Chapter 6

Categorical equivalence of ornaments and relational algebras

Consider the AlgList datatype in **??** again. The way it is refined relative to the plain list datatype looks canonical, in the sense that any variation of the list datatype can be programmed as a special case of AlgList: we can choose whatever index set we want by setting the carrier of the algebra R; and by carefully programming R, we can insert fields into the list datatype that add more information or put restriction on fields and indices. For example, if we want some new information in the nil case, we can program R such that $R\ (nil - tag\ ,\ \blacksquare)\ x$ contains a field requesting that information; if, in the cons case, we need the targeted index x, the head element a, and the index x' of the recursive position to be related in some way, we can program R such that $R\ (cons - tag\ ,\ a\ ,\ x')\ x$ expresses that relationship.

The above observation leads to the following general theorem: Let O : Orn $e\ D\ E$ be an ornament from D : Desc I to E : Desc J. There is a classifying algebra for O

$$clsAlg\ O\ :\ \mathbb{F}\ D\ (\mathsf{InvImage}\ e) \rightsquigarrow \mathsf{InvImage}\ e$$

such that there are isomorphisms

$$\mu\ \lfloor algOrn\ D\ (clsAlg\ O) \rfloor\ (e\ j\ ,\ \mathsf{ok}\ j)\ \cong\ \mu\ E\ j$$

for all $j : J$. That is, the algebraic ornamentation of D using the classifying algebra derived from O produces a datatype isomorphic to $\mu\ E$, so intuitively the algebraic ornament has the same content as O. We may interpret this theorem as saying that algebraic ornaments are "complete" for the ornament language: any relationship between datatypes that can be described by an ornament can be described up to isomorphism by an algebraic ornament.

The completeness theorem brings up a nice algebraic intuition about inductive families. Consider the ornament from lists to vectors, for example. This ornament specifies that the type List A is refined by the collection of types Vec A n for all n : Nat. A list, say $a :: b :: [] :$ List A, can be reconstructed as a vector by starting in the type Vec A zero as $[]$, jumping to the next type Vec A (suc zero) as $b :: []$, and finally landing in Vec A (suc (suc zero)) as $a :: b :: []$. The list is thus classified as having length 2, as computed by the fold function *length*, and the resulting vector is a fused representation of the list and the classification proof. In the case of vectors, this classification is total and deterministic: every list is classified under one and only one index. But in general, classifications can be partial and nondeterministic. For example, promoting a list to an ordered list is classifying the list under an index that is a lower bound of the list. The classification process checks at each jump whether the list is still ordered; this check can fail, so an unordered list would "disappear" midway through the classification. Also there can be more than one lower bound for an ordered list, so the list can end up being classified under any one of them. Algebraic ornamentation in its original functional form can only capture part of this intuition about classification, namely those classifications that are total and deterministic. By generalising algebraic ornamentation to accept relational algebras, bringing in partiality and nondeterminacy, this idea about classification is captured in its entirety — a classification is just a relational fold computing the index that classifies an element. All ornaments specify classifications, and thus can be transformed into algebraic ornaments.

For more examples, let us first look at the classifying algebra for the ornament from natural numbers to lists. The base functor for natural numbers is

$$\mathbb{F}\ \mathit{NatD} : (\top \to \mathsf{Set}) \to (\top \to \mathsf{Set})$$
$$\mathbb{F}\ \mathit{NatD}\ X\ \_ = \Sigma\ \mathit{LTag}\ (\lambda\ \{\mathit{nil} - \mathit{tag} \to \top;\ \mathit{cons} - \mathit{tag} \to X\ \blacksquare\})$$

And the classifying algebra for the ornament *NatD-ListD A* is essentially

$$\mathit{clsAlg}\ (\mathit{NatD\text{-}ListD}\ A) : \mathbb{F}\ \mathit{NatD}\ (\mathsf{InvImage}\ !) \rightsquigarrow \mathsf{InvImage}\ !$$
$$\mathit{clsAlg}\ (\mathit{NatD\text{-}ListD}\ A)\ (\mathit{nil} - \mathit{tag}\ \ ,\ \_\ \ )\ (\mathsf{ok}\ \blacksquare) = \top$$
$$\mathit{clsAlg}\ (\mathit{NatD\text{-}ListD}\ A)\ (\mathit{cons} - \mathit{tag}\ ,\ \mathsf{ok}\ t)\ (\mathsf{ok}\ \blacksquare) = A \times (t \equiv \blacksquare)$$

The result of folding a natural number n with this algebra is uninteresting, as it can only be $\mathsf{ok}\ \blacksquare$. The fold, however, requires an element of A for each successor node it encounters, so a proof that n goes through the fold consists of n elements of A. Another example is the ornament $\mathit{OL} = \lceil \mathit{OrdListOD}\ A\ \_ \leqslant A\_ \rceil$ from lists to ordered lists, whose classifying algebra is essentially

$$\mathit{clsAlg}\ \mathit{OL} : \mathbb{F}\ (\mathit{ListD}\ A)\ (\mathsf{InvImage}\ !) \rightsquigarrow \mathsf{InvImage}\ !$$
$$\mathit{clsAlg}\ \mathit{OL}\ (\mathit{nil} - \mathit{tag}\ \ ,\ \_\ \ \ \ \ \ )\ (\mathsf{ok}\ b) = \top$$
$$\mathit{clsAlg}\ \mathit{OL}\ (\mathit{cons} - \mathit{tag}\ ,\ a\ ,\ \mathsf{ok}\ b')\ (\mathsf{ok}\ b) = (b \leqslant A\ a) \times (b' \equiv a)$$

In the nil case, the empty list can be mapped to any $\mathsf{ok}\ b$ because any $b : A$ is a lower bound of the empty list; in the cons case, where $a : A$ is the head and $\mathsf{ok}\ b'$ is a result of classifying the tail, i.e., $b' : A$ is a lower bound of the tail, the list can be mapped to $\mathsf{ok}\ b$ if $b : A$ is a lower bound of a and a is exactly b'.

Perhaps the most important consequence of the completeness theorem (in its present form) is that it provides a new perspective on the expressive power of ornaments and inductive families. We showed in a previous paper **?** that every ornament induces a promotion predicate and a corresponding family of isomorphisms. But one question was untouched: can we determine (independently from ornaments) the range of predicates induced by ornaments? An answer to this question would tell us something about the expressive power of ornaments, and also about the expressive power of inductive families in general, since the inductive families we use are usually ornamentations of simpler algebraic datatypes from traditional functional programming. The completeness theorem offers such an answer: ornament-induced promotion predicates are exactly those expressible as relational folds (up to pointwise isomorphism). In other words, a predicate can be baked into a datatype by ornamentation if

and only if it can be thought of as a nondeterministic classification of the elements of the datatype with a relational fold. This is more a guideline than a precise criterion, though, as the closest work about characterisation of the expressive power of folds discusses only functional folds **?** (however, we believe that those results generalise to relations too). But this does encourage us to think about ornamentation computationally and to design new datatypes with relational algebraic methods. We illustrate this point with a solution to the minimum coin change problem in the next section.

6.1 Ornaments and horizontal transformations

6.2 Ornaments and relational algebras

6.3 Consequences

6.3.1 Parallel composition and the banana-split law

algebras corresponding to singleton ornaments and ornaments for optimised predicates

6.3.2 Ornamental algebraic ornaments

6.4 Discussion

bad computational behaviour; ornaments for optimised representation; compare the **?** version (compatible with the two-constructor universe) and the **?** version of algebraic ornamentation in terms of "quality" (amount of σ's used); proof-relevant Algebra of Programming (e.g., *fun-preserves-fold*)

Todo list

algebras corresponding to singleton ornaments and ornaments for optimised predicates 4

bad computational behaviour; ornaments for optimised representation; compare the **?** version (compatible with the two-constructor universe) and the **?** version of algebraic ornamentation in terms of "quality" (amount of σ's used); proof-relevant Algebra of Programming (e.g., *fun-preserves-fold*; linking to the next chapter) 4