Path Operators

by Sven Nilsen, 2019

A path operator is a function that associates an optional symbol for every symbol:

path_operator : symbol → opt[symbol]

In practice, a symbol is just a string, but it can also be thought of as a generic type:

symbol <=> string	When a symbol is a string
symbol <=> T	When a symbol is a generic type `T`

For example, the membership operator `:` associates `bool` for `true` and `false`:

`:`(“true”) := some(“bool”)	<=>	true : bool
`:`(“false”) := some(“bool”)	<=>	false : bool

In atomic path semantics, this is encoded the following way with atomic functions:

true(bool) = true
false(bool) = false

With other words, atomic path semantics implicitly constructs one or more path operators.

Another way to define a path operator, is as a dynamically typed object (using Dyon/Javascript syntax):

`:` := { true: “bool”, false: “bool” }

When the object contains some key, e.g. `true: “bool”`, the path operator returns `some(“bool”)`.
For all keys that the object does not contain, the path operator returns `none()`.

Here is another example with natural numbers:

`:` := { “z”: “nat”, “s(z)”: “nat”, “s(s(z))”: “nat”, … }

Path operators are useful because it lets us think about type-similar problems as constructing or talking about a specific object, without relying on the interpretation of inductively defined data structures.

An inductively defined data structure can be thought of as a grammar constraining the membership path operator such that its existential path returns `true` for the data type and only for that data type:

x : ∃`:`{inductive_data_structure(X)}   <=>   x : X

inductive_data_structure : T × symbol → bool

Rules for interpreting inductively defined data structures follows from semantics of path operators.