

Trees

At this point it is important to understand the two usages of the word *tree*.

We introduced a map between terms and trees, because this new point of view gives some insights (e.g. the \mathcal{H} function). In this context, a tree is another **representation** of the subject (terms) we are studying, it is a tool.

But this section now presents trees as a **data structure** on its own. In this context, a tree is the subject of our study, they are given. This is why, when studying the stacks (the subject), we displayed them as trees (an intuitive graphics representation).

Tree traversals

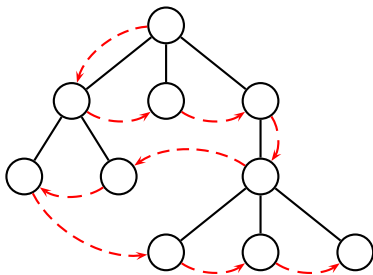
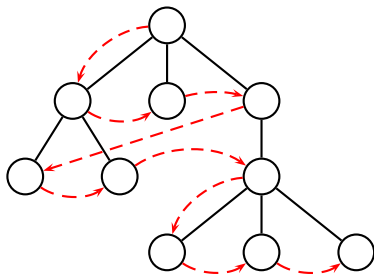
Given a tree, we can traverse it in many ways, but we must start from the root since we do not have any other node at this level. There are two main kind of traversals:

- **Breadth-first** traversal consists in walking the tree by increasing levels: first the root (level 0), the sons of the root (level 1), then the sons of the sons (level 2) etc.
- **Depth-first** traversal consists in walking the tree by reaching the leaves as soon as possible.

In both cases, we are finished when all the leaves have been encountered.

Tree traversals/Breadth-first

Let us consider two examples of breadth-first traversals.

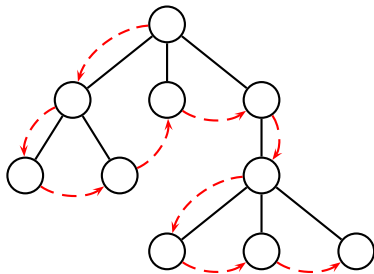


This is a **left to right** traversal.

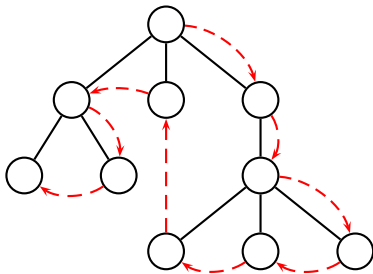
Many others are possible, like choosing randomly the next node of the following level.

Tree traversals/Depth-first

Let us consider two examples of depth-first traversals.



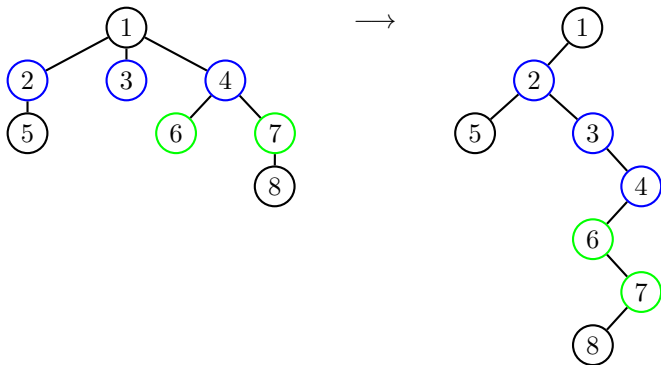
This is a **left to right** traversal.



This is a **right to left** traversal.

Binary trees

In order to simplify, let us consider here trees with two direct subtrees. They are called **binary trees**. We do not lose generality with this restriction: it is always possible to map any tree to a binary tree in a unique way: the **left son, right brother** technique:




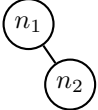
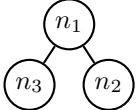
Binary trees/Signature

Let us formally define a binary tree and call it $\text{BIN-TREE}(\text{node})$, where node is the type of the nodes. Here is the **signature**:

- **Parameter types**
 - The type node of the nodes.
- **Defined types**
 - The type of the binary trees is t .
- **Constructors**
 - $\text{EMPTY} : t$
The constant EMPTY is the empty tree (it is a constant function).
 - $\text{MAKE} : \text{node} \times t \times t \rightarrow t$
The tree $\text{MAKE}(n, t_1, t_2)$ as root n and subtrees t_1 and t_2 .

Binary trees/Signature (cont)

Let us show some examples of tree construction:

EMPTY	\emptyset
MAKE(n_1 , EMPTY, EMPTY)	
	
MAKE(n_1 , EMPTY, MAKE(n_2 , EMPTY, EMPTY))	
MAKE(n_1 , MAKE(n_3 , EMPTY, EMPTY), MAKE(n_2 , EMPTY, EMPTY))	

Binary trees/Signature (cont)

The only projection for binary trees is the reverse function for MAKE:

$$\text{MAKE}^{-1} \circ \text{MAKE} = id$$

where id is the identity function. In other words

$$\forall n, t_1, t_2 \quad \text{MAKE}^{-1}(\text{MAKE}(n, t_1, t_2)) = (n, t_1, t_2)$$

We gave no name to the reverse of MAKE because (n, t_1, t_2) does not correspond to a well-identified concept. Then let us choose more intuitive projections.

Binary trees/Signature (cont)

- **Projections**

- $\text{MAKE}^{-1} : t \rightarrow \text{node} \times t \times t$

This is the inverse function of constructor MAKE.

- $\text{ROOT} : t \rightarrow \text{node}$

Expression $\text{ROOT}(t)$ represents the root of tree t .

- $\text{LEFT} : t \rightarrow t$

Expression $\text{LEFT}(t)$ denotes the left subtree of tree t .

- $\text{RIGHT} : t \rightarrow t$

Expression $\text{RIGHT}(t)$ denotes the right subtree of tree t .

Binary trees/Equations

As we guessed with the case of MAKE^{-1} , we can complement our signature using **equations** (or **axioms**):

$$\forall n, t_1, t_2 \quad \text{ROOT}(\text{MAKE}(n, t_1, t_2)) = n$$

$$\forall n, t_1, t_2 \quad \text{LEFT}(\text{MAKE}(n, t_1, t_2)) = t_1$$

$$\forall n, t_1, t_2 \quad \text{RIGHT}(\text{MAKE}(n, t_1, t_2)) = t_2$$

The signature and the equations make a **specification**.

This abstract definition allows you to use the **programming language** you prefer to implement the specification $\text{BIN-TREE}(\text{node})$, where the type `node` is a **parameter**

Binary trees/Equations (cont)

As a remark, let us show how `ROOT`, `LEFT` and `RIGHT` can actually be defined by composing projections — hence they are projections themselves.

The only thing we need is the projections p_1 , p_2 and p_3 on 3-tuples:

$$p_1(a, b, c) = a$$

$$p_2(a, b, c) = b$$

$$p_3(a, b, c) = c$$

Then it is obvious that we could have defined `ROOT`, `LEFT` and `RIGHT` only with basic projections:

$$\text{ROOT} = p_1 \circ \text{MAKE}^{-1}$$

$$\text{LEFT} = p_2 \circ \text{MAKE}^{-1}$$

$$\text{RIGHT} = p_3 \circ \text{MAKE}^{-1}$$

Let us define $\text{FST}(x, y) = x$ and $\text{SND}(x, y) = y$.

Binary trees/Equations (cont)

Note that our definition of `BIN-TREE` is incomplete on purpose: taking the root of an empty tree is undefined (i.e. there is no equation about this).

The reason is that we want the implementation, i.e. the program, to refine and handle such kind of situations.

For instance, if your programming language features **exceptions**, you may use them and raise an exception when taking the root of an empty tree. So, it is up to the programmer to make the required tests prior to call a partially defined function.

Binary trees/Equations (cont)

The careful reader may have noticed that some fundamental and necessary equations were missing:

- $\forall n, t_1, t_2 \quad \text{MAKE}(n, t_1, t_2) \neq \text{EMPTY}$

This equation states that the **constructors** of trees (i.e. `EMPTY` and `MAKE`) are unique.

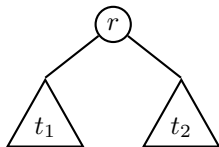
- $\forall n, t_1, t_2, n', t'_1, t'_2 \quad \text{MAKE}(n, t_1, t_2) = \text{MAKE}(n', t'_1, t'_2) \implies (n, t_1, t_2) = (n', t'_1, t'_2)$

This equation states that the constructors with parameters (here `MAKE`) are **injective functions**.

These kind of equations (i.e. uniqueness and injection of constructors) are in fact always desirable, that is why they are usually assumed without explicit statement.

Binary trees/Left to right traversals

Consider a non-empty binary tree:



A depth-first traversal from left to right visits first node r , then the left subtree t_1 and finally the right subtree t_2 . But if we want to keep track of the visited nodes, we have several ways.

- We can record r , then nodes of t_1 and finally nodes of t_2 : this is **left prefix traversal**;
- we can record nodes of t_1 , then r and nodes of t_2 : this is a **left infix traversal**;
- we can record nodes of t_1 , then nodes of t_2 and finally r : this is a **left postfix traversal**.

Binary trees/Left prefix traversal

Let us augment the specification $\text{BIN-TREE}(\text{node})$ with a new function realising a **left prefix traversal**. In order to record the traversed nodes, we need an additional structure. Let us take a stack and call our traversal LPREF .

The additional signature is straightforward:

$$\text{LPREF} : \text{BIN-TREE}(\text{node}).t \rightarrow \text{STACK}(\text{node}).t$$

The corresponding equations are

$$\text{LPREF}(\text{EMPTY}) = \text{EMPTY}$$

$$\text{LPREF}(\text{MAKE}(\mathbf{e}, t_1, t_2)) = \text{PUSH}(\mathbf{e}, \text{APPEND}(\text{LPREF}(t_1), \text{LPREF}(t_2)))$$

where we omitted the prefixes “ $\text{BIN-TREE}(\text{node})$ ” and “ $\text{STACK}(\text{node})$ ”.

Binary trees/Left prefix traversal (cont)

These equations must obviously be oriented from left to right:

$$\text{LPREF}(\text{EMPTY}) \rightarrow \text{EMPTY}$$

$$\text{LPREF}(\text{MAKE}(e, t_1, t_2)) \rightarrow \text{PUSH}(e, \text{APPEND}(\text{LPREF}(t_1), \text{LPREF}(t_2)))$$

where we omitted the specification prefixes.

This is a left to right traversal if the evaluation strategy computes the value of arguments *from left to right*.

It is important to distinguish between the moment when a node is encountered and when it is added in the resulting stack. Therefore, if the arguments of LPREF are computed from right to left, the traversal is from right to left, but the nodes in the final stack will be ordered from left to right (as specified).

Binary trees/Left postfix traversal

The additional signature for left postfix traversal is:

$$\text{LPOST} : \text{BIN-TREE}(\text{node}).t \rightarrow \text{STACK}(\text{node}).t$$

The corresponding equations are

$$\text{LPOST}(\text{EMPTY}) = \text{EMPTY}$$

$$\text{LPOST}(\text{MAKE } (e, t_1, t_2)) =$$

$$\text{APPEND}(\text{LPOST}(t_1), \text{APPEND}(\text{LPOST}(t_2), \text{PUSH}(e, \text{EMPTY})))$$

where we omitted the specification prefixes.

Binary trees/Left postfix traversal (cont)

We orient these equations from left to right:

$$\text{LPOST}(\text{EMPTY}) \rightarrow \text{EMPTY}$$

$$\text{LPOST}(\text{MAKE}(e, t_1, t_2)) \rightarrow$$

$$\text{APPEND}(\text{LPOST}(t_1), \text{APPEND}(\text{LPOST}(t_2), \text{PUSH}(e, \text{EMPTY})))$$

where we omitted the specification prefixes.

Binary trees/Left infix traversal

The additional signature for left infix traversal is simply:

$$\text{LINF} : \text{BIN-TREE}(\text{node}).t \rightarrow \text{STACK}(\text{node}).t$$

The corresponding equations are

$$\text{LINF}(\text{EMPTY}) = \text{EMPTY}$$

$$\text{LINF}(\text{MAKE } (e, t_1, t_2)) = \text{APPEND}(\text{LINF}(t_1), \text{PUSH}(e, \text{LINF}(t_2)))$$

where we omitted the specification prefixes.

Binary trees/Left infix traversal (cont)

We must orient these equations from left to right:

$$\text{LINF}(\text{EMPTY}) \rightarrow \text{EMPTY}$$

$$\text{LINF}(\text{MAKE}(e, t_1, t_2)) \rightarrow \text{APPEND}(\text{LINF}(t_1), \text{PUSH}(e, \text{LINF}(t_2)))$$

where we omitted the specification prefixes.

Binary trees/Breadth-first traversal

Let us consider the pictures of page 46.

The idea is that we need to find at each level the nodes belonging to the next level, adding them to the previous nodes and repeat the search. So we need to handle at each level a **forest**, i.e. a set (or stack) of trees, not just nodes because nodes do not contain the subtrees (thus the next level nodes).

Therefore let us add first a function to the signature of `BIN-TREE(node)`:

$$\mathcal{B} : \text{STACK}(\text{BIN-TREE}(\text{node}).t).t \rightarrow \text{STACK}(\text{node}).t$$

such that expression $\mathcal{B}(f)$ is a stack of the nodes of forest f traversed in a breadth-first way from left to right.

Let specify $\text{FOREST}(\text{node}) = \text{STACK}(\text{BIN-TREE}(\text{node}).t)$

Binary trees/Breadth-first traversal (cont)

Then we define a function

$$\text{BFS} : \text{BIN-TREE}(\text{node}).t \rightarrow \text{STACK}(\text{node}).t$$

such that expression $\text{BFS}(t)$ is a stack of the nodes of the tree t traversed in a breadth-first way from left to right. The corresponding equation is simply

$$\text{BFS}(t) = \mathcal{B}(\text{STACK}(\text{node}).\text{PUSH}(t, \text{EMPTY}))$$

Now, in order to define $\mathcal{B}(f)$ we need to get

- the roots of the forest f ,
- the forest rooted at level 1 of the forest f .

Binary trees/Breadth-first traversal (cont)

Let ROOTS have the signature

$$\text{ROOTS} : \text{FOREST}(\text{node}).t \rightarrow \text{STACK}(\text{node}).t$$

The corresponding equations are not difficult to guess:

$$\text{ROOTS}(\text{FOREST}(\text{node}).\text{EMPTY}) = \text{STACK}(\text{node}).\text{EMPTY}$$

$$\text{ROOTS}(\text{PUSH}(\text{EMPTY}, f)) = \text{ROOTS}(f)$$

$$\text{ROOTS}(\text{PUSH}(\text{MAKE}(r, t_1, t_2), f)) = \text{PUSH}(r, \text{ROOTS}(f))$$

These equations must be oriented from left to right (do you see why?).

Binary trees/Breadth-first traversal (cont)

We have to define now a function which, given a forest, returns the forest at level 1, i.e. all the subtrees rooted at level 1 for each tree in the initial forest. Let us call it

$$\text{NEXT} : \text{FOREST}(\text{node}).t \rightarrow \text{FOREST}(\text{node}).t$$

The equations are

$$\text{NEXT}(\text{FOREST}(\text{node}).\text{EMPTY}) = \text{FOREST}(\text{node}).\text{EMPTY}$$

$$\text{NEXT}(\text{PUSH}(\text{EMPTY}, f)) = \text{NEXT}(f)$$

$$\text{NEXT}(\text{PUSH}(\text{MAKE}(r, t_1, t_2), f)) = \text{PUSH}(t_1, \text{PUSH}(t_2, \text{NEXT}(f)))$$

Note the similarities between NEXT and ROOTS. Note also that t_1 and t_2 may be EMPTY, for the sake of simplicity.

Binary trees/Breadth-first traversal (cont)

Now we can write the equations of \mathcal{B} , using ROOTS and NEXT:

$$\mathcal{B}(\text{FOREST}(\text{node}).\text{EMPTY}) = \text{STACK}(\text{node}).\text{EMPTY}$$

$$\mathcal{B}(f) = \text{APPEND}(\text{ROOTS}(f), \mathcal{B}(\text{NEXT}(f)))$$

where $f \neq \text{EMPTY}$.

Match the traversal as defined formally here against a figure page 46.

Let us orient these equations from left to right.

In order to show that the resulting rewriting system terminates, we have to compare the height of the *values* of f and $\text{NEXT}(f)$.

Binary trees/Breadth-first traversal (cont)

The intuition is that expression $\text{NEXT}(f)$ denotes a *larger* forest than f , because it is made of the direct subtrees of f (if they are all non-empty, then we get twice as more trees than in f), but the trees are strictly *smaller*.

Since we traverse the levels in increasing order, these trees will finally be empty. But the second equation discards empty trees, so in the end, we really get an empty forest.

With this reasoning we show that the terminating orientation is

$$\text{NEXT}(\text{FOREST}(\text{node}).\text{EMPTY}) \rightarrow \text{FOREST}(\text{node}).\text{EMPTY}$$

$$\text{NEXT}(\text{PUSH}(\text{EMPTY}, f)) \rightarrow \text{NEXT}(f)$$

$$\text{NEXT}(\text{PUSH}(\text{MAKE}(r, t_1, t_2), f)) \rightarrow \text{PUSH}(t_1, \text{PUSH}(t_2, \text{NEXT}(f)))$$