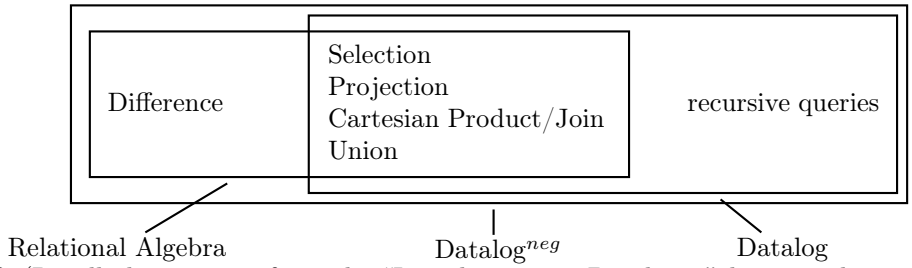
Database Theory
Winter Term 2016/17
Prof. Dr. W. May

2. Unit: Datalog

Discussion by 14./21.12.2016

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

Relationships between Relationaler Algebra, Datalog und Datalog^{neg}:



Proof: (Recall the exercise from the "Introduction to Databases" lecture where the relational completeness has been shown.)

It has to be shown that for every expression e of the relarional algebram there is an equivalent Datalog program. I.e., let $\mathbf{R}$ a relational schema and $\mathcal{S}$ a database state over $\mathbf{R}$, then there is a Datalog program P_e with a distinguished result predicate res_e such that the answer set to res_e equals $\mathcal{S}(e)$ ist.

To show this for *all* algebra expressions, the proof is done by *structural induction* following the definition of the relational algebra expressions:

Induction base: the atomic cases:

- let e a relation name r. Then, e is an n-ary EDB predicate for some n. Let $res_e = e$.
- Sei $e = cname : \{x\}$, i.e., the "atomic table"

cname
x

  Then, `{res`$_e$`(X) :- X =` x`.}` is the program P_e, containing just a single rule.

Induction step: Let e a composite algebra expression, i.e., the application of an algebra operator on one or two simpler expressions e_1 and e_2. By induction hypothesis, there are Datalog programs P_{e_i} that are equivalent to e.
Case split according to the outermost algebra operator of e (consider only the base operators $\cup$, $\setminus$, $\sigma[...]$, $\pi[...]$, $\rho[...]$ und $(...) \bowtie (...)$ from the definition of the relationale algebra):
For the first 5 cases, assume that e_1 has the format $[A_1, \ldots, A_n]$.

- $e = e_1 \cup e_2$: $P_e = P_{e_1} \cup P_{e_2} \cup$
  `{res`$_e$`(X`$_1$`,...,X`$_n$`) :- res`$_{e_1}$`(X`$_1$`,...,X`$_n$`).} ∪`
  `{res`$_e$`(X`$_1$`,...,X`$_n$`) :- res`$_{e_2}$`(X`$_1$`,...,X`$_n$`).}`
- $e = e_1 \setminus e_2$: $P_e = P_{e_1} \cup P_{e_2} \cup$
  `{res`$_e$`(X`$_1$`,...,X`$_n$`) :- res`$_{e_1}$`(X`$_1$`,...,X`$_n$`), not res`$_{e_1}$`(X`$_1$`,...,X`$_n$`).}`

- $\sigma[cond](e_1)$: If *cond* is a conjunction of atomic comparisons like $a\theta c$ or $a_i\theta a_j$, then $P_e = P_{e_1} \cup \{\texttt{res}_e(\texttt{X}_1,\ldots,X_n) \texttt{ :- } \texttt{res}_{e_1}(\texttt{X}_1,\ldots,\texttt{X}_n),\ cond\}$.
  Otherwise, transform *cond* in DNF and translate into a union of selections, where each condition is such a conjunction of atomic comparisons.
- $\pi[A_{i_1},\ldots,A_{i_k}](e_1)$ where $1 \le i_j \le n$, $A_{i_1},\ldots,A_{i_k}$ pairwise distinct:
  $P_e = P_{e_1} \cup P_{e_2} \cup$
  $\{\texttt{res}_e(\texttt{X}_{i_1},\ldots,X_{i_k}) \texttt{ :- } \texttt{res}_{e_1}(\texttt{X}_1,\ldots,\texttt{X}_n).\}$.
  If the A_{i_j} are not pairwise distinct (i.e., some columns are duplicated), the corresponding $\texttt{X}_{i_j}$ must also occur multiply in the head atom.
- $\rho[A_1 \to C_1,\ldots,A_n \to C_n](e_1)$: can be skipped since Datalog is not a named model, but a positional one. Thus, renaming has no effect.
- $e = e_1 \bowtie e_2$, where e_2 has the format $[A_{i_1},\ldots,A_{i_k},B_{k+1},\ldots,B_m]$, $\{i_1,\ldots,i_k\} \subseteq \{1\ldots n\}$ and pairwise distinct: $P_e = P_{e_1} \cup P_{e_2} \cup$
  $\{\texttt{res}(\texttt{X}_1,\ldots,X_n,\texttt{Y}_1,\ldots,Y_n) \texttt{ :- } \texttt{res}_{e_1}(\texttt{X}_1,\ldots,\texttt{X}_n),\ \texttt{res}_{e_2}(\texttt{X}_{i_1},\ldots,\texttt{X}_{i_k},\texttt{Y}_1,\ldots,Y_n)\}$.
  If the columns in common are not the first k ones of e_2, the variables in the $\texttt{res}_{e_2}$ atom must be reordered accordingly.

The program that corresponds to a composite 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 $B \leftarrow C_1 \wedge \ldots \wedge C_m \wedge \neg D_{m+1} \wedge \ldots \wedge \neg D_{m+n}$
  where the C_i and D_i are of the form $R_i(a_1,\ldots,a_\ell)$, a_j constants or variables. Give an algebra expression that returns the relation defined by it.
- Which additional constructs must also be translated?
- Consider the following program:

  ```
  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 $C_i(X_1,\ldots,X_{k_i})$ and $D_i(X_1,\ldots,X_{k_i})$, there is an equivalent algebra expression $E_i = \rho[\ldots](\pi[\ldots](R_i))$ (note that R_i may be a composite expression if R_i is an IDB predicate) with format $(X_1,\ldots,X_{m_i})$ that selects the relevant attributes/variables/columns and renames them to $X_1,\ldots,X_{k_i}$.
  Safety implies that all variables that occur in any of the D_i also occur in at least one of the C_i.
  Let $C := E_1 \bowtie \ldots \bowtie E_m$. Then,

  $$\pi[\mathsf{Vars}(B)](C \bowtie (\pi[\mathsf{Vars}(D_{m+1})](C) - E_{m+1}) \bowtie \ldots \bowtie (\pi[\mathsf{Vars}(D_{m+n})](C) - E_{m+n})) \qquad (*)$$

  is the required expression.
  (Note that this is analogous to the RANF to Algebra transformation with "push-into-negation"; cf. Proof "Calculus to Algebra" in the lecture.)
  Note: there is also another strategy that takes the join of the positive literals, and subtracts

one after the other negative literal:

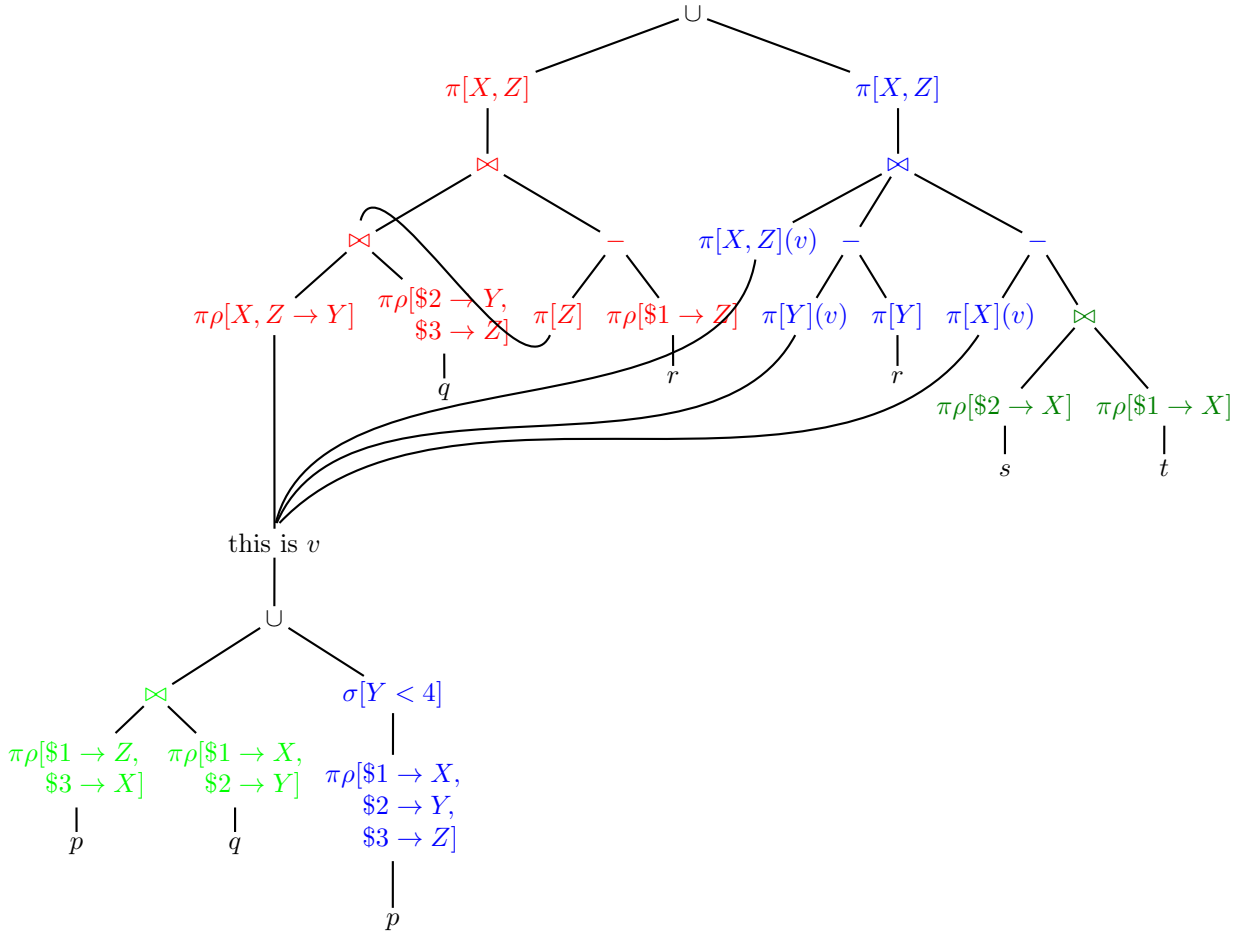
$$\pi[\mathsf{Vars}(B)](...(C - (C \bowtie E_{m+1}) - ... - (C \bowtie E_{m+k}))) \ .$$

This one in course, can be simplified using the "anti-join" ($R \triangleright S = R \setminus R \ltimes S$) operator which corresponds to "NOT EXISTS":

$$\pi[\mathsf{Vars}(B)](...(C \triangleright E_{m+1}) \triangleright ... \triangleright E_{m+k}) \ .$$

The antijoin also corresponds to the "Negation as failure" proof strategy for negative literals in Prolog/Datalog.

- – comparison atoms of the form X_1 *op* X_j or X_i *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:

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

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(M,NZ), 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:

- $\mathcal{M}'$ ist 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 $P := \{p \text{ :- } \neg q\}$

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 $P_2 = \{p \text{ :- } \neg q\}$.

Thus, $\begin{array}{rcl} \mathcal{I}_0 & = & \emptyset, \\ \mathcal{I}_1 & = & \emptyset \cup T^\omega_{P_1}(\emptyset) = \emptyset, \\ \mathcal{I}_2 & = & \emptyset \cup T^\omega_{P_2}(\emptyset) = \{p\} =: \mathcal{S}(P) \end{array}$

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 $P := \{p \text{ :- } \neg q\}$ 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: Schweizer Sprachen) Give Datalog programs for the following queries against the Mondial database. Compare with the same queries in the algebra and in the relational calculus.

a) All codes of countries in which some language is spoken that is also spoken in Switzerland.

b) All codes of countries in which only languages are spoken that are not spoken in Switzerland.

c) All codes of countries in which only languages are spoken that are also spoken in Switzerland.

d) All codes of countries in which all languages are spoken that are spoken in Switzerland.

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

% first attempt: contains duplicates
```

```
aufgA(C) :- language(C,_L,_), language('CH',_L,_).
% eliminate them by using a cut:
aufgAdistinct(C) :- country(_,C,_,_,_,_), commonlang(C,'CH').
commonlang(X,Y) :- language(X,_L,_), language(Y,_L,_),!.

aufgB(C) :- country(_,C,_,_,_,_), not aufgA(C).

nonCHLgCtry(C) :- language(C,_L,_), not language('CH',_L,_).
onlyCHLgCtry(C) :- country(_,C,_,_,_,_), not nonCHLgCtry(C).

chLgMissing(C) :- country(_,C,_,_,_,_), language('CH',_L,_), not language(C,_L,_).
noCHLgMissing(C) :- country(_,C,_,_,_,_), not chLgMissing(C).
?- aufgA(C).
?- aufgAdistinct(C).
?- aufgB(C).
?- onlyCHLgCtry(C).   % note: also countries with no language!
?- noCHLgMissing(C).
```

Exercise 5 (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 6 (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:

a) Give for each country the name and the number of neighbors.

b) Give the name of the country that has the highest number of neighbors (and how many).

c) Give the average area of all continents (to test avg).

d) 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).
```