Chapter 7

Conclusion

We have shown that an interconnection between internalism and externalism (**??**) can be established in the form of conversion isomorphisms between inductive families and their less informative variants paired with suitable predicates. Using ornaments (**??**), two datatype-generic ways of computing components in conversion isomorphisms are proposed:

- analytically, given two variants of inductive families related by an ornament, we can compute the optimised predicate (**??**) that captures the residual part of the more informative variant relative to the less informative one;
- synthetically, given a relational fold (**??**) — which is a predicate — on an inductive family, we can compute the algebraic ornamented version of the inductive family (**??**), into which proofs about the relational fold are embedded.

Actual conversion isomorphisms are computed by the refinement semantics of ornaments (**??**), which completes the analytic direction; the conversion isomorphisms for the synthetic direction are indirectly computed through the refinement semantics and predicate swapping (**??**). The analysis and synthesis of inductive families then respectively lead to two applications in internalist programming, both supported by several examples (Sections **??** and **??**):

- By computing the optimised predicates, we can consider their pointwise conjunction, which corresponds to the composite inductive family computed by parallel composition (**??**). With the help of the mechanism of refinements (**??**)

and upgrades (**??**), we are then able to synthesise composite internalist datatypes and operations from externalist library modules.

- Starting from a relational specification, which involves relational folds or can be transformed to one involving relational folds, we translate the specification into an internalist type involving algebraic ornamented datatypes for type-directed programming. Any program inhabiting this internalist type can be analysed to yield proofs about relational folds for discharging the specification.

In respective applications, analysis and synthesis are followed by synthesis and analysis. Hence both applications essentially rely on change of representation: internalist datatypes and operations are switched to their externalist representations for modular composition, and externalist relational specifications are converted to internalist types to be discharged by type-directed programming — both importing benefit from the other side by changing representation back and forth.

Theoretically, both parallel composition and algebraic ornamentation receive further characterisation by category-theoretic notions, the former as a pullback (**??**) and the latter as part of a categorical equivalence (**??**). Consequently:

- parallel composition is pinned down extensionally up to isomorphism, and from its pullback properties we can construct the ornamental conversion isomorphisms and the modularity isomorphisms (**??**), abstracting away from encoding detail of the universes;
- the categorical equivalence in which algebraic ornamentation takes part establishes a correspondence between ornamental and relational algebraic universal constructions, like parallel composition and the banana-split algebra (**??**); it also reveals the possibility of reasoning about datatypes in terms of relational algebras (**??**).

The whole development of this dissertation is a demonstration of the need for internalism and externalism to coexist: The datatype of ornaments is indexed by descriptions such that every typechecked ornament is a valid one between the descriptions in its type; various subsequent constructions (e.g., parallel composition) then manipulate valid ornaments only, without having

to deal with nonsensical cases. Categorical properties, on the other hand, can hardly be encoded into the types of these constructions, and hence need to be proved separately. Formal reasoning about complicatedly typed terms in Agda is exceedingly difficult, however — the actual proof terms are skipped over throughout the presentation since most of them are incomprehensible if not boring. Given that the formalisation of **??** already demonstrates the possibility, complete formalisation of **??** is deemed unfruitful since it can further demonstrate nothing but perseverance that should actually be avoided — instead, better ways to manage such reasoning are needed.

Putting formalisation aside, the theoretical development in Chapters **??** and **??** — while achieving the goals successfully — is admittedly unpolished. For example, the apparent similarity among the indexing structures of ℂOrn, 𝔽Fref, 𝔽Fam, and 𝔽FHTrans calls for a more systematic treatment by fibred category theory [Jacobs, 1999]. Practically, while Agda already provides adequate syntax and we have invoked the yet-to-exist elaboration mechanism from higher-level presentations of datatypes and functions to their encodings throughout the dissertation, it is still desirable to have more notational innovation and even some degree of automation. An excellent example is McBride's treatment of ordered data structures [2014] using Agda's instance arguments [Devriese and Piessens, 2011] as a lightweight proof searching mechanism to pass around ordering proofs silently — a technique that can dramatically improve the appearance of, e.g., the *merge* function on leftist heaps shown in **??**.

We conclude this dissertation with a final remark on internalism. The most important technique used in this dissertation is undoubtedly type computation, ranging from simpler ones like upgrades and predicate swaps to more sophisticated ones involving universes like various ornamental constructions. With internalism, this seems to be a natural tendency: as types start to play a more significant role of specifications (as opposed to the traditional, minor role that ensures only basic sanity) and have direct influence on program structure, their design requires more attention and mechanisation to the extent that it has really become programming. Thus it probably should not come as a surprise that relational program derivation is employed in this dissertation for type derivation. Stretching the imagination, perhaps internalism will eventually

lead to a unification of types and programs, yielding a uniform and scalable system for program correctness by construction that consists of only levels of programs, one level acting as specifications of the next, such that the idea of "levels of abstraction" (see, e.g., Dijkstra [1972]) can be formally captured, and programming techniques can be uniformly applied to all levels.

Bibliography

Dominique Devriese and Frank Piessens [2011]. On the bright side of type classes: Instance arguments in Agda. In *International Conference on Functional Programming*, ICFP'11, pages 143–155. ACM. doi: 10.1145/2034773.2034796. ↰ page 3

Edsger W. Dijkstra [1972]. Notes on structured programming. In *Structured Programming*, pages 1–82. Academic Press. ISBN: 978-0122005503. ↰ page 4

Bart Jacobs [1999]. *Categorical Logic and Type Theory*. Elsevier B.V. ISBN: 978-0444508539. ↰ page 3

Conor McBride [2014]. How to keep your neighbours in order. URL: https://personal.cis.strath.ac.uk/conor.mcbride/Pivotal.pdf. ↰ page 3

Todo list