Higher Order Operator Overloading and Notation for Parameters

by Sven Nilsen, 2019

In this paper I represent a notation for expressing the type of a function of something.

When working with Higher Order Operator Overloading[1] (HOOO) for theorem proving, it is common to use functions of something that returns a type:

f : \→bool `f` is a function of something that returns `bool`

e.g. lerp(a : \→real, b : \→real, t : \→real) = a + (b – a) · t

An alternative notation is to write:

f : \bool

The first version is preferred when one wants to be explicit and clear, but the second version might also be used for efficiency when drafting proofs. The second version is not expected to be understood by people who are not familiar with higher order reasoning in path semantics[2].

The second version might be confusingly similar to lambda notation with parentheses:

id := \(x) = x

f : \(bool, bool)

Here, `f` is a function that returns a tuple `(bool, bool)`, but it looks a bit like an unfinished lambda.

f : \→(bool, bool)

By using the first version instead, the expression gets clearer.

However, since HOOO holds for all operators, one could write proofs that uses this trick:

lerp(a : real, b : real, t : real) = a + (b – a) · t

By interpreting the type `real` with HOOO, the `lerp` function is generalized.
The problem is that it is hard to switch interpretation without training.

Putting `\` in front of a type is a quick way to remind oneself that HOOO is intended:

lerp(a : \real, b : \real, t : \real) = a + (b – a) · t

This is the motivation for the second version.

References:

[1] “Higher Order Operator Overloading”
Sven Nilsen, 2019
https://github.com/advancedresearch/path_semantics/blob/master/papers-wip/higher-order-operator-overloading.pdf

[2] “Path Semantics”
Sven Nilsen, 2016-2019
https://github.com/advancedresearch/path_semantics/blob/master/papers-wip/path-semantics.pdf