TECHNISCHE UNIVERSITÄT DRESDEN

DATABASE THEORY

Lecture 13: Datalog Expressivity and Containment

David Carral

Knowledge-Based Systems

TU Dresden, 29th May 2019

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 are 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?

Datalog is more complex than FO query answering:

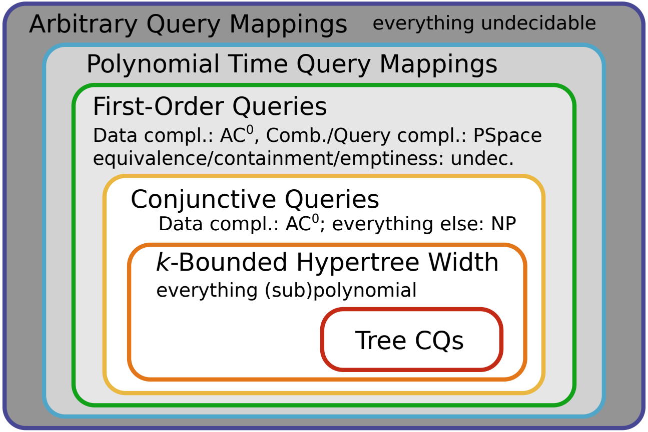
- ExpTime-complete for query and combined complexity
- P-complete for data complexity

Next question: Is Datalog also more expressive than FO query answering?

Expressivity

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?

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

- Parity: Is the number of elements in the database even?
- 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 13.1: 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

- $\textsc{Parity}$ cannot be expressed
- $\textsc{Connectivity}$ cannot be expressed
- It cannot be checked if the input database is a chain
- Many FO queries with negation cannot be expressed (e.g., $\neg p(a)$)
- ...

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 13.2: 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 an extended version of the given database:

- 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 PTime Capturing Result

Theorem 13.3: 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 13.4: We can express $\textsc{Connectivity}$ for binary graphs as follows:

$$\begin{aligned}
\text{Reachable}(x,x) &\leftarrow \\
\text{Reachable}(x,y) &\leftarrow \text{Reachable}(y,x) \\
\text{Reachable}(x,z) &\leftarrow \text{Reachable}(x,y) \wedge \text{edge}(y,z) \\
\text{Connected}(x) &\leftarrow \text{first}(x) \\
\text{Connected}(y) &\leftarrow \text{Connected}(x) \wedge \text{succ}(x,y) \wedge \text{Reachable}(x,y) \\
\text{Accept}() &\leftarrow \text{last}(x) \wedge \text{Connected}(x)
\end{aligned}$$

Datalog Expressivity: Summary

The 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 2018):

- What exactly can we express in Datalog without EDB negation and/or successor ordering?
  - Does a weaker language suffice to capture PTime? $\leadsto$ No!
  - When omitting negation, do we get query mappings closed under homomorphism? No![1]
- How about query mappings in PTime that are closed under homomorphism?
  - Does plain Datalog capture these? $\leadsto$ No![2]
  - Does Datalog with successor ordering capture these? $\leadsto$ No![3]

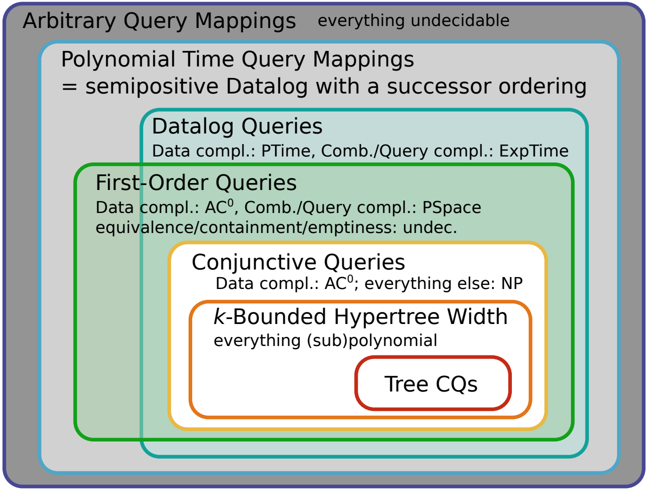
---

[1] Counterexample on previous slide

[2] [A. Dawar, S. Kreutzer, ICALP 2008]

[3] [S. Rudolph, M. Thomazo, IJCAI 2016]: "We are somewhat baffled by this result: in order to express queries which satisfy the strongest notion of monotonicity, one cannot dispense with negation, the epitome of non-monotonicity."

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

Datalog Containment

Datalog Implementation and Optimisation

How can Datalog query answering be implemented?
How can Datalog queries be optimised?

Recall: static query optimisation

- Query equivalence
- Query emptiness
- Query containment

$\rightsquigarrow$ all undecidable for FO queries, but decidable for (U)CQs

Learning from CQ Containment?

How did we manage to decide the question $Q_1 \stackrel{?}{\sqsubseteq} Q_2$ for conjunctive queries Q_1 and Q_2?

Key ideas were:

- We want to know if all situations where Q_1 matches are also matched by Q_2.
- We can simply view Q_1 as a database $\mathcal{I}_{Q_1}$: the most general database that Q_1 can match to
- Containment $Q_1 \stackrel{?}{\sqsubseteq} Q_2$ holds if Q_2 matches the database $\mathcal{I}_{Q_1}$.

$\leadsto$ decidable in NP

A CQ $Q[x_1, \ldots, x_n]$ can be expressed as a Datalog query with a single rule
Ans$(x_1, \ldots, x_n) \leftarrow Q$
$\leadsto$ Could we apply a similar technique to Datalog?

Checking Rule Entailment

The containment decision procedure for CQs suggests a procedure for single Datalog rules:

- Consider a Datalog program P and a rule $H \leftarrow B_1 \wedge \ldots \wedge B_n$.
- Define a database $\mathcal{I}_{B_1 \wedge \ldots \wedge B_n}$ as for CQs:
  - For every variable x in $H \leftarrow B_1 \wedge \ldots \wedge B_n$,
    we introduce a fresh constant c_x, not used anywhere yet
  - We define H^c to be the same as H but with each variable x replaced by c_x;
    similarly we define B_i^c for each $1 \leq i \leq n$
  - The database $\mathcal{I}_{B_1 \wedge \ldots \wedge B_n}$ contains exactly the facts B_i^c $(1 \leq i \leq n)$
- Now check if $H^c \in T_P^\infty(\mathcal{I}_{B_1 \wedge \ldots \wedge B_n})$:
  - If no, then there is a database on which $H \leftarrow B_1 \wedge \ldots \wedge B_n$
    produces an entailment that P does not produce.
  - If yes, then $P \models H \leftarrow B_1 \wedge \ldots \wedge B_n$

Example: Rule Entailment

Let P be the program

$$\text{Ancestor}(x, y) \leftarrow \text{parent}(x, y)$$
$$\text{Ancestor}(x, z) \leftarrow \text{parent}(x, y) \wedge \text{Ancestor}(y, z)$$

and consider the rule $\text{Ancestor}(x, z) \leftarrow \text{parent}(x, y) \wedge \text{parent}(y, z)$.

Then $\mathcal{I}_{\text{parent}(x,y)\wedge\text{parent}(y,z)} = \{\text{parent}(c_x, c_y), \text{parent}(c_y, c_z)\}$ (abbreviate as $\mathcal{I}$)
We can compute $T_P^\infty(\mathcal{I})$:

$$T_P^0(\mathcal{I}) = \mathcal{I}$$
$$T_P^1(\mathcal{I}) = \{\text{Ancestor}(c_x, c_y), \text{Ancestor}(c_y, c_z)\} \cup \mathcal{I}$$
$$T_P^2(\mathcal{I}) = \{\text{Ancestor}(c_x, c_z) \cup T_P^1(\mathcal{I})$$
$$T_P^3(\mathcal{I}) = T_P^2(\mathcal{I}) = T_P^\infty(\mathcal{I})$$

Therefore, $\text{Ancestor}(x, z)^c = \text{Ancestor}(c_x, c_z) \in T_P^\infty(\mathcal{I})$,
so P entails $\text{Ancestor}(x, z) \leftarrow \text{parent}(x, y) \wedge \text{parent}(y, z)$.

Deciding Datalog Containment?

Idea for two Datalog programs P_1 and P_2:

- If $P_2 \models P_1$, then every entailment of P_1 is also entailed by P_2
- In particular, this means that P_1 is contained in P_2
- We have $P_2 \models P_1$ if $P_2 \models H \leftarrow B_1 \wedge \ldots \wedge B_n$ for every rule $H \leftarrow B_1 \wedge \ldots \wedge B_n \in P_1$
- We can decide $P_2 \models H \leftarrow B_1 \wedge \ldots \wedge B_n$.

Can we decide Datalog containment this way?

$\leadsto$ No! In fact, Datalog containment is undecidable. What's wrong?

Implication Entailment vs. Datalog Entailment

P_1 :

$\mathsf{A}(x, y) \leftarrow \mathsf{parent}(x, y)$

$\mathsf{A}(x, z) \leftarrow \mathsf{parent}(x, y) \wedge \mathsf{A}(y, z)$

P_2 :

$\mathsf{B}(x, y) \leftarrow \mathsf{parent}(x, y)$

$\mathsf{B}(x, z) \leftarrow \mathsf{parent}(x, y) \wedge \mathsf{B}(y, z)$

Consider the Datalog queries $\langle A, P_1 \rangle$ and $\langle B, P_2 \rangle$:

- Clearly, $\langle A, P_1 \rangle$ and $\langle B, P_2 \rangle$ are equivalent (and mutually contained in each other).
- However, P_2 entails no rule of P_1 and P_1 entails no rule of P_2.

$\leadsto$ IDB predicates do not matter in Datalog, but predicate names matter in first-order implications

Datalog as Second-Order Logic

Datalog is a fragment of second-order logic:
IDB predicates are like variables that can take any set of tuples as value!

Example 13.5: The previous query $\langle A, P_1 \rangle$ can be expressed by the formula

$$\forall \mathsf{A}. \left(\begin{array}{lll} \forall x, y.\mathsf{A}(x, y) & \leftarrow \mathsf{parent}(x, y) & \wedge \\ \forall x, y, z.\mathsf{A}(x, z) & \leftarrow \mathsf{parent}(x, y) \wedge \mathsf{A}(y, z) & \end{array} \right) \rightarrow \mathsf{A}(v, w)$$

- This is a formula with two free variables v and w.
  $\rightsquigarrow$ query with two result variables
- Intuitive semantics: "$\langle c, d \rangle$ is a query result if $\mathsf{A}(c, d)$ holds for all possible values of A that satisfy the rules"
  $\rightsquigarrow$ Datalog semantics in other words

We can express any Datalog query like this, with one second-order variable per IDB predicate.

First-Order vs. Second-Order Logic

A Datalog program looks like a set of first-order implications,
but it has a second-order semantics

We have already seen that Datalog can express things that are impossible to express in FO queries – that's why we introduced it![1]

Consequences for query optimisation:

- Entailment between sets of first-order implications is decidable (shown above)
- Containment between Datalog queries is not decidable (shown next)

[1] Possible confusion when comparing of FO and Datalog: entailments of first-order implications agree with answers of Datalog queries, so it seems we can break the FO locality restrictions; but query answering is model checking not entailment; FO model checking is much weaker than second-order model checking

Undecidability of Datalog Query Containment

A classical undecidable problem:

Post Correspondence Problem:

- Input: two lists of words $\alpha_1, \ldots, \alpha_n$ and $\beta_1, \ldots, \beta_n$
- Output: "yes" if there is a sequence of indices $i_1, i_2, i_3, \ldots, i_m$ such that $\alpha_{i_1}\alpha_{i_2}\alpha_{i_3} \cdots \alpha_{i_m} = \beta_{i_1}\beta_{i_2}\beta_{i_3} \cdots \beta_{i_m}$.

$\leadsto$ we will reduce PCP to Datalog containment

We need to define Datalog programs that work on databases that encode words:

- We represent words by chains of binary predicates
- Binary EDB predicates represent letters
- For each letter σ, we use a binary EDB predicate letter[σ]
- We assume that the words α_i have the form $a^i_1 \cdots a^i_{|\alpha_i|}$, and that the words β_i have the form $b^i_1 \cdots b^i_{|\beta_i|}$

Solving PCP with Datalog Containment

A program P_1 to recognise potential PCP solutions.

Rules to recognise words α_i and β_i for every $i \in \{1, \ldots, m\}$:

$$\mathsf{A}_i(x_0, x_{|\alpha_i|}) \leftarrow \mathsf{letter}[a^i_1](x_0, x_1) \wedge \ldots \wedge \mathsf{letter}[a^i_{|\alpha_i|}](x_{|\alpha_i|-1}, x_{|\alpha_i|})$$

$$\mathsf{B}_i(x_0, x_{|\beta_i|}) \leftarrow \mathsf{letter}[b^i_1](x_0, x_1) \wedge \ldots \wedge \mathsf{letter}[b^i_{|\beta_i|}](x_{|\beta_i|-1}, x_{|\beta_i|})$$

Rules to check for synchronised chains (for all $i \in \{1, \ldots, m\}$):

$$\mathsf{PCP}(x, y_1, y_2) \leftarrow A_i(x, y_1) \wedge B_i(x, y_2)$$

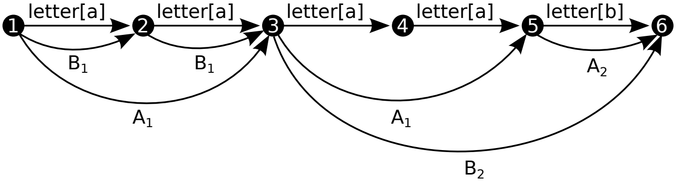
$$\mathsf{PCP}(x, z_1, z_2) \leftarrow \mathsf{PCP}(x, y_1, y_2) \wedge A_i(y_1, z_1) \wedge B_i(y_2, z_2)$$

$$\mathsf{Accept}() \leftarrow \mathsf{PCP}(x, z, z)$$

Solving PCP with Datalog Containment (2)

Example: $\alpha_1 = aa$, $\beta_1 = a$, $\alpha_2 = b$, $\beta_2 = aab$

Example for an intended database and least model (selected parts):



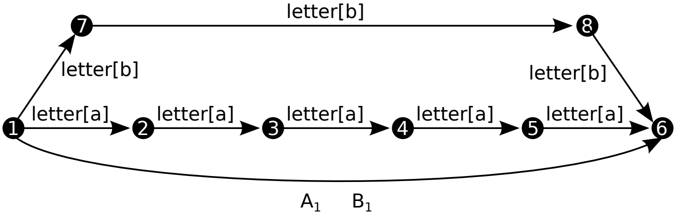
Additional IDB facts that are derived (among others):

$$\text{PCP}(1,3,2) \quad \text{PCP}(1,5,3) \quad \text{PCP}(1,6,6) \quad \text{Accept}()$$

Solving PCP with Datalog Containment (3)

Example: $\alpha_1 = aaaaa, \beta_1 = bbb$

Problem: P_1 also accepts some unintended cases



Additional IDB facts that are derived:

$$\mathsf{PCP}(1,6,6) \quad \mathsf{Accept}()$$

Solving PCP with Datalog Containment (4)

Solution: specify a program P_2 that recognises all unwanted cases

P_2 consists of the following rules (for all letters σ, σ'):

$$\mathsf{EP}(x,x) \leftarrow$$

$$\mathsf{EP}(y_1,y_2) \leftarrow \mathsf{EP}(x_1,x_2) \wedge \mathsf{letter}[\sigma](x_1,y_1) \wedge \mathsf{letter}[\sigma](x_2,y_2)$$

$$\mathsf{Accept}() \leftarrow \mathsf{EP}(x_1,x_2) \wedge \mathsf{letter}[\sigma](x_1,y_1) \wedge \mathsf{letter}[\sigma'](x_2,y_2) \qquad \sigma \neq \sigma'$$

$$\mathsf{NEP}(x_1,y_2) \leftarrow \mathsf{EP}(x_1,x_2) \wedge \mathsf{letter}[\sigma](x_2,y_2)$$

$$\mathsf{NEP}(x_1,y_2) \leftarrow \mathsf{NEP}(x_1,x_2) \wedge \mathsf{letter}[\sigma](x_2,y_2)$$

$$\mathsf{Accept}() \leftarrow \mathsf{NEP}(x,x)$$

Intuition:

- EP defines equal paths (forwards, from one starting point)
- NEP defines paths of different length (from one starting point to the same end point)

$\rightsquigarrow$ P_2 accepts all databases with distinct parallel paths

Solving PCP with Datalog Containment (5)

What does it mean if $\langle \text{Accept}, P_1 \rangle$ is contained in $\langle \text{Accept}, P_2 \rangle$?

The following are equivalent:

- All databases with potential PCP solutions also have distinct parallel paths.
- Databases without distinct parallel paths have no PCP solutions.
- Linear databases (words) have no PCP solutions.
- The answer to the PCP is "no".

$\leadsto$ If we could decide Datalog containment, we could decide PCP

Theorem 13.6: Containment and equivalence of Datalog queries are undecidable.

(Note that emptiness of Datalog queries is trivial)

Summary and Outlook

Datalog cannot express all query mappings in P ...

... but semipositive Datalog with a successor ordering can

First-order rule entailment is decidable ...

... but Datalog containment is not.

Next question:

- How can we implement Datalog in practice?