# Full Binary Trees

by Sven Nilsen, 2019

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

A full binary tree is a tree structure where every branch has two children, called “left” and “right”:

```
full_binary_tree : type

left : branch → full_binary_tree
right : branch → full_binary_tree
branch : full_binary_tree → bool
```

It follows that if `left` returns something, the input must be a branch:

```
∃ x { (∃left{y})(x) }          <=>          y : branch
```

Similarly, if `right` returns something, the input must be a branch:

```
∃ x { (∃right{y})(x) }          <=>          y : branch
```

Since `left` and `right` take branches, it means that they return something for all branches:

```
∀ y : branch { ∃ x { (∃left{y})(x) } ∧ ∃ x { (∃right{y})(x) } }
```

Notice that I have not added axioms, I just reason about what follows from the modest definition.

This gives a definition of branches.
However, the definition of branches this way can be unclear.
Even if it is correct, it might be counter-intuitive.
The definition used above might appear to define left or right children in terms of branches.
One would like to express what a branch is in terms of whether it has a left or right child.

To do this, I will take a different starting position and work backward toward the same result.

When either `left` or `right` returns something, one says that the full binary tree has a child:

```
has_child : full_binary_tree → bool
has_child(a) = ∃ x { left(a) == x ∨ right(a) == x }
```

From this one can derive that if a full binary is a branch, it has a child:

```
branch => has_child
```

However, this is not sufficient to prove that every branch has both a left and right child.

What one would like to show, is the following:

```
has_children : full_binary_tree → bool
has_children(a) = ∃ x { left(a) == x } ∧ ∃ x { right(a) == x }
```

A branch is logically equivalent to a full binary tree that has children:

```
branch <=> has_children
```

This means that if `left` returns something, the `right` function must return something for the same input. As a result, every branch has a left and a right sub-tree.

A leaf is a full binary tree that is not a branch:

```
leaf : full_binary_tree → bool
not · branch <=> leaf
```

Not all full binary trees contain leafs. An infinite and complete full binary tree contains only branches, which means that there is no nodes that can be called leafs.

One must be careful to not say too much about full binary trees in general, since there are properties which one can not prove or disprove from the definition.

For example, one can imagine a full binary tree where every node has a parent, which left or right children is itself. Usually, there is a root node which is kind of special: It might have a parent (itself), but its children are other nodes. However, if one allows the tree to be more than countable infinite in size, then reaching the root can be made impossible. Therefore, every *reachable* node has a parent, which left or right children is itself.

When one says that a full binary tree contains another, it means that either the full binary trees are equal, or if the first argument is a branch, the left or right children contains the second argument:

```
contains : full_binary_tree × binary_tree → bool
contains(a, b) = a == b ∨ if branch(a) { contains(left(a), b) ∨ contains(right(a), b) } else { false }
```