Rooted Full Binary Trees

by Sven Nilsen, 2019

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

A rooted full binary tree is a full binary tree[1] where there exists a function `root`:

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

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

The parent of a full binary tree is defined as following:

```
parent : full_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[2]. This can be thought of as extracting the `x` for which left or right child is the node.

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

```
parent : full_binary_tree → branch
parent(a) = parent := arg_any x { left(x) == a ∨ right(x) == a }
            if let some(x) = parent { x } else { a }
```

Since `parent` constrained[3] to roots is the identity function:

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

$$\text{id} : T \to T$$
$$\text{id}_T := \backslash(x : T) = x$$

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`:

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

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

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

References:

[1] “Full Binary Trees”
Sven Nilsen, 2019
https://github.com/advancedresearch/path_semantics/blob/master/papers-wip/full-binary-trees.pdf

[2] “Dyon”
A rusty dynamically typed scripting language
https://github.com/pistondevelopers/dyon

[3] “Domain Constraint Notation”
Sven Nilsen, 2017
https://github.com/advancedresearch/path_semantics/blob/master/papers-wip/domain-constraint-notation.pdf