Porting Text to UTF-8

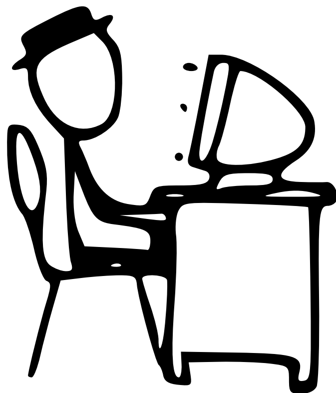
Dutch Haskell User Group

Jasper Van der Jeugt

July 14, 2011

Hello!

My name is Jasper
Student at UGent
I write Haskell
GhentFPG
`@jaspervdj`
`jaspervdj.be`



Overview

Credit where credit is due

High-Performance Haskell, advice
Johan Tibell

Mentoring
Edward Kmett

Overview

Introduction

UTF-8 vs. UTF-16

Porting Text to UTF-8

Benchmarking pitfalls

GHC Core

Results

Overview

Introduction

UTF-8 vs. UTF-16

Porting Text to UTF-8

Benchmarking pitfalls

GHC Core

Results

UTF-8 vs. UTF-16

Number of unicode characters?

17 planes
Each plane: 2^{16} characters

UTF-8 vs. UTF-16

Number of unicode characters?

$17 * 2^{16}$ characters

UTF-8 vs. UTF-16

Number of bits needed?

$\log_2(17 * 2^{16}) = 20.087...$

21 bits per character

UTF-8 vs. UTF-16

String is (often) a bad choice

data Char = C# **Char**#

`C#`: word

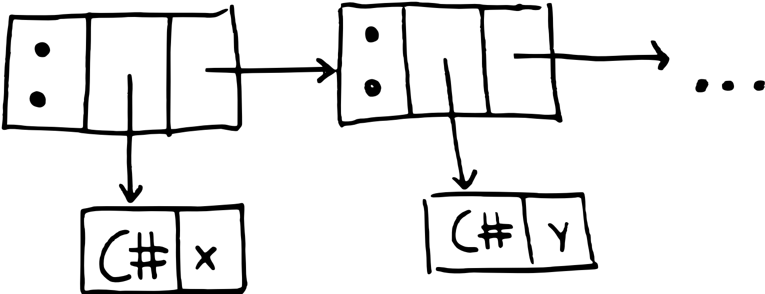
`Char#`: 32 bits

UTF-8 vs. UTF-16

String is (often) a bad choice

```
data [] a = [] | a : [a]
```

UTF-8 vs. UTF-16



UTF-8 vs. UTF-16

Two points:

1. Use strict arrays
2. Don't use a 32-bit encoding

UTF-8 vs. UTF-16

0xxx xxxx

110x xxxx <cb>

1110 xxxx <cb> <cb>

11110xxx <cb> <cb> <cb>

<cb> = 10xx xxxx

UTF-8 vs. UTF-16

0xxx xxxx xxxx xxxx

110110xx xxxx xxxx
110111xx xxxx xxxx

UTF-8 vs. UTF-16

Two points:

1. Some things are inherently faster using UTF-8
2. Some things are inherently faster using UTF-16

Overview

Introduction
UTF-8 vs. UTF-16
Porting Text to UTF-8
Benchmarking pitfalls
GHC Core
Results

Porting Text to UTF-8

```
encodeUtf8 :: Text -> ByteString
```

Implementation very simple

Porting Text to UTF-8

```
encodeUtf8 (Text arr off len) =
  unsafePerformIO $ do
    fp <- mallocByteString len
    withForeignPtr fp $ <memcpy>
    return $! PS fp 0 len
```

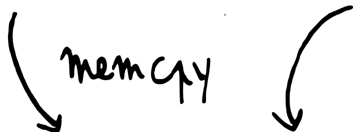
Porting Text to UTF-8

```
decodeUtf8 :: ByteString -> Text
```

Very important to validate first!

Porting Text to UTF-8

```
decodeUtf8 =
```



0xFFFD ...

Porting Text to UTF-8

Implementations of `map`, `filter`...

```
stream   :: Text -> Stream Char
unstream :: Stream Char -> Text
```

Porting Text to UTF-8

```haskell
data Stream a =
    forall s. Stream
    (s -> Step s a) -- stepper
    !s              -- state
    !Size           -- size hint
```

Porting Text to UTF-8

```
data Step s a
    = Done
    | Skip !s
    | Yield !a !s
```

Porting Text to UTF-8

Most functions written as:

```
f :: Text -> Text
f = unstream . f' . stream
```

Porting Text to UTF-8

```
f = unstream . f' . stream
g = unstream . g' . stream
```

Stream fusion:

```
f . g =
    unstream . f' . g' . stream
```

Porting Text to UTF-8

These are not the only streaming combinators...

```
stream . decodeUtf8 = streamUtf8
```

```
streamUtf8 ::
    ByteString -> Stream Char
```

Overview

Introduction
UTF-8 vs. UTF-16
Porting Text to UTF-8
Benchmarking pitfalls
GHC Core
Results

Benchmarking pitfalls

Haskell is a lazy language
This makes benchmarking hard

Benchmarking pitfalls

Two types of benchmarks:
Functions and programs

(we focus on the former)

Benchmarking pitfalls

Benchmarking some function

```haskell
f :: Int -> Int
```

Benchmarking pitfalls

In e.g. Python

```
total = 0
for i in range(100):
    start = time.time()
    f()
    end = time.time()
    total += (end - start) / 100
```

Benchmarking pitfalls

In Haskell?

```
replicateM 100 $ do
    start <- getTime
    let y = f x
    end <- y `seq` getTime
```

This is pretty hard to get right

Benchmarking pitfalls

Conclusion?

Never write your own benchmarking code

Benchmarking pitfalls

Criterion

By Bryan O'Sullivan

Benchmarking pitfalls

Criterion

```
bench "f" $ nf   f x
bench "g" $ whnf g x
```

Benchmarking pitfalls

Eq for string types

```
whnf (== T.init t
      `T.snoc` '\xfffd') t

whnf (== BL.init bl
      `BL.snoc` '\xfffd') bl
```

Benchmarking pitfalls

But `ByteString.Lazy` is a little faster

`Text`: 2.489305 us
`ByteString.Lazy`: 39.29312 **ns**

Benchmarking pitfalls

Digging into the code...

```
eq (Chunk a as) (Chunk b bs) =
  case compare (S.length a)
               (S.length b) of
    ...
    EQ -> a == b && eq as bs
    ...
```

Benchmarking pitfalls

Digging further...

```
eq a@(PS p s l) b@(PS p' s' l')
    -- short cut on length
    | l /= l'                    = False
    -- short cut for same string
    | p == p' && s == s' = True
    | ...
```

Benchmarking pitfalls

Conclusion?

Libraries can be smarter than you think they are, make sure you know what you are benchmarking!

Benchmarking pitfalls

Benchmarking IO

```
bench "HtmlCombinator" $ do
  putStr "Content-Type: ..."
  ...
  putStr "<table>"
  putStr $ toLazyText $
    makeTable 20000
  putStr "</table>"
```

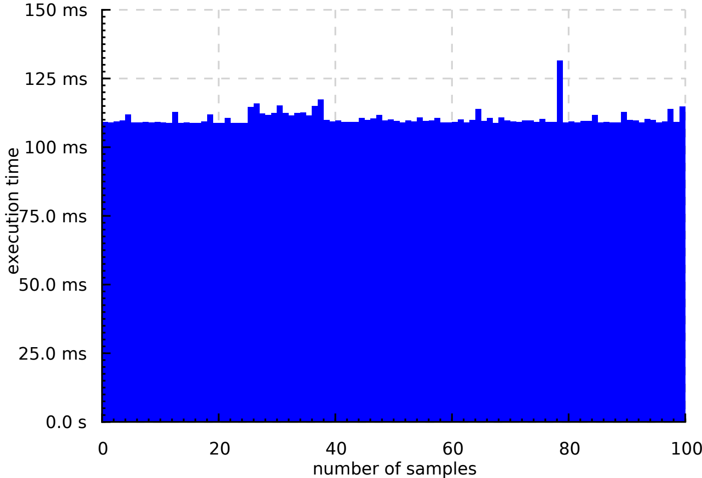
Benchmarking pitfalls

This looks suspicious

```
benchmarking HtmlCombinator
collecting 100 samples (...)
    estimated 30.80161 s
mean: 107.6378 ms (...)
```

$$100 * 100ms \neq 30s$$

Benchmarking pitfalls



Benchmarking pitfalls

Where is the issue?

```
bench "HtmlCombinator" $ do
  putStr "Content-Type: ..."
  ...
  putStr "<table>"
  putStr $ toLazyText $
    makeTable 20000
  putStr "</table>"
```

Benchmarking pitfalls

```
putStr . toLazyText .
    makeTable =<< rows
...
where
  rows :: IO Int
  rows = return 20000
  {-# NOINLINE rows #-}
```

Benchmarking pitfalls

Conclusion?

GHC is pretty smart as well

Overview

Introduction
UTF-8 vs. UTF-16
Porting Text to UTF-8
Benchmarking pitfalls
GHC Core
Results

GHC Core

What is GHC Core?
Why should we care?

GHC Core

What is GHC Core?

Internal representation used by GHC
A kernel language
Optimizations are applied here

GHC Core

Why should we care?

Understanding benchmark results
Know what is going on
Impress your friends!

GHC Core

A few basic rules

GHC Core

Function pattern matching, guards, if's are translated to `case`

GHC Core

`where` is translated to `let`

GHC Core

Type annotations everywhere

GHC Core

Reading core

Clean up qualified names
Use proper variable names
Remove unnecessary type annotations

GHC Core

Demo

Overview

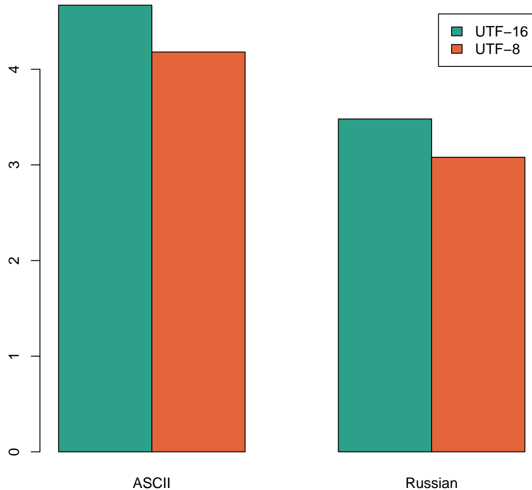
Introduction
UTF-8 vs. UTF-16
Porting Text to UTF-8
Benchmarking pitfalls
GHC Core
Results

Results

```
text <- T.decodeUtf8 <$>
  B.readFile filePath

B.putStr $ T.encodeUtf8 $
  T.toUpper text
```

Results



Questions?