Evaluation Strategies and Termination

Call-by-name, Call-by-value and Termination

- Call-by-name and Call-by-value evaluation strategies reduce an expression to the same value, as long as both expressions terminate

THEOREM: ***If Call-by-value evaluation of an expression e terminates, then Call-by-name evaluation of e terminates too. The other direction is not true.***

Question: *Find an expression that terminates under Call-by-name but doesn't terminate under Call-by-value.*

def loop: Int = loop
def first (x: Int, y: Int): Int = x
consider the expression: first (1, loop)

- ⇨ *Under Call-by-name, it evaluates at 1*
- ⇨ *Under Call-by-value, it goes around on the evaluation of argument loop*

Scala normally uses call-by-value, usually because it can avoid exponentially many reduction steps because it avoids duplications of evaluations. You can use call-by-name by adding => in front of the type of the function parameter.

Example:

def constOne (x: Int, y: => Int) = 1
let's trace the evaluations of:

- ∗ *constOne (1+2, loop) => constOne (3, loop) => 1*
- ∗ *constOne (loop, 1+2) => constOne (loop, 1+2) => constOne (loop, 1+2) => …*