fmap (f . g) for free

Kirill Elagin

```haskell
class Functor f where
  fmap :: (a -> b) -> f a -> f b
```

---

1. fmap id ≡ id
2. fmap (f . g) ≡ fmap f . fmap g

Theorems for free!

Philip Wadler
University of Glasgow*

June 1989

Abstract

From the type of a polymorphic function we can derive a theorem that it satisfies. Every function of the same type satisfies the same theorem. This provides a free source of useful theorems, courtesy of Reynolds' abstraction theorem for the polymorphic lambda calculus.

1 Introduction

Write down the definition of a polymorphic function on a piece of paper. Tell me its type, but be careful not to let me see the function's definition. I will tell you a theorem that the function satisfies.

list of A yielding a list of A', and $r_A : A^* \to A^*$ is the instance of r at type A.

The intuitive explanation of this result is that r must work on lists of X for *any* type X. Since r is provided with no operations on values of type X, all it can do is rearrange such lists, independent of the values contained in them. Thus applying a to each element of a list and then rearranging yields the same result as rearranging and then applying a to each element.

For instance, r may be the function $reverse : \forall X.\ X^* \to X^*$ that reverses a list, and a may be the function $code : Char \to Int$ that converts a character to its ASCII code. Then we have

$$code^*\ (reverse_{Char}\ [\text{‘a’}, \text{‘b’}, \text{‘c’}])$$

1. fmap id ≡ id
2. some free theorems
3. ????
4. PROFIT