Path semantics for dynamical types

by Sven Nilsen, 2016

Abstract:
In this paper I formalize the notation for path semantics in a dynamical type system. I explain how the notation gets away with no type operator, and why this does not result in ambiguity by following a set of rules.

Path semantics for dynamical types is perhaps the most convenient and practical dialect of path semantics.

Distinction between values and types

A term is the abstract type of the union of all values and all types.

> type(term) → term

All terms that are types are not values, and all terms that are values are not types.

> is_value [type] (term) → bool
> is_type [type] (term) → bool
>
> is_value([is_type] true) → false
> is_type([is_value] true) → false

Example:

> is_value(bool) = false
> is_type(bool) → true
> Proof:
>> is_value([is_type] is_type(bool)) = false
>> is_value([is_type] true) = false
>> false = false
>> Reflection.
>
> Qed.

The `type` function returns only terms that are types:

> type(_) → [is_type] true

The type `term` is a type:

> is_type(term) → true

So how to distinguish `type(term) → term` from `type(term) → [is_type] true`? It makes no sense to write:

> type [type] (term) → term

This leads to circular reasoning. Luckily, there is a way out.

The absence of a `[type]` path makes a statement is about values, unless all terms are types. It makes an escape clause for `term` while allowing `type` to be used in asymmetric cases.

If a term is not a concrete value or a type, then it is assumed to be a variable, which is a value unless `[type]` is expressed explicitly.