Function Currying Notation

by Sven Nilsen, 2018

In functional programming, it is common to write the following:

∴ $f(a) : B \to C$

∵ $f : A \times B \to C$
∵ $a : A$

This is called "function currying"[1] and can be thought of as auto-constructing a function `f'`

∵ $f' := \backslash(a : A) = \backslash(b : B) = f(a, b)$

∴ $f(a) \Leftrightarrow f'(a)$

Path semantics uses functional currying a lot, because of sub-types[2]:

∵ $x : [f(a)]\ c$

∴ $x : B$

In addition to left-argument currying, it is common in path semantics to use a right-argument version:

∵ $x : [f\ b]\ c$

∴ $x : A$

When a right-argument version returns `bool`, one can use parentheses like this:

∴ $x : [g\ b]\ \mathrm{true} \quad \Leftrightarrow \quad x : (g\ b)$

∵ $g : A \times B \to \mathrm{bool}$

For example:

$x : (> 2)$
$x : (= 10)$

A more complex example:

∵ $(x, y) : (f\ g) \quad \Rightarrow \quad (x, y) : A \times A$

∴ $f : A \times A \to (A \to \mathrm{bool}) \to \mathrm{bool}$
∴ $g : A \to \mathrm{bool}$

References:

[1] “Currying”
Wikipedia
https://en.wikipedia.org/wiki/Currying

[2] “Sub-Types as Contextual Notation”
Sven Nilsen, 2018
https://github.com/advancedresearch/path_semantics/blob/master/papers-wip/sub-types-as-contextual-notation.pdf