

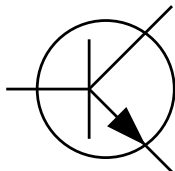
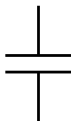
# **Computergrundlagen**

## **Boolesche Logik, Zahlensysteme und Arithmetik**

**Axel Arnold**

Institut für Computerphysik  
Universität Stuttgart

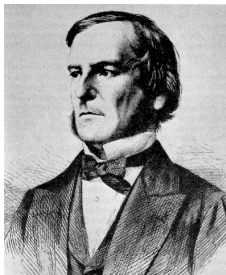
Wintersemester 2011/12



Ein Mikroprozessor

- ist ein Netz von Transistoren, Widerständen und Kondensatoren
- Leitungen kennen nur zwei Zustände: Spannung oder nicht
- Interpretation als ja/nein, 0/1, an/aus, ...
- Schaltungen entsprechen logischen Operationen  
(„Spannung an Pin a und Spannung an Pin b  
⇒ Spannung am Ausgang“)

Wie kann ich damit rechnen?



G. Boole,  
1815 - 1864

- erlaubt formale Beweise über logische Aussagen
- Grundlage der Computerlogik
- kann mathematisch als Algebra formalisiert werden  
– als eine *boolesche Algebra*

Wir betrachten eine Menge

- mit zwei Elementen 1 („wahr“) und 0 („falsch“)
- mit zwei Verknüpfungen  $\vee$  („oder“) und  $\wedge$  („und“)
- mit einer einstelligen Verknüpfung  $\neg$  („nicht“, Negation)

Ferner gilt für beliebige  $a, b, c \in \{0, 1\}$ :

$$\begin{aligned} 1. \quad & a \vee (b \vee c) = (a \vee b) \vee c, \\ & a \wedge (b \wedge c) = (a \wedge b) \wedge c \end{aligned} \quad \text{(Assoziativität)}$$

$$\begin{aligned} 2. \quad & a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c), \\ & a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c) \end{aligned} \quad \text{(Distributivität)}$$

$$3. \quad a \vee b = b \vee a, \quad a \wedge b = b \wedge a \quad \text{(Kommutativität)}$$

$$4. \quad a \vee (a \wedge b) = a, \quad a \wedge (a \vee b) = a \quad \text{(Adsorption)}$$

$$5. \quad a \vee \neg a = 1, \quad a \wedge \neg a = 0 \quad \text{(Komplemente)}$$

- Neutralität:  
 $a \vee 0 = a, \quad a \wedge 1 = a$
- Idempotenz:  
 $a \vee a = a, \quad a \wedge a = a$
- Extremalgesetze:  
 $a \vee 1 = 1, \quad a \wedge 0 = 0$
- Doppelnegation:  $\neg(\neg a) = a$
- Dualität:  
 $\neg 0 = 1, \quad \neg 1 = 0$
- De Morgansche Gesetze:  
 $\neg(a \vee b) = \neg a \wedge \neg b, \quad \neg(a \wedge b) = \neg a \vee \neg b$

Die zweielementige boolesche Algebra entspricht genau der Aussagenlogik – und unserem Verständnis von Logik.

- Wie kann ich mit nur zwei Elementen Zahlen darstellen?

Sei  $B > 0$  eine natürliche Zahl. Dann kann jede natürliche Zahl  $z$  *eindeutig* dargestellt werden als

$$z = \sum_{i=0}^{\infty} B^i z_i,$$

wobei für alle  $k$   $0 \leq z_k < B$  und nur endliche viele  $z_k \neq 0$ .

## Beispiel

$B = 10$  entspricht unserem Dezimalsystem:

$$1042 = 10^0 \cdot 2 + 10^1 \cdot 4 + 10^3 \cdot 1 = 1042d$$

$B = 8$  ergibt das Oktalsystem:

$$1042 = 8^0 \cdot 2 + 8^1 \cdot 2 + 8^3 \cdot 2 = 2022o$$

- Wie kann ich mit nur zwei Elementen Zahlen darstellen?

Wir benutzen das *Binärsystem* mit  $B = 2$  und Ziffern 0 und 1 (Bits)

## Beispiele

$$1042 = 2^{10} + 2^4 + 2^1 = 10.000.010.010b$$

- Umrechnung von Binär- auf Dezimalzahlen ist umständlich
- Binär ↔ oktal ist einfach:

$$\begin{array}{cccc} 2 & 1 & 4 & 2 & 1 & 4 & 2 & 1 & 4 & 2 & 1 \\ 10.000.010.010b = 2022o \end{array}$$

- Hexadezimal* ( $B = 16$ , Ziffern 1–9, A–F) auch:

$$\begin{array}{cccc} 4 & 2 & 1 & 8 & 4 & 2 & 1 & 8 & 4 & 2 & 1 \\ 10.000.010.010b = 812h \end{array}$$

$$1010.1111.1111.1110b = AFFEh$$

Genau wie im Dezimalsystem:

$$\begin{array}{r} 101010 \quad (\text{Summand } a) \\ + \quad 1111 \quad (\text{Summand } b) \\ \quad 1110 \quad (\text{Übertrag } c) \\ \hline = 111001 \quad (\text{Ergebnis } e) \end{array}$$

$$\begin{array}{r} 101010 \quad (1. \text{ Summand } a) \\ - \quad 1111 \quad (2. \text{ Summand } b) \\ \quad 11111 \quad (\text{geborgt } c) \\ \hline = 11011 \quad (\text{Ergebnis } e) \end{array}$$

Betrachten wir die Ziffern als Wahrheitswerte, so gilt:

$$e_0 = (a_0 \vee b_0) \wedge \neg(a_0 \wedge b_0) =: a_0 \oplus b_0$$

$$c_1 = a_0 \wedge b_0$$

$$e_i = (a_i \oplus b_i) \oplus c_i$$

$$c_{i+1} = (a_i \wedge b_i) \vee (a_i \wedge c_i) \vee (b_i \wedge c_i)$$

Damit kann die Addition als Schleifen-Schaltkreis realisiert werden!



Was ist z.B.  $-5 = 0 - 5$ ?

$$\begin{array}{r} 0 \\ - \quad 101 \\ \hline \dots 11111 \\ \hline = \quad \dots 111011 \end{array}$$

- im Prinzip unendlich viele führende 1er
- in der Praxis endliche Darstellung (z.B. 32 Bit)
- Bei  $n$  Bit-Darstellung wird  $-z$  als  $2^n - z$  dargestellt
- Bei 8 Bit z.B. ist  $-5 = 256 - 5 = 251 = 11111011b$
- Addition und Subtraktion funktionieren ohne Änderung, solange module  $2^n$  gerechnet wird



$$\begin{array}{r} 101111 \quad / \quad 101 \quad = \quad 1001 \quad (\text{Ergebnis}) \\ \hline - 101\dots \\ \hline = 111 \\ - 0\dots \\ \hline = 111 \\ - 0. \\ \hline = 111 \\ - 101 \\ \hline = 10 \quad (\text{Rest}) \end{array}$$

↑                    ↑                    ↑                    ↑

Binäre Division analog zur Multiplikation:

- Der Divisor wird zunächst ganz nach links geschoben
- Und dann bitweise wieder nach links
- Wenn der Rest größer als der Divisor ist, abziehen, und das entsprechende Bit des Ergebnisses setzen

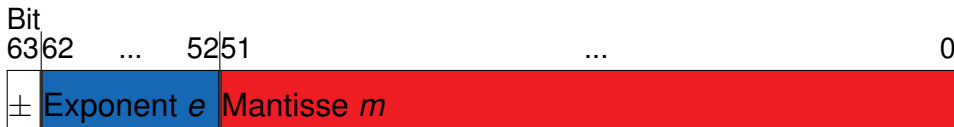
$$\pm 1,23456789 \cdot 10^{\pm 123}$$

- Eine Fließkommazahl besteht aus:
  - Vorzeichen
  - **Mantisse**
  - (10er-)Exponent
- Stellen daher nur einen Teil der rationalen Zahlen exakt dar
- Alle anderen Zahlen werden angenähert
- Die Mantissenlänge bestimmt die Genauigkeit der Näherung

Binär entsprechend:

$$\pm 1,0011110000001100101 \cdot 2^{\pm 1111011}$$

Speicherung einer 64-Bit-Fließkommazahl nach IEEE 754-Standard:



- Keine Komplementdarstellung von Mantisse oder Exponent
- Mantisse wird ohne führende Stelle gespeichert, die immer 1 ist
- Der 11-bittige Exponent  $e$  ist um 1023 verschoben gespeichert und nimmt Werte von -1022 ( $e = 1$ ) bis 1023 ( $e = 2046$ ) an
- Der Wert einer Zahl ist also:

$$\pm 1, m \cdot 2^{e-1023}$$

Was ist mit den Exponenten  $-1023$  ( $e = 0$ ) und  $1024$  ( $e = 2047$ )?

Diese stellen spezielle Werte dar:

$e = 0$ und $m = 0$	$\pm 0$
$e = 0$ und $m \neq 0$	Zahlen der Form $\pm 0, m \cdot 2^{-1022}$
$e = 2047$ und $m = 0$	$\pm \infty$
$e = 2047$ und $m \neq 0$	$\pm \text{NaN}$ (not a number)

- Da die erste Stelle immer 1 ist, kann 0 nur so dargestellt werden
- $\pm \infty$  ergibt sich z.B. bei Berechnung von  $\pm 1/0$
- $\pm \text{NaN}$  ergibt sich z.B. bei Berechnung von  $\sqrt{-1}$
- Python fängt NaNs mit Fehlern ab, nicht aber z.B. C

Beispiele:

$$\begin{aligned} & 1,0 \cdot 10^0 + 1,0 \cdot 10^{-5} \\ & = 1,0 \cdot 10^0 + 0,00001 \cdot 10^0 = 1,00001 \cdot 10^0 \end{aligned}$$

$$\begin{aligned} & 1,000002 \cdot 10^0 - 1,000001 \cdot 10^0 \\ & = 0,000001 \cdot 10^0 = 1,0 \cdot 10^{-6} \end{aligned}$$

- Verschieben der Mantisse der kleineren Zahl, bis beide denselben Exponenten haben
- Dabei gehen Stellen der kleineren Zahl verloren
- Dann gewöhnliche Addition/Subtraktion der Mantissen
- Schließlich den Exponenten verringern, bis die Mantisse keine führenden Nullen hat

Ist  $x < 2^{-l_m}$ , wobei  $l_m$  die Bitlänge der Mantisse ist, so ist in Fließkommaarithmetik

$$x \rightarrow x + 1 - 1 \rightarrow (x + 1) - 1 \rightarrow 1 - 1 \rightarrow 0$$

- *Auslöschung* der vorderen Stellen bei Subtraktion ungefähr gleich großer Zahlen führt zu fehlenden hinteren Stellen im Ergebnis
- Im Beispiel verschwindet die kleinere Zahl  $x$  komplett
- Ist  $x \approx 2^{-l_m}$ , verliert  $x$  fast alle signifikanten Stellen
- Formeln daher wenn möglich stets so umformen, dass keine Auslöschung passieren kann



# Beispiel: quadratische Gleichung

- Lösen der quadratischen Gleichung  $x^2 + ax + b = 0$ :

$$x_{\pm} = -\frac{a}{2} \pm \sqrt{\left(\frac{a}{2}\right)^2 - b}$$

- Bei kleinem  $b$  ist  $\sqrt{\left(\frac{a}{2}\right)^2 - b} \approx \frac{a}{2}$
- $a > 0$ : *Auslöschung* bei der größeren Nullstelle  $x_+$
- $a < 0$ : *Auslöschung* bei der kleineren Nullstelle  $x_-$
- Ausweg: Es gilt

$$x_+ x_- = \left(-\frac{a}{2} + \sqrt{\left(\frac{a}{2}\right)^2 - b}\right) \left(-\frac{a}{2} - \sqrt{\left(\frac{a}{2}\right)^2 - b}\right) = b$$

- Berechne also nur die stabile Summe ( $x_-$  bei  $a > 0$ , sonst  $x_+$ ), und die zweite Nullstelle durch  $x_+ = b/x_-$  bzw.  $x_- = b/x_+$

Beispiel:

$$\begin{aligned} & 1,01 \cdot 10^0 \times 1,01 \cdot 10^5 \\ &= 1,0101 \cdot 10^{0+5} = 1,0101 \cdot 10^5 \end{aligned}$$

- Multiplikation und Division sind unkritisch, es kann zu keiner Auslöschung kommen
- Allerdings zu Über- oder Unterläufen des Exponenten
- In diesem Fall ist das Ergebnis  $\infty$  bzw. 0
- Meist kann man dies aber durch geeignete Umformungen auch vermeiden, z.B. durch logarithmisches Rechnen

- **Numerische Fehler**

- Auslöschung
- Rundungsfehler durch endliche Mantissenlänge

- **Algorithmische (Verfahrens-) Fehler**

- Abschneidefehler: Unendliche Summen müssen durch endliche Summen ersetzt werden
- Diskretisierungsfehler: Funktionsauswertung immer nur an endlich vielen Punkten

- **Modellierungsfehler**

Das der Rechnung zugrundeliegende Modell erfasst wesentliche Details des Experiments nicht

- **Datenfehler**

Genauigkeit der Anfangswerte (z.B. aus einem Experiment)

## Simulation

### Modellierungsfehler:

- entspricht keinem bekannten Atom/Molekül
- vernachlässigt quantenmechanische Effekte
- periodische Randbedingungen ersetzen unendliches Volumen

### Algorithmische Fehler:

- Das Potential  $U_{LJ}(r)$  wird für  $r \gg 0$  rasch klein, daher betrachtet man nur Teilchenpaare mit kleiner Distanz
  - Fehler durch die weggelassenen Wechselwirkungen?
  - Wie klein kann man den Abschneideradius wählen?
- Simulation mit diskretem Zeitschritt
  - Wie groß darf der Zeitschritt sein?
  - Wie lange muss man simulieren?

### Numerische Fehler:

- Trajektorien instabil bei kleinen Störungen  $\implies$  Lyapunov-Instabilität
- Für die Statistik meist unwesentlich

- **Datenfehler**
  - Bessere Messverfahren entwickeln
  - Algorithmen wählen, die mit Störungen zurechtkommen – **Stabilität**
- **Numerische Fehler und algorithmische Fehler**
  - Algorithmen nutzen, die numerische Fehler minimieren
  - Algorithmen mit bekannten Fehlerabschätzungen einsetzen – **Konvergenz**
- **Modellierungsfehler**
  - Das Modell verfeinern
  - Dazu sind leistungsfähigere Algorithmen nötig – **Effizienz**

Wie kann ich Stabilität, Konvergenz und Effizienz eines Verfahrens beschreiben?

Seien  $f, g$  zwei Funktionen.

Falls

$$\lim_{x \rightarrow a} \frac{|f(x)|}{|g(x)|} < \infty,$$

so schreiben wir kurz  $f = \mathcal{O}_{x \rightarrow a}(g)$  („ $f$  ist von der Ordnung  $g$ “).

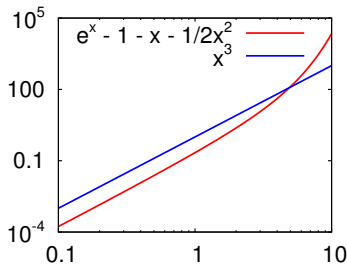
- Es gilt zum Beispiel

$$e^x - \left(1 + x + \frac{1}{2}x^2\right) = \mathcal{O}_{x \rightarrow 0}(x^3),$$

bzw.

$$e^x = 1 + x + \frac{1}{2}x^2 + \mathcal{O}_{x \rightarrow 0}(x^3).$$

- Bubble-Sort hat einen asymptotischen Aufwand von  $\mathcal{O}(N^2)$  für Listenlängen  $N \rightarrow \infty$ .



Taylorentwicklung für die Position eines Teilchens gemäß der Newtonschen Gleichung:

$$x(t + \Delta t) = x(t) + \dot{x}(t)\Delta t + \frac{1}{2}\ddot{x}(t)\Delta t^2 + \frac{1}{6}\dddot{x}(t)\Delta t^3 + \mathcal{O}(\Delta t^4)$$

und analog

$$x(t - \Delta t) = x(t) - \dot{x}(t)\Delta t + \frac{1}{2}\ddot{x}(t)\Delta t^2 - \frac{1}{6}\dddot{x}(t)\Delta t^3 + \mathcal{O}(\Delta t^4)$$

Addition ergibt das *Verlet*-Verfahren zur numerischen Integration der Bewegungsgleichung:

$$x(t + \Delta t) = 2x(t) - x(t - \Delta t) + a(t)\Delta t^2 + \mathcal{O}_{\Delta t \rightarrow 0}(\Delta t^4)$$

- Fehler ist von der Ordnung  $\mathcal{O}_{\Delta t \rightarrow 0}(\Delta t^4)$
- Mit beliebig kleinem Zeitschritt  $\Delta t$  wird das Verfahren exakt
- Man sagt, das Verfahren **konvergiert**

Die wesentlichen Eigenschaften eines Algorithmus können mit Hilfe der Landau-Symbole charakterisiert werden:

- **Stabilität:** Wir betrachten den Fehler  $\Delta f(x)$  in Abhängigkeit von der Genauigkeit  $\Delta x$  der Eingabedaten. Dann soll

$$\Delta f(x) = \mathcal{O}_{\Delta x \rightarrow 0}(\Delta x^n)$$

mit möglichst großem  $n$  sein.

- **Konvergenz:** Z.B. der Fehler als Funktion des Zeitschritts  $\Delta t$ . Dann soll

$$\Delta f(x) = \mathcal{O}_{\Delta t \rightarrow 0}(\Delta t^n)$$

sein mit möglichst großem  $n$ .

- **Effizienz:** Die Rechenzeit  $T(N)$  z.B. bei  $N$  Teilchen. Dann soll

$$T(N) = \mathcal{O}_{N \rightarrow \infty}(N^n)$$

mit möglichst *kleinem*  $n$  sein.