

Berechnung aller ableitbaren Fakten:

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

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

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

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

// $[\underline{a}/\underline{X}]$ 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* :

	a	b	c	d
a				
b				
c				
d				

The table shows a 4x4 grid with columns labeled 'a', 'b', 'c', 'd' and rows labeled 'a', 'b', 'c', 'd'. The cells (a,b), (a,c), (b,d), and (d,a) are shaded red, indicating the presence of the relation 'edge'. All other cells are light blue, indicating the absence of the relation.

$t^{(0)}$

	a	b	c	d
a				
b				
c				
d				

$t^{(1)}$

	a	b	c	d
a				
b				
c				
d				

$t^{(2)}$

	a	b	c	d
a	light blue	red	red	red
b	red	light blue	light blue	red
c	light blue	light blue	light blue	light blue
d	red	red	red	light blue

$t^{(3)}$

	a	b	c	d
a	red	red	red	red
b	red	red	red	red
c	light blue	light blue	light blue	light blue
d	red	red	red	red

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 :-))

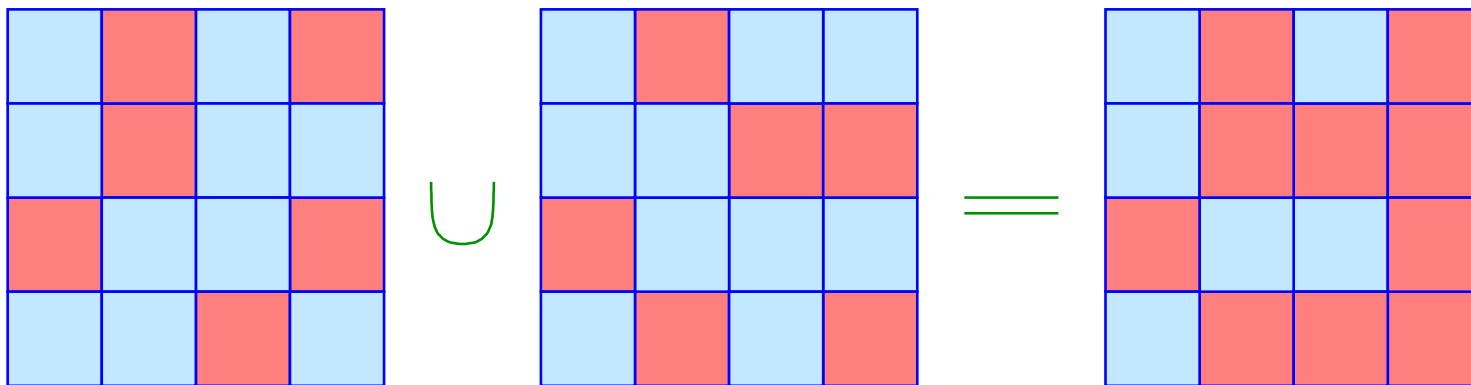
Diskussion:

- 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
⇒ **Compilerbau, Datenbanken**

12.2 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:

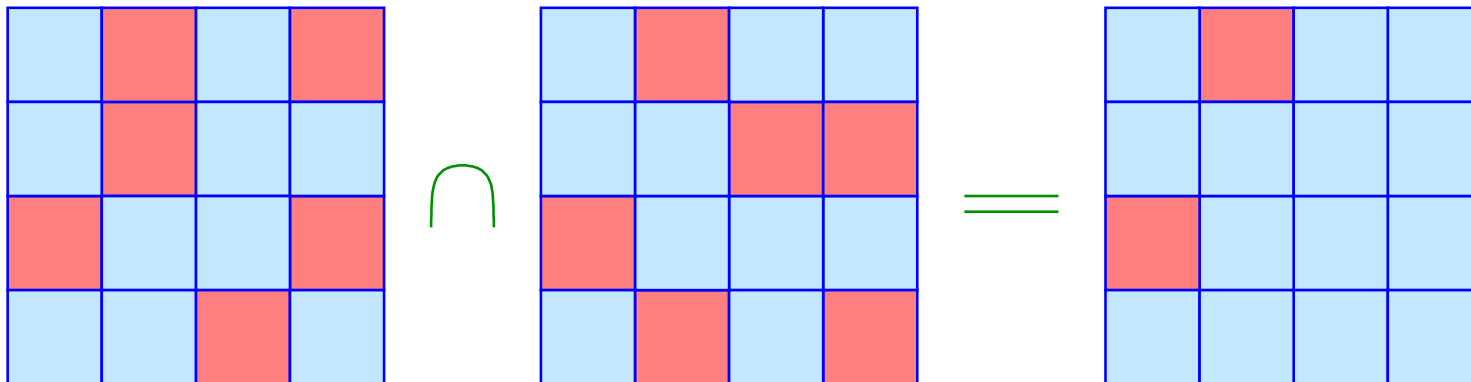
$$r(X_1, \dots, X_k) \quad :- \quad s_1(X_1, \dots, X_k).$$
$$r(X_1, \dots, X_k) \quad :- \quad s_2(X_1, \dots, X_k).$$

Beispiel:

```
hört_Brauer_oder_Seidl (X) :- hat_Hörer ("Brauer", X).
```

```
hört_Brauer_oder_Seidl (X) :- hat_Hörer ("Seidl", X).
```

2. Durchschnitt:



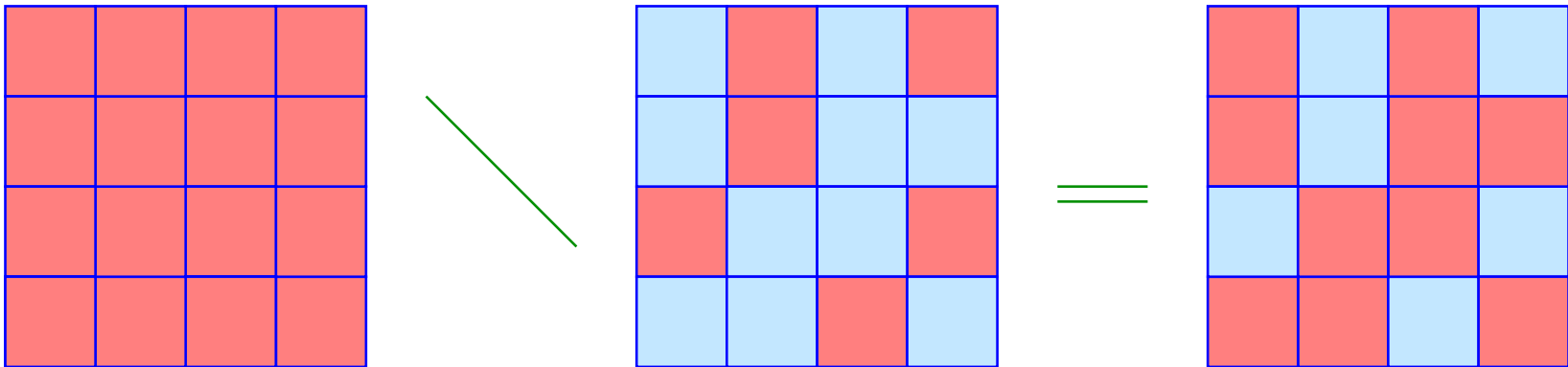
... in Datalog:

$$r(X_1, \dots, X_k) \quad :- \quad s_1(X_1, \dots, X_k), \\ s_2(X_1, \dots, X_k).$$

Beispiel:

```
hört_Brauer_und_Seidl (X) :- hat_Hörer ("Brauer", X),  
                             hat_Hörer ("Seidl", X).
```

3. Relatives Komplement:



... in Datalog:

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

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

⇒ 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 ...

$$p(X) \text{ :- not } (p(X)).$$

... ist **nicht leicht** zu interpretieren.

⇒ Wir erlauben $\text{not}(s(\dots))$ nur in Regeln für Prädikate r , von denen s nicht abhängt

⇒ **stratifizierte Negation**

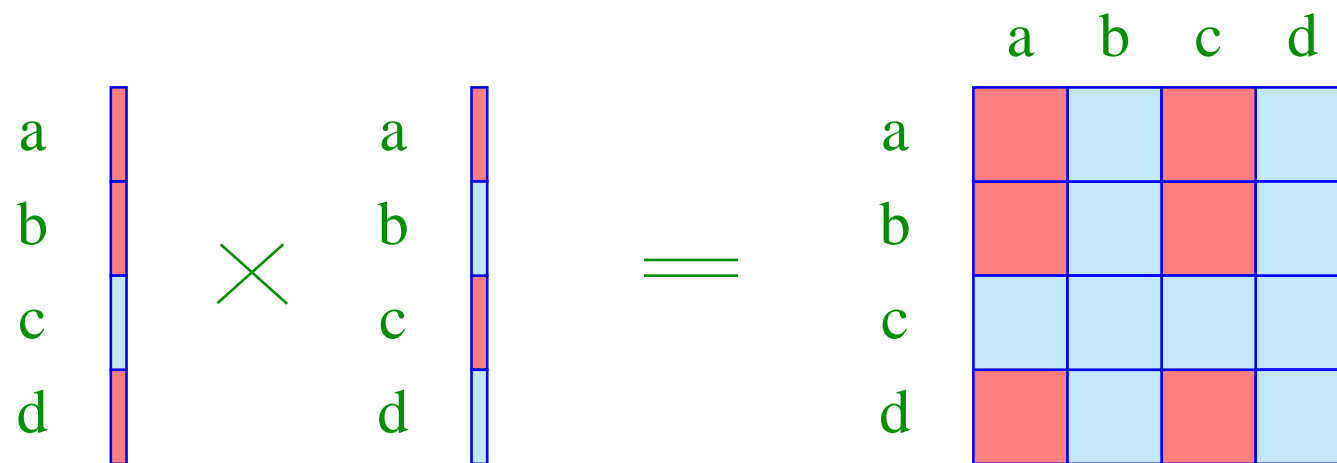
// Ohne rekursive Prädikate ist jede Negation stratifiziert :-)

4. Cartesisches Produkt:

$$S_1 \times S_2 = \{(a_1, \dots, a_k, b_1, \dots, b_m) \mid (a_1, \dots, a_k) \in S_1, \\ (b_1, \dots, b_m) \in S_2 \}$$

... in Datalog:

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



Beispiel:

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

Bemerkung:

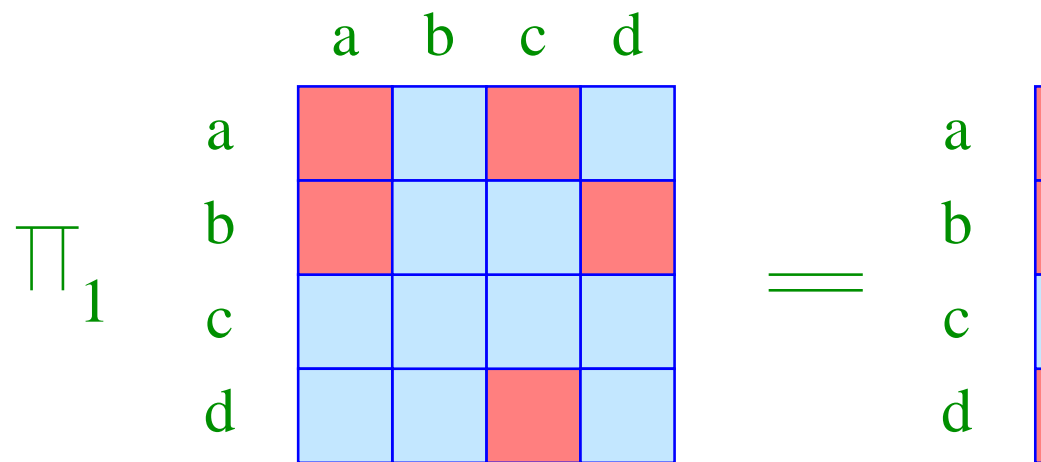
- Das Produkt unabhängiger Relationen ist sehr **teuer** :-)
- Man sollte es nach Möglichkeit **vermeiden** ;-)

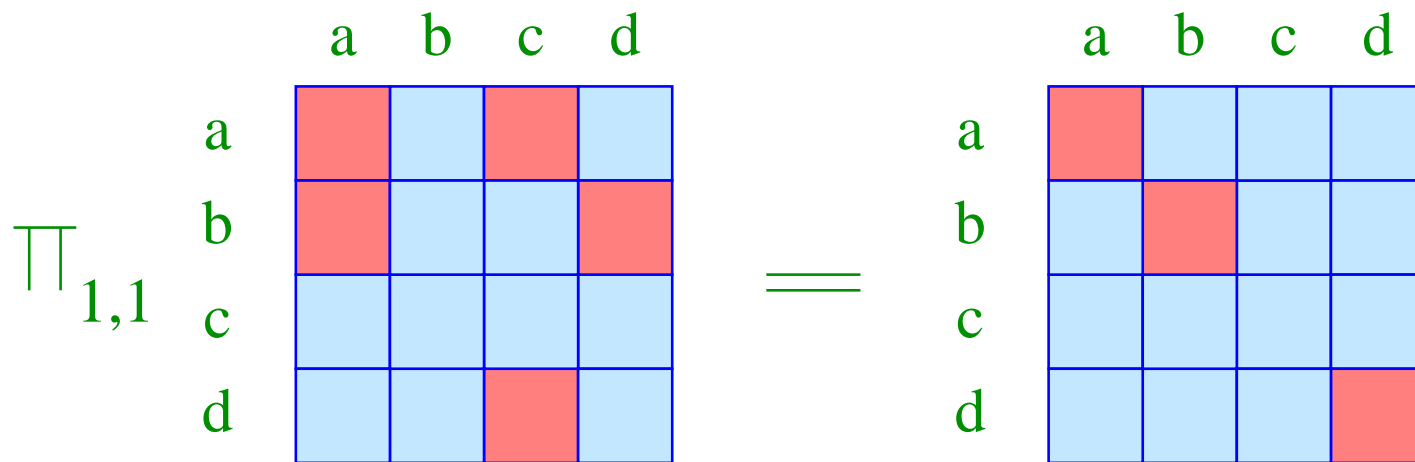
5. Projektion:

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

... in Datalog:

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





6. Join:

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

... in Datalog:

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

Diskussion:

Joins können durch die anderen Operationen definiert werden ...

$$S_1 \bowtie S_2 = \pi_{1,\dots,k,k+2,\dots,k+1+m} \left(S_1 \times S_2 \cap \mathcal{U}^k \times \pi_{1,1}(\mathcal{U}) \times \mathcal{U}^{m-1} \right)$$

// 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**
 \implies 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

⇒ Datenbank-Vorlesung