Laziness in GHC Haskell

The features and principles

Presented by chip

ZJU Lambda
From here to World

ZJU-Lambda Conference, May 2019

Contents

Example 1: No evaluation

```haskell
f :: Int -> Int -> Int
f x y = case x > 0 of
    True  -> x - 1
    False -> x + 1

main = print $ f 1 (product [1..])
```

Well, it prints `0`

Example 2: Evaluate to WHNF

```haskell
length' :: [a] -> Int
length' lst = go lst 0 where
    go [] acc     = acc
    go (x:xs) acc = go xs (acc+1)

main = let x = product [1..]
       in print $ length' [1, x]
```

It prints 2 !
What happened here?

Example 2: Evaluate to WHNF

The actual evaluating process:

```
  length' [1, x]
= length' 1:(x:[])          -- 1:(x:[]) matches (x:xs)
= 1 + length' (x:[])        -- (x:[]), same with above
= 1 + 1 + length' []        -- [] matches []
= 1 + 1 + 0
= 2
```

Concept

In WHNF, we only evaluate the outermost constructor

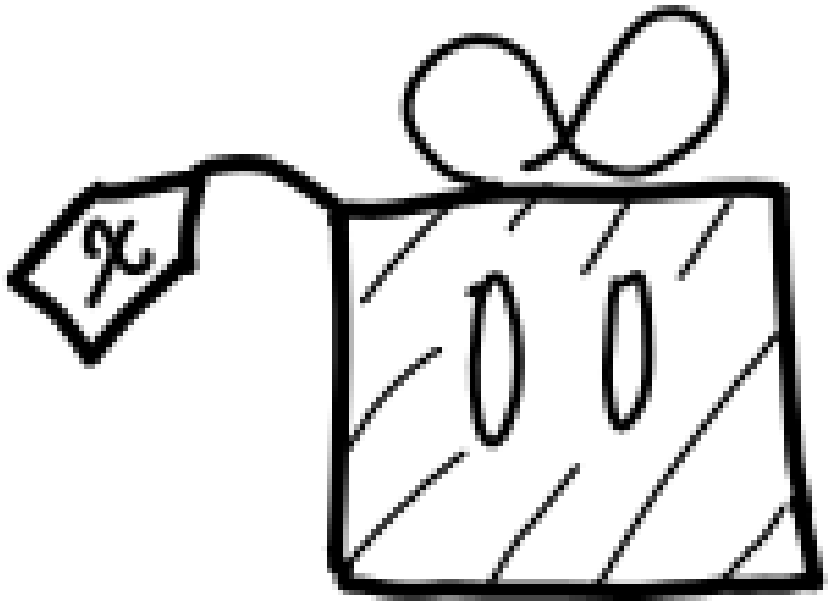
Contents

The Haskell Heap

The Haskell heap is a rather strange place.

Thunk

Every item is wrapped up nicely in a box:
The Haskell heap is a heap of *presents* (thunks).



Present

When you actually want what's inside the present, you *open it up* (evaluate it).

Gift card

Sometimes you open a present, you get a *gift card* (data constructor).
Gift cards have two traits.

- A name. (the **Just** gift card or **Right** gift card)
- And they tell you where the rest of your presents are.

There might be more than one (the tuple gift card), if you're a lucky duck!