Mathematik 1 für Informatiker

Grundbegriffe

Einführung in die
Logik u. Algebra

31.11.2006

M. B. Wischnewsky

Beispiele für binäre Relationen

1) $f : A \longrightarrow B \quad a \mapsto f(a)$

2) Äquivalenzrelation

Vorbemerkung

Partition (Zerlegung) von A in disjunkte Teilmengen.

1) $C_i \subseteq A$; $C_i \neq \emptyset$ ($\emptyset$ leere Menge)

2) $\bigcup_{i=1}^{n} C_i = A$

3) $C_i \cap C_j \neq \emptyset \Rightarrow C_i = C_j$

Äquivalenzrelationen

Behpt. Die Menge der Äquivalenzklassen von A ist eine Partition auf A.

$$\mathcal{P} = \{[x];\ x \in A\}$$

Beweis

1) $\forall_{x \in A}\ [x] \neq \emptyset$, da $x \sim x$

$$[x] := \{y \in A;\ y \sim x\}$$

2) $\bigcup_{x \in A} [x] = A$, da

(Bem. $A = B \Leftrightarrow A \subseteq B$ und $B \subseteq A$)

(i) $[x] \subseteq A \Rightarrow \bigcup_{x \in A} [x] \subseteq A$

(ii) $x_0 \in A \Rightarrow x_0 \in [x_0] \Rightarrow x_0 \in \bigcup_{x \in A} [x]$

$A \subseteq \bigcup_{x \in A} [x]$

3) $[x] \cap [y] \neq \emptyset \overset{?}{\Rightarrow} [x] = [y]$

$z \in [x] \cap [y] \Rightarrow x \sim z$ und $z \sim y$

$\overset{\text{transitiv}}{\Rightarrow} \boxed{x \sim y} \Rightarrow x \in [y]$ und $y \in [x]$

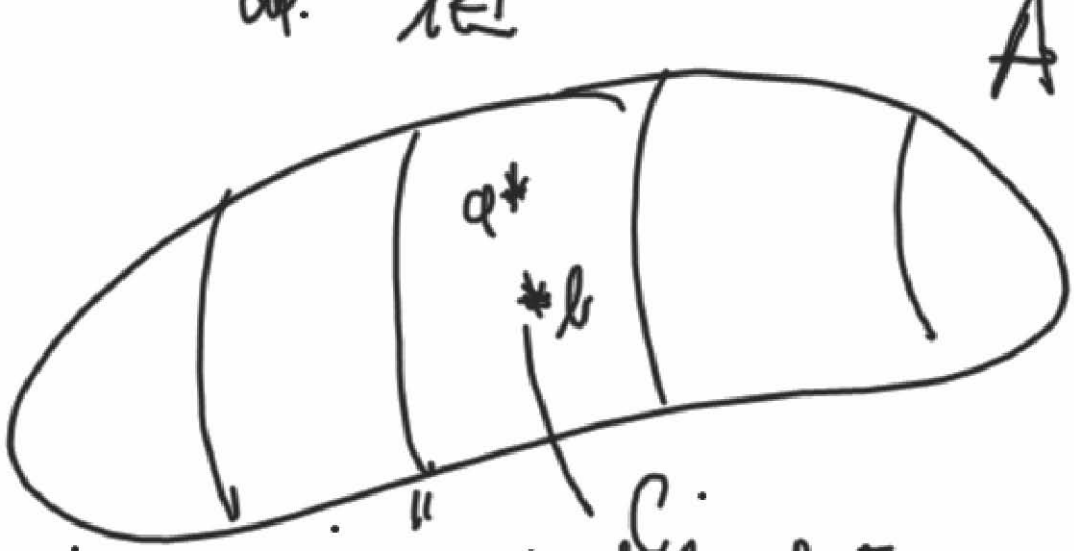
$\Rightarrow [x] = [y]$

$[x] \subseteq [y]$ u. $[y] \subseteq [x]$

Satz geg Partition $\mathcal{P} = \{C_i ; i \in I\}$ auf A

sei für $a, b \in A$ folg. Relation auf A definiert

$$a \sim b \overset{\text{def.}}{\Leftrightarrow} \exists_{i \in I} \; a, b \in C_i$$



Dann ist $\sim$ eine Äquivalenzrelation auf A.

Mengenbegriff
Wiederholung

„Definition“ des Mengenbegriffs nach G. Cantor (1895)

- Unter einer „Menge“ verstehen wir jede Zusammenfassung *M* von bestimmten, wohlunterschiedenen Objekten *m* unserer Anschaung oder unseres Denkens (welche die „Elemente“ von *M* genannt werden) zu einem Ganzen

Zwei Formen der Darstellung von Mengen gebräuchlich

- **Aufzählung aller Elemente der Menge (falls abzählbar)**
  - **{e1, e2, e3, ...}**
- **Mittels eines definierenden Ausdrucks**
  - **{*x* | A(*x*)}**

Vergleiche von Mengen

- Teilmenge $B \subseteq A$
  - $x \in B \Rightarrow x \in A$
  - z.B. $B = \{ x / x \in A \wedge H(x) \}$
- Gleichheit von Mengen $A = B$
  - $A = B \Leftrightarrow A \subseteq B \wedge B \subseteq A$
- Potenzmenge $\mathbb{P}(A)$ von A
  - $\mathbb{P}(A) = \{ B / B \subseteq A \}$
  - $\mathbb{P}(\emptyset) = \{\emptyset\} \neq \emptyset$

Elementare Mengenoperationen

- Die Mengenlehre kennt drei grundlegende Operationen, mit denen man zwei Mengen verknüpfen kann:

Vereinigung	$A \cup B$	$=_{def} \{ e \mid e \in A \text{ oder } e \in B \}$
Durchschnitt	$A \cap B$	$=_{def} \{ e \mid e \in A \text{ und } e \in B \}$
Differenz	$A \setminus B$	$=_{def} \{ e \mid e \in A \text{ und } e \notin B \}$

- Beispiele dazu:

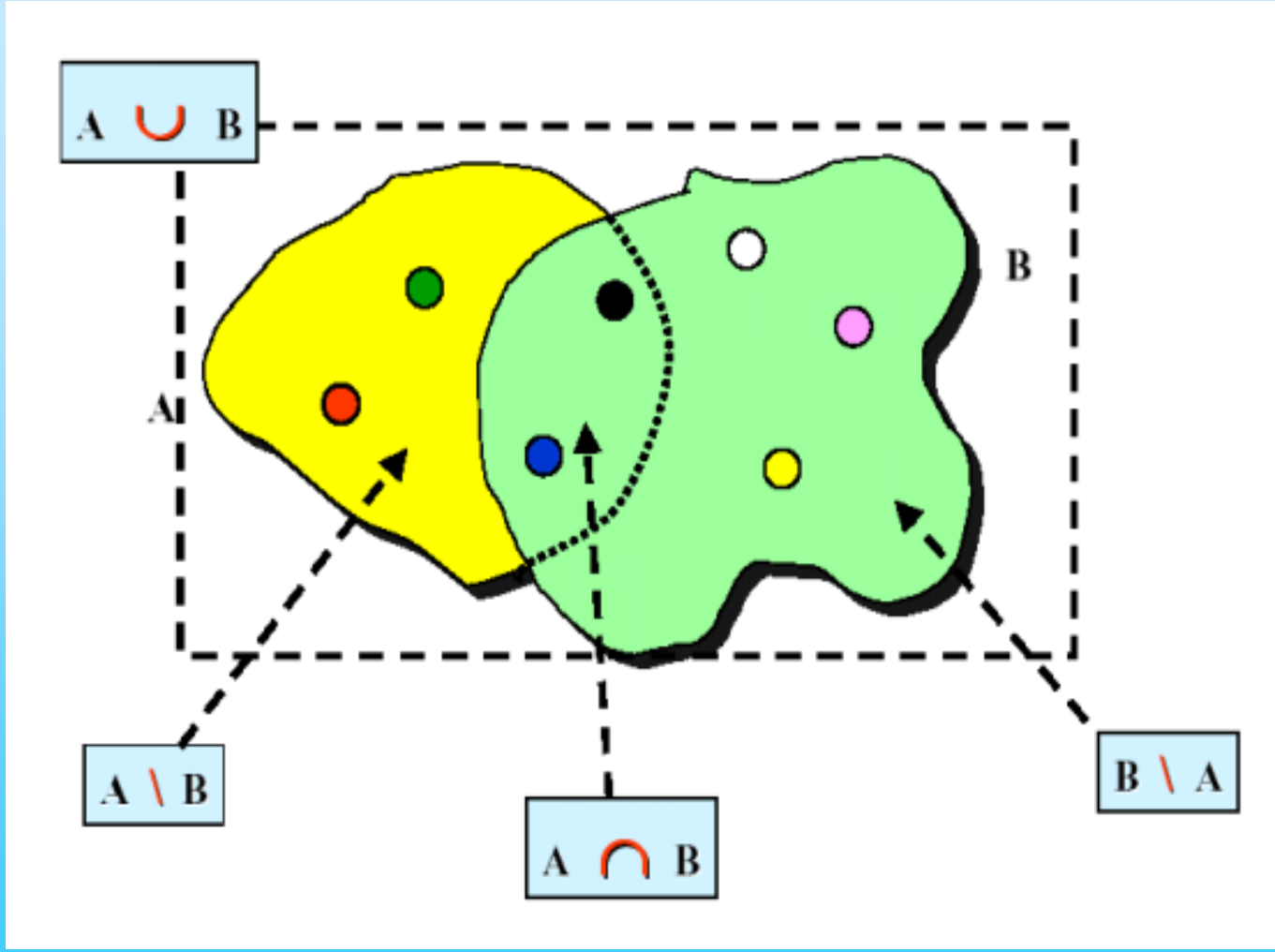
$$\{1, 2\} \cup \{2, 3\} = \{1, 2, 3\}$$

$$\{1, 2\} \cap \{2, 3\} = \{2\}$$

$$\{1, 2\} \setminus \{2, 3\} = \{1\}$$

$\overline{M}$

Beispiele für elementare Mengenoperationen



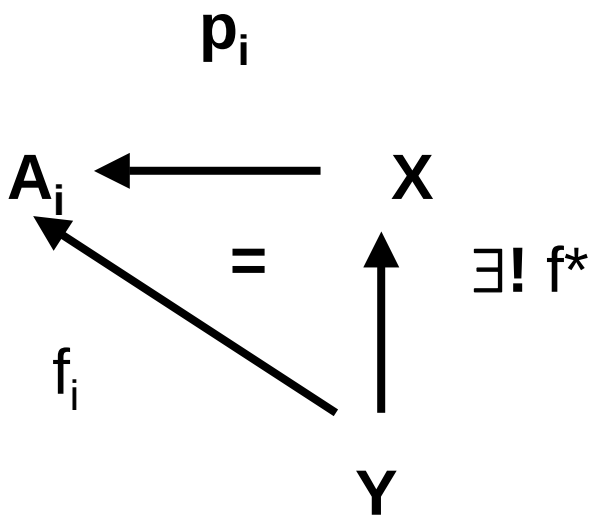
Produkt von Mengen

Definition Sei $(A_i, i \in I)$ **eine Familie von Mengen.**

Eine Menge X zusammen mit einer Familie von Abbildungen p_i: X → A_i heißt Produkt der $(A_i, i \in I)$, wenn folgendes gilt:

Zu jeder Menge Y und jeder Familie von Abbildungen $f_i : Y \to A_i$ existiert genau eine Abbildung $f^*: Y \to X$ mit

$$\forall_{i \in I} \quad p_i * f^* = f_i$$



Produkt von Mengen

Beispiele

$$A \times B = \{ (a, b) \mid a \in A \text{ und } b \in B \}$$

(kartesisches) Produkt

- verallgemeinerte Produktbildung für n Mengen ($n \geq 2$) :

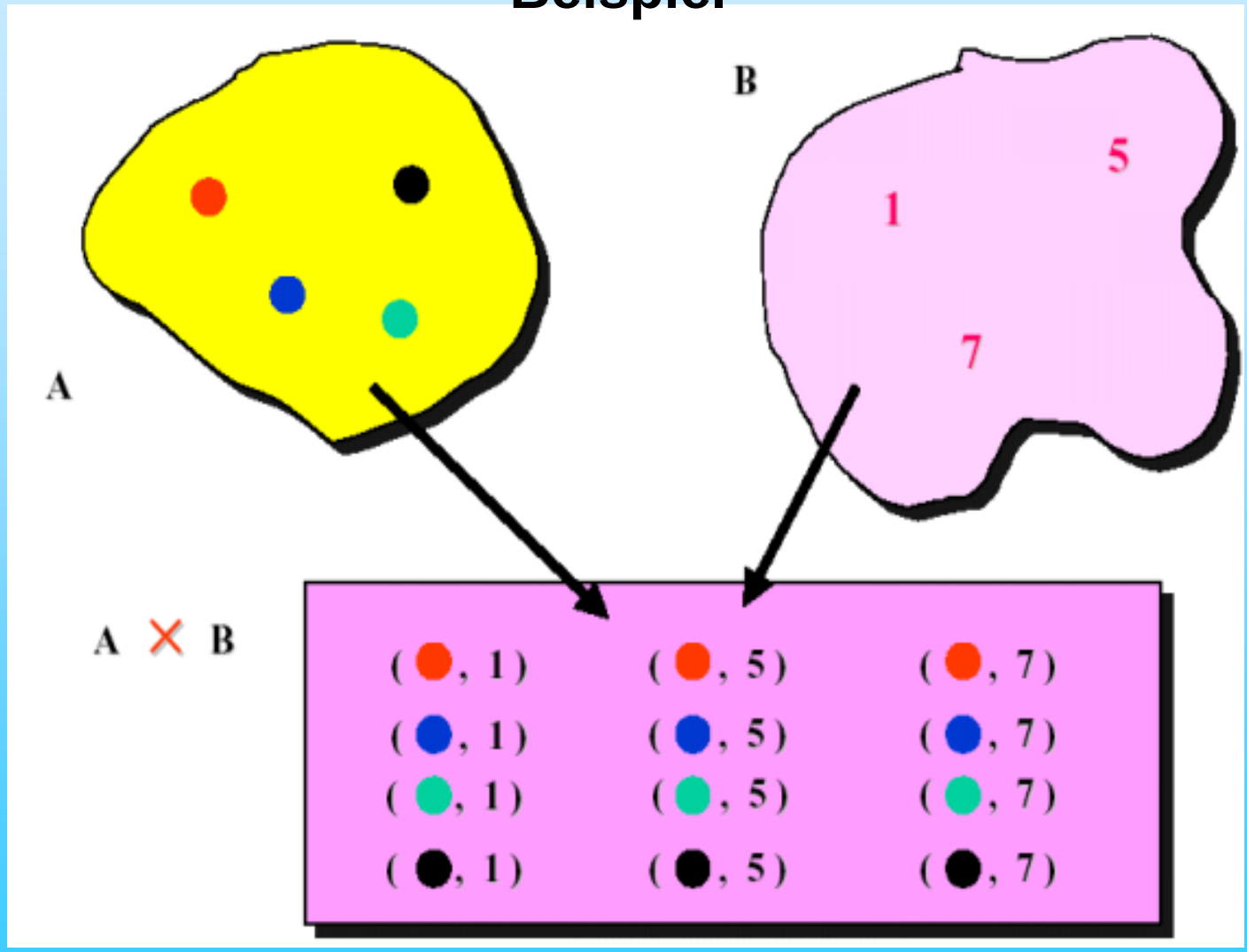
$$A1 \times \ldots \times An = \{ (a1, \ldots, an) \mid ai \in Ai \}$$

- Elemente des Produkts von n Mengen heissen (n)-Tupel.

- spezielle Bezeichnungen für Tupel:
  - n = 2: Paare
  - n = 3: Tripel
  - n = 4: Quadrupel

Produkt AxB von Mengen

Beispiel



Relationen

Definition

Eine n-stellige Relation R ist eine Teilmenge des (karthesischen) Produktes von n Mengen A_1, A_2,...,A_n:

$$R \subseteq A_1 \times A_2 \times ... \times A_n$$

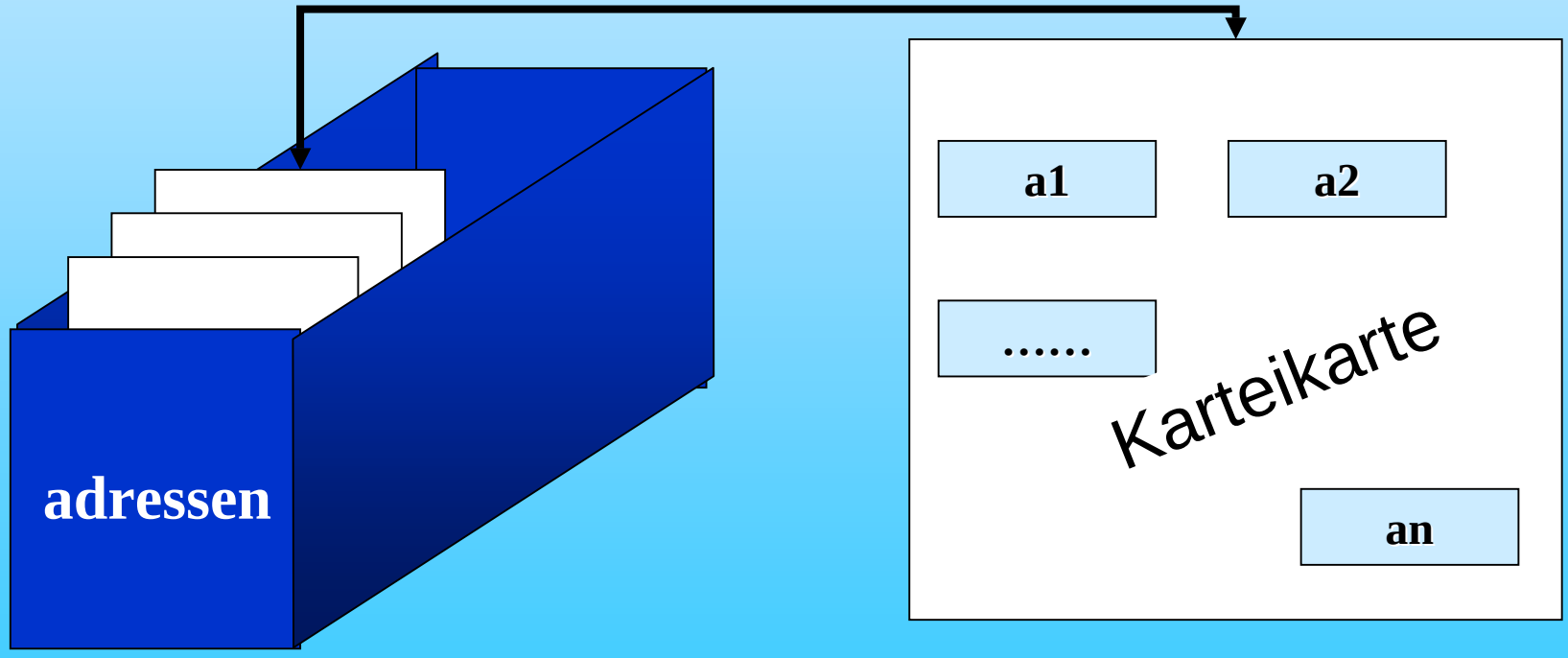
Schreibweise:

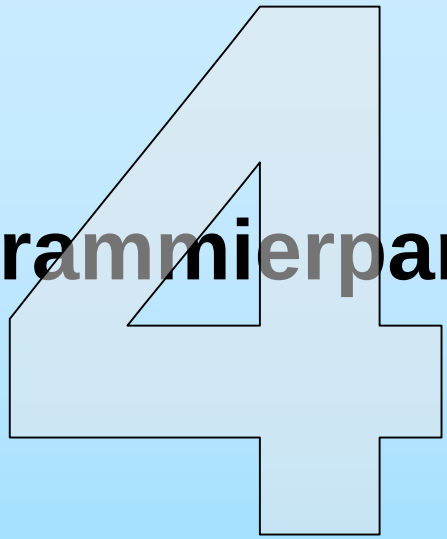
$$(a_1, a_2,...,a_n) \in R \leftrightarrow R\ (a_1, a_2,...,a_n)$$

Relationen
Beispiele

1. **Datenbanken: Tabelle**

   Beispiel: adressen(a1,a2,…,an)





Programmierparadigmen

- **Imperatives Programmieren**
- **Funktionales Programmieren**
- **Deklaratives Programmieren**
- **Objektorientiertes Programmieren**

Programmlängen-Vergleich

Programmiersprache	Programmgröße in Quellseiten
Fortran	36
Cobol	25
Ada	24
PL/I	22
C	22
Pascal	20
Basic	19
MProlog	9

1. Imperatives Programmieren

Es wird beschrieben, **WIE** ein bestimmtes Problem gelöst werden soll.

- Assembler
- ADA
- BASIC
- C / C++
- COBOL
- FORTRAN
- Java
- Modula
- PASCAL
- Perl
- PL/1
- Simula
- Smalltalk
- und viele mehr...

2. Funktionales Programmieren

Das Programm ist eine Folge von Funktionen.

- Lisp
- Logo
- Haskell
- ML
- Hope
- Scheme
- Concurrent Clean
- Erlang
- NESL
- Sisal
- Miranda

3. Deklaratives Programmieren

Es wird beschrieben, **WAS** das Problem ist, nicht jedoch wie dieses zu lösen ist. Die Lösung muß der Computer finden.

- **Prolog**
- Goedel
- Escher
- Elf
- Mercury

4. Objektorientiertes Programmieren

Wird in Verbindung mit den drei vorhergehenden Paradigmen verwendet.

Imperativ:

- Smalltalk
- Java
- Eiffel
- C++
- Object Pascal

Funktional:

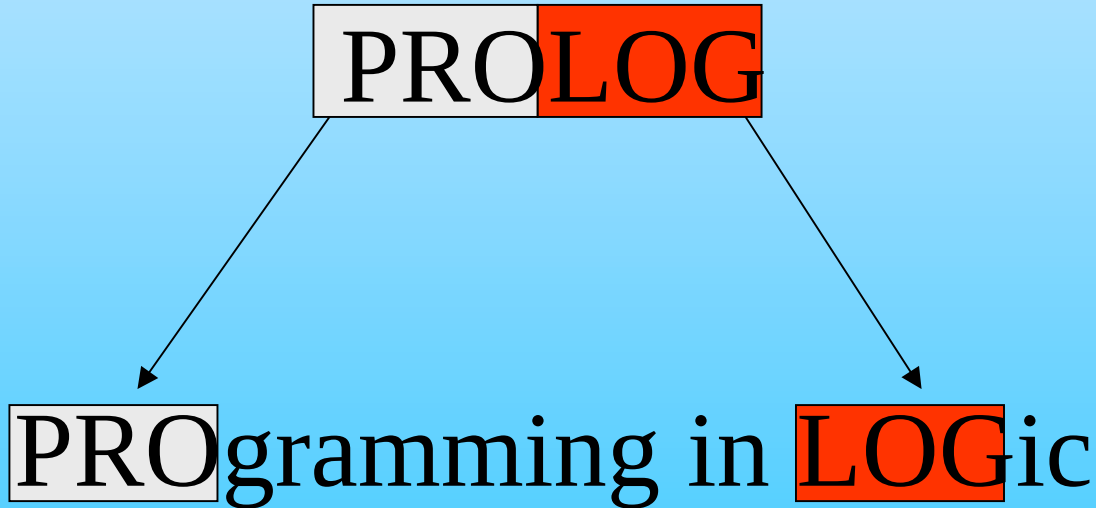
- XLISP
- Haskell
- andere...

Deklarativ:

- Bestimmte Versionen von Prolog
- und andere...

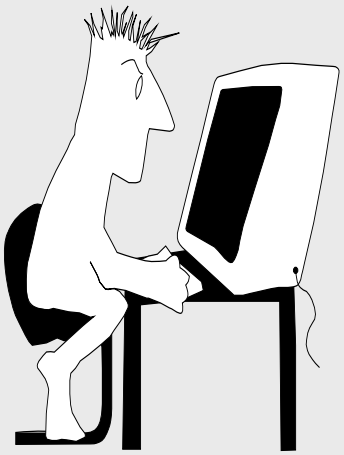
Nahezu alle neuen Programmiersprachen sind objektorientiert!

Für was steht PROLOG?



PROLOG

PROgrammieren in LOGig



Fakten und Regeln

Fakten und Regeln

Behauptung (Frage)

wahr/falsch (Antwodert)

Prolog-Progammierer

Prolog-Programiersystem

Benutzer

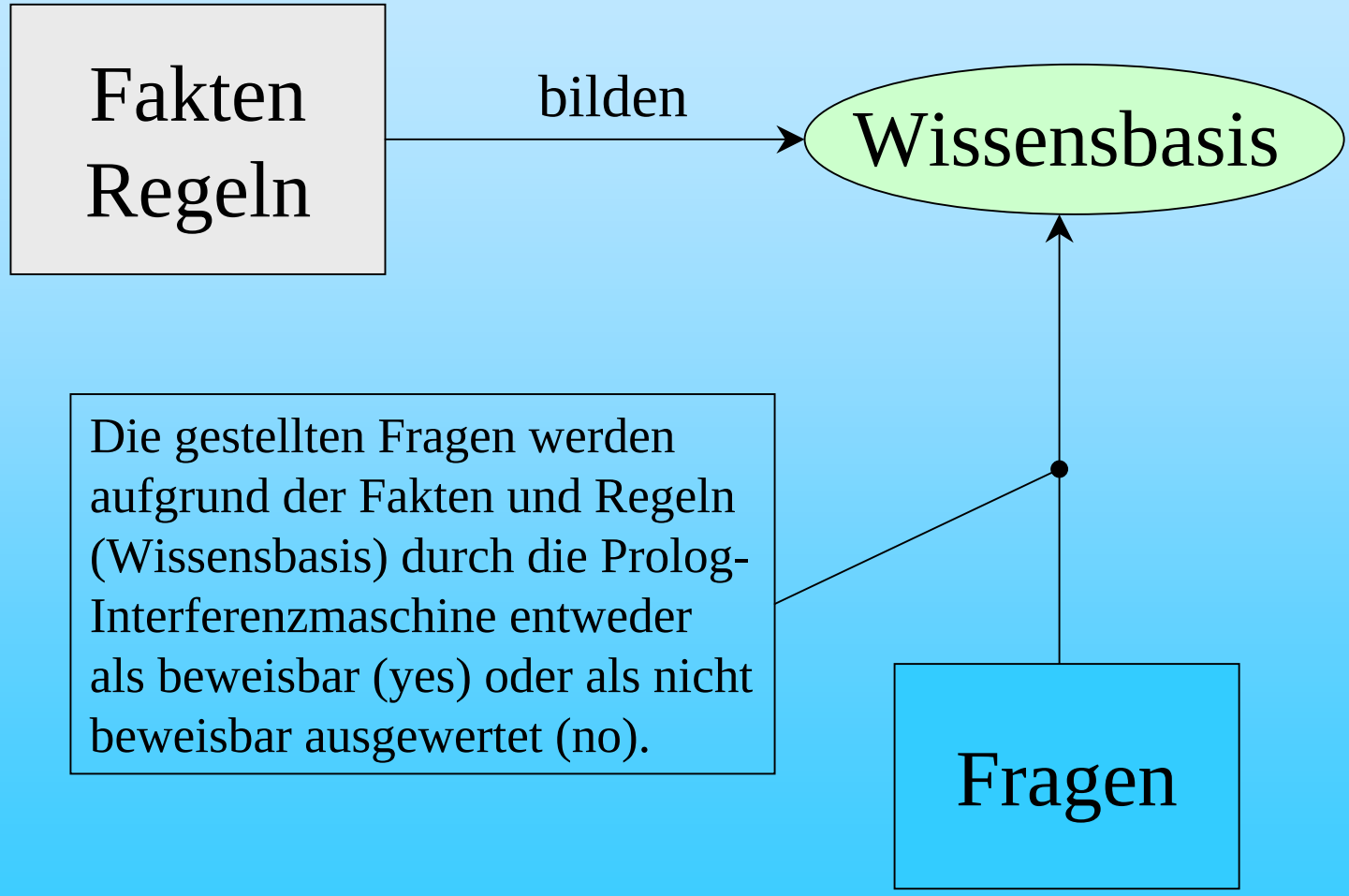
Anwendungsgebiete

- Expertensystem (Diagnosesystem)
- Relationale Datenbanken
- mathematische Logik
- abstrakte Problemlösungen
- Simulieren des menschlichen Sprachverstehens
- automatischer Entwurf
- Lösung von symbolischen Gleichungen
- Analyse biochemischer Strukturen
- zahlreiche Gebiete der KI

Geschichte der Logischen Programmierung

Die Geschichte der Logischen Programmierung ist nicht sehr lang. Die theoretischen Grundlagen wurden in den 70er Jahren erarbeitet. Der erste Prolog-Interpreter wurde 1972 von Alain Colmerauer in Marseilles geschrieben. Erst Anfang der 80er Jahre kamen die ersten kommerziellen Prolog-Interpreter auf den Markt.
Durch die Wahl von PROLOG als Sprache der Rechner der 5. Generation bei einem jap. Forschungsprojekt gelang der weltweite Durchbruch.

Das Prinzip von PROLOG



PROLOG

PROgrammieren in LOGig

Prologprogramm

- **Fakten**
- **Regeln**
- **Anfragen**

$p(a_1,\ldots,a_n).$

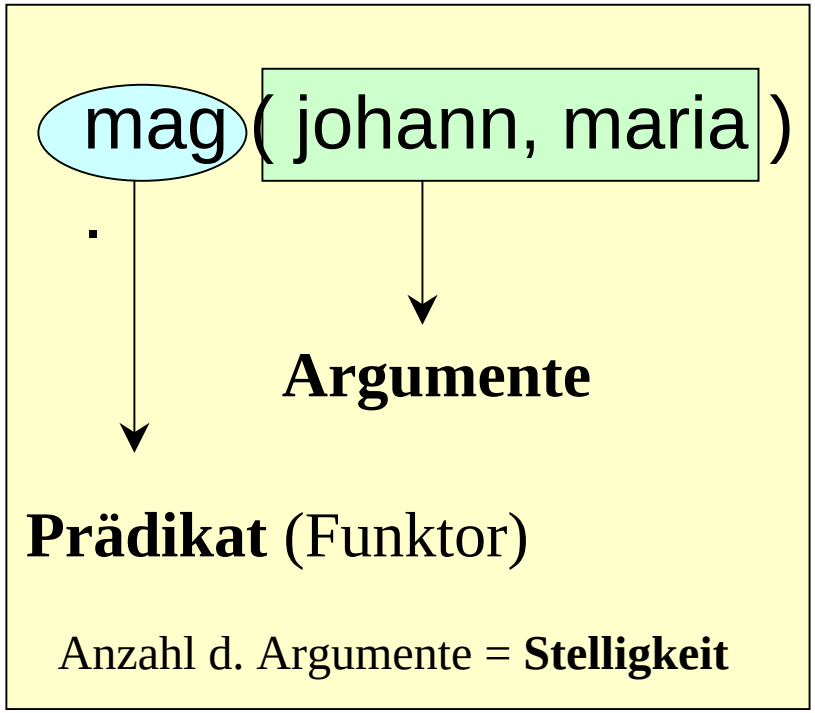
- p ist der Name des Fakts
- $a_1, \ldots, a_n$ sind die Argumente des Fakts

Fakten 2

Beispiele:

Schreibweise in Prolog:	Natürliche Bedeutung:
• ‘die Sonne scheint‘.	• Die Sonne scheint.
• es_regnet.	• Es regnet.
• mensch(sokrates).	• Sokrates ist ein Mensch.
• männlich(daniel).	• Daniel ist männlich.
• mag(johann,maria).	• Johann mag Maria.
• besitzt(johann,gold).	• Johann besitzt Gold.
• vater(hans, gabriel).	• Hans ist der Vater von Gabriel.

Fakten 2



Die Reihenfolge von johann und maria ist **nicht** egal, d.h. es ist ein Unterschied ob man mag(johann,maria) oder mag(maria,johann) schreibt. Ersteres könnte z.B. „Johann mag Maria“, das andere „Maria mag Johann“ bedeuten.

Prolog weiß nicht, was ein Fakt bedeutet. Deshalb muß man selber festlegen was z.B. mag(johann,maria) bedeutet.

- Ein Fakt kann beliebig viele Argumente haben.
- Ein Fakt muß mit einem Punkt beendet werden.
- Ein Fakt muß mit einem Kleinbuchstabe oder ‘ beginnen.

PROLOG

PROgrammieren in LOGig

Prologprogramm

- **Fakten**
- **Regeln**
- **Anfragen**

Regel

b1 und b2,…und bn → f

Wenn b1 und b2,..,und bn gelten,dann gilt auch f.

Prologschreibweise

f :- b1,b2,…,bn.

Regeln 2

Die linke Seite (vor :-) ist wahr, wenn die rechte Seite bewiesen werden kann.

Ein Komma zwischen zwei Fakten auf der rechten Seite entspricht einer logischen UND-Verknüpfung. Ein Semikolon entspricht einer ODER-Verknüpfung. Es kann geklammert werden.

Die UND- bzw. ODER-Verknüpfungen können auch bei Fragen angewendet werden.

Mit not kann verneint werden.

Die Regeln werden auch Horn-Klauseln genannt.

PROLOG

PROgrammieren in LOGig

Prologprogramm

- **Fakten**
- **Regeln**
- **Anfragen**

Gelten die Fakten
p1, p2,…, pn ?

Prologschreibweise
? p1, p2,…, pn

Variablen

Wissensbasis:

besitzt(john, gold).

besitzt(john, buch).

mag(john, mary).

mag(joe, fisch).

Um festzustellen, was John besitzt, kann man die Frage **besitzt(john,X)** stellen. X ist dabei eine Variable. Prolog antwortet dabei mit folgendem:

X = gold
X = buch

_ ist eine anonyme Variable. Bei dem Beispiel oben, würde _ anstatt X folgendes bedeuten: Besitzt John etwas? (Antwort: yes).

Variablen beginnen mit einem Großbuchstaben oder Unterstrich!

PROLOG

PROgrammieren in LOGig

Fakten

Sokrates ist ein Mensch.

•Regeln

Alle Menschen sind sterblich.

Anfragen

Ist Sokrates sterblich?

Prologschreibweise

mensch(sokrates).

mensch(X) → sterblich(X)

sterblich(X):- mensch(X).

? sterblich(sokrates)

Expertensysteme

Prinzipieller Aufbau

Benutzer :
Experte …. Laie

Dialogkomponente

Erklärungskomponente

Inferenzkomponente

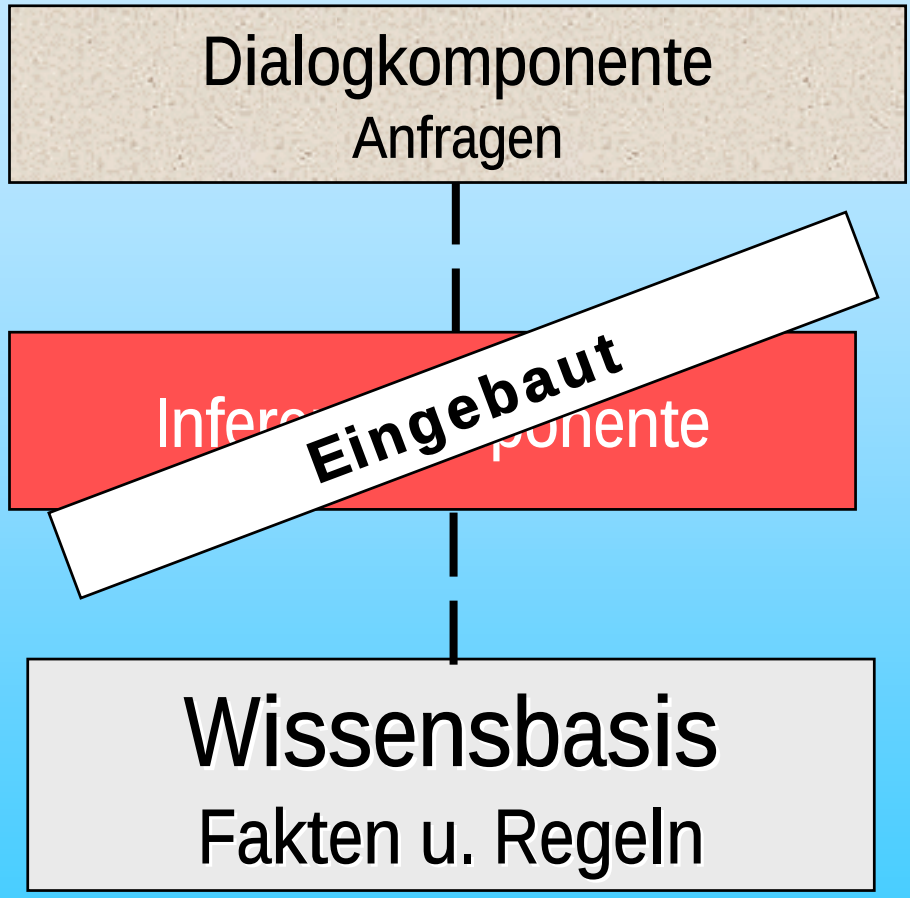
Wissensbasis

Wissenserwerbskomponente

Datenbank

Experte

Prologprogramm <-> Expertensystem



PROLOG

PROgrammieren in LOGig

A, B und C stehen vor Gericht.

- A sagt aus, dass B lügt.
- B sagt aus, dass C lügt.
- C sagt aus, dass A und B lügen.

Wer lügt, wer sagt die Wahrheit?

Prologprogramm

```
ist_Lügner(wahr,lügt).
ist_Lügner(lügt,wahr).
beide_lügen(wahr,lügt,lügt).
beide_lügen(lügt,wahr,lügt).
beide_lügen(lügt,lügt,wahr).
beide_lügen(lügt,wahr,wahr).
```

```
?- ist_Lügner(A,B),
   ist_Lügner(B,C),
   beide_lügen(C,A,B).
```

Arithmetik

+ - * /	Addition, Subtraktion, Multiplikation, Division
mod	Modulo
// ^	Gleitzahl-Division, Potenzierung
( )	Priorität
is	Zuweisen eines arith. Ausdruckes
> <	größer, kleiner
=> =<	größer gleich, kleiner gleich (zuerst = dann > !)
=:=	gleich (arithmetisch)
=\=	ungleich (arithmetisch)

Beispiel: Berechnen der Fakultät

```
fak(0,1).

fak(N,X) :-    N > 0,   M is N - 1,   fak(M,Y),   X is N * Y.
```

Aufruf: ?- fak(0,N).

N = 1

?- fak(6,N).

N = 720

Relationen

Binäre Relation

Definition

Eine binäre Relation R ist eine Teilmenge des karthesischen Produktes zweier Mengen A und B:

$$R \subseteq A \times B$$

Für

$$(a,b) \in R$$

schreibt man auch

aRb oder R (a,b)

Funktionen

- Eine binäre Relation $R \subseteq A \times B$ heisst
  - rechtseindeutig, wenn es für jedes $a \in A$ höchstens ein $b \in B$ gibt mit $(a, b) \in R$
  - linkseindeutig, wenn es für jedes $b \in B$ höchstens ein $a \in A$ gibt mit $(a, b) \in R$

- Rechtseindeutige Relationen werden auch Funktionen (bzw. Abbildungen) genannt. Sie dienen als mathematische Modelle eindeutiger Zuordnungen.

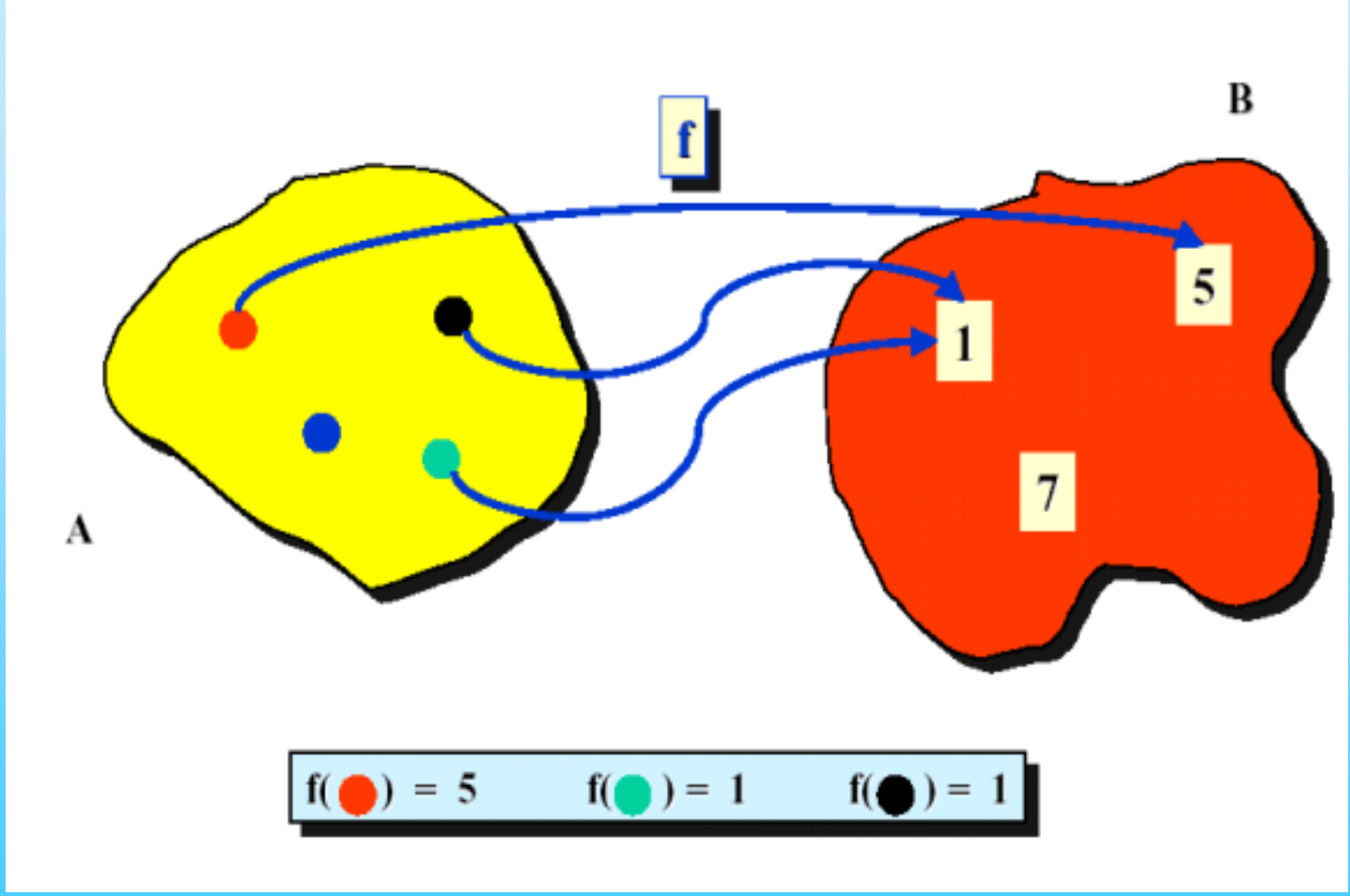
- Eine rechtseindeutige Relation f über $A \times B$ wird dann als Funktion von A nach B bezeichnet.

- Notation: $f : A \to B$ statt $f \subseteq A \times B$ und
  $f(a) = b$ statt $(a, b) \in f$

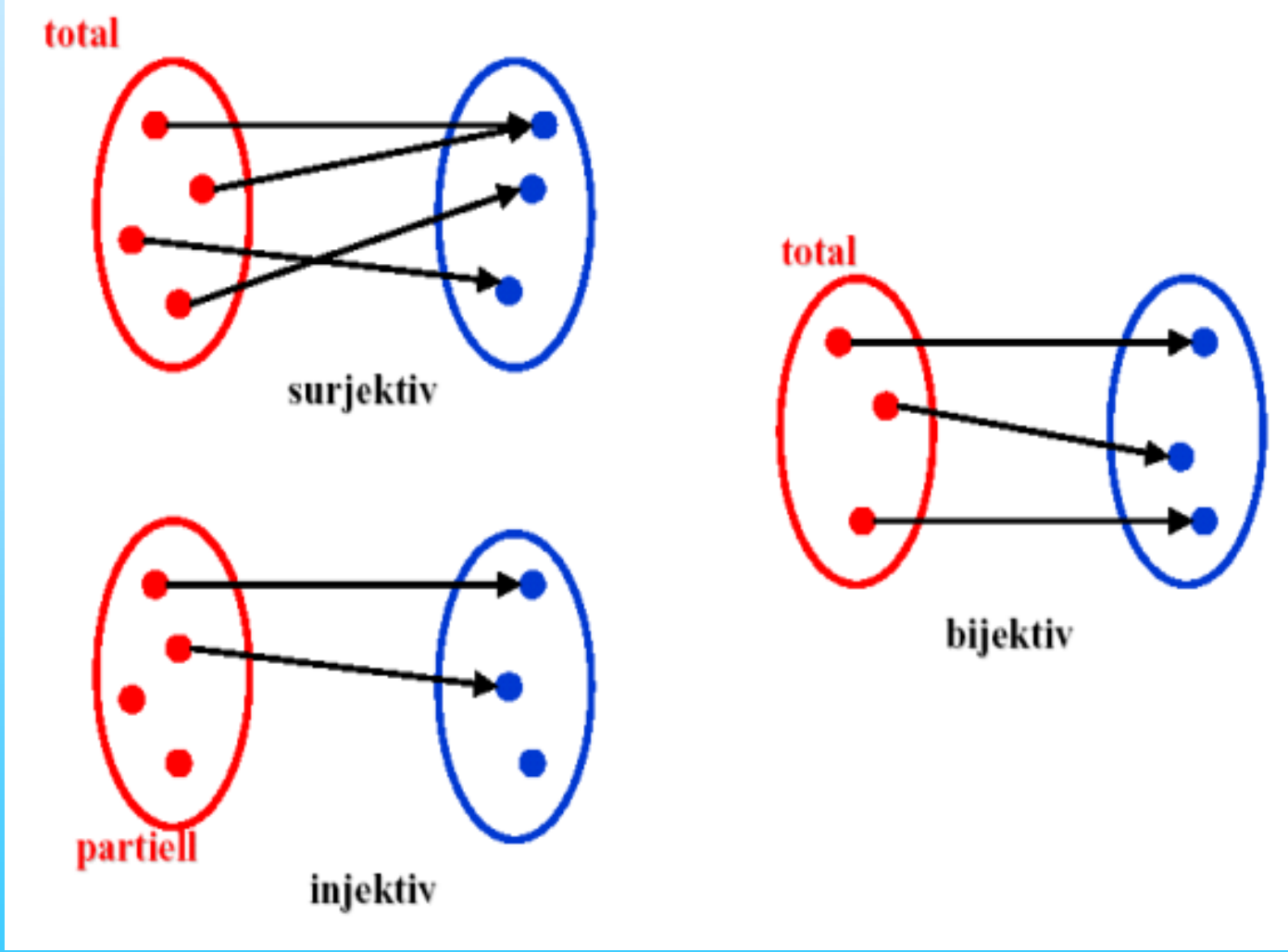
- A heisst Definitionsbereich von f, B wird Bildbereich von f genannt.

- Statt als Tabelle werden Funktionen meist als Pfeildiagramme notiert.

Funktionen: Beispiel



Spezielle Funktionen



Binäre Relationen und Graphen

Eine binäre Relation R kann durch einen Graphen veranschaulicht werden in dem jedes Tupel (a,b) als Kante zwischen den Knoten a und b interpretiert wird.

$(a,b) \in R$  **a**  **b**

Umgekehrt entspricht jede Kante (a,b) eines Graphen die zwei Knoten a und b verbindet einer Relation R für die

aRb als „a ist mit b direkt verbunden“ interpretiert werden kann.

Reflexivität

Eine binäre Relation $R \subseteq S \times S$ ist reflexiv, wenn jedes Element von S zu sich selbst in Relation steht:

$$\forall x: x \in S: xRx$$

zB: die Relation „hat dieselbe Mutter wie“ ist reflexiv

Reflexive Relationen können durch einen Graphen modelliert werden, bei dem alle Knoten Schlaufen haben.

Symmetrie

Eine binäre Relation $R \subseteq S \times S$ ist symmetrisch, wenn aus xRy auf yRx geschlossen werden kann:

$$\forall\ x,y: x,y \in S: xRy \Rightarrow yRx$$

zB: die Relation „ist verheiratet mit“ ist symmetrisch

Symmetrische Relationen entsprechen ungerichteten Graphen.

Transitivität

Eine binäre Relation $R \subseteq S \times S$ ist transitiv, wenn aus xRy und yRz auf xRz geschlossen werden kann:

$$\forall\, x,y,z: x,y,z \in S: (xRy \wedge yRz) \Rightarrow xRz$$

zB: die Relation „ist Vorfahre von“ ist transitiv

Äquivalenzrelationen

Definition:

Eine Äquivalenzrelation auf einer Menge M ist eine binäre Relation ~ auf M, die

- **reflexiv,**
- **symmetrisch und**
- **transitiv ist.**

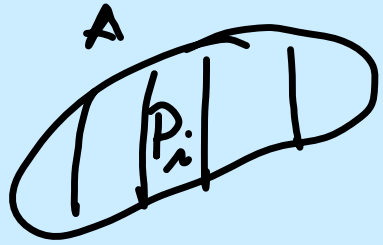
Definition Äquivalenzklasse

Sei ~ eine Äquivalenzrelation auf M und a ∈M. Dann heißt die Menge der zu a äquivalenten Elemente die Äquivalenzklasse von a.

$$\bar{a} := \{x \in M \mid a \sim x\} \subseteq M$$

Definition $\mathcal{P} = \{P_i \subseteq A ; i \in I\}$

$\mathcal{P}$ heißt **Partition auf A** $\Leftrightarrow$ def.



P1) $\forall_{i \in I} \; P_i \neq \emptyset$

P2) $\forall_{i,j \in I} \; P_i \cap P_j \neq \emptyset \Rightarrow P_i = P_j$

P3) $\bigcup_{i \in I} P_i = A$