A Haskell Roadshow

Joachim Breitner

January 20th 2011

What are Haskell's features

- It is a functional language
- It is pure, i.e. side effect free
- It employs lazy evaluation
- It is strongly typed with type inference
- It can be interpreted or compiled
- Large number of libraries available centrally

What are Haskell's features

- It is a functional language
- It is pure, i.e. side effect free
- It employs lazy evaluation
- It is strongly typed with type inference
- It can be interpreted or compiled
- Large number of libraries available centrally

What are Haskell's features

- It is a functional language
- It is pure, i.e. side effect free
- It employs lazy evaluation
- It is strongly typed with type inference
- It can be interpreted or compiled
- Large number of libraries available centrally

What are Haskell's features

- It is a functional language
- It is pure, i.e. side effect free
- It employs lazy evaluation
- It is strongly typed with type inference
- It can be interpreted or compiled
- Large number of libraries available centrally

What are Haskell's features

- It is a functional language
- It is pure, i.e. side effect free
- It employs lazy evaluation
- It is strongly typed with type inference
- It can be interpreted or compiled
- Large number of libraries available centrally

What are Haskell's features

- It is a functional language
- It is pure, i.e. side effect free
- It employs lazy evaluation
- It is strongly typed with type inference
- It can be interpreted or compiled
- Large number of libraries available centrally

Haskell, a Functional language

- Functions are first-class citizens, e.g. can be passed to other functions ("higher order functions").
- Re-use of programing structure, separation of program flow and program logic.
- Example: Modifying each value in a linked list.

Haskell, a Functional language

Example: Modifying each value in a linked list.

Dysfunctional code

```
for (list i = a; i.next != NULL; i = i.next) {
    i.value = i.value + 1;
}
...
for (list i = b; i.next != NULL; i = i.next) {
    i.value = i.value * 2;
}
```

Haskell, a Functional language

Example: Modifying each value in a linked list.

Functional code

```
modifyEach f [] = []
modifyEach f (x:xs) = f x : modifyEach f xs
...
modifyEach (\v -> v + 1) a
...
modifyEach (\v -> v * 2) b
```

Haskell, a pure language

- Variables do not represent memory locations, but values.
  $\Longrightarrow$ not assigned, but bound; no modifications.
- Functions are functions in the mathematical sense
  $\Longrightarrow$ for identical paramters, identical results are calculated.
  ("Referential transparency").
- . . . but we can still do useful things!
- Example: Factorial numbers

Haskell, a pure language

- Variables do not represent memory locations, but values.
  $\Longrightarrow$ not assigned, but bound; no modifications.
- Functions are functions in the mathematical sense
  $\Longrightarrow$ for identical paramters, identical results are calculated.
  ("Referential transparency").
- . . . but we can still do useful things!
- Example: Factorial numbers

Haskell, a pure language

Example: Factorial numbers

Dysfunctional code

```
int fac(int n) {
  int f = 1;
  while (n>0) {
    f = f * n;
    n = n - 1;
  }
  return f;
}
```

Haskell, a pure language

Example: Factorial numbers

Functional code

```
fac n = if n > 0
        then n * fac (n-1)
        else 1
```

Recursion is your staff of life with functional languages!

Haskell, a pure language

Example: Factorial numbers

Functional code

```
fac n = if n > 0
        then n * fac (n-1)
        else 1
```

Recursion is your staff of life with functional languages!

Haskell, a lazy language

- Arguments are not evaluated before a function call, but when they are needed.
- Allows for infinite data structures and other treats!

Functional code

```
const x y = x
... if const (a/42) (42/a) > 0 then ...
```

This code does not divide by zero, even if $a = 0$.

Haskell, a lazy language

- Arguments are not evaluated before a function call, but when they are needed.
- Allows for infinite data structures and other treats!

Functional code

const x y = x
... **if const** (a/42) (42/a) > 0 **then** ...

This code does not divide by zero, even if $a = 0$.

Haskell, a strongly typed language

- Type system is very expressive (Basic types, function types, container types, **newtype**s, phantom types, type classes. . .)
- Can be used to statically guarantee some properties.
- Types need not to be given explicitly.
- "If it compiles, it works."

Dysfunctional code

```
int natSquareRoot(int n) {
  ...
    return s; // found a square root
  ...
  return −1; // no square root found, indicate error
}
```

Haskell, a strongly typed language

- Type system is very expressive (Basic types, function types, container types, **newtype**s, phantom types, type classes. . .)
- Can be used to statically guarantee some properties.
- Types need not to be given explicitly.
- "If it compiles, it works."

Functional code

```
natSquareRoot :: Integer -> Maybe Integer
natSquareRoot n = if {- found a square root s -}
                  then (Just s)
                  else Nothing
```

Haskell, a strongly typed language

- Type system is very expressive (Basic types, function types, container types, **newtype**s, phantom types, type classes. . .)
- Can be used to statically guarantee some properties.
- Types need not to be given explicitly.
- "If it compiles, it works."

Functional code

```
natSquareRoot :: Integer -> Maybe Integer
natSquareRoot n = if {- found a square root s -}
                  then (Just s)
                  else Nothing
```

Haskell, a strongly typed language

- Type system is very expressive (Basic types, function types, container types, **newtype**s, phantom types, type classes. . .)
- Can be used to statically guarantee some properties.
- Types need not to be given explicitly.
- "If it compiles, it works."

Type inference

f a b c d = **if** d **then** c (a, b ++ *"a string"*) **else not** (a b d)

has inferred type

```
f :: ([Char] -> Bool -> Bool) -> [Char]
     -> (([Char] -> Bool -> Bool, [Char]) -> Bool)
     -> Bool -> Bool
```

Haskell, an interpreted and compiled language

- ghci: Interpreter allows for quick experiments
- ghc: Industry-strength compiler, suports several operating systems and architectures
- Free software and comes with your favourite Linux distribution
- Other compilers around as well (mostly research)

Haskell's rich ecosystem

- Haskell Platform: Carefully chosen selection of most common libraries.
- Hackage: Repository with more than 2 700 libraries and programs.
- cabal-install: Downloads libraries form hackage, resolves their dependencies, builds and installs them
- You can contribute!

A life demonstration

The task

Write a program that

- parses a comma-separated value file “pen.csv”, describing the motion of a pen and
- renders the resulting image.

```
GO,80
LEFT
GO,150
RIGHT
GO,20
SAY,"Hello, World!"
RIGHT
RIGHT
GO,160
RIGHT
RIGHT
SAY,"Hello, Haskell!"
```

What we skipped today

All the small things...

- More about data types
- Polymorphism
- Type classes
- Monads
- Foreign Function Interface

...you will find here

- Tutorial "Learn you a Haskell"
- O'Reilly book "Real World Haskell"
- Tutorial "Write Yourself a Scheme in 48 Hours"

What we skipped today

All the small things...

- More about data types
- Polymorphism
- Type classes
- Monads
- Foreign Function Interface

...you will find here

- Tutorial "Learn you a Haskell"
- O'Reilly book "Real World Haskell"
- Tutorial "Write Yourself a Scheme in 48 Hours"

Conclusion

Writing Haskell code

- takes less time
- produces less bugs
- is more fun