serialization),
- software model transformations (diagrams $\leftrightarrow$ code),
- user interfaces (connecting two widgets' state), and
- sysadmin (custom configurations $\leftrightarrow$ unified format).

In each setting,

- two pieces of data (henceforth, *repositories*) are related, and
- we would like to avoid writing two related transformations.

Dissatisfaction

Language-based research is centered on asymmetric lenses. But:

Asymmetry A canonical repository stores all information,

Misalignment Lenses have limited access to information connecting old and new repositories, and

Performance Traversing entire repositories requires high computation and memory resources.

. . . though the extensive syntax is a key feature to keep.

Contributions

Symmetric lenses are the first lens framework that:

- Gives both repositories equal status
- Provide a computable sequential composition
- Retain modular reasoning principles

Edit lenses extend symmetric lenses with:

- Explicit representation of and computation with changes
- Support for incremental operation
- Behavioral laws constraining update

A prototype implementation explores the problem of generating change information.

Related work

	Alignment	Symmetry	Performance	Syntax
algebraic	edits	no	possibly, but unexplored	not a goal
matching	mapping from holes to holes	no	repository and alignment information both processed	variants of most AS-lens combinators
annotated	insertion, deletion, modification markers	no	alignment information includes repository	includes $\mathit{diag} \in X \leftrightarrow X \times X$
asymm. δ	explicit alignments	not a goal	edits include repositories	via alternate framework
symm. δ	edits	yes, but equiv. not explored	edits include repositories	alternate frameworks not instantiated
const. maint.	uninterpreted edits	yes; does not require equiv.	no; all edits relative to *init*	many primitives, but no composition
symm. state	very bad	yes; requires equivalence	no	mostly domain agnostic
edit lenses	edits	yes; requires equivalence	small edits support incremental operation	most standard lenses, and container map

green means satisfies the objective, red indicates some shortcomings

Other models of edits

- $X \times X$ (before and after)
  - State-based lenses
  - ✓ Very simple starting point
  - ✗ Not enough information about alignment
- $X \to X$ (extensional edit operation)
  - Stevens' algebraic study of delta lenses
  - ✓ Models many behaviors
  - ✗ Difficult to recover intensional data
- category on X (collection of edits for each before/after pair)
  - Diskin, et al's delta lenses
  - ✓ Very rich information about change
  - ✗ Very rich information about change

Modules

∂	changes to
$\odot$	apply
init	initial value
$\rightharpoonup$	partial function to

Keep the best features of each: collection of edits for easy introspection + mapping to functions to cover many behaviors.

Module $\langle X, \partial X, \odot_X, \mathit{init}_X \rangle$ is:

- Set of values to be edited X
- Monoid of edits ∂X
- Homomorphism from edits to operations $\odot_X \in \partial X \to X \rightharpoonup X$
- Default value init_X is a technical detail; explanation later

Monoids

$\mathbf{1}$	identity
$m\ n$	m times n

Quick review: monoid means

- There is an identity $\mathbf{1}$
- and an associative binary operation (juxtaposition).

Homomorphisms f respect this structure.

$$f(\mathbf{1}) = \mathbf{1}$$

$$f(m\ n) = f(m)\ f(n)$$

In particular, for edits: identity always succeeds and does nothing, and edits can be run in sequence.

Partiality

$$\odot_X \in \partial X \to X \rightharpoonup X$$

Partiality

$$\odot_X \in \partial X \to X \rightharpoonup X$$

Why not just do nothing instead of failing?

Partiality

$\triangleq$	is defined equal to
$\mapsto$	becomes

$$\odot_X \in \partial X \to X \rightharpoonup X$$

Requiring totality forces you to include unnatural edits.

$$M \triangleq \{\mathbf{1}\} \cup \{a \mapsto b \mid a, b \in \mathbb{N}\}$$

With totality:

$$(a \mapsto b)\;(b \mapsto c) \odot a = c$$
$$(a \mapsto b)\;(b \mapsto c) \odot b = c$$

... must expand M to accommodate this.
With partiality, can define

$$(a \mapsto b)\;(b \mapsto c) \triangleq a \mapsto c$$

Partiality

$\triangleq$	is defined equal to
$\mapsto$	becomes

$$\odot_X \in \partial X \to X \rightharpoonup X$$

Requiring totality forces you to include unnatural edits.

$$M \triangleq \{\mathbf{1}\} \cup \{a \mapsto b \mid a, b \in \mathbb{N}\}$$

With totality:

$$(a \mapsto b)\ (b \mapsto c) \odot a = c$$
$$(a \mapsto b)\ (b \mapsto c) \odot b = c$$

... must expand M to accommodate this.
With partiality, can define

$$(a \mapsto b)\ (b \mapsto c) \triangleq a \mapsto c$$

Theorem: Partiality is an illusion.

Data structures

Common approach to implementing complex data structures:

$$\tau := 0 \mid 1 \mid X \mid \tau + \tau \mid \tau \times \tau \mid \mu X.\, \tau \mid \tau \to \tau$$

Try to design edit modules for each of these types.

Data structures

Common approach to implementing complex data structures:

$$\tau := 0 \mid 1 \mid X \mid \tau + \tau \mid \tau \times \tau \mid \mu X.\, \tau \mid \tau \to \tau$$

Try to design edit modules for each of these types.

Data structures

Common approach to implementing complex data structures:

$$\tau := 0 \mid 1 \mid X \mid \tau + \tau \mid \tau \times \tau \mid \mu X.\, \tau \mid \tau \to \tau$$

Try to design edit modules for each of these types.

Does not work well.

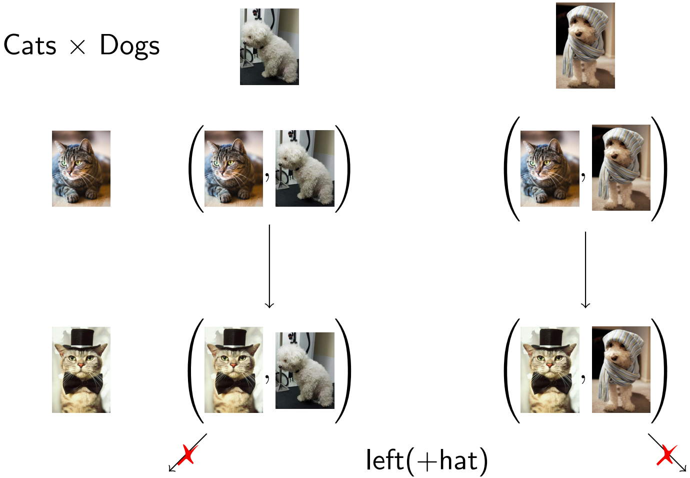
Products

How to edit $X \times Y$? Either edit X or edit Y.

Cats $\times$ Dogs

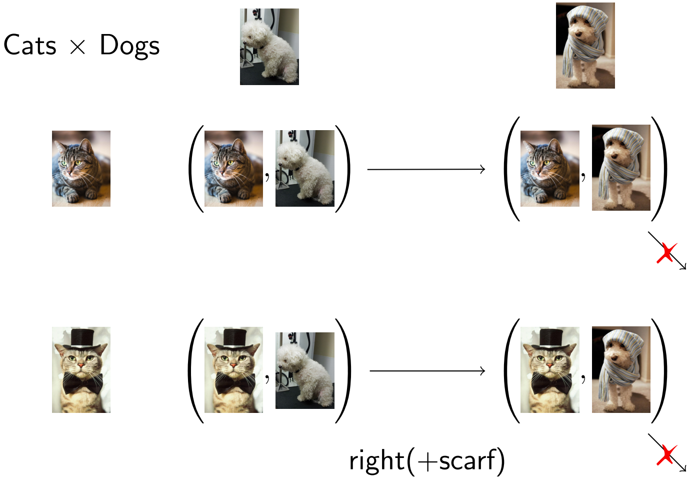
Products

How to edit $X \times Y$? Either edit X or edit Y.



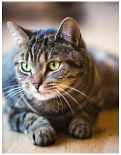
Products

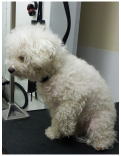
How to edit $X \times Y$? Either edit X or edit Y.

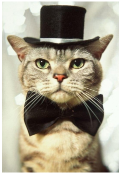


Sums

Cats $+$ Dogs

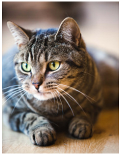


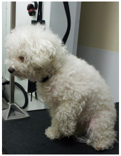


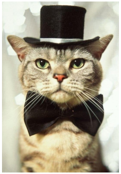


Sums

Cats $+$ Dogs



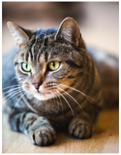


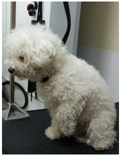


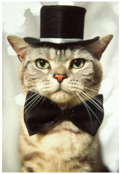
$\mathrm{stay}_L(+\mathrm{hat})$

Sums

Cats $+$ Dogs

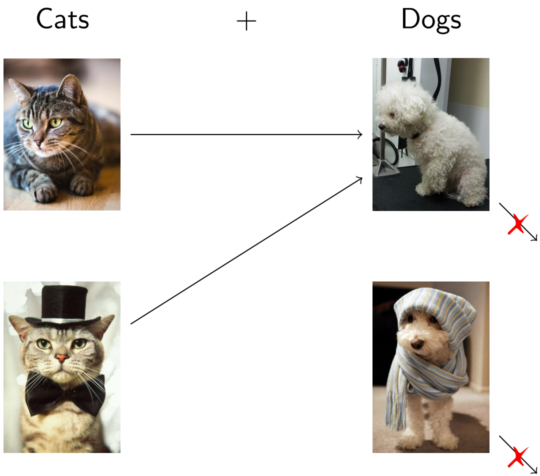






$\mathrm{stay}_R(+\mathrm{scarf})$

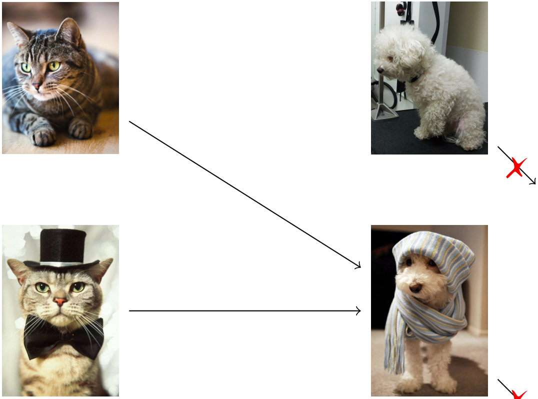
Sums



$\mathsf{switch}_{LR}(\mathbf{1})$

Sums

Cats $+$ Dogs



$\text{switch}_{LR}(+\text{scarf})$

Sums

Cats $+$ Dogs

$\mathsf{switch}_{LL}(\mathbf{1})$

Sums, recap

Six kinds of sum edit for $X + Y$:

$$\begin{gathered}\mathsf{stay}_L(\mathrm{d}x)\\ \mathsf{switch}_{LL}(\mathrm{d}x)\\ \mathsf{switch}_{RL}(\mathrm{d}x)\end{gathered}$$

$$\begin{gathered}\mathsf{stay}_R(\mathrm{d}y)\\ \mathsf{switch}_{LR}(\mathrm{d}y)\\ \mathsf{switch}_{RR}(\mathrm{d}y)\end{gathered}$$

Inductive types

$\tau[X/Y]$ replace Y by X in τ

First idea: $\partial(\mu X.\, \tau) \simeq \partial(\tau[\mu X.\, \tau/X])$.
For lists with elements from module A, i.e. $\mu X.\, 1 + A \times X$:

$\mathsf{stay}_L(\_)$	Do nothing to the currently empty list.
$\mathsf{switch}_{\_L}(\_)$	Delete the entire list.
$\mathsf{stay}_R(\mathsf{left}(\mathrm{d}a))$	Modify the head of the list.
$\mathsf{stay}_R(\mathsf{right}(\mathrm{d}x))$	Modify the tail of the list.
$\mathsf{switch}_{\_R}(\_)$	Replace the current list.

No way to insert or delete in the middle of the list.
Information never migrates; can't swap list elements.

Inductive types

$\tau[X/Y]$ replace Y by X in τ

First idea: $\partial(\mu X.\, \tau) \simeq \partial(\tau[\mu X.\, \tau/X])$.
For lists with elements from module A, i.e. $\mu X.\, 1 + A \times X$:

$\mathsf{stay}_L(\_)$	Do nothing to the currently empty list.
$\mathsf{switch}_{\_L}(\_)$	Delete the entire list.
$\mathsf{stay}_R(\mathsf{left}(\mathrm{d}a))$	Modify the head of the list.
$\mathsf{stay}_R(\mathsf{right}(\mathrm{d}x))$	Modify the tail of the list.
$\mathsf{switch}_{\_R}(\_)$	Replace the current list.

No way to insert or delete in the middle of the list.
Information never migrates; can't swap list elements.

More baroque approaches have other problems.

Containers

A standard container $\langle I, P \rangle$ is

- A set of shapes I and
- For each shape i, a set P_i of positions.

An X*-instance* $\langle i, f \rangle$ of container $\langle I, P \rangle$ is

- A shape $i \in I$ and
- A function $f \in P_i \to X$.

Lists as containers

$\mathbb{N}$ natural numbers

$$I \triangleq \mathbb{N}$$
$$P_i \triangleq \{0, \ldots, i-1\}$$

The list $[3, 6, 2]$ is represented as the pair

$$\left\langle \begin{array}{l} 3, \lambda p.\ \text{if} \quad p = 0 \text{ then } 3 \\ \qquad\quad\ \text{elif } p = 1 \text{ then } 6 \\ \qquad\quad\ \text{elif } p = 2 \text{ then } 2 \end{array} \right\rangle$$

Container restrictions

Three important changes:

- Module of shape edits
- Universe of positions P_U
- Partial order $\leq$ on shapes (with P monotone)

$\simeq$	bijection to

Container module

$$
\begin{aligned}
\partial \langle I, P \rangle_X \triangleq\ & \{\mathsf{mod}(p, \mathrm{d}x) \mid p \in P_U, \mathrm{d}x \in \partial X\} \\
\cup\ & \{\mathsf{ins}(\mathrm{d}i) \mid \mathrm{d}i\ i \geq i \text{ whenever defined}\} \\
\cup\ & \{\mathsf{del}(\mathrm{d}i) \mid \mathrm{d}i\ i \leq i \text{ whenever defined}\} \\
\cup\ & \{\mathsf{swap}(\mathrm{d}i, f) \mid f_i \in P_{\mathrm{d}i\ i} \simeq P_i \text{ whenever defined}\} \\
\cup\ & \{\mathsf{fail}\}
\end{aligned}
$$

Other results: composition

First in-depth study of machinery needed for sequential composition in the presence of symmetry:

- Complements enable computable composition
- Equality is too fine a distinction, but a coarser equivalence relation identifies $j ; (k ; \ell)$ and $(j ; k) ; \ell$
- All lens combinators are proven to respect equivalence classes
- An induced category whose arrows are lenses

Other results: algebraic study

For symmetric lenses:

- Symmetric monoidal product structure
- Symmetric monoidal sum structure
- *Non*-existence of true products and sums
- Projections (natural up to indexing)
- Injections (non-natural)
- Iterator lenses, combined folds and unfolds on inductive types
- Functorial container mapping lens

Other results: algebraic study

For edit lenses:

- Symmetric monoidal product structure
- Tensor sum structure which is bifunctorial and commutative (up to *init* bias) but not associative
- Functorial container mapping lens

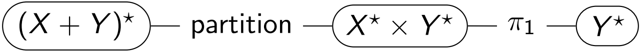
Partition, reshaping (not motivated by algebraic considerations)

Other results: miscellaneous

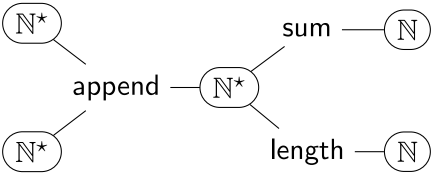
- Asymmetric lenses can be lifted to symmetric lenses
- Symmetric lenses + change detection algorithms can be lifted to edit lenses
- Monoid homomorphism laws refine state-based behavioral laws
- Prototype implementation explores the generation of alignment information

Hyperlenses

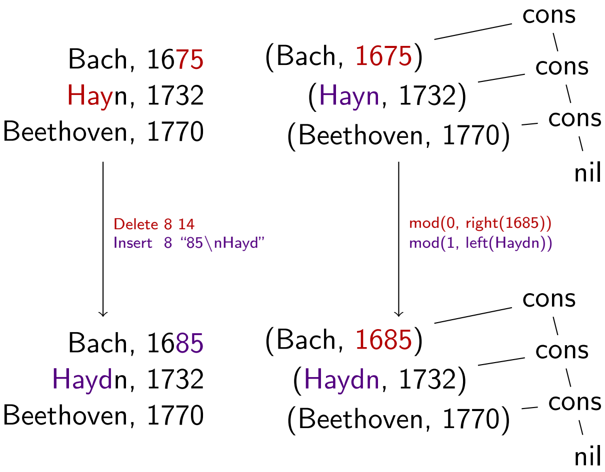
Bidirectional transformations:



Multi-directional transformations:



Edit parsing



Conclusion

Tackled four important problems:

Symmetry,	treating both repositories equally,
Alignment,	tracking changes to improve updates,
Performance,	processing only the data that matters, and
Syntax,	instantiating the framework with many lenses,

an important step for the maintenance of replicated data.

Realistic assumption: symmetry

Mozart	Austria
Mahler	Czech R.
Stevens	England

Mozart	1756
Mahler	1860
Stevens	1948

Handling lists

Mozart	Austria
Mahler	Czech R.
Stevens	England

Mozart	1756
Liszt	1811
Mahler	1860
Stevens	1948

Handling lists

Mozart	Austria
Liszt	Czech R.
Mahler	England
Stevens	missing

Mozart	1756
Liszt	1811
Mahler	1860
Stevens	1948

Alignment

Mozart	Austria
Mahler	Czech R.
Stevens	England

Mozart	1756
Liszt	1811
Mahler	1860
Stevens	1948

Alignment

Mozart	Austria
Mahler	Czech R.
Stevens	England

Mozart	1756
Liszt	1811
Mahler	1860
Stevens	1948

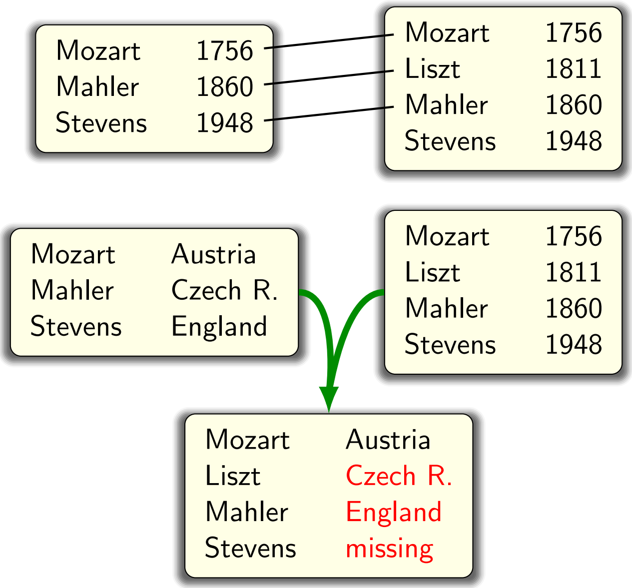
Alignment

Mozart	Austria
Mahler	Czech R.
Stevens	England

Mozart	1756
Mahler	1860
Stevens	1948

Mozart	1756
Liszt	1811
Mahler	1860
Stevens	1948

Alignment failure modes



Alignment failure modes

Mozart	1756
Mahler	1860
Stevens	1948

Mozart	1756
Liszt	1811
Mahler	1860
Stevens	1948

Mozart	Austria
Mahler	Czech R.
Stevens	England

Mozart	1756
Liszt	1811
Mahler	1860
Stevens	1948

Mozart	Austria
Liszt	missing
Mahler	missing
Stevens	England

Alignment failure modes

Mozart	1756
Mahler	1860
Stevens	1948

Mozart	1756
Liszt	1811
Mahler	1860
Stevens	1948

Mozart	Austria
Mahler	Czech R.
Stevens	England

Mozart	1756
Liszt	1811
Mahler	1860
Stevens	1948

Mozart	England
Liszt	Austria
Mahler	<<loop>>
Stevens	Austria

Strange, but true

Alignment		What Changed
Mozart 1756 Stevens 1948	Mozart 1756 Mahler 1860 Stevens 1948	insert Mahler
Mozart 1756 Maller 1860 Stevens 1948	Mozart 1756 Mahler 1860 Stevens 1948	correct typo
Mozart 1756 Liszt 1811 Stevens 1948	Mozart 1756 Mahler 1860 Stevens 1948	replace a composer
Mozart 1756 Mahler 1860 Stevens 1948	Mozart 1756 Mahler 1860 Islam 1948	name change
Bach Germany Liszt Hungary Stevens England	Bach Germany Liszt Hungary Stevens England	replace Bach with son

Future work

- Hyperlenses: multi-repository lenses
- Transforming string edits into structured edits
- Further exploration of the possible edit lenses
- More breadth in the algebraic study of edit lenses
- Many ideas for applications
- Others: variations of the behavioral laws, typed edits, asymmetric edit lenses, connections between various lens frameworks, automatic weight function discovery

Potential application areas

- Filesystem synchronization
- Text editing (decoding, parsing, highlighting)
- GUI internals
- Extensions of Boomerang, Augeas, Forest
- Many-directional spreadsheet
- Relational database
- Bidirectional Datalog
- Server/client applications (e.g. on mobile phones)
- Software model transformations