Strict Observations

Jan Christiansen
Department of Computer Science
CAU Kiel, Germany
jac@informatik.uni-kiel.de

Abstract

Call-by-name is known to be an optimal evaluation strategy with respect to termination. In practice you can only benefit from this result if functions are not unnecessarily strict. Often it is not trivial to implement a function in a least strict way. In the following we use the lazy functional programming language Haskell.

Consider a data type for Peano Numbers. A Peano Number is either zero or the successor of another Peano Number.

data *Peano* = *Zero* | *Succ Peano*

We define the multiplication of two numbers by means of the addition *plus*.

mult :: *Peano* → *Peano* → *Peano*
mult Zero y = *Zero*
mult (*Succ x*) *y* = *plus y* (*mult x y*)

Furthermore we define an infinite Peano number which is an infinite sequence of successors.

infinity :: *Peano*
infinity = *Succ infinity*

Thanks to non-strictness the evaluation of *mult Zero infinity* yields *Zero*. But the evaluation of *mult infinity Zero* does not terminate. We can improve *mult* with respect to strictness and therefore termination. Consider the following implementation of the multiplication.

mult′ :: *Peano* → *Peano* → *Peano*
mult′ *Zero y* = *Zero*
mult′ (*Succ* _) *Zero* = *Zero*
mult′ (*Succ x*) *y* = *plus y* (*mult*′ *x y*)

For all total arguments *mult* and *mult*′ yield the same results. But the evaluation of *mult*′ *infinity Zero* yields *Zero*. Note that it is not trivial to check whether *mult*′ is less strict than *mult* as the two functions can still be incomparable.

This is not an artificial example. The paper "Declaring Numbers" [1] introduces lazy natural numbers by a binary representation. As a motivating example they present a multiplication of Peano Numbers which is implemented in the exact same manner as *mult*. Furthermore the library "Numbers"[1] provides an implementation of Peano Numbers which implements the multiplication in the same manner as *mult* . Both paper and library care about least strict implementations. This demonstrates that it is not even trivial to implement a simple function like the multiplication of Peano Numbers in a least strict way. On the other hand you can only benefit from non-strict programming languages if functions are least strict.

Olaf Chitil first presented the idea of least-strictness and implemented a tool called StrictCheck [2]. The tool was supposed to check whether a function is least strict. But it has some shortcomings. In particular it does not consider sequentiality. That is, it suggests implementations that are not implementable in a sequential language. In the long term we aim at a lightweight tool for checking whether a function is least strict that does consider sequentiality.

References

[1] Bernd Brassel, Sebastian Fischer, and Frank Huch. Declaring numbers. In *Proc. of the 16th International Workshop on Functional and (Constraint) Logic Programming WFLP 2007*, 2007.

[2] Olaf Chitil. Promoting non-strict programming. In *Draft Proceedings of the 18th International Symposium on Implementation and Application of Functional Languages, IFL 2006*, pages 512–516, Budapest, Hungary, September 2006. Eotvos Lorand University.

[1] available via hackage (`http://hackage.haskell.org`)