Chapter 5

Conclusion

Our work has identified several areas for improvement in the foundations of existing bidirectional transformation tools. The development of symmetric lenses enables both repositories associated with a lens to store locally interesting information. Considering stateful, rather than pure, transformations enables the lens to store this local information on the side; though a theory of behavioral equivalence is needed to restore equational reasoning in the presence of this state. This is an improvement on the asymmetric lens framework's need for a canonical, centralized repository, but the behavior of many of the actual lenses proposed in Chapter 2 remain undesirable in several ways. Most significantly, they employ a somewhat naïve strategy for divining the connection between original and updated repositories. This manifests itself as lenses which follow the letter of the law—produce synchronized repositories—but mangle the meaning of the repositories during synchronization by inappropriately mixing and matching data.

Edit lenses address this problem by elevating the status of the data describing how original and updated repositories are connected. Treating edits as first-class means that the lenses need not guess about alignment information, and consequently will mangle the meaning of repositories less often. Indeed, as a side effect, promoting edits in this way allows the creation of stronger laws, so that the letter of the law and the spirit of the law are no longer quite so far from each other. We have also shown that one may realize performance benefits with this approach with careful design of the edit languages being manipulated by edit lenses.

Related approaches to this problem are summarized in Table 5.1; the edit lens framework is the first approach to address all of the issues raised above. Asymmetric delta lenses seem like a promising approach for their ability to handle alignment well, but the formalism does not accomodate small edits well even if syntax could be designed that could operate on them. Symmetric delta lenses extend these to the symmetric setting, but the reasoning principles that require behavioral equivalence discussed in conjunction with our symmetric lenses are not considered, and the framework itself is not yet instantiated with a syntax. The algebraic study of asymmetric lenses exposes many surprising features of edits, but does not address many of the

	Alignment	Symmetry	Performance	Syntax
symm. state	very bad	yes; requires equivalence	no	mostly domain agnostic
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
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 $diag \in X \leftrightarrow X \times X$
const. maint.	uninterpreted edits	yes; does not require equiv.	no; all edits relative to *init*	many primitives, but no composition

Table 5.1: Feature coverage for various alternatives to edit lenses

issues needed to create a practical system. Matching lenses and annotation-based lenses are very natural extensions of asymmetric, state-based lenses, but are also conservative in many ways. The more radical changes proposed in edit lenses allow for symmetric operation and eliminate the need to pass repositories to the lenses. Notably, annotation-based lenses extend a variant of the asymmetric, state-based lenses that allows for the construction of a lens that duplicates information, making it the only approach that takes a serious stab at alignment handling while enabling this lens. Finally, constraint maintainers tackle many practical issues, but do not fully explore the power of edits and lack the ability to perform sequential composition, a key piece of syntax.

Though the issues that edit lenses handle are important ones, there remain many other foundational issues that one might want to tackle to strengthen the practicality of bidirectional transformation frameworks. The following section surveys a few of the most pressing needs.

5.1 Future Work

Hyperlenses The lens framework focuses itself on the problem of synchronizing two repositories at a time. Consequently, current lenses do not generalize smoothly to more than two pieces of data, but many real-world scenarios involve synchronizing many (potentially quite small and loosely related) repositories. One example (which we are not the first to propose [28]) would be a multi-directional spreadsheet, where we treat each cell as a repository. Some cells are computed from others; these computations are the transformations that one might like to bidirectionalize. There seem to be a variety of additional challenges associated with generalizing from bi-directional to many-directional updating, chief among them being a significantly larger update space to search through on each synchronization action. We have explored a few restricted settings—for example, where no repository is connected to another in two different ways, or where all repositories are numbers and all connections are linear functions—that seem to admit partial solutions, but none are really satisfactory. There seem to be deep connections to the literature on constraint propagation and (in the special case of spreadsheets) computer algebra systems.

Additional Syntax We have noted in our discussions above several places where one could wish for a more expressive collection of lens constructions. In particular, we were not able to recapitulate the symmetric lens' development of fold and unfold lens combinators for recursive types, in part because it is not clear how to build edit modules for recursive types in a compositional way. (Edits to "roll" and "unroll" one layer of the recursion are not enough: as with lists, one wants a way to shuffle data between depths and across pairs and sums; this is the part that seems tricky.) Even restricting our view to containers, it would be interesting to investigate edit modules for more container shapes, especially graphs—the basis of the data model usually used in model-driven development—and relations—the data model usually used in databases. More speculatively, it is well-known that symmetric monoidal categories are closely connected to wiring diagrams [39] and to first-order linear lambda calculus [38]. Perhaps one could exploit this correspondence to design a lambda-calculus-like syntax or diagrammatic language for symmetric or edit lenses. The linear lambda calculus has judgments of the form $x_1{:}A_1, \ldots, x_n{:}A_n \vdash t : A_0$, where $A_0, \ldots, A_n$ are sets or possibly syntactic type expressions and where t is a linear term made up from basic lenses, lens combinators, and the variables $x_1, \ldots, x_n$. This could be taken as denoting a symmetric lens $A_1 \otimes \cdots \otimes A_n \leftrightarrow A_0$. For example, here is such a term for the lens *concat′* from §2.6.2:

$$
\begin{aligned}
z{:}\mathit{Unit} \oplus A \otimes A^\star \otimes A^\star \vdash\ & \mathit{match}\ z\ \mathit{with} \\
& \mid \mathsf{inl}\ () \mapsto \mathit{term}^{op}_{\langle\rangle} \\
& \mid \mathsf{inr}\ (a, al, ar) \mapsto \mathit{concat}(a{:}al, ar)
\end{aligned}
$$

The interpretation of such a term in the category of lenses then takes care of the appropriate insertion of bijective lenses for regrouping and swapping tensor products.

Algebraic Properties A number of algebraic oddities have cropped up during our development which it would be nice, as a matter of polish, to settle one way or another. The status of sums (and in particular injection lenses) has been somewhat in question for some time, so it is nice that in the symmetric lens case we have settled this question by elucidating the symmetric monoidal structure available. On the other hand, this makes the lack of associativity for tensor sum in the edit lens category all the more surprising; it may be interesting to pursue (or prove impossible) an associative tensor sum. Similarly, we have shown that our tensor product has many of the properties we expect of a symmetric monoidal category structure, but not quite all. We conjecture that adding appropriate monoid laws would resolve this problem; and anyway adding appropriate monoid laws in all the modules discussed is a serious undertaking of its own worth consideration. Finally, it has been suggested by a reviewer of one of our papers that, although we cannot have a categorical product or sum in the category of lenses, it may be worth considering placing a partial order on lenses. Perhaps this would enable a variant of the categorical product where the usual defining equations of a product are replaced by inequalities.

Implementation Effort spent on an implementation could be aimed at a variety of purposes. One might be interested in verifying whether the performance promises of edit lenses could be realized by running experiments. For example, one might imagine comparing the size of typical repository edits to the size of the repositories as the repository grows; comparing the runtime of edit translation to the runtime of a more traditionally designed lens; or analyzing any of half a dozen other metrics. Alternately, one could focus on breadth rather than depth, implementing a variety of transformations, to find out whether the syntax developed here is expressive enough. There are also many practical tools that may benefit from edit lenses: file synchronizers keeping two file systems synchronized, text editors keeping parse trees synchronized, database backends keeping queries synchronized with data, log summarizers keeping summaries and log files synchronized, software model transformations keeping architectural diagrams and code synchronized, perhaps even mobile phone applications keeping a client's display and server data synchronized may all benefit from bidirectional techniques.

Miscellaneous Extensions Besides the broad categories discussed above, there are a handful of other curiosities suggested throughout the development which we gather here. During the development of iterator symmetric lenses, it was observed that correctness of the lens depends on the existence of an appropriate weight function guaranteeing termination. We anticipate that this function will be simple in the majority of cases; automatically discovering it for a broad class of lenses seems plausible

and would remove a significant annoyance from the lens programmer. Next, in the passage from asymmetric, state-based lenses to symmetric lenses, we gave a theorem connecting various asymmetric lens constructions to symmetric lens constructions. It would be nice to know whether this connection could be extended to edit lenses. For example, do edit lens constructions like tensor product correspond in some way to the lifting of symmetric lens tensor product? One similarly wonders whether there is an edit lens analog of the theorem showing how to split a symmetric lens into two asymmetric lenses. On a slightly different line of inquiry, one may wonder just how canonical our choices of edit structure and lens laws are. Several related lines of work have proposed more intricate structures—for example, a popular choice is to require that edits have some way of being undone, either with an inverse or something weaker—and stronger laws—for example, requiring that $\Rightarrow$ and $\Leftarrow$ produce minimal edits by adding a triple-trip law—without instantiating their frameworks. Perhaps enforcing stronger requirements would suggest ways to improve the behavior of our existing constructions, and on the other hand perhaps our constructions would reveal that some of the requirements are too strong.

Another idea suggested by related work is to consider typed edits. In our development of edit lenses, we allow partial edits so that we may represent edits that work on some, but not all, repositories in a uniform way. We then go to great lengths to assure that the partiality is purely formal: we have a theorem showing that lenses never introduce partiality where none was before. On the other hand, the symmetric delta lens approach has no such problem—each edit is applicable to one and only one repository—but pays the price of having to duplicate the modifications which can apply to many repositories [13]. Perhaps it would be possible to find a middle ground by designing a typed edit language, in which edits may apply to many repositories (that share a type), but where the types are specific enough that all edits are total. For example, for list edits, one might consider having one type for each possible length of list. Then one would have, for example, deletion edges $\mathsf{del} : m \to n$ when $m < n$; such an edge must store marginally more information than our edit module did (the domain and codomain length rather than a single number telling their difference), but the set of repositories to which it applies is much more clearly delimited.

Finally, many asymmetric lens frameworks ground their constructions by showing how to use them with a string-based data model. This significant task was not undertaken here.[1] Preliminary efforts suggest that this is an especially difficult task in the world of edit lenses: though we have several decades of experience parsing strings into more structured data, there is comparatively little effort expended on parsing string edits into edits on more structured data.

[1]The use of "significant" in this sentence is true in both the "important" and "difficult" senses.

5.2 Closing Thought

Though there are many opportunities for further improvements, edit lenses are part of a growing ecosystem of bidirectional techniques. In this arena, our development expands what is known about incrementalizing and symmetrizing bidirectional transformations—an important step in the development of practical tools for the common task of maintaining replicated data.