TECHNISCHE UNIVERSITÄT DRESDEN

FOUNDATIONS OF DATABASES AND QUERY LANGUAGES

Lecture 10: Expressive Power and Complexity of Datalog

Markus Krötzsch

TU Dresden, 22 June 2015

Overview

1. Introduction | Relational data model
2. First-order queries
3. Complexity of query answering
4. Complexity of FO query answering
5. Conjunctive queries
6. Tree-like conjunctive queries
7. Query optimisation
8. Conjunctive Query Optimisation / First-Order Expressiveness
9. First-Order Expressiveness / Introduction to Datalog
10. Expressive Power and Complexity of Datalog
11. Implementation techniques for Datalog
12. Path queries
13. Constraints
14. Outlook: database theory in practice

See course homepage [⇒ link] for more information and materials

Review: Datalog

A rule-based recursive query language

$$
\begin{aligned}
\text{father}(\text{alice}, \text{bob}) & \\
\text{mother}(\text{alice}, \text{carla}) & \\
\text{Parent}(x, y) & \leftarrow \text{father}(x, y) \\
\text{Parent}(x, y) & \leftarrow \text{mother}(x, y) \\
\text{SameGeneration}(x, x) & \\
\text{SameGeneration}(x, y) & \leftarrow \text{Parent}(x, v) \wedge \text{Parent}(y, w) \wedge \text{SameGeneration}(v, w)
\end{aligned}
$$

There three equivalent ways of defining Datalog semantics:

- Proof-theoretic: What can be proven deductively?
- Operational: What can be computed bottom up?
- Model-theoretic: What is true in the least model?

Next questions:

- What can we express in this language?
- How hard is it in terms of complexity?

Datalog and UCQs

We have seen in the exercise that UCQs can be expressed in Datalog. $\leadsto$ Let's make this relationship more precise

For a Datalog program P:

- An IDB predicate R depends on an IDB predicate S if P contains a rule with R in the head and S in the body.
- P is non-recusrive if there is no cyclic dependency.

Theorem

UCQs have the same expressivity as non-recursive Datalog.

That is: a query mapping can be expressed by some UCQ if and only if it can be expressed by a non-recursive Datalog program.

However, Datalog can be exponentially more succinct (shorter queries), as illustrated in exercise.

Datalog and Domain Independence

Domain independence was considered useful for FO queries
$\rightsquigarrow$ results should not change if domain changes

Several solutions:

- Active domain semantics: restrict to elements mentioned in database or query
- Domain-independent queries: restrict to query where domain does not matter
- Safe-range queries: decidable special case of domain independence

Our definition of Datalog uses the active domain (=Herbrand universe) to ensure domain independence

Safe Datalog Queries

Similar to safe-range FO queries, there are also simple syntactic conditions that ensure domain independence for Datalog:

Definition

A Datalog rule is safe if all variables in its head also occur in its body. A Datalog program/query is safe if all of its rules are.

Simple observations:

- safe Datalog queries are domain independent
- every Datalog query can be expressed as a safe Datalog query . . .
- . . . and un-safe queries are not much more succinct either (exercise)

Some texts require Datalog queries to be safe in general
but in most contexts there is no real need for this

Complexity of Datalog

How hard is answering Datalog queries?

Recall:

- Combined complexity: based on query and database
- Data complexity: based on database; query fixed
- Query complexity: based on query; database fixed

Plan:

- First show upper bounds (outline efficient algorithm)
- Then establish matching lower bounds (reduce hard problems)

A Simpler Problem: Ground Progams

Let's start with Datalog without variables

$\leadsto$ sets of ground rules a.k.a. propositional Horn logic program

Naive computation of T_P^∞:

$T_P^0 := \emptyset$

$i := 0$

repeat :

  $T_P^{i+1} := \emptyset$

  for $H \leftarrow B_1 \wedge \ldots \wedge B_\ell \in P$:

    if $\{B_1, \ldots, B_\ell\} \subseteq T_P^i$:

      $T_P^{i+1} := T_P^{i+1} \cup \{H\}$

  $i := i + 1$

until $T_P^{i-1} = T_P^i$

return T_P^i

A Simpler Problem: Ground Progams

Let's start with Datalog without variables
$\leadsto$ sets of ground rules a.k.a. propositional Horn logic program

Naive computation of T_P^∞:

$T_P^0 := \emptyset$
$i := 0$
repeat :
  $T_P^{i+1} := \emptyset$
  for $H \leftarrow B_1 \wedge \ldots \wedge B_\ell \in P$:
    if $\{B_1, \ldots, B_\ell\} \subseteq T_P^i$:
      $T_P^{i+1} := T_P^{i+1} \cup \{H\}$
  $i := i + 1$
until $T_P^{i-1} = T_P^i$
return T_P^i

How long does this take?

- At most $|P|$ facts can be derived
- Algorithm terminates with $i \leq |P| + 1$
- In each iteration, we check each rule once (linear), and compare its body to T_P^i (quadratic)

$\leadsto$ polynomial runtime

Complexity of Propositional Horn Logic

Much better algorithms exist:

Theorem (Dowling & Gallier, 1984)

For a propositional Horn logic program P, the set T_P^∞ can be computed in linear time.

Complexity of Propositional Horn Logic

Much better algorithms exist:

Theorem (Dowling & Gallier, 1984)

For a propositional Horn logic program P, the set T_P^∞ can be computed in linear time.

Nevertheless, the problem is not trivial:

Theorem

For a propositional Horn logic program P and a proposition (or ground atom) A, deciding if $A \in T_P^\infty$ is a P-complete problem.

Remark:
all P problems can be reduced to propositional Horn logic entailment
yet not all problems in P (or even in NL) can be solved in linear time!

Datalog Complexity: Upper Bounds

A straightforward approach:

(1) Compute the grounding ground(P) of P w.r.t. the database $\mathcal{I}$

(2) Compute $T^{\infty}_{\text{ground}(P)}$

Datalog Complexity: Upper Bounds

A straightforward approach:

(1) Compute the grounding ground(P) of P w.r.t. the database $\mathcal{I}$

(2) Compute $T^{\infty}_{\text{ground}(P)}$

Complexity estimation:

- The number of constants N for grounding is linear in P and $\mathcal{I}$
- A rule with m distinct variables has N^m ground instances
- Step (1) creates at most $|P| \cdot N^M$ ground rules, where M is the maximal number of variables in any rule in P
  - ground(P) is polynomial in the size of $\mathcal{I}$
  - ground(P) is exponential in P
- Step (2) can be executed in linear time in the size of ground(P)

Summing up: the algorithm runs in P data complexity and in ExpTime query and combined complexity

Datalog Complexity

These upper bounds are tight:

Theorem

Datalog query answering is:

- ExpTime-complete for combined complexity
- ExpTime-complete for query complexity
- P-complete for data complexity

It remains to show the lower bounds.

P-Hardness of Data Complexity

We need to reduce a P-hard problem to Datalog query answering
$\leadsto$ propositional Horn logic programming

We restrict to a simple form of propositional Horn logic:

- facts have the usual form $H \leftarrow$
- all other rules have the form $H \leftarrow B_1 \wedge B_2$

Deciding fact entailment is still P-hard (exercise)

We can store such programs in a database:

- For each fact $H \leftarrow$, the database has a tuple Fact(H)
- For each rule $H \leftarrow B_1 \wedge B_2$,
  the database has a tuple Rule(H, B_1, B_2)

P-Hardness of Data Complexity (2)

The following Datalog program acts as an interpreter for propositional Horn logic programs:

$$\mathsf{True}(x) \leftarrow \mathsf{Fact}(x)$$

$$\mathsf{True}(x) \leftarrow \mathsf{Rule}(x, y, z) \wedge \mathsf{True}(y) \wedge \mathsf{True}(z)$$

Easy observations:

- $\mathsf{True}(A)$ is derived if and only if A is a consequence of the original propositional program
- The encoding of propositional programs as databases can be computed in logarithmic space
- The Datalog program is the same for all propositional programs

$\leadsto$ Datalog query answering is P-hard for data complexity

$\textsc{ExpTime}$-Hardness of Query Complexity

A direct proof:
Encode the computation of a deterministic Turing machine for up to exponentially many steps

Recall that $\textsc{ExpTime} = \bigcup_{k\geq 1} \textsc{Time}(2^{n^k})$

- in our case, $n = N$ is the number of database constants
- k is some constant

$\leadsto$ we need to simulate up to 2^{N^k} steps (and tape cells)

Main ingredients of the encoding:

- $\text{state}_q(X)$: the TM is in state q after X steps
- $\text{head}(X, Y)$: the TM head is at tape position Y after X steps
- $\text{symbol}_\sigma(X, Y)$: the tape cell at position Y holds symbol σ after X steps

$\leadsto$ How to encode 2^{N^k} time points X and tape positions Y?

Preparing for a Long Computation

We need to encode 2^{N^k} time points and tape positions
$\rightsquigarrow$ use binary numbers with N^k digits

So X and Y in atoms like head(X, Y) are really lists of variables $X = x_1, \ldots, x_{N^k}$ and $Y = y_1, \ldots, y_{N^k}$, and the arity of head is $2 \cdot N^k$.

Todo: define predicates that capture the order of N^k-ary binary numbers

Preparing for a Long Computation

We need to encode 2^{N^k} time points and tape positions
$\leadsto$ use binary numbers with N^k digits

So X and Y in atoms like head(X, Y) are really lists of variables $X = x_1, \ldots, x_{N^k}$ and $Y = y_1, \ldots, y_{N^k}$, and the arity of head is $2 \cdot N^k$.

Todo: define predicates that capture the order of N^k-ary binary numbers

For each arity $i \in \{1, \ldots, N^k\}$, we use predicates:

- succ$^i(X, Y)$: the $X + 1 = Y$, where X and Y are i-ary numbers
- first$^i(X)$: X is the i-ary encoding of 0
- last$^i(X)$: X is the i-ary encoding of $2^i - 1$

Finally, we can define the actual order for $i = N^k$

- $\leq^i (X, Y)$: the $X < Y$, where X and Y are i-ary numbers

Defining a Long Chain

We can define $\mathsf{succ}^i(X, Y)$, $\mathsf{first}^i(X)$, and $\mathsf{last}^i(X)$ as follows:

$$\mathsf{succ}^1(0,1) \quad \mathsf{first}^1(0) \quad \mathsf{last}^1(1)$$

$$\left.\begin{aligned}
\mathsf{succ}^{i+1}(0,X,0,Y) &\leftarrow \mathsf{succ}^i(X,Y) \\
\mathsf{succ}^{i+1}(1,X,1,Y) &\leftarrow \mathsf{succ}^i(X,Y) \\
\mathsf{succ}^{i+1}(0,X,1,Y) &\leftarrow \mathsf{last}^i(X) \wedge \mathsf{first}^i(Y) \\
\mathsf{first}^{i+1}(0,X) &\leftarrow \mathsf{first}^i(X) \\
\mathsf{last}^{i+1}(1,X) &\leftarrow \mathsf{last}^i(X)
\end{aligned}\right\} \begin{array}{l}\text{for } X = x_1, \ldots, x_i \\ \text{and } Y = y_1, \ldots, y_i \\ \text{lists of } i \text{ variables}\end{array}$$

Defining a Long Chain

We can define $\mathsf{succ}^i(X, Y)$, $\mathsf{first}^i(X)$, and $\mathsf{last}^i(X)$ as follows:

$$\mathsf{succ}^1(0, 1) \quad \mathsf{first}^1(0) \quad \mathsf{last}^1(1)$$

$$\left.\begin{aligned}
\mathsf{succ}^{i+1}(0, X, 0, Y) &\leftarrow \mathsf{succ}^i(X, Y) \\
\mathsf{succ}^{i+1}(1, X, 1, Y) &\leftarrow \mathsf{succ}^i(X, Y) \\
\mathsf{succ}^{i+1}(0, X, 1, Y) &\leftarrow \mathsf{last}^i(X) \wedge \mathsf{first}^i(Y) \\
\mathsf{first}^{i+1}(0, X) &\leftarrow \mathsf{first}^i(X) \\
\mathsf{last}^{i+1}(1, X) &\leftarrow \mathsf{last}^i(X)
\end{aligned}\right\} \begin{array}{l}\text{for } X = x_1, \ldots, x_i \\ \text{and } Y = y_1, \ldots, y_i \\ \text{lists of } i \text{ variables}\end{array}$$

Now for $M = N^k$, we define $\leq^M(X, Y)$ as the reflexive, transitive closure of $\mathsf{succ}^M(X, Y)$:

$$\leq^M(X, X) \leftarrow$$

$$\leq^M(X, Z) \leftarrow \leq^M(X, Y) \wedge \mathsf{succ}^M(Y, Z)$$

Initialising the Computation

We can now encode the initial configuration of the Turing Machine for an input word $\sigma_1 \cdots \sigma_n \in (\Sigma \setminus \{\llcorner\!\lrcorner\})^*$.

We write B_i for the binary encoding of a number i with $M = N^k$ digits, and $Y = y_1, \ldots, y_M$.

$$\begin{aligned} \mathsf{state}_{q_0}(B_0) & & \text{where } q_0 \text{ is the TM's initial state} \\ \mathsf{head}(B_0, B_0) & & \\ \mathsf{symbol}_{\sigma_i}(B_0, B_i) & & \text{for all } i \in \{1, \ldots, n\} \\ \mathsf{symbol}_{\llcorner\!\lrcorner}(B_0, Y) & \leftarrow \leq^M(B_{n+1}, Y) & \end{aligned}$$

TM Transition and Acceptance Rules

For each transition $\langle q, \sigma, q', \sigma', d\rangle \in \Delta$, we add rules:

$$\mathsf{symbol}_{\sigma'}(X', Y) \leftarrow \mathsf{succ}^M(X, X') \wedge \mathsf{head}(X, Y) \wedge \mathsf{symbol}_{\sigma}(X, Y) \wedge \mathsf{state}_q(X)$$
$$\mathsf{state}_{q'}(X') \leftarrow \mathsf{succ}^M(X, X') \wedge \mathsf{head}(X, Y) \wedge \mathsf{symbol}_{\sigma}(X, Y) \wedge \mathsf{state}_q(X)$$

Similar rules are used for inferring the new head position (depending on d)

TM Transition and Acceptance Rules

For each transition $\langle q, \sigma, q', \sigma', d\rangle \in \Delta$, we add rules:

$$\mathsf{symbol}_{\sigma'}(X', Y) \leftarrow \mathsf{succ}^M(X, X') \wedge \mathsf{head}(X, Y) \wedge \mathsf{symbol}_{\sigma}(X, Y) \wedge \mathsf{state}_q(X)$$

$$\mathsf{state}_{q'}(X') \leftarrow \mathsf{succ}^M(X, X') \wedge \mathsf{head}(X, Y) \wedge \mathsf{symbol}_{\sigma}(X, Y) \wedge \mathsf{state}_q(X)$$

Similar rules are used for inferring the new head position (depending on d)

Further rules ensure the preservation of unaltered tape cells:

$$\begin{aligned}\mathsf{symbol}_{\sigma}(X', Y) \leftarrow\ & \mathsf{succ}^M(X, X') \wedge \mathsf{symbol}_{\sigma}(X, Y) \wedge \\ & \mathsf{head}(X, Z) \wedge \mathsf{succ}^M(Z, Z') \wedge \leq^M(Z', Y)\end{aligned}$$

$$\begin{aligned}\mathsf{symbol}_{\sigma}(X', Y) \leftarrow\ & \mathsf{succ}^M(X, X') \wedge \mathsf{symbol}_{\sigma}(X, Y) \wedge \\ & \mathsf{head}(X, Z) \wedge \mathsf{succ}^M(Z', Z) \wedge \leq^M(Y, Z')\end{aligned}$$

TM Transition and Acceptance Rules

For each transition $\langle q, \sigma, q', \sigma', d \rangle \in \Delta$, we add rules:

$$\mathsf{symbol}_{\sigma'}(X', Y) \leftarrow \mathsf{succ}^M(X, X') \wedge \mathsf{head}(X, Y) \wedge \mathsf{symbol}_\sigma(X, Y) \wedge \mathsf{state}_q(X)$$
$$\mathsf{state}_{q'}(X') \leftarrow \mathsf{succ}^M(X, X') \wedge \mathsf{head}(X, Y) \wedge \mathsf{symbol}_\sigma(X, Y) \wedge \mathsf{state}_q(X)$$

Similar rules are used for inferring the new head position (depending on d)

Further rules ensure the preservation of unaltered tape cells:

$$\begin{aligned}\mathsf{symbol}_\sigma(X', Y) \leftarrow{} & \mathsf{succ}^M(X, X') \wedge \mathsf{symbol}_\sigma(X, Y) \wedge \\ & \mathsf{head}(X, Z) \wedge \mathsf{succ}^M(Z, Z') \wedge \leq^M(Z', Y) \\ \mathsf{symbol}_\sigma(X', Y) \leftarrow{} & \mathsf{succ}^M(X, X') \wedge \mathsf{symbol}_\sigma(X, Y) \wedge \\ & \mathsf{head}(X, Z) \wedge \mathsf{succ}^M(Z', Z) \wedge \leq^M(Y, Z')\end{aligned}$$

The TM accepts if it ever reaches the accepting state q_{acc}:

$$\mathsf{accept}() \leftarrow \mathsf{state}_{q_{\mathsf{acc}}}(X)$$

Hardness Results

Lemma

A deterministic TM accepts an input in $\textsc{Time}(2^{n^k})$ if and only if the Datalog program defined above entails the fact accept().

We obtain $\textsc{ExpTime}$-hardness of Datalog query answering:

- The decision problem of any language in $\textsc{ExpTime}$ can be solved by a deterministic TM in $\textsc{Time}(2^{n^k})$ for some constant k
- In particular, there are $\textsc{ExpTime}$-hard languages $\mathcal{L}$ with suitable deterministic TM $\mathcal{M}$ and constant k
- For any input word w, we can reduce acceptance of w by $\mathcal{M}$ in $\textsc{Time}(2^{n^k})$ to entailment of accept() by a Datalog program $P(w, \mathcal{M}, k)$
- $P(w, \mathcal{M}, k)$ is polynomial in k and the size of $\mathcal{M}$ and w (in fact, it can be constructed in logarithmic space)

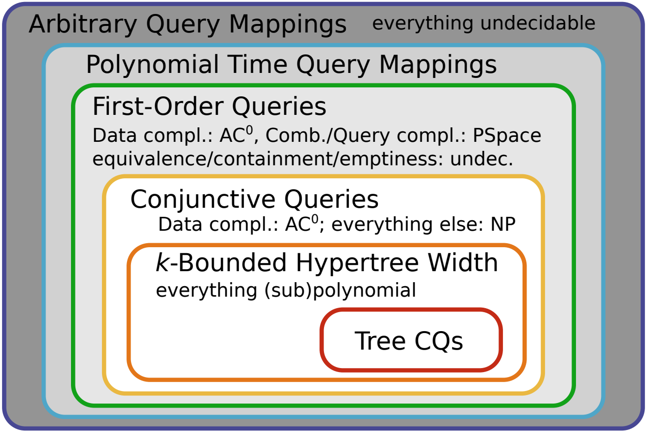
ExpTime-Hardness: Notes

Some further remarks on our construction:

- The constructed program does not use EDB predicates
  $\leadsto$ database can be empty
- Therefore, hardness extends to query complexity
- Using a fixed (very small) database, we could have avoided the use of constants
- We used IDB predicates of unbounded arity
  $\leadsto$ they are essential for the claimed hardness

The Big Picture

Where does Datalog fit in this picture?



Expressivity of Datalog

Datalog is P-complete for data complexity:

- Entailments can be computed in polynomial time with respect to the size of the input database $\mathcal{I}$
- There is a Datalog program P, such that all problems that can be solved in polynomial time can be reduced to the question whether P entails some fact over a database $\mathcal{I}$ that can be computed in logarithmic space.

$\leadsto$ So Datalog can solve all polynomial problems?

Expressivity of Datalog

Datalog is P-complete for data complexity:

- Entailments can be computed in polynomial time with respect to the size of the input database $\mathcal{I}$
- There is a Datalog program P, such that all problems that can be solved in polynomial time can be reduced to the question whether P entails some fact over a database $\mathcal{I}$ that can be computed in logarithmic space.

$\leadsto$ So Datalog can solve all polynomial problems?

No, it can't. Many problems in P that cannot be solved in Datalog:

- $\textsc{Parity}$: Is the number of elements in the database even?
- $\textsc{Connectivity}$: Is the input database a connected graph?
- Is the input database a chain (or linear order)?
- ...

Datalog Expressivity and Homomorphisms

How can we know that something is not expressible in Datalog?

A useful property: Datalog is "closed under homomorphisms"

Theorem

Consider a Datalog program P, an atom A, and databases $\mathcal{I}$ and $\mathcal{J}$. If P entails A over $\mathcal{I}$, and there is a homomorphism μ from $\mathcal{I}$ to $\mathcal{J}$, then $\mu(P)$ entails $\mu(A)$ over $\mathcal{J}$.

(By $\mu(P)$ and $\mu(A)$ we mean the program/atom obtained by replacing constants in P and A, respectively, by their μ-images.)

Proof (sketch):

- Closure under homomorphism holds for conjunctive queries
- Single rule applications are like conjunctive queries
- We can show the claim for all $T^i_{P,\mathcal{I}}$ by induction on i

Limits of Datalog Expressiveness

Closure under homomorphism shows many limits of Datalog

Special case: there is a homomorphism from $\mathcal{I}$ to $\mathcal{J}$ if $\mathcal{I} \subset \mathcal{J}$
$\leadsto$ Datalog entailments always remain true when adding more facts

Limits of Datalog Expressiveness

Closure under homomorphism shows many limits of Datalog

Special case: there is a homomorphism from $\mathcal{I}$ to $\mathcal{J}$ if $\mathcal{I} \subset \mathcal{J}$
$\leadsto$ Datalog entailments always remain true when adding more facts

- $\textsc{Parity}$ can not be expressed
- $\textsc{Connectivity}$ can not be expressed
- It cannot be checked if the input database is a chain
- ...

However this criterion is not sufficient!

Datalog cannot even express all polynomial time query mappings that are closed under homomorphism

Capturing PTIME in Datalog

How could we extend Datalog to capture all query mappings in P?
$\rightsquigarrow$ semipositive Datalog on an ordered domain

Definition

Semipositive Datalog, denoted Datalog$^{\perp}$, extends Datalog by allowing negated EDB atoms in rule bodies.
Datalog (semipositive or not) with a successor ordering assumes that there are special EDB predicates succ (binary), first and last (unary) that characterise a total order on the active domain.

Semipositive Datalog with a total order corresponds to standard Datalog on extended databases:

- For each ground fact $r(c_1, \ldots, c_n)$ with $\mathcal{I} \not\models r(c_1, \ldots, c_n)$, add a new fact $\bar{r}(c_1, \ldots, c_n)$ to $\mathcal{I}$, using a new EDB predicate $\bar{r}$
- Replace all uses of $\neg r(t_1, \ldots, t_n)$ in P by $\bar{r}(t_1, \ldots, t_n)$
- Define extensions for the EDB predicates succ, first and last to characterise some (arbitrary) total order on the active domain.

A $\mathrm{P}\textsc{Time}$ Capturing Result

Theorem

A Boolean query mapping defines a language in P if and only if it can be described by a query in semipositive Datalog with a successor ordering.

Example: expressing $\textsc{Connectivity}$ for binary graphs

$$
\begin{aligned}
\mathsf{Reachable}(x,x) &\leftarrow \\
\mathsf{Reachable}(x,y) &\leftarrow \mathsf{Reachable}(y,x) \\
\mathsf{Reachable}(x,z) &\leftarrow \mathsf{Reachable}(x,y) \wedge \mathsf{edge}(y,z) \\
\mathsf{Connected}(x) &\leftarrow \mathsf{min}(x) \\
\mathsf{Connected}(y) &\leftarrow \mathsf{Connected}(x) \wedge \mathsf{succ}(x,y) \wedge \mathsf{Reachable}(x,y) \\
\mathsf{Accept}() &\leftarrow \mathsf{max}(x) \wedge \mathsf{Connected}(x)
\end{aligned}
$$

Datalog Expressivity: Summary

The $\textsc{PTime}$ capturing result is a powerful and exhaustive characterisation for semipositive Datalog with a successor ordering

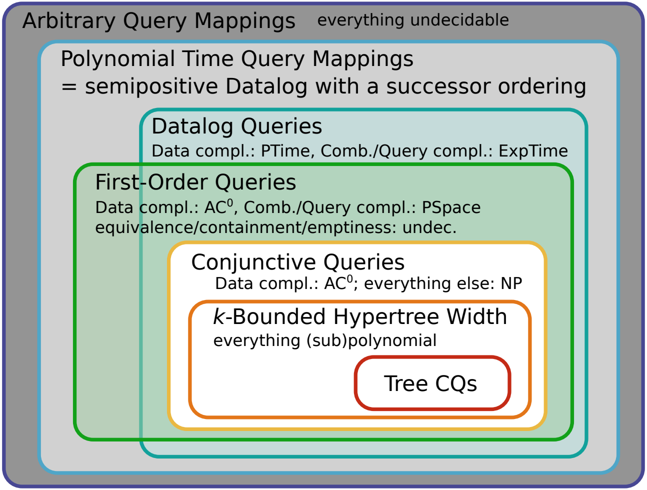
Situation much less clear for other variants of Datalog (as of 2015):

- What exactly can we express in Datalog without EDB negation and/or successor ordering?
  - Does a weaker language suffice to capture $\textsc{PTime}$? $\leadsto$ No!
  - When omitting negation, do we get query mappings closed under homomorphism? No![1]
- How about query mappings in $\textsc{PTime}$ that are closed under homomorphism?
  - Does plain Datalog capture these? $\leadsto$ No!
  - Does Datalog with successor ordering capture these? $\leadsto$ No![2]

[1] Counterexample on previous slide

[2] [S. Rudolph, personal communication, 2015]

The Big Picture



Note: languages that capture the same query mappings must have the same data complexity, but may differ in combined or in query complexity

Summary and Outlook

Non-recursive Datalog can express UCQs

Datalog is more complex than FO query answering:

- $\textsc{ExpTime}$-complete for query and combined complexity
- P-complete for data complexity

Datalog cannot express all query mappings in P
but semipositive Datalog with a successor ordering can

Next topics:

- Query containment for Datalog
- Implementation techniques for Datalog