Rooted Binary Trees

by Sven Nilsen, 2019

In this paper I formalize rooted binary trees in path semantics.

A rooted binary tree is a binary tree where there exists a function `root`:

```
root : binary_tree → bool
root(a) = parent(a) == a
```

The `root` is a node which parent is itself.

The parent of a binary tree is defined as following:

```
parent : binary_tree → branch
parent(a) = has_parent := ∃ x { left(x) == a ∨ right(x) == a }
            if has_parent { why(has_parent) } else { a }
```

Here, the `why` function uses the secret-notation from the language Dyon. This can be thought of as extracting the `x` for which left or right child is the node.

An alternative definition using a loop which returns `opt[binary_tree]`:

```
parent : binary_tree → branch
parent(a) = parent := argany x { left(x) == a || right(x) == a }
            if let some(x) = parent { x } else { a }
```

Since `parent` constrained to roots is the identity function

$$\text{parent}\{\text{root}\} <=> \text{id}$$
$$\text{parent}\{\text{root}\} <=> \text{id}_T$$

What can be said about `T` is that it must be a `root`, since `id` returns the input, which is a `root`. However, `parent` returns only branches, so since `id` returns the input, the input must be a `branch`:

$$T <=> \text{root} \wedge \text{branch}$$

$$\text{id} : T \to T$$

This means that if a node is a root, is also a branch:

$$\text{root} => \text{branch}$$