Higher Order Operator Overloading

by Sven Nilsen, 2018

In this paper I represent a syntax for higher order operator overloading. This syntax is used to reason about problems where multiple functions operate on the same source of data.

Assume there are two functions:

$$f_0 : A \rightarrow T$$
$$f_1 : A \rightarrow T$$

Some operator `g` takes the outputs from `$f_0$` and `$f_1$` for the same input `a`:

$$g(f_0(a), f_1(a))$$

$$g : T \times T \rightarrow T$$

A higher order operator overloading is a function defined as following:

$$g(f_0, f_1) := \backslash(a : A) = g(f_0(a), f_1(a))$$

$$g(f_0, f_1) : A \rightarrow T$$

Since the types of `$f_0$` and `$f_1$` does not match the argument types of `g`, it is assumed that higher order operator overloading is used instead. This is useful when functions operate on the same source of data.

For example, one can have two utility functions that are composed into a new utility function:

$$\text{utility}_0 : \text{world} \rightarrow \text{real}$$
$$\text{utility}_1 : \text{world} \rightarrow \text{real}$$

$$\text{utility}_0 + \text{utility}_1 := \backslash(x : \text{real}) = \text{utility}_0(x) + \text{utility}_1(x)$$

One can also generalize this syntax to various mathematical loops:

$$\sum i \{ \text{utility}_i \} = \text{utility}_0 + \text{utility}_1 + \ldots + \text{utility}_{n-1}$$
$$\prod i \{ \text{utility}_i \} = \text{utility}_0 \cdot \text{utility}_1 \cdot \ldots \cdot \text{utility}_{n-1}$$

Since utility functions usually operate on the same “world” state, one can use higher order operator overloading to talk about the composition of utility functions without refering to the world directly.

It is common in mathematics to reason about functions instead of data, because proofs that holds for functions are valid for any data they operate on. Higher order operator overloading makes it possible to eliminate the need to specify a variable for the data and therefore more easily reason about the functions themselves.