# Chapter 5

# Conclusion

## 5.1 Future Work

1. hyperlenses
2. more syntax
   - edit modules for recursive types, fold/unfold lens combinators, investigate shape alignment
   - edit modules for more container shapes, esp. graphs (for model-driven development) and relations (for databases)
3. algebraic properties
   - associative sum
   - symmetric monoidal structure of tensor product edit lens
   - monoid laws for the various edit monoids
   - partially-ordered homsets: perhaps we can't get a product or sum in our category, but could get a variant where the required equations are replaced by inequalities
4. wiring diagrams; some text copied here from the original complements paper:

   More speculatively, it is a well-known folklore result that symmetric monoidal categories are in 1-1 correspondence with wiring diagrams and with first-order linear lambda calculus. We would like to exploit this correspondence to design a lambda-calculus-like syntax for symmetric lenses and perhaps also a diagrammatic language. 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.

5. typed edits/complements; some text moved here from the related work section:

   One can view the two approaches as two extremes, with on one end graphs with a single node representing all possible repository states and on the other end graphs with many nodes where each node represents a single repository state. There may be a middle ground in which graph nodes each represent many possible repository states; the hope then would be that one could keep the benefit of a total edit application function while reusing single edits on many different states. For example, for list edits, one might consider a graph with one node 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. Attempting to recast the edit modules and lenses proposed above in this light would be an interesting area for future work.

6. parsing/unparsing string edits to structured edits

7. automatically discover weight functions for symmetric lens iteration

8. laws about undo-able edits, and requiring that *edits* "roundtrip"

9. how closely do symmetric lens and edit lens constructions like tensor product and sum correspond? (can we get a theorem like the one saying how asymmetric lens constructions lift to symmetric lens constructions?)

10. how closely do (symmetric) edit lenses and asymmetric delta lenses correspond? (can we get a theorem like the one saying how to factor a symmetric lens into two asymmetric ones?)

11. implementation