Domain Constraint Notation

by Sven Nilsen, 2017

A domain constraint turns a total function into a partial function. This path semantical notation is used to add support for reasoning about partial functions and relations between domain and co-domains. The notation is designed to work seamlessly with asymmetric path notation.

Here is a domain constraint of a single argument function:

$f\{T_A\}$

$f : A \to B$

Notice that the curly braces are written after the function, similar to when calling a function with arguments. The difference is that, instead of returning a value, the function is converted into a partial function.

For example, the following partial function:

f(a : [g] true) = { … }

$g : A \to bool$

Can be written as:

f{[g] true}(a) = { … }

[g] true : T_A

Domain constraints can be used as an intermediate step to transform a function definition with dependent sub-types into paths:

add(a : [even] x, b : [even] y) → [even] x == y { a + b }
add{[even] x, [even] y}(a, b) → [even] x == y { a + b }
add[even × even → even](x, y) = x == y
add[even](x, y) = x == y
add[even] <=> eq

Empty pair of curly braces creates a higher order function that takes a domain constraint for each input:

$f : A \to B$
$f\{\} : T_A \to A \to B$

$f : A \to B \to C$
$f\{\} : T_A \to T_B \to A \to B \to C$

Domain constraints follow a different application rule than normal variables, a bit similar to slot lambda calculus. If you pass a function that ends with `A → bool` to an argument of domain constraint type `$T_A$`, then the application rule behaves like a higher order function.

f{}(g)(b) <=> f{g}(b) <=> f{g(b)} <=> f{[g(b)] true}

f : A → C
g : B → A → bool
f{} : T_A → A → C
f{g} : B → A → C
f{g}(b) : A → C

The function `f{}` is called the universal of `f`.