15–150: Principles of Functional Programming: *Totality of* `depth`

Michael Erdmann

Spring 2026

This note proves totality of the function `depth` discussed in lecture. Here is the relevant code:

```
datatype tree = Empty | Node of tree * int * tree

(* depth : tree -> int
   REQUIRES: true
   ENSURES: depth(T) computes the depth of tree T.
*)
fun depth (Empty : tree) : int = 0
  | depth (Node(t1, _, t2) : tree) : int = 1 + Int.max(depth t1, depth t2)
```

Theorem: `depth` is total.

Proof: Establishing totality of `depth` means proving that `depth(T)` reduces to a value (of type `int`) for all values `T : tree`. We will prove that assertion by structural induction on `T`.

BASE CASE: `T = Empty`.

NEED TO SHOW: `depth (Empty)` reduces to a value (of type `int`).

SHOWING: `depth (Empty)`

$\Longrightarrow$ `0` [first clause of `depth`]

`0` is a value (of type `int`), so we have established the base case.

INDUCTIVE CASE: `T = Node(`t_1`,x,`t_2`)`, for some values t_1`:tree`, `x:int`, and t_2`:tree`.

INDUCTION HYPOTHESIS: We have one hypothesis for each subtree:
(1) `depth(`t_1`)` $\hookrightarrow$ d_1, for some value d_1 `: int`
(2) `depth(`t_2`)` $\hookrightarrow$ d_2, for some value d_2 `: int`.

NEED TO SHOW: `depth(Node(`t_1`, x, `t_2`))` $\hookrightarrow$ `d` for some value `d : int`.

SHOWING:

`depth (Node(`t_1`, x, `t_2`))`

$\Longrightarrow$ `1 + Int.max(depth t1, depth t2)` [second clause of `depth`]

$\Longrightarrow$ `1 + Int.max(`d_1`, depth t2)` [Induction Hypothesis (1)]

$\Longrightarrow$ `1 + Int.max(`d_1`, `d_2`)` [Induction Hypothesis (2)]

$\Longrightarrow$ `d` [for some value `d:int`, assuming SML math is correct (*)]

(*) Relevant here in particular is that the functions `Int.max` and `+` are total.
That establishes the inductive case.

The base case and the inductive case together establish the Theorem, by a principle of structural induction for type `tree`.