Combined Tractability of Query Evaluation via Tree Automata and Cycluits

Antoine Amarilli¹, Pierre Bourhis², Mikaël Monet¹,⁴, Pierre Senellart³,⁴
March 23th, 2017

¹LTCI, Télécom ParisTech, Université Paris-Saclay; Paris, France
²CNRS, CRIStAL, Inria Lille; Lille, France
³École normale supérieure, PSL Reasearch University; Paris, France
⁴Inria Paris; Paris, France
Conjunctive query $Q$ on relational instance $I$
Conjunctive query $Q$ on relational instance $I$

Complexity: NP-complete in combined, PTIME in data
Conjunctive query $Q$ on relational instance $I$

Complexity: NP-complete in combined, PTIME in data

- PTIME is not enough! We want $O(|Q| \times |I|)$
Conjunctive query $Q$ on relational instance $I$

Complexity: NP-complete in combined, PTIME in data

- PTIME is not enough! We want $O(|Q| \times |I|)$
- More elaborate tasks? Counting, probabilistic evaluation, etc.
Conjunctive query $Q$ on relational instance $I$

Complexity: NP-complete in combined, PTIME in data

- PTIME is not enough! We want $O(|Q| \times |I|)$
- More elaborate tasks? Counting, probabilistic evaluation, etc.
  → Efficient provenance computation
Restrict the query:

Current approaches

Restrict the instance:

• Bounded treewidth data: MSO has $O(|I|)$ time

• Problem: nonelementary in the query
Current approaches

Restrict the query:

- $\alpha$-acyclic CQs $\rightarrow O(|Q| \times |I|)$
Restrict the query:

- $\alpha$-acyclic CQs $\rightarrow O(|Q| \times |I|)$
- GF, RPQs, etc. $\rightarrow O(|Q| \times |I|)$
Restrict the query:

- $\alpha$-acyclic CQs $\rightarrow O(|Q| \times |I|)$
- GF, RPQs, etc. $\rightarrow O(|Q| \times |I|)$
- $\text{FO}^k$, bounded hypertreewidth, etc. $\rightarrow \text{PTIME but not } O(|Q| \times |I|)$
Current approaches

Restrict the query:

\[ \alpha \text{-acyclic CQs} \rightarrow O(|Q| \times |I|) \]
\[ \text{GF, RPQs, etc.} \rightarrow O(|Q| \times |I|) \]
\[ \text{FO}^k, \text{bounded hypertreewidth, etc.} \rightarrow \text{PTIME but not } O(|Q| \times |I|) \]

Restrict the instance:
Restrict the query:

- $\alpha$-acyclic CQs $\rightarrow \mathcal{O}(|Q| \times |I|)$
- GF, RPQs, etc. $\rightarrow \mathcal{O}(|Q| \times |I|)$
- $\text{FO}^k$, bounded hypertreewidth, etc. $\rightarrow \text{PTIME but not } \mathcal{O}(|Q| \times |I|)$

Restrict the instance:

- Bounded treewidth data: MSO has $\mathcal{O}(|I|)$ time data complexity
Current approaches

Restrict the query:

- $\alpha$-acyclic CQs $\rightarrow O(|Q| \times |I|)$
- GF, RPQs, etc. $\rightarrow O(|Q| \times |I|)$
- $\text{FO}^k$, bounded hypertreewidth, etc. $\rightarrow \text{PTIME but not } O(|Q| \times |I|)$

Restrict the instance:

- Bounded treewidth data: MSO has $O(|I|)$ time data complexity $|Q|$
- Problem: nonelementary in the query $2^{|Q|}$ (EXPTIME for CQs)
### Our Approach

<table>
<thead>
<tr>
<th>Approach</th>
<th>Restrict $Q$</th>
<th>Restrict $I$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>linear in combined</td>
<td>linear in data</td>
</tr>
<tr>
<td>Expressivity</td>
<td>😞</td>
<td>😊</td>
</tr>
</tbody>
</table>
## Our Approach

<table>
<thead>
<tr>
<th>Approach</th>
<th>Restrict $Q$</th>
<th>Restrict $I$</th>
<th>Restrict $Q$ and $I$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>linear in combined</td>
<td>linear in data</td>
<td>linear in combined</td>
</tr>
<tr>
<td>Expressivity</td>
<td>😞</td>
<td>😊</td>
<td>😊</td>
</tr>
</tbody>
</table>

Best of both worlds!
Idea: one parameter $k_I$ for the instance (treewidth) AND one parameter $k_Q$ for the query
Parameterized Complexity

Idea: one parameter $k_I$ for the instance (treewidth) AND one parameter $k_Q$ for the query

- **Instance** classes $\mathcal{I}_1, \mathcal{I}_2, \cdots$

Definition

The problem is fixed-parameter tractable (FPT) linear if there exists a computable function $f$ such that it can be solved in time $f(k_I, k_Q) \times |Q| \times |I|$.
Parameterized Complexity

Idea: one parameter $k_I$ for the instance (treewidth) AND one parameter $k_Q$ for the query

- **Instance** classes $\mathcal{I}_1, \mathcal{I}_2, \cdots$
- **Query** classes $\mathcal{Q}_1, \mathcal{Q}_2, \cdots$

Definition: The problem is fixed-parameter tractable (FPT) linear if there exists a computable function $f$ such that it can be solved in time $f(k_I, k_Q) \times |Q| \times |I|$. 
Idea: one parameter $k_I$ for the instance (treewidth) AND one parameter $k_Q$ for the query

- **Instance** classes $I_1, I_2, \ldots$
- **Query** classes $Q_1, Q_2, \ldots$

**Definition**

The problem is **fixed-parameter tractable (FPT) linear** if there exists a computable function $f$ such that it can be solved in time

$$f(k_I, k_Q) \times |Q| \times |I|$$
Main contributions

1) A new language...

- We introduce the language of \textit{intentional-clique-guarded Datalog} (ICG-Datalog), parameterized by body-size $k_P$
Main contributions

1) A new language...

   • We introduce the language of intentional-clique-guarded Datalog (ICG-Datalog), parameterized by body-size $k_P$

2) ... with FPT-linear (combined) evaluation...

   • Given an ICG-Datalog program $P$ with body-size $k_P$ and a relational instance $I$ of treewidth $k_I$, checking if $I \models P$ can be done in time $f(k_P, k_I) \times |P| \times |I|$
Main contributions

1) A new language...
   - We introduce the language of **intentional-clique-guarded Datalog** (ICG-Datalog), parameterized by **body-size** $k_P$

2) ... with **FPT-linear** (combined) evaluation...
   - Given an ICG-Datalog program $P$ with body-size $k_P$ and a relational instance $I$ of treewidth $k_I$, checking if $I \models P$ can be done in time $f(k_P, k_I) \times |P| \times |I|$

3) ... and also **FPT-linear** (combined) computation of provenance
   - We design a new concise provenance representation based on cyclic Boolean circuits: **cycluits**
• Fragment of Datalog with stratified negation
ICG-Datalog

- Fragment of Datalog with stratified negation
- \( \sigma = \sigma^{\text{ext}} \sqcup \sigma^{\text{int}} = \{R_1, R_2, \ldots\} \sqcup \{S_1, S_2, \ldots\} \)
ICG-Datalog

- Fragment of Datalog with stratified negation
- $\sigma = \sigma^{\text{ext}} \sqcup \sigma^{\text{int}} = \{R_1, R_2, \ldots\} \sqcup \{S_1, S_2, \ldots\}$
- Boolean programs: special 0-ary intensional predicate Goal()
ICG-Datalog

- Fragment of Datalog with stratified negation
- $\sigma = \sigma^{\text{ext}} \sqcup \sigma^{\text{int}} = \{R_1, R_2, \ldots\} \sqcup \{S_1, S_2, \ldots\}$
- Boolean programs: special 0-ary intensional predicate $\text{Goal}()$

\[
\begin{aligned}
S(x, t) &\leftarrow R_1(x, y) \land R_2(y, t, z) \land R_3(z, x) \land S'(x, y, z) \\
\vdots
\\text{Goal}() &\leftarrow \ldots
\end{aligned}
\]
ICG-Datalog

- Fragment of Datalog with stratified negation
- \( \sigma = \sigma^{\text{ext}} \sqcup \sigma^{\text{int}} = \{R_1, R_2, \ldots\} \sqcup \{S_1, S_2, \ldots\} \)
- Boolean programs: special 0-ary intensional predicate \textbf{Goal}()

\[
egin{align*}
\vdots \\
S(x, t) & \leftarrow R_1(x, y) \land R_2(y, t, z) \land R_3(z, x) \land S'(x, y, z) \\
\vdots \\
\textbf{Goal}() & \leftarrow \ldots
\end{align*}
\]

- Intensional clique-guarded
ICG-Datalog

- Fragment of Datalog with stratified negation
- $\sigma = \sigma^{\text{ext}} \sqcup \sigma^{\text{int}} = \{R_1, R_2, \ldots\} \sqcup \{S_1, S_2, \ldots\}$
- Boolean programs: special 0-ary intensional predicate Goal()

```
\begin{align*}
\vdots \\
S(x, t) & \leftarrow R_1(x, y) \land R_2(y, t, z) \land R_3(z, x) \land S'(x, y, z) \\
\vdots \\
\text{Goal()} & \leftarrow \cdots
\end{align*}
```

- Intensional clique-guarded
ICG-Datalog

- Fragment of Datalog with stratified negation
- \( \sigma = \sigma^{\text{ext}} \uplus \sigma^{\text{int}} = \{R_1, R_2, \ldots\} \uplus \{S_1, S_2, \ldots\} \)
- Boolean programs: special 0-ary intensional predicate \( \text{Goal}() \)

\[
\begin{align*}
S(x, t) & \leftarrow R_1(x, y) \land R_2(y, t, z) \land R_3(z, x) \land S'(x, y, z) \\
\vdots & \\
\text{Goal}() & \leftarrow \ldots
\end{align*}
\]

- Intensional clique-guarded
ICG-Datalog

- Fragment of Datalog with stratified negation
- \( \sigma = \sigma^{\text{ext}} \sqcup \sigma^{\text{int}} = \{R_1, R_2, \ldots\} \sqcup \{S_1, S_2, \ldots\} \)
- Boolean programs: special 0-ary intensional predicate Goal()

\[
\begin{align*}
S(x, t) &\leftarrow R_1(x, y) \land R_2(y, t, z) \land R_3(z, x) \land S'(x, y, z) \\
\vdots \\
\text{Goal()} &\leftarrow \ldots
\end{align*}
\]

- Intensional clique-guarded (\(\neq\) frontier-guarded Datalog!)
ICG-Datalog

- Fragment of Datalog with stratified negation
- $\sigma = \sigma^\text{ext} \sqcup \sigma^\text{int} = \{R_1, R_2, \ldots\} \sqcup \{S_1, S_2, \ldots\}$
- Boolean programs: special 0-ary intensional predicate Goal()

\[
\begin{align*}
\vdots \\
S(x, t) & \leftarrow R_1(x, y) \land R_2(y, t, z) \land R_3(z, x) \land S'(x, y, z) \\
\vdots \\
\text{Goal()} & \leftarrow \ldots
\end{align*}
\]

- Intensional clique-guarded (≠ frontier-guarded Datalog!)
- body-size = $\text{MaxArity}(\sigma) \times \max_{\text{rule } r} \text{NbAtoms}(r)$
  "size to write a rule"
ICG-Datalog

- Fragment of Datalog with stratified negation
- \( \sigma = \sigma^{\text{ext}} \sqcup \sigma^{\text{int}} = \{ R_1, R_2, \ldots \} \sqcup \{ S_1, S_2, \ldots \} \)
- Boolean programs: special o-ary intensional predicate Goal()

\[
\begin{align*}
\vdots \\
S(x, t) & \leftarrow R_1(x, y) \land R_2(y, t, z) \land R_3(z, x) \land S'(x, y, z) \\
\vdots \\
\text{Goal()} & \leftarrow \ldots
\end{align*}
\]

- Intensional clique-guarded (\( \neq \) frontier-guarded Datalog!)
- body-size = MaxArity(\( \sigma \)) \( \times \) max\( _\text{rule} \) \( _r \) NbAtoms(\( r \))
- "size to write a rule"
- We also allow stratified negation
Database I of treewidth ≤ $k_1$

ICG-Datalog program $P$

- $C(x) \leftarrow \text{Subway("Corvisart",x)}$
- $C(x) \leftarrow C(y) \land \text{Subway}(y,x)$

(Paris Metro map)
Database $I$ of treewidth $\leq k_1$

ICG-Datalog program $P$

1. $C(x) \leftarrow \text{Subway("Corvisart",x)}$
2. $C(x) \leftarrow C(y) \land \text{Subway}(y,x)$
3. $\text{Goal()} \leftarrow \neg C("\text{Châtelet"})$

(Paris Metro map)
Database I of treewidth $\leq k_1$

ICG-Datalog program $P$

1. $C(x) \leftarrow \text{Subway("Corvisart",x)}$
   $C(x) \leftarrow C(y) \land \text{Subway}(y,x)$

2. $\text{Goal()} \leftarrow \neg C("\text{Châtelet}")$

"Is it impossible to go from station Corvisart to station Châtelet with the subway?"
Example

**Database I**
of treewidth $\leq k_i$

(Paris Metro map)

**ICG-Datalog program P**
of body-size 4

1. $C(x) \leftarrow \text{Subway("Corvisart",} x\text{)}$
2. $C(x) \leftarrow C(y) \land \text{Subway}(y, x)$
3. $\text{Goal()} \leftarrow \neg C("\text{Châtelet"})$

"Is it impossible to go from station Corvisart to station Châtelet with the subway?"
CQs captured

- ICG-Datalog can express any Boolean CQ (unlike, e.g, CGF)
CQs captured

- ICG-Datalog can express any Boolean CQ (unlike, e.g., CGF)
- *Simplicial width* of a CQ: interface between bags are cliques
CQs captured

- ICG-Datalog can express any Boolean CQ (unlike, e.g, CGF)
- Simplicial width of a CQ: interface between bags are cliques
  → upper bound of treewidth
ICG-Datalog can express any Boolean CQ (unlike, e.g., CGF)

*Simplicial width* of a CQ: interface between bags are cliques → upper bound of treewidth

**Theorem**

*Bounded simplicial width* conjunctive queries can be captured by *bounded body-size* ICG-Datalog programs
CQs captured

- ICG-Datalog can express any Boolean CQ (unlike, e.g., CGF)
- Simplicial width of a CQ: interface between bags are cliques → upper bound of treewidth

**Theorem**

Bounded simplicial width conjunctive queries can be captured by bounded body-size ICG-Datalog programs

- Cannot capture bounded treewidth CQs with the same tools
Other languages captured

- $\alpha$-acyclic CQs for $k_P \leq \text{MaxArity}(\sigma^\text{ext})$
Other languages captured

- $\alpha$-acyclic CQs for $k_P \leq \text{MaxArity}(\sigma^\text{ext})$
- Boolean 2RPQs, SAC2RPQs for $k_P \leq 4$
Other languages captured

- $\alpha$-acyclic CQs for $k_p \leq \text{MaxArity}(\sigma^{\text{ext}})$
- Boolean 2RPQs, SAC2RPQs for $k_p \leq 4$
- Monadic Datalog of bounded body-size
Other languages captured

- $\alpha$-acyclic CQs for $k_p \leq \text{MaxArity}(\sigma^\text{ext})$
- Boolean 2RPQs, SAC2RPQs for $k_p \leq 4$
- Monadic Datalog of bounded body-size
- Some Guarded Negation fragments (e.g. GNF with $CQ$-rank)
ICG-Datalog program $P$
of body-size $\leq k_p$

1

$C(x) \leftarrow \text{Subway}("Corvisart", x)$
$C(x) \leftarrow C(y) \land \text{Subway}(y, x)$

2

$\text{Goal}() \leftarrow \neg C("\text{Châtelet"})$

Database $I$
of treewidth $\leq k_i$

(Paris Metro map)
Proof Structure

ICG-Datalog program $P$ of body-size $\leq k_p$

1. $C(x) ← \text{Subway("Corvisart",x)}$
   $C(x) ← C(y) \land \text{Subway}(y,x)$

2. $\text{Goal()} ← \neg C("\text{Châtelet"})$

Database $I$ of treewidth $\leq k_i$

Tree encoding $E$

$O(g'(k_i) |I|)$

(Paris Metro map)
ICG-Datalog program $P$ of body-size $\leq k_p$

$C(x) \leftarrow \text{Subway}("\text{Corvisart}";x)$
$C(x) \leftarrow C(y) \land \text{Subway}(y,x)$

Goal() $\leftarrow \neg C("\text{Châtelet"})$

Database $I$ of treewidth $\leq k_i$

Tree Automaton $A$

Tree encoding $E$

$O(\ g'(k_i) \ |I| \ )$

(Paris Metro map)
**ICG-Datalog program P of body-size $\leq k_p$**

1. $C(x) \leftarrow \text{Subway}("Corvisart", x)$
2. $C(x) \leftarrow C(y) \land \text{Subway}(y, x)$

**Goal**

$\neg C("Châtelet")$

**Database I of treewidth $\leq k_i$**

**(Paris Metro map)**

**Tree encoding E**

$O(\ g'(k_i) \ |I| \ )$

**Tree Automaton A**

$O( \ |A| \cdot |E| )$

**Answer**

“Is it impossible to go from station Corvisart to station Châtelet with the subway?”

YES/NO
ICG-Datalog program $P$ of body-size $\leq k_p$

1. $C(x) \leftarrow \text{Subway("Corvisart",x)}$
2. $C(x) \leftarrow C(y) \land \text{Subway}(y,x)$

Goal() $\leftarrow \neg C("\text{Châtelet}"")$

Database $I$ of treewidth $\leq k_i$

Two-way Alternating Tree Automaton $A$

Tree encoding $E$

$O(\min(1, |A| \cdot |E|))$

"Is it impossible to go from station Corvisart to station Châtelet with the subway?"

Answer

YES/NO
Proof Structure

ICG-Datalog program $P$ of body-size $\leq k_p$

1. $C(x) \leftarrow \text{Subway("Corvisart",x)}$
2. $C(x) \leftarrow C(y) \land \text{Subway(y,x)}$
3. $\text{Goal()} \leftarrow \neg C("\text{Châtelet"})$

Database $I$ of treewidth $\leq k_i$

Two-way Alternating Tree Automaton $A$

Tree encoding $E$

$O(g(k_p, k_i) |P|)$

$O(|A| \cdot |E|)$

Answer

"Is it impossible to go from station Corvisart to station Châtelet with the subway?"

Yes/No

$O(g'(k_i) |I|)$

(Paris Metro map)
Provenance

Definition
The provenance \( \text{Prov}(P, I) \) of program \( P \) on instance \( I \) is the function that takes as input a subinstance \( I' \subseteq I \) and outputs \( \text{TRUE} \) iff \( I' \models P \).
Definition
The provenance $\text{Prov}(P, I)$ of program $P$ on instance $I$ is the function that takes as input a subinstance $I' \subseteq I$ and outputs TRUE iff $I' \models P$

Possible representations:

- Boolean formulas (with the facts as variables)
**Definition**

The *provenance* $\text{Prov}(P, I)$ of program $P$ on instance $I$ is the function that takes as input a subinstance $I' \subseteq I$ and outputs TRUE iff $I' \models P$.

Possible representations:

- Boolean formulas (with the facts as variables)
- Boolean circuits (with the facts as inputs)
Definition
The provenance $\text{Prov}(P, I)$ of program $P$ on instance $I$ is the function that takes as input a subinstance $I' \subseteq I$ and outputs TRUE iff $I' \models P$

Possible representations:

- Boolean formulas (with the facts as variables)
- Boolean circuits (with the facts as inputs)

New Boolean circuits... with cycles! (cycluits)
Definition

The provenance $\text{Prov}(P, I)$ of program $P$ on instance $I$ is the function that takes as input a subinstance $I' \subseteq I$ and outputs TRUE iff $I' \models P$

Possible representations:

- Boolean formulas (with the facts as variables)
- Boolean circuits (with the facts as inputs)

New Boolean circuits... with cycles! (cycluits)

Theorem

Given an ICG-Datalog program $P$ with body-size $k_P$ and a relational instance $I$ of treewidth $k_I$, we can compute in time $f(k_P, k_I) \times |P| \times |I|$ a Boolean cycluit capturing $\text{Prov}(Q, I)$
ICG-Datalog program $P$ of body-size $\leq k_P$

1. $C(x) \leftarrow \text{Subway("Corvisart",}x\text{)}$
2. $C(x) \leftarrow C(y) \land \text{Subway}(y,x)$

Goal() $\leftarrow \neg C("\text{Châtelet"})$

Database $I$ of treewidth $\leq k_I$

Tree encoding $E$

Two-way Alternating Tree Automaton $A$

$O(g(k_P, k_I) |P|)$

$O(|A| \cdot |E|)$

"Is it impossible to go from station Corvisart to station Châtelet with the subway?"

YES/NO

$O(g(k, |I|) |P|)$

(Paris Metro map)
ICG-Datalog program $P$ of body-size $\leq k_P$

1. $C(x) \leftarrow \text{Subway}("Corvisart", x)$
   $C(x) \leftarrow C(y) \land \text{Subway}(y, x)$

2. $\text{Goal()} \leftarrow \neg C("Châtelet")$

Database $I$ of treewidth $\leq k_I$

$C(x) \leftarrow \text{Subway}("Corvisart", x)$
$C(x) \leftarrow C(y) \land \text{Subway}(y, x)$
$\text{Goal()} \leftarrow \neg C("Châtelet")$

Tree encoding $E$

Two-way Alternating Tree Automaton $A$

Provenance Cycluit

$O(g(k_P, k_I) |P|)$
$O(|A| \cdot |E|)$
$O(g'(k_I) |I|)$
$O(|A| \cdot |E|)$

(Paris Metro map)
ICG-Datalog program $P$ of body-size $\leq k_p$

1. $C(x) \leftarrow \text{Subway}("Corvisart", x)$
2. $C(x) \leftarrow C(y) \land \text{Subway}(y, x)$

Goal() $\leftarrow \neg C("Châtelet")$

Database $I$ of treewidth $\leq k_i$

"Under which conditions is it impossible to go from station Corvisart to station Châtelet with the subway?"
- Circuit with cycles
Cycluits

- Circuit with cycles
- Forbid cycles of negation
  $\Rightarrow$ the cycluit is *stratified*
• Circuit with cycles
• Forbid cycles of negation
  \[ \implies \text{the cycluit is stratified} \]
• **Semantics**: least fixed-point
Cycluits

- Circuit with cycles
- Forbid cycles of negation \(\implies\) the cycluit is *stratified*
- **Semantics**: least fixed-point
- **Evaluation**: linear time
Cycluits

- Circuit with cycles
- Forbid cycles of negation \( \Rightarrow \) the cycluit is \textit{stratified}
- \textbf{Semantics}: least fixed-point
- \textbf{Evaluation}: linear time
• Introduced ICG-Datalog, FPT-linear parameterized by body-size of program $P$ and instance treewidth: $f(k_P, k_I) \times |P| \times |I|$
Conclusion

- Introduced ICG-Datalog, FPT-linear parameterized by body-size of program $P$ and instance treewidth: $f(k_P, k_I) \times |P| \times |I|$
- We propose a new concise provenance representation: cycluits
Conclusion

• Introduced ICG-Datalog, FPT-linear parameterized by body-size of program $P$ and instance treewidth: $f(k_P, k_I) \times |P| \times |I|$

• We propose a new concise provenance representation: cycluits

Other results:
Conclusion

- Introduced ICG-Datalog, FPT-linear parameterized by body-size of program $P$ and instance treewidth: $f(k_P, k_I) \times |P| \times |I|$
- We propose a new concise provenance representation: cycluits

Other results:

- Application to probabilistic evaluation, model counting
Conclusion

- Introduced ICG-Datalog, FPT-linear parameterized by body-size of program $P$ and instance treewidth: $f(k_P, k_I) \times |P| \times |I|$
- We propose a new concise provenance representation: cycluits

Other results:

- Application to probabilistic evaluation, model counting
- Lower bounds (bounded treewidth CQs, type of automata)
Conclusion

• Introduced ICG-Datalog, FPT-linear parameterized by body-size of program $P$ and instance treewidth: $f(k_P, k_I) \times |P| \times |I|$

• We propose a new concise provenance representation: cycluits

Other results:

• Application to probabilistic evaluation, model counting
• Lower bounds (bounded treewidth CQs, type of automata)

Future work:

• Improve ICG-Datalog $\rightarrow$ Clique-Frontier-Guarded Datalog
Conclusion

• Introduced **ICG-Datalog**, FPT-linear parameterized by body-size of program $P$ and instance treewidth: $f(k_P, k_I) \times |P| \times |I|$
• We propose a new concise provenance representation: cycluits

Other results:

• Application to probabilistic evaluation, model counting
• Lower bounds (bounded treewidth CQs, type of automata)

Future work:

• Improve ICG-Datalog $\rightarrow$ Clique-Frontier-Guarded Datalog
• Show PTIME combined complexity when body-size only is bounded (on arbitrary instances) $O(|P| \times |I|^{k_P})$
Conclusion

• Introduced ICG-Datalog, FPT-linear parameterized by body-size of program $P$ and instance treewidth: $f(k_P, k_I) \times |P| \times |I|$

• We propose a new concise provenance representation: cycluits

Other results:

• Application to probabilistic evaluation, model counting
• Lower bounds (bounded treewidth CQs, type of automata)

Future work:

• Improve ICG-Datalog $\rightarrow$ Clique-Frontier-Guarded Datalog
• Show PTIME combined complexity when body-size only is bounded (on arbitrary instances) $O(|P| \times |I|^{k_P})$
• Extend cycluit framework to more expressive provenance semirings
ICG-Datalog program $P$ of body-size $\leq k_P$

1. \[ C(x) \leftarrow \text{Subway("Corvisart",x)} \]
2. \[ C(x) \leftarrow C(y) \land \text{Subway(y,x)} \]

Goal() $\leftarrow \neg C(\text{"Châtelet"})$

Database $I$ of treewidth $\leq k_I$

Tree encoding $E$

Two-way Alternating Tree Automaton $A$

Provenance Cycluit

$O( g(k_P, k_I) |P| )$

$O( |A| \cdot |E| )$

$O( g'(k_I) |I| )$

"Under which conditions is it impossible to go from station Corvisart to station Châtelet with the subway?"