Higher Order Operator Overloading and Self Reference

by Sven Nilsen, 2019

Higher Order Operator Overloading states that when you have functions of the same input, you can overload any operator on the return values of the function:

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

This makes it possible to treat functions as values you can compute with, constructing new programs.

Assume that you have the following function (the set of points on a unit circle):

$$circle := \backslash(x : real, y : real) = x^2 + y^2 <= 1$$

Here, there are two arguments, `x` and `y`.

To reference `x` and `y` one can do the following:

$$x := \backslash(x : real, y : real) = x$$

$$y := \backslash(x : real, y : real) = y$$

With Higher Order Operator Overloading, the following expression re-constructs the unit circle:

$$x^2 + y^2 <= 1 \quad <=> \quad \backslash(x : real, y : real) = x^2 + y^2 <= 1$$

What happens here is that `x` and `y` are defined as functions that refers to themselves in the context of the function argument tuple `(x : real, y : real)` that is used to construct new functions.

This technique is called “self reference” and states that:

- If we have a function for every input ...
- … and overload all operators with Higher Order Operator Overloading …
- … then the original function can be re-constructed by re-interpreting the original function

By passing `x` and `y` as self referential functions, the following is true:

$$circle(x, y) <=> circle$$

One can use this technique to prove the commutative property of the `circle` function in a new way:

$$circle(y, x) <=> circle \quad <=> \quad \forall\ x, y \{ circle(x, y) = circle(y, x) \}$$