A Categorical Approach
to the Theory of Lists

Mike Spivey
Programming Research Group
Oxford University Computing Laboratory
11, Keble Road, Oxford, OX1 3QD, England

Abstract
Many of the laws in Bird's 'theory of lists' [1, 2] are precisely the conditions for various constructions to be functors, natural transformations, adjunctions, and so on. In this paper, I explore this categorical background to the theory, and - by generalizing one law and adding another - establish a completeness result for part of the theory. In the final section of the paper, I indicate how a theory of expression trees could be compiled along similar lines.

None of the mathematical results in this paper are new; instead, its contribution is in showing how category theory can be used to organize a complete set of laws for program transformation. I hope, too, that the paper will provide a readable introduction to category theory for those already acquainted with functional programming.

1 Types and functions
Functional programming is concerned with functions from one type to another. Although there are various ways of understanding in full generality what these functions and types are, it will be enough for us to consider the types to be certain sets, and the functions to be ordinary mathematical functions between these sets.

For each type $\alpha$, there is an identity function $id_\alpha : \alpha \rightarrow \alpha$. It will suit our purposes to make explicit in the name $id_\alpha$ the dependence on the type $\alpha$; we are considering a family of functions indexed by a type, rather than a single 'polymorphic' identity function.

If $f : \alpha \rightarrow \beta$ and $g : \beta \rightarrow \gamma$ are functions, we can form their composition $g \cdot f$, a function in $\alpha \rightarrow \gamma$. Familiar laws of mathematics state that composition is associative:

$$(h \cdot g) \cdot f = h \cdot (g \cdot f),$$

and that the identify functions $id_\alpha$ and $id_\beta$ are right and left identities for composition with $f : \alpha \rightarrow \beta$:

$$f \cdot id_\alpha = f = id_\beta \cdot f.$$ 

In category theory, these properties of $id$ and $\cdot$ are summarized by saying that types and functions form a category.
2 Lists

If $\alpha$ is a type, we can form the type $\alpha^*$: its elements are lists of elements of $\alpha$. Also, if $f : \alpha \to \beta$ is a function, we can form the function $f^* : \alpha^* \to \beta^*$ which maps $f$ over the elements of its argument: it is the function *mapcar* of LISP.

The notation $\ast$ is used here for both the operation which takes a type $\alpha$ to the type of lists of elements of $\alpha$, and the operation which takes a function $f$ to the function that applies $f$ uniformly to each member of a list. This choice of notation may seem like an attack of the Computing Science disease — that of an unhealthy preoccupation with notational issues — but it is justified by the patterns it reveals in the laws of the theory of lists.

Two laws show how $\ast$ interacts with the identity function and composition:

$$id_{\alpha^*} = id_{\alpha^*},$$
$$ (g \cdot f)^* = g^* \cdot f^*. $$

So $\ast$ takes identity functions to identity functions and the composition of two functions to the result of composing their images under $\ast$. In categorical language, we say $\ast$ is a functor from types to types.

3 Flatten

Many families of functions can be defined uniformly for any type. These families of functions obey laws which follow a certain pattern.

For a first example, consider the "flatten" function $++/\alpha : \alpha^{**} \to \alpha^*$, defined for any type $\alpha$. This function takes a list of lists and forms a list by concatenating all the members of its argument. If $f : \alpha \to \beta$ is a function, then we can form the functions

$$f^* : \alpha^* \to \beta^*,$$
$$f^{**} : \alpha^{**} \to \beta^{**}. $$

The family of flatten functions also gives us the two functions

$$++/\alpha : \alpha^{**} \to \alpha^*,$$
$$++/\beta : \beta^{**} \to \beta^*, $$

and these four functions can be made into the following picture:

$$\begin{array}{c}
\alpha^{**} \xrightarrow{f^{**}} \beta^{**} \\
\downarrow^{++/\alpha} \quad \quad \downarrow^{++/\beta} \\
\alpha^* \xrightarrow{f^*} \beta^*
\end{array}$$

A law in the theory of lists states that

$$f^* \cdot ++/\alpha = ++/\beta \cdot f^{**}. $$