Presentation of Haskell

Hackerspace Trento

Corentin Dupont

March 19, 2014

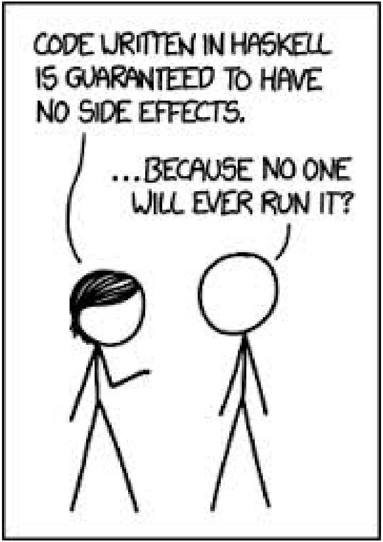
Table of Contents

1 Motivation

2 Features

- Functional
- Pure
- Lazy
- Static Typing

3 IO


CODE WRITTEN IN HASKELL IS GUARANTEED TO HAVE NO SIDE EFFECTS.
...BECAUSE NO ONE WILL EVER RUN IT?

What does this code do?

```
void f(int a[], int lo, int hi)
{
  int h, l, p, t;

  if (lo < hi) {
    l = lo;
    h = hi;
    p = a[hi];

    do {
      while ((l < h) && (a[l] <= p))
          l = l+1;
      while ((h > l) && (a[h] >= p))
          h = h-1;
      if (l < h) {
          t = a[l];
          a[l] = a[h];
          a[h] = t;
      }
    } while (l < h);

    a[hi] = a[l];
    a[l] = p;

    f( a, lo, l-1 );
    f( a, l+1, hi );
  }
```

The same in Haskell

```
qsort [] = []
qsort (p:xs) = (qsort lesser) ++ [p] ++ (qsort greater)
  where
      lesser = filter (< p) xs
      greater = filter (>= p) xs
```

Motivation

- 10 time less lines than in C,
- Great expressivity,
- Great genericity,
- More readable, more maintainable,
- Very few bugs: if it compiles, 90% chances it works on the first try

Features

- Functional
- Pure
- Statically typed
- Lazy

Functional

- Functions are first order values
- Anonymous functions
- Partial application

Examples

```
myFunc a = a + 1

map (+1) [1..10]

filter (>3) [1, 4, 5, 2]

zip [1, 2, 3] ['a', 'b', 'c']
```

Functional

- Lambda functions
- Pattern matching

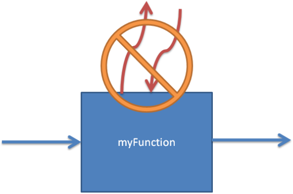
Examples

```
map (\a -> a + 1) [1..10]

fib 0 = 0
fib 1 = 1
fib n = fib (n - 2) + fib (n - 1)
```

Pure

- Functions don't read from environment
- Functions don't write to environment



The output of a function will always be the same for a given input.

Counter examples in C

$$a = f() + g()$$

Can we refactor this into $a = g() + f()$?

$$a = b$$

Can I replace a by b in all my code?

Pure

- No variables!
- No for loops!
- No assignment operator!
- Order of instructions doesn't matter

Pure

Do you really need for loops?

Java

```
for(int i = 0; i < 10; i++) {
    list[i] = list[i] + 1
}

int total = 0;
for(int i = 0; i < 10; i++) {
    total = total + list[i]
}
```

Haskell

```
map (+1) list
foldr 0 (+) list
sum list
```

Pure

What do I win with purity?

- Much less bugs
- Easier to reason with
- Easier to refactor
- Easier to parallelize
- Enables equational reasoning
- Enables laziness

Lazy

- A value is not calculated if it is not used
- Infinite data structures
- Better design for programs
- Memoization

Example

```
take 5 [1..]

fibs = 0 : 1 : (zipWith (+) fibs (tail fibs))
```

Static Typing

- Static typing
  - Declare your intentions to the compiler
  - Filter bugs at compile time
- Genericity
- Type inference

Quizz

What are these functions doing?

```
inc :: Int -> Int
id :: a -> a
map :: (a -> b) -> [a] -> [b]
```

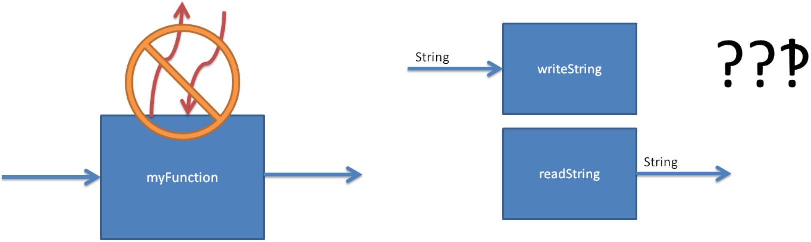
Exercices

Using GHCi, what is the type of:

- 1
- (+1)
- map
- map (+1)
- [1..]
- filter
- foldr
- (.)
- flip

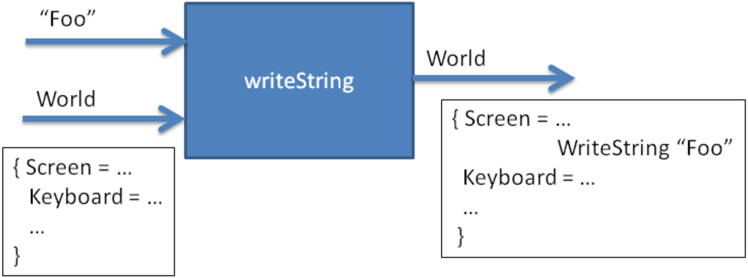
IO

How do we ever perform IO if every function is pure?



IO

Passing the world around

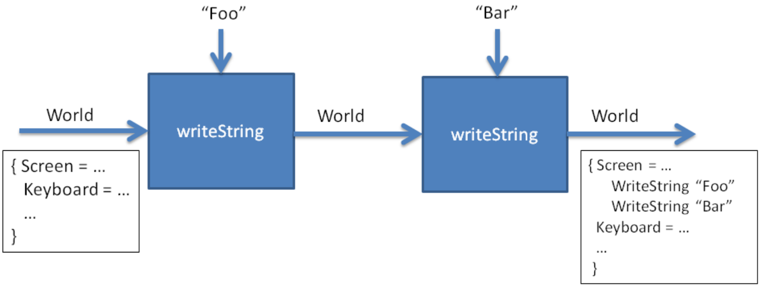


```
{ Screen = ...
  Keyboard = ...
  ...
}
```

```
{ Screen = ...
           WriteString "Foo"
  Keyboard = ...
  ...
}
```

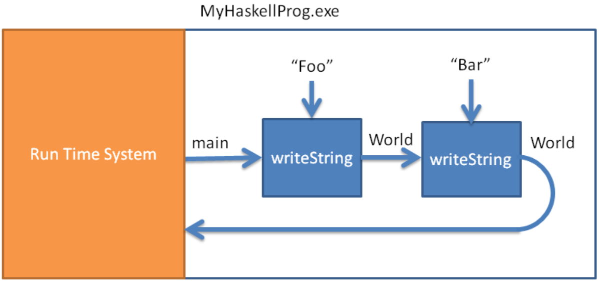
IO

IO operations can now be chained!



IO

The run-time is reading lazily the IO instructions from the output of the chain, and perform them.

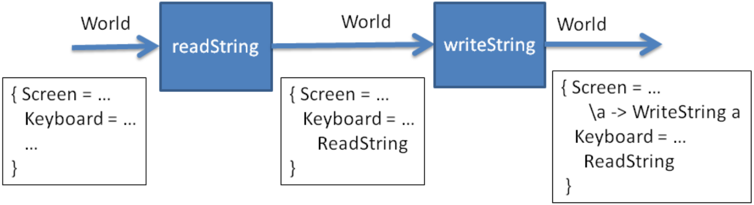


Example

```
main = do
  putStrLn Foo
  putStrLn Bar
```

IO

readString and writeString can also interact because of lazyness.



World

readString

World

writeString

World

```
{ Screen = ...
  Keyboard = ...
  ...
}
```

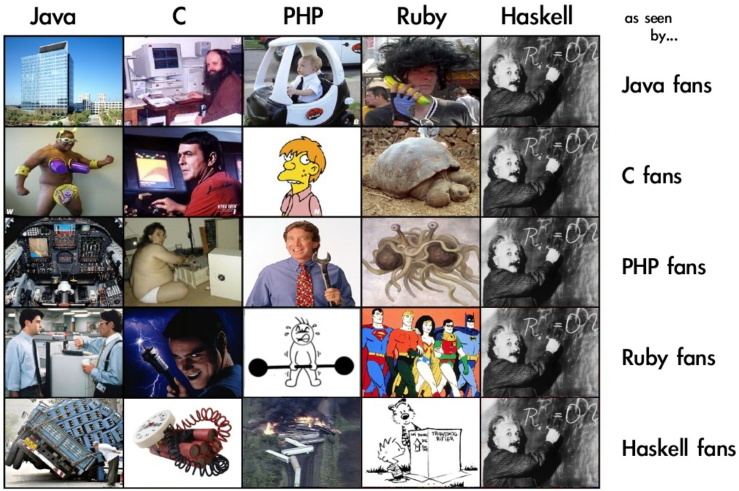
```
{ Screen = ...
  Keyboard = ...
     ReadString
}
```

```
{ Screen = ...
     \a -> WriteString a
  Keyboard = ...
     ReadString
}
```

Example

```
main = do
  a <- readStrLn
  putStrLn a
```

Humor


Java
C
PHP
Ruby
Haskell
as seen by...
Java fans
C fans
PHP fans
Ruby fans
Haskell fans