Abschluss:

- Jenseits der hier besprochenen Sprachkonzepte gibt es in Ocaml einige weitere Konzepte, die insbesondere objekt-orientierte Programmierung ermöglichen.
- Darüberhinaus bietet Ocaml elegante Möglichkeiten, Betriebssystemsfunktionalität auszunutzen, graphische Bibliotheken anzusteuern, mit anderen Rechnern zu kommunizieren ...

$\Longrightarrow$ Ocaml ist eine interessante Alternative zu Java.

8 Datalog: Rechnen mit Relationen

Beispiel 1: Das Lehrangebot einer TU

$\Longrightarrow$ Entity-Relationship Diagram

Diskussion:

- Viele Anwendungsbereiche lassen sich mit Hilfe von Entity-Relationship-Diagrammen beschreiben.
- Entitäten im Beispiel: Dozent, Vorlesung, Student.
- Die Menge aller vorkommenden Entitäten d.h. Instanzen lassen sich mit einer Tabelle beschreiben ...

Dozent :

Name	Telefon	Email
Esparza	17204	`esparza@in.tum.de`
Nipkow	17302	`nipkow@in.tum.de`
Seidl	18155	`seidl@in.tum.de`

Vorlesung:

Titel	Raum	Zeit
Diskrete Strukturen	MI 1	Di 13:45-15:15, Do 10-11:30
Perlen der Informatik III	MI 3	Do 8:30-10
Einführung in die Informatik II	MI 1	Di 15:30-17:00
Optimierung	02.07.014	Mo 12:15-13:45, Di 12:15-13:45

Student:

Matr.nr.	Name	Sem.
123456	Hans Dampf	03
007042	Fritz Schluri	11
543345	Anna Blume	03
131175	Effi Briest	05

Diskussion (Forts.):

- Die Zeilen entsprechen den Instanzen.
- Die Spalten entsprechen den Attributen.
- Annahme: das erste Attribut identifiziert die Instanz

  $\Longrightarrow$ Primärschlüssel

  Folgerung: Beziehungen sind ebenfalls Tabellen ...

liest:

Name	Titel
Esparza	Diskrete Strukturen
Nipkow	Perlen der Informatik III
Seidl	Einführung in die Informatik II
Seidl	Optimierung

hört:

Matr.nr.	Titel
123456	Einführung in die Informatik II
123456	Optimierung
123456	Diskrete Strukturen
543345	Einführung in die Informatik II
543345	Diskrete Strukturen
131175	Optimierung

Mögliche Anfragen:

- In welchen Semestern sind die Studierenden der Vorlesung "Diskrete Strukturen" ?
- Wer hört eine Vorlesung bei Dozent "Seidl" ?
- Wer hört sowohl "Diskrete Strukturen" wie "Einführung in die Informatik II" ?

$\Longrightarrow$ Datalog

Idee: Tabelle $\Longleftrightarrow$ Relation

Eine Relation R ist eine Menge von Tupeln, d.h.

$$R \subseteq \mathcal{U}_1 \times \ldots \times \mathcal{U}_n$$

wobei $\mathcal{U}_i$ die Menge aller möglicher Werte für die i-te Komponente ist. In unserem Beispiel kommen etwa vor:

`int`, `string`, möglicherweise Aufzähldatentypen

// Einstellige Relationen sind Mengen :-)

Relationen können durch Prädikate beschrieben werden ...

Prädikate können wir definieren durch Aufzählung von Fakten ...

... im Beispiel:

```
liest ("Esparza", "Diskrete Strukturen").
liest ("Nipkow", "Perlen der Informatik III").
liest ("Seidl", "Einführung in die Informatik II").
liest ("Seidl", "Optimierung").

hört (123456, "Optimierung").
hört (123456, "Einführung in die Informatik II").
hört (123456, "Diskrete Strukturen").
hört (543345, "Einführung in die Informatik II").
hört (543345, "Diskrete Strukturen").
hört (131175, "Optimierung").
```

Wir können aber auch Regeln benutzen, mit denen weitere Fakten abgeleitet werden können ...

... im Beispiel:

```
hat_Hörer (X,Y) :- liest (X,Z), hört (M,Z), student (M,Y,_).
semester (X,Y) :- hört (Z,X), student (Z,_,Y).
```

- `:-` bezeichnet die logische Implikation "$\Leftarrow$".
- Die komma-separierte Liste sammelt die Voraussetzungen.
- Die linke Seite, der Kopf der Regel, ist die Schlussfolgerung.
- Die Variablen werden groß geschrieben.
- Die anonyme Variable `_` bezeichnet irrelevante Werte :-)

An die Wissensbasis aus Fakten und Regeln können wir jetzt Anfragen stellen ...

... im Beispiel:

```
?- hat_Hörer ("Seidl", Z).
```

- Datalog findet alle Werte für `Z`, für die die Anfrage aus den gegebenen Fakten mit Hilfe der Regeln beweisbar ist :-)
- In unserem Beispiel ist das:

```
Z = "Hans Dampf"
Z = "Anna Blume"
Z = "Effi Briest"
```

Weitere Anfragen:

```
?- semester ("Diskrete Strukturen", X).
   X = 3
   X = 5

?- hört (X, "Einführung in die Informatik II"),
   hört (X, "Diskrete Strukturen").
   X = 123456
   X = 543345
```

Weitere Anfragen:

```
?- semester ("Diskrete Strukturen", X).
   X = 3
   X = 5

?- hört (X, "Einführung in die Informatik II"),
   hört (X, "Diskrete Strukturen").
   X = 123456
   X = 543345
```

Achtung:

Natürlich kann die Anfrage auch gar keine oder mehr als eine Variable enthalten :-)

Ein Beispiel-Beweis:

Die Regel:

```
hat_Hörer (X,Y) :- liest (X,Z), hört (M,Z), student (M,Y,_).
```

gilt für alle `X`, `M`, `Y`, `Z`.

Ein Beispiel-Beweis:

Die Regel:

```
hat_Hörer (X,Y) :- liest (X,Z), hört (M,Z), student (M,Y,_).
```

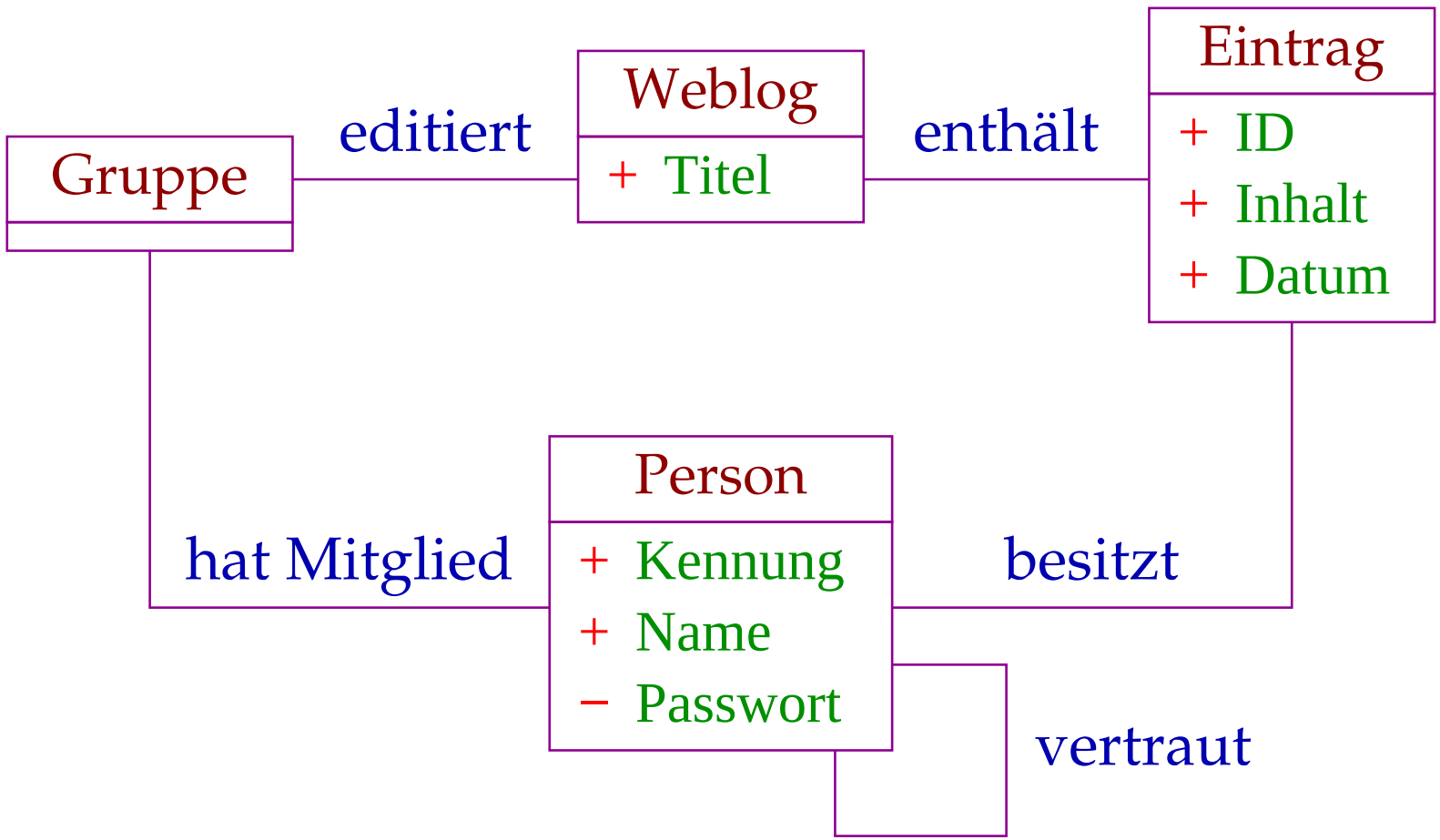
gilt für alle `X`, `M`, `Y`, `Z`. Mit Hilfe der Substitution:

`"Seidl"/X` `"Einführung ..."/Z` `543345/M` `"Anna Blume"/Y`

können wir schließen:

```
liest ("Seidl", "Einführung ...")
                hört (543345, "Einführung ....")
                student (543345, "Anna Blume", 3)
-------------------------------------------------
          hat_Hörer ("Seidl", "Anna Blume")
```

Beispiel 2: Ein Weblog



Aufgabe: Festlegung der Zugriffsberechtigung

- Jedes Mitglied der editierenden Gruppe darf einen neuen Eintrag hinzufügen.
- Nur die Besitzerin eines Eintrags darf ihn löschen.
- Modifizieren darf ihn jeder, dem die Besitzerin traut.
- Lesen darf ihn jedes Mitglied der Gruppe und jeder ihrer mittelbar Vertrauten ...

Spezifikation in Datalog:

```
darf_hinzufügen (X,W) :- editiert (Z,W),
                         hat_Mitglied (Z,X).
darf_löschen (X,E) :- besitzt (X,E).
darf_modifizieren (X,E) :- besitzt (X,E).
darf_modifizieren (X,E) :- besitzt (Y,E),
                           vertraut (Y,X).
darf_lesen (X,E) :- enthält (W,E),
                    darf_hinzufügen (X,W).
darf_lesen (X,E) :- darf_lesen (Y,E),
                    vertraut (Y,X).
```

Beachte:

- Zur Definition neuer Prädikate dürfen wir selbstverständlich alle vorhandenen benutzen oder sogar Hilfsprädikate definieren.
- Offenbar können Prädikatsdefinitionen auch rekursiv sein :-)
- Mit einer Person `X`, die einen Eintrag besitzt, dürfen auch alle Personen modifizieren, denen `X` traut.
- Mit einer Person `Y`, die einen Eintrag lesen darf, dürfen auch alle Personen lesen, denen `Y` traut :-))

8.1 Beantwortung von Anfragen

Gegeben: eine Menge von Fakten und Regeln.

Gesucht: die Menge aller ableitbaren Fakten.

Problem:

```
equals (X,X).
```

$\Longrightarrow$ Die Menge aller ableitbaren Fakten ist nicht endlich :-(

Satz:

Sei W eine endliche Menge von Fakten und Regeln mit den folgenden Eigenschaften:

(1) Fakten enthalten keine Variablen.

(2) Jede Variable im Kopf kommt auch im Rumpf vor.

Dann ist die Menge der ableitbaren Fakten endlich.

Satz:

Sei W eine endliche Menge von Fakten und Regeln mit den folgenden Eigenschaften:

(1) Fakten enthalten keine Variablen.

(2) Jede Variable im Kopf kommt auch im Rumpf vor.

Dann ist die Menge der ableitbaren Fakten endlich.

Beweisskizze:

Man zeigt für jedes beweisbare Faktum `p(a1,...,ak)`, dass jede Konstante `ai` bereits in W vorkommt :-))

Berechnung aller ableitbaren Fakten:

Berechne sukzessiv Mengen $R^{(i)}$ der Fakten, die mithilfe von Beweisen der Tiefe maximal i abgeleitet werden können ...

$$R^{(0)} = \emptyset \qquad\qquad R^{(i+1)} = \mathcal{F}(R^{(i)})$$

wobei der Operator $\mathcal{F}$ definiert ist durch:

$$\mathcal{F}(M) = \{h[\underline{a}/\underline{X}] \mid \exists\, h :\!- l_1, \ldots, l_k. \in W : l_1[\underline{a}/\underline{X}], \ldots, l_k[\underline{a}/\underline{X}] \in M\}$$

// $[\underline{a}/\underline{X}]$ eine Substitution der Variablen $\underline{X}$

// k kann auch 0 sein :-)

Es gilt: $R^{(i)} = \mathcal{F}^i(\emptyset) \subseteq \mathcal{F}^{i+1}(\emptyset) = R^{(i+1)}$

Es gilt: $R^{(i)} = \mathcal{F}^i(\emptyset) \subseteq \mathcal{F}^{i+1}(\emptyset) = R^{(i+1)}$

Die Menge R aller implizierten Fakten ist gegeben durch:

$$R = \bigcup_{i \geq 0} R^{(i)} = R^{(n)}$$

für ein geeignetes n — da R endlich ist :-)

Es gilt: $R^{(i)} = \mathcal{F}^i(\emptyset) \subseteq \mathcal{F}^{i+1}(\emptyset) = R^{(i+1)}$

Die Menge R aller implizierten Fakten ist gegeben durch:

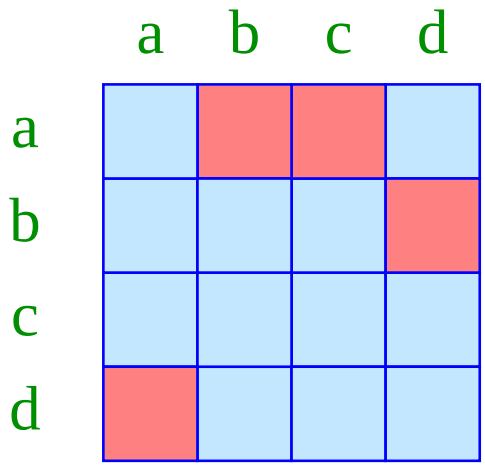
$$R = \bigcup_{i \geq 0} R^{(i)} = R^{(n)}$$

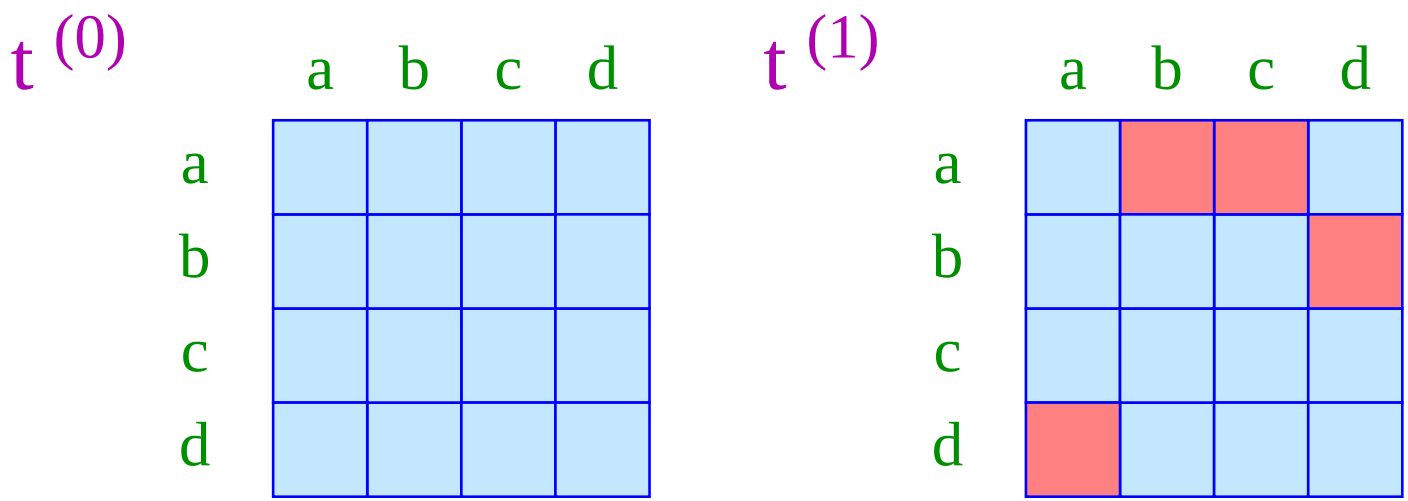
für ein geeignetes n — da R endlich ist :-)

Beispiel:

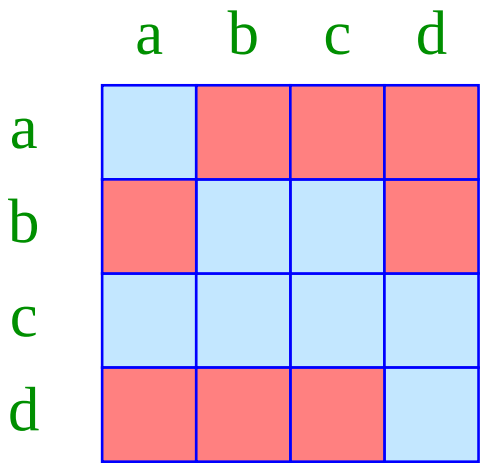
```
edge (a,b).
edge (a,c).
edge (b,d).
edge (d,a).
t (X,Y) :- edge (X,Y).
t (X,Y) :- edge (X,Z), t (Z,Y).
```

Relation edge :

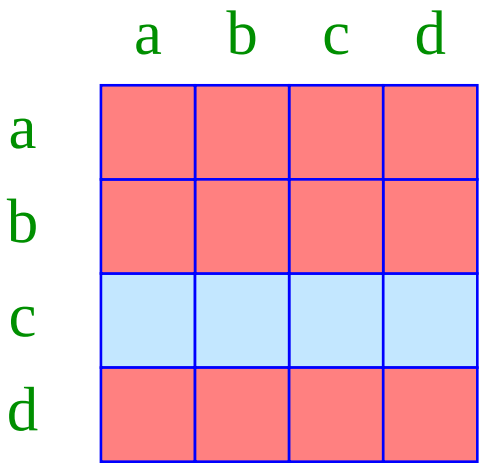



t (0)
a b c d
a
b
c
d
t (1)
a b c d
a
b
c
d

$t^{(2)}$



$t^{(3)}$



Diskussion:

- Unsere Überlegungen reichen aus, um für ein Datalog-Programm die Menge aller implizierten Fakten zu berechnen :-)
- Aus diesen können wir die Antwort-Substitutionen für die Anfrage ablesen :-))

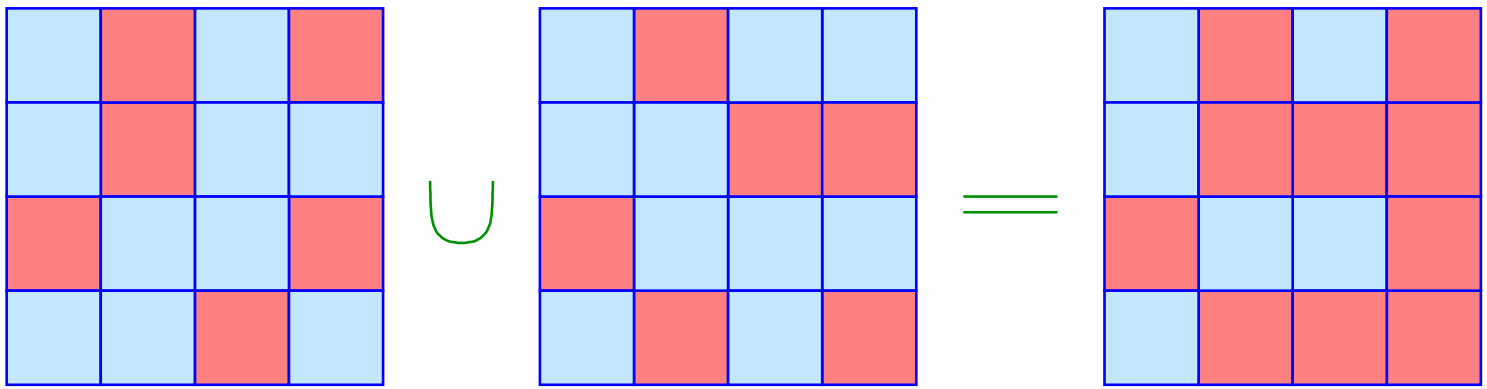
Diskussion:

- Unsere Überlegungen reichen aus, um für ein Datalog-Programm die Menge aller implizierten Fakten zu berechnen :-)
- Aus diesen können wir die Antwort-Substitutionen für die Anfrage ablesen :-))
- Die naive Vorgehensweise ist allerdings hoffnungslos ineffizient :-(
- Intelligentere Verfahren versuchen, Mehrfachberechnungen immer der gleichen Fakten zu vermeiden ...
- Insbesondere braucht man ja auch nur solche Fakten abzuleiten, die zur Beantwortung der Anfrage nützlich sind
  $\Longrightarrow$ Compilerbau, Datenbanken

8.2 Operationen auf Relationen

- Wir benutzen Prädikate, um Relationen zu beschreiben.
- Auf Relationen gibt es natürliche Operationen, die wir gerne in Datalog, d.h. für Prädikate definieren möchten :-)

1. Vereinigung:



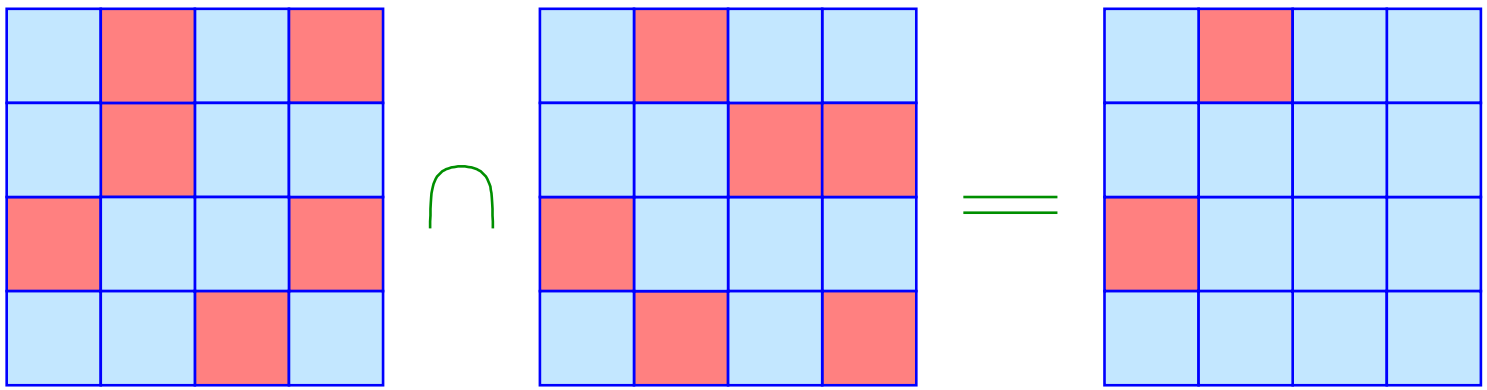
... in Datalog:

$$r(X_1, \ldots, X_k) \quad \texttt{:-} \quad s_1(X_1, \ldots, X_k).$$
$$r(X_1, \ldots, X_k) \quad \texttt{:-} \quad s_2(X_1, \ldots, X_k).$$

Beispiel:

```
hört_Esparza_oder_Seidl (X) :- hat_Hörer ("Esparza", X).
hört_Esparza_oder_Seidl (X) :- hat_Hörer ("Seidl", X).
```

2. Durchschnitt:



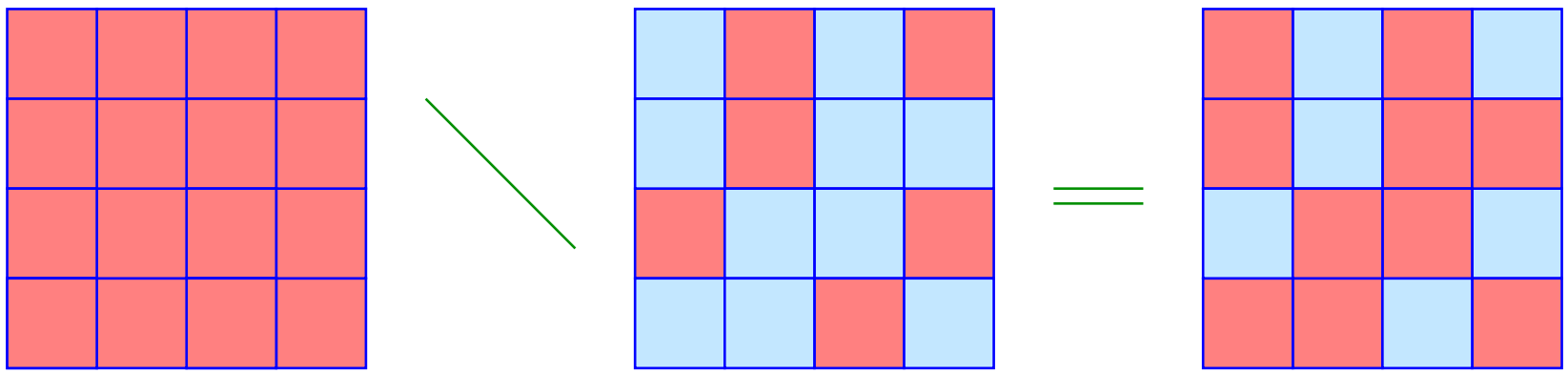
... in Datalog:

$$r(X_1, \ldots, X_k) \quad \texttt{:-} \quad s_1(X_1, \ldots, X_k),$$
$$s_2(X_1, \ldots, X_k).$$

Beispiel:

```
hört_Esparza_und_Seidl (X) :- hat_Hörer ("Esparza", X),
                              hat_Hörer ("Seidl", X).
```

3. Relatives Komplement:



... in Datalog:

$$r(X_1, \ldots, X_k) \quad \texttt{:-} \quad s_1(X_1, \ldots, X_k), \ \texttt{not}(s_2(X_1, \ldots, X_k)).$$

d.h., $r(a_1, \ldots, a_k)$ folgt, wenn sich $s_1(a_1, \ldots, a_k)$, aber nicht $s_2(a_1, \ldots, a_k)$ beweisen lässt :-)

Beispiel:

```
hört_nicht_Seidl (X) :- student (_,X,_),
                        not (hat_Hörer ("Seidl", X)).
```

Achtung:

Die Anfrage:

```
p("Hallo!").
?- not (p(X)).
```

führt zu unendlich vielen Antworten :-(

$\Longrightarrow$ wir erlauben negierte Literale nur, wenn links davon alle Variablen in nicht-negierten Literalen vorkommen :-)

```
p("Hallo!").
q("Damn ...").
?- q(X), not (p(X)).
   X = "Damn ..."
```

Achtung (Forts.):

Negation ist nur sinnvoll, wenn s nicht rekursiv von r abhängt ...

```
p(X) :- not (p(X)).
```

... ist nicht leicht zu interpretieren.

$\Longrightarrow$ Wir erlauben $\texttt{not}(s(\ldots))$ nur in Regeln für Prädikate r, von denen s nicht abhängt

$\Longrightarrow$ stratifizierte Negation

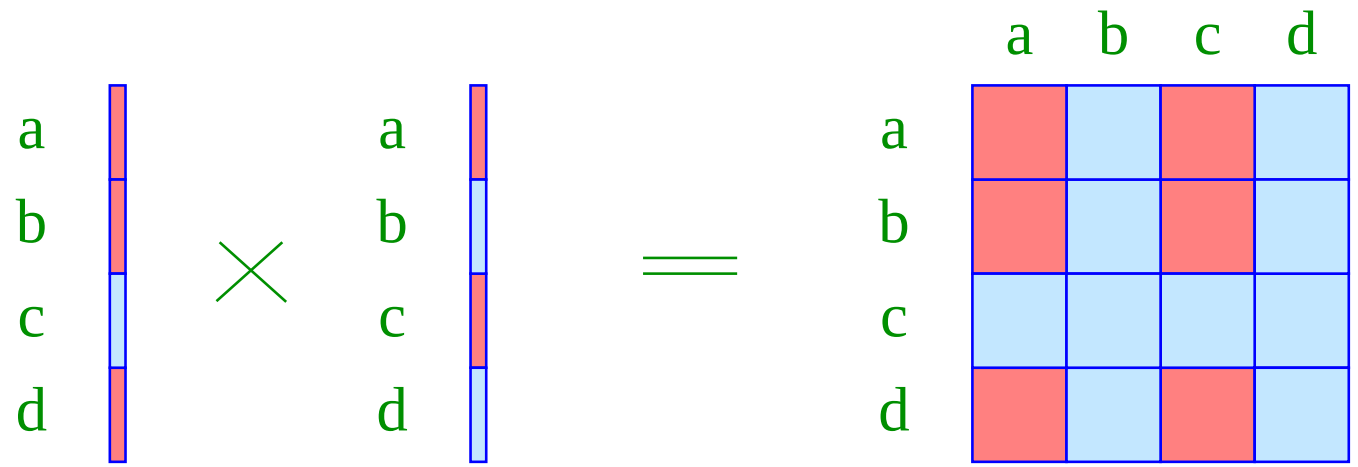
// Ohne rekursive Prädikate ist jede Negation stratifiziert :-)

4. Cartesisches Produkt:

$$S_1 \times S_2 = \{(a_1, \ldots, a_k, b_1, \ldots, b_m) \mid \begin{array}{l} (a_1, \ldots, a_k) \in S_1, \\ (b_1, \ldots, b_m) \in S_2 \end{array} \}$$

... in Datalog:

$$r(X_1, \ldots, X_k, Y_1, \ldots, Y_m) \quad :- \quad s_1(X_1, \ldots, X_k), s_2(Y_1, \ldots, Y_m).$$


a
b
c
d
a
b
c
d
a b c d
a
b
c
d

Beispiel:

```
dozent_student (X,Y) :- dozent (X,_,_),
                        student (_,Y,_).
```

Bemerkung:

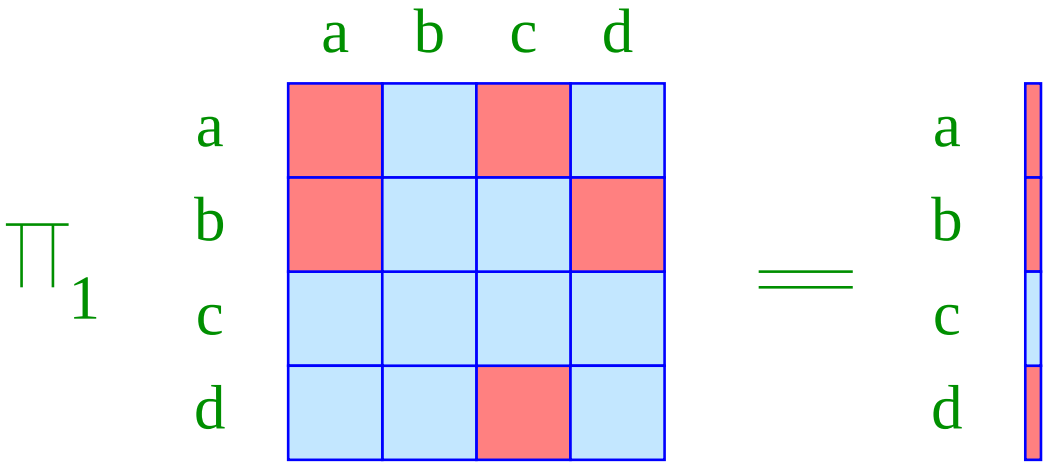
- Das Produkt unabhängiger Relationen ist sehr teuer :-(
- Man sollte es nach Möglichkeit vermeiden ;-)

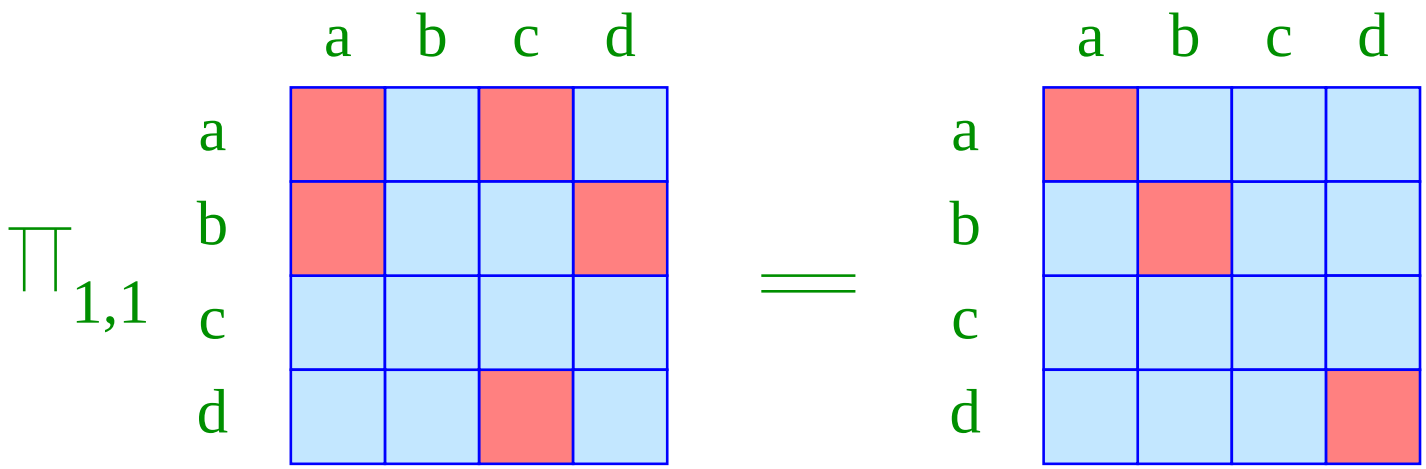
5. Projektion:

$$\pi_{i_1,\ldots,i_k}(S) = \{(a_{i_1},\ldots,a_{i_k}) \mid (a_1,\ldots,a_m) \in S\}$$

... in Datalog:

$$r(X_{i_1},\ldots,X_{i_k}) \quad :- \quad s(X_1,\ldots,X_m).$$


a b c d
a
b
c
d
$\prod_1$
=
a
b
c
d


a b c d
a
b
c
d
$\prod_{1,1}$
=
a b c d
a
b
c
d

6. Join:

$$S_1 \bowtie S_2 = \{(a_1, \ldots, a_k, b_1, \ldots, b_m) \mid \begin{array}{l} (a_1, \ldots, a_{k+1}) \in S_1, \\ (b_1, \ldots, b_m) \in S_2, \\ a_{k+1} = b_1 \end{array} \}$$

... in Datalog:

$$r(X_1, \ldots, X_k, Y_1, \ldots, Y_m) \quad :- \quad s_1(X_1, \ldots, X_k, Y_1), s_2(Y_1, \ldots, Y_m).$$

Diskussion:

Joins können durch die anderen Operationen definiert werden ...

$$
\begin{aligned}
S_1 \bowtie S_2 \quad = \quad & \pi_{1,\ldots,k,k+2,\ldots,k+1+m} \; (\\
& S_1 \times S_2 \; \cap \\
& \mathcal{U}^k \times \pi_{1,1}(\mathcal{U}) \times \mathcal{U}^{m-1})
\end{aligned}
$$

// Zur Vereinfachung haben wir angenommen, $\mathcal{U}$ sei das
// gemeinsame Universum aller Komponenten :-)

Joins erlauben oft, teure cartesische Produkte zu vermeiden :-)

Die vorgestellten Operationen auf Relationen bilden die Grundlage der relationalen Algebra ...

Hintergrund:

Relationale Algebra ...

+ ist die Basis für Anfragesprachen relationaler Datenbanken

  $\Longrightarrow$ SQL

+ erlaubt Optimierung von Anfragen.

  Idee: Ersetze aufwändig zu berechnende Teilausdrücke der Anfrage durch billigere mit der gleichen Semantik !

Hintergrund:

Relationale Algebra ...

\+ ist die Basis für Anfragesprachen relationaler Datenbanken

$$\Longrightarrow \quad \text{SQL}$$

\+ erlaubt Optimierung von Anfragen.

Idee: Ersetze aufwändig zu berechnende Teilausdrücke der Anfrage durch billigere mit der gleichen Semantik !

− ist ziemlich kryptisch.

− erlaubt keine rekursiven Definitionen.

Beispiel:

Das Datalog-Prädikat:

```
semester (X,Y) :- hört (Z,X), student (Z,_,Y)
```

... lässt sich in SQL so ausdrücken:

```
SELECT hört.Titel, Student.Semester
FROM   hört, Student
WHERE  hört.Matrikelnummer = Student.Matrikelnummer
```

Ausblick:

- Außer einer Anfragesprache muss eine praktische Datenbank-Sprache auch die Möglichkeit zum Einfügen / Modifizieren / Löschen anbieten :-)
- Die Implementierung einer Datenbank muss nicht nur Spielanwendungen wie unsere Beispiele bewältigen, sondern mit gigantischen Datenvolumen umgehen können !!!
- Sie muss viele parallel ablaufende Transaktionen zuverlässig abwickeln, ohne sie durcheinander zu bringen.
- Eine Datenbank sollte auch einen Stromausfall überstehen

$\Longrightarrow$ Datenbank-Vorlesung