Generalizing Lenses
A New Foundation for Bidirectional Programming

Daniel Wagner

June 13, 2014
History: a problem with databases
History: a problem with databases

![Diagram](image)

- Query from DB to result
- Query from new DB to new result
History: a problem with databases

DB

query → result

result → new result

new DB

query → new result
History: a problem with databases

![Diagram showing the process of querying a database and generating a new result with a reverse query to create a new database.](image)
Bidirectional programming

Many other settings with similar problems, like

- parsing (in-memory structures ↔ serialization),
- software model transformations (diagrams ↔ code),
- user interfaces (connecting two widgets’ state), and
- sysadmin (custom configurations ↔ 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

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

**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
- \( X \rightarrow X \) (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
Keep the best features of each: collection of edits for easy introspection + mapping to functions to cover many behaviors.

Module \( \langle X, \partial X, \circ_X, \text{init}_X \rangle \) is:

- Set of values to be edited \( X \)
- Monoid of edits \( \partial X \)
- Homomorphism from edits to operations \( \circ_X : \partial X \to X \)
- Default value \( \text{init}_X \) is a technical detail; explanation later
Quick review: monoid means
- There is an identity $1$
- and an associative binary operation (juxtaposition).

Homomorphisms $f$ respect this structure.

$$f(1) = 1$$

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

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

\[ \circ \in \partial X \rightarrow X \rightarrow X \]

Requiring totality forces you to include unnatural edits. With totality:

\[(a \mapsto b)(b \mapsto c) \circ a = c \]
\[(a \mapsto b)(b \mapsto c) \circ b = c . . . \text{must expand} \]

With partiality, can define

\[(a \mapsto b)(b \mapsto c) \equiv a \mapsto c \equiv \text{is defined equal to} \]
\[\mapsto \text{becomes} \]

Theorem: Partiality is an illusion.
Partiality

\[ \circ_X \in \partial X \rightarrow X \rightarrow X \]

Why not just do nothing instead of failing?

... must expand \( M \) to accommodate this.

With partiality, can define \((a \mapsto \rightarrow b) (b \mapsto \rightarrow c) \circ a = c \)

Theorem: Partiality is an illusion.
Partiality

\[ \circ_X \in \partial X \rightarrow X \rightarrow X \]

Requiring totality forces you to include unnatural edits.

\[ M \triangleq \{1\} \cup \{a \mapsto b \mid a, b \in \mathbb{N}\} \]

With totality:

\[
(a \mapsto b) (b \mapsto c) \circ a = c \\
(a \mapsto b) (b \mapsto c) \circ b = c
\]

... must expand \( M \) to accommodate this.

With partiality, can define

\[ (a \mapsto b) (b \mapsto c) \triangleq a \mapsto c \]
Partiality

\[ \otimes_X \in \partial X \rightarrow X \rightarrow X \]

Requiring totality forces you to include unnatural edits.

\[ M \triangleq \{1\} \cup \{a \mapsto b \mid a, b \in \mathbb{N}\} \]

With totality:

\[
(a \mapsto b) (b \mapsto c) \otimes a = c \\
(a \mapsto b) (b \mapsto c) \otimes b = c
\]

... 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:

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

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:

\[ \tau := 0 \mid 1 \mid X \mid \tau + \tau \mid \tau \times \tau \mid \mu X. \tau \mid \tau \rightarrow \tau \]

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

\[
\begin{pmatrix}
\text{Cat},
\end{pmatrix}
\quad
\begin{pmatrix}
\text{Dog},
\end{pmatrix}
\quad
\begin{pmatrix}
\text{Cat},
\end{pmatrix}
\quad
\begin{pmatrix}
\text{Dog},
\end{pmatrix}
\]

\[
\begin{pmatrix}
\text{Cat},
\end{pmatrix}
\quad
\begin{pmatrix}
\text{Dog},
\end{pmatrix}
\quad
\begin{pmatrix}
\text{Cat},
\end{pmatrix}
\quad
\begin{pmatrix}
\text{Dog},
\end{pmatrix}
\]

\[
\begin{pmatrix}
\text{Cat},
\end{pmatrix}
\quad
\begin{pmatrix}
\text{Dog},
\end{pmatrix}
\quad
\begin{pmatrix}
\text{Cat},
\end{pmatrix}
\quad
\begin{pmatrix}
\text{Dog},
\end{pmatrix}
\]
Products

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

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

**Cats $\times$ Dogs**

\[
\begin{pmatrix}
\text{Cats}
\end{pmatrix}
\times
\begin{pmatrix}
\text{Dogs}
\end{pmatrix}
\]

\[
\begin{pmatrix}
\text{Cats}
\end{pmatrix}
\times
\begin{pmatrix}
\text{Dogs}
\end{pmatrix}
\]

right(+$\text{scarf}$)

\[
\begin{pmatrix}
\text{Cats}
\end{pmatrix}
\times
\begin{pmatrix}
\text{Dogs}
\end{pmatrix}
\]

\[
\begin{pmatrix}
\text{Cats}
\end{pmatrix}
\times
\begin{pmatrix}
\text{Dogs}
\end{pmatrix}
\]

right(+$\text{scarf}$)
Sums

Cats + Dogs
Sums

Cats + Dogs

stay_L(±hat)
Sums

Cats + Dogs

stay_R(\text{+scarf})
Sums

\[
\text{Cats} + \text{Dogs} \rightarrow \text{switch}_{LR}(1)
\]
Sums

Cats + Dogs

switch$_{LR}(+$scarf$)$
Sums

\[ \text{Cats} \quad + \quad \text{Dogs} \]

\[ \text{switch}_{LL}(1) \]
Sums, recap

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

- stay$_L$(dx)
- switch$_{LL}$(dx)
- switch$_{RL}$(dx)
- stay$_R$(dy)
- switch$_{LR}$(dy)
- switch$_{RR}$(dy)
Inductive types

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

- \( \text{stay}_L(\_\_) \) Do nothing to the currently empty list.
- \( \text{switch}_L(\_\_) \) Delete the entire list.
- \( \text{stay}_R(\text{left}(da)) \) Modify the head of the list.
- \( \text{stay}_R(\text{right}(dx)) \) Modify the tail of the list.
- \( \text{switch}_R(\_\_) \) 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

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

- $\text{stay}_L(\_)$ Do nothing to the currently empty list.
- $\text{switch}_L(\_)$ Delete the entire list.
- $\text{stay}_R(\text{left}(\text{da}))$ Modify the head of the list.
- $\text{stay}_R(\text{right}(\text{dx}))$ Modify the tail of the list.
- $\text{switch}_R(\_)$ 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 $f \in P_i \rightarrow X$. 
Lists as containers

\[ I \triangleq \mathbb{N} \]

\[ P_i \triangleq \{0, \ldots, i - 1\} \]

The list \([3, 6, 2]\) is represented as the pair

\[
\begin{align*}
3, \lambda p. & \begin{cases} 
p = 0 \text{ then } 3 \\
 p = 1 \text{ then } 6 \\
 p = 2 \text{ then } 2 
\end{cases}
\end{align*}
\]
Three important changes:

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

\[ \partial \langle I, P \rangle_X \triangleq \{ \text{mod}(p, dx) \mid p \in P_U, dx \in \partial X \} \]
\[ \cup \{ \text{ins}(d_i) \mid d_i > i \text{ whenever defined} \} \]
\[ \cup \{ \text{del}(d_i) \mid d_i > i \text{ whenever defined} \} \]
\[ \cup \{ \text{swap}(d_i, f) \mid f_i \in P_{d_i}; i \approx P_i \text{ whenever defined} \} \]
\[ \cup \{ \text{fail} \} \]
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
- \textit{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 \textit{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:

\[(X + Y)^* \quad \text{partition} \quad X^* \times Y^* \quad \pi_1 \quad Y^*\]

Multi-directional transformations:

\[\text{append} \quad \text{sum} \quad \text{length}\]
Edit parsing

Bach, 1675
Hayn, 1732
Beethoven, 1770

(Bach, 1675)
(Hayn, 1732)
(Beethoven, 1770)

Delete 8 14
Insert 8 “85\nHayd”

mod(0, right(1685))
mod(1, left(Haydn))

Bach, 1685
Haydn, 1732
Beethoven, 1770

(Bach, 1685)
(Haydn, 1732)
(Beethoven, 1770)
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 1756
Mahler 1860
Stevens 1948
Handling lists
### Handling lists

<table>
<thead>
<tr>
<th>Composer</th>
<th>Location</th>
<th>Birth Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mozart</td>
<td>Austria</td>
<td>1756</td>
</tr>
<tr>
<td>Liszt</td>
<td>Czech R.</td>
<td>1811</td>
</tr>
<tr>
<td>Mahler</td>
<td>England</td>
<td>1860</td>
</tr>
<tr>
<td>Stevens</td>
<td>missing</td>
<td>1948</td>
</tr>
</tbody>
</table>

Source: data from Wikipedia.
## Alignment

<table>
<thead>
<tr>
<th>Composer</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mozart</td>
<td>Austria</td>
</tr>
<tr>
<td>Mahler</td>
<td>Czech R.</td>
</tr>
<tr>
<td>Stevens</td>
<td>England</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Composer</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mozart</td>
<td>1756</td>
</tr>
<tr>
<td>Liszt</td>
<td>1811</td>
</tr>
<tr>
<td>Mahler</td>
<td>1860</td>
</tr>
<tr>
<td>Stevens</td>
<td>1948</td>
</tr>
</tbody>
</table>
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 Czech R.
Mahler England
Stevens missing
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  England
Mahler  Austria
Stevens  Czech R.

Mozart  Austria
Liszt  1811
Mahler  1860
Stevens  1948

Mozart  England
Liszt  Austria
Mahler  <<loop>>
Stevens  Austria
Strange, but true

Alignment

Mozart 1756
Stevens 1948

Mahler 1860
Stevens 1948

Mozart 1756
Mahler 1860
Stevens 1948

What Changed

insert Mahler

correct typo

replace a composer

name change

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