

Kapitel 2b Halbgruppen und Relationen

- 2.1 Halbgruppen und Monoide
- 2.4 Endliche Automaten
- 2.2 Relationen und Graphen
- 2.5 Petri-Netze
- 2.3 Ordnungsrelationen, Halbverbände, Verbände
- 2.6 Relationale Algebra

2.4 Endliche Automaten

1/39

endlicher Automat:
gedachte Maschine, die Zeichenreihen liest, und über ein beschränktes Gedächtnis des Gelesenen verfügt

↳ statt Zeichen lesen: jede Art unterscheidbarer Eingangssignale verarbeiten

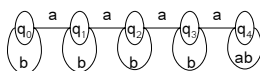
Zustand:

(endliches) Gedächtnis über Vorgeschichte der eingelesenen Zeichen
↳ Ein endlicher Automat besitzt endliche Menge Q von Zuständen

Arbeitsweise:

- beginnt mit Anfangszustand q_0
- liest Zeichen aus Eingabe und geht in neuen Zustand
- ↳ Darstellung als Semi-Thue-System oder gerichteter Graph:

$q_0 a \rightarrow q_1$ $q_0 b \rightarrow q_0$
 $q_1 a \rightarrow q_2$ $q_1 b \rightarrow q_1$
 $q_2 a \rightarrow q_3$ $q_2 b \rightarrow q_2$
 $q_3 a \rightarrow q_4$ $q_3 b \rightarrow q_3$
 $q_4 a \rightarrow q_4$ $q_4 b \rightarrow q_4$



2.4 Endliche Automaten – Einleitung

2/39

Was macht man damit?

- Reaktion bei Erreichen bestimmter Zustände (**Moore-Automat**)
 - Prüfung, daß Eingabe korrekt (**Akzeptor**, verlangter Endzustand erreicht?)
- Reaktion bei bestimmten Zustandsübergängen (**Mealy-Automat**)

Beispiele:

- primitiv, aber wirkungsvoll:
 - Schlüssel im Schloß umdrehen (Zustände offen-geschlossen)
 - Gegenstände abzählen: 0, 1, 2, 3, viele (5 Zustände)
 - Jeder vierten Person etwas schenken
- komplizierter:
 - Prüfung der korrekten Schreibweise von Gleitpunktzahlen
 - Beschreibung der Arbeitsweise von Fahrkartenautomaten usw.

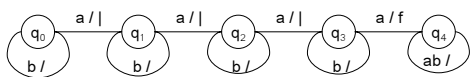
2.4 Arten endlicher Automaten

3/39

Klassifikation nach Art der Ausgabe:

Mealy-Automat: Zustandsübergänge erzeugen als Ausgabe Wörter $t \in \Delta^*$ über Zeichenvorrat Σ

Beispiel:



Moore-Automat: Zustände erzeugen als Ausgabe Wörter $t \in \Delta^*$ über Zeichenvorrat Σ

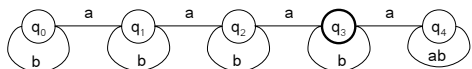


2.4 Arten endlicher Automaten

4/39

▪ **Akzeptor:** spezieller Moore-Automat, der eine Menge von Endzuständen hat. Endet der Automat in einem Endzustand, dann akzeptiert er die Eingabe.

Beispiel:



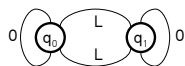
2.4 Beispiel: Parität

5/39

Parität einer Bitfolge w : gerade oder ungerade Anzahl der L in w

Automat

- q_0 : in bisher gelesenen Bits: Anzahl der L gerade (Anfangszustand)
- q_1 : in bisher gelesenen Bits: Anzahl der L ungerade



Annahme: Automat im Zustand q_0 nach Verarbeitung von Wort w
 $\Rightarrow w$ hat gerade Parität

Annahme: Automat im Zustand q_1 nach Verarbeitung von Wort w
 $\Rightarrow w$ hat ungerade Parität

\Rightarrow Akzeptor mit Endzuständen q_0 und q_1



2.4 Akzeptoren für Bezeichner

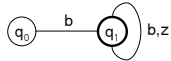
6/39

Bezeichner: Folge von Buchstaben und Ziffern, wobei erstes Zeichen Buchstabe sein muß

- typisch für Bezeichner von Variablen etc. in Programmiersprachen

Akzeptor:

- q_0 : Anfangszustand
- q_1 : Endzustand (erstes Zeichen ist Buchstabe)



- b steht für beliebigen Buchstaben
- z steht für beliebige Ziffer



2.4 Akzeptoren für Zahlen

7/39

ganze Zahlen: nicht leere Folge von Ziffern

Festpunktzahlen:

- z^n , z^m wobei $m \geq 1$ und z für beliebige Ziffern steht

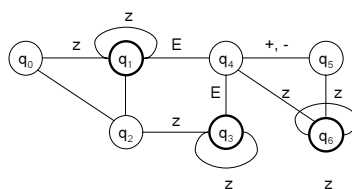
Gleitpunktzahlen:

- z^n , $z^m E e$ wobei $m \geq 1$ und e eine ganze Zahl (evtl. mit Vorzeichen + oder -) ist oder
- $z^n E e$ wobei $n \geq 1$ ist.



2.4 Akzeptoren für Zahlen

8/39



- q_0 Startzustand
- q_1 ganze Zahl
- q_2 Punkt
- q_3 Festpunktzahl
- q_4 Exponent
- q_5 Vorz. Exponent
- q_6 Gleitpunktzahl

q_1 akzeptiert natürliche Zahlen, q_3 akzeptiert Festpunktzahlen, q_6 akzeptiert Gleitpunktzahlen



2.4 Akzeptoren und reguläre Sprachen

9/39

endlicher Akzeptor:

$A = (\Sigma, Q, q_0, F, R)$ wobei

- Σ Zeichenvorrat,
- Q nicht leere Menge von Zuständen ($Q \cap \Sigma = \emptyset$),
- $q_0 \in Q$ Anfangszustand,
- $F \subseteq Q$ Menge der Endzustände und
- R Menge von Zustandsübergangsregeln der Form $qa \rightarrow q'$ mit $q, q' \in Q, a \in \Sigma$ ist.

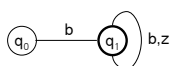
Bemerkung: $(\Sigma \cup Q, \rightarrow)$ ist Semi-Thue-System



2.4 Akzeptoren und reguläre Sprachen

10/39

Beispiel: $A = (\{b, z\}, \{q_0, q_1\}, q_0, \{q_1\}, \{q_0b \rightarrow q_1, q_1b \rightarrow q_1, q_1z \rightarrow q_1\})$



Von Automat A **akzeptierte Sprache:**

- $L(A) = \{w \in \Sigma^* \mid \text{es gibt } q_e \in F, \text{ so daß } q_0w \rightarrow^* q_e\}$

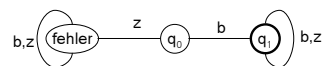
Beispiel: $L(A) = \{bw \mid w \in \{b, z\}^*\}$



2.4 Vollständige Akzeptoren

11/39

Beispiel:



- kein Übergang $q_0 z \rightarrow \dots$ im Akzeptor für Bezeichner
- Grund: Fehler
- Einführung eines Fehlerzustands $fehler \in F$
 - mit $q_0 z \rightarrow fehler$
 - $fehler b \rightarrow fehler$
 - $fehler z \rightarrow fehler$

vollständiger Akzeptor $A = (\Sigma, Q, q_0, F, R)$: für alle $q \in Q$ und $a \in \Sigma$ existiert $q' \in Q$ mit $qa \rightarrow q' \in R$



2.4 Automat als Semi-Thue-System

15/39

- Bei bisherigen Beispielen:
 - Übergänge $qa \rightarrow q'$ bilden zusammengenommen eine Übergangsfunktion $\delta: Q \times \Sigma \rightarrow Q$
 - Ein endlicher Automat, der die Eigenschaft besitzt, daß die Übergänge zusammengenommen eine (Übergangs-)Funktion bilden, heißt **deterministisch**
- Allgemeiner Fall: Indeterminismus zugelassen
 - Die Übergänge definieren ein durch P endlich erzeugtes Semi-Thue-System mit Eingabezeichenvorrat Σ und syntaktischen Hilfszeichen $q \in Q$
 - Die Auffassung eines endlichen Automaten als Semi-Thue-System stellt die Beziehung zu regulären Chomsky-Grammatiken her



2.4 Regulärer Ausdruck

16/39

- Ein regulärer Ausdruck R über einem Zeichenvorrat Σ ist induktiv definiert durch:
 - Das leere Wort ist ein regulärer Ausdruck
 - $a \in \Sigma$ ist ein regulärer Ausdruck
 - Ist R ein regulärer Ausdruck, dann auch $(R)^*$
 - Sind R, S reguläre Ausdrücke, so sind auch (RS) und $(R + S)$ reguläre Ausdrücke ($R + S$ bedeutet „R oder S“ (Vereinigung))
- Meist gilt auch die leere Menge \emptyset als regulärer Ausdruck

Beispiele:

- Bezeichner: $b(b+z)^*$
- Ganze Zahlen: zz^*
- Dezimalbrüche: $zz^* \cdot / \cdot zz^*$



2.4 Rechnen mit regulären Ausdrücken

17/39

| | | | |
|-----------------|-----|---------------|-----------------|
| $R + S$ | $=$ | $S + R$ | Kommutativität |
| $(R + S) + T$ | $=$ | $R + (S + T)$ | Assoziativität |
| $(R S) T$ | $=$ | $R (S T)$ | Assoziativität |
| $(R + S) T$ | $=$ | $R T + S T$ | Distributivität |
| $R (S + T)$ | $=$ | $R S + R T$ | Distributivität |
| $R + \emptyset$ | $=$ | R | Neutralement |
| $R \varepsilon$ | $=$ | R | Neutralement |
| $R \emptyset$ | $=$ | \emptyset | Nullelement |
| $R + R$ | $=$ | R | Idempotenz |
| $(R^*)^*$ | $=$ | R^* | Idempotenz |



2.4 Rechnen mit regulären Ausdrücken

18/39

$$R^* = \epsilon + RR^*$$

$$R^* = R + R^*$$

$$\epsilon^* = \epsilon$$

$$\emptyset^* = \epsilon$$

Beweis: Übung

Satz:

Seien R, S reguläre Ausdrücke über Σ . Dann ist $X = S^*R$ Lösung der Gleichung $X = R + S X$.

Beweis: Übung



2.4 Eigenschaften regulärer Ausdrücke

19/39

Für reguläre Ausdrücke gelten die Gesetze:

Kommutativität, Assoziativität, Distributivität, Neutrales Element, Idempotenz

Satz:

Eine Menge L^* ist genau dann Sprache einer regulären Grammatik G , wenn L durch einen regulären Ausdruck beschrieben wird.

Beweis-Skizze:

- Zeige, daß aus einem regulären Ausdruck eine reguläre Grammatik konstruiert werden kann

Prinzip: induktiv aus den Schritten 1 - 4 der Definition

- Zeige umgekehrt, daß aus einer regulären Grammatik der zugehörige reguläre Ausdruck konstruiert werden kann

Prinzip: jedes Nichtterminal der Grammatik wird als Bezeichner für reguläre Ausdrücke gebraucht



2.4 Vom regulären Ausdruck zum endlichen Akzeptor

20/39

- Beweis des vorhergehenden Satzes liefert ein konstruktives Vorgehen zum Aufbau eines endlichen Akzeptors

Skizze dieses konstruktiven Vorgehens:

Geg.: regulärer Ausdruck R

Durch folgende Regeln wird R in einen endlichen Akzeptor überführt:

1. Füge zu Beginn von R sowie nach jedem terminalen Zeichen eine Zahl ein, die einen Zustand repräsentiert
2. Einzelnes Zeichen bedeutet: Zustandsübergang
3. Vereinigung führt zu Zustandsübergängen der zu Beginn gegebenen Zustände in alle durch Einzelzeichen erreichbaren Folgezustände
4. Kleenescher Stern S^* führt zu Zustandsübergängen aus sämtlichen Endzuständen von S in die aus den Anfangszuständen aus S mit einem Zeichen erreichbaren Zustände



2.5 Petrinetze

21/39

Carl Adam Petri, geb. 1926, früherer Institutsleiter der GMD, Bonn

Petri-Netze: verallgemeinern endliche Automaten und sind die einfachste Form, parallele und verteilte Systeme zu beschreiben

Grundidee: Gegeben ein paarer Graph von Übergängen (**Transitionen:** Kästchen) und Zuständen (**Stellen:** Kreise).

- eine Stelle kann mehrere Transitionen als Nachfolger haben
- eine Transition kann mehrere Stellen als Nachfolger haben
- Regeln: jede **Stelle** kann eine oder mehrere Marken enthalten
 - **Feuern** einer Transition: jede Vorgängerstelle gibt eine Marke ab, jede Nachfolgerstelle erhält eine Marke (dabei Veränderung der Markenanzahl möglich)

Interpretation z.B.:

- Stelle: Prozeß, Vorgang; Transition: Botschaft übergeben
- Stelle: Zustand; Transition: Zustandsübergang



2.5 Petrinetze

22/39

▪ **Petrinetze** werden in der Informatik als Systemmodelle für Vorgänge und Organisationen verwendet, um Zustandsübergänge und Datenflüsse auch in parallelen und verteilten Systemen zu beschreiben

- Bestandteile und Arbeitsweise eines Petrinetzes $P = (S, T, F)$
 - **S: Stellen** repräsentieren Zustände, Bedingungen, Pufferinhalte
 - **T: Transitionen** (Übergänge) repräsentieren Zustandsübergänge, Aktivitäten oder Verarbeitung von Pufferinhalten
 - **F: Kanten** repräsentieren kausale oder zeitliche Abhängigkeiten
 - $s \rightarrow t$ bedeutet: Bedingung $s \in S$ muß erfüllt sein, damit die Transition $t \in T$ schaltet (feuert)
 s heißt **Vorstelle** von t
 - $t \rightarrow s$ bedeutet: Zustand $s \in S$ wird erreicht, wenn die Transition $t \in T$ feuert
 s heißt **Nachstelle** von t



2.5 Markierungsfunktion und Schaltregel

23/39

- Eine **Markierungsfunktion** $M_P(s)$ weist jeder Stelle $s \in S$ eine Anzahl von Marken zu
- Das Petrinetz heißt **binär** falls für alle $s \in S$: $M_P(s) \leq 1$

Schaltregel

▪ führt das Petrinetz von einer Markierung in eine neue Markierung über

▪ **Vorbedingung:** Transition t kann schalten, wenn für alle Vorstellen s_v gilt: $M_P(s_v) \geq 1$

▪ **Nachbedingung:** Wenn eine Transition schaltet, gilt für die neue Markierung M_P' :

- $M_P'(s_v) = M_P(s_v) - 1$ für alle Vorstellen s_v von t ,
- $M_P'(s_n) = M_P(s_n) + 1$ für alle Nachstellen s_n von t ,
- $M_P'(s) = M_P(s)$ sonst



2.5 Markierungsgraph

24/39

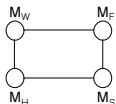
Mögliche Markierungen M_P bilden ihrerseits die Ecken eines gerichteten Graphen, dem sog. **Markierungsgraph** $G = (M, U)$

- $M = \{M_P\}$ Menge der Ecken
- $U \in M \times M$ Menge der Kanten

Es gilt $M_P \rightarrow M_{P'} \in U$, wenn es eine Transition t so gibt, daß das Schalten der Transition t die Markierung M_P in die Markierung $M_{P'}$ überführt

Beispiel: Markierungsgraph zum Jahreszeiten-Petrinetz

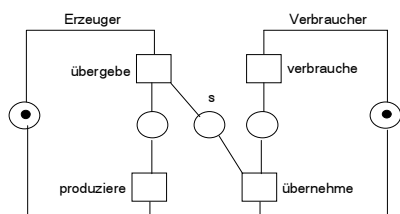
- Ecken: vier mögliche Markierungen M_F, M_S, M_H, M_W
- Kanten:



2.5 Beispiel: Erzeuger-Verbraucher-Problem

25/39

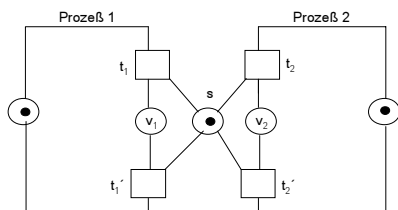
Ziel: Synchronisierung von zwei nebenläufigen Prozessen



2.5 Gegenseitiger Ausschluß

26/39

Ziel: Verhinderung, daß zwei Prozesse eine exklusiv nutzbare Ressource gleichzeitig verwenden.



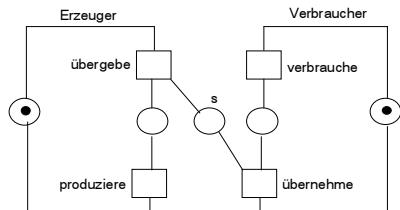
- Die Sequenzen $t_i \rightarrow v_i \rightarrow t_i'$ ($i = 1, 2$) heißen **kritische Abschnitte**
- s heißt ein **Mutex-Semaphor**



2.5 nochmals: Erzeuger-Verbraucher

27/39

Ziel: der Verbraucher darf nur weitermachen, wenn er vom Erzeuger etwas geliefert bekommt



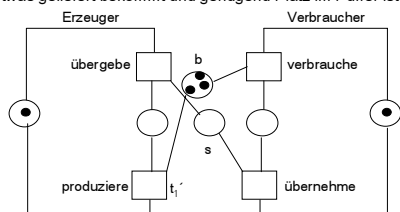
- Die Stelle s kann unbeschränkt viele Marken aufnehmen:
 - Erzeuger-Verbraucher mit unbeschränktem Puffer s



2.5 Erzeuger-Verbraucher mit beschränktem Puffer

28/39

Ziel: der Verbraucher darf nur weitermachen, wenn er vom Erzeuger etwas geliefert bekommt und genügend Platz im Puffer ist



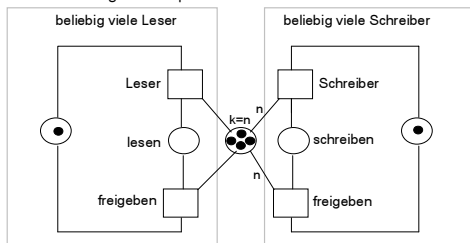
- Wenn die Stelle b n Marken enthält, kann Übergang t_1 n -mal feuern, sonst wartet der Erzeuger
 - Hier sind also $3+1 = 4$ Marken in Puffer s möglich, bevor der Erzeuger stoppt



2.5 Leser-Schreiber-System

29/39

Ziel: n Leser dürfen gleichzeitig Daten lesen, ein Schreiber verlangt Exklusivzugriff und sperrt alle Leser aus



- $k=n$: Stelle kann maximal n Marken aufnehmen
- n an einer Kante: n Marken fließen gleichzeitig
- keine Fairneß!



2.6 Schema-Operationen, Projektion, Kart. Produkt

33/39

- Schema-Operationen: $\mathcal{A} \cap \mathcal{B}$, $\mathcal{A} \cup \mathcal{B}$, $\mathcal{A} \setminus \mathcal{B}$ wie angegeben als DB-Operationen definiert

Unterschema $\mathcal{B} \subseteq \mathcal{A}$

- $\mathcal{B} = \{a_{i_1}:D_{i_1}, \dots, a_{i_k}:D_{i_k}\} \subseteq \{a_1:D_1, \dots, a_n:D_n\} = \mathcal{A}$, $k \leq n$

Projektion $P_{\mathcal{B}}: \rho(\mathcal{A}) \rightarrow \rho(\mathcal{B})$

- $\rho(\mathcal{A})$ als Restriktion auf Schema \mathcal{B} , für das $\mathcal{B} \subseteq \mathcal{A}$ gilt, definiert
- $\rho(\mathcal{B}) = \{(a_{i_1}:v_{i_1}, \dots, a_{i_k}:v_{i_k}) \mid \text{es gibt } (a_1:v_1, \dots, a_n:v_n) \in \rho(\mathcal{A}) \text{ so, daß } (a_{i_1}:v_{i_1}, \dots, a_{i_k}:v_{i_k}) \subseteq (a_1:v_1, \dots, a_n:v_n)\}$

Kartesisches Produkt $\rho \times \sigma$ zweier Relationen $\rho = \rho(\mathcal{A})$, $\sigma = \sigma(\mathcal{B})$

- Jedes n-Tupel aus ρ wird mit jedem m-Tupel aus σ kombiniert
- $\rho \times \sigma = \{(v_1, \dots, v_n, v_{n+1}, \dots, v_{n+m}) \mid (v_1, \dots, v_n) \in \rho, (v_{n+1}, \dots, v_{n+m}) \in \sigma\}$



2.6 Selektion

34/39

Selektion $\text{sel}_{\sigma}(\rho)$ zweier Relationen $\rho(\mathcal{A})$, $\sigma(\mathcal{B})$ mit $\mathcal{B} \subseteq \mathcal{A}$

- $\text{sel}_{\sigma}(\rho) = \{t \in \rho \mid P_{\mathcal{B}}(t) \in \sigma\}$
- Relation zum Schema \mathcal{A} , die nur solche Tupel enthält, deren Projektion auf $\sigma(\mathcal{B})$ auch in σ vorkommt
- Spezialfall:
 - Statt expliziter Angabe von σ werden Bedingungen formuliert, die zwischen bestimmten Feldern der Relation ρ gelten müssen
 - Beispiel: $\text{sel}_{a_3 = v_3}(\rho)$ bezeichnet die Menge aller Tupel mit $v_3 = v_3$

Selektion über drei Relationen $\rho(\mathcal{A})$, $\sigma(\mathcal{B})$, $\tau(\mathcal{C})$ mit $\mathcal{B}, \mathcal{C} \subseteq \mathcal{A}$

- $\text{sel}_{\sigma}(\text{sel}_{\tau}(\rho)) = \{t \in \rho \mid P_{\mathcal{B}}(t) \in \sigma \text{ und } P_{\mathcal{C}}(t) \in \tau\}$
- Kommutativität: $\text{sel}_{\sigma}(\text{sel}_{\tau}(\rho)) = \text{sel}_{\tau}(\text{sel}_{\sigma}(\rho))$



2.6 Verbund

35/39

Verbund (engl. join):

Wichtigste auf 2 Relationen $\rho = \rho(\mathcal{A})$, $\sigma = \sigma(\mathcal{B})$ anwendbare Operation

natürlicher Verbund $\rho \bowtie_{\varphi} \sigma$:

- φ ist ein Unterschema zu \mathcal{A} und \mathcal{B} ($\varphi \subseteq \mathcal{A}$, $\varphi \subseteq \mathcal{B}$)
- Idee: nimm alle Tupel aus $\rho \times \sigma$, die in φ übereinstimmen
- $\rho \bowtie_{\varphi} \sigma = \{z \mid z \in P_{\text{nat}}(\rho \times \sigma), P_{\mathcal{A}}(z) \in \rho, P_{\mathcal{B}}(z) \in \sigma, P_{\varphi}(P_{\mathcal{A}}(z)) = P_{\varphi}(P_{\mathcal{B}}(z))\}$
- Eigenschaften: kommutativ, assoziativ, idempotent

Θ -Verbund $\rho[X \Theta Y]\sigma$ (engl. theta-join):

- ρ und σ sind Relationen
- X aus ρ und Y aus σ sind Attribute über demselben Wertebereich
- Θ -Verbund sind die Tupel aus $\rho \times \sigma$, die Bedingung $X \Theta Y$ erfüllen:
 $\rho[X \Theta Y]\sigma = \text{sel}_{X \Theta Y}(\rho \times \sigma)$



2.6 Datenbankoperationen: Division

36/39

Division ρ / σ :

- $\rho = \rho(A)$, $\sigma = \sigma(B)$, $B \subseteq A$ und $C = A \setminus B$

Forderung: Es kann nur eine Relation eines Schema durch eine Relation eines Unterschemas geteilt werden

- $\rho / \sigma = P_C(\rho) \setminus P_C((P_C(\rho) \times \sigma) \setminus \rho)$

Klammerausdruck: alle Tupel t mit der Eigenschaft

- $t \in \rho$, $P_B(t) \in \sigma$
- Ergebnis der Division: alle Tupel $P_C(t)$ mit $t \in \rho$, $P_B(t) \in \sigma$

- Gleichwertige Definition:

$$\rho / \sigma = P_C(\rho) \setminus P_C((P_C(\rho) \bowtie \sigma) \setminus \rho)$$



2.6 Realisierung der Datenbankoperationen in SQL

37/39

- SQL setzt sich aus zwei Teilen zusammen:

- **Datendefinitionssprache:** Sprachelemente zur Definition von Datenbankschemata
- **Datenmanipulationssprache:** Sprachelemente zur Manipulation von Datenobjekten

- Wertebereiche D_i zur Definition eines relationalen Schemas

- **CHARACTER** ein einzelnes Zeichen
- **CHARACTER(n)** Zeichenkette der Länge n
- **INTEGER** ganze Zahl mit Vorzeichen
- **FLOAT** Gleitkommazahl

- Sprachelement zur Definition eines Schemas

- **CREATE TABLE** *Relationenname* ($a_1 D_1, \dots, a_n D_n$)



2.6 Operationen in SQL

38/39

INSERT-Operation

- Eintragen einer Relation bei vorgegebenem Schema
- INSERT INTO Name(a_1, \dots, a_n) VALUES (v_1, \dots, v_n)
- Typ der v_i muß mit den Wertebereichen der a_i übereinstimmen

UPDATE- und DELETE-Operation

- Ändern und Löschen von Tupeln einer Relation

SELECT-Operation

- Lesen bzw. Verknüpfen von Tupeln einer Relation
- SELECT Attribute FROM Relationen WHERE Bedingungen



2.6 Beispiel zur SELECT-Operation

39/39

Beispiel: für Datenbankrelationen

Studierender:

| Name | Vorname | Matr_Nr | Semester |
|--------|---------|---------|----------|
| Meier | Hans | 123456 | 1 |
| Müller | Petra | 654321 | 3 |
| Bauer | Klaus | 232323 | 1 |
| Schmid | Astrid | 865921 | 5 |
| Bähr | Claudia | 987654 | 3 |

Fächerbelegung:

| Fach | Name |
|------|--------|
| Inf | Meier |
| Bio | Müller |
| Med | Bauer |
| Inf | Schmid |
| Med | Bähr |

Beispiel von SELECT-Operationen auf obigen Datenbankrelationen:

1. SELECT Name FROM Studierender
2. SELECT Fb.Fach, St.Semester FROM Fächerbelegung Fb,
Studierender St WHERE Fb.Name = St.Name