Complexity and Expressive Power of Datalog
A Datalog Program $P$ consists of a finite set of rules of form

$$A_0 \leftarrow A_1, \ldots, A_m \quad (m \geq 0),$$

where each $A_i$ is a positive atom of the form $r(t_1, \ldots, t_k)$ where each $t_i$ is a variable or a constant.

Two important settings

1. Datalog programs are “stand alone”. Program may contain variables and constants.

2. Datalog programs operate over factual databases. The database contains ground facts, no constants occur within the program. Distinction between EDB and IDB Predicates.
Example of stand-alone Datalog

- Datalog program:

```prolog
parent(X, Y) :- father(X, Y)
parent(X, Y) :- mother(X, Y)
ancestor(X, Y) :- parent(X, Y)
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y)
person(X) :-
father(john, mary) :-
father(joe, kurt) :-
mother(mary, joe) :-
mother(tina, kurt) :-
```
Datalog as a Query Language

- Datalog is used as a database query language

- In this context, a datalog program is evaluated over a database, which is a set of facts.

- Programs are composed of a “derived” part $P$ (defined predicates) and an “input part” $D_{in}$ (database facts): $P \cup D_{in}$

**Example:**

\[
\begin{align*}
\text{parent}(X, Y) & : - \text{father}(X, Y) \\
\text{parent}(X, Y) & : - \text{mother}(X, Y) \\
\text{ancestor}(X, Y) & : - \text{parent}(X, Y) \\
\text{ancestor}(X, Y) & : - \text{parent}(X, Z), \text{ancestor}(Z, Y) \\
\text{person}(X) & : - \\
\text{father}(john, mary) & : - \quad \text{father}(joe, kurt) : - \\
\text{mother}(mary, joe) & : - \quad \text{mother}(tina, kurt) : -
\end{align*}
\]

\[\begin{array}{c}
\text{defined part } P \\
\text{database part } D_{in}
\end{array}\]
Refined Notions of Datalog Complexity

- The **data complexity** is the complexity of checking whether \( D_{in} \cup P \models A \) when datalog programs \( P \) are fixed, while input databases \( D_{in} \) and ground atoms \( A \) are an input.

- The **program complexity** (also called expression complexity) is the complexity of checking whether \( D_{in} \cup P \models A \) when input databases \( D_{in} \) are fixed, while datalog programs \( P \) and ground atoms \( A \) are an input.

- The **combined complexity** is the complexity of checking whether \( D_{in} \cup P \models A \) when input databases \( D_{in} \), datalog programs \( P \), and ground atoms \( A \) are an input.
The semantics of a datalog program $P$ is defined by reduction to the propositional case (by “Grounding”)

- Let $P$ be a datalog program operating on a database $D$.
- Let $U_D$ be the universe of $D$ (usually the active universe, i.e., the set of all domain elements present in $D$).
- The *grounding* of a rule $r$, denoted $\text{ground}(r, D)$, is the set of all rules obtained from $r$ by all possible uniform substitutions of elements of $U_D$ for the variables in $r$. 

Semantics of Datalog as a Query Language
Semantics of Datalog

• For any datalog program $P$ and database $D$,

$$\text{ground}(P, D) = \bigcup_{r \in P} \text{ground}(r, D).$$

• If $S$ is a set of atoms then $\text{IDB}_P(S)$ denotes those facys of $S$ whose predicate symbol is an IDB predicate symbol of $P$.

• The semantics of $P$ is given by

$$\mathcal{M}_P : D \rightarrow \text{IDB}_P(T^\infty_{\text{ground}(P,D) \cup D}).$$
Examples /2

Program $P$:

```
parent(X, Y) :- father(X, Y)
parent(X, Y) :- mother(X, Y)
ancestor(X, Y) :- parent(X, Y)
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y)
person(X) :-
father(john, mary) :- father(joe, kurt) ←
mother(mary, joe) :- mother(tina, kurt) ←
```

$ground(P)$:

```
parent(john, john) :- father(john, john)
parent(john, mary) :- father(john, mary)

... ...

parent(john, john) :- mother(john, john)
parent(john, mary) :- mother(john, mary)

... ...

ancestor(john, john) :- parent(john, john)

... ...

father(john, mary) :- father(joe, kurt)
mother(mary, joe) :- mother(tina, kurt)
```

- Herbrand Universe: $john, mary, joe, kurt, tina$
- Herbrand Base: $person(john)$ $person(mary)$, …, $parent(john, john)$, $parent(john, mary)$, …
- $LM(P) = \{\text{father}(john, mary), \text{father}(joe, kurt), \text{mother}(mary, joe), \text{mother}(tina, kurt), \text{parent}(john, mary), \text{...}, \text{ancestor}(john, mary), \text{...}, \text{person}(john), \text{...} \text{person}(tina), \text{...}\}$
Complexity of Datalog Programs

• For Datalog programs, both “$A \in \text{lm}(P)$” is decidable, similarly “$A \in \text{lm}(P \cup D)$” in case $P$ operates on a database $D$.

• **Reason:** $\text{Ground}(P)$ is finite (as $U_P$, $B_P$ are finite)

  Effective reduction to Propositional Logic Programming is possible:
  
  – Generate $\text{Ground}(P)$
  
  – Decide whether $A \in \text{lm}(\text{Ground}(P))$

• **Questions:**
  
  – What is the complexity of this algorithm? (Key: How expensive is computing $\text{Ground}(P)$?)
  
  – Is this the best algorithm to decide $A \in \text{lm}(P)$?
Complexity of Grounding Strategy

- Given $P, D$, the number of rules in $\text{ground}(P, D)$ is bounded by

$$|P| \times \#\text{consts}(D)^{v_{\text{max}}}$$

- $v_{\text{max}} \geq 1$ is the maximum number of different variables in any rule $r \in P$
- $\#\text{consts}(P) = |U_D|$ is the number of constants in $D$ (ass.: $|U_D| > 0$).

- $\text{ground}(P, D)$ can be naively generated in time

$$O(|P| \times \#\text{consts}(D)^{v_{\text{max}}}) = O(2^{\log |P| + v_{\text{max}} \times \log \#\text{consts}(D)}) =$$

$$O(2^{p(\|P \cup D\|)}),$$

where $p(\ldots)$ is some polynomial and $\|P \cup \|$ is the size of $P \cup D$.

- Therefore, $A \in \text{lm}(P \cup D)$ is decidable in exponential time.

- **Observation:** $\text{ground}(P \cup D)$ can be exponential in the size of $P$. 
• **Question:** Is $A \in \text{lm}(P)$ feasible in polynomial space?
**Theorem.** Given a positive Datalog program $P$ and a ground atom $A$, deciding whether $A \in lm(P)$ is EXPTIME-complete.

**Proof Sketch.**

- **Membership:** By reduction to propositional case (grounding)
- **Hardness:**
  - Adapt the propositional program $P(T, I, N)$ deciding acceptance of input $I$ for $T$ within $N$ steps, where $N = 2^m$, $m = n^k$ ($n = |I|$) to a datalog program $P_{dat}(T, I, N)$
  - Note: We can’t simply generate $P(T, I, N)$, since this program is exponentially large (and thus the reduction would not be polynomial!)
EXPTIME-Hardness of Datalog Programs

Main ideas for lifting $P(T, I, N)$ to $P_{dat}(T, I, N)$:

- Use predicates $\text{symbol}_\sigma(\vec{x}, \vec{y})$, $\text{cursor}(\vec{x}, \vec{y})$ and $\text{state}_s(\vec{x})$ instead of the propositional atoms $\text{symbol}_\sigma[X, Y]$, $\text{cursor}[X, Y]$ and $\text{state}_s[X]$ respectively.

- The time points $\tau$ and tape positions $\pi$ from 0 to $N - 1$ are encoded in binary, i.e. by $m$-ary tuples $t_\tau = \langle c_1, \ldots, c_m \rangle$, $c_i \in \{0, 1\}$, $i = 1, \ldots, m$, such that $0 = \langle 0, \ldots, 0 \rangle$, $1 = \langle 0, \ldots, 1 \rangle$, $\ldots$, $N - 1 = \langle 1, \ldots, 1 \rangle$.

- The functions $\tau + 1$ and $\pi + d$ are realized by means of the successor $\text{Succ}^m$ w.r.t. a linear order $\leq^m$ on $U^m$, built in $P$. 
Modification for Datalog-Complexity Hardness

Modify the program $P(T, I, N)$ as follows ($N = 2^m$, where $m = n^k$):

- Provide facts $\text{succ}^1(0, 1)$, $\text{first}^1(0)$, and $\text{last}^1(1)$ in $P$.

- Initialization facts:
  
  - Translate $symbol_{\sigma}[0, \pi]$ into rules
    
    $$symbol_{\sigma}(\vec{x}, \vec{t}) \leftarrow \text{first}^m(\vec{x}),$$
    
    where $\vec{t}$ represents the position $\pi$;
  
  - translate similarly the facts $\text{cursor}[0, 0]$ and $\text{state}_{s_0}[0]$.
  
  - Translate $symbol_{\underline{\cdot}}[0, \pi]$, where $|I| \leq \pi \leq N$, to the rule
    
    $$symbol_{\underline{\cdot}}(\vec{x}, \vec{y}) :- \text{first}^m(\vec{x}), \leq^m(\vec{t}, \vec{y})$$
    
    where $\vec{t}$ represents the number $|I|$.
• transition and inertia rules: For realizing $\tau + 1$ and $\pi + d$, use in the body atoms $\text{succ}^m(\vec{x}, \vec{x}')$.

**Example:**

\[
\text{symbol}_{\sigma'}[\tau + 1, \pi] : - \text{state}_s[\tau], \text{symbol}_\sigma[\tau, \pi], \text{cursor}[\tau, \pi]
\]

is translated into

\[
\text{symbol}_{\sigma'}(\vec{x}', \vec{y}) : - \text{state}_s(\vec{x}), \text{symbol}_\sigma(\vec{x}, \vec{y}), \text{cursor}(\vec{x}, \vec{y}), \text{succ}^m(\vec{x}, \vec{x}').
\]

• accept rules: translation is straightforward.
Defining $\text{succ}^m$ and $\leq_m$

- Add facts $\text{succ}^1(0, 1)$, $\text{first}^1(0)$, and $\text{last}^1(1)$.

- Inductively define $\text{succ}^{i+1}$:

  \[
  \text{succ}^{i+1}(z, \vec{x}, z', \vec{y}) \leftarrow \text{succ}^i(\vec{x}, \vec{y})
  \]

  \[
  \text{succ}^{i+1}(z, \vec{x}, z', \vec{y}) \leftarrow \text{succ}^1(z, z'), \text{last}^i(\vec{x}), \text{first}^i(\vec{y})
  \]

  \[
  \text{first}^{i+1}(z, \vec{x}) \leftarrow \text{first}^1(z), \text{first}^i(\vec{x})
  \]

  \[
  \text{last}^{i+1}(z, \vec{x}) \leftarrow \text{last}^1(z), \text{last}^i(\vec{x})
  \]

  (where $\vec{x} = x_1, \ldots, x_i, \vec{y} = y_1, \ldots, y_i$, and $\vec{z} = z_1, \ldots, z_i$.)

- The order $\leq^m$ is then easily defined by rules:

  \[
  \leq^m(\vec{x}, \vec{x}) \leftarrow
  \]

  \[
  \leq^m(\vec{x}, \vec{y}) \leftarrow \text{succ}^m(\vec{x}, \vec{z}), \leq^m(\vec{z}, \vec{y})
  \]

  (where $\vec{x} = x_1, \ldots, x_m, \vec{y} = y_1, \ldots, y_m$, and $\vec{z} = z_1, \ldots, z_m$.)
Concluding EXPTIME Hardness of Datalog

Let $P_{dat}(T, I, N)$ denote the datalog program with empty $edb$ described for $T$, $I$, and $N = 2^m$, $m = n^k$ (where $n = |I|$)

- $P_{dat}(T, I, N)$ is constructible from $T$ and $I$ in polynomial time (in fact, careful analysis shows feasibility in logarithmic space).

- $P_{dat}(T, I, N)$ has accept in its least model $\iff T$ accepts input $I$ within $N$ steps.

- Thus, the decision problem for any language in EXPTIME is reducible to deciding $P \models A$ for datalog program $P$ and fact $A$.

- Consequently, deciding $P \models A$ for a given datalog program $P$ and fact $A$ is EXPTIME-hard.
Program and Combined Complexity

- Clearly, combined complexity matches the problem $P \models A$ we considered so far $\Rightarrow$ Datalog is $\text{EXPTIME}$-complete w.r.t. combined complexity.

- As for program complexity, $\text{EXPTIME}$ is an upper bound.

- From the $\text{EXPTIME}$-hardness proof of $P \models A$, we can conclude that Datalog is $\text{EXPTIME}$-hard w.r.t. program complexity (take empty $D_{\text{in}}$).

- This can be sharpened to instances where program $P$ contains no constants (take $D_{\text{in}}$ to be $\text{succ}^1(0, 1)$, $\text{first}^1(0)$, and $\text{last}^1(1)$.)
Data Complexity

• For fixed $P$, the grounding $\text{ground}(D_{\text{in}} \cup P)$ has size polynomial in the size of $D_{\text{in}} \cup P (|P| \times \#\text{consts}(P)^{v_{\text{max}}}) = O(\|P\|^k)$ for some constant $k$).

• Moreover, $\text{ground}(D_{\text{in}} \cup P)$ can be easily generated in polynomial time.

• Therefore, $LM(D_{\text{in}} \cup P)$ is computable in polynomial time, and Datalog has polynomial-time data complexity.

• Furthermore, $P \models A$ is P-hard w.r.t. data complexity. This can be shown by proving that a fixed datalog program is able to act as a meta-interpreter for propositional logic programming.
A Datalog Meta-Interpreter for Propositional LP

**Note:** It is sufficient to interpret propositional logic programs whose clauses have at most 3 atoms in the rule bodies. In fact, we have shown that atom-inference from such programs is P-hard.

Encode a propositional LP as follows by a unary relation $T_0$ and a 4-ary relation $R$.

**Encoding of facts:** The fact “$p \leftarrow$” is encoded by the tuple $T(p)$.

**Encoding of rules:** A rule “$p \leftarrow q_1, q_2, q_3$” is encoded by the tuple $R(p, q_1, q_2, q_3)$. In case a rule has less than 3 atoms in its body, a body-atom can be repeated to get a tuple of length 4.

This encoding of a propositional logic program $P$, which is obviously feasible in logspace, is denoted by $D(P)$. 
The meta-interpreter $M$:

\[
T(X_0) : - R(X_0, X_1, X_2, X_3), T(X_1), T(X_2), T(X_3) \\
T(X) : - T_0(X)
\]

We have $P \models A$ iff $M \cup D(P) \models T(A)$.

Therefore the data complexity of datalog is PTIME-complete.
Semipositive Datalog (Datalog$\perp$)

So far, only positive atoms were allowed in rule bodies.

We are going to define a slight extension.

**Semipositive datalog programs:** EDB-atoms in rule bodies may occur both in positive and negated form. IDB-atoms cannot be negated.

Semantics: Obvious. Let $P$ be a semipositive program and $D$ a database. Add the complement relation $\overline{r}$ for each relation $r$ to the database, yielding $D^+$. Replace each atom $\neg r(x)$ in a rule body by $\overline{r}(x)$, yielding $P^+$. Then:

$$P(D) := P^+(D^+).$$

We denote semipositive datalog by datalog$\perp$. 
**Expressive Power of Semipositive Datalog**

A *successor ordering* of a structure consists of a successor relation $Succ$ on its universe and special relations $Min$ and $Max$ with the obvious meanings.

**THEOREM:** On structures provided with a successor ordering, $\text{datalog} \perp = \text{PTIME}$.

**PROOF SKETCH:**

We outline this for ordered graphs $G = (V, Succ, Min, Max, E)$.

We have to show that each PTIME property over such databases can be encoded by a semipositive datalog program.

Let us assume some property $\pi$ is computable in time $n^k$, where $n = |V|$. There must exist a Turing machine $T$ that does this job on a suitable binary encoding of $G$. Our intention is to simulate (the behaviour of) $T$ by a datalog $\perp$ program.
Ideas:

1.) We use vectors $\vec{x} = (x_1, \ldots, x_k)$ to encode time instants and workhead position (cell numbers). Here the arguments range over all domain elements from $V$, and hence we can encode exactly $|V|^k = n^k$ elements (or numbers) with each such vector.

2.) We define a vectorized successor relation $\text{succ}^k(\vec{x}, \vec{y})$ on vectors of length $k$ in a similar way as we did it before for binary vectors. (Iteratively, by defining $\text{succ}^i$ for $i = 0 \ldots k$, and based on the $\text{Min}$, $\text{Max}$, and $\text{Succ}$ predicates).
3.) We put the graph $G$ on the (datalog-simulated) input tape of the 
datalog-simulated Turing machine $T$ that runs in time $n^k$ by using the following 
binary encoding $\vec{e}$ of $E$. $E$ is encoded as a bit vector $\vec{e}$ of size $n^2$ such that 
$\vec{e}[i \cdot n + j]$ is 1 iff $(i, j) \in E$ and 0 otherwise.

This vector $\vec{e}$ is “put on the input tape” by the following 2 rules:

```
symbol_1(0^k, 0^{k-2}, X, Y) :- E(X, Y)
symbol_0(0^k, 0^{k-2}, X, Y) :- \neg E(X, Y)
```

4.) We simulate $T$ on this input in the usual way. Note that the resulting program is 
semipositive.

QED
Bibliography


