Review: Datalog

A rule-based recursive query language

```
father(alice, bob)
mother(alice, carla)
Parent(x, y) ← father(x, y)
Parent(x, y) ← mother(x, y)
SameGeneration(x, x)
SameGeneration(x, y) ← Parent(x, v) ∧ Parent(y, w) ∧ SameGeneration(v, w)
```

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 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 $I$.
- 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 $I$ that can be computed in logarithmic space.

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 $I$ and $J$. If $P$ entails $A$ over $I$, and there is a homomorphism $\mu$ from $I$ to $J$, then $\mu(P)$ entails $\mu(A)$ over $J$.

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

Proof (sketch):
- Closure under homomorphism holds for conjunctive queries
- Single rule applications are like conjunctive queries
- We can show the claim for all $T_{i,P}$ by induction on $i$.

Limtes of Datalog Expressiveness

Closure under homomorphism shows many limits of Datalog

**Special case:** there is a homomorphism from $I$ to $J$ if $I \subseteq J$.

- Datalog entailments always remain true when adding more facts
- Parity cannot be expressed
- 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?

\[ \text{semipositive Datalog on an ordered domain} \]

**Definition 13.2:** Semipositive Datalog, denoted $\text{Datalog}^+$, 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 $\text{succ}$ (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 $I \not\models r(c_1, \ldots, c_n)$, add a new fact $\bar{r}(c_1, \ldots, c_n)$ to $I$, 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 $\text{succ}$, 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 **Connectivity** for binary graphs as follows:

- `Reachable(x, x) :-` 
- `Reachable(x, y) :- Reachable(y, x)` 
- `Reachable(x, z) :- Reachable(y, z)` 
- `Connected(x) :- first(x)` 
- `Connected(x) :- Connected(x) ∧ succ(x, y) ∧ Reachable(x, y)` 
- `Accept() :- last(x) ∧ Connected(x)`

---

**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? \(\sim\) **No!**
  - When omitting negation, do we get query mappings closed under homomorphism? \(\sim\) **No!**
- How about query mappings in PTime that are closed under homomorphism?
  - Does plain Datalog capture these? \(\sim\) **No!**
  - Does Datalog with successor ordering capture these? \(\sim\) **No!**

\(^1\)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**

- **Tree CQs**
- **k-Bounded Hypertree Width**
- **Conjunctive Queries**
- **First-Order Queries**
- **DatalogQueries**
  - Data compl.: PTime, Comb./Query compl.: ExpTime
- **Polyominal Time Query Mappings**
  - everything undecidable
- **Data complexity:** AC\(^0\); everything else: NP
  - Equivalence/containment/emptiness: undec.

---

**Datalog Containment**

Note: languages that capture the same query mappings must have the same data complexity, but may differ in combined or in query complexity.
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
\sim all undecidable for FO queries, but decidable for (U)CQs

Checking Rule Entailment

The containment decision procedure for CQs suggests a procedure for single Datalog rules:
- Consider a Datalog program $P$ and a rule $H \leftarrow B_1 \land \ldots \land B_n$.
- Define a database $I_{B_1, \ldots , B_n}$ as for CQs:
  - For every variable $x$ in $H \leftarrow B_1 \land \ldots \land B_n$, we introduce a fresh constant $c_x$, not used anywhere yet.
  - We define $H'$ to be the same as $H$ but with each variable $x$ replaced by $c_x$.
  - Similarly we define $B'_i$ for each $1 \leq i \leq n$.
- The database $I_{B_1, \ldots , B_n}$ contains exactly the facts $B'_i (1 \leq i \leq n)$
- Now check if $H' \in T^P_{\sigma}(I_{B_1, \ldots , B_n})$:
  - If no, then there is a database on which $H \leftarrow B_1 \land \ldots \land B_n$ produces an entailment that $P$ does not produce.
  - If yes, then $P \models H \leftarrow B_1 \land \ldots \land B_n$.

Learning from CQ Containment?

How did we manage to decide the question $Q_1 \subseteq Q_2$ 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 $I_{Q_1}$: the most general database that $Q_1$ can match to.
- Containment $Q_1 \subseteq Q_2$ holds if $Q_2$ matches the database $I_{Q_1}$.
\sim decidable in NP

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

Example: Rule Entailment

Let $P$ be the program
\begin{align*}
\text{Ancestor}(x, y) & \leftarrow \text{parent}(x, y) \\
\text{Ancestor}(x, z) & \leftarrow \text{parent}(x, y) \land \text{Ancestor}(y, z)
\end{align*}
and consider the rule $\text{Ancestor}(x, z) \leftarrow \text{parent}(x, y) \land \text{parent}(y, z)$.

Then $I_{\{ \text{parent}(x, y) \land \text{parent}(y, z) \}} \models \{ \text{parent}(c_x, c_y), \text{parent}(c_y, c_z) \}$ (abbreviate as $I$).

We can compute $T^P_{\sigma}(I)$:
\begin{align*}
T^P_{\sigma}(I) &= I \\
T^P_{\sigma}(I) &= \{ \text{Ancestor}(c_x, c_y), \text{Ancestor}(c_y, c_z) \} \cup I \\
T^P_{\sigma}(I) &= \{ \text{Ancestor}(c_x, c_y) \} \cup T^P_{\sigma}(I) \\
T^P_{\sigma}(I) &= T^P_{\sigma}(I) = T^P_{\sigma}(I)
\end{align*}

Therefore, $\text{Ancestor}(x, z) \in T^P_{\sigma}(I)$, so $P$ entails $\text{Ancestor}(x, z) \leftarrow \text{parent}(x, y) \land \text{parent}(y, z)$. 
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 $P_2 \models P_1$.

Can we decide Datalog containment this way?

No! In fact, Datalog containment is undecidable. What’s wrong?

Implication Entailment vs. Datalog Entailment

P_1:

\[
A(x, y) \leftarrow \text{parent}(x, y) \\
A(x, z) \leftarrow \text{parent}(x, y) \wedge A(y, z)
\]

P_2:

\[
B(x, y) \leftarrow \text{parent}(x, y) \\
B(x, z) \leftarrow \text{parent}(x, y) \wedge B(y, z)
\]

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 $P_2$.

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!

\[
\forall A. \left( \forall x, y. A(x, y) \leftarrow \text{parent}(x, y) \right) \wedge A(x, z) \leftarrow \text{parent}(x, y) \wedge A(y, z) \rightarrow A(y, w)
\]

- This is a formula with two free variables $v$ and $w$.
- $\forall x, y. A(x, y) \leftarrow \text{parent}(x, y)$
- Intuitive semantics: "$c, d$ is a query result if $A(c, d)$ holds for all possible values of $A$ that satisfy the rules"

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!

Consequences for query optimisation:

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

\begin{enumerate}
\item 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.
\end{enumerate}
Undecidability of Datalog Query Containment

**A classical undecidable problem:**

**Post Correspondence Problem:**
- Input: two lists of words $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$
- Output: "yes" if there is a sequence of indices $i_1, i_2, \ldots, i_n$ such that $a_{i_1}a_{i_2}\cdots a_{i_n} = b_{i_1}b_{i_2}\cdots b_{i_n}$.

\[ \sim \text{ we will reduce PCP to Datalog containment} \]

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 $\sigma$, we use a binary EDB predicate $\text{letter}[\sigma]$
- We assume that the words $a_i$ have the form $a_1' \cdots a_{m_i}'$ and that the words $b_i$ have the form $b_1 \cdots b_{n_i}$

Solving PCP with Datalog Containment

A program $P_1$ to recognise potential PCP solutions.

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

\[ A_i(x_0, y_0) \leftarrow \text{letter}[a'_i](x_0, x_1) \land \ldots \land \text{letter}[a'_{m_i}](x_0, y_{m_i}) \]
\[ B_i(x_0, y_0) \leftarrow \text{letter}[b'_i](x_0, x_1) \land \ldots \land \text{letter}[b'_{n_i}](y_0, y_{n_i}) \]

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

\[ \text{PCP}(x, y_1, y_2) \leftarrow A_i(x, y_1) \land B_i(x, y_2) \]
\[ \text{PCP}(x, z_1, z_2) \leftarrow \text{PCP}(x, y_1, y_2) \land A_i(y_1, z_1) \land B_i(z_2, z_2) \]
\[ \text{Accept}() \leftarrow \text{PCP}(x, z, z) \]

**Example:** $a_1 = aa, b_1 = a, a_2 = b, b_2 = aab$

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

![Diagram of a database with EDB predicates representing letters and chains of indices](image)

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 (4)

Solution: specify a program $P_2$ that recognises all unwanted cases

$P_2$ consists of the following rules (for all letters $\sigma, \sigma'$):

- $EP(x, x) \leftarrow$
- $EP(y_1, y_2) \leftarrow EP(x_1, x_2) \land \text{letter}(\sigma')(x_1, y_1) \land \text{letter}(\sigma')(x_2, y_2)$
- $\text{Accept}(\sigma) \leftarrow EP(x_1, x_2) \land \text{letter}(\sigma')(x_1, y_1) \land \text{letter}(\sigma')(x_2, y_2) \quad \sigma \neq \sigma'$
- $\text{NEP}(x_1, y_2) \leftarrow EP(x_1, x_2) \land \text{letter}(\sigma')(x_2, y_2)$
- $\text{NEP}(x_1, y_2) \leftarrow \text{NEP}(x_1, x_2) \land \text{letter}(\sigma')(x_2, y_2)$
- $\text{Accept}(\sigma) \leftarrow \text{NEP}(x, x)$

Intuition:
- $EP$ defines equal paths (forwards, from one starting point)
- $\text{NEP}$ defines paths of different length (from one starting point to the same end point)

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

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?