Higher Order Operator Overloading by Explicit Notation

by Sven Nilsen, 2021

In this paper I introduce explicit notation for Higher Order Operator Overloading. This explicit notation is particularly useful for complex expressions and Cartesian products.

Higher Order Operator Overloading[1] (HOOO) can be used when depending on the same argument:

```
h(f, g) := \(x) = h(f(x), g(x))

f : T → U
g : T → U
h : U × U → V
```

In some cases it is desired to make HOOO more explicit.

For this purpose, I introduce 3 different explicit notations (Prefix, infix, block):

`\: □(f, g)`	<=>	`\(x) = □(f(x), g(x)`	Prefix notation
`f \□: g`	<=>	`\(x) = f(x) □ g(x)`	Infix notation
`\: { f □ (g ◇ h) }`	<=>	`\(x) = f(x) □ (g(x) ◇ h(x))`	Block notation

The symbols `□` and `◇` stands for operators.

The prefix notation acts on the top level operator.
The infix notation acts on the contained operator.
The block notation acts on any infix operator.

Block notation is needed as nested infix notation results in more complex lambda expressions:

`f \□: (g \◇: h)`	<=>	`\(x) = f(x) □ (\(y) = g(y) ◇ h(y))(x)`

These lambda expressions can be normalised to block notation by functional equivalence[2]:

`f \□: (g \◇: h)`	<=>	`\: { f □ (g ◇ h) }`

The prefix notation is in particular useful for Cartesian products[3]:

`\: (f, g)`	<=>	`\(x) = (f(x), g(x))`

References:

[1] “Higher Order Operator Overloading”
Sven Nilsen, 2018
https://github.com/advancedresearch/path_semantics/blob/master/papers-wip/higher-order-operator-overloading.pdf

[2] “function extensionality”
nLab
https://ncatlab.org/nlab/show/function+extensionality

[3] “Cartesian product”
Wikipedia
https://en.wikipedia.org/wiki/Cartesian_product