

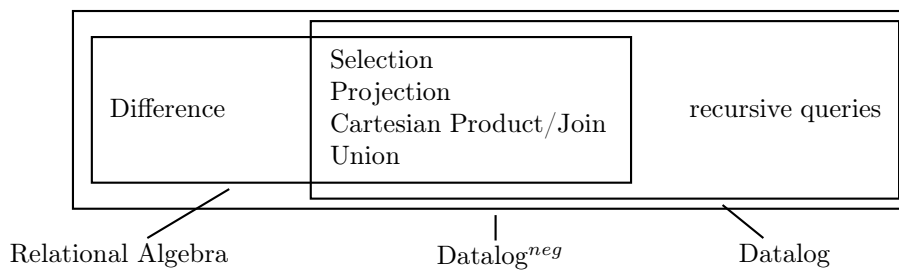
**Database Theory**  
**Winter Term 2013/14**  
 Prof. Dr. W. May

## 2. Unit: Datalog

Discussion by 15./22.1.2014

**Exercise 1 (Äquivalenz von Algebra und Datalog)** Show that for every expression of the relational algebra there is an equivalent stratified Datalog program.

---



**Union** Let  $p, q$  relations. Then, for  $u = p \cup q$

$u(X_1, \dots, X_N) :- p(X_1, \dots, X_N).$

$u(X_1, \dots, X_N) :- q(X_1, \dots, X_N).$

**Difference** Let  $p, q$  relations. Then, for  $d = p \setminus q$

$d(X_1, \dots, X_N) :- p(X_1, \dots, X_N), \text{ not } q(X_1, \dots, X_N).$

**Projection** Let  $p$  a relation with attributes  $X_1, \dots, X_n$ . Then, for  $pr = \pi[X_{i_1}, \dots, X_{i_k}](p)$  with  $X_{i_j} \in \{X_1, \dots, X_n\}$

$pr(X_{I1}, \dots, X_{IK}) :- p(X_1, \dots, X_N).$

**Selection** Let  $p$  a relation with attributes  $X_1, \dots, X_n$ ,  $\alpha$  a condition over  $X_1, \dots, X_n$ . Then, for  $s = \sigma[\alpha](p)$

$s(X_1, \dots, X_N) :- p(X_1, \dots, X_N), \alpha.$

**Join** Let  $p, q$  relations with common attributes  $X_k, \dots, X_m$ .

Then, for  $j = p \bowtie q$

$j(X_1, \dots, X_N) :- p(X_1, \dots, X_k, \dots, X_m), q(X_k, \dots, X_m, \dots, X_N).$

The program that corresponds to a complex algebra expression is stratified since each subexpression defines a new predicate symbol, and thus the dependency graph corresponds to the tree structure of the expression.

---

**Exercise 2 (Datalog to Algebra)**

Consider the translation of Datalog programs with a distinguished answer predicate to the relational algebra.

- Given a rule  $B \leftarrow C_1 \wedge \dots \wedge C_m \wedge \neg D_{m+1} \wedge \dots \wedge \neg D_{m+n}$  where the  $C_i$  and  $D_i$  are of the form  $R_i(a_1, \dots, a_\ell)$ ,  $a_j$  constants or variables. Give an algebra expression that returns the relation defined by it.
- Which additional construct must also be translated?
- Consider the following program (arbitrary arity of predicates, each rule assumed to be safe):
 

```

res(X,Z) :- v(X,_,_ Y), q(,_,_ Y,Z), ¬r(Z,_)
res(X,Z) :- v(X,_,_ Y,Z), ¬r(,_,_ Y,_) , ¬w(X)
v(X,Y,Z) :- p(Z,_,X), q(X,Y,_)
v(X,Y,Z) :- p(X,Y,Z), Y<4.
w(X) :- s(,_,X), t(X,_)

```

where  $p/3, q/3, r/2, s/2, t/2$  are EDB relations,  $v/3, w/1$  are IDB relations (views).

Give the algebra expression that corresponds to the  $res$  predicate.

- For each  $C_i(X_1, \dots, X_{k_i})$  and  $D_i(X_1, \dots, X_{k_i})$ , there is an equivalent algebra expression  $E_i = \rho[\dots](\pi[\dots](R_i))$  (note that  $R_i$  may be a complex expression if  $R_i$  is an EDB predicate) with format  $(X_1, \dots, X_{m_i})$  that selects the relevant attributes/variables/columns and renames them to  $X_1, \dots, X_{k_i}$ .

Safety implies that all variables that occur in any of the  $D_i$  also occur in at least one of the  $C_i$ .

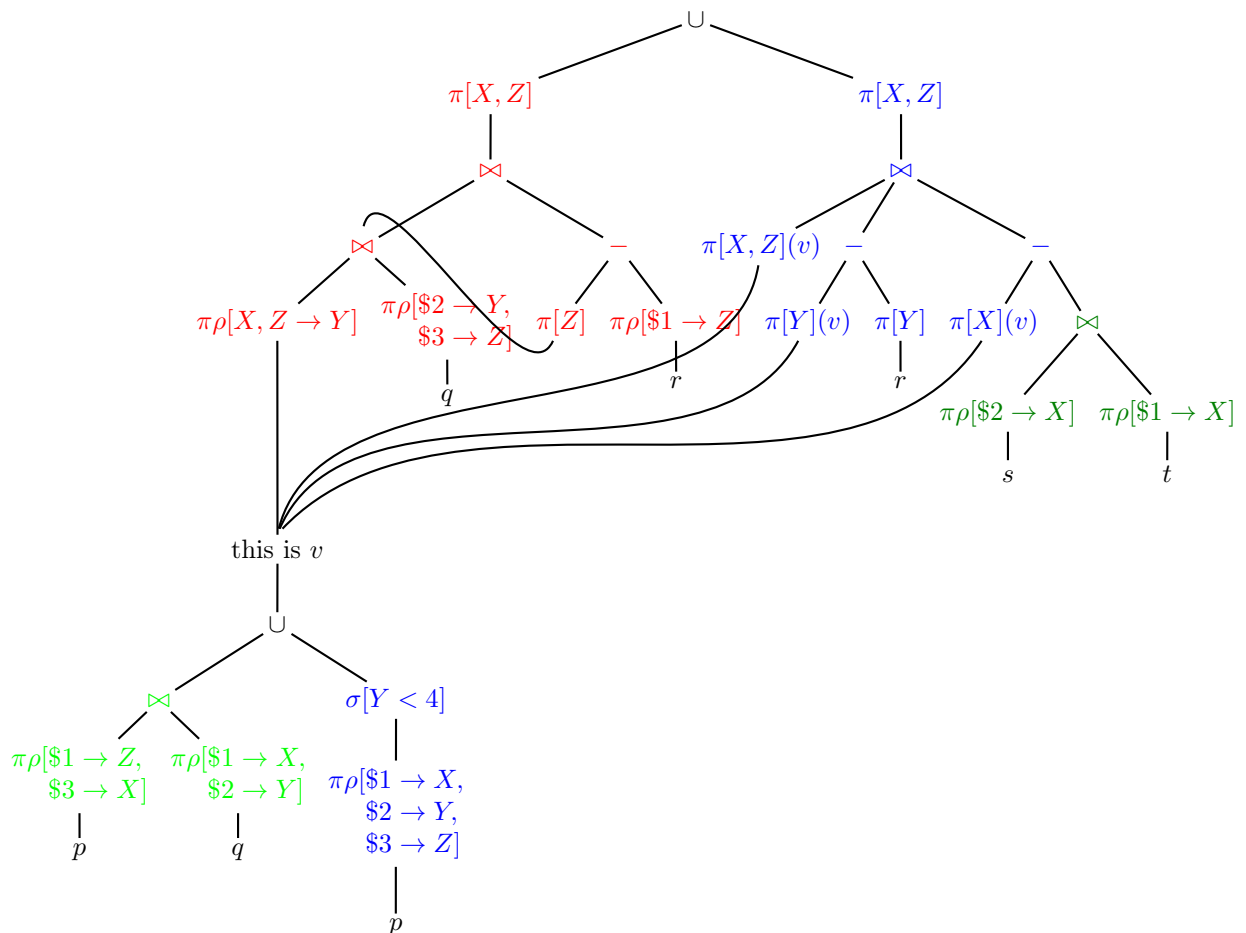
Let  $C := E_1 \bowtie \dots \bowtie E_m$ . Then,

$$\pi[\text{Vars}(B)](C \bowtie (\pi[\text{Vars}(D_{m+1})](C) - D_{m+1}) \bowtie \dots \bowtie (\pi[\text{Vars}(D_{m+n})](C) - D_{m+n})) \quad (*)$$

is the required expression.

(Note that this is analogous to the RANF to Algebra transformation; cf. Proof “Calculus to Algebra” in the lecture.)

- – comparison atoms of the form  $X_1 \text{ op } X_j$  or  $X_i \text{ op } c$ : selections applied to (\*).
- If two rules define the same predicate symbol (i.e., have the same head): union.
- Subtrees/intermediate results can sometimes be used twice. This is supported by algebraic optimization and tabling.



### Exercise 3 (Stratified Datalog)

Give an example for the nonmonotonicity of the stratified semantics,  
 show that for a stratifiable program  $P$  there can be multiple minimal models.

This can be shown and illustrated in several ways:

- use a “real” scenario and program for illustration and understanding.
- use a *small* typical, abstract program to focus on the formal aspects.

Both will be shown below.

**Intuitive example** Consider the “real” program  $P$  as follows:

```

All facts from the Mondial database.
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 the stratified model  $\mathcal{S}(P)$ , the *reachable* predicate contains e.g. the pairs (D,F), (D,SGP), . . . . The *unreachable* contains among many others e.g. the pairs (D,USA), (D,BR), and (M,NZ); the last one stands for Malta and New Zealand, which are both countries that are located on islands and do not have any neighbors.

Cf. Lecture: the stratified model  $\mathcal{S}(P)$  of  $P$  is a minimal model, i.e., no proper subset of  $\mathcal{M}$  is also a model.

**Nonmonotonicity:** If e.g. the fact `borders(M,NZ)` is added to the program (i.e., “learnt” as new knowledge), obtaining a new program  $P'$ . The conclusions about `borders` (symmetric) and `reachable` (transitive hull) must be completed:

$$\begin{aligned} \text{Then, } \mathcal{S}'(P') = & (\mathcal{S}(P) \cup \{\text{border}(M,NZ), \text{border}(M,NZ)\} \\ & \cup \{\text{reachable}(M,NZ), \text{reachable}(NZ,M)\}) \\ & \setminus \{\text{unreachable}(M,NZ), \text{unreachable}(NZ,M)\} \end{aligned}$$

The example exhibits nonmonotonicity: Here, a new fact has been learnt, and some conclusions that have been valid before, namely `unreachable(M,NZ)` and `unreachable(NZ,M)`, have been withdrawn.

More general: Closed-World reasoning is inherently nonmonotonic. When some conclusion is made from non-existence of some fact, and this fact is learned to be true later, the conclusion must be withdrawn.

**Multiple minimal models:** The above  $\mathcal{S}'(P')$  is also a model of the original program  $P$ , and also a minimal model of  $P$  (i.e., there is no proper subset of it that is also a model of  $P$ ; especially note that  $\mathcal{S}(P)$  is not a subset of  $\mathcal{S}'(P')$  because some conclusions from  $\mathcal{S}(P)$  have been *withdrawn* – **the nonmonotonicity plays an important role to make multiple minimal models possible**)

Recall that for a purely positive program  $P_2$ , the minimal model of  $P_2 \cup \{\text{any positive atoms}\}$  is always a superset of the minimal program of  $P_2$ , i.e., the minimal model theory is monotonic.

In general, adding/“inventing” *any positive fact* to  $P$  can be extended to a minimal model of  $P$ . Above, a base fact (i.e., an instance of an EDB relation) had been added, corresponding to learn a new fact about the real world.

The same holds also when instances of IDB relations are invented:

Consider just

$\mathcal{M}' = (\mathcal{S}(P) \cup \{\text{reachable}(M,NZ)\}) \setminus \{\text{unreachable}(M,NZ)\}$ , which is also a model of  $P$ , and it is also a minimal one.

$\mathcal{M}'$  better than  $\mathcal{S}'(P')$  illustrates how these minimal models of  $P$  are not “as good as” the unique stratified model of  $P$ :

- $\mathcal{M}'$  is not “nice” from the logical point of view because it contains an “unfounded” `reachable` tuple. It does not violate  $P$ , but it adds something that  $P$  would not have derived.
- $\mathcal{M}'$  is also not nice from the application point of view: `reachable(NZ,M)` is not there, but `unreachable(NZ,M)` – since the rules do not require `reachable` to be symmetric (it is symmetric in the stratified model of the original  $P$  since `borders` is required to be symmetric and `reachable` is its transitive closure).

The intuition becomes clear with this example: “invent” a fact, adapt the resulting conclusions accordingly and obtain a new minimal model.

Here, it was a “lucky choice” to retain a relatively simple example.

Consider an uglier case: add `reachable(P,USA)`. Lots of `reachable` atoms are added, i.e. all american countries are now reachable from all european countries (and via Russia) also from all asian countries, and (via Egypt) then also from all african countries, but *not* the other way round (Details: self-study).

Finally consider to add `border(P,USA)` (Details: self-study).

This intuitive example gives an understanding what happens, but contains too much overhead for theoretical characterizations.

For that, logicians prefer minimal, abstract examples.

**Abstract, minimal example** Consider the program  $P := \{p :- \neg q\}$

Its dependency graph is  $p \xleftarrow{\neg} q$ .

Thus, there is a stratification  $S_1 = \{q\}$ ,  $S_2 = \{p\}$ , resulting in  $P_1 = \emptyset$  and  $P_2 = \{p :- \neg q\}$ .

Thus,  $\mathcal{I}_0 = \emptyset$ ,  
 $\mathcal{I}_1 = \emptyset \cup T_{P_1}^\omega(\emptyset) = \emptyset$ ,  
 $\mathcal{I}_2 = \emptyset \cup T_{P_2}^\omega(\emptyset) = \{p\} =: \mathcal{S}(P)$

The (unique) *stratified model*  $\mathcal{S}(P)$  is a model of  $P$  and it is a minimal model.

Consider  $\mathcal{M}' := \{\text{II}\}$  (invented: fact  $q$ ). It is a model of  $P$  (i.e. satisfies all its rules), and it is a minimal model (i.e. no proper subset of it is also a model of  $P$ ).

Nonmonotonicity: the set of conclusions from  $P := \{p :- \neg q\}$  is  $\{p\}$  while the set of conclusions from  $P \cup \{q\}$  is  $\{q\}$  – i.e., conclusion  $p$  must be withdrawn when learning that  $q$  holds.

In the rest of the lecture, some more notions of models will be considered: The above  $\mathcal{M}'$  are neither well-founded (i.e., each atom in them has some derivation), nor stable (i.e., they do not reproduce themselves).

#### Exercise 4 (Datalog-Anfragen an Mondial: Landlocked)

- Give a Datalog program that returns the names of all countries that have no coast.
- Give a Datalog program that returns the names of all countries that have no coast and that have no neighbor country that has any coast.
- Give the dependency graph of your program.

```
:- 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).
```

---

**Exercise 5 (Aggregation in Datalog/XSB)** Define the aggregation operators in XSB in a module `aggs.P`.

The syntax of the comparison predicates and of the arithmetic operators is given in Sections 3.10.5 (Inline Predicates) and 4.3 (Operators) of the XSB Manual Part I.

Then use `aggs.P` for answering the following queries in Datalog:

- Give for each country the name and the number of neighbors.
  - Give the name of the country that has the highest number of neighbors (and how many).
  - Give the average area of all continents (to test `avg`).
  - Give the average latitude and longitude of all cities.
- 

```
:- 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 \= 0, X is Y / C.
avg(null,L) :- sum(Y,L), Y = null.
avg(null,L) :- count(C,L), C = 0.
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(M) :- bagof(_N,_C^neighbourscount(_C,_N),_CList), max(M,_CList).

% ?- maxneighbourscount(N).
% 16
?- maxneighbourscount(N), neighbourscount(_C,N), country(CName,_C,_,_,_,_).
?- neighbourscount(_C,N), country(CName,_C,_,_,_,_), maxneighbourscount(N).

% ?- avg(N,[9562488,45095292,8503474,30254708,39872000]).
% N = 26657592.4000
% ?- bagof(_Area,_CN^continent(_CN,_Area),_AreaList), avg(AvgArea,_AreaList).
% AvgArea = 26657592.4000

% ?- city('Stuttgart',_,_,_,Long,Lat).
% ?- bagof(Long,A^B^C^D^E^city(A,B,C,D,Long,E),_LongList), avg(AvgLong,_LongList).
% ?- bagof(Lat,A^B^C^D^E^city(A,B,C,D,E,Lat),_LatList), avg(AvgLat,_LatList).
avglonglat(AvgLong,AvgLat) :-
    bagof(_Long,_A^_B^_C^_D^_E^city(_A,_B,_C,_D,_Long,_E,_),_LongList), avg(AvgLong,_LongList),
    bagof(_Lat,_A^_B^_C^_D^_E^city(_A,_B,_C,_D,_E,_Lat,_),_LatList), avg(AvgLat,_LatList).

```

---