Variables in MX Dependent Types

by Sven Nilsen, 2019

In this paper I go into depth about variables in MX grammar of dependent typed languages.

A variable/value in MX grammar[1] for dependent types[2] uses the following BNF[3]:

v ::= **s** | x

Here, `**s**` means a symbol or extended grammar, and `x` means a term.

A term can be a `$type_{nat}$`, a lambda or application:

x ::= $type_{nat}$ | \(m) = mx | mx(mx)

A membership associates a variable/value with another:

m ::= v : v

It is not possible to reduce the computational value of a term into a membership:

x =¬> m

Therefore, a variable/value can not be reduced into a membership:

v =¬> m

If `a` has a type that is computed from the context by application:

a : f(b)

If `f(b)` is a symbol, then `a` is simply typed or the type is defined from context.
If `f(b)` is `$type_0$`, then `a` is a “normal” type (e.g. `false : bool`) or nested type (e.g. `foo : true`).
If `f(b)` is `$type_{n+1}$`, then `a` is `$type_n$` or some variable/value `$a : type_n$`.
If `f(b)` is a lambda, then `a` is a lambda definition.
If `f(b)` is an application, then `f` is partially evaluated with a higher order `b`[4].

A dependent language might not distinguish between symbols of “variables” and “values”.

proven a member of a type by definition	<=>	value
lambda parameter	=>	variable
proven a variable of a type by definition	=>	variable

If a language does not distinguishes between variables and values, then:

proven a member of a type by definition	<=>	proven a variable of a type by definition

References:

[1] “MX Grammar for Dependent Types”
Sven Nilsen, 2019
https://github.com/advancedresearch/path_semantics/blob/master/papers-wip/mx-grammar-for-dependent-types.pdf

[2] “Dependent type”
Wikipedia
https://en.wikipedia.org/wiki/Dependent_type

[3] “Backus-Naur form”
Wikipedia
https://en.wikipedia.org/wiki/Backus%E2%80%93Naur_form

[4] “Higher Order Operator Overloading and Partial Evaluation”
Sven Nilsen, 2019
https://github.com/advancedresearch/path_semantics/blob/master/papers-wip/higher-order-operator-overloading-and-partial-evaluation.pdf