Truth Values of Sub-Types

by Sven Nilsen, 2019

In this paper I show that it is always possible to eliminate the input variable and membership operator from expressions when talking about the truth value of sub-types.

In path semantics, it is common to write the following:

x : [f] a

This means `x` has a sub-type such that `f` returns `a`.

A such expression is consistent if and only if the following holds:

a : [∃f] true

The truth value is defined as following:

[f] a = (∃f)(a)

In first order logic, the same truth value is given by:

[f] a = ∃ x { f(x) = a }

With other words, `[f] a` is syntax sugar for a computation `(∃f)(a)` or statement in logic that requires no input variables and therefore does not require the membership operator. While it is natural to think that `x` in `x : [f] a` is the input and connected to the output `a` by `f`, it plays no significant role here. This is because the expression quantifies over all possible values such that the truth value is `true`.

The expression `x : [f] a` talks about a proof `x` such that the truth value can be computationally verified. This introduces extra complexity and cost because you choose some value among the possible ones. If you are willing to let go of this choice and ignore the identity of `x`, then the semantics of `x : [f] a` can be reduced to `[f] a`.

The truth value of a sub-type is `bool`:

[f] a : bool

Eliminating the input variable is important to understand overall sub-type semantics.
If `f` and `a` are input variables to proof talking about sub-types, one can consider the following:

∃ f, a { [f] a }

Since there exists some values of `f` and `a` such that this is true, this is a proof that it makes sense to talk about sub-types in general. This holds even if there is no reference to any input. As strange as it sounds, sub-types makes sense on their own, due to the expression `(∃f)(a)` does not require `x`.