Everything you always wanted to know about Doaitse

Andres Löh[1] and José Pedro Magalhães[2]

[1] andres@well-typed.com
Well-Typed LLP
[2] jpm@cs.ox.ac.uk
Department of Computer Science, University of Oxford

Abstract. Doaitse is a very special person. His humor, wit, brightness, and character never fail to impress any audience, independently of context or location. He has continuously inspired and motivated us, and we will always remember our time in Utrecht with fond memories of Doaitse. We collect an (obviously non-exhaustive) list of such memories here. Whether he be in Utrecht, Tynaarlo, or anywhere else, we will always remember Doaitse's unique personality.

Notation

The use of syntax such as <\$>, <∗>, and <|> when programming with applicative functors, as popularised by McBride and Paterson [1], is now commonplace. In a private communication with Doaitse, however, we were able to ascertain that this notation was, in fact, invented by him. Indeed, it can already be seen in one of his papers dating back to 1996 [2]. As is common knowledge, the syntax of a programming language is a contentious and dangerous subject, easily sparking irate online discussions and flame wars. In fact, some language feature proposals end up never being implemented due to disagreement on syntax alone. As such, we will appropriately credit Doaitse for the invention of this beautiful notation by adapting the following naming convention in our future papers:

<\$>	*Swierstra florin*
<∗>	*Swierstra star*
<|>	*Swierstra dike*

Circularity

Doaitse once came upon the following fragment of code, which diagonalises a series of (potentially infinite) lists into a single list:

```
diag :: [[α]] → [α]
diag = concat ∘ foldr skew [] ∘ map (map (:[]))
skew :: [[α]] → [[α]] → [[α]]
skew []      l = l
skew (h : t) l = h : combine (++) t l
combine :: (α → α → α) → [α] → [α] → [α]
combine _ xs     []      = xs
combine _ []      ys     = ys
combine f (x:xs) (y:ys) = f x y : combine f xs ys
```

Having found it verbose and hard to analyse in terms of computational complexity, Doaitse came up with the following code instead:

```
diag :: [[α]] → [α]
diag xs = diag′ xs [] [] where
  diag′ []            [] []      = []
  diag′ []            ll (r:rr) =   diag′ (r:ll) []       rr
  diag′ []            ll []     =   diag′ ll     []       []
  diag′ ((v:vv):ww)  ll rr     = v : diag′ ww   (vv:ll) rr
  diag′ ([]:ww)      ll rr     =   diag′ ww     ll       rr
```

Although it diagonalises in a different way, it is as valid as the original program, and more clearly linear on its input.

But Doaitse was not happy yet, and one day, while biking, he came up with a simple two-line solution, which he wrote on the whiteboard when he arrived at the office. Given the ephemeral nature of writings on a whiteboard, we cannot recall the complete details of that program. We leave it here with a few gaps, which Doaitse can certainly fill in quickly:

```
let xs = ...   -- hint: use xs here
in ...         -- hint: use xs here too
```

The most remarkable property of this implementation was not its size, nor its clear circularity, however; it was the fact that it only worked correctly when given an infinite input, diverging otherwise.

Languages

In the invitation to prepare this submission, we were told we could use "any language that you know Doaitse can read". Dado que o Doaitse consegue ler basicamente qualquer linguagem, decidimos escrever este parágrafo em diversas linguagens. Doaitse venait souvent dans le bureau et commençait une conversation dans la langue maternelle de l'auditeur. In tal fan Doaitse's sinnen wiene simpel mar soad oare wiene bysûnder slim. Se dice que el ha aprendido muchos idiomas solo por escuchar una vieja serie de lecciones grabadas. Среди его словарного запаса готовых фраз были такие как "Здравствуйте, дети!". Weitere Lieblingssätze waren "Jetzt geht's los, die Löcher aus dem Käse!", "Heute bin ich ein richtiger Krawallmacher!", "Das wird schmecken!" sowie generell jeder deutsche Satz, in dem Passiv verwendet wird. Hoewel een buitenstaander zulke zinnen alleen in specifieke omstandigheden zou gebruiken, lukt het Doaitse altijd om de actuele situatie aan zijn zinnen aan te passen.

Christmas

On the 5th of December 2008, a day that is traditionally celebrated in the Netherlands with an exchange of presents during the "Sinterklaasavond", Doaitse released his

`ChristmasTree` package for the (Haskell) world to download and unpack.[3] This package's name is, however, a misnomer; the correct rendering would be `CHRISTMASTREE`, as it is, in fact, an acronym, standing for "Changing Haskell's Read Implementation Such That by Manipulating ASTs it Reads Expressions Efficiently".

Furthermore, it is not only brilliantly named; it also does what its name promises, reading data in linear time, and avoiding the standard Haskell *read* exponential behavior in some cases of data types with infix operators.

A rather amazing program

Profiling, in GHC, adds annotations to a program, so that information can be collected at runtime to determine its space and time behaviour. This invariably makes programs slower, so profiling is only used as a debugging tool. Except Doaitse found a program which ran *faster* (nearly twice as fast) with profiling.[4] We reproduce (most of) this program below:

```
main          = parseIO uulibP inputExample >>= print
uulibP        = length <$> (pList $ pChoice $ map (pSym) ['a'..'z'])
pChoice ps    = foldr (<|>) pFail ps
inputExample  = foldr (++) "" $ replicate 1000 difficultString
difficultString = "abcdefadsjkhdasjkdasjhkdsakjdsajkdsafklfddsfajklyrrtttryytuuyttyuuytuytyuu
```

(Notice Doaitse's use of his own *florin* and *dike* operators.) Simon Peyton Jones said this was "clearly a rather amazing program". Simon Marlow considered it "fascinating", and said he would have to sleep on it to see if a fix would occur to him. Some time later, a good dream (or perhaps a nightmare) told Simon Marlow that the way to fix this behaviour was by "avoiding generating chains of indirections in stack squeezing".

Postscript

Thank you, Doaitse, for being such an amazing person, and having helped us both countless times in our days in Utrecht. You have shaped our way of thinking, and made us better persons, both personally and academically.

References

1. McBride, C., Paterson, R.: Applicative programming with effects. J. Funct. Program. **18**(1) (January 2008) 1–13
2. Swierstra, S.D., Duponcheel, L.: Deterministic, error-correcting combinator parsers. In: Advanced Functional Programming, Second International School-Tutorial Text, London, UK, UK, Springer-Verlag (1996) 184–207

[3] `http://hackage.haskell.org/package/ChristmasTree`
[4] `http://hackage.haskell.org/trac/ghc/ticket/5505`