

Haskell ByteStrings

Jan Bessai

13.7.2011

Haskell Strings

Implementierung

Diskussion

ByteStrings

Implementierung

Diskussion

Evil Hack

Lazy ByteStrings

Implementierung

Fusion

Diskussion

Abschluss

Fazit

Ausblick

Quellen

Überblick

Haskell Strings

Implementierung

Diskussion

ByteStrings

Implementierung

Diskussion

Evil Hack

Lazy ByteStrings

Implementierung

Fusion

Diskussion

Abschluss

Fazit

Ausblick

Quellen

Haskell Strings

Implementierung

Diskussion

ByteStrings

Implementierung

Diskussion

Evil Hack

Lazy ByteStrings

Implementierung

Fusion

Diskussion

Abschluss

Fazit

Ausblick

Quellen

type String = [Char]

- ▶ Einfache lineare Listen
- ▶ Lazy
- ▶ Verwenden *Boxed Types*

Haskell Strings

Implementierung

Diskussion

ByteStrings

Implementierung

Diskussion

Evil Hack

Lazy ByteStrings

Implementierung

Fusion

Diskussion

Abschluss

Fazit

Ausblick

Quellen

Boxed Types?

Alle “normalen” Haskell Daten sind *Boxed*:

- ▶ Liegen auf dem Heap
- ▶ Werden über Pointer erreicht (ermöglicht Lazyness)

`Int#`, `Double#`, `Addr#`, `Word8#` sind *Unboxed*:

- ▶ Werden direkt erreicht
- ▶ Entsprechen C primitiven (`int`, `double`, `void*`, `char`)
- ▶ Können auch auf dem Stack liegen

Der Compiler kann Boxed Types wegoptimieren (manchmal)

- ▶ Funktioniert nur bei strikten !Feldern
- ▶ GHC: `-funbox-strict-fields`

► Boxed

```
1 typedef struct _string {
2   char * head;
3   struct _string * cons;
4 } string;
5 // single = ['a']
6 string * single = (string*)malloc(sizeof(string));
7 single->next = NULL;
8 single->head = (char*)malloc(sizeof(char));
9 *(single->head) = 'a';
```

► Unboxed

```
1 typedef struct _string {
2   char head;
3   struct _string * cons;
4 } string;
5 struct string * single = (string*)malloc(sizeof(string));
6 single->next = NULL;
7 single->head = 'a';
```

Haskell Strings

Implementierung

Diskussion

ByteStrings

Implementierung

Diskussion

Evil Hack

Lazy ByteStrings

Implementierung

Fusion

Diskussion

Abschluss

Fazit

Ausblick

Quellen

- ▶ Vollständig Lazy
- ▶ Keine “Sonderbehandlung” von Strings
- ▶ Umbau in $O(1)$
- ▶ Kein interner Verschnitt

- ▶ (doppelt) Indirekte Adressierung
- ▶ Random access in $O(n)$
- ▶ Extrem viele Speicher (De-)Allokationen
- ▶ Overhead durch next-Pointer
- ▶ Anfällig für externen Verschnitt
- ▶ Schlechte Cache Performance

Haskell Strings

Implementierung

Diskussion

ByteStrings

Implementierung

Diskussion

Evil Hack

Lazy ByteStrings

Implementierung

Fusion

Diskussion

Abschluss

Fazit

Ausblick

Quellen

Bytestings sind echte Arrays:

- ▶ Zusammenhängende Speicherblöcke
- ▶ Strikt
- ▶ Nur für Character
- ▶ Unboxed

Enthalten im Paket *bytestring*

Haskell Strings

Implementierung

Diskussion

ByteStrings

Implementierung

Diskussion

Evil Hack

Lazy ByteStrings

Implementierung

Fusion

Diskussion

Abschluss

Fazit

Ausblick

Quellen

- ▶ Installation via Cabal
- ▶ Qualifizierter Import:
`import qualified Data.ByteString as B`
- ▶ Redeklarieren die üblichen Listen-, String und Filefunktionen (\Rightarrow meist Änderungen nur auf Typebene)
- ▶ Bei bekannter Länge:

```
1 unfoldrN :: Int -> (a -> Maybe (Word8, a)) -> a ->  
  (ByteString, Maybe a)
```

- ▶ Einfach zu verwenden
- ▶ Random access (z.B. tail) in $O(1)$
- ▶ Wenig Indirektion
- ▶ Brauchbares Cache-Verhalten
- ▶ Kompatibel mit C Libraries:

```
1 useAsCString :: ByteString -> (CString -> IO a) -> IO a
```

- ▶ Umbau meistens in $O(n)$
- ▶ Strikt
- ▶ Benutzen `Word8` statt `Char`:
 - ▶ Hin und her Konvertieren (`Data.ByteString.Char8`, `Data.ByteString.Internal.c2w`)
 - ▶ \Rightarrow kein voller Unicode Support
- ▶ Nur gut, wenn man Stringlängen a priori kennt
- ▶ Anfällig für internen Verschnitt

Haskell Strings

Implementierung

Diskussion

ByteStrings

Implementierung

Diskussion

Evil Hack

Lazy ByteStrings

Implementierung

Fusion

Diskussion

Abschluss

Fazit

Ausblick

Quellen

Bösartig, aber performant

ByteStrings als einmal allozierte Puffer mit Anamorphismus auf Pointern:

- ▶ `updateUnfoldrN`

- ▶ Verbindet Vorteile von ByteString mit denen von String
- ▶ Lazy Listen aus strikten ByteStrings (Chunks)
- ▶ Chunks in Cache freundlichen Größen
- ▶ In Arbeit: *Fusion*

Haskell Strings

Implementierung

Diskussion

ByteStrings

Implementierung

Diskussion

Evil Hack

Lazy ByteStrings

Implementierung

Fusion

Diskussion

Abschluss

Fazit

Ausblick

Quellen

Fusion (Idee)

Die Komposition von Funktionen erzeugt unnötige
Zwischenstrukturen:

```
1 filter f . map g . unfoldr h
```

Lösung:

- ▶ Definition der Funktionen mittels Einheitlichen primitiven (z.B. Cata-/Anamorphismen: fold, unfold)
- ▶ Optimierender Compiler kennt Rechenregeln auf Verknüpfungen von Primitiven

Haskell Strings

Implementierung

Diskussion

ByteStrings

Implementierung

Diskussion

Evil Hack

Lazy ByteStrings

Implementierung

Fusion

Diskussion

Abschluss

Fazit

Ausblick

Quellen

Beispiel (stream fusion)

Rekursive Strukturen (z.B. Listen) arbeiten wie Automaten (foldl):

- ▶ Ausgehend von einem Zeichen und einem aktuellen Zustand wird ein neuer Zustand produziert

$$a \times La \longrightarrow La$$

Co-Strukturen (z.B. Streams, die Zeichen liefern) arbeiten umgekehrt (unfoldr):

- ▶ Ausgehend vom aktuellen Zustand wird ein Zeichen und ein neuer Zustand produziert

$$a \times La \longleftarrow La$$

Implementierung

```
1 data Stream a = forall s. Stream (s -> Step a s) s
2 data Step a s = Done | Yield a s | Skip s
3
4 -- O(1)
5 stream :: [a] -> Stream a
6 stream xs = Stream next xs
7   where
8     next [] = Done
9     next x:xs' = Yield x xs
10
11 -- O(n), tail rekursiv
12 unstream :: Stream a -> [a]
13 unstream (Stream next s) = case next s of
14   Done -> []
15   Yield x s' -> x : unstream (Stream next s')
16   Skip s' -> unstream (Stream next s')
```

- ▶ $\text{stream} \circ \text{unstream} = \text{id}$
- ▶ $\text{unstream} \circ \text{stream} = \text{id}$

Haskell Strings

Implementierung

Diskussion

ByteStrings

Implementierung

Diskussion

Evil Hack

Lazy ByteStrings

Implementierung

Fusion

Diskussion

Abschluss

Fazit

Ausblick

Quellen

map:

```
1 map f (Stream next s0) = Stream next' s0
2   where
3     next' s = case next s of
4       Done -> Done
5       Yield x s' -> Yield (f x) s'
6       Skip s' -> Skip s'
```

Verknüpfungen von map werden fusioniert:

- ▶ $(\text{unstream} \circ \text{map } f \circ \text{stream}) \circ (\text{unstream} \circ \text{map } g \circ \text{stream}) = \text{unstream} \circ \text{map } f \circ \text{map } g \circ \text{stream}$

Haskell Strings

Implementierung

Diskussion

ByteStrings

Implementierung

Diskussion

Evil Hack

Lazy ByteStrings

Implementierung

Fusion

Diskussion

Abschluss

Fazit

Ausblick

Quellen

- ▶ Vorteile von Strings und ByteStrings vereint
- ▶ Cache freundlich
- ▶ Genauso einfach zu verwenden wie ByteStrings
- ▶ Verschnitt minimal

Haskell Strings

Implementierung

Diskussion

ByteStrings

Implementierung

Diskussion

Evil Hack

Lazy ByteStrings

Implementierung

Fusion

Diskussion

Abschluss

Fazit

Ausblick

Quellen

Haskell Strings

Implementierung

Diskussion

ByteStrings

Implementierung

Diskussion

Evil Hack

Lazy ByteStrings

Implementierung

Fusion

Diskussion

Abschluss

Fazit

Ausblick

Quellen

- ▶ Normale ByteStrings immernoch manchmal effizienter
- ▶ Arbeiten auch mit Word8

- ▶ Strings: Listen mit allen Problemen u. Vorteilen
- ▶ ByteStrings: Arrays mit allen Problemen u. Vorteilen
- ▶ Lazy ByteStrings: Effizienter Hybrid
- ▶ Trickkiste: Unboxing, Fusion
- ▶ Bekanntes, stabiles Paket
- ▶ Fast problemloses Refactoring bei Performanceproblemen

Haskell Strings

Implementierung

Diskussion

ByteStrings

Implementierung

Diskussion

Evil Hack

Lazy ByteStrings

Implementierung

Fusion

Diskussion

Abschluss

Fazit

Ausblick

Quellen

- ▶ Data.Text: ByteStrings in UTF-8
- ▶ Data.Rope: Lazy ByteStrings mit FingerTrees für random access
- ▶ Zukunft von Fusion: Forschungsgebiet, Bugticket #915 (Peyton-Jones)

Haskell Strings

Implementierung

Diskussion

ByteStrings

Implementierung

Diskussion

Evil Hack

Lazy ByteStrings

Implementierung

Fusion

Diskussion

Abschluss

Fazit

Ausblick

Quellen

Ausgesuchte Quellen

- ▶ Homepage über Bytestrings, inklusive Paper über Fusion
- ▶ Haskell Wiki, String Performance
- ▶ GHC Unboxing
- ▶ Data.ByteString
- ▶ Data.Text
- ▶ Data.Rope

Haskell Strings

Implementierung

Diskussion

ByteStrings

Implementierung

Diskussion

Evil Hack

Lazy ByteStrings

Implementierung

Fusion

Diskussion

Abschluss

Fazit

Ausblick

Quellen

Fragen?

Haskell Strings

Implementierung

Diskussion

ByteStrings

Implementierung

Diskussion

Evil Hack

Lazy ByteStrings

Implementierung

Fusion

Diskussion

Abschluss

Fazit

Ausblick

Quellen

▶ Danke :-)