Query Evaluation:
Enumeration, Maintenance, Reliability

Soutenance d'habilitation à diriger des recherches

Antoine Amarilli

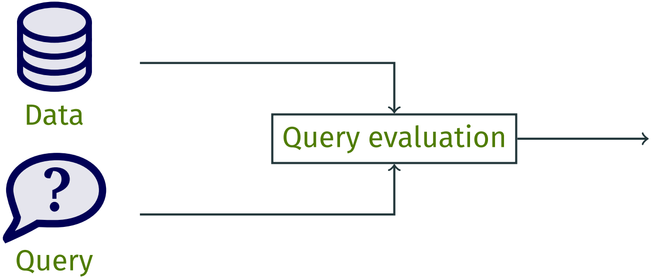
April 4, 2023

Télécom Paris

Introduction

Query evaluation

Central question studied in my research: how to efficiently evaluate queries on data?

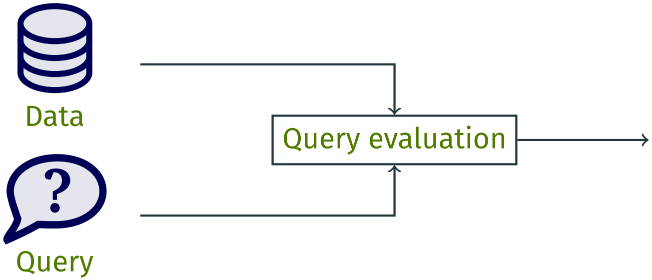


A	B
a_1	b_1
a_2	b_2
a_3	b_3

Results

Query evaluation

Central question studied in my research: how to efficiently evaluate **queries** on **data**?



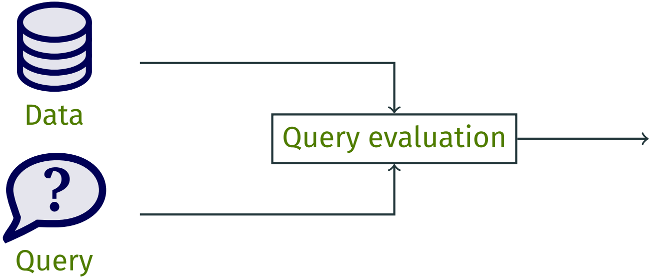
A	B
a_1	b_1
a_2	b_2
a_3	b_3

Results

- Measure the **efficiency** of this task

Query evaluation

Central question studied in my research: how to efficiently evaluate **queries** on **data**?

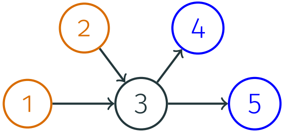


A	B
a_1	b_1
a_2	b_2
a_3	b_3

Results

- Measure the **efficiency** of this task
- **Theoretical study** (asymptotic complexity, lower bounds) rather than **practical**

Example: Reachability query



Data: Graph G

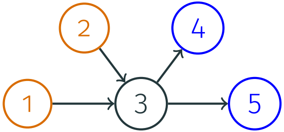
Query $Q(x, y)$: *"Which **orange** nodes x have a directed path to which **blue** nodes y?"*

Query evaluation

x	y
1	4
1	5
2	4
2	5

Results

Example: Reachability query



Data: Graph G

Query evaluation

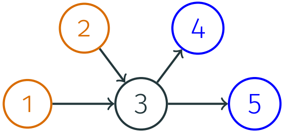
x	y
1	4
1	5
2	4
2	5

Results

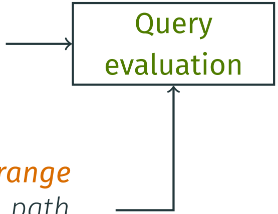
Query $Q(x, y)$: *“Which **orange** nodes **x** have a directed path to which **blue** nodes **y**?”*

Extend to **three tasks**: enumeration, maintenance, and reliability

Enumeration: Producing results in streaming



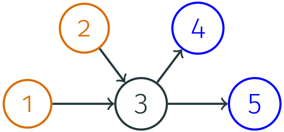
Data: Graph G



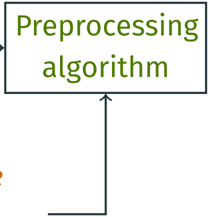
Query $Q(x, y)$: *“Which **orange** nodes **x** have a directed path to which **blue** nodes **y**?”*

- **Usual complexity measure:** time to produce the **entire output**

Enumeration: Producing results in streaming



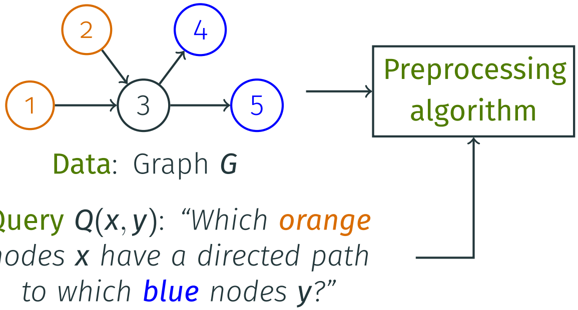
Data: Graph G



Query $Q(x, y)$: *"Which* ***orange*** *nodes* $\boldsymbol{x}$ *have a directed path to which* ***blue*** *nodes* $\boldsymbol{y}$*?"*

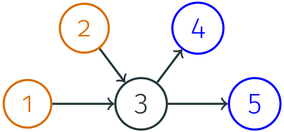
- **Usual complexity measure:** time to produce the **entire output**
- More precise measure: **enumeration algorithms**:

Enumeration: Producing results in streaming



- **Usual complexity measure:** time to produce the **entire output**
- More precise measure: **enumeration algorithms**:
    - **Preprocessing time**: time to produce **compressed representation**

Enumeration: Producing results in streaming



Data: Graph G

Query $Q(x, y)$: *"Which **orange** nodes **x** have a directed path to which **blue** nodes **y**?"*

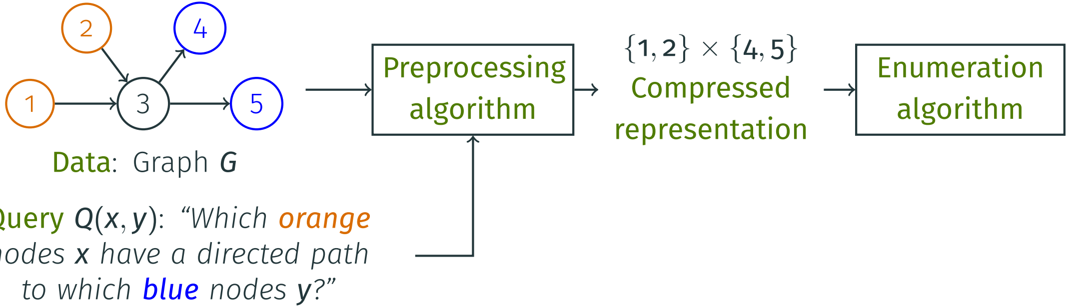
Preprocessing algorithm

$\{1, 2\} \times \{4, 5\}$

Compressed representation

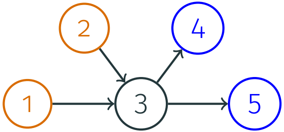
- **Usual complexity measure:** time to produce the **entire output**
- More precise measure: **enumeration algorithms**:
  - **Preprocessing time**: time to produce **compressed representation**

Enumeration: Producing results in streaming



- **Usual complexity measure:** time to produce the **entire output**
- More precise measure: **enumeration algorithms**:
  - **Preprocessing time**: time to produce **compressed representation**

Enumeration: Producing results in streaming



Data: Graph G

Query $Q(x, y)$: *"Which* ***orange*** *nodes* x *have a directed path to which* ***blue*** *nodes* y*?"*

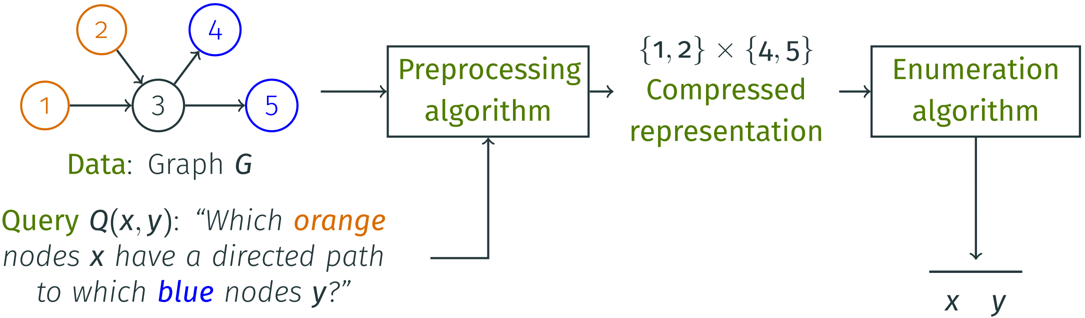
Preprocessing algorithm → $\{1, 2\} \times \{4, 5\}$ Compressed representation → Enumeration algorithm

x	y

Results

- **Usual complexity measure:** time to produce the **entire output**
- More precise measure: **enumeration algorithms**:
  - **Preprocessing time**: time to produce **compressed representation**

Enumeration: Producing results in streaming

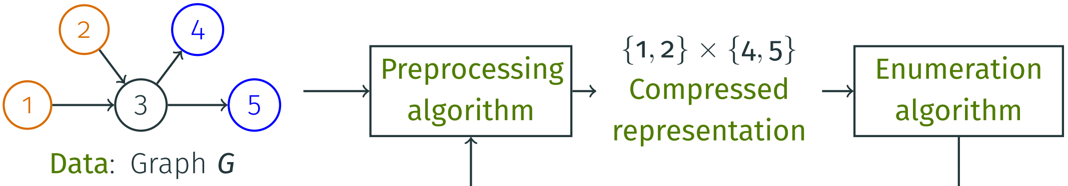


x	y
1	4

Results

- **Usual complexity measure:** time to produce the **entire output**
- More precise measure: **enumeration algorithms**:
  - **Preprocessing time**: time to produce **compressed representation**
  - **Delay** between each consecutive output

Enumeration: Producing results in streaming



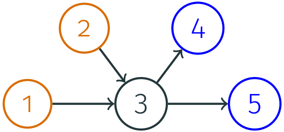
Query $Q(x, y)$: *"Which **orange** nodes **x** have a directed path to which **blue** nodes **y**?"*

x	y
1	4
1	5

Results

- **Usual complexity measure:** time to produce the **entire output**
- More precise measure: **enumeration algorithms**:
  - **Preprocessing time**: time to produce **compressed representation**
  - **Delay** between each consecutive output

Enumeration: Producing results in streaming



Data: Graph G

Query $Q(x, y)$: *"Which* ***orange*** *nodes* x *have a directed path to which* ***blue*** *nodes* y*?"*

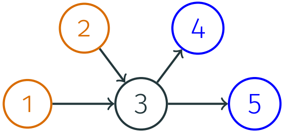
Preprocessing algorithm → $\{1, 2\} \times \{4, 5\}$ **Compressed representation** → **Enumeration algorithm**

x	y
1	4
1	5
2	4

Results

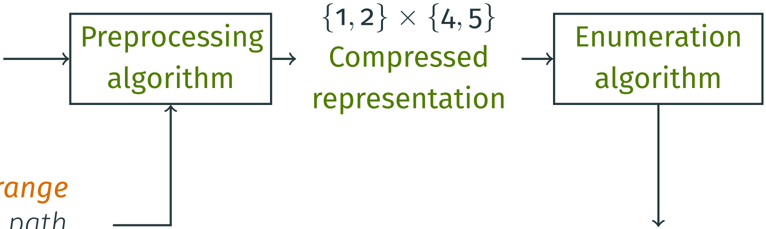
- **Usual complexity measure:** time to produce the **entire output**
- More precise measure: **enumeration algorithms**:
  - **Preprocessing time**: time to produce **compressed representation**
  - **Delay** between each consecutive output

Enumeration: Producing results in streaming



Data: Graph G

Query $Q(x, y)$: *“Which **orange** nodes **x** have a directed path to which **blue** nodes **y**?”*

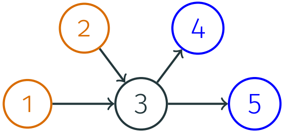


x	y
1	4
1	5
2	4
2	5

Results

- **Usual complexity measure:** time to produce the **entire output**
- More precise measure: **enumeration algorithms**:
  - **Preprocessing time**: time to produce **compressed representation**
  - **Delay** between each consecutive output

Enumeration: Producing results in streaming



Data: Graph G

Query $Q(x, y)$: *"Which **orange** nodes* x *have a directed path to which **blue** nodes* y*?"*

Preprocessing algorithm → $\{1, 2\} \times \{4, 5\}$ Compressed representation → Enumeration algorithm

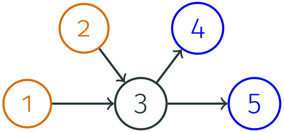
x	y
1	4
1	5
2	4
2	5

Results

- **Usual complexity measure:** time to produce the **entire output**
- More precise measure: **enumeration algorithms**:
  - **Preprocessing time**: time to produce **compressed representation**
  - **Delay** between each consecutive output

→ Test **existence of a result**, find **some results**, find **all results**...

Maintenance over dynamic data: Adapting to changes



Data: Graph G

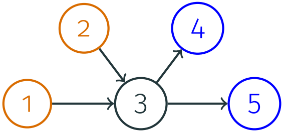
Query $Q(x, y)$: *“Which **orange** nodes **x** have a directed path to which **blue** nodes **y**?”*

Query evaluation

x	y
1	4
1	5
2	4
2	4

Results

Maintenance over dynamic data: Adapting to changes



Data: Graph G

Query $Q(x, y)$: *"Which **orange** nodes x have a directed path to which **blue** nodes y?"*

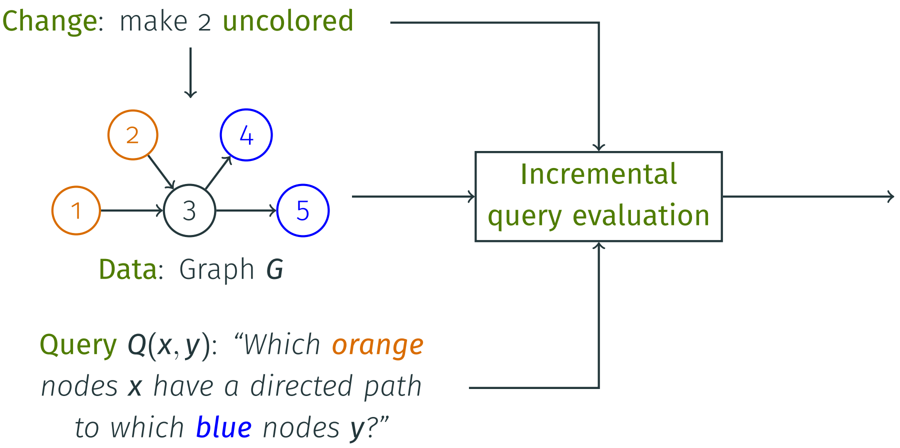
Incremental query evaluation

x	y
1	4
1	5
2	4
2	4

Results

- Whenever the data is **changed**, do not **recompute** the whole result

Maintenance over dynamic data: Adapting to changes



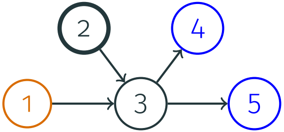
x	y
1	4
1	5
2	4
2	4

Results

- Whenever the data is changed, do not recompute the whole result

Maintenance over dynamic data: Adapting to changes

Change: make 2 **uncolored**



Data: Graph G

Query $Q(x, y)$: *"Which **orange** nodes **x** have a directed path to which **blue** nodes **y**?"*

Incremental query evaluation

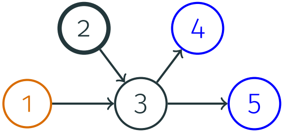
x	y
1	4
1	5
2	4
2	4

Results

- Whenever the data is **changed**, do not **recompute** the whole result

Maintenance over dynamic data: Adapting to changes

Change: make 2 **uncolored**



Data: Graph G

Query $Q(x, y)$: *“Which **orange** nodes **x** have a directed path to which **blue** nodes **y**?”*

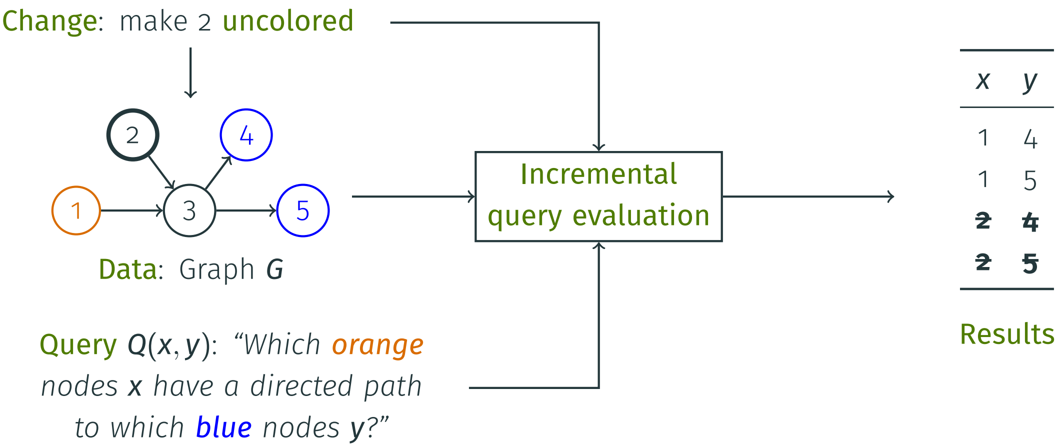
Incremental query evaluation

x	y
1	4
1	5
~~2~~	~~4~~
~~2~~	~~5~~

Results

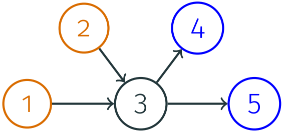
- Whenever the data is **changed**, do not **recompute** the whole result

Maintenance over dynamic data: Adapting to changes



- Whenever the data is **changed**, do not **recompute** the whole result
- **Relabeling updates** vs **more general updates**

Reliability: Probabilistic query evaluation



Data: Graph G

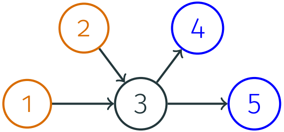
Query $Q(x, y)$: *"Which **orange** nodes **x** have a directed path to which **blue** nodes **y**?"*

Query evaluation

x	y
1	4
1	5
2	4
2	5

Results

Reliability: Probabilistic query evaluation



Data: Graph G

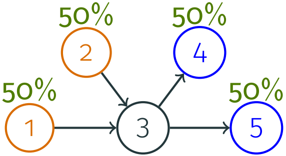
Query $Q(x, y)$: *"Which **orange** nodes x have a directed path to which **blue** nodes y?"*

Probabilistic query evaluation

x	y
1	4
1	5
2	4
2	5

Results

Reliability: Probabilistic query evaluation



Data: Graph G

Query $Q(x, y)$: *"Which **orange** nodes **x** have a directed path to which **blue** nodes **y**?"*

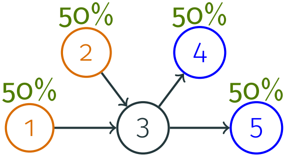
Probabilistic query evaluation

x	y
1	4
1	5
2	4
2	5

Results

- The color of each node is kept with a given **probability**, assuming **independence**

Reliability: Probabilistic query evaluation



Data: Graph G

Query $Q(x, y)$: *"Which orange nodes* x *have a directed path to which blue nodes* y*?"*

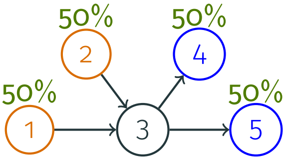
Probabilistic query evaluation

x	y	
1	4	25%
1	5	25%
2	4	25%
2	5	25%

Results

- The color of each node is kept with a given **probability**, assuming **independence**
- We want to know the **probability** of all results

Reliability: Probabilistic query evaluation



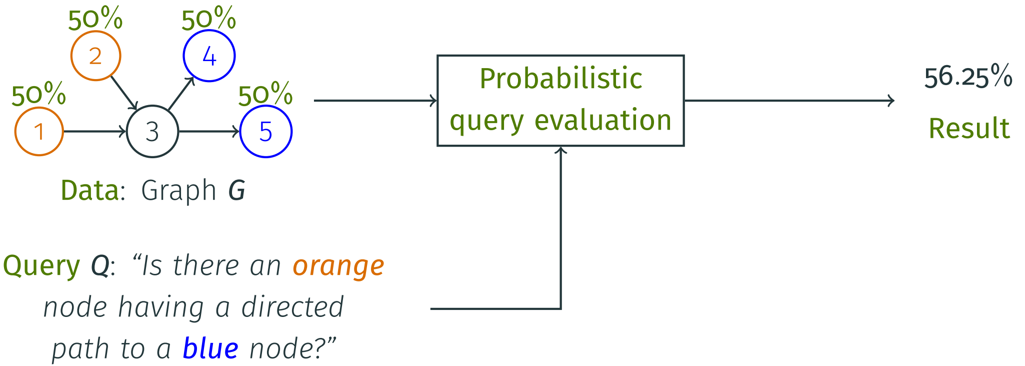
Data: Graph G

Probabilistic query evaluation

Query Q: *"Is there an **orange** node having a directed path to a **blue** node?"*

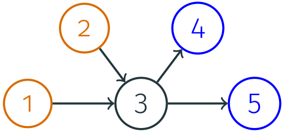
- The color of each node is kept with a given **probability**, assuming **independence**
- We want to know the **probability** of all results
- Here, more interesting: probability of the **Boolean query**

Reliability: Probabilistic query evaluation



- The color of each node is kept with a given **probability**, assuming **independence**
- We want to know the **probability** of all results
- Here, more interesting: probability of the **Boolean query**

Provenance circuits: A unified approach to these three problems

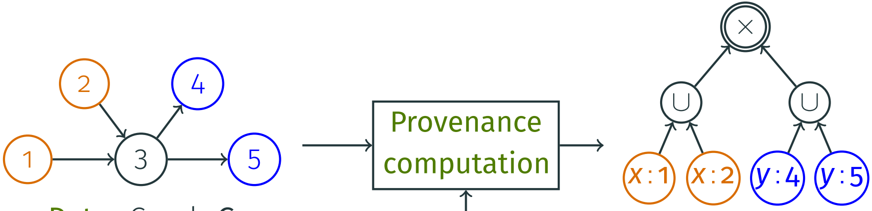


Data: Graph G

Provenance computation

Query $Q(x, y)$: *"Which **orange** nodes **x** have a directed path to which **blue** nodes **y**?"*

Provenance circuits: A unified approach to these three problems



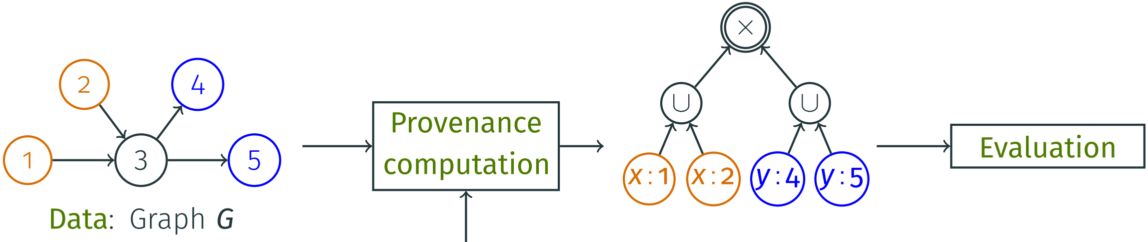
Data: Graph G

Query $Q(x, y)$: *"Which **orange** nodes **x** have a directed path to which **blue** nodes **y**?"*

Provenance circuit

- The **provenance circuit** describes how the query result depends on the data
- Show that it belongs to restricted **circuit classes** from **knowledge compilation**

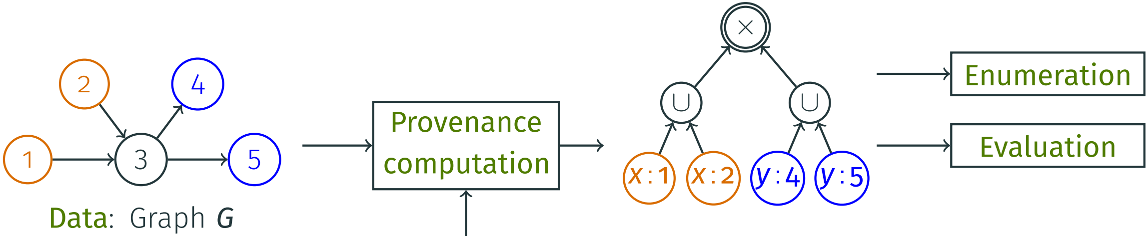
Provenance circuits: A unified approach to these three problems



Query $Q(x, y)$: *"Which* ***orange*** *nodes* $\boldsymbol{x}$ *have a directed path to which* ***blue*** *nodes* $\boldsymbol{y}$*?"*

- The **provenance circuit** describes how the query result depends on the data
- Show that it belongs to restricted **circuit classes** from **knowledge compilation**
- Use it for **evaluation**,

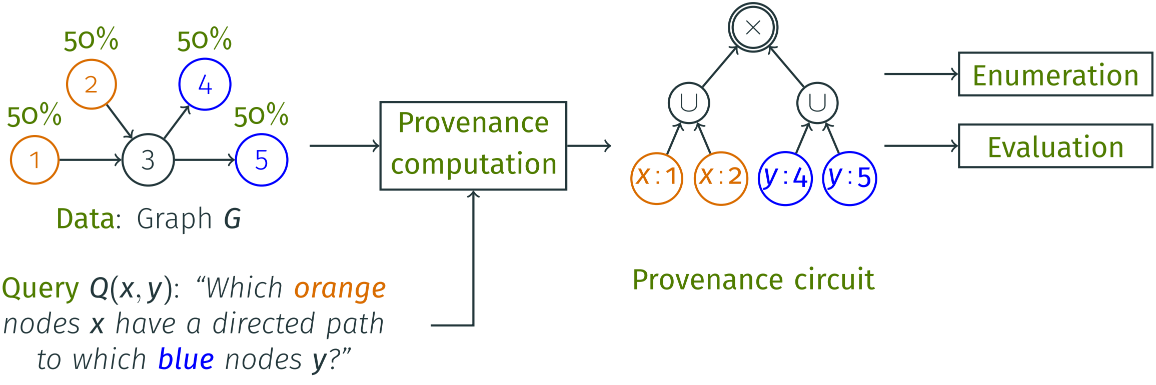
Provenance circuits: A unified approach to these three problems



Query $Q(x, y)$: *"Which orange nodes **x** have a directed path to which **blue** nodes **y**?"*

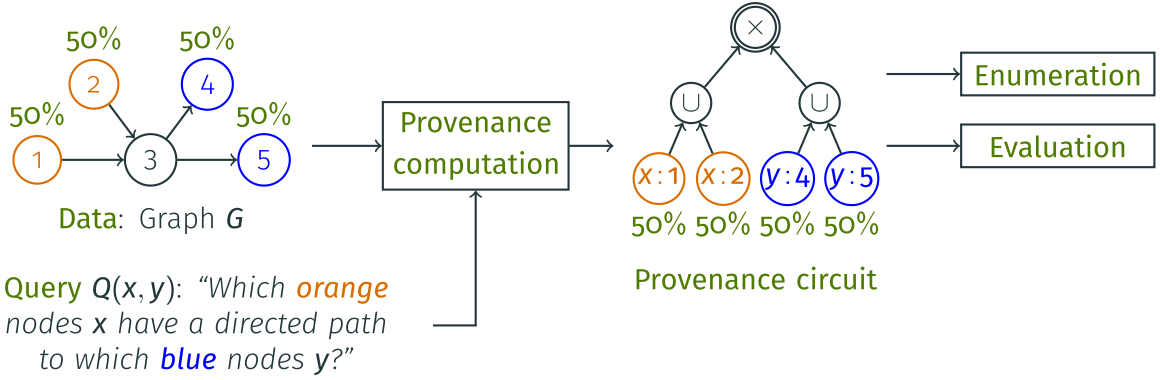
- The **provenance circuit** describes how the query result depends on the data
- Show that it belongs to restricted **circuit classes** from **knowledge compilation**
- Use it for **evaluation**, **enumeration**,

Provenance circuits: A unified approach to these three problems



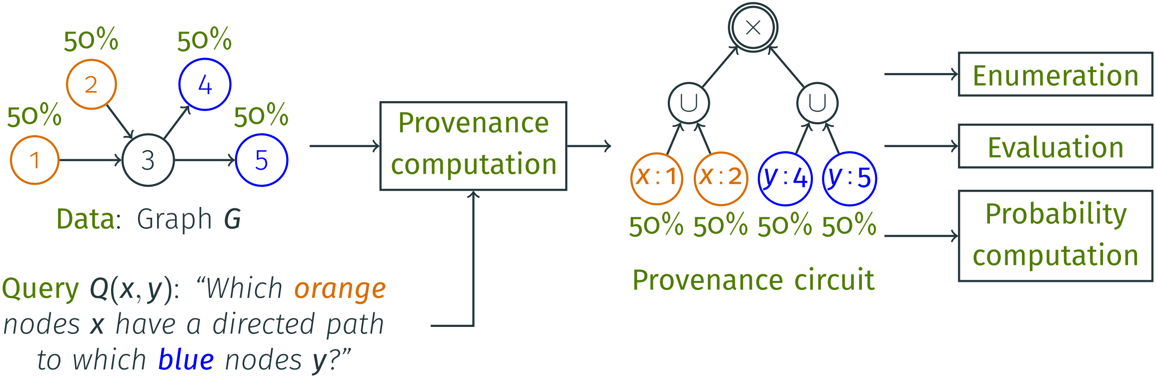
- The **provenance circuit** describes how the query result depends on the data
- Show that it belongs to restricted **circuit classes** from **knowledge compilation**
- Use it for **evaluation**, **enumeration**, **probability computation**

Provenance circuits: A unified approach to these three problems



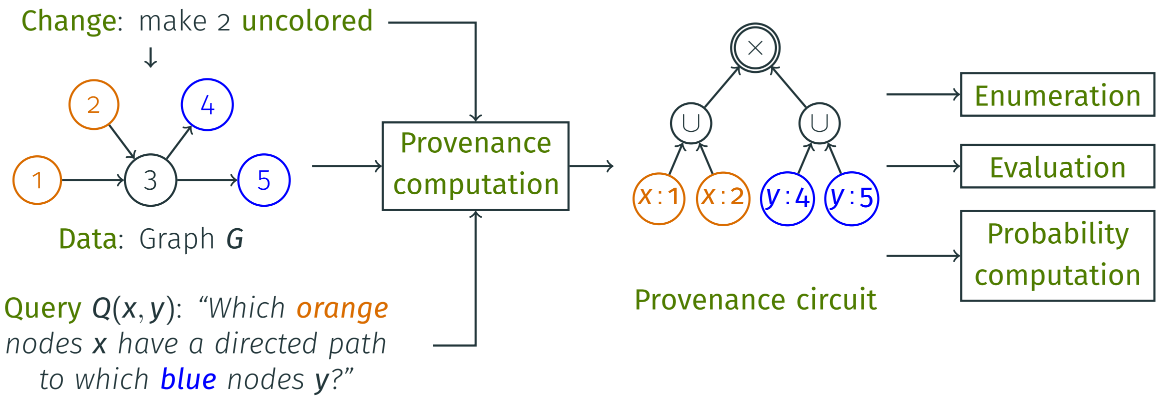
- The **provenance circuit** describes how the query result depends on the data
- Show that it belongs to restricted **circuit classes** from **knowledge compilation**
- Use it for **evaluation**, **enumeration**, **probability computation**

Provenance circuits: A unified approach to these three problems



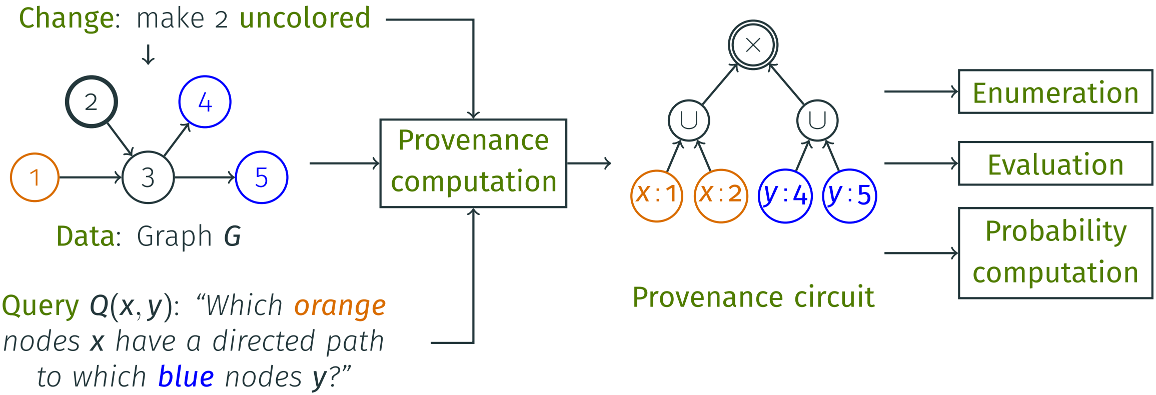
- The **provenance circuit** describes how the query result depends on the data
- Show that it belongs to restricted **circuit classes** from **knowledge compilation**
- Use it for **evaluation**, **enumeration**, **probability computation**

Provenance circuits: A unified approach to these three problems



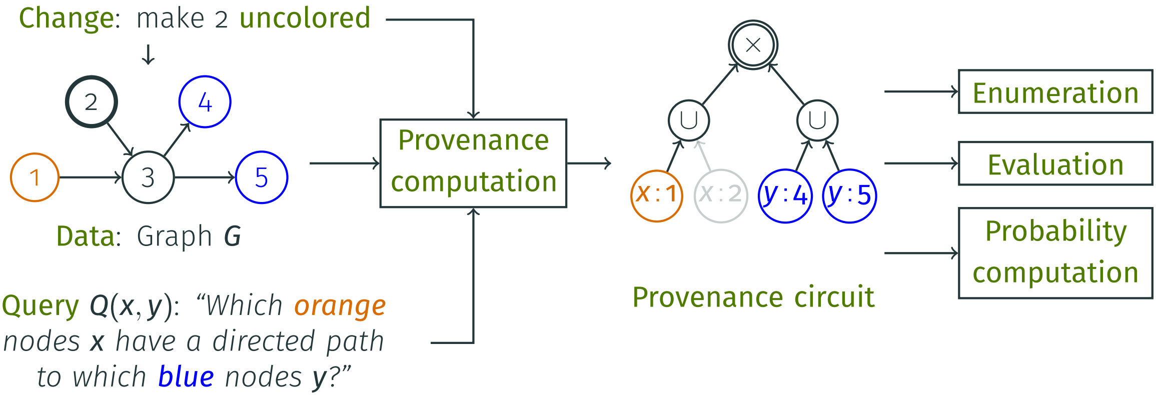
- The **provenance circuit** describes how the query result depends on the data
- Show that it belongs to restricted **circuit classes** from **knowledge compilation**
- Use it for **evaluation**, **enumeration**, **probability computation**
- Update it if there are **changes** on the data

Provenance circuits: A unified approach to these three problems



- The **provenance circuit** describes how the query result depends on the data
- Show that it belongs to restricted **circuit classes** from **knowledge compilation**
- Use it for **evaluation**, **enumeration**, **probability computation**
- Update it if there are **changes** on the data

Provenance circuits: A unified approach to these three problems



- The **provenance circuit** describes how the query result depends on the data
- Show that it belongs to restricted **circuit classes** from **knowledge compilation**
- Use it for **evaluation**, **enumeration**, **probability computation**
- Update it if there are **changes** on the data

Roadmap of the presentation

- Present data and query formalisms:
  - $\rightarrow$ Monadic second-order logic (MSO) on words/trees

Roadmap of the presentation

- Present data and query formalisms:
  $\rightarrow$ Monadic second-order logic (MSO) on words/trees
- Results on enumeration

Roadmap of the presentation

- Present **data** and **query** formalisms:
  - $\rightarrow$ **Monadic second-order** logic (MSO) on words/trees
- Results on **enumeration**
- Results on **incremental maintenance**

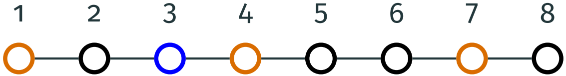
Roadmap of the presentation

- Present **data** and **query** formalisms:
  - $\rightarrow$ **Monadic second-order** logic (MSO) on words/trees
- Results on **enumeration**
- Results on **incremental maintenance**
- Results on **probabilistic query evaluation**

Context

Families of data

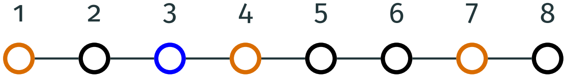
- Words:

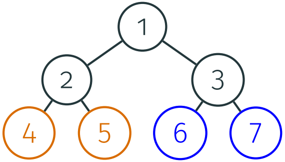


less expressive

more expressive

Families of data

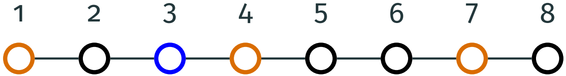
- Words: 

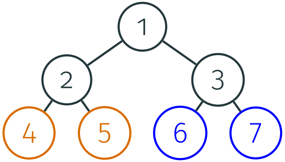
- Trees: 

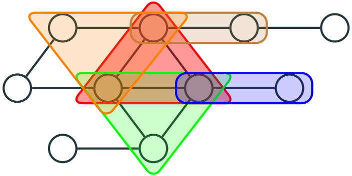
less expressive

more expressive

Families of data

- Words: 

- Trees: 

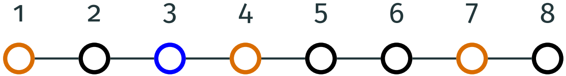
- Bounded-treewidth graphs: 

less expressive

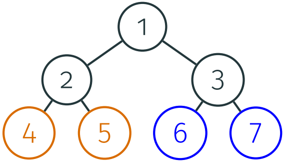
more expressive

Families of data

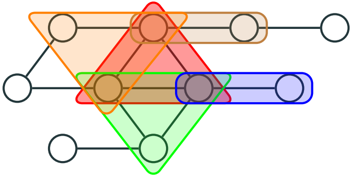
- **Words**:



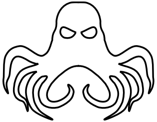
- **Trees**:



- **Bounded-treewidth** graphs:



- **Many other classes** of graphs and relational structures:

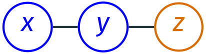


less expressive

more expressive

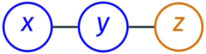
Query languages

From **least** to **most** expressive:

- **Conjunctive queries** (CQs): find a pattern
  - $Q(x, y)$: *"Find two adjacent* ***blue*** *nodes* $\boldsymbol{x}$ *and* $\boldsymbol{y}$ *with* $\boldsymbol{y}$ *having an* ***orange*** *neighbor"*
  - $Q(x, y) : \exists z$ 

Query languages

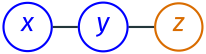
From **least** to **most** expressive:

- **Conjunctive queries** (CQs): find a pattern
  - $Q(x, y)$: "*Find two adjacent* ***blue*** *nodes* x *and* y *with* y *having an* ***orange*** *neighbor*"
  - $Q(x, y)$: $\exists z$ 

- **Unions of CQs** (UCQs): disjunction of CQs
  - $Q(x, y)$: "*Find two adjacent* ***blue*** *nodes* x *and* y *or two adjacent* ***orange*** *nodes* x *and* y"

Query languages

From **least** to **most** expressive:

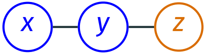
- **Conjunctive queries** (CQs): find a pattern
  - $Q(x, y)$: *"Find two adjacent* ***blue*** *nodes* x *and* y *with* y *having an* ***orange*** *neighbor"*
  - $Q(x, y) : \exists z$ 

- **Unions of CQs** (UCQs): disjunction of CQs
  - $Q(x, y)$: *"Find two adjacent* ***blue*** *nodes* x *and* y *or two adjacent* ***orange*** *nodes* x *and* y*"*
- **First-order logic** (FO):

  $\rightarrow$ conjunction, disjunction, **negation**, existential quantification, **universal quantification**

Query languages

From least to most expressive:

- Conjunctive queries (CQs): find a pattern
  - $Q(x, y)$: *"Find two adjacent* ***blue*** *nodes* x *and* y *with* y *having an* ***orange*** *neighbor"*
  - $Q(x, y) : \exists z$ 

- Unions of CQs (UCQs): disjunction of CQs
  - $Q(x, y)$: *"Find two adjacent* ***blue*** *nodes* x *and* y *or two adjacent* ***orange*** *nodes* x *and* y*"*
- First-order logic (FO):

  $\rightarrow$ conjunction, disjunction, negation, existential quantification, universal quantification
- Monadic second-order logic (MSO): extend FO with quantification over sets
  - Equivalent to finite automata on words, trees, tree encodings

Enumeration

Word automata with captures

On words, MSO queries are equivalent to automata

Word automata with captures

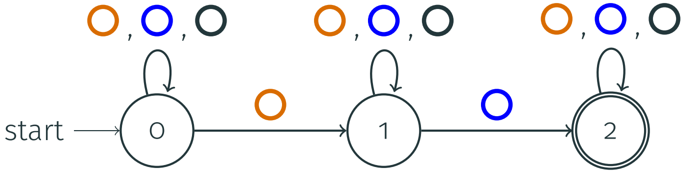
On words, MSO queries are equivalent to automata

Q: *“Is there an orange node before a blue node?”*

Word automata with captures

On words, MSO queries are equivalent to automata

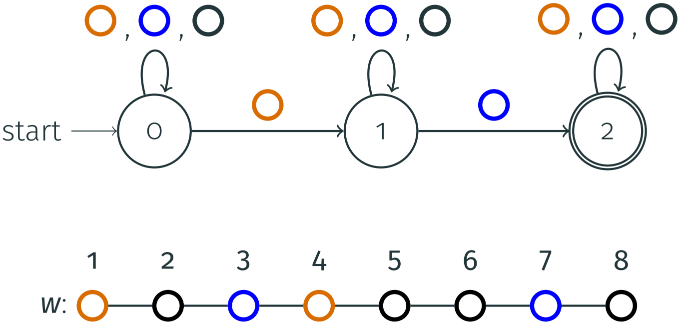
Q: *“Is there an orange node before a blue node?”*



Word automata with captures

On **words**, MSO queries are equivalent to **automata**

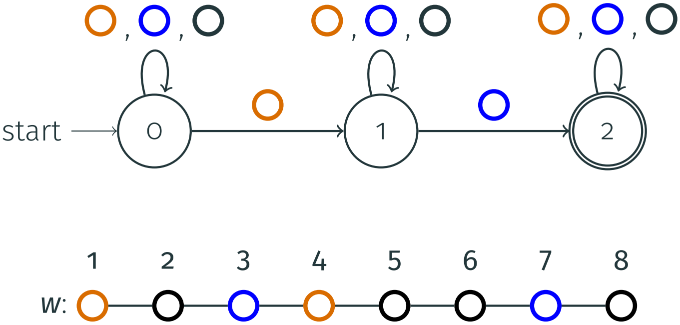
Q: *"Is there an **orange** node before a **blue** node?"*



Word automata with captures

On **words**, MSO queries are equivalent to **automata**

Q: *"Is there an* ***orange*** *node before a* ***blue*** *node?"*

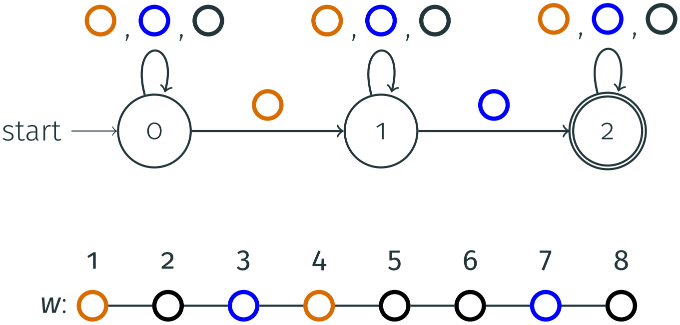


Result: YES

Word automata with captures

On **words**, MSO queries are equivalent to **automata with captures**

Q: *“Is there an* ***orange*** *node before a* ***blue*** *node?”*

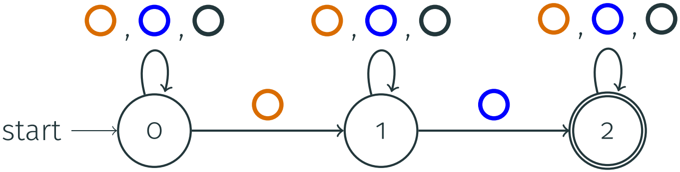


Result: YES

Word automata with captures

On **words**, MSO queries are equivalent to **automata with captures**

$Q(x, y)$: *"Find an **orange** node x before a **blue** node y"*

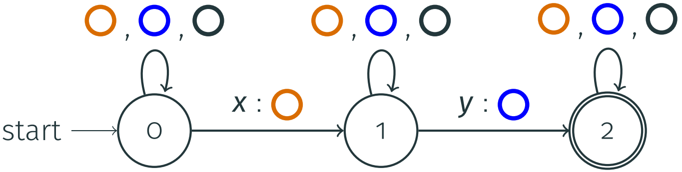


Result: YES

Word automata with captures

On **words**, MSO queries are equivalent to **automata with captures**

$Q(x, y)$: *"Find an **orange** node x before a **blue** node y"*

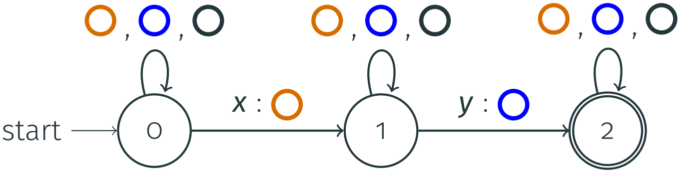


Result: YES

Word automata with captures

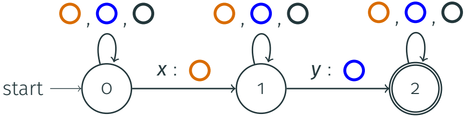
On **words**, MSO queries are equivalent to **automata with captures**

$Q(x, y)$: *"Find an **orange** node x before a **blue** node y"*



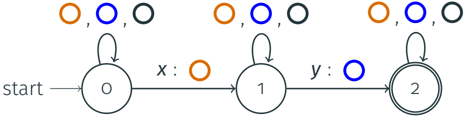
Results: $(x\!:\!1, y\!:\!3), (x\!:\!1, y\!:\!7), (x\!:\!4, y\!:\!7)$

Provenance circuit computation: Product construction



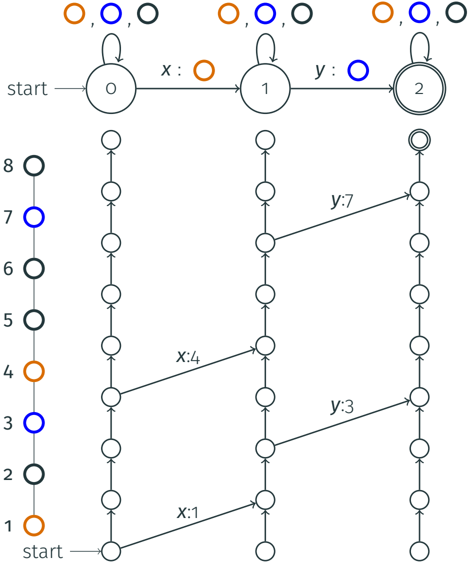
- Product of word and automaton

Provenance circuit computation: Product construction



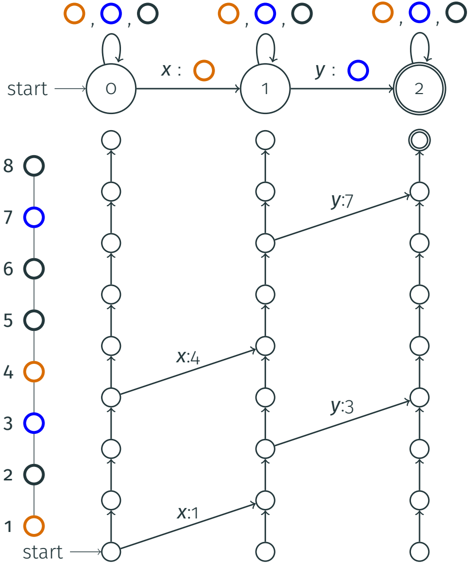
- Product of word and automaton

Provenance circuit computation: Product construction



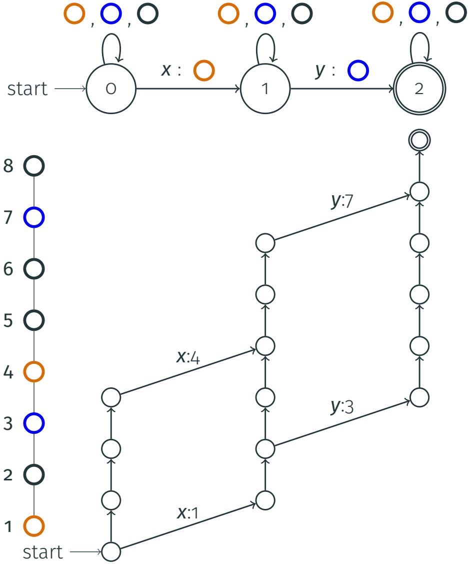
- Product of word and automaton

Provenance circuit computation: Product construction



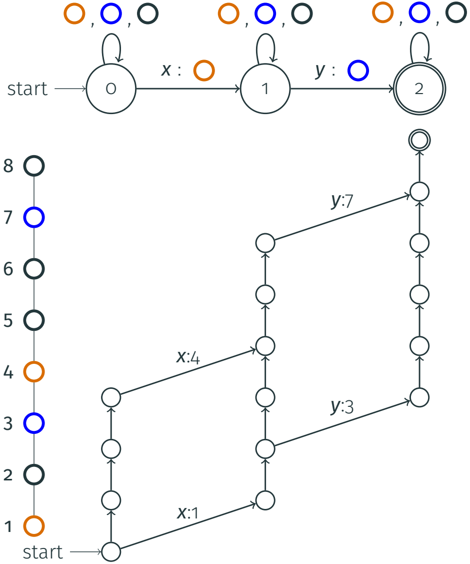
- **Product** of word and automaton
- **Trim** nodes that are not reachable/co-reachable

Provenance circuit computation: Product construction



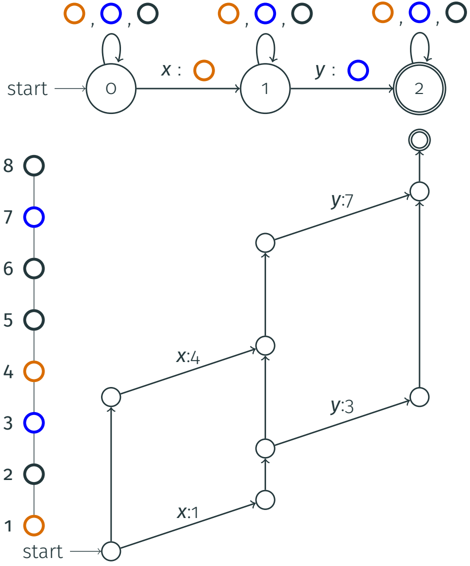
- **Product** of word and automaton
- **Trim** nodes that are not reachable/co-reachable

Provenance circuit computation: Product construction



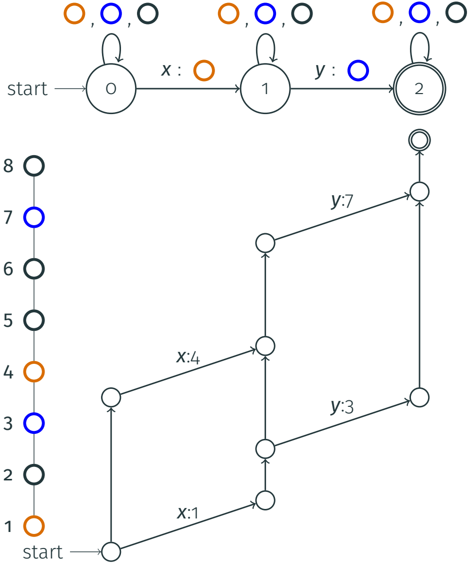
- **Product** of word and automaton
- **Trim** nodes that are not reachable/co-reachable
- **Collapse** transitions with no assignments

Provenance circuit computation: Product construction



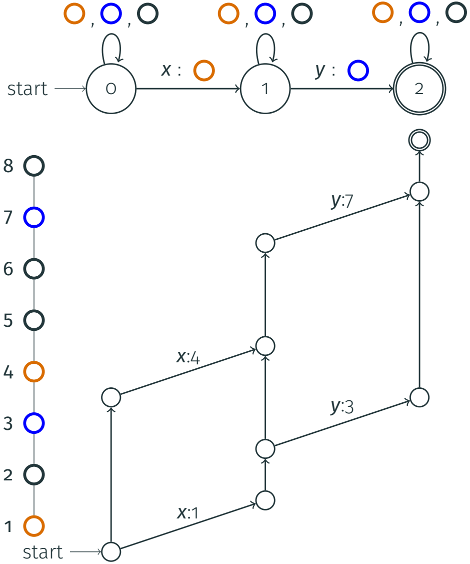
- **Product** of word and automaton
- **Trim** nodes that are not reachable/co-reachable
- **Collapse** transitions with no assignments

Provenance circuit computation: Product construction

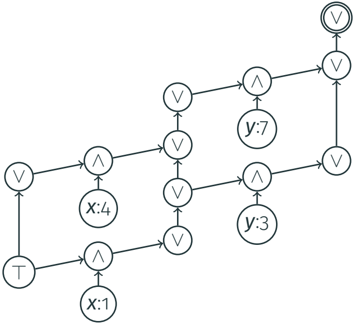


- **Product** of word and automaton
- **Trim** nodes that are not reachable/co-reachable
- **Collapse** transitions with no assignments
- Equivalent **provenance circuit**:

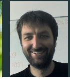
Provenance circuit computation: Product construction



- **Product** of word and automaton
- **Trim** nodes that are not reachable/co-reachable
- **Collapse** transitions with no assignments
- Equivalent **provenance circuit**:

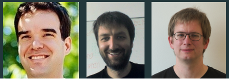


Enumerating the results of MSO queries



We can enumerate query results (= satisfying assignments) using the provenance circuit

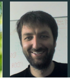
Enumerating the results of MSO queries



We can **enumerate query results** (= satisfying assignments) using the provenance circuit

- Generalizes from words to **trees**

Enumerating the results of MSO queries


We can **enumerate query results** (= satisfying assignments) using the provenance circuit

- Generalizes from words to **trees**
- Also works for **non-deterministic automata**

Enumerating the results of MSO queries



We can **enumerate query results** (= satisfying assignments) using the provenance circuit

- Generalizes from words to **trees**
- Also works for **non-deterministic automata**

Theorem (ICDT'19 on words, PODS'19 on trees; with Bourhis, Mengel, Niewerth)

Given an automaton with captures A with constant number of variables, given a word w, we can enumerate the results of A on w with preprocessing $O(Poly(|A|) \times |w|)$ and delay $O(Poly(|A|))$.

Enumerating the results of MSO queries



We can **enumerate query results** (= satisfying assignments) using the provenance circuit

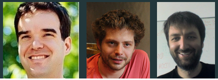
- Generalizes from words to **trees**
- Also works for **non-deterministic automata**

Theorem (ICDT'19 on words, PODS'19 on trees; with Bourhis, Mengel, Niewerth)

Given an automaton with captures A with constant number of variables, given a word w, we can enumerate the results of A on w with preprocessing $O(Poly(|A|) \times |w|)$ and delay $O(Poly(|A|))$.

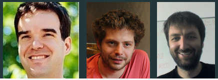
Known result [Bagan, 2006, Kazana and Segoufin, 2013] but polynomial dependency in A

Enumeration for general circuits



We can enumerate the satisfying assignments of arbitrary circuits in d-SDNNF:

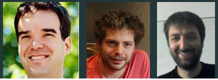
Enumeration for general circuits



We can enumerate the **satisfying assignments** of **arbitrary** circuits in **d-SDNNF**:

- **Decomposable:** no variable occurs on both inputs of an $\land$-gate

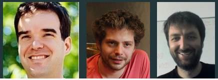
Enumeration for general circuits



We can enumerate the satisfying assignments of arbitrary circuits in d-SDNNF:

- Decomposable: no variable occurs on both inputs of an $\wedge$-gate
- Deterministic: inputs to an $\vee$-gate are mutually exclusive

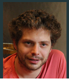
Enumeration for general circuits



We can enumerate the satisfying assignments of arbitrary circuits in d-SDNNF:

- **Decomposable:** no variable occurs on both inputs of an $\wedge$-gate
- **Deterministic:** inputs to an $\vee$-gate are mutually exclusive
- **Negation normal form:** negation on leaves

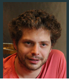
Enumeration for general circuits


We can enumerate the **satisfying assignments** of **arbitrary** circuits in **d-SDNNF**:

- **Decomposable:** no variable occurs on both inputs of an $\wedge$-gate
- **Deterministic:** inputs to an $\vee$-gate are mutually exclusive
- **Negation normal form:** negation on leaves
- **Structured** by a v-tree

Enumeration for general circuits

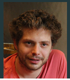

We can enumerate the **satisfying assignments** of **arbitrary** circuits in **d-SDNNF**:

- **Decomposable:** no variable occurs on both inputs of an $\land$-gate
- **Deterministic:** inputs to an $\lor$-gate are mutually exclusive
- **Negation normal form:** negation on leaves
- **Structured** by a v-tree

Theorem (ICALP'17; with Bourhis, Jachiet, Mengel)

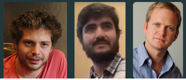
*Given a d-SDNNF C and a v-tree that structures C, we can enumerate the satisfying assignments of C with **linear preprocessing** and **output-linear delay**.*

Beyond regular languages

Generalize automata with captures into annotation context-free grammars

Beyond regular languages



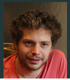
Generalize automata with captures into **annotation context-free grammars**

$Q(x, y)$: *"Find all endpoints* x, y *of factors of the form* ${\color{orange}\bigcirc}^{n}\ {\color{blue}\bigcirc}^{n}$*"*

$$S \to \Sigma^* \ (x : {\color{orange}\bigcirc}\,)\ A\ (y : {\color{blue}\bigcirc}\,)\ \Sigma^*$$

$$A \to {\color{orange}\bigcirc}\ A\ {\color{blue}\bigcirc} \mid \epsilon$$

Beyond regular languages



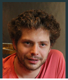
Generalize automata with captures into annotation context-free grammars

$Q(x, y)$: *"Find all endpoints* x, y *of factors of the form* $\textcolor{orange}{\bigcirc}^n \; \textcolor{blue}{\bigcirc}^n$*"*

$$S \to \Sigma^* \; (x : \textcolor{orange}{\bigcirc}) \; A \; (y : \textcolor{blue}{\bigcirc}) \; \Sigma^*$$
$$A \to \textcolor{orange}{\bigcirc} \; A \; \textcolor{blue}{\bigcirc} \mid \epsilon$$

Annotation grammar must be input-output-unambiguous: no result is captured twice

Beyond regular languages



Generalize automata with captures into **annotation context-free grammars**

$Q(x, y)$: *"Find all endpoints* x, y *of factors of the form* $\bigcirc^{n}\ \bigcirc^{n}$*"*

$$S \to \Sigma^* \ (x : \bigcirc\ )\ A\ (y : \bigcirc\ )\ \Sigma^*$$

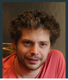
$$A \to \bigcirc\ A\ \bigcirc \mid \epsilon$$

Annotation grammar must be **input-output-unambiguous**: no result is captured twice

Theorem (PODS'22; with Jachiet, Muñoz, Riveros)

Given an unambiguous annotation grammar $\mathbf{G}$ *and word* $\mathbf{w}$*, we can enumerate the results of* $\mathbf{G}$ *on* $\mathbf{w}$ *with preprocessing* $O(|G| \times |w|^3)$ *and* ***output-linear*** *delay*

Beyond regular languages



Generalize automata with captures into annotation context-free grammars

$Q(x, y)$: *"Find all endpoints* x, y *of factors of the form* $\textcolor{orange}{\bigcirc}^n\ \textcolor{blue}{\bigcirc}^n$*"*

$$S \to \Sigma^* \ (x : \textcolor{orange}{\bigcirc}\ )\ A\ (y : \textcolor{blue}{\bigcirc}\ )\ \Sigma^*$$
$$A \to \textcolor{orange}{\bigcirc}\ A\ \textcolor{blue}{\bigcirc} \mid \epsilon$$

Annotation grammar must be input-output-unambiguous: no result is captured twice

Theorem (PODS'22; with Jachiet, Muñoz, Riveros)

Given an unambiguous annotation grammar $\mathbf{G}$ *and word* $\mathbf{w}$*, we can enumerate the results of* $\mathbf{G}$ *on* $\mathbf{w}$ *with preprocessing* $O(|G| \times |w|^3)$ *and output-linear delay*

Better preprocessing time for restricted grammar classes

Maintenance

Maintenance for MSO enumeration on trees

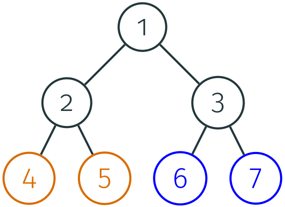
We use provenance circuits for automata on words and trees

$Q(x, y)$: *"Find pairs of an orange node x and a blue node y"*

Maintenance for MSO enumeration on trees

We use provenance circuits for automata on words and trees

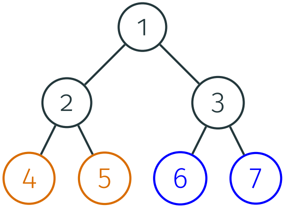
$Q(x, y)$: *"Find pairs of an orange node x and a blue node y"*

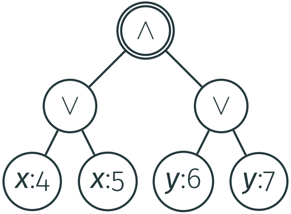


Maintenance for MSO enumeration on trees

We use provenance circuits for automata on **words** and **trees**

$Q(x, y)$: *"Find pairs of an* ***orange*** *node* x *and a* ***blue*** *node* y*"*

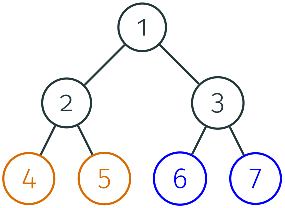


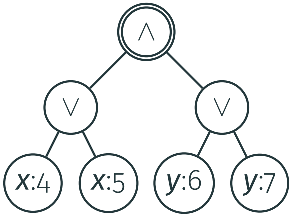


Maintenance for MSO enumeration on trees

We use provenance circuits for automata on **words** and **trees**

$Q(x, y)$: *"Find pairs of an* ***orange*** *node* x *and a* ***blue*** *node* y*"*



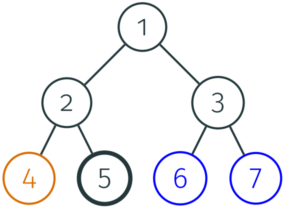


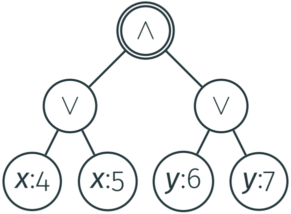
What happens if the tree is **modified**?

Maintenance for MSO enumeration on trees

We use provenance circuits for automata on **words** and **trees**

$Q(x, y)$: *"Find pairs of an* ***orange*** *node* x *and a* ***blue*** *node* y*"*



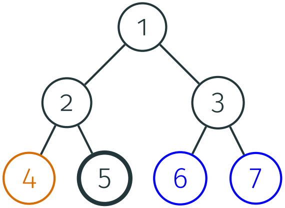


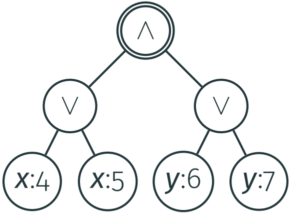
What happens if the tree is **modified**?

Maintenance for MSO enumeration on trees

We use provenance circuits for automata on **words** and **trees**

$Q(x, y)$: *“Find pairs of an* ***orange*** *node* x *and a* ***blue*** *node* y*”*





What happens if the tree is **modified**?

- Can we update the **provenance circuit** instead of recomputing it from scratch?

Maintenance for MSO enumeration on trees

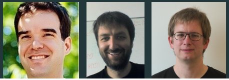
We use provenance circuits for automata on **words** and **trees**

$Q(x, y)$: *"Find pairs of an* ***orange*** *node* x *and a* ***blue*** *node* y*"*

What happens if the tree is **modified**?

- Can we update the **provenance circuit** instead of recomputing it from scratch?
- Can we avoid re-running the **preprocessing phase** of the enumeration?

Maintaining MSO enumeration structures under relabelings



We can show that relabeling updates to the tree T can be handled in $O(\text{height}(T))$

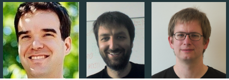
Maintaining MSO enumeration structures under relabelings



We can show that relabeling updates to the tree T can be handled in $O(\text{height}(T))$

$\rightarrow$ The provenance circuit computation and enumeration preprocessing are bottom-up

Maintaining MSO enumeration structures under relabelings



We can show that relabeling updates to the tree T can be handled in $O(\text{height}(T))$

$\rightarrow$ The provenance circuit computation and enumeration preprocessing are bottom-up

It suffices to balance the tree at the start (uses balanced tree decompositions)

Maintaining MSO enumeration structures under relabelings



We can show that **relabeling updates** to the tree T can be handled in $O(\text{height}(T))$

$\rightarrow$ The provenance circuit computation and enumeration preprocessing are **bottom-up**

It suffices to **balance the tree** at the start (uses balanced tree decompositions)

Theorem (ICDT'18; with Bourhis, Mengel)

For any fixed MSO query Q*, given an input tree* T*, we can enumerate the results of* Q *on* T *with* ***linear preprocessing*** *and* ***output-linear delay****, and we can handle* ***relabeling updates*** to T ***in time*** $O(\log |T|)$.

Maintaining MSO enumeration structures under relabelings



We can show that **relabeling updates** to the tree T can be handled in $O(\text{height}(T))$

$\rightarrow$ The provenance circuit computation and enumeration preprocessing are **bottom-up**

It suffices to **balance the tree** at the start (uses balanced tree decompositions)

Theorem (ICDT'18; with Bourhis, Mengel)

For any fixed MSO query Q*, given an input tree* T*, we can enumerate the results of* Q *on* T *with* ***linear preprocessing*** *and* ***output-linear delay****, and we can handle* ***relabeling updates*** *to* T ***in time*** $O(\log |T|)$.

Same for updates that **change the tree structure** (PODS'19; with Bourhis, Mengel, Niewerth) assuming we have an algorithm to **keep the tree balanced**

Improving the logarithmic complexity

- The update time is $O(\log n)$ and there is a lower bound of $\Omega(\log n / \log\log n)$
  - $\rightarrow$ Already for **Boolean queries** on **words** under **relabeling updates**

Improving the logarithmic complexity

- The update time is $O(\log n)$ and there is a lower bound of $\Omega(\log n / \log \log n)$
  - $\rightarrow$ Already for **Boolean queries** on **words** under **relabeling updates**
- Yet, we can do **better** for some queries, e.g.:

  Q: *"Is there both an **orange** node and a **blue** node?"*

Improving the logarithmic complexity

- The update time is $O(\log n)$ and there is a lower bound of $\Omega(\log n / \log \log n)$
  $\rightarrow$ Already for **Boolean queries** on **words** under **relabeling updates**

- Yet, we can do **better** for some queries, e.g.:

  Q: *"Is there both an **orange** node and a **blue** node?"*

- Simply **maintain the counts**! update time $O(1)$

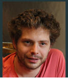
Improving the logarithmic complexity

- The update time is $O(\log n)$ and there is a lower bound of $\Omega(\log n / \log \log n)$
  - $\rightarrow$ Already for **Boolean queries** on **words** under **relabeling updates**
- Yet, we can do **better** for some queries, e.g.:

  Q: *"Is there both an **orange** node and a **blue** node?"*
- Simply **maintain the counts**! update time $O(1)$

$\rightarrow$ For a fixed language L, given a word w of length n, what is the **best update time** to maintain membership of w to L under relabelings?

Incremental maintenance for regular word languages

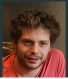


We define regular language classes **QLZG** and **QSG** such that:

Theorem (ICALP'21; with Jachiet, Paperman)

Consider the problem of maintaining membership to a regular language L *on words under relabeling updates*

Incremental maintenance for regular word languages



We define regular language classes **QLZG** and **QSG** such that:

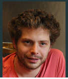
Theorem (ICALP'21; with Jachiet, Paperman)

Consider the problem of maintaining membership to a regular language L *on words under relabeling updates*

- *If* L *is in* **QLZG**, *then the problem is in* $O(1)$

QLZG: in $O(1)$

Incremental maintenance for regular word languages

We define regular language classes **QLZG** and **QSG** such that:

Theorem (ICALP'21; with Jachiet, Paperman)

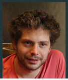
Consider the problem of maintaining membership to a regular language L on words under relabeling updates

- *If L is in* **QLZG**, *then the problem is in $O(1)$*
- *If L is in* **QSG** $\setminus$ **QLZG**, *then the problem is in $O(\log \log n)$ and conditionally not in $O(1)$*

QLZG: in $O(1)$

QSG: in $O(\log \log n)$
not in $O(1)$?

Incremental maintenance for regular word languages




We define regular language classes **QLZG** and **QSG** such that:

Theorem (ICALP'21; with Jachiet, Paperman)

Consider the problem of maintaining membership to a regular language L on words under relabeling updates

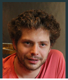
- *If L is in* **QLZG**, *then the problem is in* $O(1)$
- *If L is in* **QSG** $\setminus$ **QLZG**, *then the problem is in* $O(\log\log n)$ *and conditionally* ***not in*** $O(1)$
- *If L is not in* **QSG**, *then the problem is in* $\Theta(\log n / \log\log n)$

QLZG: in $O(1)$

QSG: in $O(\log\log n)$ not in $O(1)$?

All: in $\Theta(\log n / \log\log n)$

Incremental maintenance for regular word languages



We define regular language classes **QLZG** and **QSG** such that:

Theorem (ICALP'21; with Jachiet, Paperman)

Consider the problem of maintaining membership to a regular language L on words under relabeling updates

- *If L is in* **QLZG***, then the problem is in $O(1)$*
- *If L is in* **QSG** $\setminus$ **QLZG***, then the problem is in $O(\log\log n)$ and conditionally* ***not in $O(1)$***
- *If L is not in* **QSG***, then the problem is in $\Theta(\log n / \log\log n)$*

QLZG: in $O(1)$

QSG: in $O(\log\log n)$
not in $O(1)$?

All: in $\Theta(\log n / \log\log n)$

- **QLZG**: "in all submonoids of the stable semigroup, all subgroup elements are central"
  $\to$ Commutative languages, finite languages, disjoint shuffles, modulo, nearby positions...
- **QSG**: "the stable semigroup satisfies the equation $x^{\omega+1}yx^{\omega} = x^{\omega}yx^{\omega+1}$"
  $\to$ Aperiodic languages, tame combinations of aperiodic and commutative languages...

Reliability

Probabilistic query evaluation (PQE)

Tuple-independent probabilistic data (TID): facts carry independent probabilities

Probabilistic query evaluation (PQE)

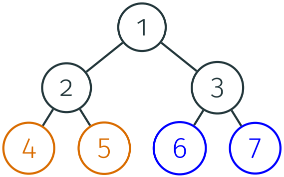
Tuple-independent probabilistic data (TID): facts carry independent probabilities

Q: *"There is both an orange node and a blue node"*

Probabilistic query evaluation (PQE)

Tuple-independent probabilistic data (TID): facts carry independent probabilities

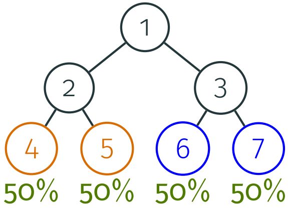
Q: *"There is both an* ***orange*** *node and a* ***blue*** *node"*



Probabilistic query evaluation (PQE)

Tuple-independent probabilistic data (TID): facts carry independent probabilities

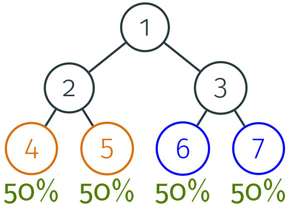
Q: *“There is both an **orange** node and a **blue** node”*



Probabilistic query evaluation (PQE)

Tuple-independent probabilistic data (TID): facts carry independent probabilities

Q: *"There is both an* ***orange*** *node and a* ***blue*** *node"*

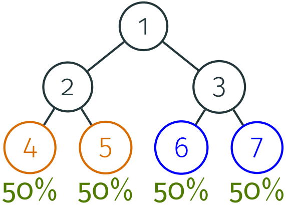


PQE(Q): compute the **total probability** that Q is satisfied, here:

Probabilistic query evaluation (PQE)

Tuple-independent probabilistic data (TID): facts carry independent probabilities

Q: *"There is both an* ***orange*** *node and a* ***blue*** *node"*

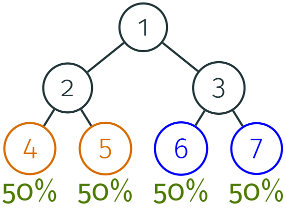


PQE(Q): compute the **total probability** that Q is satisfied, here: **56.25%**

Probabilistic query evaluation (PQE)

Tuple-independent probabilistic data (TID): facts carry independent probabilities

Q: *"There is both an **orange** node and a **blue** node"*



PQE(Q): compute the **total probability** that Q is satisfied, here: **56.25%**

- Known dichotomy for PQE on **unions of conjunctive queries** (on arbitrary data) [Dalvi and Suciu, 2013]: the problem is either **#P-hard** or **in PTIME**

Probabilistic query evaluation on trees via circuits

For MSO queries on trees, we can solve PQE using d-SDNNF provenance circuits!

Probabilistic query evaluation on trees via circuits

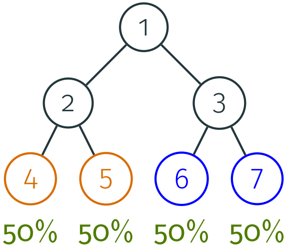
For MSO queries on trees, we can solve PQE using d-SDNNF provenance circuits!

Q: *"There is both an orange node and a blue node"*

Probabilistic query evaluation on trees via circuits

For MSO queries on trees, we can solve PQE using d-SDNNF provenance circuits!

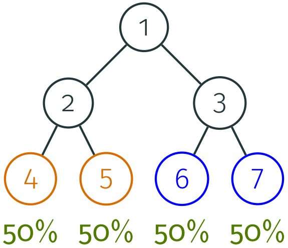
Q: *"There is both an orange node and a blue node"*

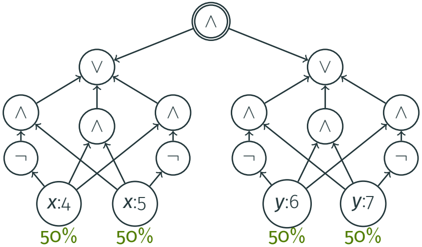


Probabilistic query evaluation on trees via circuits

For MSO queries on trees, we can solve PQE using d-SDNNF provenance circuits!

Q: *"There is both an* ***orange*** *node and a* ***blue*** *node"*

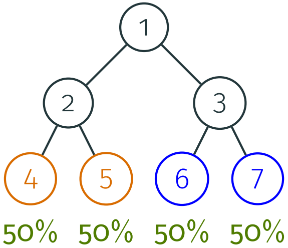


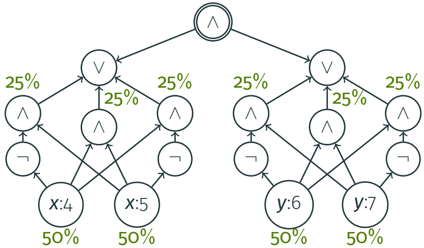


Probabilistic query evaluation on trees via circuits

For **MSO queries** on **trees**, we can solve PQE using **d-SDNNF provenance circuits**!

Q: *"There is both an **orange** node and a **blue** node"*

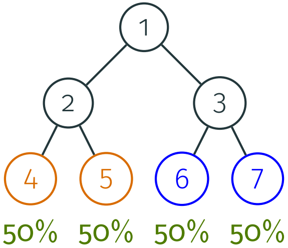


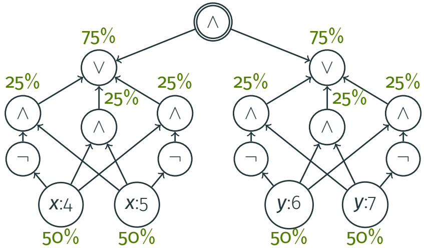


Probabilistic query evaluation on trees via circuits

For **MSO queries** on **trees**, we can solve PQE using **d-SDNNF provenance circuits**!

Q: *"There is both an **orange** node and a **blue** node"*

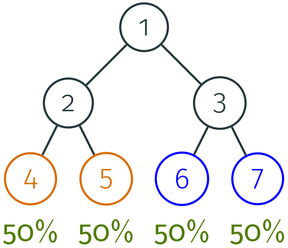


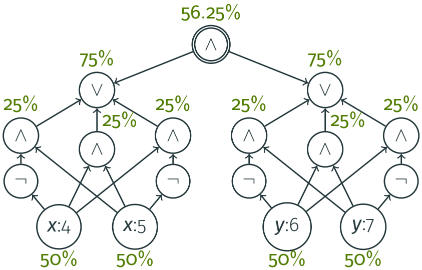


Probabilistic query evaluation on trees via circuits

For **MSO queries** on **trees**, we can solve PQE using **d-SDNNF provenance circuits**!

Q: *“There is both an* ***orange*** *node and a* ***blue*** *node”*

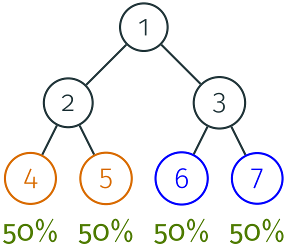


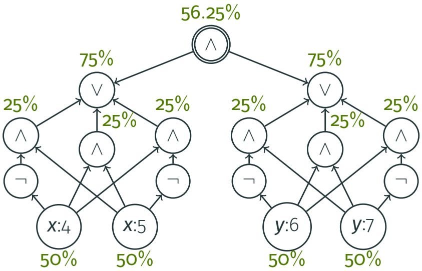


Probabilistic query evaluation on trees via circuits

For MSO queries on trees, we can solve PQE using d-SDNNF provenance circuits!

Q: *"There is both an **orange** node and a **blue** node"*





- Probability of $\wedge$ is the **product** of the probabilities (uses decomposability)
- Probability of $\vee$ is the **sum** of the probabilities (uses determinism)

Intractability of probabilistic query evaluation in the general case

What about more general data?

Intractability of probabilistic query evaluation in the general case

What about more general data? We show intractability beyond bounded-treewidth data:

Intractability of probabilistic query evaluation in the general case

What about more general data? We show intractability beyond bounded-treewidth data:

- On any unbounded-treewidth class of probabilistic graphs (conditions apply),

Intractability of probabilistic query evaluation in the general case

What about more general data? We show intractability beyond bounded-treewidth data:

- On any unbounded-treewidth class of probabilistic graphs (conditions apply), for a specific query called the matching query:

Intractability of probabilistic query evaluation in the general case

What about more general data? We show intractability beyond bounded-treewidth data:

- On any unbounded-treewidth class of probabilistic graphs (conditions apply), for a specific query called the matching query:
  - No small d-SDNNFs: we cannot efficiently solve PQE via structured circuits

Intractability of probabilistic query evaluation in the general case

What about more general data? We show **intractability** beyond bounded-treewidth data:

- On **any** unbounded-treewidth class of probabilistic graphs (conditions apply), for a specific query called the **matching query**:
  - No **small d-SDNNFs**: we cannot efficiently solve PQE via structured circuits
  - PQE is **#P-hard under randomized reductions**

Intractability of probabilistic query evaluation in the general case

What about more general data? We show **intractability** beyond bounded-treewidth data:

- On **any** unbounded-treewidth class of probabilistic graphs (conditions apply), for a specific query called the **matching query**:
  - No **small d-SDNNFs**: we cannot efficiently solve PQE via structured circuits
  - PQE is **#P-hard under randomized reductions**
- When allowing **arbitrary instances**:

Intractability of probabilistic query evaluation in the general case

What about more general data? We show **intractability** beyond bounded-treewidth data:

- On **any** unbounded-treewidth class of probabilistic graphs (conditions apply), for a specific query called the **matching query**:
  - No **small d-SDNNFs**: we cannot efficiently solve PQE via structured circuits
  - PQE is **#P-hard under randomized reductions**
- When allowing **arbitrary instances**:
  - We show hardness of PQE for **non-hierarchical self-join free CQs**, in the **uniform case** (where all probabilities are $1/2$)

Intractability of probabilistic query evaluation in the general case

What about more general data? We show **intractability** beyond bounded-treewidth data:

- On **any** unbounded-treewidth class of probabilistic graphs (conditions apply), for a specific query called the **matching query**:
  - No **small d-SDNNFs**: we cannot efficiently solve PQE via structured circuits
  - PQE is **#P-hard under randomized reductions**
- When allowing **arbitrary instances**:
  - We show hardness of PQE for **non-hierarchical self-join free CQs**, in the **uniform case** (where all probabilities are $1/2$)
  - We show the same for all **unbounded homomorphism-closed queries** on graphs

Intractability on unbounded-treewidth data

We consider graphs with probabilistic edges and the matching query Q that asks if there are no two edges that share an endpoint

Intractability on unbounded-treewidth data

We consider graphs with probabilistic edges and the matching query Q that asks if there are no two edges that share an endpoint

Theorem (ICDT'18; with Monet and Senellart)

For some $d \in \mathbb{N}$, any d-SDNNF provenance circuit for Q on a graph G of treewidth k must have size $2^{\Omega(k^{1/d})}$.

Intractability on unbounded-treewidth data


We consider graphs with probabilistic edges and the matching query Q that asks if there are no two edges that share an endpoint

Theorem (ICDT'18; with Monet and Senellart)

For some $d \in \mathbb{N}$, any d-SDNNF provenance circuit for Q on a graph G of treewidth k must have size $2^{\Omega(k^{1/d})}$.

Theorem (MFCS'22; with Monet)

On any graph family $\mathcal{G}$ in which we can efficiently find high-treewidth graphs, the PQE problem for Q on an input graph $G \in \mathcal{G}$ under an input probability distribution is #P-hard under randomized reductions.

Intractability on unbounded-treewidth data


We consider **graphs** with **probabilistic edges** and the **matching query** Q that asks if there are no two edges that share an endpoint

Theorem (ICDT'18; with Monet and Senellart)

For some $\mathbf{d} \in \mathbb{N}$, ***any*** *d-SDNNF provenance circuit for* $\mathbf{Q}$ *on a graph* $\mathbf{G}$ *of treewidth* $\mathbf{k}$ *must have* ***size*** $2^{\Omega(k^{1/d})}$.

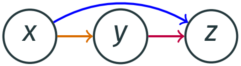
Theorem (MFCS'22; with Monet)

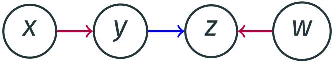
On any graph family $\mathcal{G}$ *in which we can* ***efficiently find high-treewidth graphs****, the PQE problem for* $\mathbf{Q}$ *on an input graph* $\mathbf{G} \in \mathcal{G}$ *under an input probability distribution is* ***#P-hard under randomized reductions****.*

Uses polynomial bounds on the **grid minor theorem** [Chekuri and Chuzhoy, 2016]

Intractability in the uniform setting

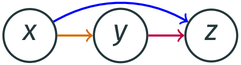
A conjunctive query is self-join-free if all edge colors are different

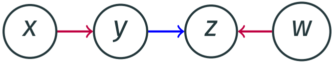
E.g., 

but not 

Intractability in the uniform setting

A conjunctive query is **self-join-free** if all **edge colors** are different

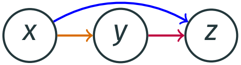
E.g., 

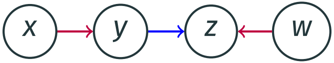
but not 

Known dichotomy: PQE on tuple-independent databases is **intractable** for the **non-hierarchical** such queries [Dalvi and Suciu, 2007]

Intractability in the uniform setting

A conjunctive query is **self-join-free** if all **edge colors** are different

E.g., 

but not 

Known dichotomy: PQE on tuple-independent databases is **intractable** for the **non-hierarchical** such queries [Dalvi and Suciu, 2007]

Theorem (ICDT'21, LMCS; with Kimelfeld)

For any non-hierarchical self-join-free conjunctive query Q, *computing probabilistic query evaluation problem for* Q *input TID databases is #P-hard* ***even if all input probabilities are*** $1/2$.

Intractability for unbounded homomorphism-closed queries

A query Q is homomorphism-closed if whenever G satisfies Q and G has a homomorphism to G' then G' satisfies Q

$\rightarrow$ Examples: CQs, UCQs, Datalog...

Intractability for unbounded homomorphism-closed queries

A query Q is homomorphism-closed if whenever G satisfies Q and G has a homomorphism to G' then G' satisfies Q

$\rightarrow$ Examples: CQs, UCQs, Datalog...

Theorem (ICDT'20, LMCS; with Ceylan)

For any unbounded homomorphism-closed query Q *on graphs, the PQE problem for* Q *is #P-hard.*

Intractability for unbounded homomorphism-closed queries

A query Q is homomorphism-closed if whenever G satisfies Q and G has a homomorphism to G' then G' satisfies Q

$\rightarrow$ Examples: CQs, UCQs, Datalog...

Theorem (ICDT'20, LMCS; with Ceylan)

For any unbounded homomorphism-closed query $\mathbf{Q}$ *on graphs, the PQE problem for* $\mathbf{Q}$ *is #P-hard.*

Theorem (ICDT'23)

This holds even if all probabilities are 1/2.

Conclusion

Summary and perspectives

- Circuits can be a unifying framework for enumeration, incremental maintenance and PQE, at least for MSO queries on bounded-treewidth data
    - Properties of the automata correspond to knowledge compilation circuit classes

Summary and perspectives

- **Circuits** can be a unifying framework for **enumeration**, **incremental maintenance** and **PQE**, at least for MSO queries on bounded-treewidth data
  - **Properties** of the automata correspond to knowledge compilation **circuit classes**
- May also extend to **other settings**:
  - Explored for **enumeration** with **annotated context-free grammars** on words
  - **Open** if circuits explain the tractability of PQE for **safe UCQs**
  - Other cases? (e.g., UCQs with tractable enumeration?)

Summary and perspectives

- Circuits can be a unifying framework for enumeration, incremental maintenance and PQE, at least for MSO queries on bounded-treewidth data
  - Properties of the automata correspond to knowledge compilation circuit classes
- May also extend to other settings:
  - Explored for enumeration with annotated context-free grammars on words
  - Open if circuits explain the tractability of PQE for safe UCQs
  - Other cases? (e.g., UCQs with tractable enumeration?)
- Finer bounds on incremental maintenance via algebraic methods
  - Connections with circuits not (yet) understood
  - Unclear if the results extend to enumeration and to trees

Summary and perspectives

- Circuits can be a unifying framework for enumeration, incremental maintenance and PQE, at least for MSO queries on bounded-treewidth data
  - Properties of the automata correspond to knowledge compilation circuit classes
- May also extend to other settings:
  - Explored for enumeration with annotated context-free grammars on words
  - Open if circuits explain the tractability of PQE for safe UCQs
  - Other cases? (e.g., UCQs with tractable enumeration?)
- Finer bounds on incremental maintenance via algebraic methods
  - Connections with circuits not (yet) understood
  - Unclear if the results extend to enumeration and to trees
- For PQE, hardness holds outside of the bounded-treewidth setting
  - Better joint criteria for width when considering the instance and query?

Summary and perspectives

- **Circuits** can be a unifying framework for **enumeration**, **incremental maintenance** and **PQE**, at least for MSO queries on bounded-treewidth data
  - **Properties** of the automata correspond to knowledge compilation **circuit classes**
- May also extend to **other settings**:
  - Explored for **enumeration** with **annotated context-free grammars** on words
  - **Open** if circuits explain the tractability of PQE for **safe UCQs**
  - Other cases? (e.g., UCQs with tractable enumeration?)
- Finer bounds on incremental maintenance via **algebraic methods**
  - Connections with **circuits** not (yet) understood
  - Unclear if the results extend to **enumeration** and to **trees**
- For PQE, hardness holds outside of the **bounded-treewidth setting**
  - Better **joint criteria** for width when considering the instance and query?
- Enumeration of large solutions by **editing** previous solutions? (STACS'23; with Monet)

Summary and perspectives

- Circuits can be a unifying framework for enumeration, incremental maintenance and PQE, at least for MSO queries on bounded-treewidth data
  - Properties of the automata correspond to knowledge compilation circuit classes
- May also extend to other settings:
  - Explored for enumeration with annotated context-free grammars on words
  - Open if circuits explain the tractability of PQE for safe UCQs
  - Other cases? (e.g., UCQs with tractable enumeration?)
- Finer bounds on incremental maintenance via algebraic methods
  - Connections with circuits not (yet) understood
  - Unclear if the results extend to enumeration and to trees
- For PQE, hardness holds outside of the bounded-treewidth setting
  - Better joint criteria for width when considering the instance and query?
- Enumeration of large solutions by editing previous solutions? (STACS'23; with Monet)

Thanks for your attention!

References i

Bagan, G. (2006).
MSO queries on tree decomposable structures are computable with linear delay.
In *CSL*.

Chekuri, C. and Chuzhoy, J. (2016).
Polynomial bounds for the grid-minor theorem.
JACM, 63(5).

Dalvi, N. and Suciu, D. (2007).
Efficient query evaluation on probabilistic databases.
VLDBJ, 16(4).

References ii

Dalvi, N. and Suciu, D. (2013).
The dichotomy of probabilistic inference for unions of conjunctive queries.
JACM, 59(6).

Kazana, W. and Segoufin, L. (2013).
Enumeration of monadic second-order queries on trees.
TOCL, 14(4).