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

Michael Erdmann
Spring 2020

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, x, 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.

INDUCTION STEP: `T = Node(`t_1`,x,`t_2`)`, for some values `t1:tree`, `x:int`, and `t2:tree`.

INDUCTION HYPOTHESIS: `depth(`t_1`)` $\hookrightarrow$ d_1 and `depth(`t_2`)` $\hookrightarrow$ d_2, for some values d_1 and d_2 of type `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`, `d_2`)` [Inductive Hypothesis]

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

That establishes the induction step.

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