MX Grammar for Dependent Types

by Sven Nilsen, 2019

In this paper I introduce a grammar that enforces one-level type information for dependent types.

The MX grammar is defined by the following BNF[1]:

mx ::= m | x	Membership or term
m ::= v : v	Membership
x ::= $type_{nat}$ | \(m) = mx | mx(mx)	Term
v ::= **s** | x	Variable/Value

Here, `**s**` stands for symbols or extended grammar for data structures etc.
The motivation with this grammar is to make it easier to study dependent typed languages[2].

This grammar has the property that it enforces one-level type information, which means that each expression `mx` has a derivable type from context:

typeof : ctx → mx → res[mx]

Either the expression is a variable/value with an associated type, or the type is derived axiomatically or computationally from the context. In the case when the type is derived axiomatically or computationally from context, it is not possible to express an associated type. The one-level type information consists of information after non-trivial type inference, but before trivial type inference. Here, “trivial” means the minimum type inference required to classify as a dependent typed language.

For example, the generic `id` function must be defined the following way to satisfy MX:

id := \($type_0$) = \(x : T) = x : T

Notice that the output also has an assigned type.
The type of `id` is the following:

id : \($type_0$: $type_1$) = \(T : $type_0$) = T : $type_0$

The type of `id` has a type, and so on:

\($type_0$: $type_1$) = \(T : $type_0$) = T : $type_0$	:	\($type_1$: $type_2$) = $type_0$: $type_1$
\($type_1$: $type_2$) = $type_0$: $type_1$	:	$type_0$

Another example:

f := \(a : \($type_0$: $type_1$) = $type_0$: $type_1$) = \(b : $type_0$) = id(a(b))

a(b)			:	$type_0$
id(a(b))	=	\(T : a(b)) = T : a(b)	:	\($type_0$: $type_1$) → $type_0$: $type_1$

References:

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

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