Universität Bremen

Praktische Informatik 3 WS 14/15

3. Übungsblatt

Ausgabe: 04.11.14
Abgabe: 14.11.14

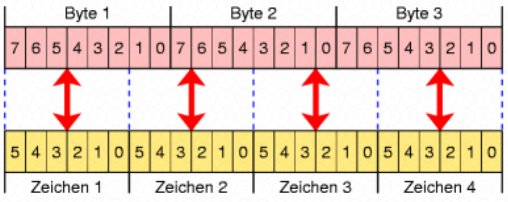
Christoph Lüth
Sandor Herms
Daniel Müller
Jan Radtke
Henrik Reichmann
Sören Schulze
Felix Thielke

3.1 *Kodierungen* — *20 Punkte*

Genug der trockenen Theorie, in dieser Aufgabe geht es mal um etwas praktisches und handfestes. Keine Angst, Sie sollen keine Serienbriefe in Word erstellen...

Wenn man Binärdateien in Textform darstellen will — beispielsweise um in Word erstellte Serienbriefe per Mail zu verschicken — dann werden die Binärdaten *kodiert*, indem sie als Wörter über einem lesbaren Basisalphabet dargestellt werden.

Die Größe des Basisalphabets sollte zweckmäßigerweise eine Zweierpotenz sein. Fig. 1 zeigt als Beispiel die Kodierung zur Basis 64: der binäre Eingabstrom wird in je sechs Bits zusammengefasst, die dann als Index in das $2^6 = 64$ Buchstaben umfassende Ausgabealphabet dienen (nicht dargestellt). Damit wird dann die binäre Eingabe `0x14fb9c03d97e` als Zeichenkette `"FPucA9l+"` kodiert.



Quelle: `http://de.wikipedia.org/wiki/Base64`

Abbildung 1: Schematische Darstellung der Kodierung zur Basis 64

Der RFC 4648 [1] definiert solche Kodierungen zur Basis 64, 32 und 16, wobei eine Kodierung durch folgende Parameter definiert wird:

- die Kodierungsbreite (d.h. die Wortbreite der Länge des Kodierungsalphabets);
- das Kodierungsalphabet und Füllzeichen (*padding*);
- eine Basislänge für die Ausgabe.

In [1] wird insbesondere definiert, wie die Kodierung sich in Randfällen (am Ende der Eingabe) verhält:

- Der Eingabestrom wird ggf. mit Null-Bits aufgefüllt, falls seine Länge kein Vielfaches der Kodierungsbreite ist.
- Der Ausgabestring wird ggf. mit den Füllzeichen auf eine Vielfaches der Basislänge aufgefüllt.

Wir wollen in dieser Aufgabe generische Kodierungen nach RFC 4648 implementieren. Dazu gehen wir in folgenden Schritten vor:

1. Die Binärdaten werden als eine Liste von positiven 8-Bit-Zahlen (Bytes) modelliert, die in Haskell durch den Datentyp `Word8` repräsentiert werden:

   ```
   import Data.Word
   type Byte = Word8
   ```

   `Word8` ist eine Instanz der Typklassen `Num` und `Integral`, wir können also damit rechnen wie mit den meisten anderen Zahlen auch, und mit `fromEnum` und `toEnum` Bytes in `Int` konvertieren und zurück.

2. Für die Bitlisten (Binärzahlen) definieren wir einen zweiwertigen Datentyp und Funktionen, von und nach `Byte` konvertieren:

   ```
   data Bit = O | I

   toByte :: [Bit] → Byte
   fromByte :: Int → Byte → [Bit]
   ```

Bei den Bitlisten soll das MSB (most significant byte) vorne stehen. Bei fromByte gibt das erste Argument die Breite der Binärzahl an (d.h. die Länge der resultierenden Liste; ist die Liste kürzer, soll sie vorne mit O aufgefüllt werden).

3. Dann definieren wir die Kodierungen:

```
data Encoding = Base16 | Base32 | Base64
encWidth :: Encoding→ Int
encSize  :: Encoding→ Int
table    :: Encoding→ String
padding  :: Encoding→ Char
```

   Hier ist encWidth die Kodierungsbreite und table das Kodierungsalphabet, mit length (table e) ==2^(encWidth e), und encSize ist die Basislänge der Ausgabe. Diese Daten entnehmen Sie der referenzierten RFC 4648 [1].

4. Die Kodierung ist ein Funktion

```
encode :: Encoding→ [Byte]→ String
```

   und verläuft in sechs Schritten, die jeder als eine Funktion implementiert werden können:

   (a) Zuerst wird die Eingabe gelesen und in einer Bitliste konvertiert;
   (b) dann wird diese Bitliste in Listen von der Länge der Kodierungsbreite unterteilt;
   (c) jede dieser einzelnen Listen wird dann zu einem Byte konvertiert, so dass wir insgesamt eine Liste von Bytes erhalten;
   (d) jedes dieser Bytes wird als Index in das Basisalphabet interpretiert, und damit in ein Zeichen konvertiert, so dass wir eine Zeichenkette erhalten;
   (e) die Zeichenkette wird am Ende durch das spezifierte Füllzeichen der Kodierung in eine Vielfaches von encSize e aufgefüllt.

5. Die Dekodierung ist eine Funktion

```
decode :: Encoding→ String→ [Byte]
```

   und verläuft analog zur Kodierung (nur umgekehrt) in vier Schritten:

   (a) Zuerst werden aus der Eingabe alle Zeichen entfernt, die nicht im Basisalphabet sind, und dann die Eingabe zu einer Liste von Bytes konvertiert, die dem Index in das Basisalphabet entsprechen;
   (b) Diese Liste von Bytes wird in eine Bitliste konvertiert;
   (c) die Bitliste wird in Listen der Länge 8 aufgeteilt;
   (d) und jede dieser Bitlisten der Länge 8 zu einem Byte konvertiert.

Hinweise:

- Zwischen Kodierung und Dekodierung können Sie viele Funktionen wiederverwenden.
- Für eine korrekte Implementierung sollte folgendes gelten: decode e (encode e s)==s.
- Der RFC enthält einige Testdaten (Abschnitt 10), die auf der Webseite zum Testen zur Verfügung stehen (in der ZIP-Datei).

? *Verständnisfragen*

1. Was ist Polymorphie?
2. Welche zwei Arten der Polymorphie haben wir kennengelernt, und wie unterschieden sie sich?
3. Was ist der Unterschied zwischen Polymorphie in Haskell, und Polymorphie in Java?

Literatur

[1] *Josefsson, S., The Base16, Base32, and Base64 Data Encodings*, RFC 4648, Oktober 2006. (Frei zugänglich unter `http://www.rfc-editor.org/info/rfc4648`.)