# Higher Kinded Embodiments

by Sven Nilsen, 2017

The use of embodiments in intentional paths makes it possible to reason about performance of various logically equivalent algorithms. An input intentional path of `f` with embodiment `g` is written:

$$f^g : A \to C$$

$$f : A \to B$$
$$g : (A, f(A)) \to C$$

When a higher order function returns a function, it makes sense that the returned function has a family of unique embodiments. This is because a concrete constructed function must necessary represent the function in some way, and the representation determines embodiments. The use of constructed functions is a key insight to proving embodiment uniqueness.

$$f(a)^g : f(A) \to D$$

$$f : A \to (B \to C)$$
$$g : (f(A), f(A)(B)) \to D$$

This means that embodiments are “inherited” from embodiments of higher order functions. Therefore, it makes sense to assign higher order embodiments to higher order functions.

To reason about higher order embodiments, a family of unique embodiments are assigned an “embodiment kind”. The kind is written using type notation combined with parentheses and a superscript star “$^*$”.

$$f(a)^* : (B \to D)^*$$
$$f^* : (A \to (B \to D)^*)^*$$

The general rule is that a higher embodiment kind determines lower embodiment kind.

In decision problems one often optimizes performance by reducing cost. To minimize cost one can find higher order functions of same higher kinded embodiments. An algorithm is locally optimal when constructed by the higher order function that produces the embodiment of logically equal functions with lowest cost.

For example, two different higher order functions constructs logically equivalent functions, but the length of one input intentional path is shorter than the other. This proves which optimization is better by some measurement of performance:

$$f_0(a_0) <=> f_1(a_1)$$
$$|f_0(a_0)^{g0}| < |f_1(a_1)^{g1}|$$

This gives a precise way of defining how decisions to optimize an algorithm are made.