

6 Übersetzungen und Codierungen

6.1 Von Wörtern zu Zahlen und zurück

6.1.1 Dezimaldarstellung von Zahlen

Num_{10}

$$\text{Num}_{10}(\varepsilon) = 0 \quad (1)$$

$$\forall w \in Z_{10}^* \forall x \in Z_{10} : \text{Num}_{10}(wx) = 10 \cdot \text{Num}_{10}(w) + \text{num}_{10}(x) \quad (2)$$

- noch ein Beispiel oder langweilig?
- Beweis, dass Definition sinnvoll:

6.1 Lemma. Durch die Gleichungen 1 und 2 ist Num_{10} wohldefiniert, das heißt, für jedes Wort $w \in Z^*$ wird eindeutig der Funktionswert $\text{Num}_{10}(w)$ festgelegt.

Um das mittels vollständiger Induktion zu beweisen, formulieren wir etwas um:

6.2 Lemma. Durch die Gleichungen 1 und 2 ist Num_{10} wohldefiniert, das heißt, für jede Wortlänge $n \in \mathbb{N}_0$ gilt: für alle $w \in Z^n$ wird eindeutig der Funktionswert $\text{Num}_{10}(w)$ festgelegt.

Beweis durch Induktion über die Wortlänge n :

Induktionsanfang $n = 0$: In diesem Fall ist zu beweisen: Für jedes Wort w der Länge $n = 0$ ist $\text{Num}_{10}(w)$ festgelegt, und zwar eindeutig.

Das ist leicht: Wenn $|w| = 0$ ist, dann ist $w = \varepsilon$. Tatsächlich legt Gleichung (1) offensichtlich eindeutig einen Funktionswert $\text{Num}_{10}(\varepsilon)$ fest, nämlich 0. Und Gleichung (2) legt *keinen* Funktionswert für $\text{Num}_{10}(\varepsilon)$ fest, denn die auf der linken Seite auftretenden Argumente haben alle eine Länge $|wx| = |w| + |x| = |w| + 1 \geq 1$, sind also ganz bestimmt *nicht* das leere Wort.

Induktionsvoraussetzung: für ein beliebiges aber festes $n \in \mathbb{N}_0$ gelte: Für alle Wörter $w \in Z^n$ ist $\text{Num}_{10}(w)$ eindeutig definiert.

Induktionsschritt $n \rightsquigarrow n + 1$: Nun müssen wir beweisen, dass die Richtigkeit der Aussage des Lemmas für ein n zwingend auch die Richtigkeit der Aussage für $n + 1$ nach sich zieht.

Bezeichne w' ein beliebiges Wort der Länge $n + 1$. Wir müssen zeigen: $\text{Num}_{10}(w')$ ist eindeutig festgelegt. Das geht so:

Wenn $|w'| = n + 1$, dann enthält w' mindestens ein Symbol, also auch ein letztes. Bezeichnen wir das mit x . Dann ist w' von der Form $w' = wx$ mit $w \in Z^n$ und $x \in Z$.

In der Definition von Num_{10} passt *nur* Gleichung 2:

$$\text{Num}_{10}(w') = \text{Num}_{10}(wx) = 10 \cdot \text{Num}_{10}(w) + \text{num}_{10}(x)$$

Nach IV ist $\text{Num}_{10}(w)$ eindeutig festgelegt, also auch $\text{Num}_{10}(w')$.

6.1.2 Andere Zahldarstellungen

Beispielrechnungen

- klar machen, wie allgemein bei Basis k die Umwandlung funktioniert: $\text{Num}_k(wx) = k \cdot \text{Num}_k(w) + \text{num}_k(x)$.

- Beispiel rechnen, z.B. $\text{Num}_3(\mathbf{111}) = \dots = 13$.

- $\text{Num}_2(\mathbf{1}) = 1$, $\text{Num}_2(\mathbf{11}) = 3$, $\text{Num}_2(\mathbf{111}) = 7$, $\text{Num}_2(\mathbf{1111}) = 15$,

Wer sieht allgemein: $\forall m \in \mathbb{N}_0 : \text{Num}_2(\mathbf{1}^m) = 2^m - 1$?

Wie überträgt sich das auf den Fall $k = 3$? $\forall m \in \mathbb{N}_0 : \text{Num}_3(\mathbf{2}^m) = 3^m - 1$.

Algorithmus für Umwandlung von Binärdarstellung nach Zahl erarbeiten:

```
// Eingabe:  $w \in Z_2^*$ 
 $x \leftarrow 0$ 
for  $i \leftarrow 0$  to  $|w| - 1$  do
     $x \leftarrow 2x + \text{num}_2(w(i))$ 
od
// am Ende:  $x = \text{Num}_2(w)$ 
```

Die Schleifeninvariante sieht man besser bei

```
// Eingabe:  $w \in Z_2^*$ 
 $x \leftarrow 0$ 
 $v \leftarrow \varepsilon$ 
for  $i \leftarrow 0$  to  $|w| - 1$  do
     $v \leftarrow v \cdot w(i)$ 
     $x \leftarrow 2x + \text{num}_2(w(i))$ 
od
// am Ende:  $v = w \wedge x = \text{Num}_2(w)$ 
```

Suchen lassen: $x = \text{Num}_2(v)$

Dass das eine Schleifeninvariante ist, nicht in allen Details beweisen. Aber den Kern erkennen: laut Definition von Num_2 ist nämlich

$$\text{Num}_2(v \cdot w(i)) = 2\text{Num}_2(v) + \text{num}_2(w(i))$$

6.2 Von einem Alphabet zum anderen

6.2.1 Übersetzungen

Warum macht man Übersetzungen? Fällt jemandem noch was ein außer

- Lesbarkeit
- Kompression
- Verschlüsselung
- Fehlererkennung und Fehlerkorrektur

6.2.2 Ein Beispiel: Übersetzung von Zahldarstellungen

6.2.3 Homomorphismen

- Beispiel:
 - $h(a) = 001$ und $h(b) = 1101$
 - dann ist $h(\mathbf{bba}) = h(\mathbf{b})h(\mathbf{b})h(\mathbf{a}) = 1101 \cdot 1101 \cdot 001 = 11011101001$
- ε -freier Homomorphismus: Warum will man das? Sonst geht „Information verloren“. keine Codierung mehr; Betrachte

- $h(a) = 001$ und $h(b) = \varepsilon$
- angenommen $h(w) = 001$ Was war dann w ? Man weiß nur: es kam genau ein a vor, aber wieviele b und wo ist nicht klar.

- Information kann aber auch anders verloren gehen;

- z. B. $h(a) = 0$, $h(b) = 1$, $h(c) = 10$ oder
- $h(a) = h(b)$, oder ...

allgemeine Formalisierung von „Information geht verloren“ suchen lassen: es gibt Wörter $w_1 \neq w_2$ (verschiedene!) mit $h^{**}(w_1) = h^{**}(w_2)$

- präfixfreier Code: für keine zwei verschiedenen Symbole $x_1, x_2 \in A$ gilt: $h(x_1)$ ist ein Präfix von $h(x_2)$.

Beispiel

- $h(a) = 001$ und $h(b) = 1101$ ist präfixfrei
- $h(a) = 01$ und $h(b) = 011$ ist nicht präfixfrei
- Präfixfreiheit leicht zu sehen, wenn alle $h(x)$ gleich lang sind: präfixfrei \iff injektiv; Beispiel: ASCII

6.2.4 Beispiel Unicode: UTF-8 Codierung

- Man könnte, wenn die Zeit reicht, ja mal für ein paar Zeichen die UTF-8 Codierung bestimmen. Zum Beispiel gibt es für π in Unicode ein Zeichen, nämlich das mit der Nummer **0x03C0**.

Wenn ich den Algorithmus richtig gemacht habe, ergibt sich

- Code Point **0x03C0**
- in Bits **0000 0011 1100 0000 = 00000 01111 000000**
- man benutzt die Zeile

Char. number range (hexadecimal)	UTF-8 octet sequence (binary)
0000 0080 - 0000 07FF	110xxxxx 10xxxxxx

- also UTF-8 Codierung **11001111 10000000**

- Man mache sich klar, dass UTF-8 präfixfrei ist.

6.3 Huffman-Codierung

6.3.1 Algorithmus zur Berechnung von Huffman-Codes

Wie nehmen acht Symbole: **a**, **b**, **c**, **d**, **e**, **f**, **g**, **h**

- 1. Fall: Jedes Zeichen kommt genau einmal vor.
Huffman-Code-Baum erstellen, Wort **badcfegh** codieren, wie lang wird die Codierung?
- 2. Fall: **a** kommt einmal vor, **b** zweimal, **c** 4-mal, **d** 8-mal, **e** 16-mal, **f** 32-mal, **g** 64-mal, **h** 128-mal.

Huffman-Code-Baum erstellen,

Ergebnis z. B.

x	a	b	c	d	e	f	g	h
$h(x)$	0000000	0000001	000001	00001	0001	001	01	1

aber nicht das Wort **abbccccc...h**¹²⁸ codieren, sondern das Wort **badcfegh**;
wie lang wird die Codierung? ... über 4 mal so lang

- Wie lange wird ein Wort mit zweiter Zeichenverteilung, wenn man es mit dem ersten Code codiert?
- Wie lange wird ein Wort mit erster Zeichenverteilung, wenn man es mit dem zweiten Code codiert?

Dreimal so lang, weil jeder Buchstabe durch 3 Bits codiert wird.

Ziel: Sehen, dass Huffman-Codierung irgendwie zu funktionieren scheint.

6.3.2 Weiteres zu Huffman-Codes

Blockcodierung mit Huffman

- Verallgemeinerung: nicht von den Häufigkeiten einzelner Symbole ausgehen, sondern für Teilwörter einer festen Länge $b > 1$ die Häufigkeiten berechnen.
- Beispiel: Man hat einen Text über dem Alphabet $\{a, b, c, d\}$, der nur aus Teilwörtern der Länge 10 zusammengesetzt ist, die denen in jedem immer nur ein Symbol vorkommt, also z. B. **aaaaaaaaabbbbbbbbbbcccccccccc** usw. Angenommen **a**¹⁰, ..., **d**¹⁰ kommen alle gleich häufig vor. Wie lang ist dann die Huffman-Codierung? Ein Fünftel, weil jeder Zehnerblock durch zwei Bits codiert wird.