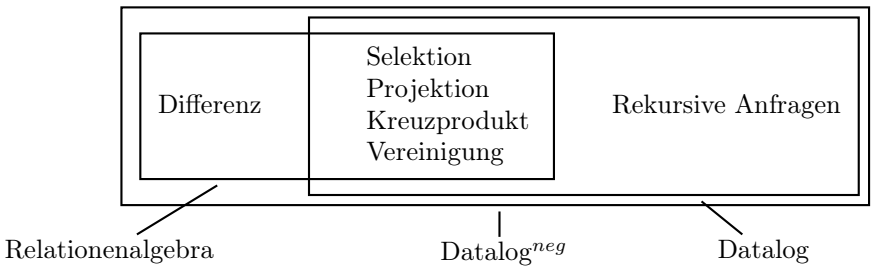
Datenbanktheorie
Sommersemester 2012
Prof. Dr. W. May

4. Übungsblatt: Datalog

Besprechung voraussichtlich am 21./28.6.2012

Aufgabe 1 (Äquivalenz von Algebra und Datalog) Zeigen Sie, das zu jedem Ausdruck der relationalen Algebra ein äquivalentes stratifiziertes Datalog-Programm existiert.

Zusammenhang zwischen Relationaler Algebra, Datalog und Datalog^{neg}



Vereinigung Seien p, q Relationen. Dann ist für $u = p \cup q$

```
u(X1,...,XN) :- p(X1,...,XN).
u(X1,...,XN) :- q(X1,...,XN).
```

Differenz Seien p, q Relationen. Dann ist für $d = p \setminus q$

```
d(X1,...,XN) :- p(X1,...,XN), not q(X1,...,XN).
```

Projektion Sei p eine Relation mit Attributen $X_1, \ldots, X_n$. Dann ist für $pr = \pi[X_{i_1}, \ldots, X_{i_k}](p)$ mit $X_{i_j} \in \{X_1, \ldots, X_n\}$

```
pr(XI1,...,XIK) :- p(X1,...,XN).
```

Selektion Sei p eine Relation mit Attributen $X_1, \ldots, X_n$, α eine Bedingung über $X_1, \ldots, X_n$. Dann ist für $s = \sigma[\alpha](p)$

```
s(X1,...,XN) :- p(X1,...,XN), α.
```

Join Seien p, q Relationen mit gemeinsamen Attributen $X_k, \ldots, X_m$.
Dann ist für $j = p \bowtie q$

```
j(X1,...,XN) :- p(X1,...,XK,...,XM), q(XK,...,XM,...,XN).
```

Das zu einem komplexen Algebra-Ausdruck äquivalente Programm ist stratifiziert, da jeder Teilausdruck ein neues Prädikatssymbol definiert, und somit der Abhängigkeitsgraph der Baumstruktur entspricht.

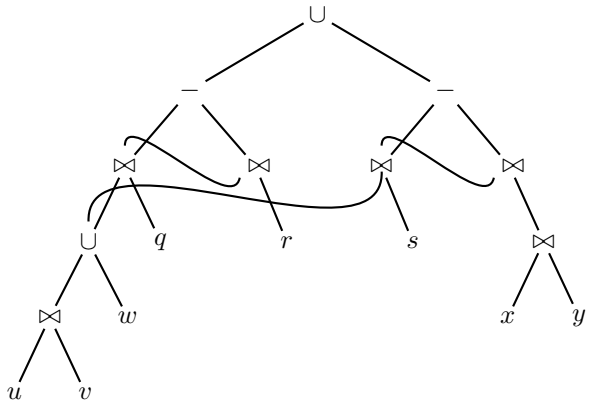
Aufgabe 2 (Datalog to Algebra)

- Betrachten Sie eine Regel der Form $B \leftarrow C_1 \wedge \ldots \wedge C_n \wedge D_{n+1} \wedge D_{n+k}$
  wobei die C_i positive Literale sind, und die D_i negative Literale sind. Geben Sie einen Algebraausdruck an, der die Relation ergibt, die durch die Regel definiert wird.
- Welches weitere Konstrukt muss man noch übersetzen?
- Betrachten Sie das folgende Programm (Stelligkeit der Prädikate beliebig; jede Regel sei sicher):
  res(...) :- *p*(...), *q*(...), ¬ *r*(...).
  res(...) :- *p*(...), *s*(...), ¬ *t*(...).
  p(...) :- *u*(...), *v*(...).
  p(...) :- *w*(...).
  t(...) :- *x*(...), *y*(...).
  wobei q, r, s, u, v, w, x, y EDB-Relationen sind.
  Geben Sie den Algebraausdruck an, der dem Prädikat res entspricht.

- Consider each of the $C_i(X_1, \ldots, X_{m_i})$ and $D_i(X_1, \ldots, X_{m_i})$ of the form $R_i(a_1, \ldots, a_\ell)$, a_j constants or variables. There is an equivalent algebra expression $E_i = \rho[\ldots](\pi[\ldots](R_i))$ with format $(X_1, \ldots, X_{m_i})$ that selects the relevant attributes and renames them to $X_1, \ldots, X_{m_i}$.
  The same holds for the D_i.
  Safety implies that all variables that occur in any of the D_i also occur in at least one of the C_i.
  Let $C := E_i \bowtie \ldots \bowtie E_n$. Then,

$$\pi[free(B)](C - \bigcup_{i=1..k} (C \bowtie E_{n+i}))$$

  is the required expression.
  (note that this is the analogous to the RANF to Algebra transformation.)
- Wenn zwei Regeln dasselbe Prädikatssymbol definieren (d.h. im Regelkopf haben): Vereinigung.
- Subtrees/intermediate results can sometimes be used twice. This is supported by algebraic optimization and tabling.



Aufgabe 3 (Stratified Datalog)

- Geben Sie ein Beispiel für die Nichtmonotonie der stratifizierten Semantik an.
- Zeigen Sie, dass ein stratifizierbares Programm P mehrere minimale Modelle haben kann.

Sei P das Programm

```
borders(Y,X,Z) :- borders(X,Y,Z).    % make it symmetric.
reachable(X,Y) :- borders(X,Y,_).
reachable(X,Y) :- reachable(X,Z), borders(Z,Y,_).
unreachable(X,Y) :- country(X), country(Y), not reachable(X,Y).
```

In reachable sind z.B. die Paare (D,F), (D,SGP), ..., enthalten. In unreachable sind z.B. die Paare (D,USA), (D,BR), ..., (MAL,CY) (Malta, Zypern, beides Inselstaaten) enthalten.

- Wenn man zu mondial noch das Faktum borders(P,USA) hinzunimmt, ist –unter anderem– (D,USA) in reachable enthalten, und entsprechend nicht mehr in unreachable.
- Vorlesung: das stratifizierte Modell $\mathcal{M}$ von Mondial+P ist ein minimales Modell. D.h., es gibt kein Modell, das eine echte Teilmenge von $\mathcal{M}$ ist.
  $\mathcal{M}'' = \mathcal{M} \cup \{\text{reachable(MAL,CY)}\} \setminus (\{\text{reachable(MAL,CY)}\})$, ist ebenfalls ein Modell, und es ist ebenfalls minimal (wieder: es gibt keine echte Teilmenge, die ebenfalls ein Modell ist).
  $\mathcal{M}''$ ist nicht "schön", weil ein "unbegründetes" reachable-Tuple hinzugenommen wurde.
  Im weiteren Verlauf der Vorlesung werden weitere Modellbegriffe betrachtet: $\mathcal{M}''$ ist weder well-founded (d.h., jedes Atom in ihm begründet, noch stabil (d.h. es reproduziert sich nicht)).

Aufgabe 4 (Datalog-Anfragen an Mondial: Schweizer Sprachen) Geben Sie Ausdrücke im relationalen Kalkül für die folgenden Anfragen an die Mondial-Datenbank an. Vergleichen Sie mit denselben Anfragen in der Algebra und in SQL.

a) Alle Landescodes von Ländern, in denen eine Sprache gesprochen wird, die auch in der Schweiz gesprochen wird.

b) Alle Landescodes von Ländern, in denen ausschliesslich Sprachen gesprochen werden, die in der Schweiz nicht gesprochen werden.

c) Alle Landescodes von Ländern, in denen nur Sprachen gesprochen werden, die auch in der Schweiz gesprochen werden.

d) Alle Landescodes von Ländern, in denen alle Sprachen gesprochen werden, die in der Schweiz gesprochen werden.

```
:- auto_table.
:- include(mondial).

aufgA(C) :- language(C,_X,_), language('CH',_X,_).
aufgB(C) :- country(_,C,_,_,_,_), not aufgA(C).
chlg(L) :- language('CH',L,_).
nonCHLgCtry(C) :- language(C,_L,_), not chlg(_L).
onlyCHLgCtry(C) :- country(_,C,_,_,_,_), not nonCHLgCtry(C).
chLgMissing(C) :- chlg(_L), not language(C,_L,_).
noChLgMissing(C) :- country(_,C,_,_,_,_), not chLgMissing(C).
?- aufgA(C).
?- aufgB(C).
?- onlyCHLgCtry(C).   % note: also countries with no language!
?- noChLgMissing(C).
```

Aufgabe 5 (Datalog-Anfragen an Mondial: Landlocked)

- Geben Sie die Namen aller Länder an, die keine Küste haben.
- Geben Sie die Namen aller Länder an, die selber keine Küste haben, und auch kein Nachbarland haben, das eine Küste hat.

- Geben Sie den Abhängigkeitsgraphen Ihres Programms an.

```
:- auto_table.
:- include(mondial).

borders(Y,X,L) :- borders(X,Y,L).
coast(C) :- geo_sea(S,C,P).
landlocked(C) :- country(_,C,_,_,_,_), not coast(C).
hasnonlandlockedneighbor(C) :- landlocked(C), borders(C,C2,_), not landlocked(C2).
landlandlocked(C) :- landlocked(C), not hasnonlandlockedneighbor(C).
```

Asking `?- hasnonlandlockedneighbor(C)` yields many countries several times, e.g., MK (Macedonia) three times since `C2` can be bound by three ways to coastal neighbors: AL, GR, BG.

This can be avoided by a Prolog cut in the "subquery" that searches for possible `C2` bindings:

```
:- auto_table.
:- include(mondial).

borders(Y,X,L) :- borders(X,Y,L).
coast(C) :- geo_sea(S,C,P).
landlocked(C) :- country(_,C,_,_,_,_), not coast(C).
%hasnonlandlockedneighbor(C) :- landlocked(C), borders(C,C2,_), not landlocked(C2).
hasnlln(C) :- landlocked(C), hasnlln2(C).
hasnlln2(C) :- borders(C,C2,_), not landlocked(C2),!.
landlandlocked(C) :- landlocked(C), not hasnlln(C).
```

Aufgabe 6 (Aggregation in Datalog/XSB) Definieren Sie die Aggregationsoperatoren in XSB in einem Modul `aggs.P`.

Die Syntax der Vergleichsprädikate sowie der arithmetischen Operatoren ist in den Abschnitten 3.10.5 (Inline Predicates) und 4.3 (Operators) des XSB-Manuals Teil 1 beschrieben.

Benutzen Sie `aggs.P` dann zur Beantwortung der folgenden Anfragen in Datalog:

a) Geben Sie für jedes Land den Namen und die Anzahl der Nachbarn aus.

b) Geben Sie den Namen des Landes aus, das am meisten (wieviele?) Nachbarn hat.

c) Geben Sie die durchschnittliche Fläche der Kontinente aus (um avg zu testen).

d) Geben Sie den durchschnittlichen Längen- und Breitengrad aller Städte aus.

```
:- table avg/2.

sum(X,[H|T]) :- sum(Y,T), H \= null, Y \= null, X is H + Y.
sum(H,[H|T]) :- sum(null,T), H \= null.
sum(X,[null|T]) :- sum(X,T).
sum(null,[]).

?- sum(N, [1,2,3]).

count(X,[H|T]) :- count(Y,T), H \= null, X is Y + 1.
count(X,[null|T]) :- count(X,T).
count(0,[]).
```

```
?- count(N, [1,2,3]).

avg(X,L) :- sum(Y,L), count(C,L), Y \= null, C \= null, X is Y / C.
avg(null,L) :- sum(Y,L), Y = null.
avg(null,L) :- count(C,L), C = null.
avg(null,[]).

?- avg(N, [1,2,3]).

min(Y,[H|T]) :- min(Y,T), H \= null, Y \= null, H > Y.
min(H,[H|T]) :- min(Y,T), H \= null, Y \= null, H =< Y.
min(H,[H|T]) :- min(null,T), H \= null.
min(X,[null|T]) :- min(X,T).
min(null,[]).

max(Y,[H|T]) :- max(Y,T), H \= null, Y \= null, H =< Y.
max(H,[H|T]) :- max(Y,T), H \= null, Y \= null, H > Y.
max(H,[H|T]) :- max(null,T), H \= null.
max(X,[null|T]) :- max(X,T).
max(null,[]).
```

```
:- auto_table.
:- table neighbourscount/2, neighbourscount2/2, maxneighbourscount/1.
:- include(mondial).
:- include(aggs).

borders(X,Y) :- borders(X,Y,_).
borders(X,Y) :- borders(Y,X,_).
neighbours(X,NList) :- bagof(Y,borders(X,Y),NList).
neighbourscount(C,N) :- neighbours(C,NList), count(N,NList).

?- neighbourscount(_C,N), country(CName,_C,_,_,_,_).
?- neighbourscount(C,N), N > 10.
% oder kurz auch so:
neighbourscount2(C,N) :- bagof(Y,borders(C,Y),NList), count(N,NList).

%neighbourscounts(CList) :- bagof(N,C^neighbourscount(C,N),CList).
maxneighbourscount(N) :- bagof(N,C^neighbourscount2(C,N),CList), max(N,CList).

% ?- maxneighbourscount(N).
% 16
?- neighbourscount(_C,N), country(CName,_C,_,_,_,_), maxneighbourscount(N).
%% buggy? returns all n (e.g., Austria,8) =:((((

% ?- avg(N,[9562488,45095292,8503474,30254708,39872000]).
% OK
% ?- bagof(Area,CN^continent(CN,Area),AreaList), avg(AvgArea,Arealist).
% never halts

?- city('Stuttgart',_,_,_,Long,Lat).
?- bagof(Long,A^B^C^D^E^city(A,B,C,D,Long,E),_LongList), avg(AvgLong,_LongList).
```

Aufgabe 7 (Datalog: Transitive Hülle) a) Geben Sie verschiedene Datalog-Programme an, die die transitive Hülle einer binären Relation berechnen (wieviele sinnvoll unterschiedliche Strategien gibt es?). Diskutieren Sie Vor- und Nachteile der Programme bzgl. ihrer effizienten Auswertbarkeit.

b) Betrachten Sie zur Aufwandsberechnung und Veranschaulichung einen Baum (um die Zahlenspiele in den Griff zu bekommen, am besten mit festem Verzweigungsgrad) sowie einen beliebigen gerichteten azyklischen bzw zyklischen Graphen.

c) Können die einzelnen Varianten für den Fall spezialisiert werden, daß lediglich die transitiven Nachfolger zu einer gegebenen Konstante berechnet werden sollen?

---

a) 3 Möglichkeiten: linksrekursiv, rechtsrekursiv, doppelt rekursiv.
Gegeben sei eine Relation p, deren transitive Hülle berechnet werden soll.

Linksrekursiv
```
tc(x,y) :- p(X,Y).
tc(X,Y) :- tc(X,Z), p(Z,Y).
```

Rechtsrekursiv
```
tc(x,y) :- p(X,Y).
tc(X,Y) :- p(X,Z), tc(Z,Y).
```

Doppelt rekursiv
```
tc(x,y) :- p(X,Y).
tc(X,Y) :- tc(X,Z), tc(Z,Y).
```

Die links- und rechtsrekursive Variante sind bezüglich ihrer Effizienz im allgemeinen äquivalent:
- Für jedes Element der transitiven Hülle gibt es eine eindeutige Zerlegung in einen Basisfall und einen rekursiven Aufruf.
- Damit sind für ein Element der transitiven Hülle, das n Schritte umfaßt auch n Berechnungsschritte notwendig.
- für einzelne Problemstellungen kann die eine der Varianten günstiger als die andere sein (wobei es zusätzlich darauf ankommt, ob man bottom-up oder top-down-Auswertung verwendet.

Die doppelt rekursive Variante liefert jedes Element der transitiven Hülle, das n Schritte umfaßt in $\log(n)$ Berechnungschritten. Dafür gibt es für jedes Element der transitiven Hülle, das mehr als zwei Schritte umfaßt, mehrere mögliche Zerlegungen. Damit ist der Suchbaum (bei top-down-Auswertung) bzw. die Menge der verwendeten Variablenbindungen (bei naiver top-down-Auswertung) exponentiell in n (und ziemlich redundant).

b) Betrachtung eines binären Baumes mit Höhe 8. Knoten in Dewey-Notation numeriert, 1, 1.1, 1.2, 1.2.1, 1.2.2, ...

Kanten: $I(p) = \{n, n.1\} \cup \{n, n.1\}$

Konten auf Höhe n: 1, 2, 4, 8, 16, 32, 64, 128.

Linksrekursive Variante: $T_P^0(\emptyset) = \emptyset$, $T_P^1(\emptyset) = I(p)$.
$T_P^2(\emptyset) = T_P^1(\emptyset) \cup \{tc(x,y) | p(x,y)\}$ umfasst alle eingliedrigen Pfade (254 Stueck - von jedem Knoten auf Ebene 1-7 zu seinen beiden Kindern). (man könnte auch sagen, von jedem Knoten auf Ebene 2-7 zu seinem Parent.)
$T_P^2(\emptyset)$: jetzt kommen alle zweigliedrigen Pfade hinzu. Wieviele gibt es? von jedem Knoten auf Ebene 1-6 zu jedem seiner 4 Enkel: $4 \cdot (1 + 2 + 4 + 8 + 16 + 32) = 4 \cdot 63 = 252$ (man könnte auch sagen, von jedem Knoten auf Ebene 3-7 zu seinem Grandparent.)
$T_P^3(\emptyset)$: jetzt kommen alle dreigliedrigen Pfade hinzu. Von jedem Knoten auf Ebene 1-5 zu jedem seiner 8 Urenkel: $8 \cdot (1 + 2 + 4 + 8 + 16) = 8 \cdot 31 = 248$
$T_P^4(\emptyset)$: jetzt kommen alle viergliedrigen Pfade hinzu. Von jedem Knoten auf Ebene 1-4 zu jedem seiner 16 Urenkel: $16 \cdot (1 + 2 + 4 + 8) = 16 \cdot 15 = 240$
$T_P^5(\emptyset)$: jetzt kommen alle fuenfgliedrigen Pfade hinzu. $32 \cdot (1 + 2 + 4) = 32 \cdot 7 = 224$
$T_P^6(\emptyset)$: jetzt kommen alle sechsgliedrigen Pfade hinzu. $64 \cdot (1 + 2) = 64 \cdot 3 = 192$

$T_P^7(\emptyset)$: jetzt kommen alle siebengliedrigen Pfade hinzu. $128 \cdot 1 = 128$. $T_P^8(\emptyset)$: stellt fest, dass keine neuen Regelinstanziierungen mit gebildet werden können, da die Endpunkte der neuen siebengliedrigen Pfade keine Kinder mehr haben. – Fixpunkt ist erreicht.

Für jeden Pfad gibt es genau eine Regelinstanziierung, die ihn ableitet.
Also 254+252+248+240+224+192+128=1538.
(eigentlich werden sogar alle niedrigeren Levels in jedem T_P erneut berechnet – hier wird seminaive Auswertung angenommen, in der jeweils ein in der vorhergehenden Iteration neu hinzugekommenes Atom beteiligt sein muss),

Rechtsrekursive Variante: Dasselbe (siehe eingeklammerte Sichtweise).

Doppelt rekursive Variante: $T_P^0(\emptyset)$, $T_P^1(\emptyset)$ und $T_P^2(\emptyset)$ wie oben. 254+252.

$T_P^3(\emptyset)$: Leitet jetzt alle 3- und 4-gliedrigen Pfad ab.
Aber für jeden 3-gliedrigen Pfad gibt es zwei mögliche Regelinstantiierungen (1+2 und 2+1 Glieder). $248 \cdot 2$.
Für jeden 4-gliedrigen Pfad gibt es erstmal nur eine mögliche Regelinstantiierung (2+2). 240.

$T_P^4(\emptyset)$: Leitet alle 4-8-gliedrigen Pfade aus den bisher gewonnenen 1-, 2-, 3- und 4-gliedrigen Pfaden ab, wobei mindestens ein 3- oder 4-gliedriger Pfad verwendet weren muss:
Für jeden 4-gliedrigen Pfad gibt es nochmal zwei mögliche Regelinstantiierungen (1+3, 3+1). $240 \cdot 2$.
Für jeden 5-gliedrigen Pfad gibt es vier mögliche Regelinstantiierungen (1+4, 2+3, 3+2, 4+1). $224 \cdot 4$.
Für jeden 6-gliedrigen Pfad gibt es erstmal nur drei mögliche Regelinstantiierungen (2+4, 3+3, 4+2): $192 \cdot 3$.
Für jeden 7-gliedrigen Pfad gibt es erstmal nur zwei mögliche Regelinstantiierungen (3+4, 4+3): $128 \cdot 2$.

$T_P^5(\emptyset)$: Mit den zuletzt neu dazugekommenen 5-7 gliedrigen Pfaden lassen sich nochmal neue Regelinstanziierungen bilden (die allerdings nichts neues mehr ableiten):
Für jeden 6-gliedrigen Pfad gibt es nochmal zwei mögliche Regelinstantiierungen (1+5, 5+1): $192 \cdot 2$.
Für jeden 7-gliedrigen Pfad gibt es nochmal vier mögliche Regelinstantiierungen (1+6, 2+5, 5+2, 6+1): $128 \cdot 4$.
Summe: 3866.

Für das Top-Down-Vorgehen per Resolutionsbeweis sieht die Lage nicht besser aus, da beliebige Kombinationen geprüft werden. Verwendung des Prolog-Cuts kann dies aber verhindern. Dann ist es egal, welche Variante man nimmt.

In einem Graphen (z.B. via borders voneinander auf dem Landweg erreichbare Länder) tritt dieser Effekt noch extremer auf, da es mehr verschiedene Wege gibt. Die Links- und rechtsrekursiven Variante machen eher eine systematische Exploration, während die doppelt rekursive "wahllos" kombiniert und daher deutlich mehr Regelinstanziierungen durchrechnet.

Dabei ist es relativ egal, ob der Graph zyklisch ist oder nicht. Zyklische Pfade werden einmal berechnet, dann als bereits bekannt festgestellt, und mit ihnen keine neuen Regelinstanziierungen erzeugt.

Hinweise:

- T_P: man kann in allen Fällen nicht verhindern, dass Regelinstanziierungen gebildet werden, deren Kopf etwas schon bekanntes ableitet - das weiss man ja vorher nicht.
- Fuer den top-down-Beweis kann man via Prolog-Cut verbieten, weitere Beweise für ein bereits als wahr nachgewiesenes Atom zu suchen.
- Tabling kombiniert top-down und bottom-up, indem gespeichert wird, was bereits nachgewiesen wurde (bzw. auch welche Instanziierungen nicht bewiesen werden konnten), um es in

anderen Teilbeweisen wiederzuverwenden.

c) Berechnung der Nachfolger zu einer gegebenen Konstante ("alle Kinder einer gegebenen Person"): Wie sich in der Analyse der Programme bereits andeutete, ist hier die linksrekursive Variante am leichtesten zu modifizieren:

```
reachable(a).
reachable(Y) :- reachable(X), p(X,Y).
```

Für die anderen Varianten gibt es keine naheliegende Modifikation.