



Kap. 2: Grundbegriffe

- 2.1 Begriffe der Mathematik (nur Wiederholung)
- 2.2 System, Abstraktion und Modell**
- 2.3 Information und ihre Repräsentation**
- 2.4 Formale Sprachen**
- 2.5 Graphen und Bäume
- 2.6 Algorithmen**



Quellen

- **M. Broy: "Informatik - Eine grundlegende Einführung", Teil 1, Springer-Verlag, 1992 (Kap. 1, 2)**
- **U. Rembold, P. Levi: "Einführung in die Informatik für Naturwissenschaftler und Ingenieure", 3. Auflage, Hanser-Verlag, 1999 (Kap. 2.2.1, 2.7)**
- **D. Werner u.a.: "Taschenbuch der Informatik", Fachbuchverlag Leipzig, 1995 (Kap. 2.3.1)**
- **U. Schöning: "Theoretische Informatik - kurz gefasst", Spektrum-Verlag, 1997**



2.1 Begriffe der Mathematik

- **Bemerkung:** Die in diesem Abschnitt besprochenen Begriffe sind entweder bereits aus der Schule bekannt oder werden in den Mathematik-Vorlesungen besprochen. Sie werden im weiteren als bekannt vorausgesetzt.

Def

Symbole in Aussagen

$\exists x$	es existiert ein x , es gibt ein x (<i>Existenz-Quantor</i>)
$\exists! x, \nexists x$	Varianten: es existiert genau ein x , es gibt kein x
$\forall x$	für alle x (<i>All-Quantor</i>)
$p \wedge q$	Aussage p und Aussage q
$p \vee q$	Aussage p oder Aussage q
$\neg p$	nicht p , Verneinung der Aussage p
$p \Rightarrow q$	wenn p , dann q
$p \Leftrightarrow q$	p genau dann, wenn q
$p :\Leftrightarrow q$	definitionsgemäß genau dann, wenn q



Mengen

Def

$\{a, b\}$

Die Menge mit den Elementen a und b

$\{x \mid p(x)\}$

Menge aller x , für die die Aussage $p(x)$ gilt

$\{\}, \emptyset$

die leere Menge

$a \in A$

a ist Element der Menge A

$A \subseteq B$

Teilmengenbeziehung

$A \subset B$

echte Teilmengenbeziehung

$A \cap B$

Durchschnitt

$A \cup B$

Vereinigung

$A \setminus B$

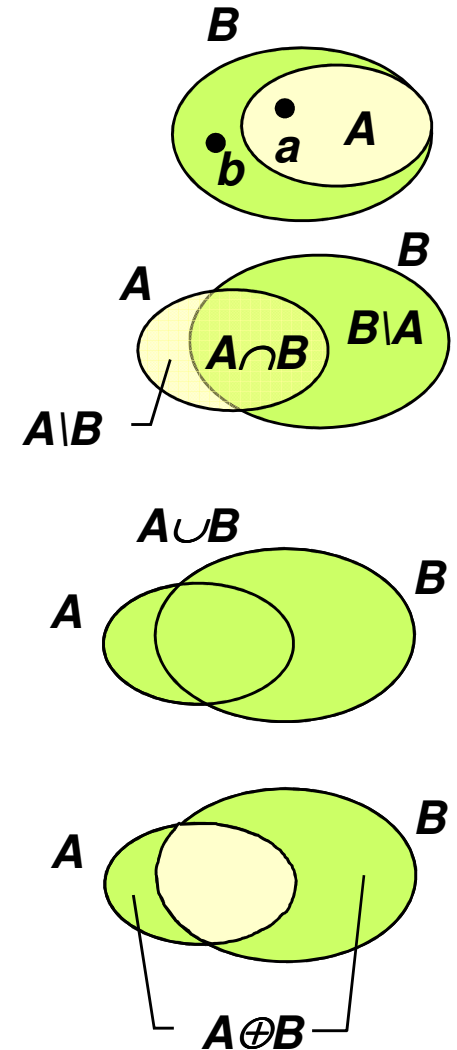
Differenz

$A \oplus B$

Disjunkte Vereinigung, $A \cup B \setminus (A \cap B)$

$|A|$

Kardinalität oder Mächtigkeit der Menge A . Bei endlichen Mengen: Anzahl der Elemente





Mengen (2)

- **Übliche Notationen für Zahlenmengen in der Mathematik**

\mathbb{N} **Die Menge der natürlichen Zahlen, $\{0, 1, 2, 3, \dots\}$**

\mathbb{N}^+ **Die natürlichen Zahlen ohne die Null, $\{1, 2, 3, \dots\}$, $\mathbb{N} \setminus \{0\}$**
Bemerkung: Manchmal wird \mathbb{N} auch ohne Null definiert: \mathbb{N} , \mathbb{N}_0

\mathbb{Z} **Die Menge der ganzen Zahlen, $\{\dots, -2, -1, 0, 1, 2, \dots\}$**

\mathbb{Q} **Die Menge der rationalen Zahlen, $\{x = p / q \mid p \in \mathbb{Z} \wedge q \in \mathbb{Z}^+\}$**

\mathbb{R} **Die Menge der reellen Zahlen**

\mathbb{C} **Die Menge der komplexen Zahlen**

$\mathbb{Z}^+, \mathbb{Q}^+, \mathbb{R}^+, \mathbb{C}^+$ **analog \mathbb{N}^+ (Die Null ist "ausgestochen")**



Potenzmenge, Produkt

Def

- Die *Potenzmenge* $P(A)$ einer Menge A ist die Menge aller Teilmengen von A , d.h. $P(A) = \{B \mid B \subseteq A\}$
 - Beispiel: $P(\{a,b\}) = \{ \{\}, \{a\}, \{b\}, \{a,b\} \}$
 - Falls $|A| < \infty$, dann gilt $|P(A)| = 2^{|A|}$
 - *Selbst-Test: Wie beweist man dies?*
- Das (*kartesische*) *Produkt* $A \times B$ der Mengen A und B ist die Menge aller geordneten Paare (a,b) mit $a \in A$ und $b \in B$.
 - Beispiel: $A=\{m,n\}, B=\{r,s,t\} \Rightarrow$
 $A \times B = \{ (m,r), (m,s), (m,t), (n,r), (n,s), (n,t) \}$
 - Notation: Man schreibt statt $A \times A$ auch A^2 .
 - Für endliche Mengen A und B gilt für die Kardinalitäten:
 $|A \times B| = |A| \cdot |B|$.



Relation

Def

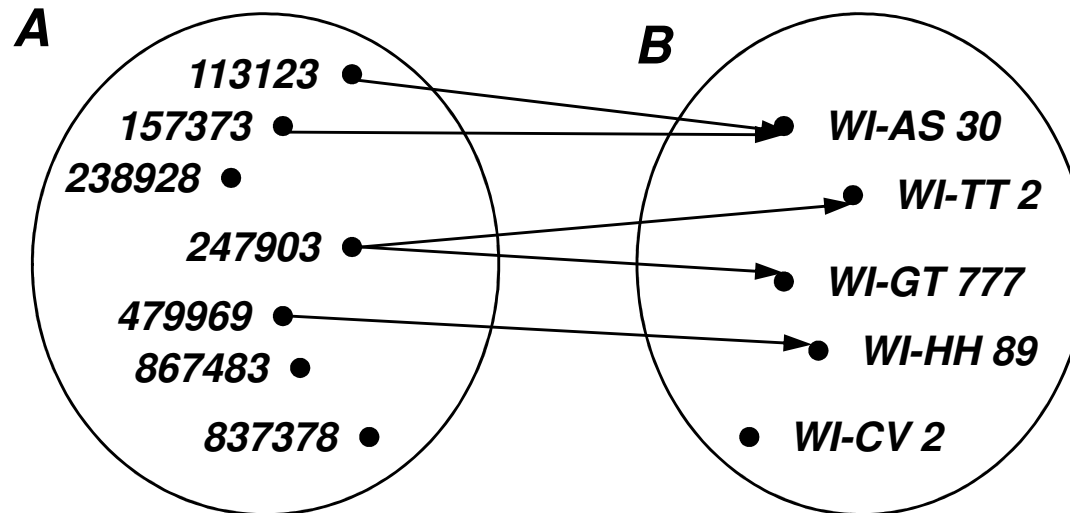
- Eine Teilmenge $R \subseteq A \times B$ des Produkts zweier Mengen A und B heißt (zweistellige oder binäre) Relation R zwischen A und B .

- Notation: statt $(a,b) \in R$ auch $R(a,b)$ oder *Infix-Notation*: $a R b$
- Beispiel:

A : Menge der Personalausweisnummern aller Wiesbadener,

B : Menge der vergebenen Autokennzeichen beginnend mit WI

fährt $\subseteq A \times B$ ist eine binäre Relation zwischen A und B .



fährt = {
(113123, WI-AS 30),
(157373, WI-AS 30),
(247903, WI-TT 2),
(247903, WI-GT 777),
(479969, WI-HH 89) }

fährt(113123, WI-AS 30)

oder 113123 **fährt** WI-AS 30

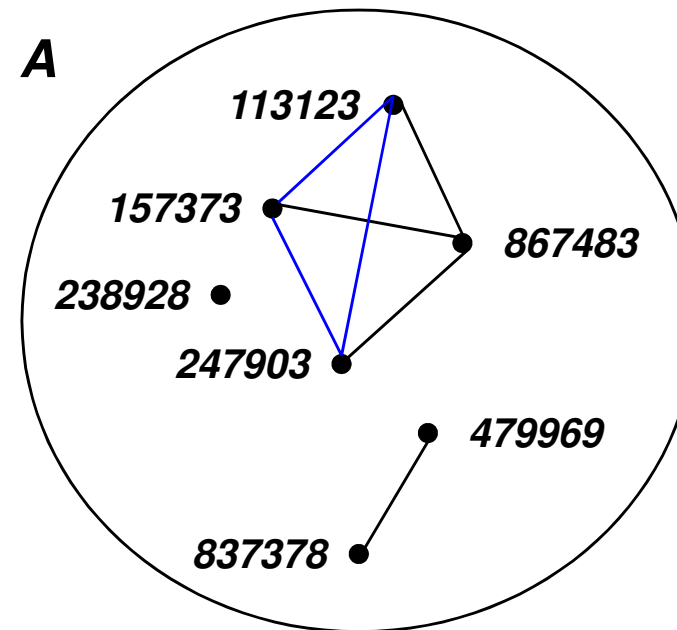


Relation (2)

- Eine Teilmenge $R \subseteq A \times A$ heißt Relation R auf der Menge A .

Def

- Beispiel:
 A : Menge der Personalausweisnummern (IDs) aller Wiesbadener,
 $R = \{ (x,y) \in A \times A \mid \text{Person mit ID } x \text{ ist verwandt mit Person mit ID } y \}$.
- Relationen besitzen spezielle Eigenschaften
hier z.B.: Transitivität





Eigenschaften von Relationen

- Sei $R \subseteq A \times A$ eine binäre Relation auf A . Dann heißt R
 - **reflexiv** $:\Leftrightarrow \forall a \in A: a R a$
 - Beispiele: Relationen $=$ und \leq auf \mathbb{N} , \subseteq auf Mengen
 - **irreflexiv** $:\Leftrightarrow \nexists a \in A: a R a$
 - Beispiele: Relationen \neq und $<$ auf \mathbb{N}
 - Hinweis: „irreflexiv“ \neq „nicht reflexiv“ (warum?)
 - **symmetrisch** $:\Leftrightarrow \forall a, b \in A: a R b \Rightarrow b R a$
 - Lies: "Für alle a und b aus A gilt: Aus a Relation b folgt b Relation a "
 - Beispiele: Relationen $=$ und \neq auf \mathbb{N}
 - **antisymmetrisch** $:\Leftrightarrow \forall a, b \in A: a R b \wedge b R a \Rightarrow a = b$
 - Beispiel: Relation \leq auf \mathbb{N} , \subseteq auf Mengen.



Eigenschaften von Relationen (2)

- (Fortsetzung)
 - ***transitiv*** : $\Leftrightarrow \forall a, b, c \in A: a R b \wedge b R c \Rightarrow a R c$
 - Beispiele: Relationen $= < > \leq$ auf \mathbb{N} , \subseteq auf Mengen
 - ***total*** : $\Leftrightarrow \forall a, b \in A: a R b \vee b R a$
 - Bemerkung: mathematisches "oder"
d.h.: es kann gleichzeitig $a R b$ und $b R a$ gelten.
 - Beispiel: Relation \leq auf \mathbb{N}



Matrixdarstellung binärer Relationen

- Sei $R \subseteq A \times A$ eine binäre Relation auf A.
- Das kartesische Produkt $A \times A$ lässt sich als Matrix veranschaulichen. Markiert man die Matrixzellen, die Elementen $(a,b) \in R$ entsprechen, erhält man eine Matrixdarstellung von R.
- Mit dieser lassen sich viele Relationseigenschaften visualisieren:

$$R = (\{1,2,3,4\}^2, =)$$

(1,1)	(1,2)	(1,3)	(1,4)
(2,1)	...		
(3,1)			
(4,1)			(4,4)

$$R = (\{1,2,3,4\}^2, \neq)$$

$$R = (\{1,2,3,4\}^2, \leq)$$

- R reflexiv
- R irreflexiv
- R symmetrisch
- R antisymm.

Matrix-Diagonale vollständig gefüllt

Matrix-Diagonale völlig leer

Matrix spiegelsymmetrisch zur Hauptdiagonalen

Matrix enthält kein spiegelsymmetrisches Zellenpaar außerhalb der Hauptdiagonalen



Äquivalenzrelation

Def

- Sei $R \subseteq A \times A$ eine Relation. Dann heißt R Äquivalenzrelation, wenn R reflexiv, transitiv und symmetrisch ist.
 - Ist R eine Äquivalenzrelation und ist $(a,b) \in R$, so heißen a und b *äquivalent*.
- Beispiel:
 - Sei $A = \mathbb{N}$, $n \in \mathbb{N}$. Dann ist $R = \{ (x,y) \mid x \bmod n = y \bmod n \}$ eine Äquivalenzrelation.
(x und y haben bei Division durch n denselben Rest)

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
...

$n = 4$

Äquivalenzklassen, d.h.
Mengen bzgl. R äquivalenter Elemente
(**Restklassen**)



Partielle und totale Ordnung

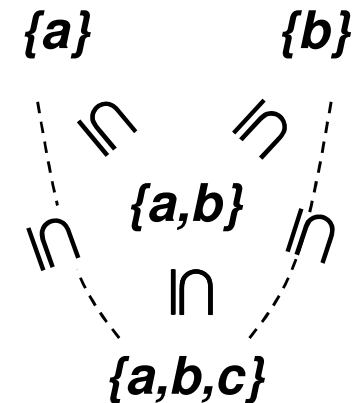
Def

- Eine reflexive, transitive und antisymmetrische Relation R auf einer Menge A heißt partielle Ordnung R auf der Menge A .

— Beispiel:

Sei $A = \{ \{a\}, \{b\}, \{a,b\}, \{a,b,c\} \}$, \subseteq die Teilmengenrelation.
Dann definiert \subseteq eine partielle Ordnung R auf A :

$$R = \{ (\{a\}, \{a,b\}), (\{a\}, \{a,b,c\}), (\{b\}, \{a,b\}), \\ (\{b\}, \{a,b,c\}), (\{a,b\}, \{a,b,c\}), (\{a\}, \{a\}), \\ (\{b\}, \{b\}), (\{a,b\}, \{a,b\}), (\{a,b,c\}, \{a,b,c\}) \}$$



— Bemerkung:

R ist keine totale Relation, z.B. gilt weder $\{a\} \subseteq \{b\}$ noch $\{b\} \subseteq \{a\}$.

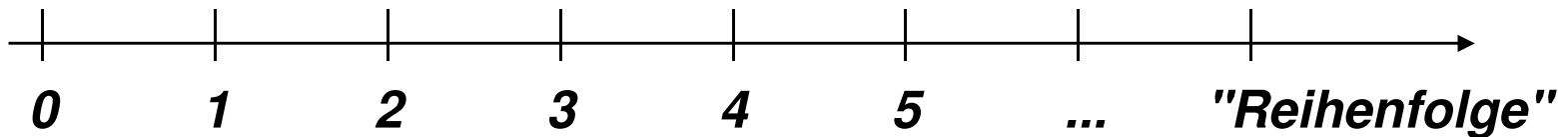


Partielle und totale Ordnung (2)

Def

- Eine reflexive, transitive, antisymmetrische und totale Relation R auf einer Menge A heißt lineare oder totale Ordnung R auf der Menge A .

- Beispiel: \leq auf natürlichen Zahlen



- „Beweis“ durch Nachprüfen der Eigenschaften:

- **Reflexivität:** $\forall a \in \mathbb{N}: a \leq a$
- **Transitivität:** $\forall a, b, c \in \mathbb{N}: a \leq b \wedge b \leq c \Rightarrow a \leq c$
- **Antisymmetrie:** $\forall a, b \in \mathbb{N}: a \leq b \wedge b \leq a \Rightarrow a = b$
- **Totalität:** $\forall a, b \in \mathbb{N}: a \leq b \vee b \leq a$





Funktion / Abbildung

Def

- Eine Relation $f \subseteq A \times B$ zwischen den Mengen A und B heißt Funktion oder Abbildung aus der Menge A in die Menge B , falls aus $(x,y) \in f$ und $(x,z) \in f$ folgt: $y = z$.

- Bemerkungen:
 - Funktionen sind also spezielle Relationen.
 - Übliche Notation: $f: A \rightarrow B$ und $f(a)=b$ statt $(a,b) \in f$
 - b heißt das **Bild** von a unter der Funktion f ,
 - a ist ein(!) **Urbild** von b .

Def

• $\text{Dom}(f) := \{a \in A \mid (a,b) \in f\}$ heißt **Definitionsbereich** von f

Def

• $\text{Rng}(f) := \{b \in B \mid (a,b) \in f\}$ heißt **Bild-** oder **Wertebereich** von f

- Englische Bezeichnungen: *Dom* - „domain“, *Rng* - „range“



Funktion / Abbildung (2)

Def

• Eine Funktion $f: A \rightarrow B$ heißt **total** : $\Leftrightarrow \text{Dom}(f) = A$.

- „Keine Definitionslücken – der Definitionsbereich ist gleich der Ausgangsmenge“

Def

• Eine Funktion $f: A \rightarrow B$ heißt **surjektiv** : $\Leftrightarrow \text{Rng}(f) = B$.

- „Jedes Element der Zielmenge besitzt (mind.) ein Urbild“

Def

• Eine Funktion $f: A \rightarrow B$ heißt **injektiv** : $\Leftrightarrow f(a)=f(b) \Rightarrow a=b$

- „Verschiedene Elemente der Definitionsmenge ergeben stets verschiedene Werte“

Def

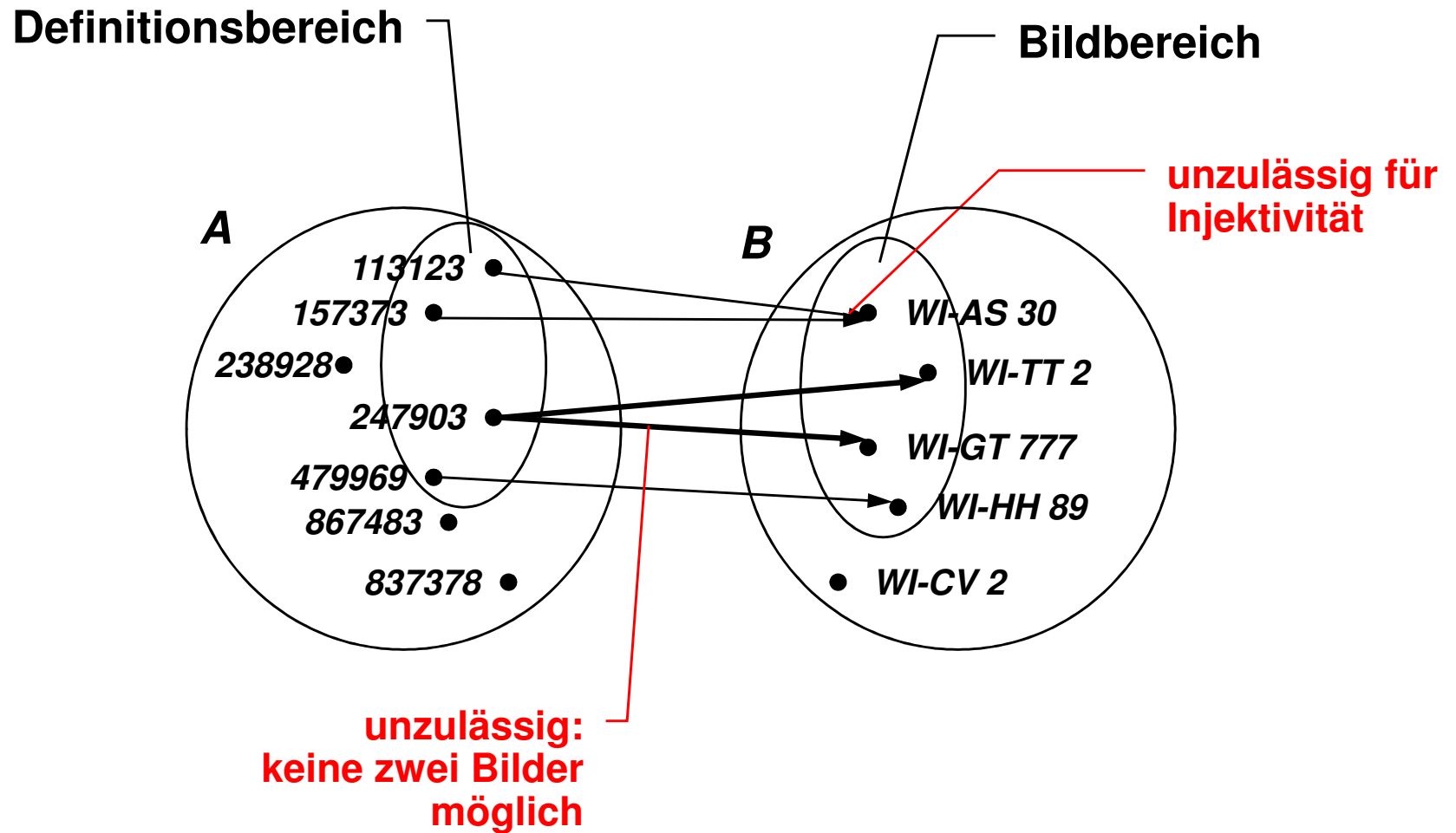
• Eine Funktion $f: A \rightarrow B$ heißt **bijektiv** : $\Leftrightarrow f$ ist total, surjektiv und injektiv.

- „Bijektive Funktionen sind umkehrbar. Jede Urbildmenge ist einelementig.“



Beispiel

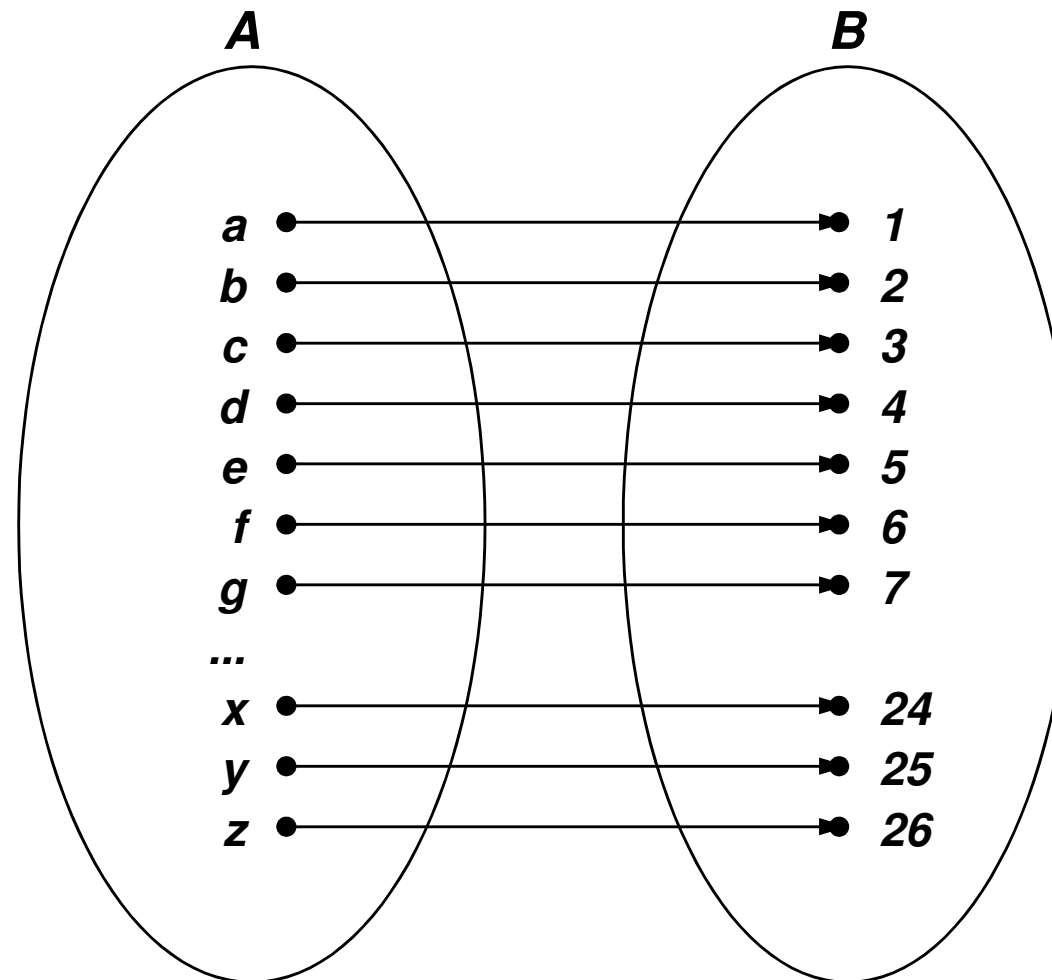
- Funktion





Beispiel

- Bijektion





2.2 System, Abstraktion und Modelle

- Der Systembegriff wird im täglichen Leben verwendet wie auch in allen wissenschaftlichen Disziplinen.

- Beispiele:

- Das politische System der Bundesrepublik Deutschland
 - Der menschliche Körper als biologisches System
 - Das Milchstraßensystem

Def

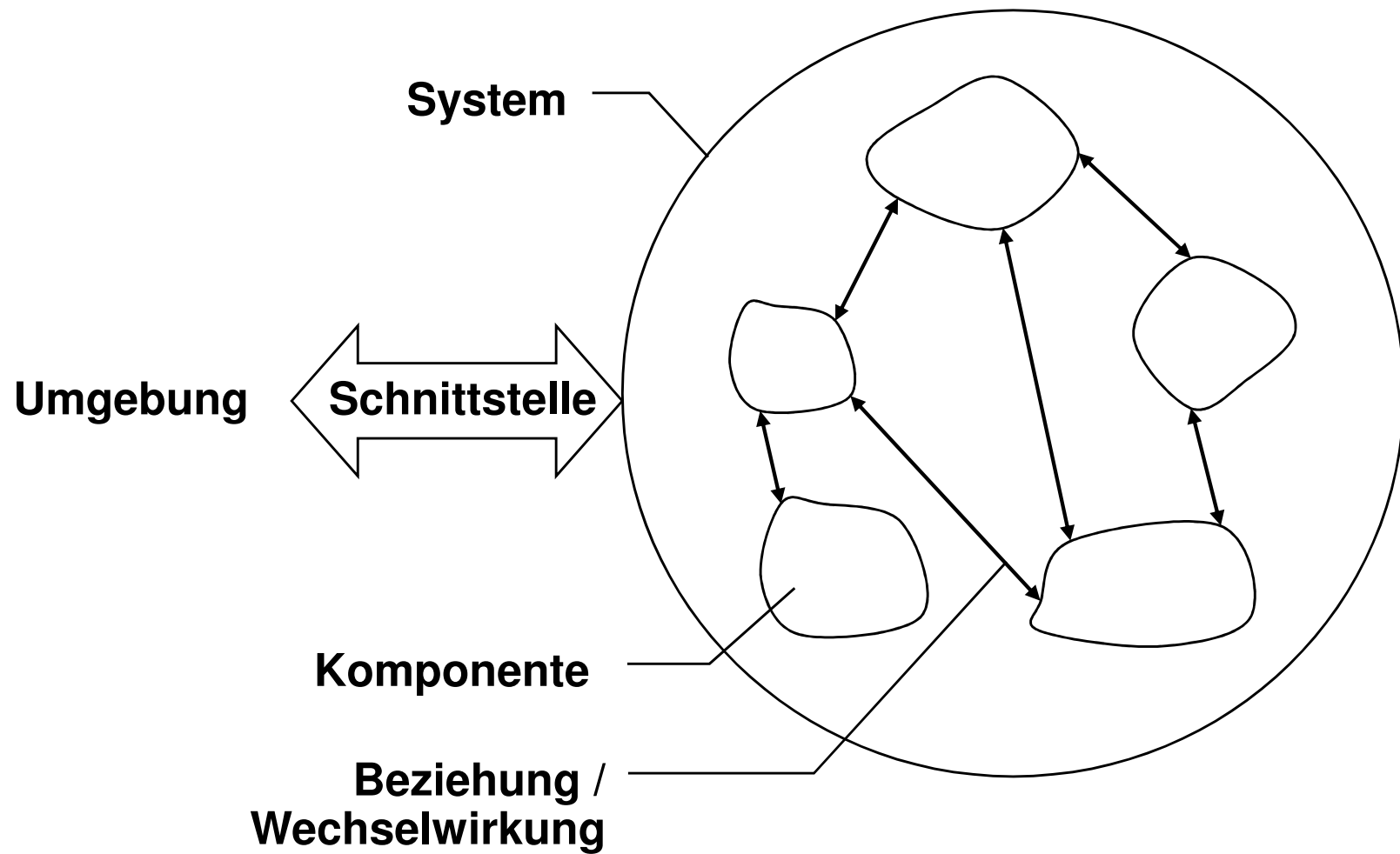
- Charakterisierende Merkmale eines *informationstechnischen Systems*:

- *Schnittstelle des Systems*:
Grenze zwischen "außerhalb" und "innerhalb"
 - *Umgebung des Systems*:
der äußere, für die Betrachtung weniger wichtige Teil
 - *System*:
innere Teil ist der eigentliche Betrachtungsgegenstand mit:
 - *Komponenten* (des Systems)
 - deren *Beziehungen* zueinander (Wechselwirkungen)



Graphische Veranschaulichung

- zum Systembegriff

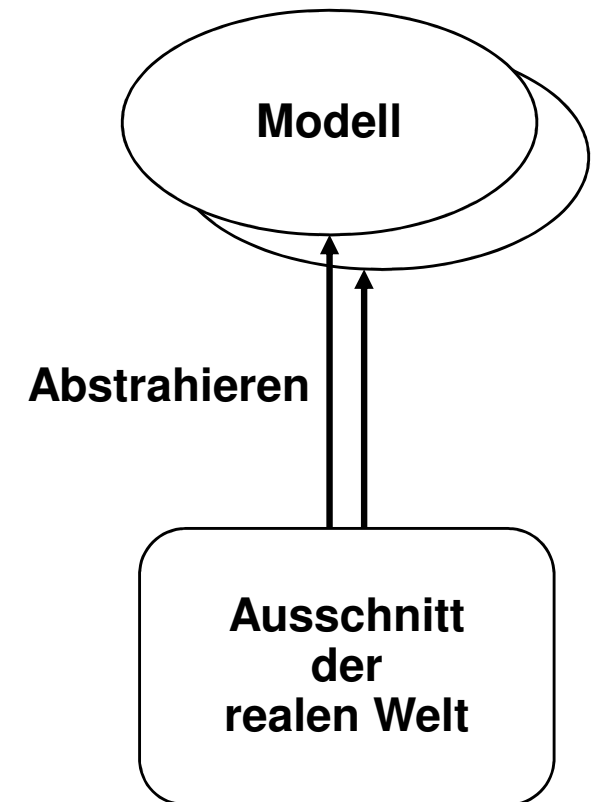




Abstraktion und Modelle

Def

- **Abstraktion** entsteht durch Erkennen von unter einer bestimmten Betrachtungsweise relevanten Gegenständen, Eigenschaften und Beziehungen eines Ausschnitts der realen Welt.
- **Modell als Ersatz der Realität**
- **zusätzliche wünschenswerte Eigenschaften wie z.B.**
 - einfacher zu verstehen (z.B. Straßenatlas)
 - billiger oder sicherer (z.B. Fahrsimulator)
 - mathematische Theorie nutzbar machen (z.B. Physik, Baustatik)
- **Für denselben Ausschnitt der Realität können verschiedene Modelle existieren.**





Abstraktion und Modelle (2)

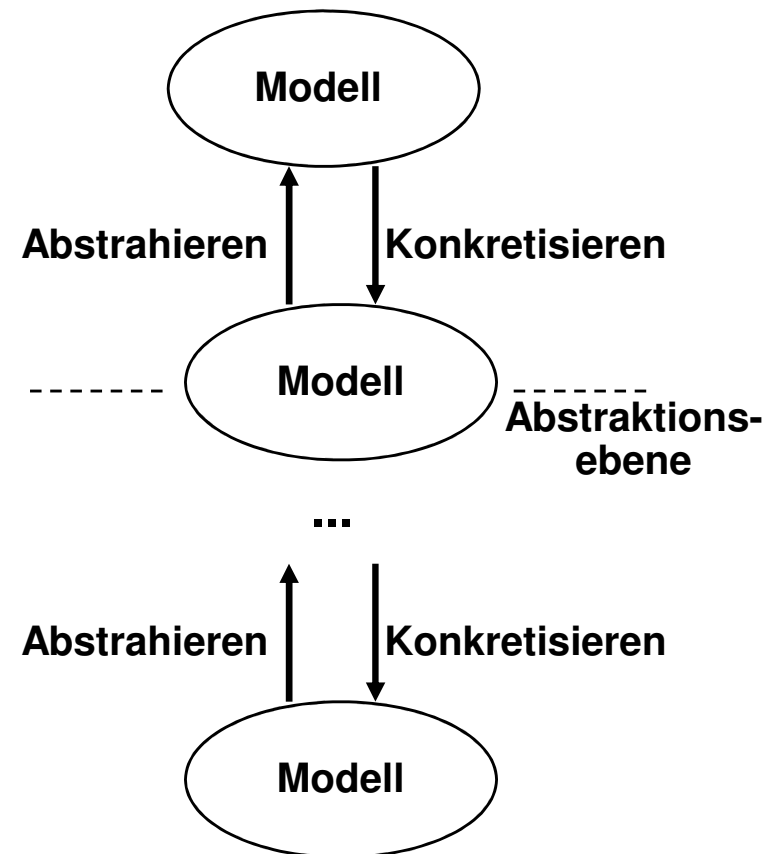
- Das Studium der Informatik beinhaltet das Kennenlernen einer Vielzahl von Modellen
 - aus der Mathematik
 - aus Ingenieur-Disziplinen
 - durch die Informatik selbst entwickelt (z.B. Graphen, Automaten, ...)
- In der Informatik sind *systemorientierte* Betrachtungsweisen verbreitet. Ziel oft: Struktur- und Verhaltensmodelle entwickeln.
- Wahl der *Abstraktionsebene* spielt oft entscheidende Rolle:
 - ⇒ Art und Umfang der Komponenten und ihrer Wechselwirkungen
 - ⇒ Komplexität des Systems.



Hierarchische Abstraktionsebenen

- Vorgehensweise häufig *"von oben nach unten"* (engl.: *top-down*)

d.h. Informatiker beginnen oft mit einem Modell der Realität auf einer sehr hohen Abstraktionsebene und *konkretisieren* dieses Modell schrittweise zu immer detaillierteren Modellen, um sich einer Realisierung zu nähern.





Beispiel

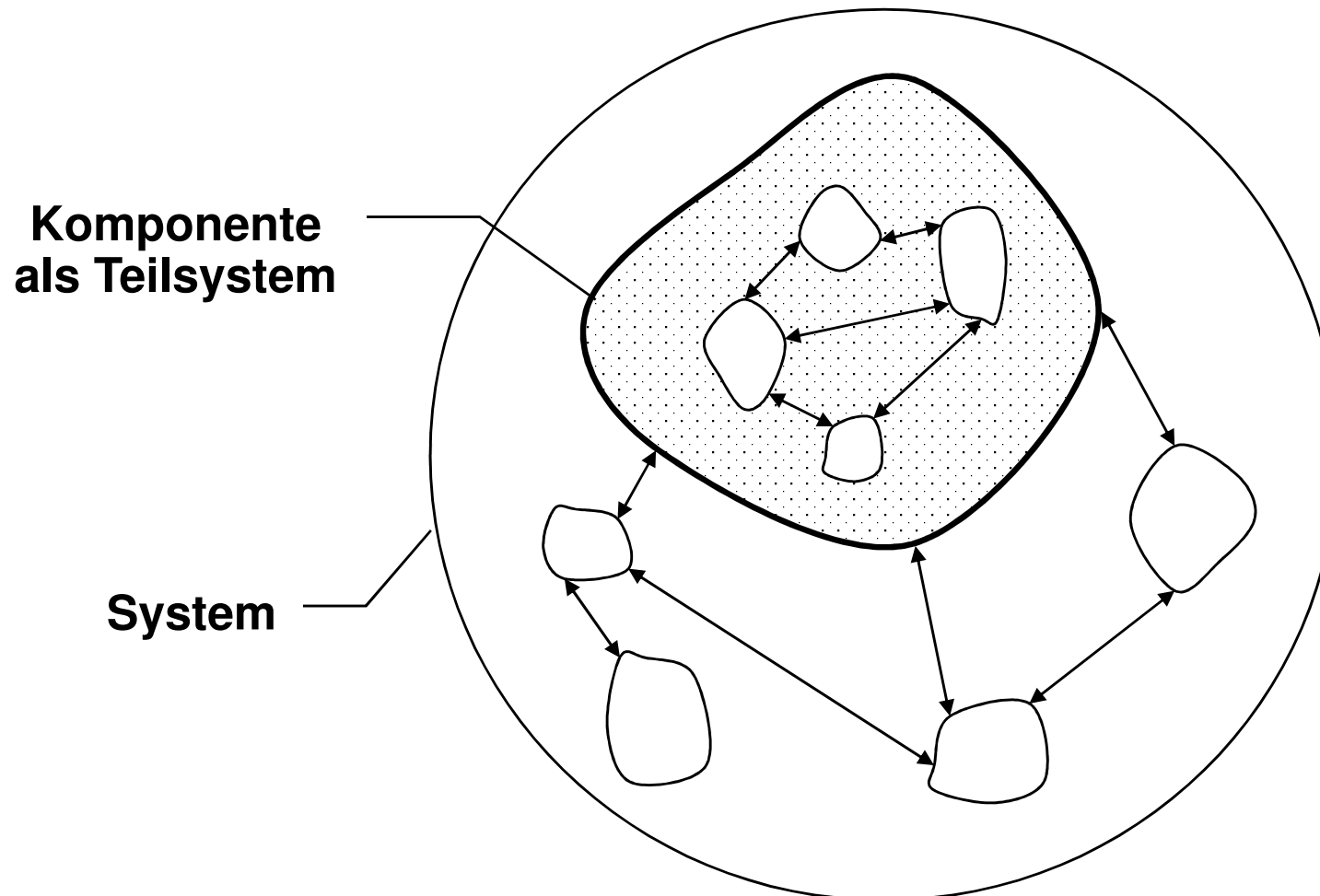
- **Ausschnitt aus den üblicherweise betrachteten Abstraktionsebenen eines Rechensystems**
- **wird im Verlaufe des Studiums konkretisiert**

		<u><i>typische Modelle</i></u>	<u><i>in Vorlesung</i></u>
6	Geschäftsprozess	Prozessketten	„E-Biz.“, evtl. BWL
5	Anwendungsprogramm	Datenflussdiagramm	Softwaretechnik
4	Betriebssystem	Prozesssysteme	Betriebssysteme
3	Prozessor	Maschinensprache	Rechnerorganisation
2	Funktionsblöcke	Register-Transfer-Sprache	Rechnerorganisation
1	digitale Signale	Gatter	Digitaltechnik
0	elektrische Signale	physikal. Modell	(Elektrotechnik)



Beispiel: Verfeinerung eines Systems

- Eine Komponente eines Systems kann auf der nächsttieferen Abstraktionsebene selbst wieder als System betrachtet werden.



2.3 Information und ihre Repräsentation

- **Information ist einer der zentralen Begriffe der Informatik:**
 - "*Informatik* ist die Wissenschaft von der systematischen Verarbeitung von Information."
 - Sie befasst sich mit Struktur, Eigenschaften und Beschreibungsmitteln von Informationen und informationsverarbeitenden Systemen und deren Betrieb und Anwendung" (vgl. Kap.1).
- **Bedeutung des Begriffs "Information" im täglichen Leben:**
 - zutreffende Aussagen über bestimmte Gegenstände, Zustände, Ereignisse oder Zusammenhänge in der realen Welt.

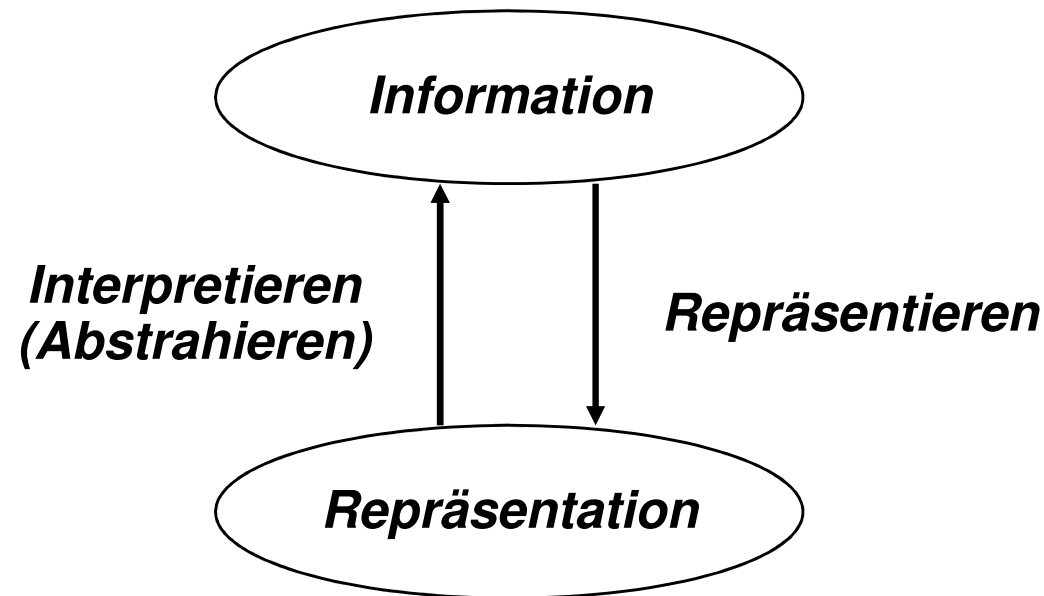
2.3 Information und ihre Repräsentation

- **Zur Bedeutung des Begriffs "Information" in der Informatik:**
 - **Unterschied: abstrakt, ohne Bezug zur realen Welt**
 - **d.h. abstrakter Bedeutungsgehalt von textuellen Ausdrücken, Grafiken, usw.**
 - **Information wird aber erst durch äußere Darstellungen verarbeitbar / kommunizierbar.**
- ⇒ **Die Informatik trennt strikt zwischen der abstrakten Information und ihren äußeren Darstellungen.**

✱ Information und Repräsentation - Definition

Def

- **Information** nennt man den abstrakten Bedeutungsgehalt (Semantik) einer Beschreibung, Aussage, Nachricht, usw.
- Äußere Form der Darstellung heißt **Repräsentation**.
- Übergang von der Repräsentation zur abstrakten Information heißt **Interpretation**, in umgekehrter Richtung spricht man von **Repräsentierung**.





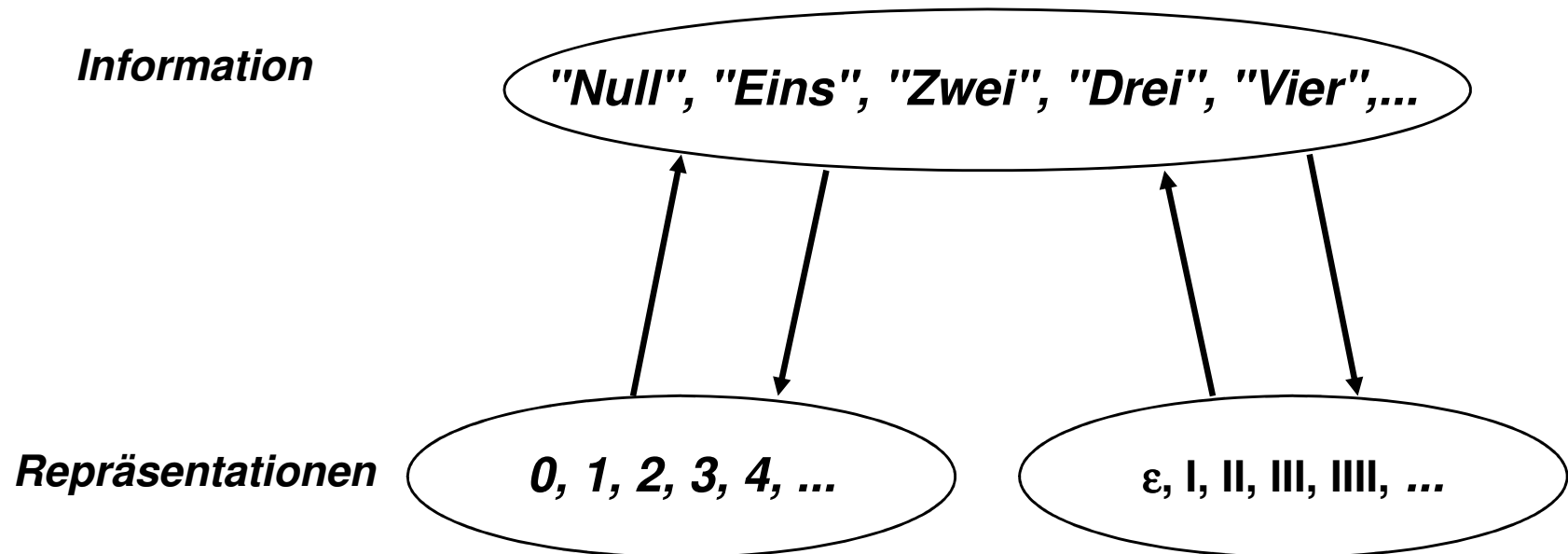
Anmerkungen

- **Typische Repräsentationen:**
 - Körperbewegungen (Handzeichen)
 - das gesprochene Wort (akustische Repräsentation)
 - Zeichenfolgen (das geschriebene Wort)
 - grafische Darstellungen (Zeichnungen, Ikonen, ...)
- **Festlegung für die Deutung von Repräsentationen notwendig. Durch Bedeutung wird die Repräsentation zu Information.**
- **Repräsentationen können mehrere Bedeutungen besitzen. Beispiel: Zeichenfolge "G", "R" "Ü" "N":**
- **Repräsentationssysteme sind i.d.R.**
 - unterschiedlich leistungsfähig (mächtig) und
 - abhängig von der darzustellenden Information unterschiedlich zweckmäßig in Hinblick auf die beabsichtigte Verarbeitung.



Anmerkungen (2)

- Dieselbe Information kann mehrere unterschiedliche (aber semantisch gleichwertige) Repräsentierungen besitzen.
- Beispiel: Die natürlichen Zahlen
 - Repräsentationssystem 1:
Notation üblicher Dezimalzahlen: 0, 1, 2, 3, 4, 5, ...
 - Repräsentationssystem 2:
Strichfolgen: leere Folge ϵ , I, II, III, IIII, IIII, ...





Verstehen

Def

- Herstellen von Beziehungen zwischen der in Repräsentationen enthaltenen abstrakten Information und der realen Welt wird **Verstehen** genannt.
- Verstehen einer Nachricht beinhaltet damit
 - Erkennen der Bedeutung der Nachricht (abstrakte Information)
und
 - Herstellen des Bezugs zur realen Welt.
- Verstehen ist ein subjektiver Prozess und nicht formalisierbar.

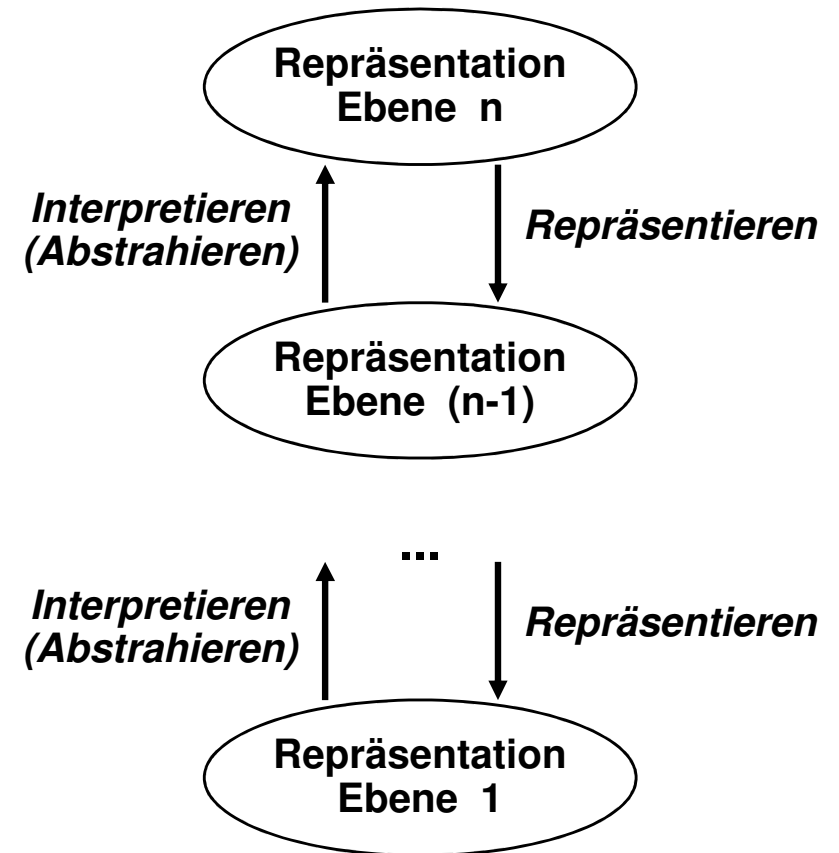
* Hierarchische Repräsentierungsebenen

- Vorgang der Repräsentierung / Interpretation kann wiederholt über mehrere Abstraktionsebenen erfolgen.

⇒

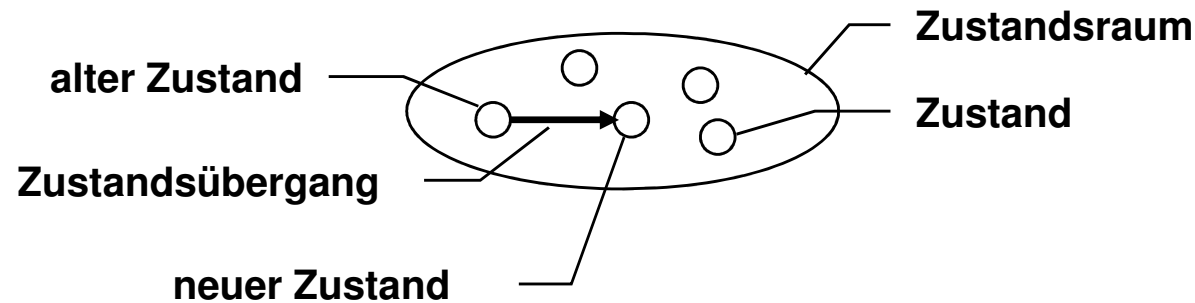
hierarchisch angelegte Repräsentierungssysteme für Information auf verschiedenen Abstraktionsstufen.

- Dieser Ansatz wird z.B. im Rahmen der Betrachtung von Datenstrukturen und der Programmierung eine große Rolle spielen.

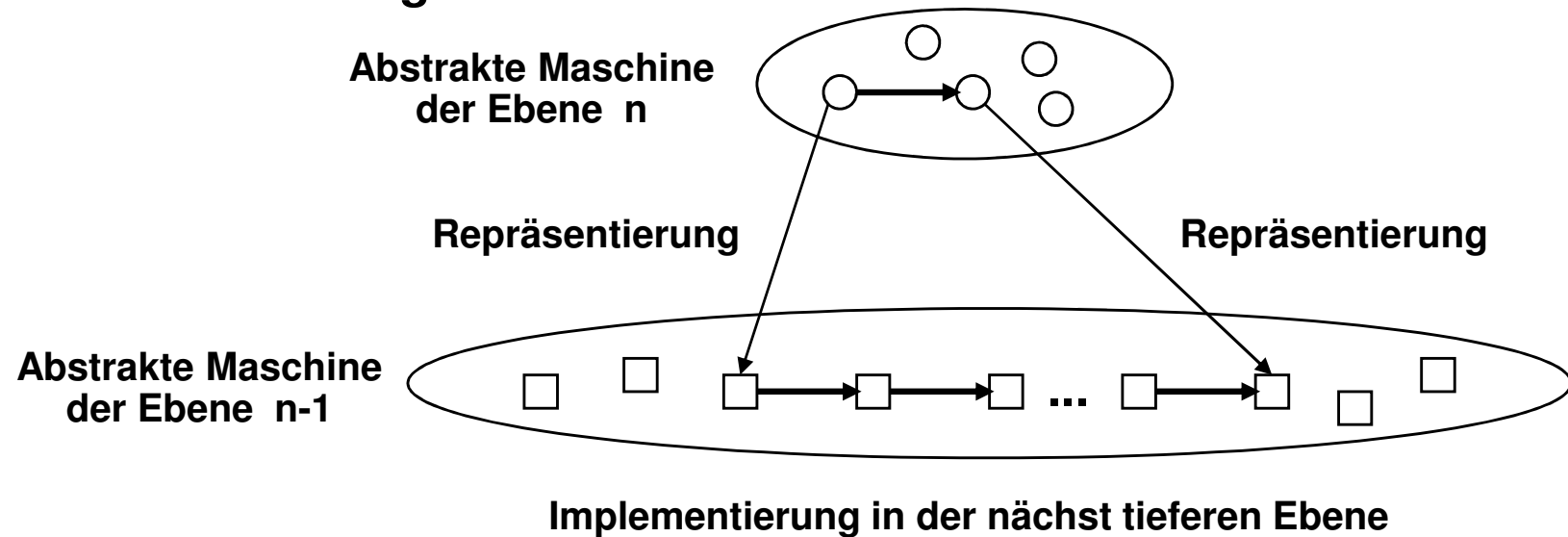


* Beispiel: Hierarchie abstrakter Maschinen

- **Repräsentationssystem jeder Ebene: abstrakte Maschine**



- **Hierarchiebildung**





Im Folgenden:

- **In den folgenden beiden Abschnitten werden zwei in der Informatik häufig eingesetzte Repräsentationssysteme vorgestellt:**
 - **(textuelle) formale Sprachen**
 - **Graphen**
- **Diese werden detailliert im weiteren Informatikstudium behandelt.**



2.4 Formale Sprachen

- **Für die automatisierte Informationsverarbeitung mit Rechensystemen sind textuelle Darstellungen immer noch am weitesten verbreitet:**
 - **für menschliche Benutzer lesbare Ein- /Ausgabe**
 - **Kommandosprachen (z.B. UNIX shell)**
 - **Programmiersprachen für Informatiker: C/C++, Java, ...**
 - **Auszeichnungssprachen: SGML, HTML, XML, ...**
- **In diesem Abschnitt:
Einführung des Begriffs der formalen Sprache**



Zeichen, Zeichenvorrat

Def

- Ein **Zeichen** (engl. *character*) ist ein Element einer vereinbarten endlichen, nicht-leeren Menge, die als **Zeichenvorrat** bezeichnet wird.
- Zeichenvorrat aus genau zwei verschiedenen Zeichen heißt **binärer Zeichenvorrat**.
- **Bit** (Abk. für **binary digit**) bezeichnet jedes Zeichen aus einem binären Zeichenvorrat.
- **Symbol**: (streng genommen) ein Zeichen zusammen mit einer vereinbarten Bedeutung. Häufig werden aber Zeichen und Symbol gleichwertig benutzt.

Beispiele:

$\{+, -, *, /\}$

$\{Mo, Di, Mi, Do, Fr, Sa, So\}$

$\{0, 1\}$, $\{dunkel, hell\}$, $\{0V, +5V\}$,
 $\{falsch, wahr\}$, $\{ja, nein\}$

i.d.R. $\{0, 1\}$.



Alphabet

Def

- Ein **Alphabet** Σ ist ein Zeichenvorrat, auf dem eine lineare Ordnung (Reihenfolge) für die Zeichen definiert ist.
- Beispiele:
 - $\{0,1\}, 0 < 1$
 - $\{0,1,2,3,4,5,6,7,8,9\}, 0 < 1 < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9$
 - $\{A,B,C, \dots, Z, a,b,c, \dots, z\}, A < B < C < \dots < Z < a < b < c < \dots < z.$



Zeichenketten

Def

- Eine endliche Folge $w=a_1...a_n$ von Zeichen eines Alphabets Σ heißt **Wort** oder **Zeichenkette** (engl.: *string*) über Σ .
- Sei $w=a_1...a_n$ Zeichenkette über Σ ,
 $|w|=n$ bezeichnet die **Länge** der Zeichenkette.
- Das **leere Wort** wird durch ε bezeichnet (*auch als "" geschrieben*), besitzt Länge 0.

Def

- $\Sigma^* :\Leftrightarrow$ Menge aller Zeichenketten über Σ
 $\Sigma^+ :\Leftrightarrow$ Menge aller nicht-leeren Zeichenketten über Σ
 $\Sigma^n :\Leftrightarrow$ Menge aller Zeichenketten der Länge n über Σ .
 - Beispiel: $\Sigma = \{0,1\}$, $\Sigma^* = \{0,1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots\}$
- $\Sigma^* = \{0,1\}^*$ heißt die Menge der Binärwörter,
Elemente von Σ^n heißen auch ***n*-Bit-Wörter** oder **Binärwörter der Länge *n***.



Konkatenation von Zeichenketten

Def

Seien Σ ein Alphabet, $u = a_1 \dots a_m$ und $v = b_1 \dots b_n$ Wörter über Σ .
Das Wort

$$w = uv = u||v = a_1 \dots a_m b_1 \dots b_n,$$

das durch Anfügen des Worts v an u entsteht, heißt
Konkatenation oder **Verkettung** von u und v .

$$\text{Es gilt: } |uv| = |u| + |v|.$$

- Ist $w \in \Sigma^*$ und n eine natürliche Zahl, dann bezeichnet w^n mit

$$w^0 := \varepsilon$$

$$w^{n+1} := w^n w$$

das Wort, das aus n aneinandergefügt Kopien von w besteht,

w^* bezeichnet ein beliebiges solches Wort
(n -fache Wiederholung von w für irgendein n),

w^+ ein nicht-leeres solches Wort.



Präfix / Suffix

Def

- Sind $x, y, z \in \Sigma^*$ (leere Wörter eingeschlossen) und ist

$$w = xyz = x//y//z,$$

dann heißt

x ein **Präfix** (*Anfangsstück*) von w

y ein **Teilwort** von w und

z ein **Suffix** (*Endstück*) von w .



Lexikographische Ordnung

Def

Sei Σ ein Alphabet und \leq die lineare Ordnung auf Σ .

Für Wörter $w_1, w_2 \in \Sigma^*$ wird nun ebenfalls eine Ordnung \leq_{lex} , die **lexikographische Ordnung**, induktiv durch folgende Festlegungen definiert:

$$\forall w \in \Sigma^* : \varepsilon \leq_{lex} w$$

$$\forall a_1, a_2 \in \Sigma :$$

$$a_1 // w_1 \leq_{lex} a_2 // w_2 \iff a_1 < a_2 \text{ oder } (a_1 = a_2 \text{ und } w_1 \leq_{lex} w_2)$$

- Die lexikographische Ordnung definiert eine lineare Ordnung auf Σ^* .

— Beispiele:

$$\Sigma = \{0, 1\}, 0 < 1$$

$$\varepsilon \leq_{lex} 0, 01 \leq_{lex} 1, 01 \leq_{lex} 10, 01 \leq_{lex} 011, 011 = 011$$



Formale Sprache

Def

- Sei Σ ein Alphabet. Eine Teilmenge $L \subseteq \Sigma^*$ heißt (formale) **Sprache**, $x \in L$ heißt **Wort der Sprache** L .

- Beispiel:

$$\Sigma = \{0,1\}, \quad L = \{1, 01, 001, 0001, 00001, \dots\} \subseteq \Sigma^*.$$

(Man kann L auch durch den Ausdruck 0^*1 charakterisieren).



Operationen auf formalen Sprachen

Def

- Sei Σ ein Alphabet und seien $L, M \subseteq \Sigma^*$ formale Sprachen.
 - $L \cup M$ bzw. $L \cap M$ bezeichnen (wie allg. für Mengen) die *Vereinigung* bzw. den *Durchschnitt* der beiden Sprachen L und M .
 - $LM = \{ uv \mid u \in L \text{ und } v \in M \}$ bezeichnet die *Konkatenation* der Sprachen L und M . Kurzschreibweisen: $L^2 = LL$, $L^n = LL \dots L$
 - L^* definiert durch $L_0 = \varepsilon$, $L_{n+1} = L_n L$, $L^* = \bigcup L_n$ beinhaltet die Menge aller Wörter, die durch Verkettung einer beliebigen Anzahl von Wörtern aus L entstehen (sog. *abgeschlossene* oder *Kleene'sche Hülle*).
 - $L^+ = L^* \setminus \{\varepsilon\}$

- **Beispiel:**

$$\Sigma = \{0, 1\}, L = \{01, 0001\} \subseteq \Sigma^*$$

$$L^* = \{\varepsilon, 01, 0101, 0001, 010101, 010001, 000101, \dots\}$$



Codes

Def

- Seien A und B Zeichenvorräte. Ein **Code** oder eine **Codierung** ist eine Abbildung

$$c:A \rightarrow B \text{ oder } c:A^* \rightarrow B^*.$$

(d.h. zwischen Zeichenvorräten A und B und auch zwischen Wörtern über Zeichenvorräten).

- Die Bildmenge $\{b \in B \mid b=c(a), a \in A\}$ unter c , d.h. die Menge der Codewörter von c , wird ebenfalls **Code** genannt.
- Die Elemente von A werden auch **Klarzeichen** genannt, die Elemente von B auch **Codezeichen**.
- Die Abbildung eines Codes kann partiell sein, d.h. nicht für jedes Wort aus A^* muss eine Darstellung existieren.



Decodierung

- In der Regel ist die Abbildung eines Codes injektiv, d.h. verschiedene Zeichen oder Wörter werden auf verschiedene Codewörter abgebildet.

Def

- Dann ist auf der Bildmenge eine umkehrbare Codierung beschrieben durch eine Abbildung

$$d: \{b \in B \mid b = c(a), a \in A\} \rightarrow A,$$

die *Decodierung* genannt wird.



Binär-Codierung

Def

Für die Informationsdarstellung in Rechensystemen werden fast ausschließlich *Binär-Codierungen (Binär-Codes)* von Alphabeten betrachtet.

Dies sind Codierungen der Form

$$c:A\rightarrow\{0,1\}^*,$$

wobei A ein vorgegebenes Alphabet ist.



2.5 Graphen und Bäume

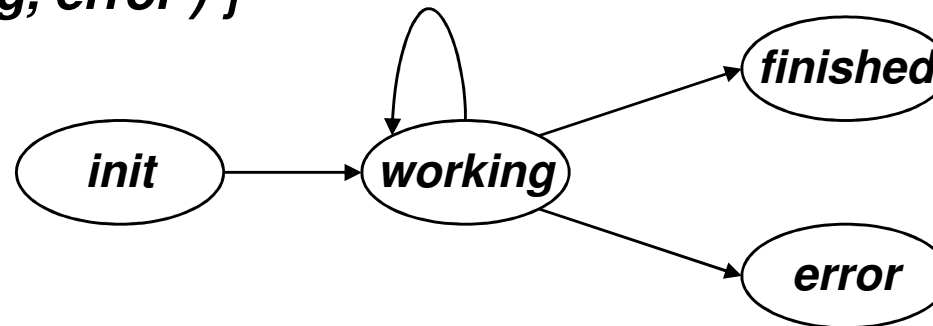
- **Graphen:** *strukturelle Modelle*
d.h. mit ihnen können identifizierte Objekte und ihre Beziehungen zueinander beschrieben werden.
- Graphen werden in der Informatik oft verwendet.
- Hier:
 - als formales Modell des intuitiven Systembegriffs
 - als weiteres konkretes Repräsentierungssystem für Information
- **Bäume:** spezielle Arten von Graphen.



Graph

Def

- Ein **gerichteter Graph** (engl. *graph*) $G = (V, E)$ ist ein Paar, bestehend aus einer endlichen, nichtleeren Menge V zusammen mit einer Relation $E \subseteq V \times V$.
 - V heißt die Menge der **Knoten** (engl.: *vertices*) des Graphen G .
 - E heißt die Menge der **Kanten** (engl.: *edges*) von G .
 - Notation: Eine Kante $(a, b) \in E$ wird graphisch durch einen Pfeil von Knoten a zu Knoten b dargestellt.
- **Beispiel**:
 - $G = (V, E)$ mit $V = \{ \textit{init}, \textit{working}, \textit{finished}, \textit{error} \}$ und
 - $E = \{ (\textit{init}, \textit{working}), (\textit{working}, \textit{working}), (\textit{working}, \textit{finished}), (\textit{working}, \textit{error}) \}$





Graph (2)

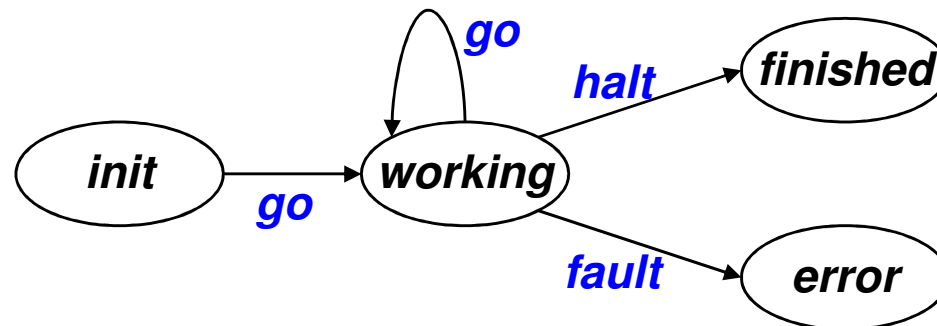
- **Ungerichtete Graphen:**
Bei Kanten werden Richtungen nicht angenommen,
d.h. die Reihenfolge der Knoten zur Bezeichnung einer Kante ist unerheblich.
- Ein Graph $G = (V, E)$ heißt **markiert** (*bewertet, attribuiert*),
wenn jedem Knoten (*knotenmarkiert*) oder jeder Kante
(*kantenmarkiert*) (oder beiden) durch eine Abbildung weitere
Größen (Werte des Bildbereichs der Abbildung) zugeordnet sind.

Def



Beispiel

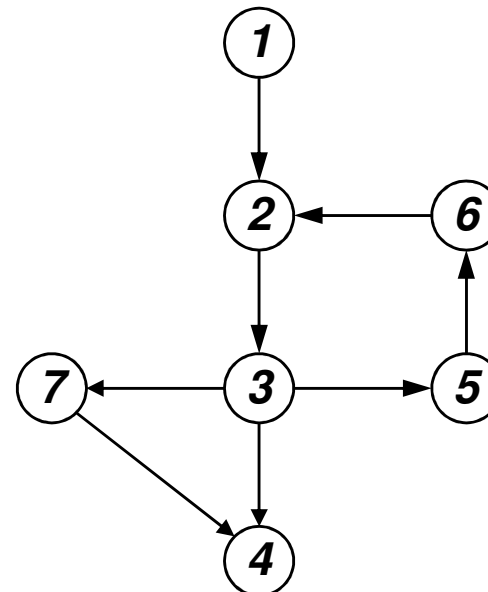
- $G = (V, E)$ mit
 - $V = \{ \text{init}, \text{working}, \text{finished}, \text{error} \}$ und
 - $E = \{ (\text{init}, \text{working}), (\text{working}, \text{working}), (\text{working}, \text{finished}), (\text{working}, \text{error}) \}$
 - Kantenbewertung $\text{action}: E \rightarrow \{ \text{go}, \text{halt}, \text{fault} \}$



✱ Gerichteter Kantenzug, gerichteter Weg

Def

- Sei $G = (V, E)$ ein gerichteter Graph. Sei $z = (v_0, \dots, v_n)$ eine Folge von $n+1$ Knoten des Graphen mit $(v_0, v_1), \dots, (v_{n-1}, v_n) \in E$; dann heißt z **gerichteter Kantenzug** in G der Länge n .
(Die Folge der Knoten ist durch Kanten verbunden, mehrfaches Durchlaufen von Knoten ist erlaubt).
- Sei $G = (V, E)$ ein gerichteter Graph. Ein gerichteter Kantenzug $w = (v_0, \dots, v_n)$ in G heißt **gerichteter Weg** in G , wenn alle Knoten verschieden sind.
- Beispiele:
 - $(3, 5, 6, 2, 3, 4)$ ist ein **gerichteter Kantenzug**
 - **Wege** sind z.B. $(1, 2, 3, 7, 4)$ und $(2, 3, 5, 6)$





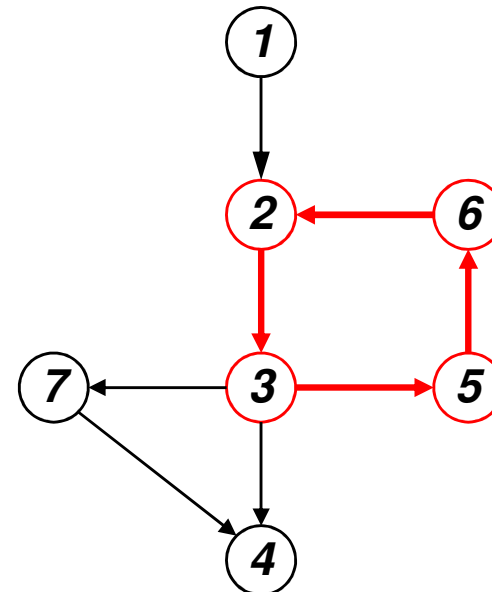
Zyklus

Def

- Sei $G = (V, E)$ ein gerichteter Graph und $w = (v_0, \dots, v_n)$ ein gerichteter Weg in G . Dann heißt $c = (v_0, \dots, v_n, v_{n+1})$ **Zyklus**, wenn $(v_n, v_{n+1}) \in E$ und $v_{n+1} = v_0$ (d.h. Anfangs- und Endknoten stimmen überein).
- Ein entarteter Zyklus $(v_i, v_i) \in E$ heißt **Schlinge** (von einem Knoten unmittelbar in ihn zurück).
- Ein Graph heißt **zyklenfrei**, wenn er keinen Zyklus enthält.

- **Beispiel:**

- $G = (V, E)$, $V = \{1, 2, 3, 4, 5, 6, 7\}$,
 $E = \{ (1, 2), (2, 3), (3, 4), (3, 5), (5, 6), (6, 2), (3, 7), (7, 4) \}$
- $(2, 3, 5, 6, 2)$ ist ein **Zyklus**.

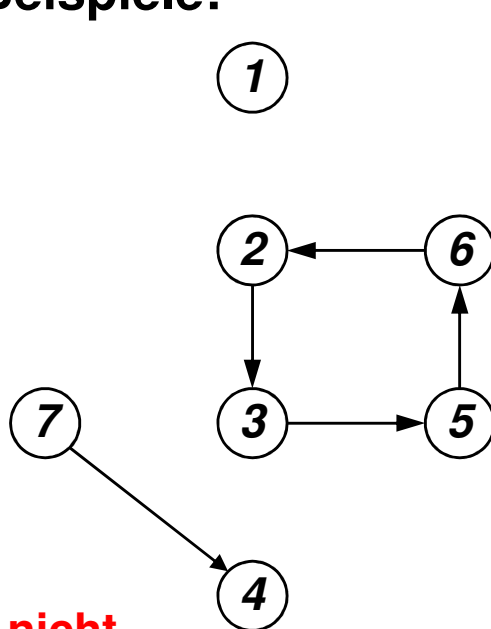




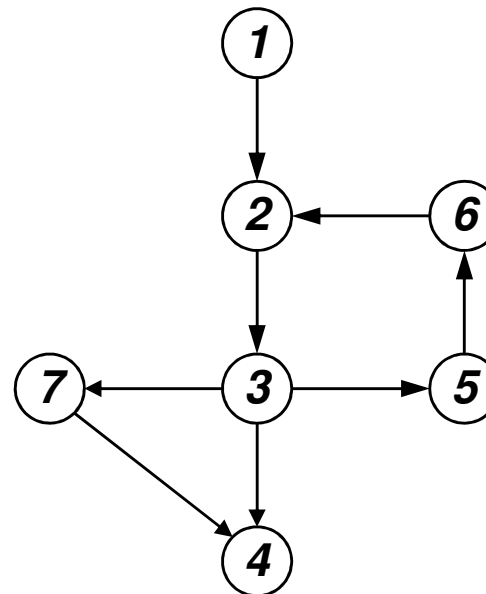
Zusammenhängender Graph

Def

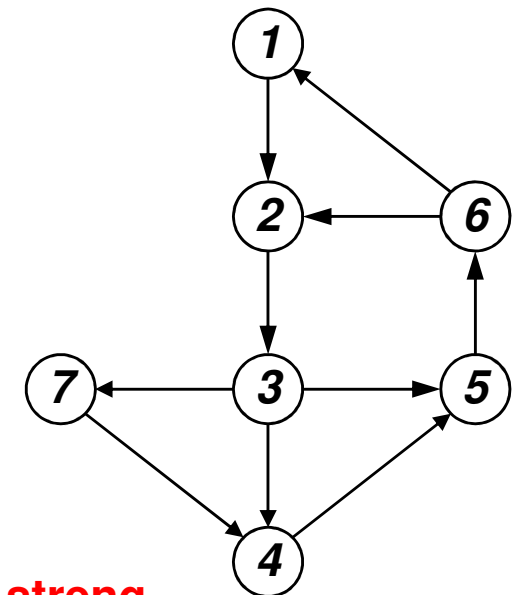
- Ein gerichteter Graph $G = (V, E)$ heißt **zusammenhängend**, wenn es für je zwei Knoten $v_1, v_2 \in V$ mindestens einen gerichteten Weg zwischen ihnen in G gibt.
- Der Graph heißt **streng zusammenhängend**, wenn es für je zwei Knoten $v_1, v_2 \in V$ einen Weg von v_1 nach v_2 und umgekehrt gibt (d.h. jeder Knoten kann von jedem anderen aus erreicht werden).
- Beispiele:



nicht
zusammenhängend



zusammenhängend
nicht streng zusammenhängend

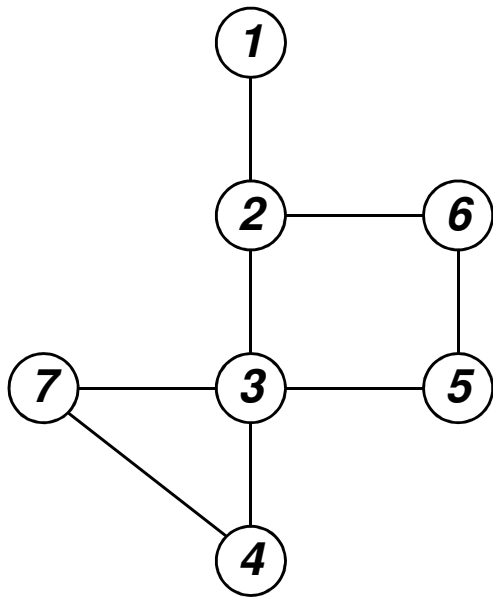


streng
zusammenhängend

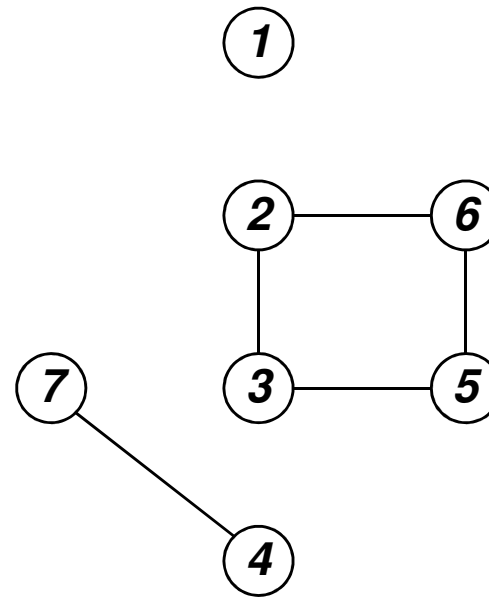


Zusammenhängender Graph (2)

- Ergänzung: Ein ungerichteter Graph heißt zusammenhängend, wenn es für je zwei Knoten $v_1, v_2 \in V$ mindestens einen ungerichteten Weg zwischen ihnen gibt.
- Beispiel:



zusammenhängend



nicht zusammenhängend



Gerichteter Baum

Def

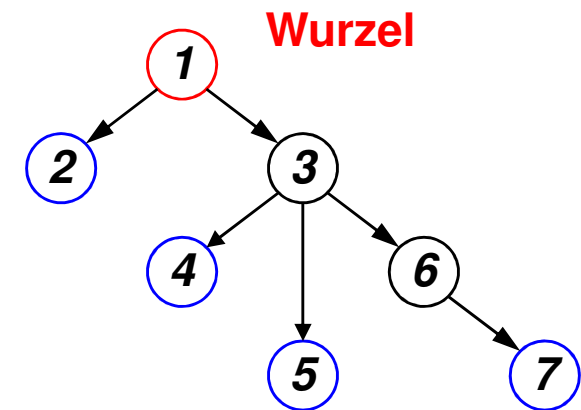
- Sei $B = (V, E)$ ein gerichteter Graph. B heißt *baumartig* oder kurz *Baum* (engl.: *tree*), wenn gilt:
 - B ist zusammenhängend und zyklensfrei.
 - Es gibt genau einen Knoten $v_w \in V$, in den keine Kante mündet. Dieser Knoten heißt **Wurzel** des Baumes.
 - Von der Wurzel v_w des Baumes gibt es zu jedem anderen Knoten $v \in V$, $v \neq v_w$ genau einen gerichteten Weg.
- Ein Knoten v heißt **Blatt** oder *Endknoten*, wenn er keine ausgehende Kante besitzt, d.h. wenn kein v' existiert mit $(v, v') \in E$.

Beispiel:

$B = (V, E)$

$V = \{1, 2, 3, 4, 5, 6, 7\}$

$E = \{ (1, 2), (1, 3), (3, 4), (3, 5), (3, 6), (6, 7) \}$



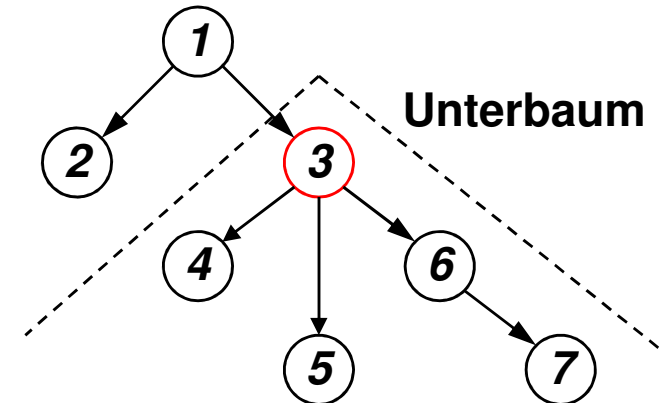
Die Knoten 2, 4, 5 und 7 sind die **Blätter** von B .



Gerichteter Baum (2)

Def

- Die Knoten $v' \in V$, die von einem Knoten v durch eine einzige Kante $(v, v') \in E$ erreicht werden, heißen **Söhne** oder **Kinder** von v (umgekehrt **Vater**).
- Die Gesamtheit aller von v (auch über Zwischenknoten) erreichbaren Knoten heißen die **Nachfahren** von v . Diese bilden wiederum einen Baum, für den v die Wurzel ist. Dieser Baum heißt auch der von v **aufgespannte Unterbaum**.
- Die Knoten auf dem Weg von der Wurzel bis vor v heißen die **Vorfahren** von v .



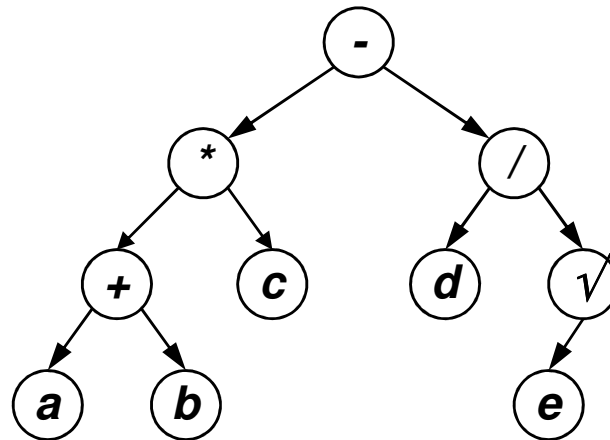
- Die Knoten 2 und 3 sind die Söhne von 1.
- 4, 5, 6, 7 sind die Nachfahren von 3.
- 1 und 3 sind die Vorfahren von 5.



Binärer Baum

Def

- Sei $B = (V, E)$ ein gerichteter Baum. B heißt **binärer Baum** oder **Binärbaum**, wenn jeder Knoten höchstens zwei Söhne hat und zwischen dem *linken Unterbaum* und dem *rechten Unterbaum* unterschieden wird.
- Beispiel: Arithmetischer Ausdruck $(a+b)*c-d/\sqrt{e}$



Operanden sind Blätter

- Im Baum werden keine Klammern benötigt
- vgl. Eingabe bei Taschenrechnern („Umgekehrte Polnische Notation“, etwa bei HP-Modellen)



2.6 Algorithmen

- In diesem Abschnitt soll ein weiterer Aspekt von Informatik angerissen werden:
"Informatik ist die Wissenschaft von der systematischen Verarbeitung von Information" (vgl. Kap.1).
- Die automatisierte Verarbeitung verlangt, dass die Verarbeitungsvorschrift
 - in ihrer Bedeutung exakt festgelegt ist,
 - eine geeignete Repräsentation in einer formalen Sprache oder einer graphischen Darstellungsform besitzt
 - und letztlich durch einen Prozessor eines Rechensystems ausführbar ist.
- Der in der Informatik verwendete Begriff für derartige Verarbeitungsvorschriften ist der des *Algorithmus*.



Einordnung

- **In der Theoretischen Informatik**
 - Algorithmus-Begriff wird exakt über math. Konzepte eingeführt
 - z.B. Markov-Algorithmen, Turing-Maschinen.
- **Hier: *Intuitiver* Algorithmus-Begriff**
 - Konkrete Algorithmen (z.B. für Sortierprobleme unter Nutzung bestimmter Datenstrukturen) werden in der Vorlesung „Algorithmen und Datenstrukturen“ im 2. Fachsemester behandelt.



Einordnung (2)

- **Herkunft des Begriffs Algorithmus (vgl. Kap.1):**
 - Rechenbuch von Muhammed ibn Musa Al-Chwarizmi
 - ca. 1750 in Zusammenhang mit den vier Grundrechenarten benutzt
 - Ab Mitte dieses Jahrhunderts zur Bezeichnung einer allgemeinen Handlungs- und Bearbeitungsvorschrift
- **Nicht-präzise Verarbeitungsvorschriften aus dem täglichen Leben:**
 - Kochrezept
 - Strick- und Häkelmuster
 - Bedienungsanleitung / Gebrauchsanweisung



Intuitiver Algorithmus-Begriff

Def

- Ein **Algorithmus** ist ein Verfahren mit einer *präzisen* (d.h. in einer genau festgelegten Sprache abgefassten) *endlichen* Beschreibung unter Verwendung *effektiver* (d.h. tatsächlich ausführbarer) elementarer Verarbeitungsschritte zur Lösung einer Klasse gleichartiger Probleme.
- Anmerkungen:
 - Unterscheidung zwischen dem Algorithmus und seiner Beschreibung (d.h. Repräsentation).
 - Das aus einer Klasse speziell zu bearbeitende Problem wird durch Eingabe-Parameter bestimmt.
 - Algorithmen liefern für Eingaben i.d.R. Resultate als Ausgaben. Algorithmus entspricht in diesem Sinne einer partiellen Abbildung.
 - Zur Lösung einer Problemklasse gibt es i.d.R. verschiedene Algorithmen.
 - Abhängig von den zur Verfügung stehenden elementaren Aktionen können Algorithmen zur Lösung derselben Problemklasse sehr unterschiedlich ausfallen.



Wichtig

- **Unabhängig von der Beschreibungsform ist es bei Algorithmen wichtig, die folgenden Aspekte zu unterscheiden:**
 - die Aufgabenstellung, d.h. die zu lösende Problemklasse.
 - Die Art und Weise, wie die Aufgabe bewältigt wird, unterschieden nach
 - den elementaren Verarbeitungsschritten, die zur Verfügung stehen,
 - der Beschreibung der Auswahl der einzelnen auszuführenden Schritte.



Eigenschaften von Algorithmen

- Merkmale eines Algorithmus zu seiner Beurteilung
- Ein Algorithmus heißt *für eine Eingabe*
 - **terminierend**: endet stets nach endlich vielen Schritten
 - **deterministisch**: keine Freiheit in der Auswahl der Verarbeitungsschritte
 - **determiniert**: Resultat/Endzustand des Algorithmus eindeutig bestimmt
 - **korrekt**: im Endzustand liegt eine Lösung des Problems vor
 - **sequenziell**: Folge von Verarbeitungsschritten
 - **parallel**: gewisse Verarbeitungsschritte werden nebeneinander ausgeführt
- Ein Algorithmus heißt insgesamt terminierend (*deterministisch, determiniert, korrekt, sequenziell*), wenn der Algorithmus diese Eigenschaft für jede zulässige Eingabe besitzt.

Def



Beispiel

- Euklids Algorithmus zur Berechnung des größten gemeinsamen Teilers (ggT).
- Aufgabenstellung: Gegeben seien zwei ganze Zahlen a und b mit $a > 0$ und $b > 0$. Gesucht wird der größte gemeinsame Teiler $ggT(a, b)$ von a und b .
- Algorithmus für $ggT(a, b)$ nach Euklid:
 - (1) falls $a = b$, dann ist $ggT(a, b) = a$;
 - (2) falls $a < b$, dann wende den Algorithmus ggT an auf $(a, b - a)$.
 - (3) falls $b < a$, dann wende den Algorithmus ggT an auf $(a - b, b)$.
- Anmerkungen:
 - arithm. Operation "-" und Vergleichsoperationen "<" und "=" werden als die elementaren Verarbeitungsschritte angenommen.
 - Lässt man die Einschränkungen $a > 0$ und $b > 0$ weg, so erhält man einen Algorithmus, der für ungleiche negative Zahlen nicht terminiert.
 - Der Algorithmus ist
 - sequenziell
 - deterministisch (damit auch determiniert, Umkehrung gilt nicht!)
 - korrekt.



Beobachtung

- **Klassische Elemente in der Beschreibung von Algorithmen sind:**
 - Ausführung elementarer Schritte
 - Fallunterscheidung über Bedingungen
 - Wiederholung und Rekursion
- **Diese Elemente treten in ähnlicher Form in allen Systemen zur Repräsentierung von Algorithmen auf.**
- **Sie bilden auch die Grundlage jeder Programmierausbildung (vgl. Vorlesung OOSE).**



Güte von Algorithmen

- Beim Vergleich von Algorithmen interessieren nicht nur die o.a. Eigenschaften, vielmehr sind auch Maße (Vergleichsmaßstäbe) für ihre Effizienz gefragt.

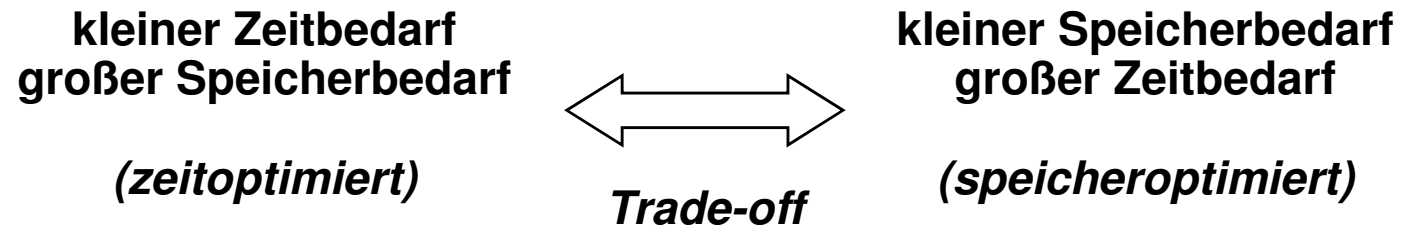
Def

- Unter der **Komplexität** eines Algorithmus versteht man den Aufwand in Abhängigkeit vom Anfangszustand, der durch die Ausführung des Algorithmus entsteht, gemessen in
 - Speicherbedarf zur Speicherung von internen Zuständen usw.
 - Zeitbedarf, gemessen in der Anzahl der benötigten Schritte



Güte von Algorithmen (2)

- I.d.R. besteht Zielkonflikt zw. Speicherbedarf und Zeitbedarf:



- Eine ausführlichere Behandlung der Komplexität von Algorithmen erfolgt in den Vorlesungen „Algorithmen und Datenstrukturen (ADS)“ sowie „Automatentheorie und Formale Sprachen (AFS)“



Repräsentierung von Algorithmen

- Die Beschreibung eines Algorithmus erfolgt in einer Sprache. Beispiele sind etwa:
 - natürliche Sprache (Kochrezept: "Man nehme ...")
 - halbformale Sprache
(Strickmuster: * 2 re, 2 li; ab * wdh. bis Ende)
 - mathematische Formeln ($f(x)=3x^2+7x+5$)
 - Graphen
 - z.B. Straßenkarte für eine Zielfahrt,
 - elektrischer Schaltplan,
 - *Unified Modeling Language* UML,
(vgl. Vorlesung Softwaretechnik).



Repräsentierung von Algorithmen (2)

- **Weitere Beispiele:**
 - **Programmiersprachen verschiedener Abstraktionsebenen und Anwendungsbereiche (vgl. Vorlesung OOSE)**
 - programmierbare Taschenrechner,
 - Maschinensprache
 - Assembler,
 - C/C++, Java, Ruby, ... (Universelle Programmiersprachen)
 - APL (Mathematik),
 - XSLT, XQuery (Auszeichnungssprachen)
 - Structured Query Language (SQL, für Datenbanken).
 - **Hardware-Beschreibungssprachen (vgl. Vorlesung Rechnerorganisation), z.B.**
 - VHDL (Beschreibung von Verfahren, die in Hardware ablaufen)



Programmiersprachen, Programme

- Für die Informatik sind nur Sprachen interessant, die eine exakte Festlegung der Algorithmen erlauben, da nur so eine maschinelle Verarbeitung erfolgen kann.

Def

- **Syntax** einer Sprache: definiert die zulässigen Anordnungen der Sprachelemente auf der Ebene der Repräsentation.
- **Semantik** einer Sprache: definiert eine Interpretation und legt fest, wie die Sprachelemente in Hinblick auf das Problemlösungsverfahren zu interpretieren sind.
- **Programmiersprache**: eine formale Sprache zur Repräsentation von Algorithmen. Ein in einer solchen Programmiersprache beschriebener Algorithmus heißt **Programm**.



Ausführung eines Programms

Def

Prozessor: eine ein Programm ausführende Instanz

Prozess: Ausführung eines Programms für ein konkretes Problem

- **Vorgehensweise**: Prozessor liest die Repräsentation des Programms, interpretiert diese in Hinblick auf die Problemlösung.
Er führt die darin vorgesehenen elementaren Aktionen aus.
- Die Ausführung paralleler Algorithmen führt zu **nebenläufigen Prozessen**.



Anmerkungen

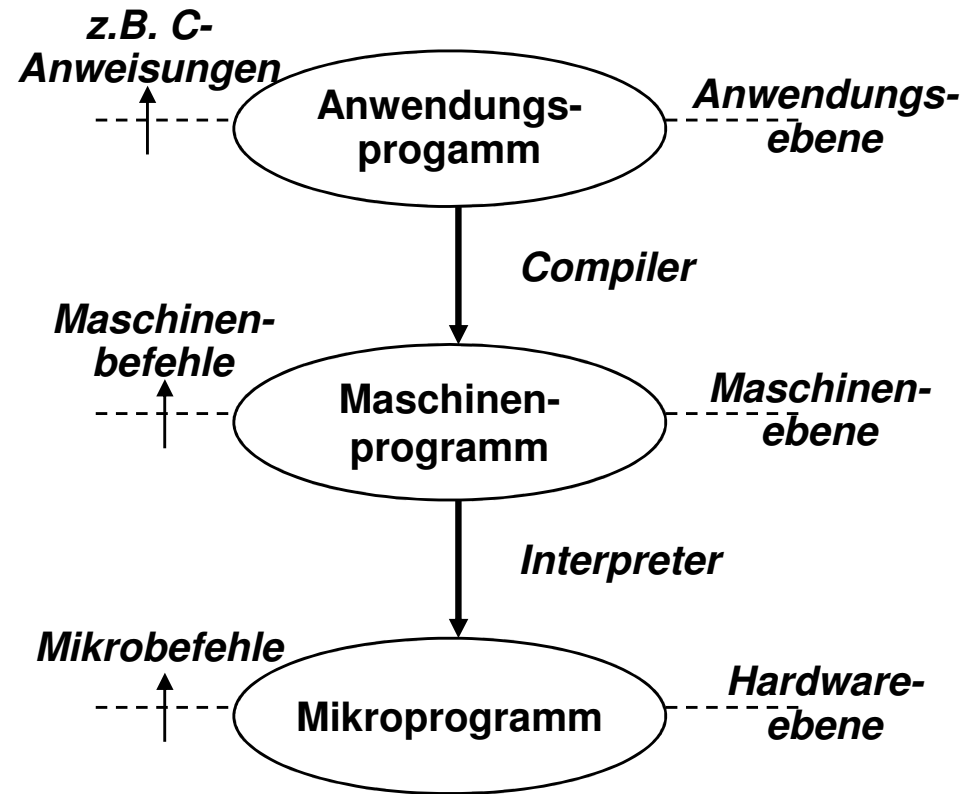
- **Nur wenige Programmiersprachen bieten ein Konzept für Parallelität.**
- **Nebenläufigkeit ("Quasiparallelität") wird detailliert in der Vorlesung Betriebssysteme (3. Semester) besprochen.**
- **Betriebssysteme unterstützen mit ihrem Prozesskonzept die nebenläufige Ausführung von Programmen (bei Vorhandensein mehrerer Prozessoren bzw. Prozessorkerne in einem Rechensystem findet die Ausführung tatsächlich parallel statt).**



Programme verschiedener Abstraktionsebenen

- Algorithmen sind auf verschiedenen Abstraktionsebenen definierbar. Diese gehen einher mit dem angenommenen Vorrat an elementaren Aktionen.
- Unterschieden mindestens: *Maschinenebene* und *Anwendungsprogrammebene*.
- Übersetzung von Programmen zwischen verschiedenen Abstraktionsebenen:
 - Compiler
 - Interpreter

Beispiel:





Abstraktionsebenen in Anwendungsprogrammen

- **Moderne Programmiersprachen (wie C++, Java, Smalltalk, Ruby) unterstützen die Definition problemangepasster (benutzerdefinierter) Abstraktionsebenen.**
- **Damit ist es Anwendungsprogrammierern möglich, sich im Sinne von abstrakten Maschinen eigene, auf der betrachteten Ebene als elementar angesehene Objekte und Aktionen zu definieren.**
- **Vorteil:**
 - **Übergang zwischen den verschiedenen Arbeitsphasen bei der Realisierung informationstechnischer Systeme wird erleichtert (von Systemanalyse über Systementwurf zur Implementierung; vgl. Vorlesungen Programmiermethoden und -techniken und Softwaretechnik).**
- **Diese Abstraktionsebenen innerhalb eines Anwendungsprogramms sind auf der Maschinenebene heutiger Prozessoren nicht sichtbar.**

* Beispiel: Primzahlssuche (einfacher Algorithmus)

a) Lösung in „C“

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

int main( int argc, char* argv[])
{
    int i, imax, n, n1, n2;

    n1 = atoi(argv[1]);
    n2 = atoi(argv[2]);
    for (n=n1; n<=n2; n++) {
        imax = (int) sqrt((double) n);
        for (i=2; i<=imax; i++)
            if (n%i==0) goto no_prime;
        printf("%d\n", n);
    no_prime:
        continue;
    }
    return 0;
}
```

b) Lösung in Hochsprache „Ruby“

```
(ARGV[0].to_i .. ARGV[1].to_i).each do |n|
    max = Math::sqrt(n).to_i
    puts n unless (2..max).find { |i| n%i==0 }
end
```

Der höhere Abstraktionsgrad der Hochsprache

- gestattet die Verwendung kompakter, problemangepasster Sprachelemente
- ermöglicht die Formulierung gut lesbaren und dennoch sehr kurzen Quellcodes
- führt so zu kürzeren Entwicklungszeiten
- erfordert mehr Rechner-Ressourcen zur Laufzeit