

# Grundbegriffe der Informatik

## Einheit 9: Speicher

Prof. Dr. Tanja Schultz

Karlsruher Institut für Technologie, Fakultät für Informatik

Wintersemester 2011/2012

## Speicher

- Bit und Byte

- Speicher als Tabellen und Abbildungen

- Binäre und dezimale Größenpräfixe

## Speicher

- Bit und Byte

- Speicher als Tabellen und Abbildungen

- Binäre und dezimale Größenpräfixe

## Speicher

Bit und Byte

Speicher als Tabellen und Abbildungen

Binäre und dezimale Größenpräfixe

- ▶ Das Wort „Bit“ hat verschiedene Bedeutungen.
  - ▶ Bezeichnung für eine Binärziffer (binary digit):  
Ein Bit ist ein Zeichen des Alphabetes {0, 1}.
  - ▶ Maßeinheit für die Datenmenge bei digitaler Speicherung oder Übertragung von Daten
  - ▶ Maßeinheit für den Informationsgehalt (Shannon)  
(siehe Vorlesung „Theoretische Grundlagen“ im 3. Semester)
- ▶ Byte: ein Wort aus acht Bits - genauer Oktett - Octet
- ▶ Historisch Byte = Anzahl Bits zur Kodierung eines einzigen Schriftzeichens in einem Computer, bzw. kleinste adressierbare Datenmenge eines technischen Systems (ASCII - 7 bit; IBM-PC - 8 bit; Nixdorf 820 - 12 bit ...)
- ▶ Abkürzungen
  - ▶ für Bit: möglichst „bit“, denn „b“ ist Flächeneinheit „barn“
  - ▶ für Byte: meist „B“, aber „B“ auch „Bel“, „deziBel“ - dB
  - ▶ für Octet: „o“

## Speicher

Bit und Byte

Speicher als Tabellen und Abbildungen

Binäre und dezimale Größenpräfixe

- ▶ Formalisierung von
  - ▶ Speicher
  - ▶ Lesen aus dem Speicher
  - ▶ Schreiben in den Speicher
- ▶ Diskussion, wozu diese Formalisierungen

- ▶ aktueller Gesamtzustand eines Speichers:
  - ▶ für jede Adresse, zu der etwas gespeichert ist:
  - ▶ welcher Wert ist unter dieser Adresse abgelegt
- ▶ Vorstellung: Tabelle mit zwei Spalten:
  - ▶ links alle Adressen
  - ▶ rechts die zugehörigen Werte

allgemein		Halbleiterspeicher	
Adresse 1	Wert 1	000	10110101
Adresse 2	Wert 2	001	10101101
Adresse 3	Wert 3	010	10011101
Adresse 4	Wert 4	011	01110110
		100	00111110
:	:	101	10101101
		110	00101011
Adresse $n$	Wert $n$	111	10101001



- ▶ aktueller Gesamtzustand eines Speichers:
  - ▶ für jede Adresse, zu der etwas gespeichert ist:
  - ▶ welcher Wert ist unter dieser Adresse abgelegt
- ▶ Vorstellung: Tabelle mit zwei Spalten:
  - ▶ links alle Adressen
  - ▶ rechts die zugehörigen Werte

allgemein		Halbleiterspeicher	
Adresse 1	Wert 1	000	10110101
Adresse 2	Wert 2	001	10101101
Adresse 3	Wert 3	010	10011101
Adresse 4	Wert 4	011	01110110
		100	00111110
:	:	101	10101101
		110	00101011
Adresse $n$	Wert $n$	111	10101001

- ▶ Tabelle: Abbildung von Adressen auf Werte, also  
Definitionsbereich: Menge aller Adressen, Wertebereich:  
Menge aller speicherbaren Werte

$$m : \text{Adr} \rightarrow \text{Val}$$

- ▶ Z. B. Halbleiterspeicher in einem PC mit „4 Gigabyte“:

$$m : \{0, 1\}^{32} \rightarrow \{0, 1\}^8$$

- ▶ Speicher im Zustand  $m$  hat an Adresse  $a \in \text{Adr}$   
( $2^{32} = 4294967296$ ) den Wert  $m(a) \in \text{Val}$  ( $2^8 = 256$ )  
gespeichert.
- ▶ bei Hauptspeicher:
  - ▶ Menge der Adressen ist fest
  - ▶ Adresse bezeichnet einen physikalischen Ort auf dem Chip
  - ▶ entspricht einer Angabe auf einem Brief, etwa wie  
„Adenauerring 4, 76131 Karlsruhe“

- ▶ Formalisierung als Abbildung `memread`
  - ▶ Argumente:
    - ▶ der gesamte Speicherinhalt  $m$  des Speichers und
    - ▶ die Adresse  $a$  aus der ausgelesen wird
  - ▶ Resultat: der in  $m$  an Adresse  $a$  gespeicherte Wert, d.h.

$$\begin{aligned}\text{memread} : \text{Mem} \times \text{Adr} &\rightarrow \text{Val} \\ (m, a) &\mapsto m(a)\end{aligned}$$

- ▶ Dabei sei  $\text{Mem}$  die Menge aller möglichen Speicherzustände, also die Menge aller Abbildungen von  $\text{Adr}$  nach  $\text{Val}$ .
- ▶ auf das „Warum überhaupt so eine Formalisierung?“ kommen wir noch zu sprechen

- ▶ nicht verwirren lassen:
  - ▶ Funktion memread bekommt als Argument eine (andere) Funktion  $m : \text{Adr} \rightarrow \text{Val}$
  - ▶ Beispiel zeigt: kein Problem; man denke an Tabellen
- ▶ die Menge aller Abbildungen der Form  $f : A \rightarrow B$ 
  - ▶ so etwas kommt noch öfter vor
  - ▶ Notation:  $B^A$
  - ▶ beachte Reihenfolge
  - ▶ für endliche Mengen  $A$  und  $B$  gilt:  $|B^A| = |B|^{|A|}$ .
- ▶ hätten also auch schreiben können:
  - ▶  $\text{Mem} = \text{Val}^{\text{Adr}}$  und
  - ▶  $\text{memread} : \text{Val}^{\text{Adr}} \times \text{Adr} \rightarrow \text{Val}$   
 $(m, a) \mapsto m(a)$

Das Speichern eines neuen Wertes  $v$  an eine Adresse  $a$  in einem Speicher  $m$  kann man auch als Funktion notieren, sieht ein wenig komplizierter aus als beim Lesen aus dem Speicher:

- ▶  $\text{memwrite} : \text{Val}^{\text{Adr}} \times \text{Adr} \times \text{Val} \rightarrow \text{Val}^{\text{Adr}}$

$$(m, a, v) \mapsto m'$$

- ▶ wobei  $m'$  festgelegt durch die Forderung, dass für alle  $a' \in \text{Adr}$  gilt:

$$m'(a') = \begin{cases} v & \text{falls } a' = a \\ m(a') & \text{falls } a' \neq a \end{cases}$$

- ▶ d.h. Speicher  $m'$  enthält an Adresse  $a$  den Wert  $v$
- ▶ Das klingt plausibel ...
  - ▶ für Hauptspeicher ist es das auch,
  - ▶ aber es gibt auch Speicher die anders arbeiten, z. B. sogenannte Cache-Speicher (siehe VL Technische Informatik)

- ▶ Was ist „das wesentliche“ an Speicher?

- ▶ Der allerwichtigste Aspekt überhaupt:

Für alle  $m \in \text{Mem}$ ,  $a, a' \in \text{Adr}$  mit  $a' \neq a$  und  $v' \in \text{Val}$  gilt:

$$\text{memread}(\text{memwrite}(m, a, v), a) = v$$

- ▶ d.h. wenn man Wert  $v$  an Adresse  $a$  in einen Speicher  $m$  schreibt und danach diesen Speicher liest, dann ist der Wert  $v$ .
- ▶ Reicht das als Spezifikation für Speicher aus? - Nicht für alle Speicher (z.B. Cache)
- ▶ Für die oben definierten Funktionen gilt nämlich auch:  
Für alle  $m \in \text{Mem}$ ,  $a, a' \in \text{Adr}$  mit  $a' \neq a$  und  $v' \in \text{Val}$  :

$$\text{memread}(m, a) = \text{memread}(\text{memwrite}(m, a', v'), a)$$

- ▶ d.h. das Ergebnis des Lesens einer Speicherstelle hängt nicht davon ab, was vorher an einer anderen Adresse gespeichert wurde.

# Wozu diese Formalisierungen?

- ▶ Die Gleichungen sagen etwas darüber aus, wie „sich der Speicher verhalten“ soll
- ▶ d.h. sie liefern eine Möglichkeit für den *Spezifizierer* von Speicher, auszudrücken
  - ▶ „wie sich Speicher verhalten soll“
  - ▶ algebraische Spezifikation (in späteren VLs lernen Sie noch komplexere Formen der algebraischen Spezifikation von Datentypen kennen)
- ▶ auch eine Möglichkeit für den *Implementierer*, sich über Testfälle klar zu werden
  - ▶ Erstellen von Testfällen, Testdurchführung
  - ▶ Testen kann nicht Korrektheit einer Implementierung beweisen,
  - ▶ aber immerhin, dass sie falsch ist.

## Das sollten Sie mitnehmen:

- ▶ es gibt nicht nur Hauptspeicher
- ▶ Adressen sind manchmal etwas anderes als „physikalische Koordinaten“
- ▶ Abbildungen kann man sich manchmal am besten als Tabelle vorstellen

## Das sollten Sie üben:

- ▶ Gewöhnung an Abbildungen, die Abbildungen auf Abbildungen abbilden



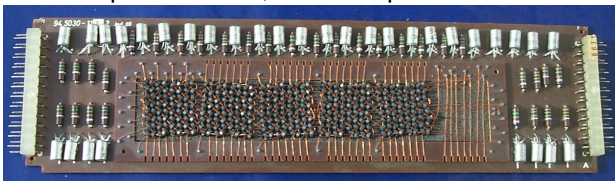
## Speicher

Bit und Byte

Speicher als Tabellen und Abbildungen

Binäre und dezimale Größenpräfixe

- ▶ früher: Speicher klein, z. B. ein paar Hundert Bits



Bildquelle: <http://de.wikipedia.org/w/index.php?title=Bild:Kernspeicher1.jpg>

- ▶ heute: groß, z. B.
  - ▶ Hauptspeicher:  $2^{32} = 4\,294\,967\,296$  Bytes
  - ▶ Festplatten: so was wie 1 000 000 000 000 Bytes
- ▶ Zahlen nur noch schlecht zu lesen
- ▶ Benutzung von **Präfixen** für kompaktere Notation:  
**Kilometer**, **Mikrosekunde**, **Megawatt**, usw.

$10^{-3}$	$10^{-6}$	$10^{-9}$	$10^{-12}$	$10^{-15}$	$10^{-18}$
$1000^{-1}$	$1000^{-2}$	$1000^{-3}$	$1000^{-4}$	$1000^{-5}$	$1000^{-6}$
<b>milli</b>	<b>mikro</b>	<b>nano</b>	<b>pico</b>	<b>femto</b>	<b>atto</b>
m	$\mu$	n	p	f	a
$10^3$	$10^6$	$10^9$	$10^{12}$	$10^{15}$	$10^{18}$
$1000^1$	$1000^2$	$1000^3$	$1000^4$	$1000^5$	$1000^6$
<b>kilo</b>	<b>mega</b>	<b>giga</b>	<b>tera</b>	<b>peta</b>	<b>exa</b>
k	M	G	T	P	E

- ▶ In Rechnern häufig Größen, die Potenzen von 2 oder  $2^{10}$  sind, und *nicht* Potenzen von 10 bzw. 1000.
- ▶ Die *International Electrotechnical Commission* hat 1999 Präfixe eingeführt, die für Potenzen von  $2^{10} = 1024$  stehen.
- ▶ motiviert durch Kunstworte „kilobinary“, „megabinary“, usw.
- ▶ Präfixe *kibi*, *mebi*, *gibi*, usw.
- ▶ abgekürzt Ki, Mi, Gi, usw.

$2^{10}$	$2^{20}$	$2^{30}$	$2^{40}$	$2^{50}$	$2^{60}$
$1024^1$	$1024^2$	$1024^3$	$1024^4$	$1024^5$	$1024^6$
kibi	mebi	gibi	tebi	pebi	exbi
Ki	Mi	Gi	Ti	Pi	Ei

Das sollten Sie mitnehmen:

- ▶ Bit und Byte / Oktett, Octet
- ▶ binäre Größenpräfixe

Das sollten Sie üben:

- ▶ Rechnen mit binären Größenpräfixen

# Grundbegriffe der Informatik

## Einheit 10: Codierungen

Prof. Dr. Tanja Schultz

Karlsruher Institut für Technologie, Fakultät für Informatik

Wintersemester 2011/2012

## Codierungen

### Von Wörtern zu Zahlen und zurück

- Dezimaldarstellung von Zahlen

- Andere Zahldarstellungen

- Von Zahlen zu ihren Darstellungen

### Von einem Alphabet zum anderen

- Ein Beispiel: Übersetzung von Zahldarstellungen

- Homomorphismen

- Beispiel Unicode: UTF-8

### Huffman-Codierung

- Algorithmus zur Berechnung von Huffman-Codes

- Weiteres zu Huffman-Codes

## Codierungen

### Von Wörtern zu Zahlen und zurück

- Dezimaldarstellung von Zahlen

- Andere Zahldarstellungen

- Von Zahlen zu ihren Darstellungen

### Von einem Alphabet zum anderen

- Ein Beispiel: Übersetzung von Zahldarstellungen

- Homomorphismen

- Beispiel Unicode: UTF-8

### Huffman-Codierung

- Algorithmus zur Berechnung von Huffman-Codes

- Weiteres zu Huffman-Codes



- ▶ aus Indien
- ▶ Ziffern des Alphabetes  $Z_{10} = \{0, \dots, 9\}$ .
- ▶ Bedeutung  $\text{num}_{10}(x)$  einer einzelnen Ziffer  $x$  als Zahl:

$x$	0	1	2	...	8	9
$\text{num}_{10}(x)$	<i>null</i>	<i>eins</i>	<i>zwei</i>	...	<i>acht</i>	<i>neun</i>

- ▶ aus Indien
- ▶ Ziffern des Alphabetes  $Z_{10} = \{0, \dots, 9\}$ .
- ▶ Bedeutung  $\text{num}_{10}(x)$  einer einzelnen Ziffer  $x$  als Zahl:

---

$x$	0	1	2	3	4	5	6	7	8	9
$\text{num}_{10}(x)$	0	1	2	3	4	5	6	7	8	9

---

- ▶ Bedeutung eines Wortes  $x_{k-1} \cdots x_0 \in Z_{10}^*$  von Ziffern

$$\text{Num}_{10} : Z_{10}^* \rightarrow \mathbb{N}_0$$

- ▶ aus Indien
- ▶ Ziffern des Alphabetes  $Z_{10} = \{0, \dots, 9\}$ .
- ▶ Bedeutung  $\text{num}_{10}(x)$  einer einzelnen Ziffer  $x$  als Zahl:

$x$	0	1	2	3	4	5	6	7	8	9
$\text{num}_{10}(x)$	0	1	2	3	4	5	6	7	8	9

- ▶ Bedeutung eines Wortes  $x_{k-1} \cdots x_0 \in Z_{10}^*$  von Ziffern

$$\text{Num}_{10} : Z_{10}^* \rightarrow \mathbb{N}_0$$

- ▶ Beispiel aus der Schule:  $164 = 10^2 \cdot 1 + 10^1 \cdot 6 + 10^0 \cdot 4$ ,  $k = 3$   
 $10^{k-1} \cdot \text{num}_{10}(x_{k-1}) + \cdots + 10^1 \cdot \text{num}_{10}(x_1) + 10^0 \cdot \text{num}_{10}(x_0)$
- ▶ Pünktchenvermeidung:

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

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

- ▶ geboren 1. Juli 1646 in Leipzig  
gestorben am 14. November 1716 in Hannover
- ▶ baute erste Maschine, die zwei Zahlen multiplizieren konnte
- ▶ in einem Brief vom 2. Januar 1697 an den Herzog von Braunschweig-Wolfenbüttel:
  - ▶ auch wenn man nur zwei Ziffern 0 und 1 hat, kann man nichtnegative Zahlen darstellen und vernünftig rechnen

Pour l'Addition  
par exemple. ☞

$\begin{array}{r} 110 \\ 111 \\ \hline 1101 \end{array} \Bigg  \begin{array}{l} 6 \\ 7 \\ 13 \end{array}$	$\begin{array}{r} 101 \\ 1011 \\ \hline 10000 \end{array} \Bigg  \begin{array}{l} 5 \\ 11 \\ 16 \end{array}$	$\begin{array}{r} 1110 \\ 10001 \\ \hline 11111 \end{array} \Bigg  \begin{array}{l} 14 \\ 17 \\ 31 \end{array}$
---	--	---

Pour la Sou-  
straction.

$\begin{array}{r} 1101 \\ 111 \\ \hline 110 \end{array} \Bigg  \begin{array}{l} 13 \\ 7 \\ 6 \end{array}$	$\begin{array}{r} 10000 \\ 1011 \\ \hline 101 \end{array} \Bigg  \begin{array}{l} 16 \\ 11 \\ 5 \end{array}$	$\begin{array}{r} 11111 \\ 10001 \\ \hline 1110 \end{array} \Bigg  \begin{array}{l} 31 \\ 17 \\ 14 \end{array}$
---	--	---

Bildquelle [http://commons.wikimedia.org/wiki/Image:Leibniz\\_binary\\_system\\_1703.png](http://commons.wikimedia.org/wiki/Image:Leibniz_binary_system_1703.png)

- ▶ Ziffernmenge  $Z_2 = \{0, 1\}$
- ▶ definiere:  $\text{num}_2(0) = 0$  und  $\text{num}_2(1) = 1$  und

$$\text{Num}_2(\varepsilon) = 0$$

$$\forall w \in Z_2^* \quad \forall x \in Z_2 : \text{Num}_2(wx) = 2 \cdot \text{Num}_2(w) + \text{num}_2(x)$$

- ▶ Z. B.:

$$\begin{aligned}\text{Num}_2(\mathbf{1101}) &= 2 \cdot \text{Num}_2(\mathbf{110}) + 1 \\&= 2 \cdot (2 \cdot \text{Num}_2(\mathbf{11}) + 0) + 1 \\&= 2 \cdot (2 \cdot (2 \cdot \text{Num}_2(\mathbf{1}) + 1) + 0) + 1 \\&= 2 \cdot (2 \cdot (2 \cdot (2 \cdot \text{Num}_2(\varepsilon) + 1) + 1) + 0) + 1 \\&= 2 \cdot (2 \cdot (2 \cdot (2 \cdot 0 + 1) + 1) + 0) + 1 \\&= 2^3 \cdot 1 + 2^2 \cdot 1 + 2^1 \cdot 0 + 2^0 \cdot 1 \\&= 13\end{aligned}$$

- ▶ Ziffern  $Z_{16} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$ .
- ▶ Ziffernwerte:

$x$	0	1	2	3	4	5	6	7
$\text{num}_{16}(x)$	0	1	2	3	4	5	6	7

---

$x$	8	9	A	B	C	D	E	F
$\text{num}_{16}(x)$	8	9	10	11	12	13	14	15

- ▶ Zuordnung von Wörtern zu Zahlen gegeben durch

$$\text{Num}_{16}(\varepsilon) = 0$$

$$\forall w \in Z_{16}^* \quad \forall x \in Z_{16} : \text{Num}_{16}(wx) = 16 \cdot \text{Num}_{16}(w) + \text{num}_{16}(x)$$

- ▶ die Alphabete  $Z_2$ ,  $Z_3$ , usw. sind nicht disjunkt
- ▶ Darstellungen mehrdeutig; Beispiel 111:
  - $\text{Num}_2(111)$  die Zahl sieben,
  - $\text{Num}_8(111)$  die Zahl dreiundsiebzig,
  - $\text{Num}_{10}(111)$  die Zahl einhundertelf und
  - $\text{Num}_{16}(111)$  die Zahl zweihundertdreiundsiebzig.
- ▶ in manchen Programmiersprachen
  - ▶ Präfix 0b für Darstellungen zur Basis 2
  - ▶ Präfix 0o für Darstellungen zur Basis 8
  - ▶ Präfix 0x für Darstellungen zur Basis 16

- ▶ Man kann zu  $w \in Z_k^*$  die repräsentierte Zahl  $\text{Num}_k(w)$  berechnen.
- ▶ Umgekehrt geht es auch:  
man kann zu  $n \in \mathbb{N}_0$  die sogenannte  **$k$ -äre Darstellung** (also Wort  $w \in Z_k^*$ ) berechnen
- ▶ Alphabet  $Z_k$  mit  $k$  Ziffern
  - ▶ Bedeutung: die Zahlen in  $\mathbb{G}_k$
  - ▶ für  $i \in \mathbb{G}_k$  sei  $\text{repr}_k(i)$  das entsprechende Zeichen
  - ▶  $\text{repr}_k$  ist also gerade die Umkehrfunktion zu  $\text{num}_k$
- ▶ Gesucht: eine Repräsentation von  $n \in \mathbb{N}_0$  als Wort  $w \in Z_k^*$  mit der Eigenschaft  $\text{Num}_k(w) = n$ 
  - ▶ für die naheliegende Definition von  $\text{Num}_k$
- ▶ gibt es immer unendlich viele passende  $w$ :  
wenn  $\text{Num}_k(w) = n$ , dann auch  $\text{Num}_k(0w) = n$   
(Beweis durch Induktion!)



// Eingabe:  $n \in \mathbb{N}_0$

$y \leftarrow n$

$w \leftarrow \varepsilon$

$m \leftarrow \begin{cases} 1 + \lfloor \log_k(n) \rfloor & \text{falls } n > 0 \\ 1 & \text{falls } n = 0 \end{cases}$

**for**  $i \leftarrow 0$  **to**  $m - 1$  **do**

$r \leftarrow y \bmod k$

$w \leftarrow \text{repr}_k(r) \cdot w$       // Konkatination

$y \leftarrow y \text{ div } k$

**od**

// am Ende:  $n = \text{Num}_k(w)$

- ▶ Nehme  $k = 10$  und  $n = 4711$ , dann ist  $m = 4$
- ▶ Für jedes  $i \in \mathbb{G}_5$  haben die Variablen  $r$ ,  $w$  und  $y$  nach  $i$  Schleifendurchläufen die folgenden Werte

$i$	4	3	2	1	0
$r$	4	7	1	1	
$w$	4711	711	11	1	$\varepsilon$
$y$	0	4	47	471	4711

- ▶ (  $r \leftarrow y \bmod k$  ,  $w \leftarrow \text{repr}_k(r) \cdot w$  und  $y \leftarrow y \text{ div } k$  )
- ▶ Schleifeninvariante  $y \cdot 10^i + \text{Num}_k(w) = n$  drängt sich auf
- ▶ Wenn man weiß, dass am Ende  $y = 0$  ist, ist man fertig.

- ▶ Algorithmus liefert kürzestes Wort  $w \in Z_k^*$  mit  $\text{Num}_k(w) = n$ .
- ▶ Wir schreiben dafür  $\text{Repr}_k(n)$ .
- ▶ Es gilt stets

$$\text{Num}_k(\text{Repr}_k(n)) = n .$$

- ▶ Beachte:
  - ▶ Die Umkehrung gilt im allgemeinen nicht

$$\text{Repr}_k(\text{Num}_k(w)) \neq w$$

- ▶ weil „überflüssige“ führende Nullen wegfallen, bspw. für  $w = 004711$  gilt  $\text{Num}_k(004711) = \text{Num}_k(4711)$

## noch ein Beispiel (1)

- ▶ Ziffernmenge  $Z = \{\bar{1}, 0, 1\}$
- ▶ definiere

$$\text{num}(\bar{1}) = -1$$

$$\text{num}(0) = 0$$

$$\text{num}(1) = 1$$

- ▶ definiere  $\text{Num} : Z^* \rightarrow \mathbb{Z}$

$$\text{Num}(\varepsilon) = 0$$

$$\forall w \in Z^* \quad \forall x \in Z : \text{Num}(wx) = 3 \cdot \text{Num}(w) + \text{num}(x)$$

- ▶ Z. B.:  $\text{Num}(\bar{1}01) = -3^2 + 0 + 3^0 = -8$  und  
 $\text{Num}(1\bar{1}01) = +3^3 - 3^2 + 0 + 3^0 = 19$

- ▶  $\text{num}(\mathbf{1}) = -1$ ,  $\text{num}(\mathbf{0}) = 0$  und  $\text{num}(\mathbf{1}) = 1$  und

$$\text{Num}(\varepsilon) = 0$$

$$\forall w \in Z^* \quad \forall x \in Z : \text{Num}(wx) = 3 \cdot \text{Num}(w) + \text{num}(x)$$

- ▶ definiere  $\text{inv} : Z^* \rightarrow Z^*$

- ▶  $\text{inv}(\mathbf{1}) = \mathbf{1}$ ,  $\text{inv}(\mathbf{1}) = \mathbf{1}$  und  $\text{inv}(\mathbf{0}) = \mathbf{0}$ .

- ▶  $\text{inv}(\varepsilon) = \varepsilon$  und  $\forall w \in Z^* : \forall x \in Z : \text{inv}(wx) = \text{inv}(w)\text{inv}(x)$

also z. B.  $\text{inv}(\mathbf{1101}) = \mathbf{1101}$

- ▶ dann gilt für alle  $w \in Z^*$ :  $\text{Num}(\text{inv}(w)) = -\text{Num}(w)$ .
- ▶  $(1 + 1 = 3 - 1 = \mathbf{11}; -1 - 1 = -3 + 1 = \mathbf{11}; 0 + 1 = 1)$

## noch ein Beispiel (2)

- ▶  $\text{num}(\overline{1}) = -1$ ,  $\text{num}(0) = 0$  und  $\text{num}(1) = 1$  und

$$\text{Num}(\varepsilon) = 0$$

$$\forall w \in Z^* \quad \forall x \in Z : \text{Num}(wx) = 3 \cdot \text{Num}(w) + \text{num}(x)$$

- ▶ definiere  $\text{inv} : Z^* \rightarrow Z^*$

- ▶  $\text{inv}(1) = \overline{1}$ ,  $\text{inv}(\overline{1}) = 1$  und  $\text{inv}(0) = 0$ .

- ▶  $\text{inv}(\varepsilon) = \varepsilon$  und  $\forall w \in Z^* : \forall x \in Z : \text{inv}(wx) = \text{inv}(w)\text{inv}(x)$

also z. B.  $\text{inv}(1\overline{1}01) = \overline{1}10\overline{1}$

- ▶ dann gilt für alle  $w \in Z^*$ :  $\text{Num}(\text{inv}(w)) = -\text{Num}(w)$ .
- ▶ „Addition“ wie „in der Schule“:

$$\begin{array}{rcccc} 1 & \overline{1} & 0 & 1 \\ & \overline{1} & 0 & 1 \\ \hline \end{array}$$

- ▶  $(1 + 1 = 3 - 1 = 1\overline{1}; -1 - 1 = -3 + 1 = \overline{1}1; 0 + 1 = 1)$

## noch ein Beispiel (2)

- ▶  $\text{num}(\overline{1}) = -1$ ,  $\text{num}(0) = 0$  und  $\text{num}(1) = 1$  und

$$\text{Num}(\varepsilon) = 0$$

$$\forall w \in Z^* \quad \forall x \in Z : \text{Num}(wx) = 3 \cdot \text{Num}(w) + \text{num}(x)$$

- ▶ definiere  $\text{inv} : Z^* \rightarrow Z^*$

- ▶  $\text{inv}(1) = \overline{1}$ ,  $\text{inv}(\overline{1}) = 1$  und  $\text{inv}(0) = 0$ .

- ▶  $\text{inv}(\varepsilon) = \varepsilon$  und  $\forall w \in Z^* : \forall x \in Z : \text{inv}(wx) = \text{inv}(w)\text{inv}(x)$

also z. B.  $\text{inv}(1\overline{1}01) = \overline{1}10\overline{1}$

- ▶ dann gilt für alle  $w \in Z^*$ :  $\text{Num}(\text{inv}(w)) = -\text{Num}(w)$ .
- ▶ „Addition“ wie „in der Schule“:

$$\begin{array}{rcccc} 1 & \overline{1} & 0 & 1 \\ & \overline{1} & 0 & 1 \\ & & 1 & \\ \hline & & & \overline{1} \end{array}$$

- ▶  $(1 + 1 = 3 - 1 = 1\overline{1}; -1 - 1 = -3 + 1 = \overline{1}1; 0 + 1 = 1)$

## noch ein Beispiel (2)

- ▶  $\text{num}(\overline{1}) = -1$ ,  $\text{num}(0) = 0$  und  $\text{num}(1) = 1$  und

$$\text{Num}(\varepsilon) = 0$$

$$\forall w \in Z^* \quad \forall x \in Z : \text{Num}(wx) = 3 \cdot \text{Num}(w) + \text{num}(x)$$

- ▶ definiere  $\text{inv} : Z^* \rightarrow Z^*$

- ▶  $\text{inv}(1) = \overline{1}$ ,  $\text{inv}(\overline{1}) = 1$  und  $\text{inv}(0) = 0$ .

- ▶  $\text{inv}(\varepsilon) = \varepsilon$  und  $\forall w \in Z^* : \forall x \in Z : \text{inv}(wx) = \text{inv}(w)\text{inv}(x)$

also z. B.  $\text{inv}(1\overline{1}01) = \overline{1}10\overline{1}$

- ▶ dann gilt für alle  $w \in Z^*$ :  $\text{Num}(\text{inv}(w)) = -\text{Num}(w)$ .
- ▶ „Addition“ wie „in der Schule“:

$$\begin{array}{rcccc} 1 & \overline{1} & 0 & 1 \\ & \overline{1} & 0 & 1 \\ & 0 & 1 & \\ \hline & 1 & \overline{1} & \end{array}$$

- ▶  $(1 + 1 = 3 - 1 = 1\overline{1}; -1 - 1 = -3 + 1 = \overline{1}1; 0 + 1 = 1)$



## noch ein Beispiel (2)

- ▶  $\text{num}(\overline{1}) = -1$ ,  $\text{num}(0) = 0$  und  $\text{num}(1) = 1$  und

$$\text{Num}(\varepsilon) = 0$$

$$\forall w \in Z^* \quad \forall x \in Z : \text{Num}(wx) = 3 \cdot \text{Num}(w) + \text{num}(x)$$

- ▶ definiere  $\text{inv} : Z^* \rightarrow Z^*$

- ▶  $\text{inv}(1) = \overline{1}$ ,  $\text{inv}(\overline{1}) = 1$  und  $\text{inv}(0) = 0$ .

- ▶  $\text{inv}(\varepsilon) = \varepsilon$  und  $\forall w \in Z^* : \forall x \in Z : \text{inv}(wx) = \text{inv}(w)\text{inv}(x)$

also z. B.  $\text{inv}(1\overline{1}01) = \overline{1}10\overline{1}$

- ▶ dann gilt für alle  $w \in Z^*$ :  $\text{Num}(\text{inv}(w)) = -\text{Num}(w)$ .
- ▶ „Addition“ wie „in der Schule“:

$$\begin{array}{rcccc} 1 & \overline{1} & 0 & 1 \\ & \overline{1} & 0 & 1 \\ \overline{1} & 0 & 1 & \\ \hline 1 & 1 & \overline{1} & \end{array}$$

- ▶  $(1 + 1 = 3 - 1 = 1\overline{1}; -1 - 1 = -3 + 1 = \overline{1}1; 0 + 1 = 1)$

## noch ein Beispiel (2)

- ▶  $\text{num}(\overline{1}) = -1$ ,  $\text{num}(0) = 0$  und  $\text{num}(1) = 1$  und

$$\text{Num}(\varepsilon) = 0$$

$$\forall w \in Z^* \quad \forall x \in Z : \text{Num}(wx) = 3 \cdot \text{Num}(w) + \text{num}(x)$$

- ▶ definiere  $\text{inv} : Z^* \rightarrow Z^*$

- ▶  $\text{inv}(1) = \overline{1}$ ,  $\text{inv}(\overline{1}) = 1$  und  $\text{inv}(0) = 0$ .

- ▶  $\text{inv}(\varepsilon) = \varepsilon$  und  $\forall w \in Z^* : \forall x \in Z : \text{inv}(wx) = \text{inv}(w)\text{inv}(x)$

also z. B.  $\text{inv}(1\overline{1}01) = \overline{1}10\overline{1}$

- ▶ dann gilt für alle  $w \in Z^*$ :  $\text{Num}(\text{inv}(w)) = -\text{Num}(w)$ .
- ▶ „Addition“ wie „in der Schule“:

$$\begin{array}{rcccc} 1 & \overline{1} & 0 & 1 \\ & \overline{1} & 0 & 1 \\ \overline{1} & 0 & 1 & \\ \hline 0 & 1 & 1 & \overline{1} \end{array}$$

- ▶  $(1 + 1 = 3 - 1 = 1\overline{1}; -1 - 1 = -3 + 1 = \overline{1}1; 0 + 1 = 1)$

## noch ein Beispiel (2)

- ▶  $\text{num}(\overline{1}) = -1$ ,  $\text{num}(0) = 0$  und  $\text{num}(1) = 1$  und

$$\text{Num}(\varepsilon) = 0$$

$$\forall w \in Z^* \quad \forall x \in Z : \text{Num}(wx) = 3 \cdot \text{Num}(w) + \text{num}(x)$$

- ▶ definiere  $\text{inv} : Z^* \rightarrow Z^*$

- ▶  $\text{inv}(1) = \overline{1}$ ,  $\text{inv}(\overline{1}) = 1$  und  $\text{inv}(0) = 0$ .

- ▶  $\text{inv}(\varepsilon) = \varepsilon$  und  $\forall w \in Z^* : \forall x \in Z : \text{inv}(wx) = \text{inv}(w)\text{inv}(x)$

also z. B.  $\text{inv}(1\overline{1}01) = \overline{1}10\overline{1}$

- ▶ dann gilt für alle  $w \in Z^*$ :  $\text{Num}(\text{inv}(w)) = -\text{Num}(w)$ .
- ▶ „Addition“ wie „in der Schule“:

$$\begin{array}{rcccccl} 1 & \overline{1} & 0 & 1 & & 19 \\ & & \overline{1} & 0 & 1 & -8 \\ & \overline{1} & 0 & 1 & & \\ \hline 0 & 1 & 1 & \overline{1} & & 11 \end{array}$$

- ▶  $(1 + 1 = 3 - 1 = 1\overline{1}; -1 - 1 = -3 + 1 = \overline{1}1; 0 + 1 = 1)$

## Das sollten Sie mitnehmen:

- ▶ Umwandlungen zwischen Zahlen und Wörtern
- ▶ Mal wieder Schleifen
  - ▶ in passenden Beispielen ist Schleifeninvariante gut zu sehen
- ▶ schon Leibniz kannte die Binärdarstellung

## Das sollten Sie üben:

- ▶ Zahlen verschieden repräsentieren
- ▶ Algorithmen auch in Randfällen ausprobieren

## Codierungen

### Von Wörtern zu Zahlen und zurück

- Dezimaldarstellung von Zahlen

- Andere Zahldarstellungen

- Von Zahlen zu ihren Darstellungen

### Von einem Alphabet zum anderen

- Ein Beispiel: Übersetzung von Zahldarstellungen

- Homomorphismen

- Beispiel Unicode: UTF-8

### Huffman-Codierung

- Algorithmus zur Berechnung von Huffman-Codes

- Weiteres zu Huffman-Codes

- ▶ Betrachte die Funktion  $\text{Trans}_{2,16} = \text{Repr}_2 \circ \text{Num}_{16}$
- ▶  $\text{Trans}_{2,16} : Z_{16}^* \rightarrow Z_2^*$
- ▶ Z. B.

$$\begin{aligned}\text{Trans}_{2,16}(\text{A3}) &= \text{Repr}_2(\text{Num}_{16}(\text{A3})) \\ &= \text{Repr}_2(163) = 10100011\end{aligned}$$

- ▶ wesentlicher Punkt:
  - ▶ A3 und 10100011 haben *die gleiche Bedeutung*
  - ▶ nämlich die Zahl einhundertdreiundsechzig
- ▶ So etwas wollen wir eine **Übersetzung** nennen.

- ▶ Allgemein: Man schreibt Wörter aus einer formalen Sprache  $L_A \subseteq A^*$  und meint aber etwas anderes, ihre Bedeutung.
- ▶ Menge der Bedeutungen je nach Anwendungsfall sehr unterschiedlich
  - ▶ einfach: Zahlen,
  - ▶ kompliziert: Ausführung von Java-Programmen
  - ▶ im Folgenden schreiben wir einfach Sem für die Menge der „Bedeutungen“
- ▶ Gegeben:
  - ▶ zwei Alphabete  $A$  und  $B$
  - ▶  $L_A \subseteq A^*$  und  $L_B \subseteq B^*$
  - ▶ zwei Abbildungen  $\text{sem}_A : L_A \rightarrow \text{Sem}$  und  $\text{sem}_B : L_B \rightarrow \text{Sem}$
- ▶ Eine Abbildung  $f : L_A \rightarrow L_B$  heie eine **Übersetzung** bezüglich  $\text{sem}_A$  und  $\text{sem}_B$ , wenn  $f$  die Bedeutung erhält, d. h.

$$\forall w \in L_A : \text{sem}_A(w) = \text{sem}_B(f(w))$$

- ▶ Allgemein: Man schreibt Wörter aus einer formalen Sprache  $L_A \subseteq A^*$  und meint aber etwas anderes, ihre Bedeutung.
- ▶ Menge der Bedeutungen je nach Anwendungsfall sehr unterschiedlich
  - ▶ einfach: Zahlen,
  - ▶ kompliziert: Ausführung von Java-Programmen
  - ▶ im Folgenden schreiben wir einfach Sem für die Menge der „Bedeutungen“
- ▶ Gegeben:
  - ▶ zwei Alphabete  $A$  und  $B$
  - ▶  $L_A \subseteq A^*$  und  $L_B \subseteq B^*$
  - ▶ zwei Abbildungen  $\text{sem}_A : L_A \rightarrow \text{Sem}$  und  $\text{sem}_B : L_B \rightarrow \text{Sem}$
- ▶ Eine Abbildung  $f : L_A \rightarrow L_B$  heie eine **Übersetzung** bezüglich  $\text{sem}_A$  und  $\text{sem}_B$ , wenn  $f$  die Bedeutung erhält, d. h.

$$\forall w \in L_A : \text{sem}_A(w) = \text{sem}_B(f(w))$$



- ▶  $\text{Trans}_{2,16} = \text{Repr}_2 \circ \text{Num}_{16}$ .
- ▶ einfacher Fall:  $L_A = A^* = Z_2^*$  und  $L_B = B^* = Z_{16}^*$ .
- ▶ Bedeutungsfunktionen:  $\text{sem}_A = \text{Num}_{16}$  und  $\text{sem}_B = \text{Num}_2$
- ▶ Nachrechnen, dass  $\text{Trans}_{2,16}$  eine Übersetzung ist:

$$\begin{aligned}\text{sem}_B(f(w)) &= \text{Num}_2(\text{Trans}_{2,16}(w)) \\ &= \text{Num}_2(\text{Repr}_2(\text{Num}_{16}(w))) & * \\ &= \text{Num}_{16}(w) \\ &= \text{sem}_A(w)\end{aligned}$$

- ▶ \* weil wie bereits erwähnt:  $\text{Num}_k(\text{Repr}_k(n)) = n$

- ▶ Im allgemeinen kann die Menge der Bedeutungen Sem kompliziert sein
- ▶ Zum Beispiel bei der Übersetzung von Programmen
- ▶ Was ist Sem, wenn man ein Java-Programm in eine Folge von Bytecodes übersetzt?
  - ▶ siehe Vorlesungen zu „Semantik von Programmiersprachen“.

- ▶ **Legalität:** Manchmal ist ein bestimmter Zeichensatz vorgeschrieben (z. B. ASCII in Emails)
- ▶ **Lesbarkeit:** Übersetzungen evtl. kürzer und besser lesbar: vergleiche A3 oder 163 mit 10100011
- ▶ **Verschlüsselung:** gerade das Umgekehrte, d.h. Erschweren der Lesbarkeit
  - ▶ am liebsten gar keine Lesbarkeit, jedenfalls für Außenstehende
  - ▶ siehe Vorlesungen über Kryptographie
- ▶ **Kompression:** Übersetzungen können kürzer sein
  - ▶ und zwar *ohne* größeres Alphabet
  - ▶ das sehen wir gleich: Huffman-Codes
- ▶ **Fehlererkennung und Fehlerkorrektur:**
  - ▶ mache Texte werden durch die Übersetzung länger,
  - ▶ so dass, falls ein korrekter Funktionswert  $f(w)$  zufällig „kaputt“ geht (Übertragungsfehler),
  - ▶ u. U. Fehlererkennung oder gar Fehlerkorrektur möglich
  - ▶ siehe Vorlesungen über Codierungstheorie

- ▶ In Spezialfällen ist die Forderung  $\text{sem}_A(w) = \text{sem}_B(f(w))$  kein Problem:
- ▶ z. B. bei Verschlüsselung und manchen Anwendungen von Kompression:
  - ▶ man will vom Übersetzten  $f(x)$  eindeutig zum ursprünglichen  $x$  zurückkommen
  - ▶  $f$  ist injektiv (linkseindeutig: wenn  $x_1 \neq x_2$  dann  $f(x_1) \neq f(x_2)$ ).
- ▶ Dann kann man die Bedeutung  $\text{sem}_B$  im wesentlichen *definieren* durch die Festlegung  $\text{sem}_B(f(x)) = \text{sem}_A(x)$ .
  - ▶ Beachte: Dafür ist Injektivität von  $f$  wichtig!
- ▶ Wenn  $f$  injektiv ist, heie eine Übersetzung auch **Codierung**.
- ▶ Die  $f(w)$  heien **Codewörter**.
- ▶ Die Menge  $\{f(w) \mid w \in L_A\}$  heit dann auch ein **Code**.

# Wie spezifiziert man eine Übersetzung?

- ▶ Wenn  $L_A$  unendlich ist, kann man nicht alle Möglichkeiten aufzählen . . .
- ▶ Auswege:
  - ▶ Homomorphismen  
(aus dem Griechischen, homos für „gleich“ und morphe für „Form“), sind strukturerhaltende Abbildungen
  - ▶ Block-Codierungen

- ▶ Gegeben:
  - ▶ zwei Alphabete  $A$  und  $B$  und
  - ▶ eine Abbildung  $h : A \rightarrow B^*$ .
- ▶ Zu  $h$  definiert man Funktion  $h^{**} : A^* \rightarrow B^*$  vermöge

$$h^{**}(\varepsilon) = \varepsilon$$

$$\forall w \in A^* : \forall x \in A : h^{**}(wx) = h^{**}(w)h(x)$$

- ▶ Beispiel:
  - ▶  $h(a) = 10$  und  $h(b) = 001$
  - ▶  $h^{**}(bab) = h(b) \cdot h(a) \cdot h(b) = 001 \cdot 10 \cdot 001 = 00110001$
- ▶ Eine solche Abbildung  $h^{**}$  nennt man einen **Homomorphismus**.
- ▶ Homomorphismus heißt  **$\varepsilon$ -frei**, wenn  $\forall x \in A : h(x) \neq \varepsilon$ .

- ▶ Gegeben: Abbildung  $h : A \rightarrow B^*$
- ▶ Frage: ist Abbildung  $h^{**} : A^* \rightarrow B^*$  eine Codierung, also injektiv?
- ▶ im allgemeinen nicht ganz einfach zu sehen.
- ▶ einfacher Spezialfall:  $h$  ist **präfixfrei**
- ▶ Das bedeutet: für *keine zwei verschiedenen* Symbole  $x_1, x_2 \in A$  gilt:  $h(x_1)$  ist ein Präfix von  $h(x_2)$ .

- ▶ Ein Code ist präfixfrei, wenn kein gültiges Codewort der Beginn eines anderen gültigen Codewortes ist.
- ▶ Für einen nicht präfixfreien Code müsste man die Symbole mit Hilfe von Trennzeichen markieren.
- ▶ Beispiel: Text ohne Trennsymbole „Urinstinkt“ könnte man dekodieren in „Urin stinkt“ - das Trennsymbol sorgt für Eindeutigkeit der Dekodierung
- ▶ Das zusätzliche Trennzeichen macht den Code allerdings weniger effizient.
- ▶ Durch die Bedingung der Präfixfreiheit definiert man den Code so, dass zwei beliebige Codeworte ohne Trennsymbol hintereinander geschrieben werden können, ohne die eindeutige Dekodierbarkeit zu gefährden
- ▶ In unserem Beispiel könnte man also statt Einführung der Trennzeichen die Präfixfreiheit fordern, d.h. es dürfte im Code keine Wörter „U, Ur, Uri, Urin, Urins, Urinst, Urinstin, ...“ geben



- ▶ allgemeines Problem: nicht alle Wörter aus  $B^*$  sind Codewort, d. h.  $h^{**}$  ist im allgemeinen nicht surjektiv
- ▶ damit Decodierung trotzdem totale Abbildung  $u$  sein kann, definiere  $u : B^* \rightarrow (A \cup \{\perp\})^*$ .
  - ▶ wenn ein  $w \in B^*$  gar nicht decodiert werden kann, dann soll in  $u(w)$  das Symbol  $\perp$  vorkommen

## Beispiel

- ▶ Homomorphismus  $h : \{a, b, c\}^* \rightarrow \{0, 1\}^*$  mit  $h(a) = 1$ ,  $h(b) = 01$ ,  $h(c) = 001$ .
  - ▶ Dieser Homomorphismus ist präfixfrei.
- ▶ dann  $u : \{0, 1\}^* \rightarrow \{a, b, c, \perp\}^*$
- ▶ z. B. soll gelten
  - ▶  $u(\varepsilon) = \varepsilon$
  - ▶  $u(001) = c$
  - ▶  $u(0101) = bb$
  - ▶  $u(0) = \perp$  o. ä.

- wir schreiben mal hin (und diskutieren das anschließend):

$$u(w) = \begin{cases} \varepsilon & \text{falls } w = \varepsilon \\ \text{a}u(w') & \text{falls } w = \text{1}w' \\ \text{b}u(w') & \text{falls } w = \text{01}w' \\ \text{c}u(w') & \text{falls } w = \text{001}w' \\ \perp & \text{sonst} \end{cases}$$

- sei  $w = \text{100101} = h(\text{acb})$ ; wir rechnen:

$$\begin{aligned} u(\text{100101}) &= \text{a}u(\text{00101}) \\ &= \text{ac}u(\text{01}) \\ &= \text{acb}u(\varepsilon) \\ &= \text{acb} \end{aligned}$$

$$\text{▶ } u(w) = \begin{cases} \varepsilon & \text{falls } w = \varepsilon \\ au(w') & \text{falls } w = 1w' \\ bu(w') & \text{falls } w = 01w' \\ cu(w') & \text{falls } w = 001w' \\ \perp & \text{sonst} \end{cases}$$

- ▶  $u(100101) = au(00101) = acu(01) = acbu(\varepsilon) = acb$
- ▶ Warum hat das geklappt?
- ▶ In jedem Schritt war klar, welche Zeile der Definition von  $u$  anzuwenden war.
- ▶ denn ...

- ▶ Man spricht hier von **Wohldefiniertheit**
- ▶ immer ein Problem, wenn Funktionswert potenziell „auf mehreren Wegen“ festgelegt;  
dann klar machen:
  - ▶ entweder nur ein Weg „gangbar“
  - ▶ oder auf allen Wegen gleicher Funktionswert
- ▶ einen *präfixfreien* Code kann man also so decodieren:

$$u(w) = \begin{cases} \varepsilon & \text{falls } w = \varepsilon \\ x u(w') & \text{falls } w = h(x)w' \text{ für ein } x \in A \\ \perp & \text{sonst} \end{cases}$$

- ▶ Beachte: das heißt nur, dass man präfixfreie Codes „so einfach“ decodieren kann.

- ▶ ASCII-Code:
  - ▶ alle Zeichen werden durch Wörter gleicher Länge codiert
  - ▶ Längere Texte werden übersetzt, indem man zeichenweise decodiert.
- ▶ Unicode: könnte man analog zu ASCII machen, ABER:  
weil Unicode-Zeichensatz so viele Zeichen enthält, bräuchte man 4 Bytes pro Zeichen
- ▶ Idee: Platzsparende Codierung verwenden, sofern ein (deutscher) Text nur sehr wenige Zeichen verwendet → UTF-8
- ▶ UTF-8 ist präfixfrei

Char. number range (hexadecimal)	UTF-8 octet sequence (binary)
0000 0000 - 0000 007F	0xxxxxxx
0000 0080 - 0000 07FF	110xxxxx 10xxxxxx
0000 0800 - 0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx
0001 0000 - 0010 FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

- ▶ Determine the number of octets required ...
- ▶ Prepare the high-order bits of the octets ...
- ▶ Fill in the bits marked x ...
  - ▶ Start by putting the lowest-order bit of the character number in the lowest-order position of the last octet of the sequence,
  - ▶ then put the next higher-order bit of the character number in the next higher-order position of that octet, ...
  - ▶ When the x bits of the last octet are filled in, move on to the next to last octet, then to the preceding one, etc. ...

- ▶ Integralzeichen  $\int$  hat Unicode Codepoint 0x222B
- ▶ 0x222B in Bits 0010 0010 0010 1011
- ▶ benutze also die Zeile

Char. number range (hexadecimal)	UTF-8 octet sequence (binary)
0000 0800 - 0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx

- ▶ also 0010 0010 0010 1011 = 0010 001000 101011
- ▶ also UTF-8 Codierung 11100010 10001000 10101011
- ▶ Man überlege sich
  - ▶ UTF-8 ist präfixfrei
  - ▶ UTF-8 ist eine platzsparendere Codierung als Unicode

## Das sollten Sie mitnehmen:

- ▶ Übersetzungen sind in verschiedenen Situationen nützlich
- ▶ Homomorphismen
- ▶ Codes
- ▶ UTF-8

## Das sollten Sie üben:

- ▶ Homomorphismen anwenden
- ▶ Zeichen in UTF-8 codieren



## Codierungen

### Von Wörtern zu Zahlen und zurück

- Dezimaldarstellung von Zahlen

- Andere Zahldarstellungen

- Von Zahlen zu ihren Darstellungen

### Von einem Alphabet zum anderen

- Ein Beispiel: Übersetzung von Zahldarstellungen

- Homomorphismen

- Beispiel Unicode: UTF-8

## Huffman-Codierung

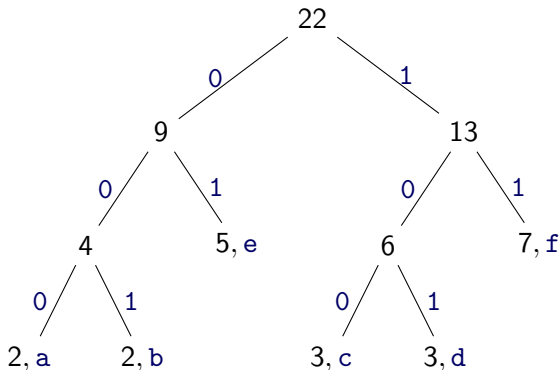
- Algorithmus zur Berechnung von Huffman-Codes

- Weiteres zu Huffman-Codes

- ▶ Gegeben: ein Alphabet  $A$  und ein Wort  $w \in A^*$
- ▶ Eine sogenannte **Huffman-Codierung** von  $w$  ist
  - ▶ eine Abbildung  $h : A^* \rightarrow Z_2^*$ ,
  - ▶ die ein  $\varepsilon$ -freier Homomorphismus ist.
  - ▶  $h$  also eindeutig festgelegt durch die  $h(x)$  für alle  $x \in A$ .
- ▶ Verwendung z. B. in Kompressionsverfahren wie gzip oder bzip2
- ▶ denn bei Huffman-Codierungen
  - ▶ häufigere Symbole durch kürzere Wörter codiert und
  - ▶ seltenere Symbole durch längere

- ▶ Gegeben
  - ▶ ein  $w \in A^*$  und
  - ▶ die Anzahlen  $N_x(w)$  aller Symbole  $x \in A$  in  $w$
  - ▶ o. B. d. A. alle  $N_x(w) > 0$  (den Rest muss man nicht codieren)
- ▶ Algorithmus zur Bestimmung eines Huffman-Codes arbeitet in zwei Phasen:
  1. es wird ein Baum konstruiert
    - ▶ dessen Blätter den  $x \in A$  entsprechen und
    - ▶ dessen Kanten mit 0 und 1 beschriftet sind
  2. Ablesen der Codes aus dem Baum (Pfadbeschriftungen)

- ▶ Beispiel:  $w = \text{afebfecaaffdeddccefbff}$
- ▶ Baum am Ende:



- ▶ Homomorphismus (präfixfrei!)

x	a	b	c	d	e	f
$h(x)$	000	001	100	101	01	11

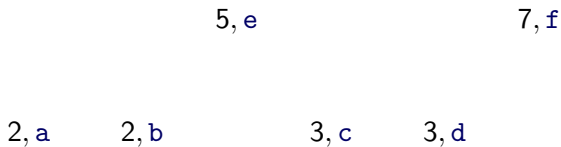
# Konstruktion des Huffman-Baumes (1)

- ▶ Zu jedem Zeitpunkt hat man
  - ▶ eine Menge  $M_i$  von „noch zu betrachtenden Symbolmengen mit ihren Häufigkeiten“
  - ▶ und eine ebenso große Menge von schon konstruierten Teilbäumen
- ▶ Initialisierung:
  - ▶  $M_0$  ist die Menge aller  $\{(N_x(w), \{x\})\}$  für  $x \in A$ ,
  - ▶ Als Anfang für die Konstruktion des Baumes zeichnet man für jedes Symbol einen Knoten mit Markierung  $(N_x(w), \{x\})$ .
- ▶ Beispiel

$$M_0 = \{ (2, \{\text{a}\}) , (2, \{\text{b}\}) , (3, \{\text{c}\}) , (3, \{\text{d}\}) , (5, \{\text{e}\}) , (7, \{\text{f}\}) \}$$

# Konstruktion des Huffman-Baumes (2)

Anfang im Beispiel:



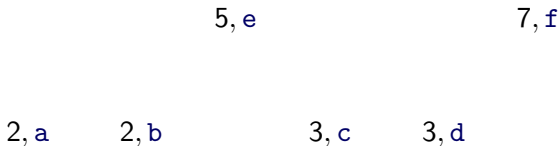
Iterationsschritt des Algorithmus:

- ▶ Solange Menge  $M_i$  noch mindestens zwei Paare enthält, Bestimme Menge  $M_{i+1}$  wie folgt:
  - ▶ Wähle zwei Paare  $(k_1, X_1)$  und  $(k_2, X_2)$ , deren Häufigkeiten zu den kleinsten noch vorkommenden gehören.
  - ▶ Entferne diese Paare aus  $M_i$  und fügt statt dessen das eine Paar  $(k_1 + k_2, X_1 \cup X_2)$  hinzu. Das ergibt  $M_{i+1}$ .
- ▶ im Graphen
  - ▶ Füge einen weiteren Knoten hinzu,
  - ▶ markiert mit der Häufigkeit  $k_1 + k_2$  und Kanten zu den Knoten, die für  $(k_1, X_1)$  und  $(k_2, X_2)$  eingefügt worden waren.

# Konstruktion des Huffman-Baumes (4)

Beispiel:

$$M_0 = \{ (2, \{a\}) , (2, \{b\}) , (3, \{c\}) , (3, \{d\}) , (5, \{e\}) , (7, \{f\}) \}$$

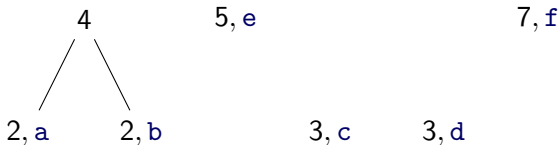




# Konstruktion des Huffman-Baumes (4)

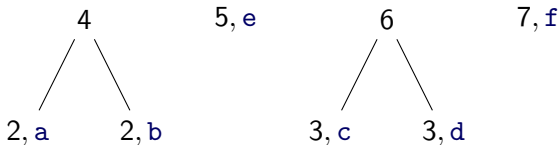
Beispiel:

$$M_1 = \{ (4, \{a, b\}) , (3, \{c\}) , (3, \{d\}) , (5, \{e\}) , (7, \{f\}) \}$$



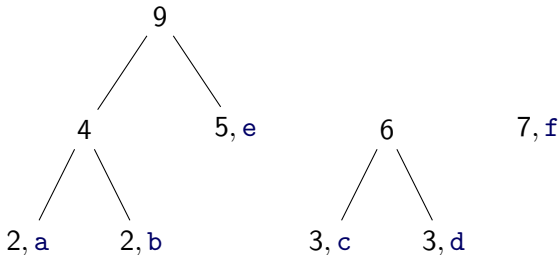
Beispiel:

$$M_2 = \{ (4, \{a, b\}) , (6, \{c, d\}) , (5, \{e\}) , (7, \{f\}) \}$$



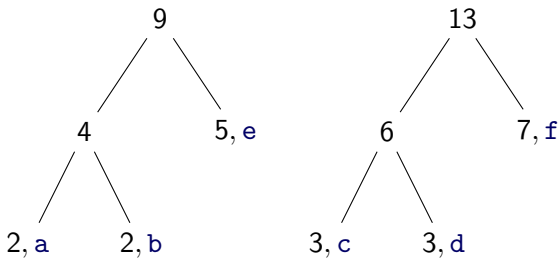
## Beispiel

$$M_3 = \{ (9, \{a, b, e\}) , (6, \{c, d\}) , (7, \{f\}) \}$$



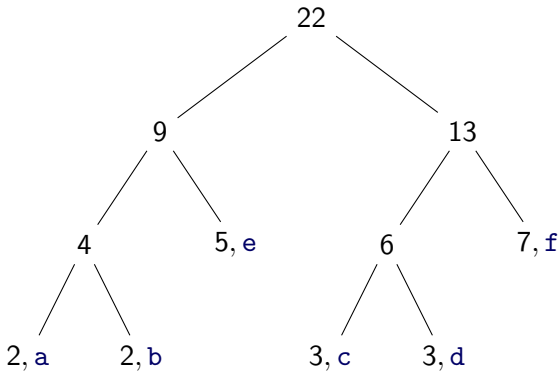
Beispiel

$$M_4 = \{ (9, \{a, b, e\}) , (13, \{c, d, f\}) \}$$

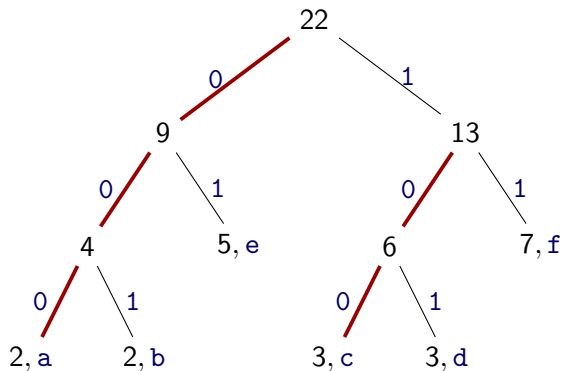


Beispiel

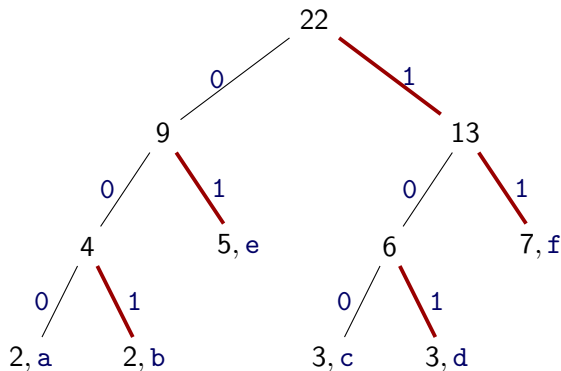
$$M_5 = \{ (22, \{a, b, c, d, e, f\}) \}$$



- ▶ nach links führende Kanten werden mit 0 beschriftet
- ▶ nach rechts führende Kanten werden mit 1 beschriftet



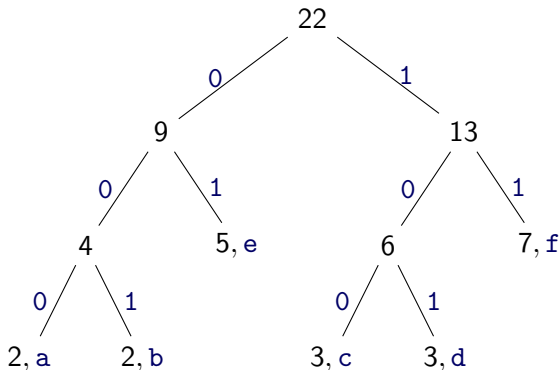
- ▶ nach links führende Kanten werden mit 0 beschriftet
- ▶ nach rechts führende Kanten werden mit 1 beschriftet



- ▶ gehe auf kürzestem Weg von der Wurzel des Baumes zu dem Blatt, das  $x$  entspricht,
- ▶ konkateniere der Reihe nach alle Symbole, mit denen die Kanten auf diesem Weg beschriftet sind.



- ▶ Beispiel:  $w = \text{afebfecaaffdeddccefbff}$
- ▶ Baum am Ende:



- ▶ Homomorphismus (präfixfrei!)

x	a	b	c	d	e	f
$h(x)$	000	001	100	101	01	11

- ▶ Huffman-Code nicht eindeutig
  - ▶ im allgemeinen mehrere Möglichkeiten, welche zwei Knoten zu einem neuen vereinigt werden
  - ▶ im Baum links und rechts vertauschbar
- ▶ aber alle sind „gleich gut“:
  - ▶ Unter allen präfixfreien Codes führen Huffman-Codes zu kürzesten Codierungen  
*des Wortes, für das die Huffman-Codierung konstruiert wurde.*

- ▶ Verallgemeinerung des obigen Verfahrens:
  - ▶ Betrachte nicht Häufigkeiten einzelner Symbole,
  - ▶ sondern für Teilwörter einer festen Länge  $b > 1$ .
  - ▶ einziger Unterschied: an den Blättern des Huffman-Baumes stehen Wörter der Länge  $b$ .
- ▶ So etwas nennt man eine **Block-Codierung**.
  - ▶ Statt  $h(x)$  für  $x \in A$  festzulegen,
  - ▶ legt man  $h(w)$  für alle **Blöcke**  $w \in A^b$  fest, und
  - ▶ erweitert dies zu einer Funktion  $h : (A^b)^* \rightarrow B^*$ .

## Das sollten Sie mitnehmen:

- ▶ Huffman-Codierung liefert kürzest mögliche präfixfreie Codes
- ▶ „Algorithmus“ zur Bestimmung des Huffman-Baumes
- ▶ warum die Anführungszeichen?

## Das sollten Sie üben:

- ▶ Huffman-Codes berechnen