Evaluation of Variable in MX Dependent Types

by Sven Nilsen, 2019

In this paper I represent evaluation of variable in MX grammar for dependent types.

The `v` stands for variable in the MX grammar for dependent types[1]:

v ::= **s** | x

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

An evaluation rule is an algorithm describing what the variable means when it is computed:

```
eval_var : ctx → v → (v, opt[v])

eval_var := \(ctx₀ : ctx) = \(v₀ : v) =
    if let some((v₁ : v₂)) = find_def_id(ctx₀)(v₀) { (v₁, some(v₂)) }
    else {match v₀ {
        s₀ : s => (s, none())
        x₀ : x => match eval_x(ctx₀)(x₀) {
            (v₃ : v₄) : m => (v₃, some(v₄))
            x₁ : x => (x₁, none())
        }
    }}
```

The return value is a tuple with the evaluated value and an optional type variable.
The evaluation rule for membership might use the optional type to check for correctness.
Membership evaluation is not treated in this paper.

A function for looking up the definition from context is required:

find_def_id : ctx → v → opt[mem]

The context might define substitutions not only from symbols, but also terms.

The implementation of the evaluation rule for variable assumes some evaluation rule for `mx`:

eval_x : ctx → x → mx

There are two styles of evaluation of a term `x`: One that returns the input if the expression can not be reduced, and another that reports an error in that case (returning `res[mx]` instead of `mx`). The style of evaluation is not very relevant for understanding the underlying semantics. Here, I assume the first style since it is simpler than the second.

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