Context

Computing is full of situations where transforming a data format into a different format is essential to “bridge the gap” between technology layers and ensure sharing of information among software applications. Changes in data types call for corresponding changes in data values. Moreover, to yield model interoperability, bidirectional transformations capable of translating between models are required.

A formal approach to this problem is provided by two-level transformations: bidirectional transformations based on calculational data refinement \[3\]. A type-level transformation from an abstract type \(A\) to a concrete type \(B\) is coupled with a pair of value-level functions \(\text{to (representation)}\) and \(\text{from (abstraction)}\), that allow us to migrate values of type \(A\) into values of type \(B\), and vice-versa.

The 2LT framework \[1\] addresses the task of levelling these concepts into a practical Haskell implementation, that provides a type-safe rewrite system for two-level transformations. Based on generic functional programming and strategic term rewriting, the framework is compositional and provides transformation rules from simple type transformers to complete strategies. It also provides front-ends to common data programming paradigms like XML or SQL.

Another well-known formalization of bidirectional transformation is given by Pierce et al \[2\]: using the so-called lenses they provide an elegant solution to the view-update problem from relational-database theory. A lens consists of a type transformation coupled with an abstraction relation \(\text{get}\) and a stateful representation relation \(\text{putback}\), that pushes data in the abstract view back into the original concrete model. Lenses where also used to implement a data synchronization tool called Harmony.

Objectives

- Currently, the 2LT framework only supports non-recursive types. Although the theory of refinement for single-recursive types is well known, it is not clear how to implement it in the 2LT kernel due to limitations of the Haskell type system that supports it. As such, the first objective of this project is to investigate how to accommodate recursive types in the
2LT framework. On the theoretical level, it is also necessary to investigate how to extend the refinement theory to mutually recursive types, in order to tackle realistic examples.

- Oliveira already showed that a relational point-free calculus can be a more natural setting to formalize bidirectional transformations [3]. However, since it is not clear how this calculus can be effectively mechanized, the 2LT kernel is currently based solely on its functional fragment. The second objective of this project is thus to investigate how such mechanization can be achieved, and to investigate how the additional power of the relational calculus can be used to enhance the current functionalities of the 2LT framework. In particular, we want to investigate how coreflexives can be used to effectively support invariants on data types.

- The third objective is to investigate how can lenses be accommodated in the 2LT framework. For this, it is necessary, at the theoretical level, to investigate the formal connection between lenses and refinements, and how to reason generically about lenses using the same point-free calculus that is already used to reason about refinements. On a more practical side, it is necessary to study the interplay between lenses and strategic combinators, and to find application examples where such combination can be useful.

- The last objective is to extend the range of applications to which the 2LT framework can be applied, namely in the context of Model Driven Software Engineering. Stevens already argued that bidirectional transformations can be used to automate model-driven development and guarantee model synchronization [4]. However, a bidirectional model transformation must be expressed as a relation $R$ between a pair of models $M \times N$. It is stateful in both directions and thus more general than both refinements and lenses.

References


