Combined Tractability of Query Evaluation via Tree Automata and Cycluits

Antoine Amarilli[1], Pierre Bourhis[2], **Mikaël Monet**[1,4], Pierre Senellart[3,4]

March 23th, 2017

[1]LTCI, Télécom ParisTech, Université Paris-Saclay; Paris, France

[2]CNRS, CRIStAL, Inria Lille; Lille, France

[3]École normale supérieure, PSL Reasearch University; Paris, France

[4]Inria Paris; Paris, France

Big data, big queries

Conjunctive query Q on relational instance I

Big data, big queries

Conjunctive query Q on relational instance I

Complexity: NP-complete in combined, PTIME in data

Big data, big queries

Conjunctive query Q on relational instance I

Complexity: NP-complete in combined, PTIME in data

- PTIME is not enough! We want $O(|Q| \times |I|)$

Big data, big queries

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.

Big data, big queries

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.

$\rightarrow$ Efficient provenance computation

Current approaches

Restrict the query:

Current approaches

Restrict the query:

- α-acyclic CQs $\rightarrow O(|Q| \times |I|)$

Current approaches

Restrict the query:

- α-acyclic CQs $\rightarrow O(|Q| \times |I|)$
- GF, RPQs, etc. $\rightarrow O(|Q| \times |I|)$

Current approaches

Restrict the query:

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

Current approaches

Restrict the query:

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

Restrict the instance:

Current approaches

Restrict the query:

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

Restrict the instance:

- Bounded treewidth data: MSO has $O(|I|)$ time *data* complexity

Current approaches

Restrict the query:

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

Restrict the instance:

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

Our Approach

Approach	Restrict $\mathcal{Q}$	Restrict $\mathcal{I}$	
Complexity	linear in combined	linear in data	
Expressivity			

Our Approach

Approach	Restrict $\mathcal{Q}$	Restrict $\mathcal{I}$	**Restrict $\mathcal{Q}$ *and* $\mathcal{I}$**
Complexity	linear in combined	linear in data	**linear in combined**
Expressivity			

Best of both worlds!

Parameterized Complexity

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 $\mathcal{I}_1, \mathcal{I}_2, \cdots$

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 $\mathcal{Q}_1, \mathcal{Q}_2, \cdots$

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

Main contributions

1) A new language...

- We introduce the language of ***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***

ICG-Datalog

- 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
- $\sigma = \sigma^{\mathrm{ext}} \sqcup \sigma^{\mathrm{int}} = \{R_1, R_2, \ldots\} \sqcup \{S_1, S_2, \ldots\}$
- Boolean programs: special 0-ary intensional predicate Goal()

ICG-Datalog

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

$$\begin{cases} \vdots \\ S(x,t) \leftarrow R_1(x,y) \wedge R_2(y,t,z) \wedge R_3(z,x) \wedge S'(x,y,z) \\ \vdots \\ \mathrm{Goal}() \leftarrow \cdots \end{cases}$$

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()

$$\begin{cases} \vdots \\ S(x,t) \leftarrow R_1(x,y) \land R_2(y,t,z) \land R_3(z,x) \land S'(x,y,z) \\ \vdots \\ \text{Goal}() \leftarrow \cdots \end{cases}$$

- 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()

$$\begin{cases} \vdots \\ S(x, t) \leftarrow R_1(x, y) \land R_2(y, t, z) \land R_3(z, x) \land S'(x, y, z) \\ \vdots \\ \text{Goal}() \leftarrow \cdots \end{cases}$$

- 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()

$$\begin{cases} \vdots \\ S(x,t) \leftarrow R_1(x,y) \wedge R_2(y,t,z) \wedge R_3(z,x) \wedge S'(x,y,z) \\ \vdots \\ \text{Goal}() \leftarrow \cdots \end{cases}$$

- Intensional clique-guarded

ICG-Datalog

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

$$\begin{cases} \vdots \\ S(x,t) \leftarrow R_1(x,y) \wedge R_2(y,t,z) \wedge R_3(z,x) \wedge S'(x,y,z) \\ \vdots \\ \mathrm{Goal}() \leftarrow \cdots \end{cases}$$

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

ICG-Datalog

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

$$\begin{cases} \vdots \\ S(x,t) \leftarrow R_1(x,y) \land R_2(y,t,z) \land R_3(z,x) \land S'(x,y,z) \\ \vdots \\ \mathrm{Goal}() \leftarrow \cdots \end{cases}$$

- Intensional clique-guarded ($\neq$ frontier-guarded Datalog!)
- body-size $= \mathrm{MaxArity}(\sigma) \times \max_{\text{rule } r} \mathrm{NbAtoms}(r)$
  "*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 0-ary intensional predicate Goal()

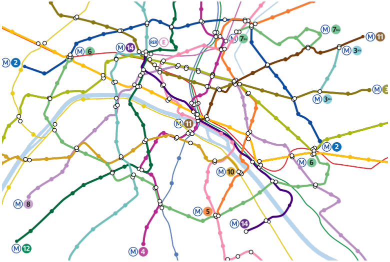
$$\begin{cases} \vdots \\ S(x,t) \leftarrow R_1(x,y) \wedge R_2(y,t,z) \wedge R_3(z,x) \wedge S'(x,y,z) \\ \vdots \\ \text{Goal}() \leftarrow \cdots \end{cases}$$

- Intensional clique-guarded ($\neq$ frontier-guarded Datalog!)
- body-size $= \text{MaxArity}(\sigma) \times \max_{\text{rule } r} \text{NbAtoms}(r)$
  "*size to write a rule*"
- We also allow stratified negation

Example

Database I

of treewidth ≤ k_I



(Paris Metro map)

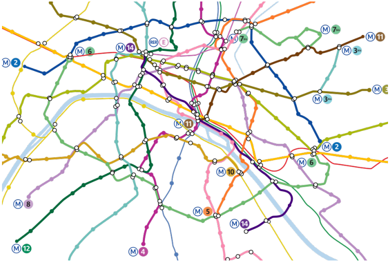
ICG-Datalog program P

C(x) ← Subway("Corvisart",x)

C(x) ← C(y) ∧ Subway(y,x)

Example

Database I
of treewidth ≤ k_I



(Paris Metro map)

ICG-Datalog program P

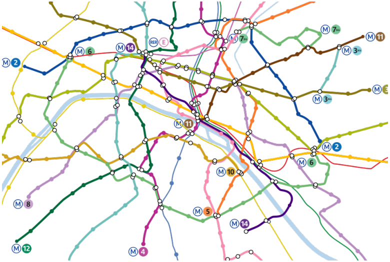
1
$$C(x) \leftarrow \text{Subway}(\text{"Corvisart"}, x)$$
$$C(x) \leftarrow C(y) \wedge \text{Subway}(y, x)$$

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

Example

Database I

of treewidth $\leq k_I$



(Paris Metro map)

ICG-Datalog program P

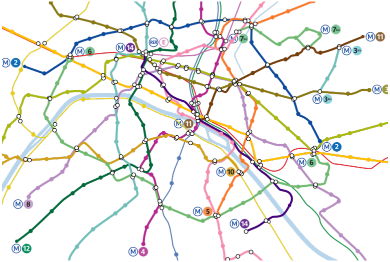
1
C(x) ← Subway("Corvisart",x)
C(x) ← C(y) ∧ Subway(y,x)

2
Goal() ← ¬ C("Châtelet")

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

Example

Database I
of treewidth ≤ k_I



(Paris Metro map)

ICG-Datalog program P
of body-size 4

1
C(x) ← Subway("Corvisart",x)
C(x) ← C(y) ∧ Subway(y,x)

2
Goal() ← ¬ C("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

$\rightarrow$ upper bound of treewidth

CQs captured

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

$\rightarrow$ 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

$\rightarrow$ 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

- α-acyclic CQs for $k_P \leqslant \mathrm{MaxArity}(\sigma^{\mathrm{ext}})$

Other languages captured

- α-acyclic CQs for $k_P \leqslant \mathrm{MaxArity}(\sigma^{\mathrm{ext}})$
- Boolean 2RPQs, SAC2RPQs for $k_P \leqslant 4$

Other languages captured

- α-acyclic CQs for $k_P \leqslant \mathrm{MaxArity}(\sigma^{\mathrm{ext}})$
- Boolean 2RPQs, SAC2RPQs for $k_P \leqslant 4$
- Monadic Datalog of bounded body-size

Other languages captured

- α-acyclic CQs for $k_P \leqslant \mathrm{MaxArity}(\sigma^{\mathrm{ext}})$
- Boolean 2RPQs, SAC2RPQs for $k_P \leqslant 4$
- Monadic Datalog of bounded body-size
- Some Guarded Negation fragments (e.g GNF with *CQ-rank*)

Proof Structure

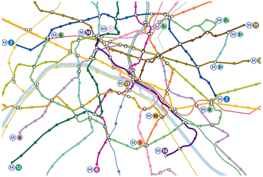
Proof Structure

ICG-Datalog program P
of body-size ≤ k_P

1 C(x) ← Subway("Corvisart",x)
C(x) ← C(y) ∧ Subway(y,x)

2 Goal() ← ¬ C("Châtelet")

Database I
of treewidth ≤ k_I



(Paris Metro map)

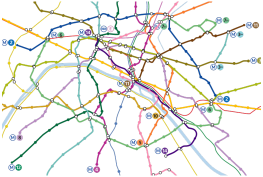
Proof Structure

ICG-Datalog program P
of body-size ≤ k_P

1
C(x) ← Subway("Corvisart",x)
C(x) ← C(y) ∧ Subway(y,x)

2
Goal() ← ¬ C("Châtelet")

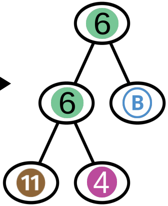
Database I
of treewidth ≤ k_I



(Paris Metro map)

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

Tree encoding E



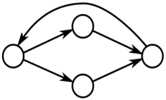
Proof Structure

ICG-Datalog program P
of body-size ≤ k_P

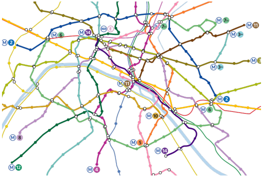
1 | C(x) ← Subway("Corvisart",x)
C(x) ← C(y) ∧ Subway(y,x)

2 | Goal() ← ¬ C("Châtelet")

Tree Automaton A



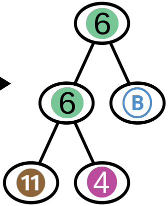
Database I
of treewidth ≤ k_I



(Paris Metro map)

O(g'(k_I) |I|)

Tree encoding E



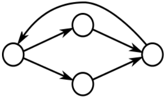
Proof Structure

ICG-Datalog program P
of body-size ≤ k_P

1 C(x) ← Subway("Corvisart",x)
C(x) ← C(y) ∧ Subway(y,x)

2 Goal() ← ¬ C("Châtelet")

Tree Automaton A



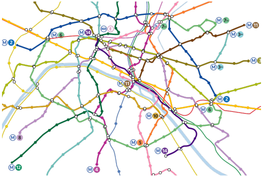
$O(|A| \cdot |E|)$

Answer

YES/NO

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

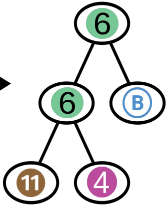
Database I
of treewidth ≤ k_I



(Paris Metro map)

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

Tree encoding E



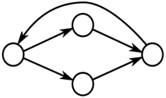
Proof Structure

ICG-Datalog program P
of body-size ≤ k_P

1
C(x) ← Subway("Corvisart",x)
C(x) ← C(y) ∧ Subway(y,x)

2
Goal() ← ¬ C("Châtelet")

Two-way Alternating Tree Automaton A



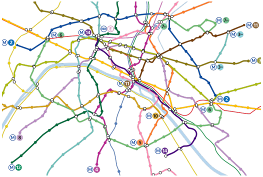
O(|**A**| • |**E**|)

Answer

YES/NO

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

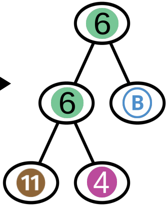
Database I
of treewidth ≤ k_I



(Paris Metro map)

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

Tree encoding E



Proof Structure

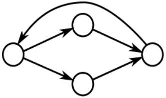
ICG-Datalog program P
of body-size ≤ k_P

1 C(x) ← Subway("Corvisart",x)
C(x) ← C(y) ∧ Subway(y,x)

2 Goal() ← ¬ C("Châtelet")

O(g(k_P, k_I) |**P**|) →

Two-way Alternating Tree Automaton A



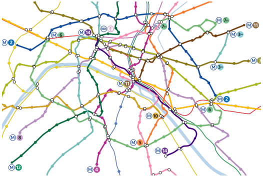
O(|**A**| • |**E**|) →

Answer

YES/NO

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

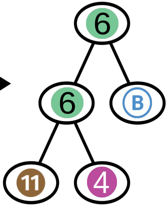
Database I
of treewidth ≤ k_I



(Paris Metro map)

O(g'(k_I) |**I**|) →

Tree encoding E



Provenance

Definition

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

Provenance

Definition

The *provenance* $\mathrm{Prov}(P, I)$ 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* $\mathrm{Prov}(P, I)$ 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)

Provenance

Definition

The *provenance* $\mathrm{Prov}(P, I)$ 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* $\mathrm{Prov}(P, I)$ 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 $\mathrm{Prov}(Q, I)$*

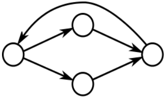
ICG-Datalog program P
of body-size ≤ k_P

1
C(x) ← Subway("Corvisart",x)
C(x) ← C(y) ∧ Subway(y,x)

2
Goal() ← ¬ C("Châtelet")

$O(\, g(k_P, k_I)\, |P|\,)$ →

Two-way Alternating Tree Automaton A



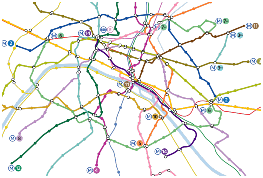
$O(\, |A| \cdot |E|\,)$ →

Answer

YES/NO

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

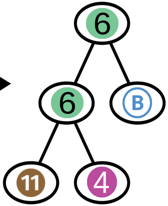
Database I
of treewidth ≤ k_I

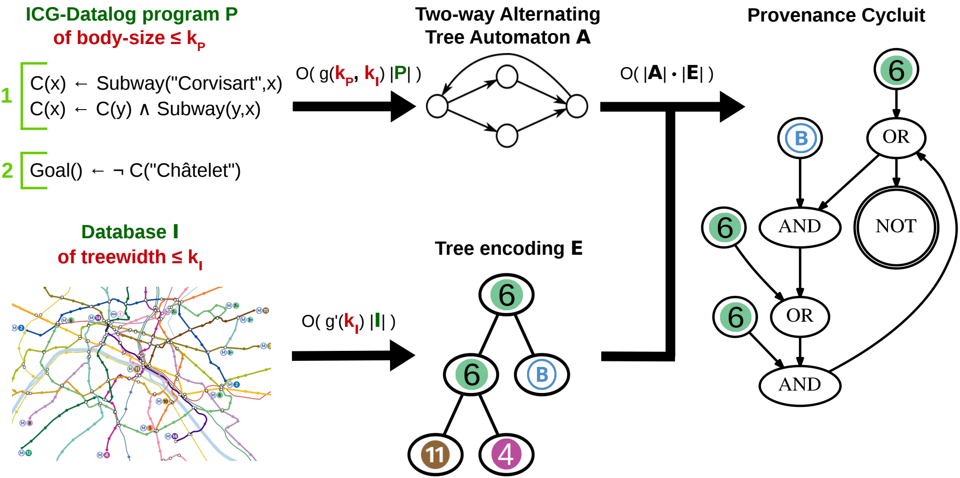


(Paris Metro map)

$O(\, g'(k_I)\, |I|\,)$ →

Tree encoding E





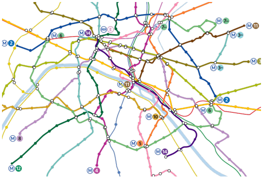
(Paris Metro map)

ICG-Datalog program P
of body-size ≤ k_P

1 C(x) ← Subway("Corvisart",x)
C(x) ← C(y) ∧ Subway(y,x)

2 Goal() ← ¬ C("Châtelet")

Database I
of treewidth ≤ k_I

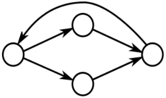


(Paris Metro map)

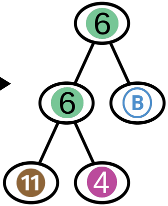
O(g(k_P, k_I) |P|)

O(g'(k_I) |I|)

Two-way Alternating Tree Automaton A

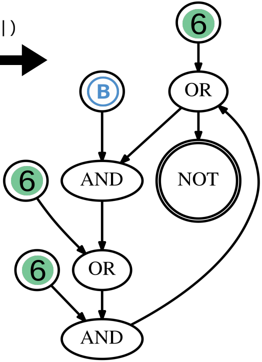


Tree encoding E



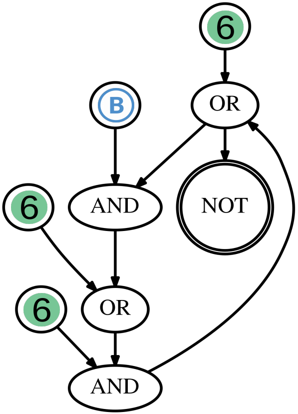
O(|A| · |E|)

Provenance Cycluit



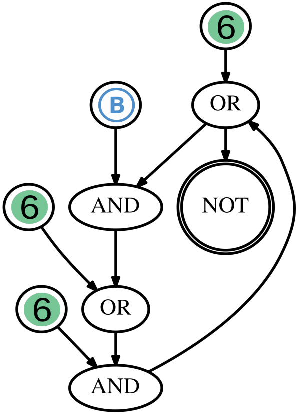
“***Under which conditions*** *is it impossible to go from station Corvisart to station Châtelet with the subway?*"

Cycluits



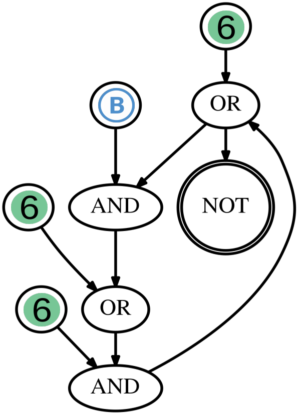
- Circuit with cycles

Cycluits



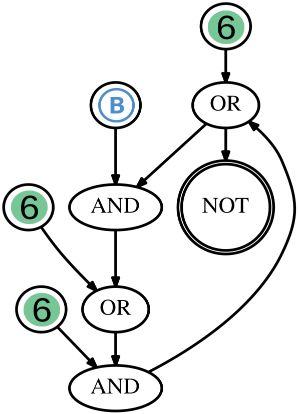
- Circuit with cycles
- Forbid cycles of negation
  $\implies$ the cycluit is *stratified*

Cycluits



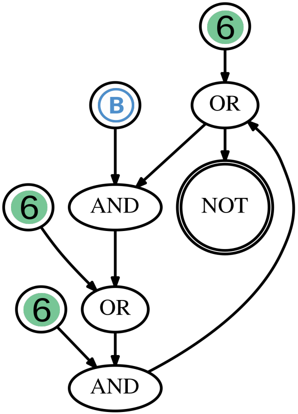
- Circuit with cycles
- Forbid cycles of negation
  $\implies$ the cycluit is *stratified*
- **Semantics**: least fixed-point

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
  $\implies$ the cycluit is *stratified*
- **Semantics**: least fixed-point
- **Evaluation**: linear time

Conclusion

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

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

Thank you!

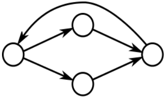
ICG-Datalog program P
of body-size ≤ k_P

1 C(x) ← Subway("Corvisart",x)
C(x) ← C(y) ∧ Subway(y,x)

2 Goal() ← ¬ C("Châtelet")

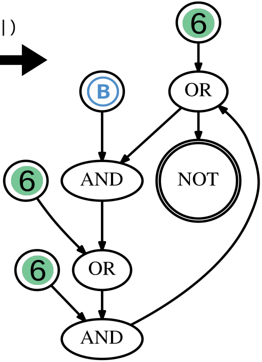
O(g(k_P, k_I) |**P**|) →

Two-way Alternating Tree Automaton A

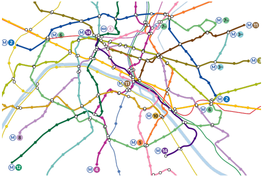


O(|**A**| · |**E**|) →

Provenance Cycluit



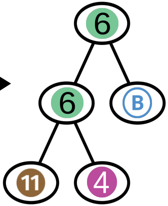
Database I
of treewidth ≤ k_I



(Paris Metro map)

O(g'(k_I) |**I**|) →

Tree encoding E



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