



Einführung in die Informatik I  
Dinner for 2001

Prof. Bernd Brügge, Ph.D  
Dr. Christian Herzog  
Technische Universität München

Wintersemester 2000/2001  
8. Januar 2001

# Überblick über die heutige Wiederholungs-Vorlesung

- ❖ Einführung: Realität, Model
- ❖ Informatik-Systeme
- ❖ Klassen und Algebren
- ❖ Algorithmen und Textersetzungssysteme
- ❖ Boolesche Algebra
- ❖ Aussagenlogik
- ❖ Termersetzungssysteme
- ❖ Funktionale Programmierung
- ❖ Imperative Programmierung
  
- ❖ Wichtig: Der Streifzug durch die Folien aus 2000
  - soll einen schnellen Überblick geben und „aufwärmen“
  - ist nicht gedacht als vollständige Darstellung des Klausurstoffes.

# Wirklichkeit und Modell

## ❖ Wirklichkeit W:

- Dinge, Personen, Abläufe in der Zeit
- Beziehungen zwischen diesen Gegenständen

## ❖ Modell M:

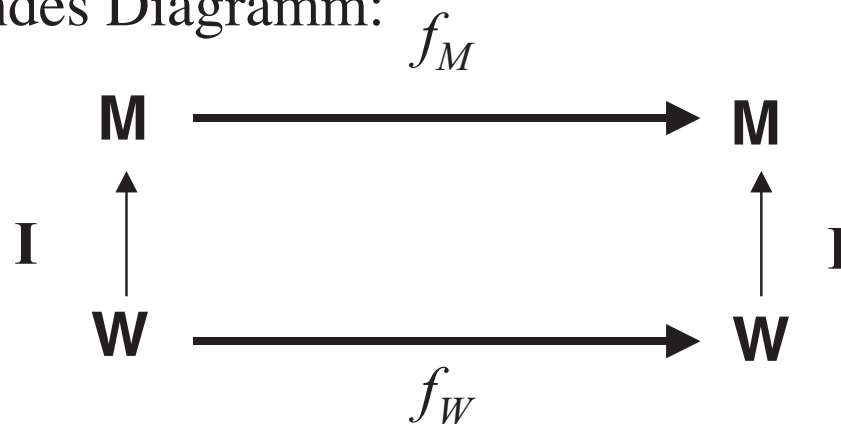
- Begriffe von (real existierenden oder nur gedachten) Dingen
- Begriffe von Personen
- Begriffe von Abläufen in der Zeit
- Beziehungen zwischen diesen Begriffen.

## ❖ Wirklichkeit nennen wir im folgenden auch oft Realität

# Was ist ein “gutes” Modell?

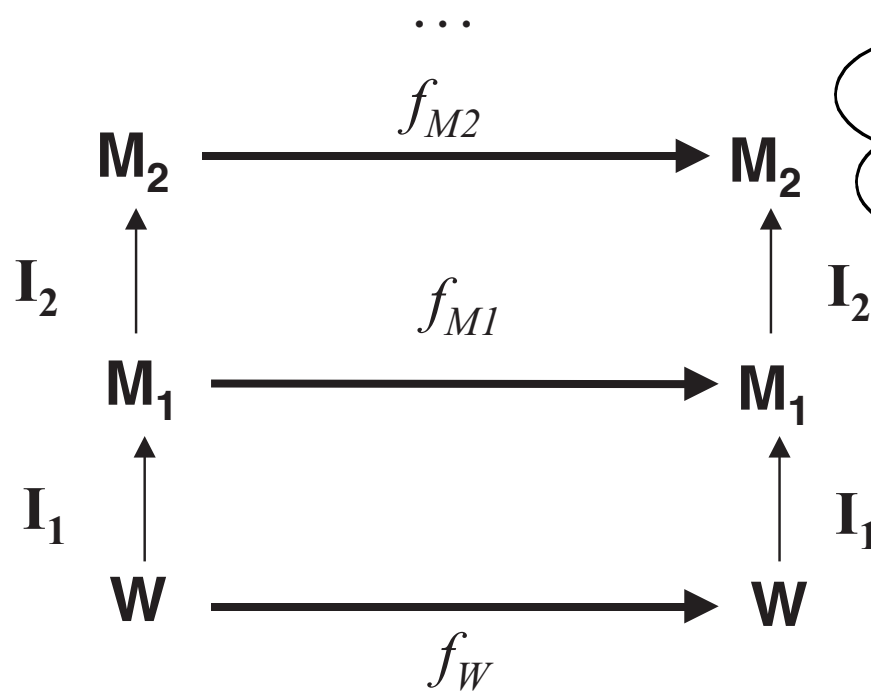
- ❖ Beziehungen, die in der Wirklichkeit  $W$  gelten, gelten auch im Modell  $M$ .
  - $I$  : Abbildung, die wirkliche Gegenstände in  $W$  auf ihre Begriffe in  $M$  abbildet (Interpretation)
  - $f_M$ : beliebige Beziehung zwischen Begriffen in  $M$
  - $f_W$ : beliebige Beziehung zwischen Gegenständen in  $W$

- ❖ Es gilt folgendes Diagramm:



# Modelle von Modellen von Modellen ...

- ❖ Modellierung ist ein relativer Begriff.
- ❖ Man kann auch ein Modell wieder als reale Welt auffassen und davon ein weiteres Modell (mit noch mehr Abstraktionen) erstellen.



Die Erstellung von Informatik-Systemen kann als Transformation von Modellen gesehen werden: Analyse, Entwurf, Implementation, Testen

# Was ist ein System?

- ❖ Ein System ist eine gedankliche Konstruktion
- ❖ Ein System kann in vielerlei Hinsicht gesehen werden.
- ❖ 2 wichtige Ansichtsarten:
  - Statische Sicht: Ein System ist eine komplexe Ansammlung von Elementen plus Beziehungen zwischen den Elementen
  - Dynamische Sicht: Ein System ist eine Folge von Zustandsübergängen
- ❖ Definition:
  - Ein Informatik-System ist ein System, das auf einem Rechner ausgeführt wird.

# Informatik-Systeme

- ❖ Diese Vorlesung beschäftigt sich mit Informatik-Systemen
  - Informatik-Systeme werden oft auch Datenverarbeitungssysteme genannt
- ❖ Wir kategorisieren Informatik-Systeme in 5 Typen:
  - Berechnung von Funktionen
  - Interaktive Systeme
  - Prozessüberwachung
  - Eingebettete Systeme
  - Adaptive Systeme

# Gliederung der heutigen Wiederholungs-Vorlesung

❖ Einführung: Realität, Model

❖ Informatik-Systeme

❖ Klassen und Algebren

❖ Algorithmen und Textersetzungssysteme

❖ Boolesche Algebra

❖ Aussagenlogik

❖ Termersetzungssysteme

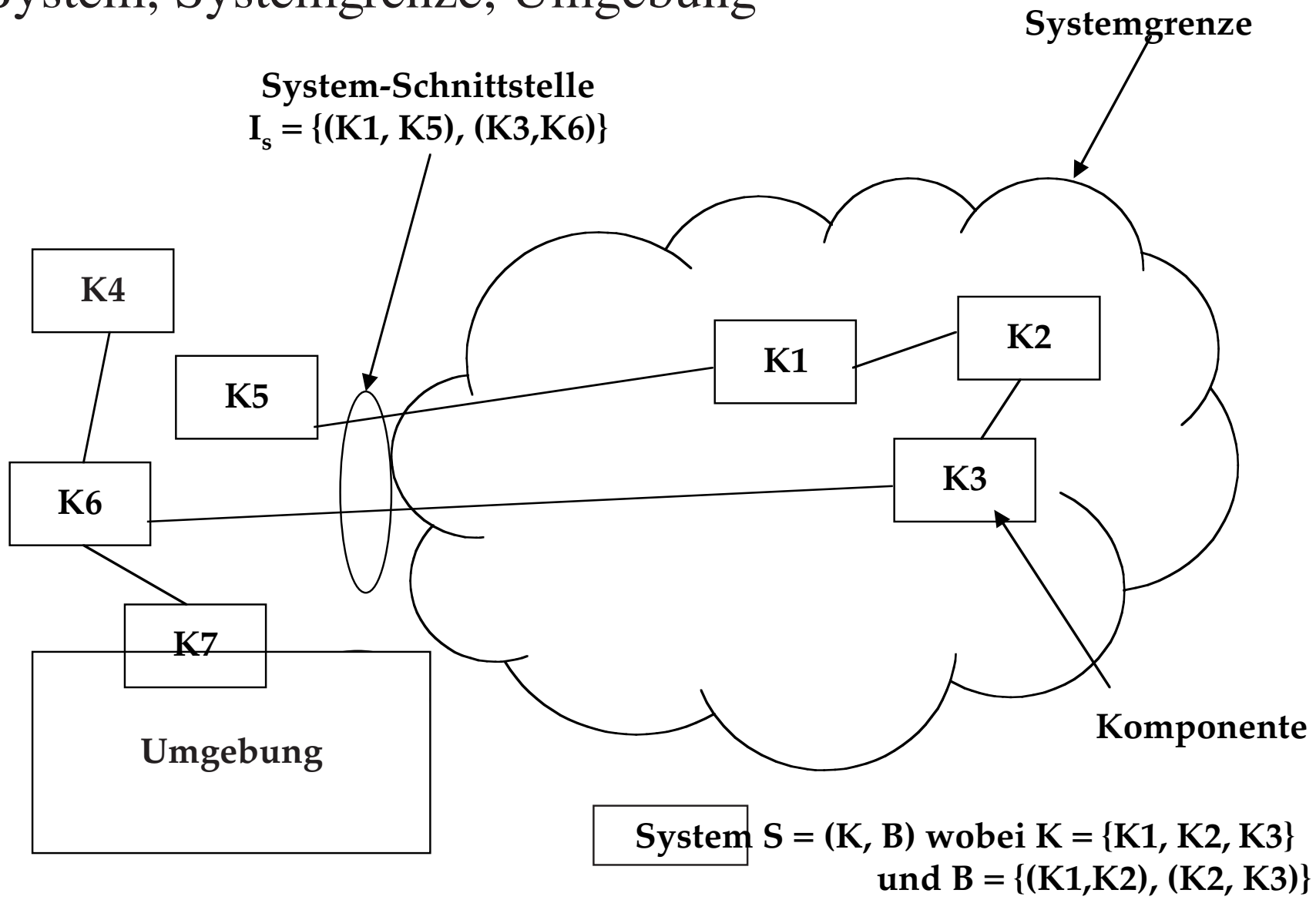
❖ Funktionale Programmierung

❖ Imperative Programmierung

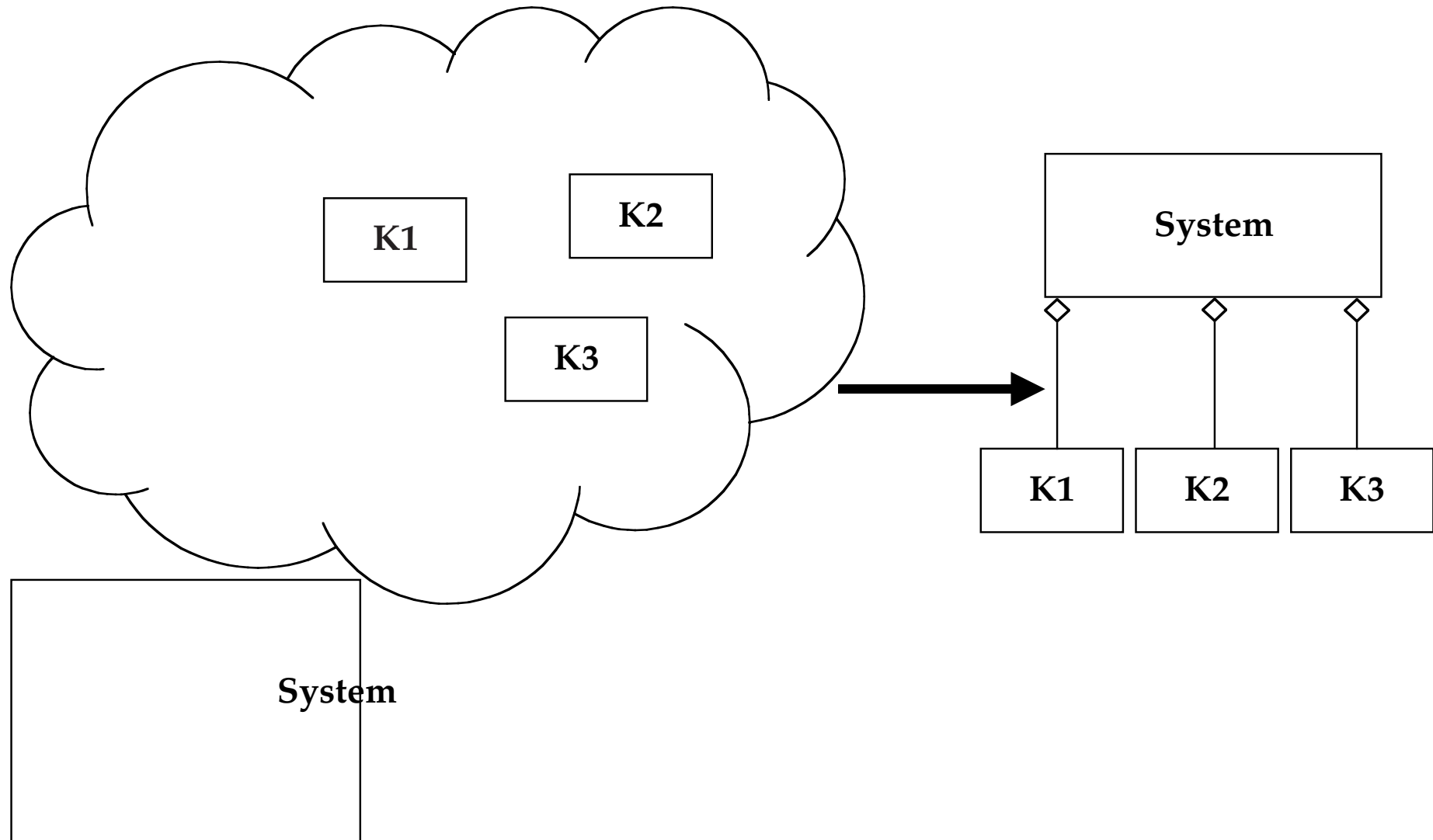
# Der Begriff des Systems

- ❖ Definition eines Systems: Unter einem System versteht man
  - eine Menge von Gegenständen, die in einem gegebenen Bezugssystem in einem Zusammenhang stehen,
  - und die Beziehungen zwischen diesen Gegenständen.
  - Die Gegenstände heißen Bausteine oder Komponenten.
- ❖ Jedes System hat eine Systemgrenze
  - Die Systemgrenze legt fest, welche Komponenten zu einem System gehören. Alle anderen Komponenten bezeichnet man als die Umgebung.
- ❖ Definition System-Schnittstelle: Die Beziehungen, die über die Systemgrenze laufen, d.h.
  - die Beziehungen zwischen Komponenten des Systems und Komponenten in der Umgebung.

# System, Systemgrenze, Umgebung

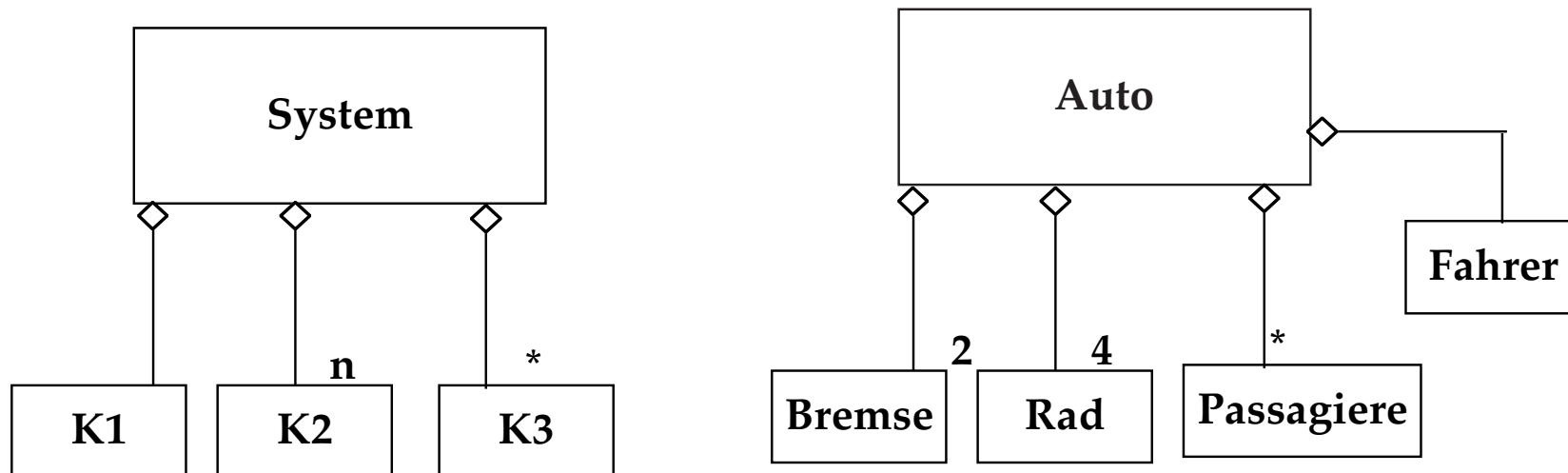


# Statt der Wolke benutzen wir einen Graphen

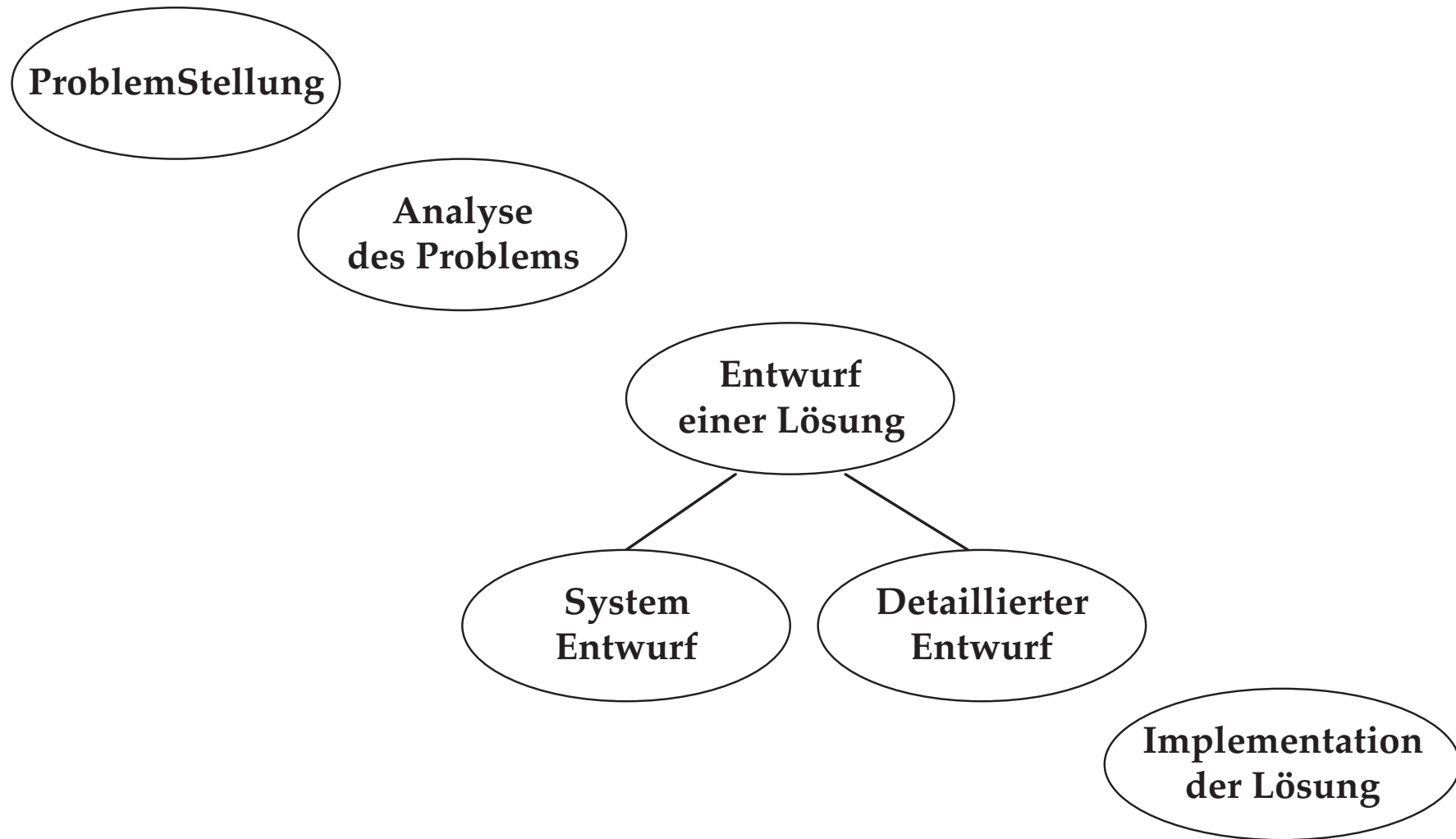


# Modellierung der Vielfachheit in Aggregationen

- ❖ In der Aggregationsbeziehung kann die Vielfachheit von Komponenten angegeben werden.
- ❖ Eine natürliche Zahl am Ende der Kante bezeichnet die Anzahl der Komponenten. 2 besondere Fälle:
  - Vielfachheit 1 wird nicht explizit angezeigt,
  - Vielfachheit “viele” wird durch ein Sternchen \* angezeigt.



# Aktivitäten bei der Entwicklung eines Informatik-Systems

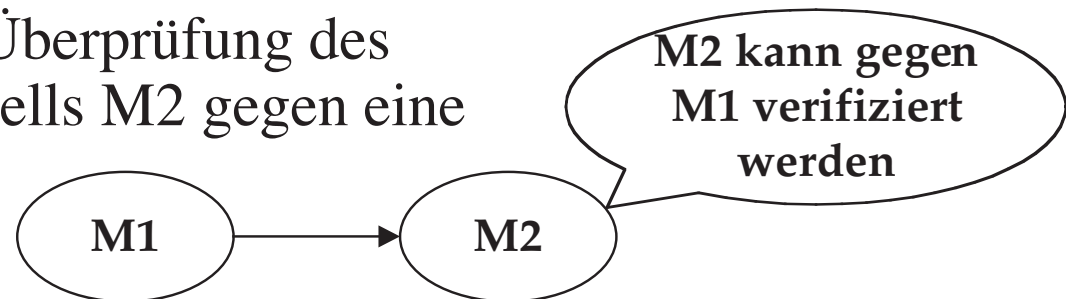


# Aktivitäten bei der Entwicklung eines Informatik-Systems

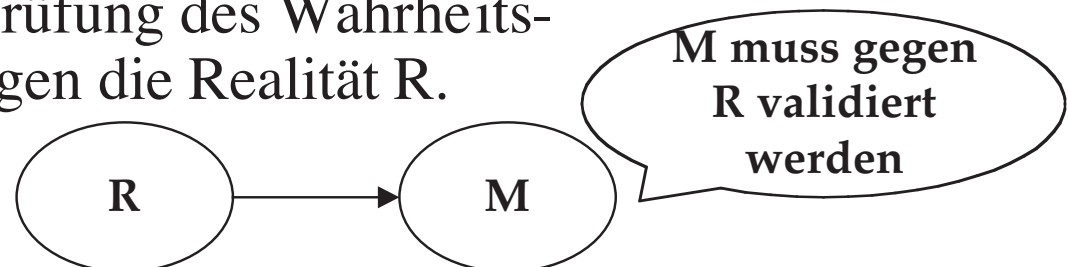
- ❖ Problemstellung:
  - Beschreibung des Problems und Ziele, Lösungsanforderungen
- ❖ Analyse:
  - Modellierung der Realität und Erstellung eines Modells
- ❖ Entwurf (Design):
  - Konstruktion einer Lösung, die die Machbarkeit des Modells demonstriert und den Anforderungen gerecht wird
    - System-Entwurf: Zerlegung der Lösung in Komponenten
    - Detaillierter Entwurf: Spezifikation der Schnittstellen
- ❖ Implementation:
  - Programmierung der Lösung mit einer Programmiersprache
- ❖ Test:
  - Überprüfung der Lösung: Sind die gewünschten Ziele erreicht?

# Wahrheitsgehalt von Modellen

- ❖ Definition Spezifikation: Eine Wirklichkeit, die unserer Gedankenwelt entstammt, oder ein Modell.
- ❖ Die Übereinstimmung einer Spezifikation mit einem Modell lässt sich mit mathematischen und logischen Schritten prüfen.
  - Definition Verifikation: Die Überprüfung des Wahrheitsgehaltes eines Modells M2 gegen eine Spezifikation M1.



- ❖ Wenn die Realität gegeben ist und nicht selbst der Welt des Denkens entstammt, dann können wir den Wahrheitsgehalt eines Modells nur durch Experimente feststellen:
  - Definition Validation: Überprüfung des Wahrheitsgehaltes eines Modells M gegen die Realität R.



# Gliederung der heutigen Wiederholungs-Vorlesung

- ❖ Einführung: Realität, Model
- ❖ Informatik-Systeme
- ❖ Klassen und Algebren
- ❖ Algorithmen und Textersetzungssysteme
- ❖ Boolesche Algebra
- ❖ Aussagenlogik
- ❖ Termersetzungssysteme
- ❖ Funktionale Programmierung
- ❖ Imperative Programmierung

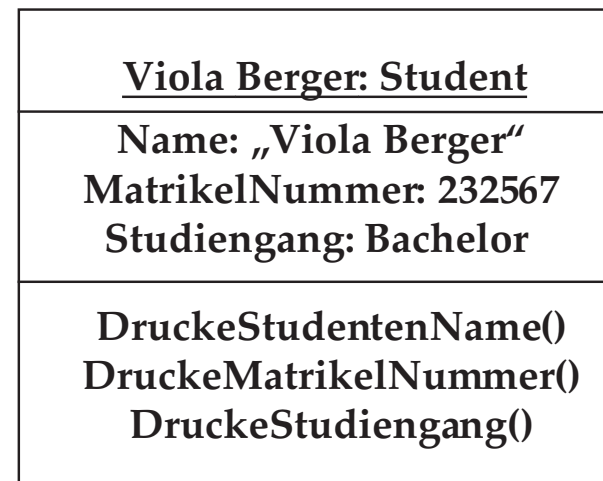
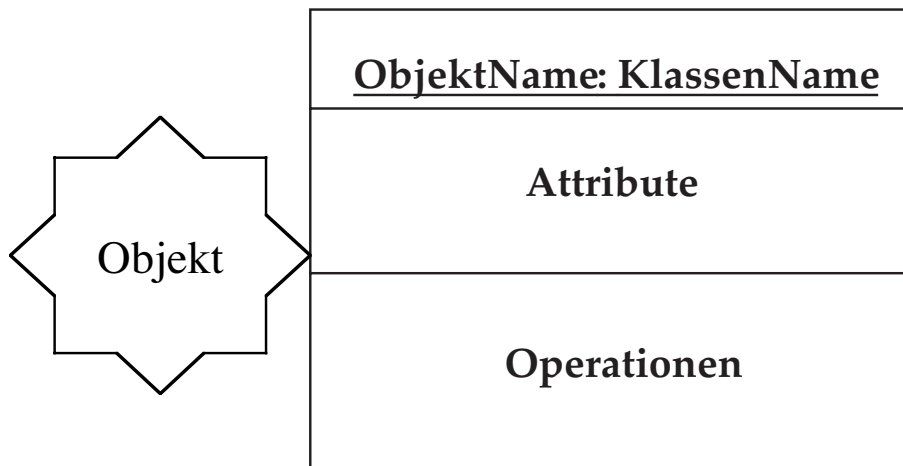
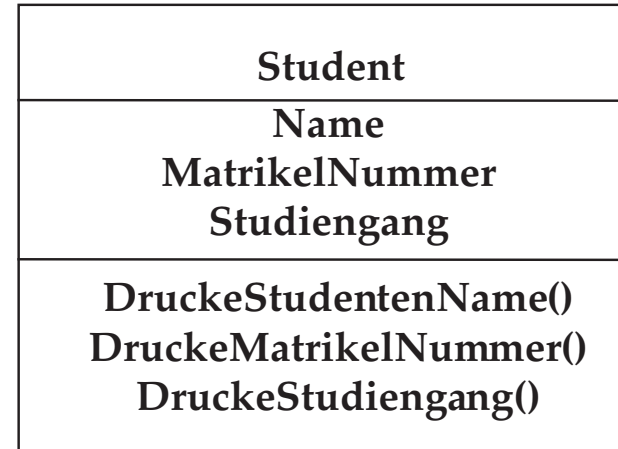
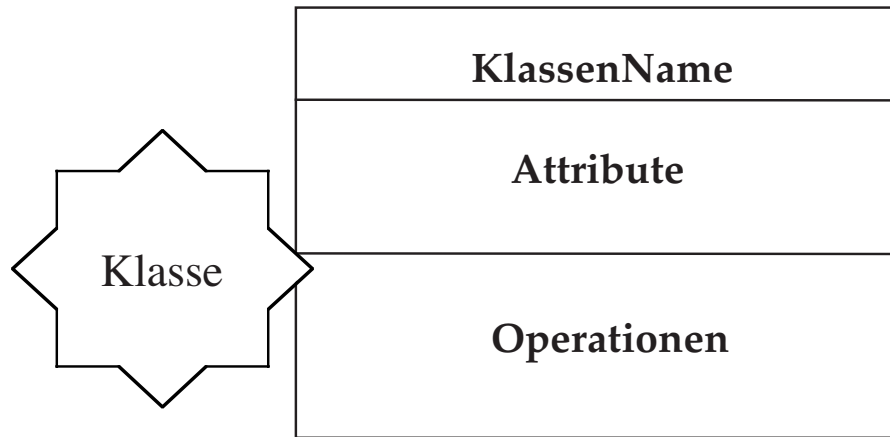
# Objekte, Attribute, Operationen, Merkmale

- ❖ Definition Objekt: Ein Objekt ist ein elementares Teilsystem. Es repräsentiert einen beliebigen Gegenstand in einem System.
- ❖ Ein Objekt besitzt Attribute und Operationen
  - Definition Attribut: Messbare, durch Werte erfassbare Eigenschaften des Objektes.
  - Definition Operation: Die Tätigkeiten, die ein Objekt ausführen kann, um Berechnungen durchzuführen, Ereignisse auszulösen, sowie Botschaften zu übermitteln.
- ❖ Definition Merkmal: Die zu einem Objekt gehörigen Attribute und Operationen.
- ❖ Definition Schnittstelle: Die Menge der Operationen eines Objektes, die von anderen Objekten aufgerufen werden können.

# Klasse

- ❖ Jedes Objekt gehört zu einer Menge von Objekten mit gleichen Merkmalen. Wir nennen diese Menge auch Klasse.
- ❖ Definition Klasse:
  - Die Menge aller Objekte mit gleichen Merkmalen, d.h. mit gleichen Attributen und Operationen.
- ❖ Definition Instanz:
  - Ein Objekt ist eine Instanz einer Klasse  $K$ , wenn es Element der Menge aller Objekte der Klasse  $K$  ist.

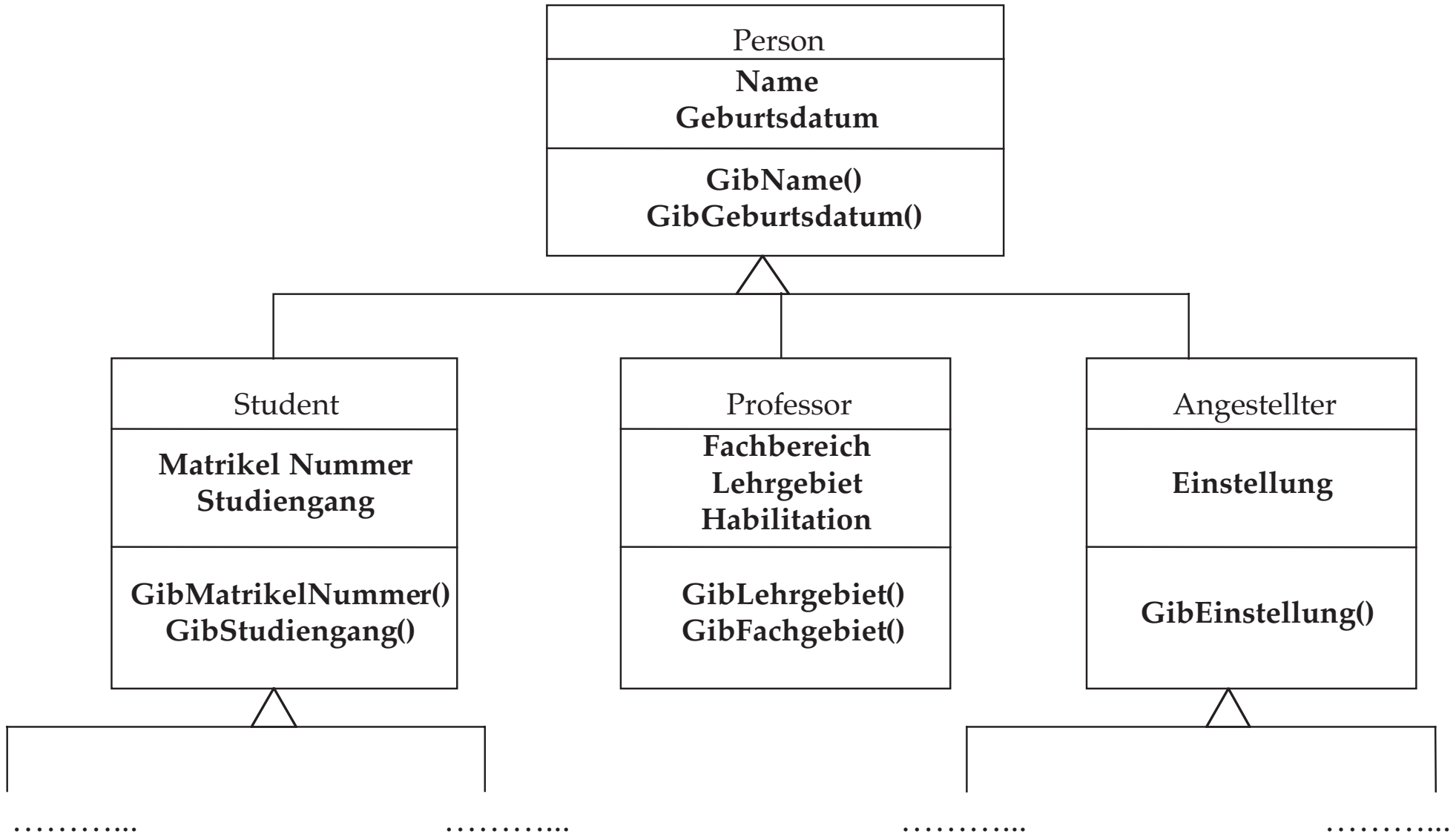
# Graphische Darstellung: Objekt vs. Klasse



## Zwei wichtige Prinzipien der Modellierung

- ❖ Informationskapselung (information hiding): Objekte können auf andere Objekte nur über deren Schnittstelle zugreifen.
  - Ein Objekt kann also nicht direkt auf die Attribute eines anderen Objektes zugreifen.
- ❖ Klassifikation: Komponenten können nach ihrer Schnittstelle klassifiziert werden.
  - Beispiel: Zwei Objekte Obj1 und Obj2 können zusammengefasst werden, wenn sie dieselbe Operation Print() verstehen.
- ❖ Klassifikationen kann man benutzen, um Mengen von Objekten hierarchisch zu strukturieren.
  - Beispiel: Die Personengruppen an einer Universität.

# Klassifikation von Personengruppen an einer Universität

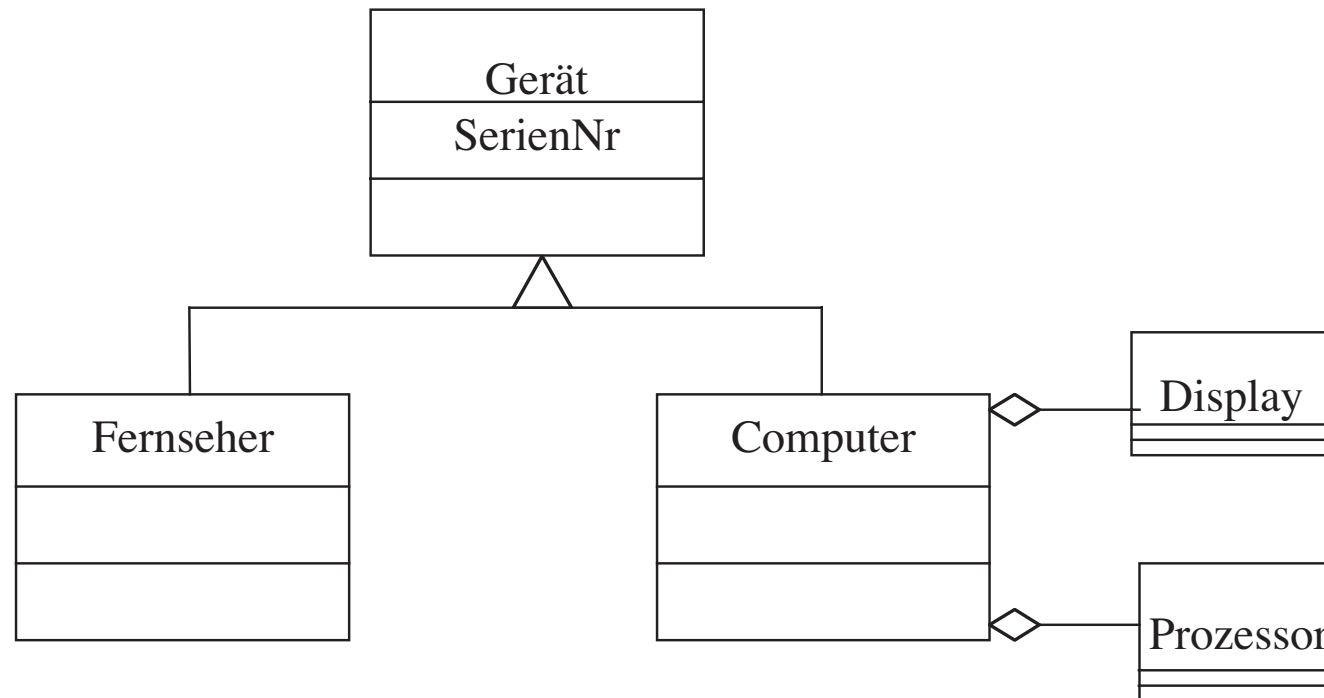


# Die Vererbungsbeziehung

- ❖ Zwei Klassen stehen in einer Vererbungsbeziehung (inheritance relationship) zueinander, falls die eine Klasse, auch Unterklasse (Subklasse) genannt, alle Merkmale der anderen Klasse, auch Oberklasse genannt, besitzt, und darüber hinaus noch zusätzliche Merkmale.

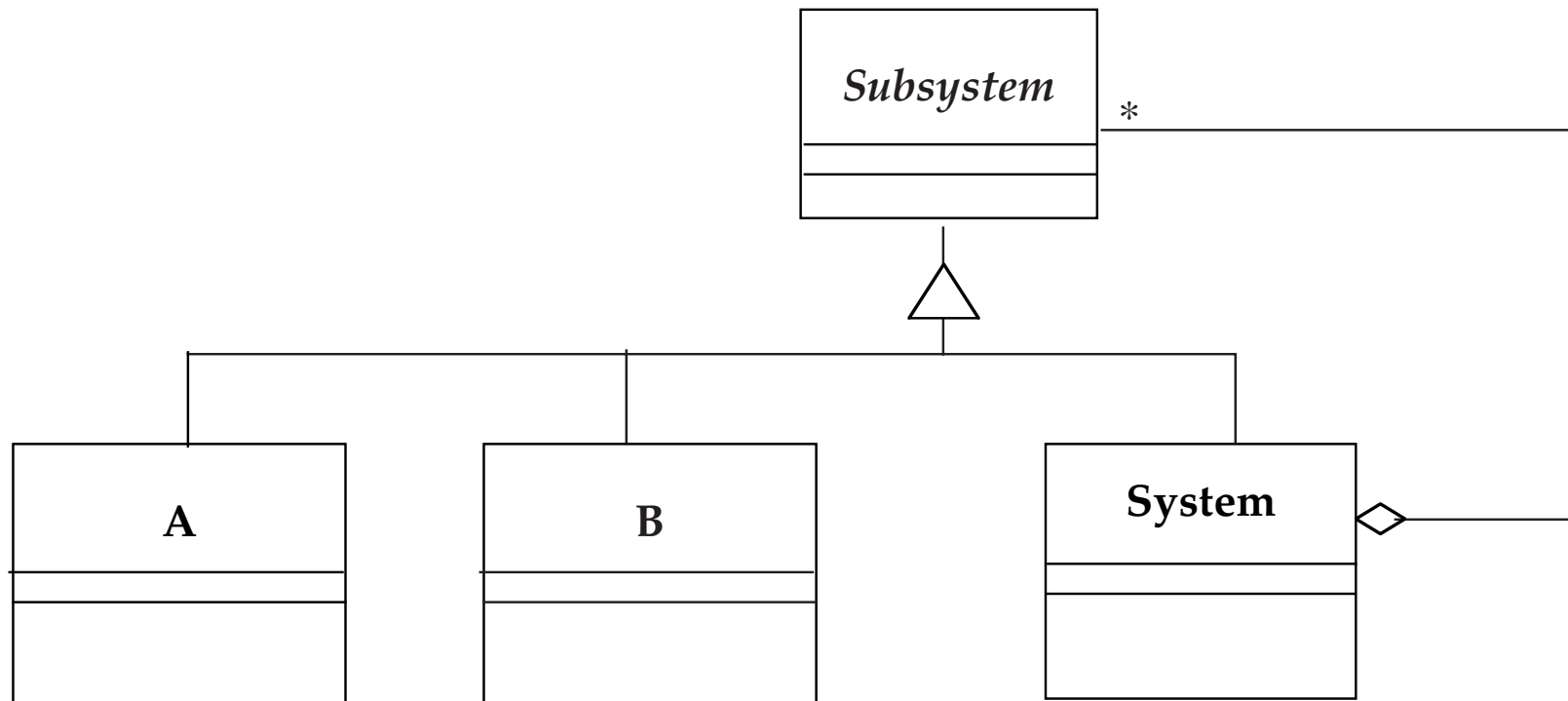
# Aggregation und Vererbung lassen sich kombinieren

- ❖ In der Modellierung tritt oft der Fall auf, dass wir Gegenstände klassifizieren müssen, aber gleichzeitig auch deren Struktur erkenntlich machen wollen.

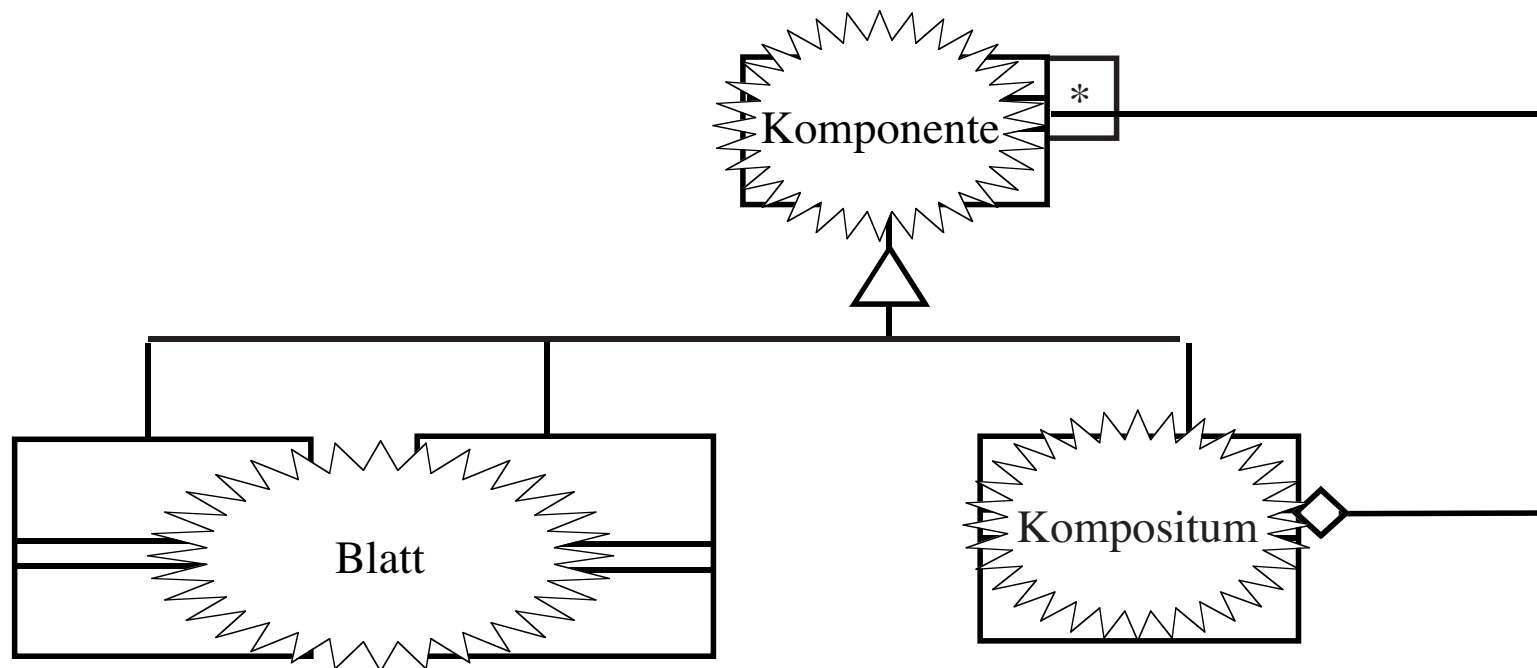


# Modellierung des Systembegriffs

- ❖ Beispiel: Ein System kann beliebig viele Subsysteme und 2 Arten von Komponenten A und B haben.



# Kompositionsmuster



Gamma et. al: Composite Pattern,  
Kompositum

# Formalisierung von Schnittstellen: Algebra und Rechenstruktur

- ❖ Wir wollen jetzt Schnittstellen formalisieren:
- ❖ Eine Algebra ist eine Familie von Mengen und eine Menge von Abbildungen zwischen diesen Mengen.
  - Bei Broy heißen diese Mengen auch Trägermengen, und Bezeichner für diese Trägermengen heißen Sorten.
  - Beispiel für Trägermengen: {Wahr, Falsch}, {0,1,2.....}
  - Beispiel für Sorten: bool, nat
- ❖ Bei den Informatikern heißt die Algebra auch Rechenstruktur.
  - Wir verwenden beide Begriffe.

## Definition Abstrakte Algebra

- ❖ Gegeben sei eine Signatur  $\Sigma$ , eine Menge von Elementaroperanden  $X$  und eine Menge von Gesetzen  $Q$  für die Anwendung von Operationen aus  $\Sigma$ .
- ❖ Dann heißt  $A = (\Sigma, Q)$  eine abstrakte Algebra.
- ❖ Wichtig
  - Durch die Definition der Signatur und der Menge der Elementaroperanden  $X$  ist die Menge  $T$  aller syntaktisch korrekten Terme für  $X$  und  $\Sigma$  definiert.
- ❖ Jede abstrakte Algebra  $A$  besitzt also eine Menge  $T$  von syntaktisch korrekten Termen.
  - Ein syntaktisch korrekter Term kann als eine Folge von Operationen auf anderen Termen verstanden werden.

## Definition Konkrete Algebra

- ❖ Gegeben sei eine abstrakte Algebra  $\mathbf{A} = (\Sigma, Q)$ .
- ❖ Definition:  $A = (M, \Sigma, Q)$  heißt eine konkrete Algebra, wenn
  - eine Trägermenge  $M$  mit  $X \subseteq M$  gegeben ist, und
  - es zu jeder Operation  $f \in \Sigma^{(n)}$  in der Signatur eine konkrete Funktion  $f_M : M^n \rightarrow M$  gibt, für welche die Gesetze von  $Q$  gelten.

# Gliederung der heutigen Wiederholungs-Vorlesung

- ❖ Einführung: Realität, Model
- ❖ Informatik-Systeme
- ❖ Klassen und Algebren
- ❖ Algorithmen und Textersetzungssysteme
- ❖ Boolesche Algebra
- ❖ Aussagenlogik
- ❖ Termersetzungssysteme
- ❖ Funktionale Programmierung
- ❖ Imperative Programmierung

# Algorithmus: Definition

## ❖ Definition Algorithmus:

– Ein Algorithmus ist ein Verfahren zur Verarbeitung von Daten mit einer präzisen, endlichen Beschreibung unter Verwendung effektiver Arbeitsschritte

❖ Präzise: In einer eindeutigen Sprache abgefasst

❖ Endlich: In einer endlichen Form beschrieben

❖ Effektiv: Die Arbeitsschritte sind tatsächlich ausführbar

❖ Ein Algorithmus muss nicht terminieren

❖ Definition Terminierender Algorithmus: Das Verfahren hat endlich viele Schritte

– Broy (S.31) verwendet diese Definitionen.

# Textersetzungssystem

❖ Definition: Eine Menge  $T = \{a \rightarrow b, c \rightarrow d, \dots\}$  von Regeln über einem Zeichenvorrat  $V$  heißt Textersetzungssystem (auch Semi-Thue System), wenn die folgenden Metaregeln gelten:

– Sei  $x$  ein Wort über  $V$  mit dem Teilwort  $a \circ \dots \circ b$ , d.h.

$$x = x_1 \circ x_2 \circ a \circ \dots \circ b \circ x_n$$

– Wenn  $a \circ \dots \circ b \rightarrow c \circ \dots \circ d$  eine Regel von  $T$  ist, dann kann man das Teilwort  $a \circ \dots \circ b$  in  $x$  durch  $c \circ \dots \circ d$  ersetzen:

$$x_1 \circ x_2 \circ a \circ \dots \circ b \circ x_n \Rightarrow x_1 \circ x_2 \circ c \circ \dots \circ d \circ x_n$$

– Wenn  $a \circ \dots \circ b$  mehrfach vorkommt oder mehrere Regeln anwendbar sind, so kann man das Teilwort bzw. die Regel beliebig wählen.

– Die Regeln können beliebig oft angewandt werden.

# Textersetzungssystem und Algorithmen

- ❖ Ein Textersetzungssystem zeigt die Grundform eines Algorithmus:
  - Die Eingabe ist ein Wort  $x$
  - Es gibt nur endlich viele Operationen  $o_1, o_2, \dots, o_n$ , nämlich die Regeln des Textersetzungssystems.
  - Der Algorithmus ist ausführbar, indem man Regeln auf die Eingabe  $x$  anwendet.

# Taxonomie von Algorithmen

Algorithmen unterscheiden sich nach

- ❖ der Anwendbarkeit von Regeln
  - Terminierender Algorithmus
  - Nichtterminierender Algorithmus
  - Deterministischer Algorithmus
  - Indeterministischer Algorithmus
- ❖ der Beziehung zwischen Ein- und Ausgabe
  - Determinierter Algorithmus
  - Indeterminierter Algorithmus

# Markov-Algorithmen

- ❖ Definition: Ein Markov-Algorithmus ist ein deterministisches Textersetzungssystem mit endlich vielen Regeln, sowie einigen speziellen Regeln, Zeichen und 2 Metaregeln:
  - Markov-Algorithmen enthalten spezielle Regeln, auch haltende Regeln  $x \rightarrow y$  genannt, die durch einen Punkt nach dem Pfeil  $\rightarrow$  gekennzeichnet sind.
  - Markov-Algorithmen enthalten zusätzliche Zeichen  $a, b, c, \dots$ , sogenannte Schiffchen.
- ❖ Markov-Algorithmen haben immer 2 spezielle Anweisungen ("Metaregeln") für die Ausführung von Regeln:
  - 1. Wähle in jedem Schritt die erste anwendbare Regel. Falls sie auf mehrere Teilwörter anwendbar ist, wende sie auf das am weitesten links stehende Teilwort an.
  - 2. Wende die Regeln solange an, bis eine haltende Regel angewandt wurde, oder bis keine Regel mehr anwendbar ist.

# Markov Algorithmus Beispiel

Zeichenvorrat  $V = \{0,L\}$

Schiffchen: a,b

Regelsystem:

(1)  $aL \rightarrow La$

(2)  $a0 \rightarrow 0a$

(3)  $a \rightarrow b$

(4)  $Lb \rightarrow b0$

(5)  $0b \rightarrow \cdot L$

(6)  $b \rightarrow \cdot L$

(7)  $\varepsilon \rightarrow a$

❖ Anwendung der Regeln auf das Eingabewort LOLL

L0LL  $\Rightarrow$  aL0LL (7)

$\Rightarrow$  La0LL (1)

$\Rightarrow$  L0aLL (2)

$\Rightarrow$  L0LaL (1)

$\Rightarrow$  L0LLa (1)

$\Rightarrow$  L0LLb (3)

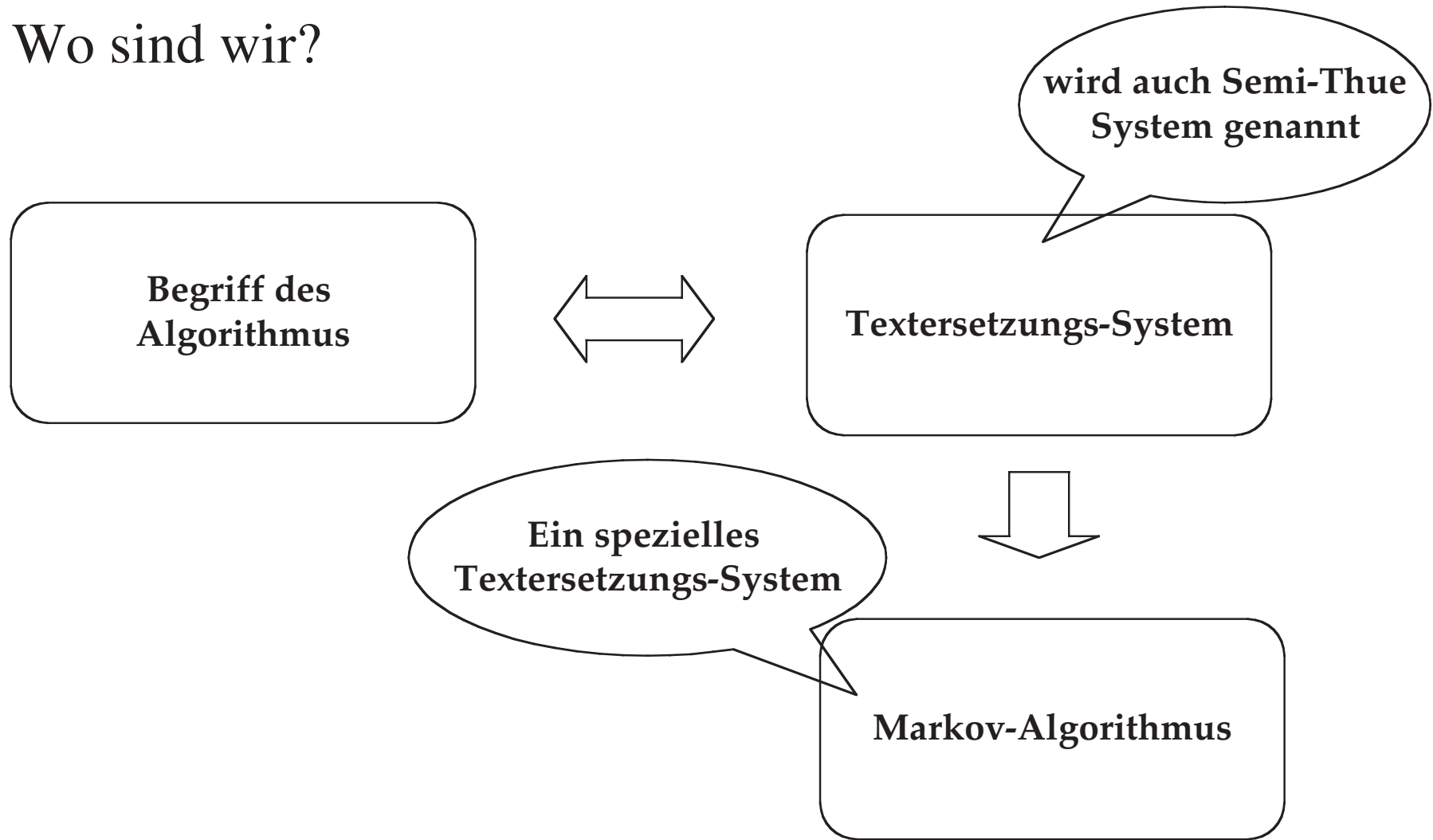
$\Rightarrow$  L0Lb0 (4)

$\Rightarrow$  L0b00 (4)

$\Rightarrow$  LL00 (5)

**Frage: Welches Problem löst der Algorithmus?**

# Wo sind wir?



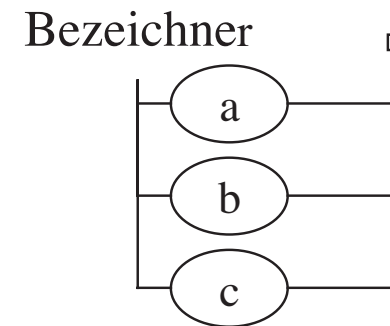
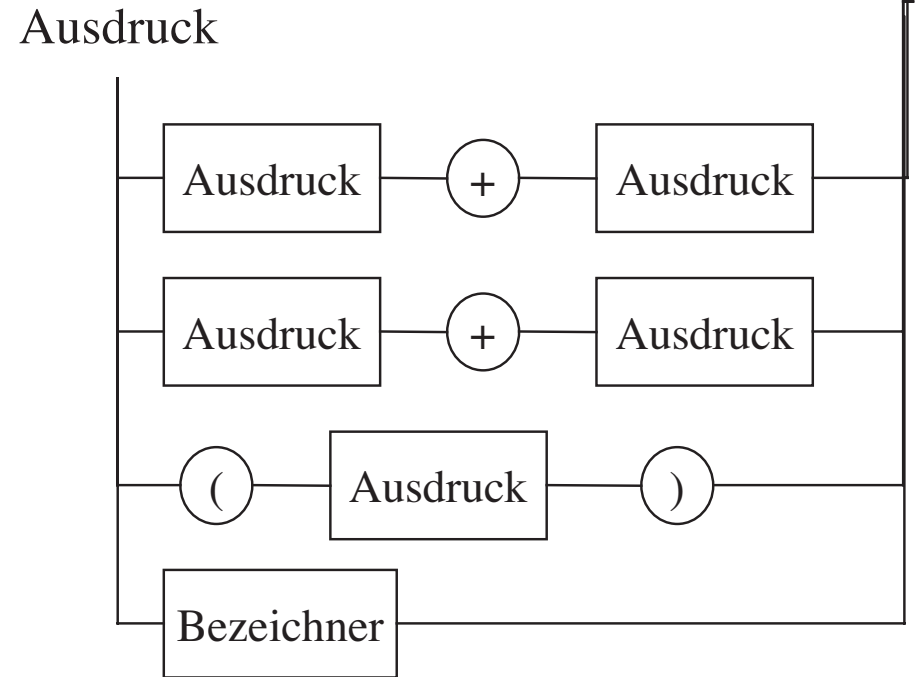
# Einfaches Beispiel einer Chomsky Grammatik

- ❖ Gegeben sei eine Chomsky-Grammatik  $G_A = (T, N, P, Z)$  mit
- ❖ Nichtterminale  $N = \{\text{Sentence, NounPhrase, VerbPhrase, Noun, Verb, Auxiliary}\}$
- ❖ Terminale  $T = \{\text{I, is, it, think, working}\}$
- ❖ Axiom  $Z = \text{Sentence}$
- ❖ Produktionen  $P = \{$ 
  - Sentence  $\rightarrow$  NounPhrase VerbPhrase Sentence
  - Sentence  $\rightarrow$  NounPhrase Auxiliary Verb
  - NounPhrase  $\rightarrow$  Noun
  - VerbPhrase  $\rightarrow$  Verb
  - Verb  $\rightarrow$  think
  - Verb  $\rightarrow$  working
  - Noun  $\rightarrow$  I
  - Noun  $\rightarrow$  it
  - Auxiliary  $\rightarrow$  is $\}$



# Beispiel: Konvertierung von Produktionen in ein Syntaxdiagramm

- ❖  $G_A = (T, N, P, Z)$
- ❖  $T = \{a, b, c, +, *\}$
- ❖  $N = \{\text{Ausdruck, Term, Faktor, Bezeichner}\}$
- ❖  $P = \{ \text{Ausdruck} \rightarrow \text{Term},$   
 $\text{Ausdruck} \rightarrow \text{Ausdruck} + \text{Term},$   
 $\text{Term} \rightarrow \text{Faktor},$   
 $\text{Term} \rightarrow \text{Term} * \text{Faktor},$   
 $\text{Faktor} \rightarrow \text{ID},$   
 $\text{Faktor} \rightarrow (\text{Ausdruck})$   
 $\text{Bezeichner} \rightarrow a \mid b \mid c \}$
- ❖  $Z = \text{Ausdruck}$



# Gliederung der heutigen Wiederholungs-Vorlesung

- ❖ Einführung: Realität, Model
- ❖ Informatik-Systeme
- ❖ Klassen und Algebren
- ❖ Algorithmen und Textersetzungssysteme
- ❖ Boolesche Algebra
- ❖ Aussagenlogik
- ❖ Termersetzungssysteme
- ❖ Funktionale Programmierung
- ❖ Imperative Programmierung

# Nachricht, Repräsentation, Information

- ❖ Signal: Die Darstellung einer Mitteilung durch die zeitliche Veränderung einer physikalischen Größe.
  - Beispiel: Akustische Welle, Lichtwelle, elektromagnetische Welle.
- ❖ Inschrift: Die dauerhafte Darstellung einer Mitteilung auf einem physikalischen Medium (Schriftträger).
  - Beispiel: Aufzeichnung auf einer CD, auf einem Stück Papier, auf einem Magnetband.
- ❖ Nachricht: Eine Mitteilung, bei der wir von dem Übertragungsmedium und der Darstellung durch Signale oder Inschriften abstrahieren (Eine Nachricht ist entweder ein Signal oder eine Inschrift).
  - Beispiel: Es interessiert uns bei der Mitteilung nicht, ob sie per Fax, E-Mail oder mit der normalen Post gekommen ist.
- ❖ Repräsentation: Die äußere Form einer Nachricht
- ❖ Information: Der abstrakte Gehalt einer Nachricht.  
Auch: Die Bedeutung (Semantik) einer Nachricht.

# Interpretation, Informationsbezugssystem

- ❖ Definition Interpretation: Der Vorgang der Ermittlung der Information aus einer Repräsentation. Geschieht normalerweise
  - Statisch durch ein Wörterbuch: Auflistung Nachricht, Information
  - oder prozedural durch eine Interpretationsvorschrift, die für jede Nachricht die jeweilige Information liefert.
- ❖ Definition Informationsbezugssystem: Die Menge der Gegenstände und Beziehungen zwischen Gegenständen, die nötig sind, um Information aus einer Repräsentation zu ermitteln.

# Interpretation von Booleschen Termen

Definition: Eine Abbildung  $\beta: ID \rightarrow IB$ , die jedem Element aus  $ID$  einen Wahrheitswert zuordnet, heißt Boolesche Belegung.

❖ Beispiel einer Belegung  $\beta$  für  $ID = \{a,b,c, x, y, z\}$ :

–  $\beta [a] = L$

–  $\beta [b] = O$

–  $\beta [c] = L$

–  $\beta [x] = L$

–  $\beta [y] = L$

–  $\beta [z] = L$

❖ Beispiel einer anderen Belegung  $\beta_1$  für  $ID = \{a,b,c, x, y, z\}$ :

–  $\beta_1 [a] = O$

–  $\beta_1 [b] = O$

–  $\beta_1 [c] = L$

–  $\beta_1 [x] = O$

–  $\beta_1 [y] = O$

–  $\beta_1 [z] = O$

- Die Menge aller Belegungen heißt ENV. Also:  $\beta, \beta_1 \in ENV$ .
- Für eine gegebene Belegung  $\beta$  läßt sich nicht nur den Elementen aus  $ID$ , sondern jedem Booleschen Term ein Wahrheitswert zu ordnen.
- Wir definieren dafür die Interpretation  $I_\beta$ .

## Interpretation $I_\beta$ von Booleschen Termen

- ❖  $I_\beta[\text{True}] = L$                        $I_\beta[\text{False}] = 0$
- ❖  $I_\beta[x] = \beta(x)$  d.h. für alle Elemente aus ID ist der Wahrheitswert der Wert, der sich aus der Belegung  $\beta$  ergibt.
  - Für unsere Beispiel Belegung  $\beta$ :  $I_\beta[a] = \beta[a] = L$
- ❖  $I_\beta[\neg t] = \neg (I_\beta[t])$ 
  - d.h. wir finden den Wahrheitswert von  $\neg t$ , indem wir den Wahrheitswert von Term  $t$  finden, und ihn dann negieren
- ❖  $I_\beta[t1 \vee t2] = I_\beta[t1] \vee I_\beta[t2]$ 
  - d.h. wir finden den Wahrheitswert von  $t1 \vee t2$ , indem wir die Wahrheitswerte von Term  $t1$  und  $t2$  finden, und sie dann mit Hilfe der Wahrheitstafel für  $\vee$  verknüpfen.
- ❖  $I_\beta[t1 \wedge t2] = I_\beta[t1] \wedge I_\beta[t2]$
- ❖  $I_\beta[t1 \Rightarrow t2] = I_\beta[t1] \Rightarrow I_\beta[t2]$
- ❖  $I_\beta[t1 \Leftrightarrow t2] = I_\beta[t1] \Leftrightarrow I_\beta[t2]$



Alles klar?

# Gliederung der heutigen Wiederholungs-Vorlesung

- ❖ Einführung: Realität, Model
- ❖ Informatik-Systeme
- ❖ Klassen und Algebren
- ❖ Algorithmen und Textersetzungssysteme
- ❖ Boolesche Algebra
- ❖ Aussagenlogik
- ❖ Termersetzungssysteme
- ❖ Funktionale Programmierung
- ❖ Imperative Programmierung

# Die Logik als Formales System

- ❖ Definition Logik: Eine Logik ist eine Menge von Regeln, definiert durch
  - eine Menge von Symbolen (Signatur)
  - eine Menge von Formeln, die aus diesen Symbolen konstruiert werden können (wohldefinierte Terme)
  - Eine Menge von ausgezeichneten Formeln (Axiome)
  - Eine Menge von Inferenzregeln
- ❖ Die Inferenzregeln sind die Metaregeln der Logik. Sie werden manchmal auch Ableitungsregeln genannt.

# Beweis und Theorem

## ❖ Definition Beweis:

- Ein Beweis ist eine endliche Zeichenkette von wohldefinierten Formeln (syntaktisch korrekten Termen), wobei jede Formel entweder ein Axiom ist oder durch Anwendung einer Inferenzregel abgeleitet worden ist.

## ❖ Definition Theorem:

- Ein Theorem einer Logik ist entweder ein Axiom oder eine Formel, die mit den Inferenzregeln aus den Axiomen und/oder aus bereits bewiesenen Theoremen abgeleitet wird.

# Korrektheit und Vollständigkeit einer Logik

- ❖ Der Zusammenhang zwischen der rein syntaktischen Definition einer Logik und ihrer Interpretation ist sehr wichtig.
  - Eine Interpretation ist ein Modell für eine Logik, wenn jedes Theorem unter dieser Interpretation wahr ist.
- ❖ Beziehungen zwischen Logik und möglichen Interpretationen:
  - Definition: Eine Logik ist korrekt (sound) wenn jedes Theorem allgemeingültig (valid) ist.
  - Definition: Eine Logik ist vollständig (complete) wenn jede allgemeingültige Formel auch ein Theorem ist.

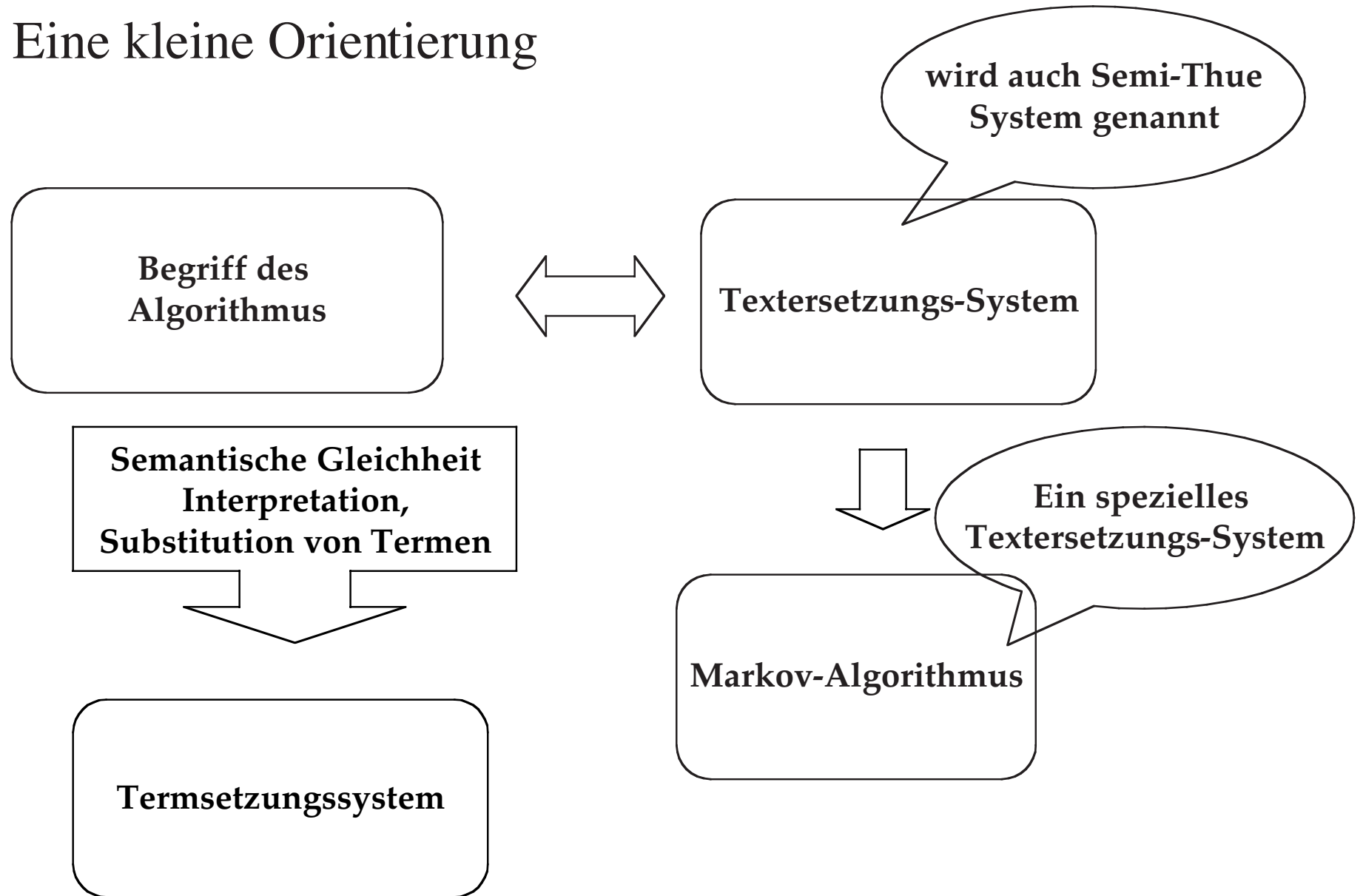
# Definition der Aussagenlogik

- ❖ Symbole: Der Zeichenvorrat  $T$  der Booleschen Algebra
- ❖ Formeln: Alle syntaktisch korrekten Terme der Booleschen Algebra
- ❖ Axiome: Formeln der Booleschen Algebra Inferenzregeln:
  - 1. Modus Ponens: Wenn  $x$  wahr ist, und wenn  $x \Rightarrow y$  gilt, dann ist  $y$  wahr.
  - 2. Tertium non datur: Entweder  $t$  oder  $\neg t$  ist wahr.
  - 3. Gleichheitsgesetz: Wenn zwei Ausdrücke  $X$  und  $Y$  semantisch gleich sind, dann kann man  $Y$  aus  $X$  sowie  $X$  aus  $Y$  ableiten.
  - Beispiel fürs Gleichheitsgesetz:
    - $y \wedge \neg y = \text{false}$
    - Damit kann ich ableiten  $y \wedge \neg y \vdash \text{false}$  sowie  $\text{false} \vdash y \wedge \neg y$

# Gliederung der heutigen Wiederholungs-Vorlesung

- ❖ Einführung: Realität, Model
- ❖ Informatik-Systeme
- ❖ Klassen und Algebren
- ❖ Algorithmen und Textersetzungssysteme
- ❖ Boolesche Algebra
- ❖ Aussagenlogik
- ❖ Termersetzungssysteme
- ❖ Funktionale Programmierung
- ❖ Imperative Programmierung

# Eine kleine Orientierung



# Substitution von Termen

- ❖ Definition Substitution in Termen mit freien Identifikatoren: Seien  $t_1$  und  $t_2$  Boolesche Terme mit freien Identifikatoren  $ID$  und sei  $x \in ID$  ein Identifikator. Dann bezeichnen wir mit

$t_1[t_2/x]$  den Term, den wir aus  $t_1$  erhalten, indem wir den Identifikator  $x$  an allen Stellen in  $t_1$  durch den Term  $t_2$  ersetzen.

Beispiel:

$x \wedge (\neg y) [False/x]$  bedeutet: Ersetze alle  $x$  durch  $False$ . Resultat:  
 $False \wedge (\neg y)$

## Instanz eines Terms

- ❖ Sei  $t$  ein Term mit freien Identifikatoren. Ein Term  $r$  heißt Instanz des Terms  $t$ , wenn  $r$  durch Substitution von Identifikatoren aus  $t$  entsteht.
  - Beispiel: Gegeben sei der Term  $t = \text{mult}(\text{add}(\text{succ}(x), y), z)$ , wobei  $x$ ,  $y$  und  $z$  freie Identifikatoren sind.

Wir benutzen die Substitution

- $t[\text{zero}/x, \text{succ}(\text{zero})/y, \text{succ}(\text{succ}(\text{zero}))/z]$

und erhalten die folgende Instanz von  $t$

- $\text{mult}(\text{add}(\text{succ}(\text{zero}), \text{succ}(\text{zero})), \text{succ}(\text{succ}(\text{zero})))$

## Definition: Interpretation

Für jede Belegung  $\beta$  ist die Interpretation  $I_\beta[t]$  eines Terms  $t$  mit freien Variablen (Identifikatoren) aus  $X$  für eine Rechenstruktur  $A$ :

- ❖ Für alle Elemente  $x \in X : I_\beta[x] = \beta(x)$
- ❖ Für alle  $f \in \Sigma$  mit den zugeordneten konkreten Funktionen  $f^k$ :
  - $I_\beta[f(t_1, \dots, t_n)] = f^k(I_\beta[t_1], \dots, I_\beta[t_n])$

## Interpretation eines Beispiel-Grundterms

$$I_\beta[\text{mult}(\text{add}(\text{succ}(\text{zero}), \text{succ}(\text{zero})), \text{succ}(\text{succ}(\text{zero})))] =$$

-- Anwendung von  $I_\beta[\text{mult}(x,y)] = I_\beta[x] * I_\beta[y]$

$$I_\beta[\text{add}(\text{succ}(\text{zero}), \text{succ}(\text{zero}))] * I_\beta[\text{succ}(\text{succ}(\text{zero}))] =$$

-- Anwendung von  $I_\beta[\text{add}(x,y)] = I_\beta[x] + I_\beta[y]$ , und  $I_\beta[\text{succ}(x)] = I_\beta[x] + 1$

$$(I_\beta[\text{succ}(\text{zero})] + I_\beta[\text{succ}(\text{zero})]) * (I_\beta[\text{succ}(\text{zero})] + 1) =$$

-- Mehrfache Anwendung von  $I_\beta[\text{succ}(\text{zero})] = I_\beta[\text{zero}] + 1$

$$(I_\beta[\text{zero}] + 1 + I_\beta[\text{zero}] + 1) * (I_\beta[\text{zero}] + 1 + 1) =$$

-- Mehrfache Anwendung von  $I_\beta[\text{zero}] = 0$

$$(0 + 1 + 0 + 1) * (0 + 1 + 1) =$$

-- Anwendung der konkreten Funktionen + und \* für die gegebenen Argumente

4

# Termersetzungsgesetze

- ❖ Definition Termersetzungsgesetz: Sei eine Familie  $X$  von Identifikatoren (Variablen) gegeben. Ein Paar  $(t, r)$  von Termen  $t, r$  gleicher Sorte mit freien Identifikatoren aus  $X$  heißt eine Termersetzungsgesetz.
  - Für die Gesetz schreiben wir wieder  $t \rightarrow r$
- ❖ Eine Termersetzungsgesetz bezeichnet in Wirklichkeit eine Menge von Gesetzen, weil sie freie Identifikatoren enthält:
  - Ersetzen wir einige oder alle Identifikatoren in der Gesetz, so erhalten wir eine Instanz der Gesetz.
- ❖ Definition Instanz einer Gesetz:
  - $t[t_1/x_1, \dots, t_n/x_n] \rightarrow r[t_1/x_1, \dots, t_n/x_n]$  ist eine Instanz von  $t \rightarrow r$
- ❖ Beispiel:
  - Die Gesetz  $\text{pred}(\text{succ}(x)) \rightarrow x$  mit Substitution  $[\text{zero}/x]$  ergibt die Instanz  $\text{pred}(\text{succ}(\text{zero})) \rightarrow \text{zero}$

# Termersetzungssystem

- ❖ Definition Termersetzungsschritt: Aus einer Regel wird ein Termersetzungsschritt gewonnen, indem wir die Instanz einer Regel auf einen beliebigen Teilterm eines vorliegenden Terms anwenden.
- ❖ Definition Termersetzungssystem: Eine Menge  $R$  von Termersetzungregeln über einer Signatur  $\Sigma$  heißt Termersetzungssystem über  $\Sigma$ .
- ❖ Definition Berechnung: Gilt für die Folge von Termen  $t_0, t_1, t_2, \dots, t_n$ 
  - $t_i \Rightarrow t_{i+1}$ , d.h.  $t_{i+1}$  entsteht aus  $t_i$  durch einen Termersetzungsschritt mit der Instanz einer Regel aus  $R$ ,
  - so heißt die Folge  $t_0, t_1, t_2, \dots, t_n$  Berechnung von  $R$  für den Term  $t_0$ .

# Ein Termersetzungssystem als Algorithmus

Termersetzungssystem  $R = \{$

$\text{pred}(\text{succ}(x)) \rightarrow x$  --1

$\text{add}(x, \text{succ}(y)) \rightarrow \text{succ}(\text{add}(x,y))$  --2

$\text{add}(x, \text{zero}) \rightarrow x$  --3

$\text{mult}(\text{succ}(x),y) \rightarrow \text{add}(\text{mult}(x,y), y)$  --4

$\text{mult}(x, \text{succ}(y)) \rightarrow \text{add}(\text{mult}(x,y), x)$  --5

$\text{mult}(\text{zero},\text{zero}) \rightarrow \text{zero}$  --6

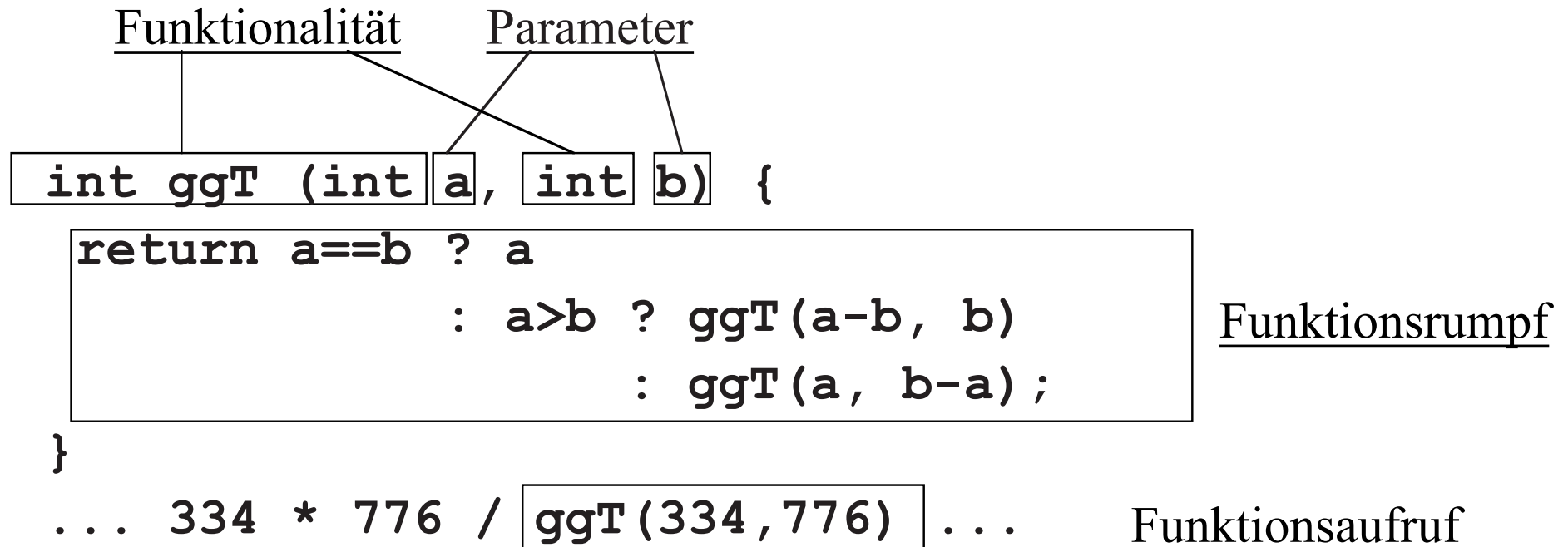
$\}$

- ❖ Regel 1 eliminiert pred
- ❖ Regeln 2 und 3 definieren einen Algorithmus für die Addition
- ❖ Regeln 4, 5 und 6 definieren einen Algorithmus für die Multiplikation

# Gliederung der heutigen Wiederholungs-Vorlesung

- ❖ Einführung: Realität, Model
- ❖ Informatik-Systeme
- ❖ Klassen und Algebren
- ❖ Algorithmen und Textersetzungssysteme
- ❖ Boolesche Algebra
- ❖ Aussagenlogik
- ❖ Termersetzungssysteme
- ❖ Funktionale Programmierung
- ❖ Imperative Programmierung

# Funktionale Sprachkonzepte in Java



# Rekursive Funktionen bzw. Funktionsdeklarationen

## ❖ Beispiel: ggT

```
int ggT (int a, int b) {  
    return a==b ? a : a>b ? ggT(a-b, b)  
                        : ggT(a, b-a);  
}
```

- ❖ Im Rumpf der Funktion ggT treten Aufrufe von ggT auf (Selbstaufufe).
- ❖ Definition: Eine Funktion bzw. eine Funktionsdeklaration heißt rekursiv, falls der Ausdruck im Rumpf der Funktion einen Aufruf derselben Funktion enthält.

# Rekursionsarten: Lineare Rekursion

## ❖ Definition Lineare Rekursion:

– Eine rekursive Funktion bzw. Funktionsdeklaration heißt linear rekursiv, wenn in jedem Zweig des bedingten Ausdrucks höchstens ein Selbstaufruf der Funktion auftritt.

❖ Bemerkung: Die näher betrachteten rekursiven Funktionen ggT, summe, fakultaet und mult sind linear rekursiv.

❖ Eine Funktion ist genau dann linear rekursiv, wenn ihre Aufrufstruktur linear ist:

ggT (9, 12)  
|  
ggT (9, 3)  
|  
ggT (6, 3)  
|  
ggT (3, 3)

summe (3)  
|  
summe (2)  
|  
summe (1)  
|  
summe (0)

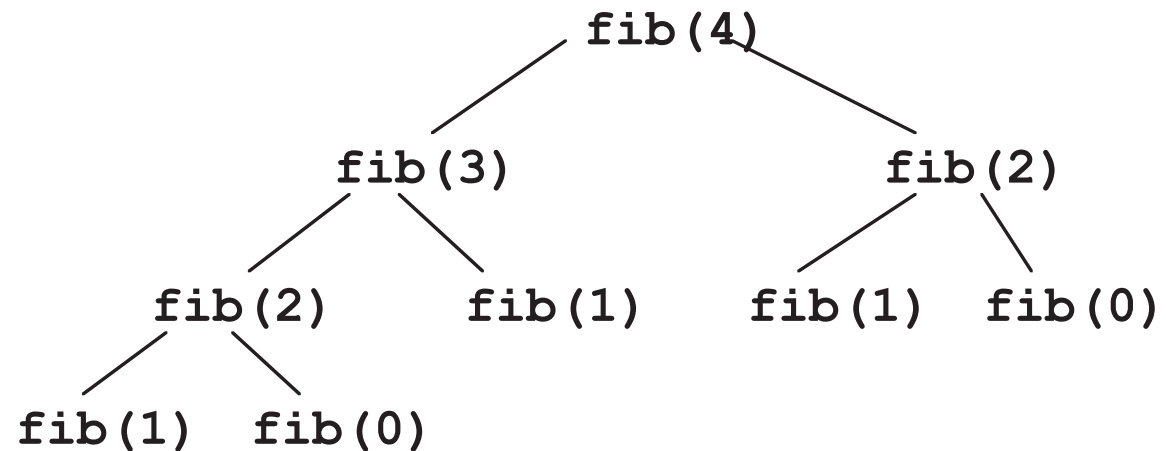
fakultaet (3)  
|  
fakultaet (2)  
|  
fakultaet (1)  
|  
fakultaet (0)

mult (5, -2)  
|  
mult (5, 2)  
|  
mult (5, 1)  
|  
mult (5, 0)

# Kaskadenrekursion: Fibonacci-Zahlen

```
int fib (int n) {  
    return n==0 ? 0  
           : n==1 ? 1  
           : fib(n-1) + fib(n-2);  
}
```

❖ Aufrufstruktur:

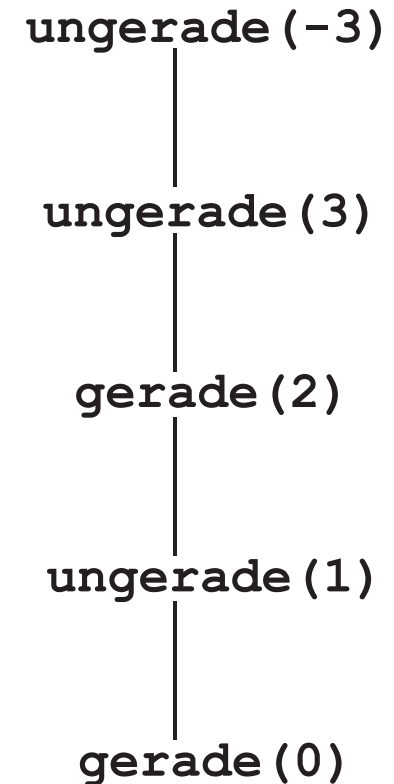


❖ fib ist kaskadenartig rekursiv.

# Verschränkte Rekursion: die Funktionen gerade bzw. ungerade

```
boolean gerade (int x) {  
    return x<0 ? gerade(-x)  
           : x==0 ? true  
           : ungerade(x-1) ;  
}
```

```
boolean ungerade (int x) {  
    return x<0 ? ungerade(-x)  
           : x==0 ? false  
           : gerade(x-1) ;  
}
```



## Nachweis der Terminierung (allgemeine Fassung)

- ❖ Gegeben sei folgende rekursive Funktionsdeklaration:

$T f (T_1 x_1, T_2 x_2, \dots, T_n x_n) \{ \text{return } A; \}$

wobei der Ausdruck  $A$  rekursive Funktionsaufrufe der Form  $f(A_1, A_2, \dots, A_n)$  enthalte.

- ❖ Seien  $T'_1, T'_2, \dots, T'_n$  Teilmengen von  $T_1, T_2, \dots, T_n$  derart,
  - dass sich die Ausdrücke  $A_1, A_2, \dots, A_n$  eines jeden rekursiven Aufrufes zu Werten  $s_1, s_2, \dots, s_n$  aus  $T'_1, T'_2, \dots, T'_n$  berechnen, falls die Parameter  $t_1, t_2, \dots, t_n$  des ursprünglichen Aufrufes ebenfalls Werte dieser Teilmengen sind

und gelte mit einer Funktion  $h: T'_1 \times T'_2 \times \dots \times T'_n \rightarrow \mathbb{N}$ , die jeder Kombination von Parameterwerten aus dem eingeschränkten Bereich  $T'_1 \times T'_2 \times \dots \times T'_n$  eine natürliche Zahl zuordnet,

–  $h(s_1, s_2, \dots, s_n) < h(t_1, t_2, \dots, t_n)$ ,

- ❖ Dann terminiert  $f$  für Parameterwerte aus  $T'_1, T'_2, \dots, T'_n$

# Varianten zur vollständigen Induktion

- ❖ Die Standard-Variante der vollständigen Induktion:
  - gilt  $A(n_0)$
  - und folgt aus  $A(n)$  auch  $A(n+1)$
  - dann gilt  $A(n)$  für alle  $n \in \mathbb{N}$  mit  $n \geq n_0$
- ❖ Eine Variante mit zwei Spezialfällen als Induktionsanfang:
  - gelten  $A(n_0)$  und  $A(n_0+1)$
  - und folgt aus  $A(n)$  und  $A(n+1)$  auch die Gültigkeit von  $A(n+2)$
  - dann gilt  $A(n)$  für alle  $n \in \mathbb{N}$  mit  $n \geq n_0$
- ❖ Die Varianten mit drei oder mehr Spezialfällen als Induktionsanfang sind analog.
- ❖ Eine andere Variante des Induktionsschlusses:
  - gilt  $A(n_0)$
  - und folgt aus  $A(n_0), A(n_0+1), A(n_0+2), \dots, A(n)$  auch  $A(n+1)$
  - dann gilt  $A(n)$  für alle  $n \in \mathbb{N}$  mit  $n \geq n_0$

# Induktion und Rekursion

- ❖ Induktion und Rekursion sind zwei Seiten derselben Medaille:
  - Induktion startet im Spezialfall.  
Die Annahme kann für alle natürlichen Zahlen ab dem Startwert gezeigt werden, da der Induktionsschritt „nur“ endlich oft angewendet werden muss.
  - Die Rekursion führt ein komplexes Problem durch endlichmalige Anwendung des Rekursionsschrittes auf einen Spezialfall zurück.

## Korrektheit rekursiver Funktionen

- ❖ Eine (rekursive) Funktion  $f$  heißt für Parameterwerte  $t_1, t_2, \dots, t_n$  korrekt,
  - falls sie für die Parameterwerte  $t_1, t_2, \dots, t_n$  partiell korrekt ist
    - (partielle Korrektheit bedeutet, dass das berechnete Ergebnis der Spezifikation entspricht, falls die Funktion terminiert; Terminierung wird aber nicht vorausgesetzt)
  - und falls sie für die Parameterwerte  $t_1, t_2, \dots, t_n$  terminiert.
- ❖ Zum Nachweis der Terminierung eignet sich die Methode mit der Abstiegsfunktion  $h$ .
- ❖ Zum Nachweis der partiellen Korrektheit eignet sich Induktion.

# Operationale vs. funktionale Semantik

- ❖ Zwei mögliche Sichtweisen bei der Angabe einer Semantik
    - operationale Semantik:  
Beschreibung der Abfolge der einzelnen Berechnungsschritte bei der Ausführung des Programms
    - funktionale Semantik:  
Beschreibung der Funktion eines Programmes durch Festlegung des Ein- /Ausgabeverhaltens (extensionales bzw. beobachtbares Verhalten)
  - ❖ Speziell für funktionale Programmiersprachen wählen wir:
    - für die operationale Semantik:
      - Ein Programm ist ein Termersetzungssystem.
    - für die funktionale Semantik:
      - Eine Interpretationsfunktion  $I$  ordnet
        - einer Funktionsdeklaration eine Funktion und
        - einem Ausdruck einen Wert
- ZU.

# Gliederung der heutigen Wiederholungs-Vorlesung

- ❖ Einführung: Realität, Model
- ❖ Informatik-Systeme
- ❖ Klassen und Algebren
- ❖ Algorithmen und Textersetzungssysteme
- ❖ Boolesche Algebra
- ❖ Aussagenlogik
- ❖ Termersetzungssysteme
- ❖ Funktionale Programmierung
- ❖ Imperative Programmierung

# Klassenspezifikation *Student*

- ❖ Klassenname: Student
  - Aufgabe: Repräsentation typischer Aktivitäten von Studenten
- ❖ Variablen:
  - studiert: Wird auf true gesetzt, wenn Student studiert (private)
  - schlaeft: Wird auf true gesetzt, wenn Student schläft (private)
- ❖ Methoden:
  - Student(): Eine Methode um ein Objekt vom Typ Student zu initialisieren. (Kriegen wir umsonst in Java: Konstruktormethode)
  - schlafe(): Eine Methode um ein Objekt vom Typ Student in den Schlafzustand zu bringen.
  - studiere(): Eine Methode um ein Objekt vom Typ Student in den Schlafzustand zu bringen.

# Java Implementation der Klasse Student

```
public class Student
{
    // Data
    private boolean studiert = true; // Student's state
    private boolean schlaeft = false;

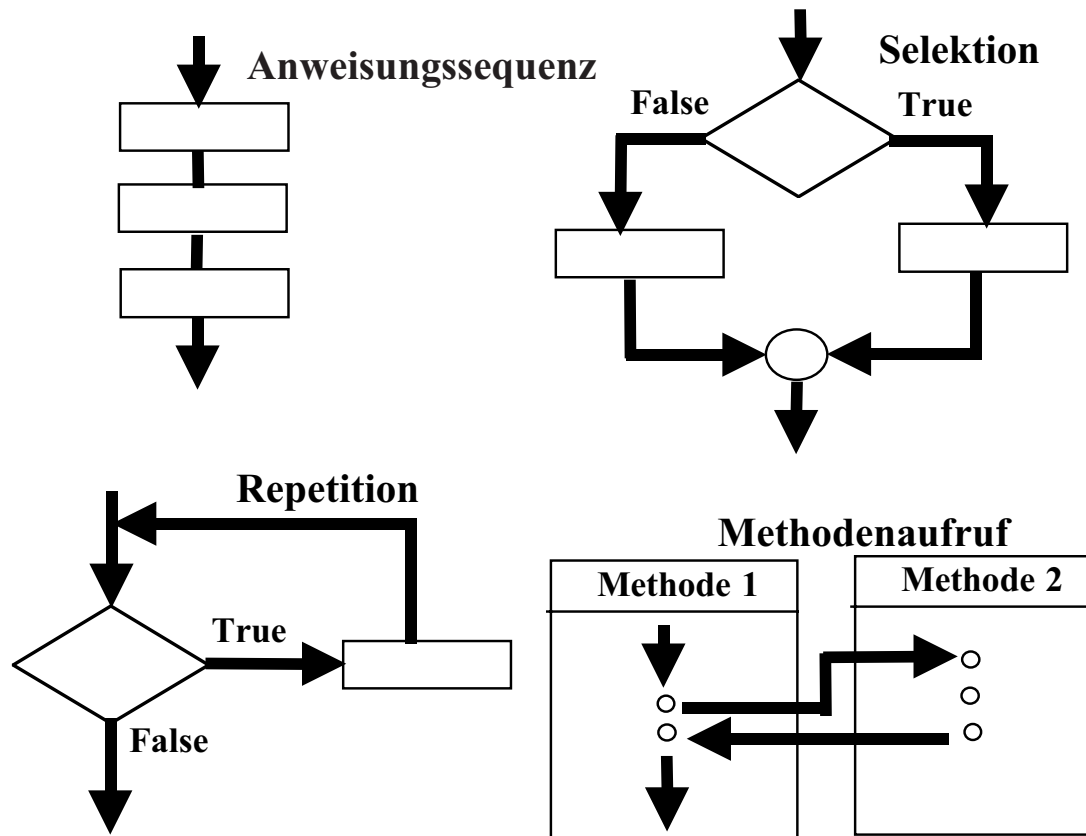
    // Methoden
    public void studiere() // Start von studiere
    {
        studiert = true; // Ändere den Zustand
        schlaeft = false;
        System.out.println("Student studiert");
        return;
    } // studiere()

    public void schlafe() // Start schlafe
    {
        schlaeft = true; // Ändere den Zustand
        studiert = false;
        System.out.println("Student schläft");
        return;
    } // schlafe()
} // Student Klasse
```

# Strukturierte Programmierung

- ❖ Definition Strukturierte Programmierung: Imperative Programme die nur mit bestimmten Kontrollstrukturen (Typen von Anweisungen) geschrieben werden. Diese sind:
  - Anweisungssequenz --- Eine Folge von Anweisungen, die eine nach der anderen sequentiell exekutiert wird.
  - Selektion---Eine Anweisung, die eine Wahl zwischen zwei oder mehr Alternativen von Anweisungssequenzen erlaubt (Bedingte Anweisung (if, if-else), Fallunterscheidung (switch)).
  - Repetition --- Eine Anweisung, die es erlaubt eine Anweisungssequenz zu wiederholen (for, while, und do-while Struktur).
  - Methodenaufruf --- Eine Anweisung, die die Kontrolle im Programm zur benannten Methode transferiert. Wenn diese Methode ausgeführt worden ist, geht die Kontrolle an die Stelle nach dem Methodenaufruf zurück.

# Die 4 Konstrukte der Strukturierten Programmierung



❖ Egal wie gross oder klein ein Programm ist, es kann nur durch eine beliebige Kombination dieser 4 Konstrukte beschrieben werden.

❖ Jede dieser Konstrukte hat genau einen Eingang (Entry) und genau einen Ausgang (exit)

# Anweisungstypen in Java

- ❖ In Java gibt es verschiedenen Typen von Anweisungen:
  - Deklarationsanweisung (declaration statement)
  - Zuweisung (assignment)
  - Bedingte Anweisung (conditional statement)
  - Returnanweisung (return statement)
  - Schleifenanweisungen (iteration statements)