Provenance

MPRI 2.26.2: Web Data Management

Antoine Amarilli

Provenance Definition

Provenance management

- Data management is **all about query evaluation**
- What if we want **something more** than the query result?
  - **Where** does the result come from?
  - **Why** was this result obtained?
  - **How** was the result produced?
  - What is the **probability** of the result?
  - How many **times** was the result obtained?
  - How would the result **change** if part of the input data was missing?
  - What is the minimal **security clearance** I need to see the result?
  - What is the **most economical way** of obtaining the result?
  - How can a result be **explained** to the user?
- **Provenance management:** along with query evaluation, record **additional bookkeeping information** to answer the questions above

Provenance data model

- Relational data model: data decomposed into relations, with labeled attributes…

Provenance data model

- **Relational data model**: data decomposed into relations, with labeled attributes…

name	position	city	classification
John	Director	New York	unclassified
Paul	Janitor	New York	restricted
Dave	Analyst	Paris	confidential
Ellen	Field agent	Berlin	secret
Magdalen	Double agent	Paris	top secret
Nancy	HR director	Paris	restricted
Susan	Analyst	Berlin	secret

Provenance data model

- **Relational data model**: data decomposed into relations, with labeled attributes…
- … with an extra **provenance annotation** for each tuple (think of it first as a tuple id)

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

Outline

Provenance Definition

Preliminaries

Boolean Provenance

Provenance for Probability Computation

Applications to Enumeration

Semiring Provenance

Implementing Provenance Support

Boolean valuations

- Database D with n tuples
- $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ the **Boolean variables** annotating the tuples
- **Valuation** over $\mathcal{X}$: function $\nu : \mathcal{X} \to \{\bot, \top\}$
- **Possible world** $\nu(D)$: the subset of D where we keep precisely the tuples whose annotation evaluates to $\top$

Example of possible worlds

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

$$\nu: \quad \begin{array}{ccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ \top & \top & \top & \top & \top & \top & \top \end{array}$$

Example of possible worlds

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Dave	Analyst	Paris	confidential	x_3
Magdalen	Double agent	Paris	top secret	x_5
Susan	Analyst	Berlin	secret	x_7

$$\nu: \begin{array}{ccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \\ \top & \bot & \top & \bot & \top & \bot & \top \end{array}$$

Boolean provenance of query results

- **Goal:** Evaluate a positive relational algebra query (UCQ) Q on a database D...

Boolean provenance of query results

- **Goal:** Evaluate a **positive relational algebra query** (UCQ) Q on a database D... whose tuples are annotated with $\mathcal{X} = x_1, \ldots, x_n$

Boolean provenance of query results

- Goal: Evaluate a positive relational algebra query (UCQ) Q on a database D... whose tuples are annotated with $\mathcal{X} = x_1, \ldots, x_n$
- The result is a relation instance R...

Boolean provenance of query results

- **Goal:** Evaluate a **positive relational algebra query** (UCQ) Q on a database D… whose tuples are annotated with $\mathcal{X} = x_1, \ldots, x_n$
- The result is a **relation instance** R… where each tuple is annotated with a **Boolean function** on $\mathcal{X}$

Boolean provenance of query results

- **Goal:** Evaluate a **positive relational algebra query** (UCQ) Q on a database D... whose tuples are annotated with $\mathcal{X} = x_1, \ldots, x_n$
- The result is a **relation instance** R... where each tuple is annotated with a **Boolean function** on $\mathcal{X}$
- **Semantics:** For every tuple t of the result, for every valuation ν of $\mathcal{X}$, the annotation of t evaluates to true on ν iff $t \in Q(\nu(D))$

Boolean provenance of query results

- **Goal:** Evaluate a **positive relational algebra query** (UCQ) Q on a database D... whose tuples are annotated with $\mathcal{X} = x_1, \ldots, x_n$
- The result is a **relation instance** R... where each tuple is annotated with a **Boolean function** on $\mathcal{X}$
- **Semantics:** For every tuple t of the result, for every valuation ν of $\mathcal{X}$, the annotation of t evaluates to true on ν iff $t \in Q(\nu(D))$

Example (What cities are in the table?)

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

city	prov
New York	$x_1 \lor x_2$
Paris	$x_3 \lor x_5 \lor x_6$
Berlin	$x_4 \lor x_7$

Boolean provenance of query results

- **Goal:** Evaluate a **positive relational algebra query** (UCQ) Q on a database D... whose tuples are annotated with $\mathcal{X} = x_1, \ldots, x_n$
- The result is a **relation instance** R... where each tuple is annotated with a **Boolean function** on $\mathcal{X}$
- **Semantics:** For every tuple t of the result, for every valuation ν of $\mathcal{X}$, the annotation of t evaluates to true on ν iff $t \in Q(\nu(D))$

Example (What cities are in the table?)

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

city	prov
New York	$x_1 \vee x_2$
Paris	$x_3 \vee x_5 \vee x_6$
Berlin	$x_4 \vee x_7$

Claim: we can compute this while evaluating the query!

Selection, renaming

Provenance annotations of selected tuples are unchanged

Example ($\rho_{\mathbf{name}\to\mathbf{n}}(\sigma_{\mathbf{city}=\text{“New York”}}(R))$)

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

n	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2

Projection

Take the OR of provenance annotations of identical, merged tuples

Example ($\pi_{\mathbf{city}}(R)$)

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

city	prov
New York	$x_1 \lor x_2$
Paris	$x_3 \lor x_5 \lor x_6$
Berlin	$x_4 \lor x_7$

Union

Take the OR of provenance annotations of identical, merged tuples

Example

$\pi_{\text{city}}(\sigma_{\textit{ends-with}(\text{position},\text{“agent”})}(R)) \cup \pi_{\text{city}}(\sigma_{\text{position}=\text{“Analyst”}}(R))$

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

city	prov
Paris	$x_3 \vee x_5$
Berlin	$x_4 \vee x_7$

Cross product

Take the AND of provenance annotations of combined tuples

Example

$\pi_{\text{city}}(\sigma_{\textit{ends-with}(\text{position},\text{“agent”})}(R)) \bowtie \pi_{\text{city}}(\sigma_{\text{position}=\text{“Analyst”}}(R))$

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

city	prov
Paris	$x_3 \wedge x_5$
Berlin	$x_4 \wedge x_7$

How is provenance actually represented?

Provenance annotations are Boolean functions

- The simplest representation is Boolean formulas
- Formalism used in most of the provenance literature

Example

Is there a city with two different agents?

$$(x_1 \land x_2) \lor (x_3 \land x_6) \lor (x_3 \land x_5) \lor (x_4 \land x_7) \lor (x_5 \land x_6)$$

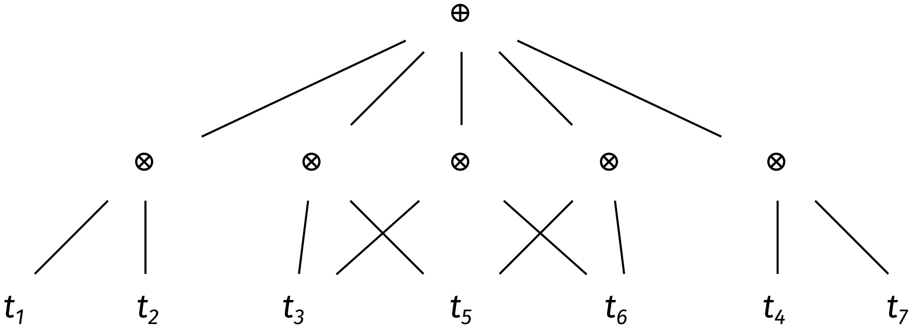
Theorem (PTIME overhead)

For any fixed positive relational algebra expression, given an input database, we can compute in PTIME the provenance annotation of every tuple in the result

Other representation: Provenance circuits
[Deutch et al., 2014]

- Use **Boolean circuits** to represent provenance
- Every time an operation reuses a previously computed result, link to the **previously created circuit gate**
- **Never larger** than provenance formulas
- Sometimes **more concise**

Example provenance circuit



What can we do with Boolean provenance?

$$(x_1 \wedge x_2) \vee (x_3 \wedge x_6) \vee (x_3 \wedge x_5) \vee (x_4 \wedge x_7) \vee (x_5 \wedge x_6)$$

- The provenance describes, for each result tuple, the subsets of the input database for which it appears in the query result

What can we do with Boolean provenance?

$$(x_1 \wedge x_2) \vee (x_3 \wedge x_6) \vee (x_3 \wedge x_5) \vee (x_4 \wedge x_7) \vee (x_5 \wedge x_6)$$

- The provenance describes, for each result tuple, the subsets of the input database for which it appears in the query result
- SAT: test if the tuple can be an answer when we delete some input tuples (trivial for monotone queries)

What can we do with Boolean provenance?

$$(x_1 \wedge x_2) \vee (x_3 \wedge x_6) \vee (x_3 \wedge x_5) \vee (x_4 \wedge x_7) \vee (x_5 \wedge x_6)$$

- The provenance describes, for each result tuple, the subsets of the input database for which it appears in the query result
- SAT: test if the tuple can be an answer when we delete some input tuples (trivial for monotone queries)
- #SAT: number of sub-databases where the tuple is a result
  - $\rightarrow$ Useful for probabilistic query evaluation

What can we do with Boolean provenance?

$$(x_1 \wedge x_2) \vee (x_3 \wedge x_6) \vee (x_3 \wedge x_5) \vee (x_4 \wedge x_7) \vee (x_5 \wedge x_6)$$

- The provenance describes, for each result tuple, the **subsets** of the input database for which it appears in the query result
- **SAT**: test if the tuple can be an answer when we delete some input tuples (trivial for monotone queries)
- **#SAT**: number of sub-databases where the tuple is a result
  - → Useful for **probabilistic query evaluation**
- **Enumerating models:** enumerating sub-databases where the tuple is a result
  - → Useful to **enumerate query results** (see later)

Outline

Reminder: TIDs

- **Tuple-independent database D**: each tuple t in D is annotated with **independent** probability $\Pr(t)$ of existing

name	position	city	classification	prob
John	Director	New York	unclassified	0.5
Paul	Janitor	New York	restricted	0.7
Dave	Analyst	Paris	confidential	0.3
Ellen	Field agent	Berlin	secret	0.2
Magdalen	Double agent	Paris	top secret	1.0
Nancy	HR director	Paris	restricted	0.8
Susan	Analyst	Berlin	secret	0.2

$\rightarrow$ Probability of a possible world $D' \subseteq D$:

$$\Pr(D') = \prod_{t \in D'} \Pr(t) \times \prod_{t \in D' \setminus D} (1 - \Pr(t'))$$

PQE via provenance

name	position	city	classification	prov	prob
John	Director	New York	unclassified	x_1	0.5
Paul	Janitor	New York	restricted	x_2	0.7
Dave	Analyst	Paris	confidential	x_3	0.3
Ellen	Field agent	Berlin	secret	x_4	0.2
Magdalen	Double agent	Paris	top secret	x_5	1.0
Nancy	HR director	Paris	restricted	x_6	0.8
Susan	Analyst	Berlin	secret	x_7	0.2

city	prov	prob
New York	$x_1 \vee x_2$	$1 - (1 - 0.5) \times (1 - 0.7) = 0.85$
Paris	$x_3 \vee x_5 \vee x_6$	1.00
Berlin	$x_4 \vee x_7$	$1 - (1 - 0.2) \times (1 - 0.2) = 0.36$

Extensional PQE vs intensional PQE

- Recall that PQE for UCQs is:
    - PTIME in some cases
    - #P-hard in general
    - There is a dichotomy separating tractable and intractable cases

Extensional PQE vs intensional PQE

- Recall that PQE for UCQs is:
  - PTIME in some cases
  - #P-hard in general
  - There is a dichotomy separating tractable and intractable cases

- Extensional PQE: computing the probability by evaluating the query "following the relational algebra operators"
  - This covers the tractable cases of PQE for select-project-join queries (CQs) without self-joins with an easy algorithm
  - This covers all tractable cases (for UCQs) with a far more complicated algorithm

Extensional PQE vs intensional PQE

- Recall that PQE for UCQs is:
    - PTIME in some cases
    - #P-hard in general
    - There is a dichotomy separating tractable and intractable cases
- Extensional PQE: computing the probability by evaluating the query "following the relational algebra operators"
    - This covers the tractable cases of PQE for select-project-join queries (CQs) without self-joins with an easy algorithm
    - This covers all tractable cases (for UCQs) with a far more complicated algorithm
- Intensional PQE: compute the provenance of the query as a Boolean circuit (or formula) and compute the probability of the provenance

Outline

Provenance Definition

Provenance for Probability Computation

Applications to Enumeration

Semiring Provenance

Implementing Provenance Support

Enumerating query results

Idea: Often, we do not need to compute all results of a query
we just need to be able to enumerate results quickly

Enumerating query results

Idea: Often, we do not need to compute all results of a query
we just need to be able to enumerate results quickly

Enumerating query results

Idea: Often, we do not need to compute **all results** of a query
we just need to be able to **enumerate** results quickly

Enumerating query results

Idea: Often, we do not need to compute **all results** of a query
we just need to be able to **enumerate** results quickly

how to find patterns | **Search**

Results **1 - 20** of **10,514**

...

Enumerating query results

Idea: Often, we do not need to compute **all results** of a query
we just need to be able to **enumerate** results quickly

Enumerating query results

Idea: Often, we do not need to compute **all results** of a query
we just need to be able to **enumerate** results quickly

→ Formalization: **enumeration algorithms**

→ Currently a pretty important topic in database theory

Enumeration algorithm (linear preprocessing, constant delay)

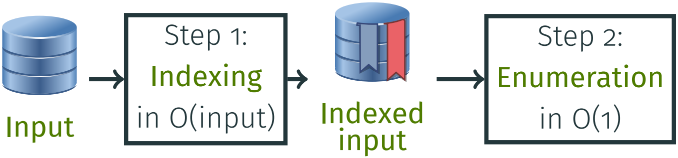
Input

Enumeration algorithm (linear preprocessing, constant delay)

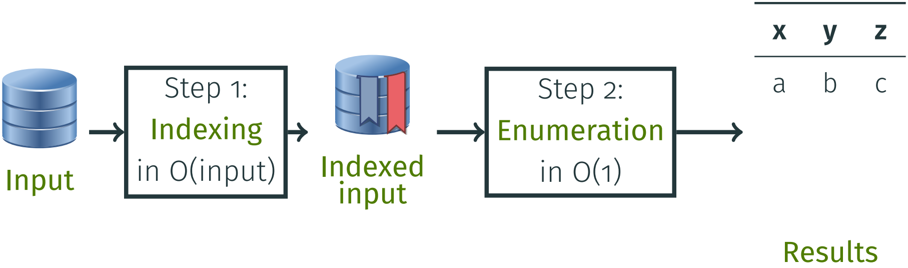
Input → Step 1: Indexing in O(input)

Enumeration algorithm (linear preprocessing, constant delay)

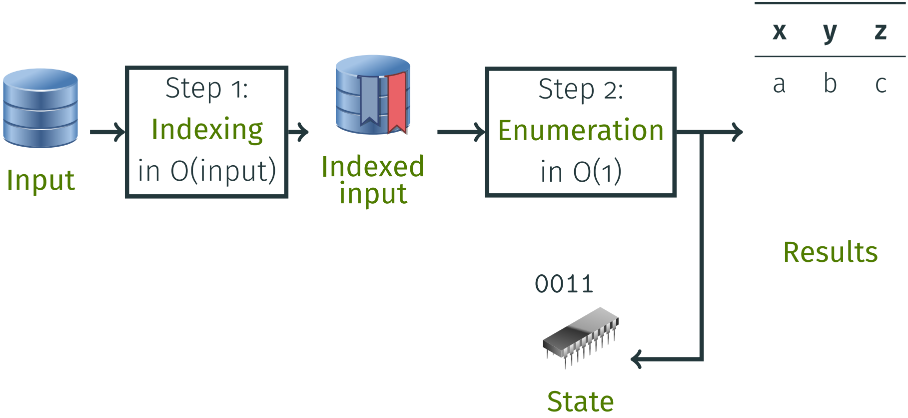
Enumeration algorithm (linear preprocessing, constant delay)



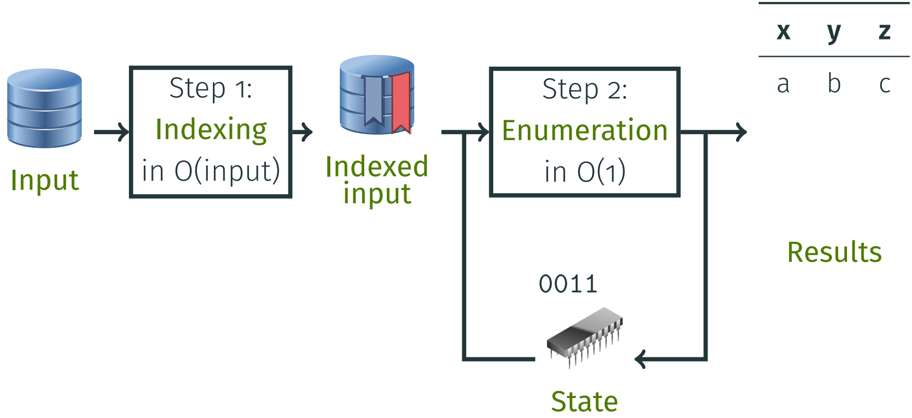
Enumeration algorithm (linear preprocessing, constant delay)



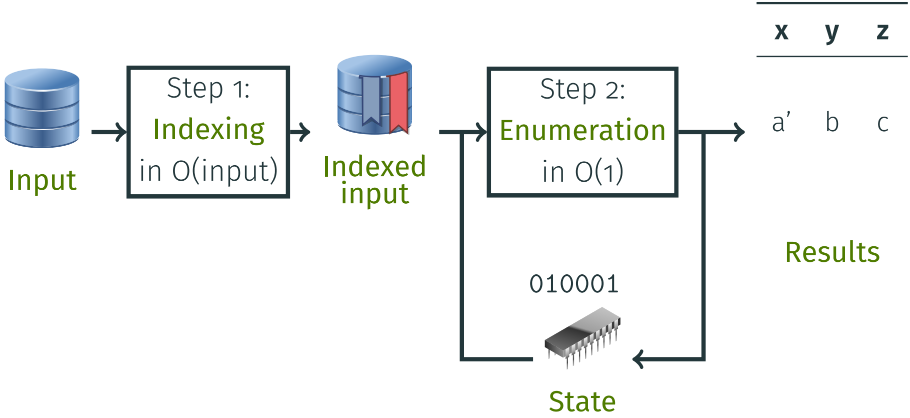
Enumeration algorithm (linear preprocessing, constant delay)



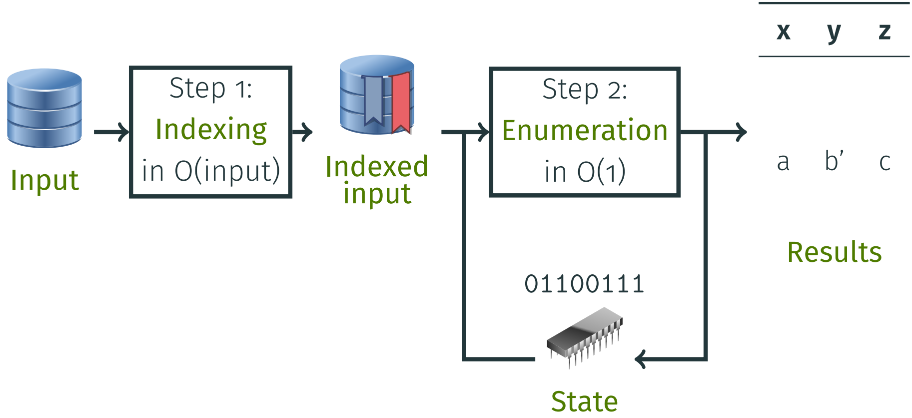
Enumeration algorithm (linear preprocessing, constant delay)



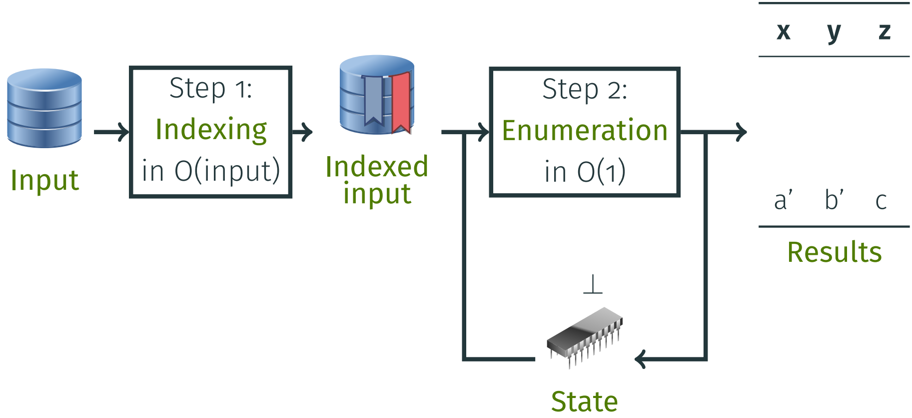
Enumeration algorithm (linear preprocessing, constant delay)



Enumeration algorithm (linear preprocessing, constant delay)



Enumeration algorithm (linear preprocessing, constant delay)



Connection to provenance

Provenance can also represent query answers!

Connection to provenance

Provenance can also represent **query answers**!

- Study answers of **non-Boolean query** $Q(x, y)$ on database D

Connection to provenance

Provenance can also represent query answers!

- Study answers of non-Boolean query $Q(x, y)$ on database D

$$Q(x, y) : \exists z \; R(x, y) \wedge S(y, z)$$

$$D : R(a, b), R(a', b), S(b, c)$$

Connection to provenance

Provenance can also represent **query answers**!

- Study answers of **non-Boolean query** $Q(x, y)$ on database D

  $Q(x, y) : \exists z\ R(x, y) \land S(y, z)$

  $D : R(a, b), R(a', b), S(b, c)$

- Add **assignment facts** $X(v), Y(v)$ to D for each element v (linear)

Connection to provenance

Provenance can also represent **query answers**!

- Study answers of **non-Boolean query** $Q(x, y)$ on database D

  $Q(x, y) : \exists z \; R(x, y) \wedge S(y, z)$

  $D : R(a, b), R(a', b), S(b, c)$

- Add **assignment facts** $X(v), Y(v)$ to D for each element v (linear)

  $X(a), X(a'), X(b), X(c)$

  $Y(a), Y(a'), Y(b), Y(c)$

Connection to provenance

Provenance can also represent **query answers**!

- Study answers of **non-Boolean query** $Q(x,y)$ on database D — $Q(x,y) : \exists z\ R(x,y) \wedge S(y,z)$; $D : R(a,b), R(a',b), S(b,c)$
- Add **assignment facts** $X(v), Y(v)$ to D for each element v (linear) — $X(a), X(a'), X(b), X(c)$; $Y(a), Y(a'), Y(b), Y(c)$
- Consider the **Boolean query** $Q' : X(x) \wedge Y(y) \wedge Q(x,y)$

Connection to provenance

Provenance can also represent **query answers**!

- Study answers of **non-Boolean query** $Q(x, y)$ on database D

  $Q(x, y) : \exists z\ R(x, y) \land S(y, z)$

  $D : R(a, b), R(a', b), S(b, c)$

- Add **assignment facts** $X(v), Y(v)$ to D for each element v (linear)

  $X(a), X(a'), X(b), X(c)$

  $Y(a), Y(a'), Y(b), Y(c)$

- Consider the **Boolean query** $Q' : X(x) \land Y(y) \land Q(x, y)$

  $X(x) \land Y(y) \land (\exists z\ R(x, y) \land S(y, z))$

Connection to provenance

Provenance can also represent **query answers**!

- Study answers of **non-Boolean query** $Q(x, y)$ on database D

  $Q(x, y) : \exists z \; R(x, y) \wedge S(y, z)$

  $D : R(a, b), R(a', b), S(b, c)$

- Add **assignment facts** $X(v), Y(v)$ to D for each element v (linear)

  $X(a), X(a'), X(b), X(c)$

  $Y(a), Y(a'), Y(b), Y(c)$

- Consider the **Boolean query** $Q' : X(x) \wedge Y(y) \wedge Q(x, y)$

  $X(x) \wedge Y(y) \wedge (\exists z \; R(x, y) \wedge S(y, z))$

- Compute the **provenance** C' of Q' on D plus assignment facts

Connection to provenance

Provenance can also represent **query answers**!

- Study answers of **non-Boolean query** $Q(x, y)$ on database D — $Q(x, y) : \exists z\ R(x, y) \wedge S(y, z)$; $D : R(a, b), R(a', b), S(b, c)$
- Add **assignment facts** $X(v), Y(v)$ to D for each element v (linear) — $X(a), X(a'), X(b), X(c)$; $Y(a), Y(a'), Y(b), Y(c)$
- Consider the **Boolean query** $Q' : X(x) \wedge Y(y) \wedge Q(x, y)$ — $X(x) \wedge Y(y) \wedge (\exists z\ R(x, y) \wedge S(y, z))$
- Compute the **provenance** C' of Q' on D plus assignment facts — $(X(a) \wedge R(a, b) \vee X(a') \wedge R(a', b)) \wedge Y(b) \wedge S(b, c)$

Connection to provenance

Provenance can also represent **query answers**!

- Study answers of **non-Boolean query** $Q(x, y)$ on database D — $Q(x, y) : \exists z\ R(x, y) \wedge S(y, z)$, $D : R(a, b), R(a', b), S(b, c)$
- Add **assignment facts** $X(v), Y(v)$ to D for each element v (linear) — $X(a), X(a'), X(b), X(c)$, $Y(a), Y(a'), Y(b), Y(c)$
- Consider the **Boolean query** $Q' : X(x) \wedge Y(y) \wedge Q(x, y)$ — $X(x) \wedge Y(y) \wedge (\exists z\ R(x, y) \wedge S(y, z))$
- Compute the **provenance** C' of Q' on D plus assignment facts — $(X(a) \wedge R(a, b) \vee X(a') \wedge R(a', b))$ $\wedge Y(b) \wedge S(b, c)$
- Define C by replacing all variables by 1 except assignment facts

Connection to provenance

Provenance can also represent **query answers**!

- Study answers of **non-Boolean query** $Q(x, y)$ on database D — $Q(x, y) : \exists z\; R(x, y) \wedge S(y, z)$; $D : R(a, b), R(a', b), S(b, c)$
- Add **assignment facts** $X(v), Y(v)$ to D for each element v (linear) — $X(a), X(a'), X(b), X(c)$; $Y(a), Y(a'), Y(b), Y(c)$
- Consider the **Boolean query** $Q' : X(x) \wedge Y(y) \wedge Q(x, y)$ — $X(x) \wedge Y(y) \wedge (\exists z\; R(x, y) \wedge S(y, z))$
- Compute the **provenance** C' of Q' on D plus assignment facts — $(X(a) \wedge R(a, b) \vee X(a') \wedge R(a', b)) \wedge Y(b) \wedge S(b, c)$
- Define C by replacing all variables by 1 except assignment facts — $(X(a) \vee X(a')) \wedge Y(b)$

Connection to provenance

Provenance can also represent **query answers**!

- Study answers of **non-Boolean query** $Q(x, y)$ on database D — $Q(x, y) : \exists z\ R(x, y) \land S(y, z)$; $D : R(a, b), R(a', b), S(b, c)$
- Add **assignment facts** $X(v), Y(v)$ to D for each element v (linear) — $X(a), X(a'), X(b), X(c)$; $Y(a), Y(a'), Y(b), Y(c)$
- Consider the **Boolean query** $Q' : X(x) \land Y(y) \land Q(x, y)$ — $X(x) \land Y(y) \land (\exists z\ R(x, y) \land S(y, z))$
- Compute the **provenance** C' of Q' on D plus assignment facts — $(X(a) \land R(a, b) \lor X(a') \land R(a', b)) \land Y(b) \land S(b, c)$
- Define C by replacing all variables by 1 except assignment facts — $(X(a) \lor X(a')) \land Y(b)$

$\rightarrow$ The circuit C represents the **query answers**

Connection to provenance

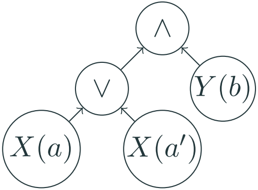
Provenance can also represent **query answers**!

- Study answers of **non-Boolean query** $Q(x, y)$ on database D — $Q(x,y) : \exists z\ R(x,y) \wedge S(y,z)$; $D : R(a,b), R(a',b), S(b,c)$
- Add **assignment facts** $X(v), Y(v)$ to D for each element v (linear) — $X(a), X(a'), X(b), X(c)$; $Y(a), Y(a'), Y(b), Y(c)$
- Consider the **Boolean query** $Q' : X(x) \wedge Y(y) \wedge Q(x,y)$ — $X(x) \wedge Y(y) \wedge (\exists z\ R(x,y) \wedge S(y,z))$
- Compute the **provenance** C' of Q' on D plus assignment facts — $(X(a) \wedge R(a,b) \vee X(a') \wedge R(a',b)) \wedge Y(b) \wedge S(b,c)$
- Define C by replacing all variables by 1 except assignment facts — $(X(a) \vee X(a')) \wedge Y(b)$

$\rightarrow$ The circuit C represents the **query answers** (a,b) and (a',b)

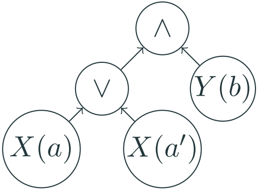
Enumeration via provenance

- We have a provenance circuit representing the query answers



Enumeration via provenance

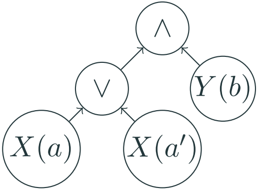
- We have a provenance circuit representing the query answers



- So to enumerate query answers we can:
  - Compute this provenance circuit
  - Enumerate its satisfying assignments

Enumeration via provenance

- We have a **provenance circuit** representing the query answers



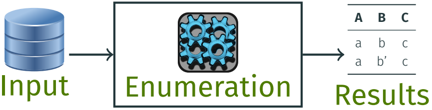
- So to **enumerate query answers** we can:
  - **Compute** this provenance circuit
  - **Enumerate** its satisfying assignments

$\rightarrow$ We want **linear preprocessing** and **constant delay**
so we designed an enumeration algorithm for circuits:

Theorem ([Amarilli et al., 2017])

Given a d-SDNNF circuit, we can preprocess it in linear time and then enumerate its satisfying assignments with constant delay (if the assignments have constant size)

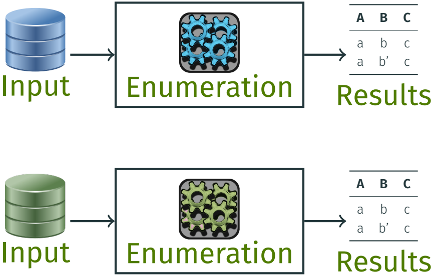
Currently:



A	B	C
a	b	c
a	b’	c

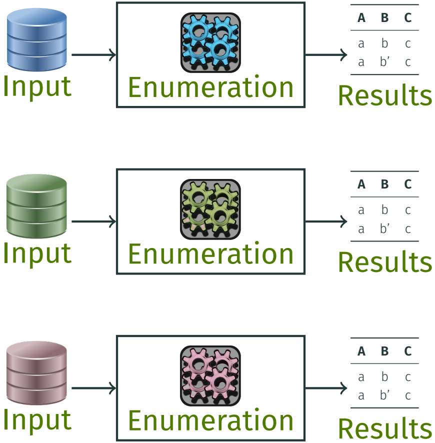
Enumeration via provenance: motivation

Currently:



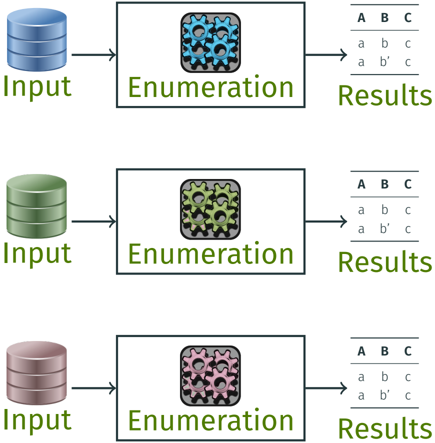
Enumeration via provenance: motivation

Currently:



Enumeration via provenance: motivation

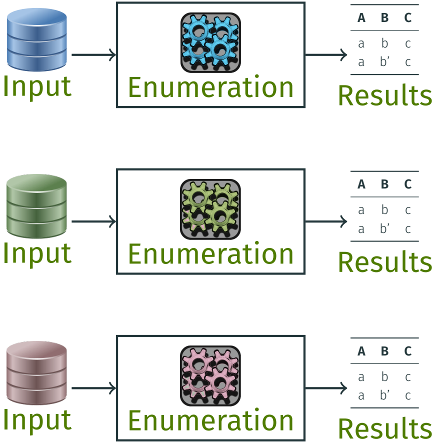
Currently:



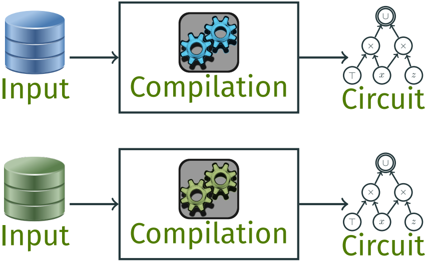
Our idea:

Enumeration via provenance: motivation

Currently:

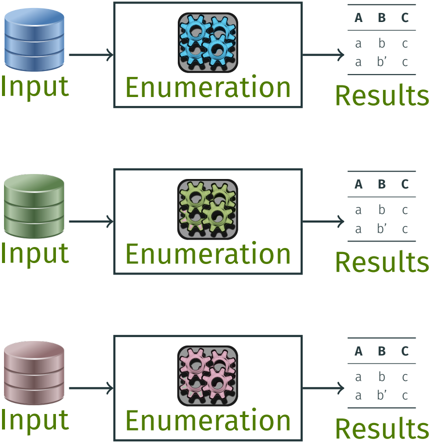


Our idea:

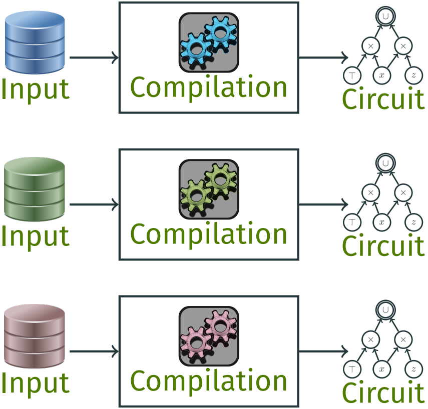


Enumeration via provenance: motivation

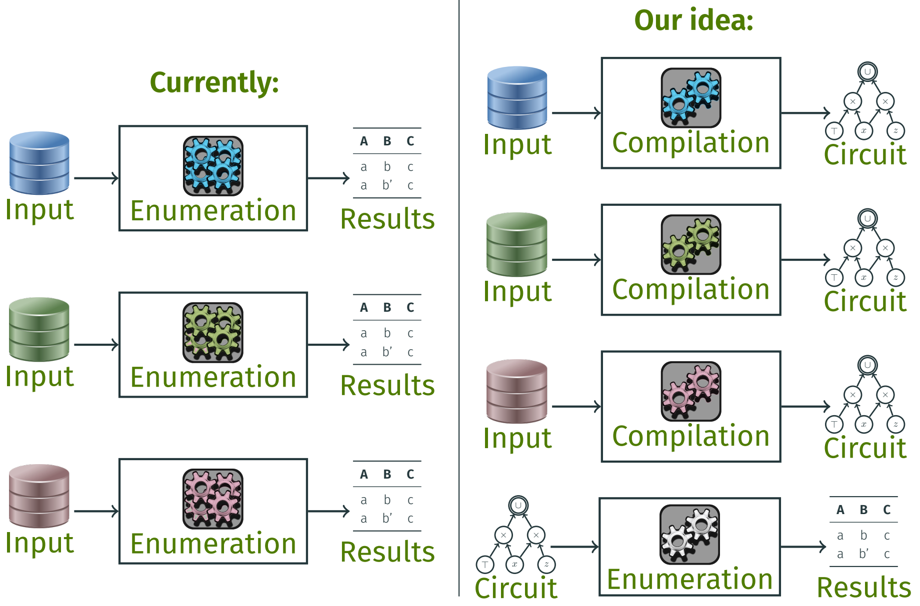
Currently:



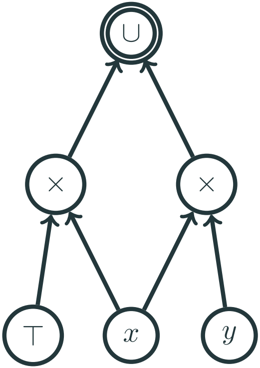
Our idea:



Enumeration via provenance: motivation

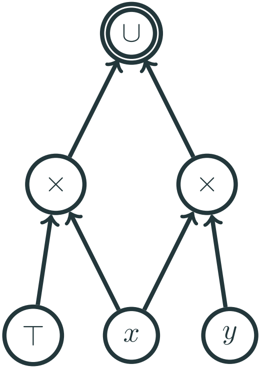


Set circuits



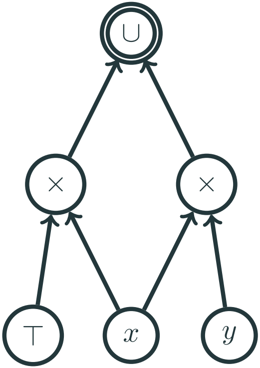
- Directed acyclic graph of **gates**

Set circuits



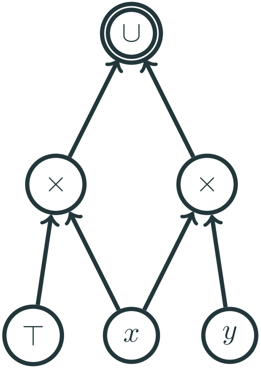
- Directed acyclic graph of **gates**
- **Output** gate: 

Set circuits



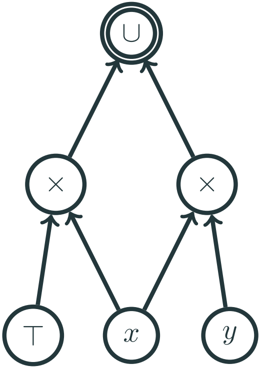
- Directed acyclic graph of **gates**
- **Output** gate: 
- **Variable** gates: 

Set circuits



- Directed acyclic graph of **gates**
- **Output** gate: 
- **Variable** gates: x
- **Constant** gates: $\top$ $\bot$

Set circuits

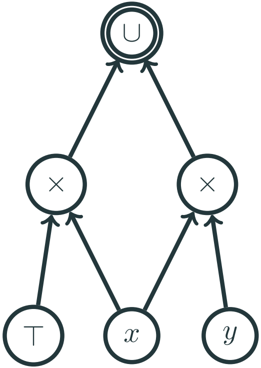


- Directed acyclic graph of **gates**
- **Output** gate: 
- **Variable** gates: 

- **Constant** gates: 

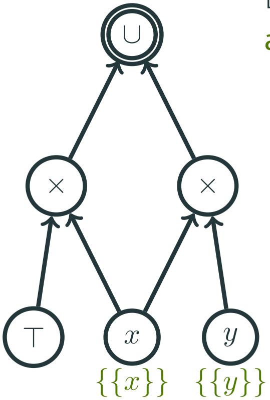
- **Internal** gates: 

Semantics of set circuits



Every gate g **captures** a set $S(g)$ of sets (called **assignments**)

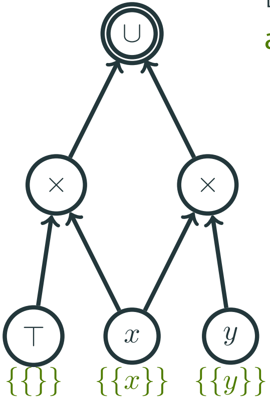
Semantics of set circuits



Every gate g **captures** a set $S(g)$ of sets (called **assignments**)

- **Variable** gate with label x: $S(g) := \{\{x\}\}$

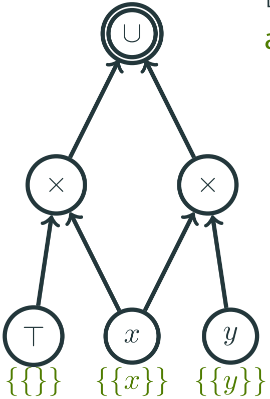
Semantics of set circuits



Every gate g **captures** a set $S(g)$ of sets (called **assignments**)

- **Variable** gate with label x: $S(g) := \{\{x\}\}$
- $\top$-gates: $S(g) = \{\{\}\}$

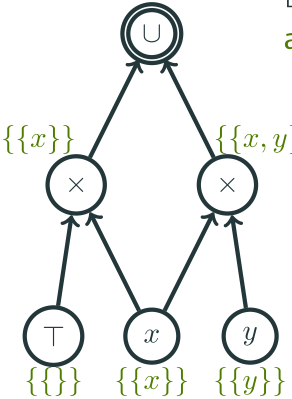
Semantics of set circuits



Every gate g **captures** a set $S(g)$ of sets (called **assignments**)

- **Variable** gate with label x: $S(g) := \{\{x\}\}$
- $\top$-gates: $S(g) = \{\{\}\}$
- $\bot$-gates: $S(g) = \emptyset$

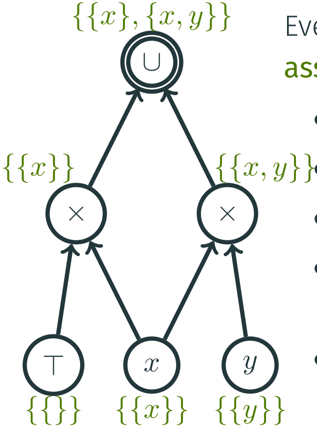
Semantics of set circuits



Every gate g **captures** a set $S(g)$ of sets (called **assignments**)

- **Variable** gate with label x: $S(g) := \{\{x\}\}$
- $\top$-gates: $S(g) = \{\{\}\}$
- $\bot$-gates: $S(g) = \emptyset$
- $\times$-gate with children g_1, g_2:
  $S(g) := \{s_1 \cup s_2 \mid s_1 \in S(g_1), s_2 \in S(g_2)\}$

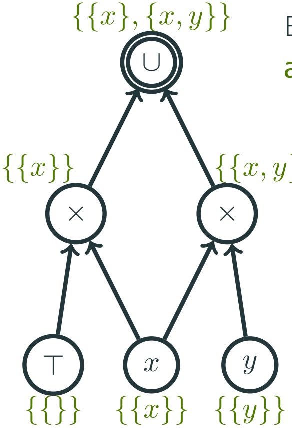
Semantics of set circuits



Every gate g **captures** a set $S(g)$ of sets (called **assignments**)

- **Variable** gate with label x: $S(g) := \{\{x\}\}$
- $\top$-gates: $S(g) = \{\{\}\}$
- $\bot$-gates: $S(g) = \emptyset$
- $\times$-gate with children g_1, g_2:
  $S(g) := \{s_1 \cup s_2 \mid s_1 \in S(g_1), s_2 \in S(g_2)\}$
- $\cup$-gate with children g_1, g_2:
  $S(g) := S(g_1) \cup S(g_2)$

Semantics of set circuits



Every gate g **captures** a set $S(g)$ of sets (called **assignments**)

- **Variable** gate with label x: $S(g) := \{\{x\}\}$
- $\top$-gates: $S(g) = \{\{\}\}$
- $\bot$-gates: $S(g) = \emptyset$
- $\times$-gate with children g_1, g_2:
  $S(g) := \{s_1 \cup s_2 \mid s_1 \in S(g_1), s_2 \in S(g_2)\}$
- $\cup$-gate with children g_1, g_2:
  $S(g) := S(g_1) \cup S(g_2)$

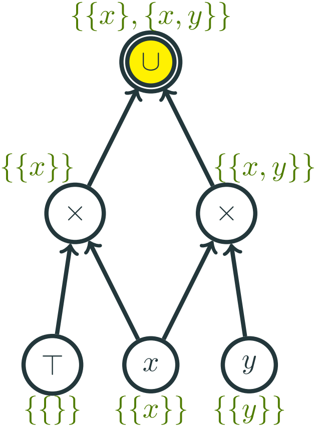
Task: Enumerate the assignments of the set $S(g)$ captured by a gate g
$\rightarrow$E.g., for $S(g) = \{\{x\},\{x,y\}\}$, enumerate $\{x\}$ and then $\{x,y\}$

Circuit restrictions

d-DNNF set circuit:

- $\cup$ are all **deterministic**:

The inputs are **disjoint**
(= no assignment is captured by two inputs)



Circuit restrictions

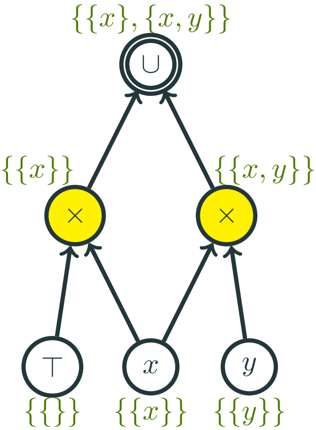
d-DNNF set circuit:

- ∪ are all **deterministic**:

The inputs are **disjoint**
(= no assignment is captured by two inputs)

- × are all **decomposable**:

The inputs are **independent**
(= no variable x has a path to two different inputs)



Main results

Theorem

Given a d-DNNF set circuit C, we can enumerate its captured assignments with preprocessing linear in $|C|$ and delay linear in each assignment

Main results

Theorem

Given a d-DNNF set circuit C, we can enumerate its captured assignments with preprocessing linear in $|C|$ and delay linear in each assignment

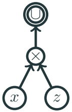
Also: restrict to assignments of **constant size** $k \in \mathbb{N}$

Theorem

Given a d-DNNF set circuit C, we can enumerate its captured assignments of size $\leq k$
with preprocessing linear in $|C|$ and constant delay

Proof overview

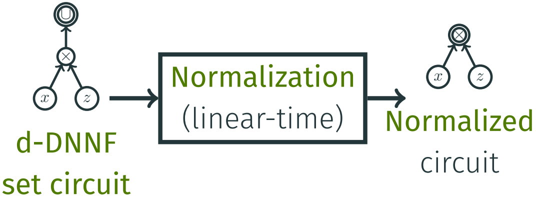
Preprocessing phase:



d-DNNF
set circuit

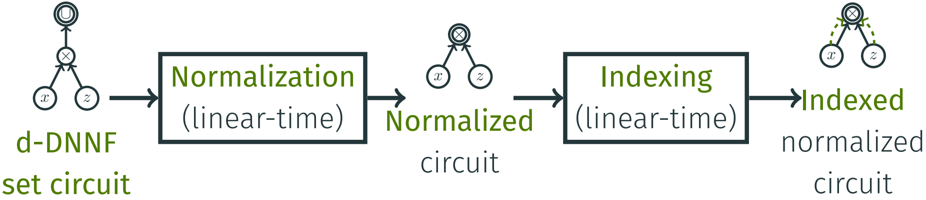
Proof overview

Preprocessing phase:



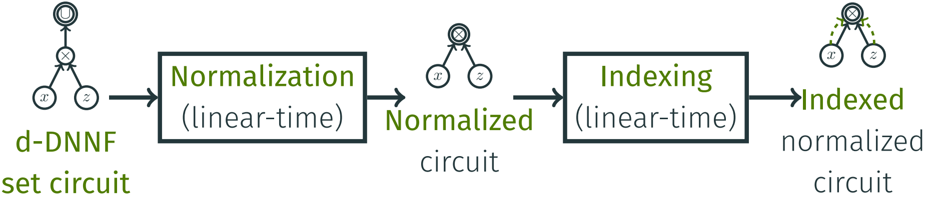
Proof overview

Preprocessing phase:

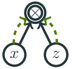


Proof overview

Preprocessing phase:



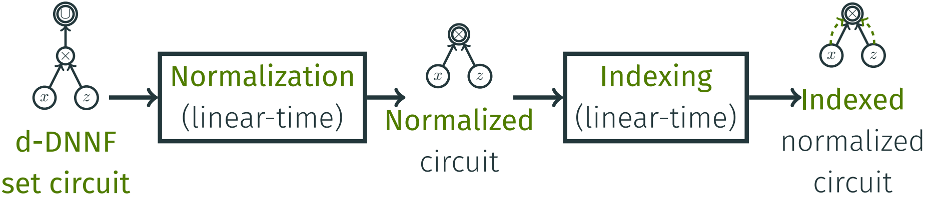
Enumeration phase:



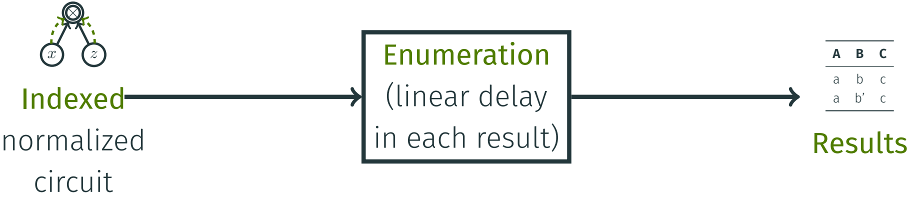
Indexed
normalized
circuit

Proof overview

Preprocessing phase:



Enumeration phase:



Enumerating captured assignments of d-DNNF set circuits

Task: Enumerate the assignments of the set $S(g)$ captured by a gate g

$\rightarrow$ E.g., for $S(g) = \{\{x\}, \{x, y\}\}$, enumerate $\{x\}$ and then $\{x, y\}$

Enumerating captured assignments of d-DNNF set circuits

Task: Enumerate the assignments of the set $S(g)$ captured by a gate g

$\rightarrow$ E.g., for $S(g) = \{\{x\}, \{x, y\}\}$, enumerate $\{x\}$ and then $\{x, y\}$

Base case: variable x :

Enumerating captured assignments of d-DNNF set circuits

Task: Enumerate the assignments of the set $S(g)$ captured by a gate g

$\rightarrow$ E.g., for $S(g) = \{\{x\}, \{x, y\}\}$, enumerate $\{x\}$ and then $\{x, y\}$

Base case: variable 

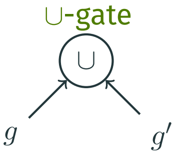
: enumerate $\{x\}$ and stop

Enumerating captured assignments of d-DNNF set circuits

Task: Enumerate the assignments of the set $S(g)$ captured by a gate g

$\rightarrow$ E.g., for $S(g) = \{\{x\}, \{x, y\}\}$, enumerate $\{x\}$ and then $\{x, y\}$

Base case: variable x : enumerate $\{x\}$ and stop



Concatenation: enumerate $S(g)$
and then enumerate $S(g')$

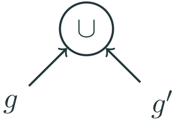
Enumerating captured assignments of d-DNNF set circuits

Task: Enumerate the assignments of the set $S(g)$ captured by a gate g

$\rightarrow$ E.g., for $S(g) = \{\{x\}, \{x, y\}\}$, enumerate $\{x\}$ and then $\{x, y\}$

Base case: variable x : enumerate $\{x\}$ and stop

$\cup$-gate



Concatenation: enumerate $S(g)$ and then enumerate $S(g')$

Determinism: no duplicates

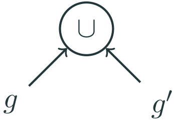
Enumerating captured assignments of d-DNNF set circuits

Task: Enumerate the assignments of the set $S(g)$ captured by a gate g

$\rightarrow$ E.g., for $S(g) = \{\{x\}, \{x, y\}\}$, enumerate $\{x\}$ and then $\{x, y\}$

Base case: variable x : enumerate $\{x\}$ and stop

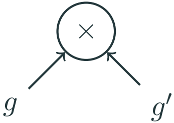
$\cup$-gate



Concatenation: enumerate $S(g)$ and then enumerate $S(g')$

Determinism: no duplicates

$\times$-gate



Lexicographic product: enumerate $S(g)$ and for each result t enumerate $S(g')$ and concatenate t with each result

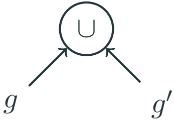
Enumerating captured assignments of d-DNNF set circuits

Task: Enumerate the assignments of the set $S(g)$ captured by a gate g

$\rightarrow$ E.g., for $S(g) = \{\{x\}, \{x, y\}\}$, enumerate $\{x\}$ and then $\{x, y\}$

Base case: variable x : enumerate $\{x\}$ and stop

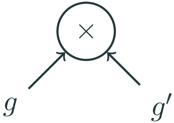
∪-gate



Concatenation: enumerate $S(g)$ and then enumerate $S(g')$

Determinism: no duplicates

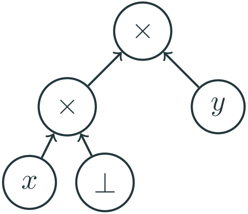
×-gate



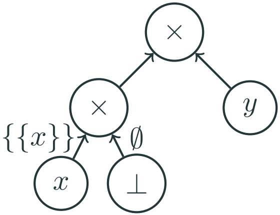
Lexicographic product: enumerate $S(g)$ and for each result t enumerate $S(g')$ and concatenate t with each result

Decomposability: no duplicates

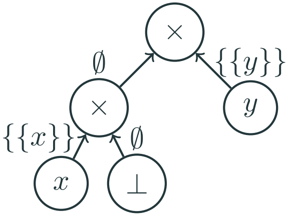
Normalization: handling $\emptyset$



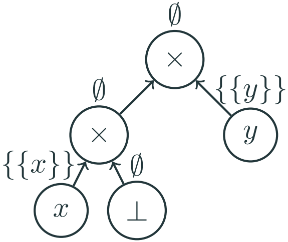
Normalization: handling $\emptyset$



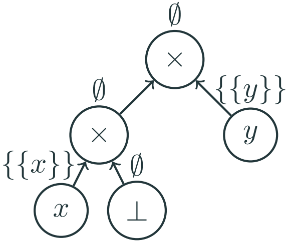
Normalization: handling $\emptyset$



Normalization: handling $\emptyset$

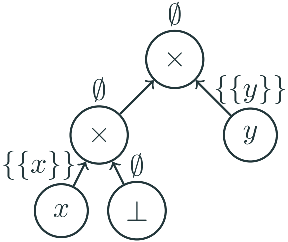


Normalization: handling $\emptyset$



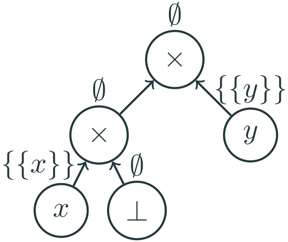
- **Problem:** if $S(g) = \emptyset$ we waste time

Normalization: handling $\emptyset$



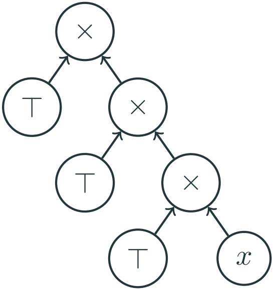
- **Problem:** if $S(g) = \emptyset$ we waste time
- **Solution:** in preprocessing
  - compute **bottom-up** if $S(g) = \emptyset$

Normalization: handling $\emptyset$

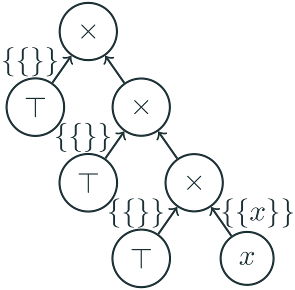


- **Problem:** if $S(g) = \emptyset$ we waste time
- **Solution:** in preprocessing
  - compute **bottom-up** if $S(g) = \emptyset$
  - then get rid of the gate

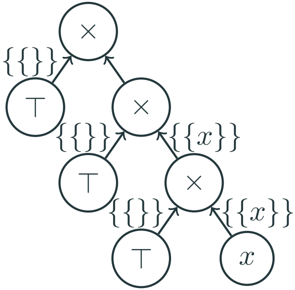
Normalization: handling empty assignments



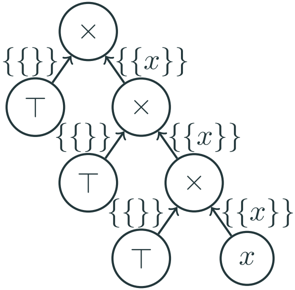
Normalization: handling empty assignments



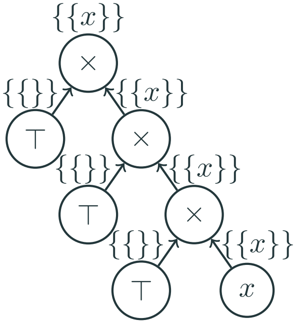
Normalization: handling empty assignments



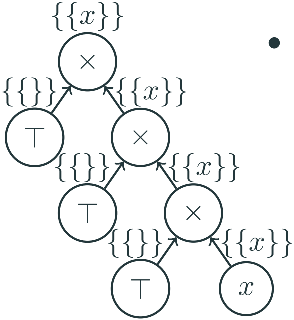
Normalization: handling empty assignments



Normalization: handling empty assignments

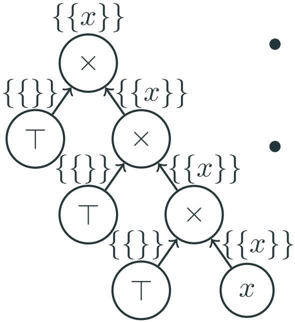


Normalization: handling empty assignments



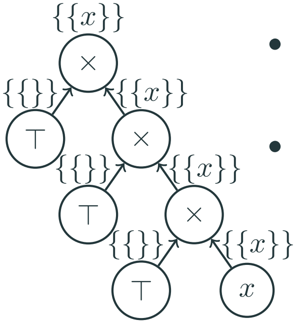
- **Problem:** if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates

Normalization: handling empty assignments



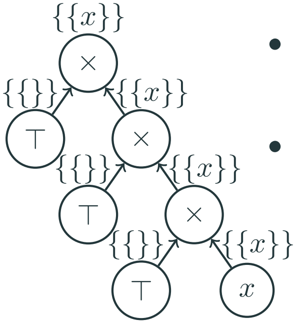
- **Problem:** if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- **Solution:**

Normalization: handling empty assignments



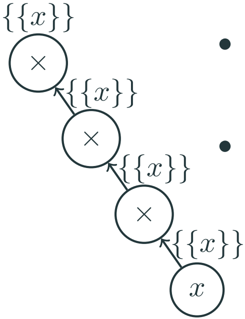
- **Problem:** if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- **Solution:**
  - **split** g between $S(g) \cap \{\{\}\}$ and $S(g) \setminus \{\{\}\}$ (homogenization)

Normalization: handling empty assignments



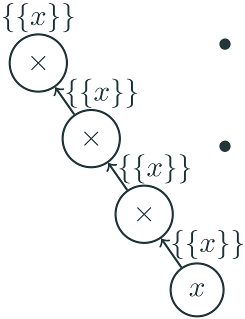
- **Problem:** if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- **Solution:**
  - **split** g between $S(g) \cap \{\{\}\}$ and $S(g) \setminus \{\{\}\}$ (homogenization)
  - **remove** inputs with $S(g) = \{\{\}\}$ for $\times$-gates

Normalization: handling empty assignments



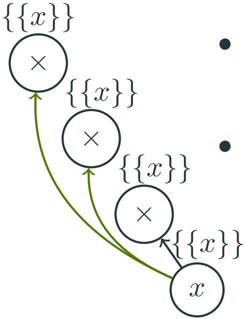
- **Problem:** if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- **Solution:**
  - **split** g between $S(g) \cap \{\{\}\}$ and $S(g) \setminus \{\{\}\}$ (homogenization)
  - **remove** inputs with $S(g) = \{\{\}\}$ for $\times$-gates

Normalization: handling empty assignments



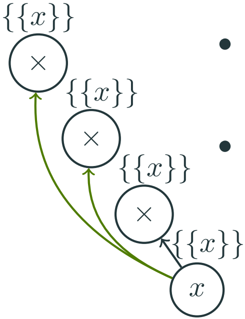
- **Problem:** if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- **Solution:**
  - **split** g between $S(g) \cap \{\{\}\}$ and $S(g) \setminus \{\{\}\}$ (homogenization)
  - **remove** inputs with $S(g) = \{\{\}\}$ for $\times$-gates
  - **collapse** $\times$-chains with fan-in 1

Normalization: handling empty assignments



- **Problem:** if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- **Solution:**
  - **split** g between $S(g) \cap \{\{\}\}$ and $S(g) \setminus \{\{\}\}$ (homogenization)
  - **remove** inputs with $S(g) = \{\{\}\}$ for $\times$-gates
  - **collapse** $\times$-chains with fan-in 1

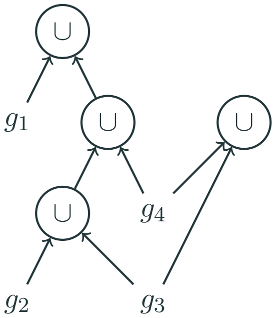
Normalization: handling empty assignments



- **Problem:** if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- **Solution:**
  - **split** g between $S(g) \cap \{\{\}\}$ and $S(g) \setminus \{\{\}\}$ (homogenization)
  - **remove** inputs with $S(g) = \{\{\}\}$ for $\times$-gates
  - **collapse** $\times$-chains with fan-in 1

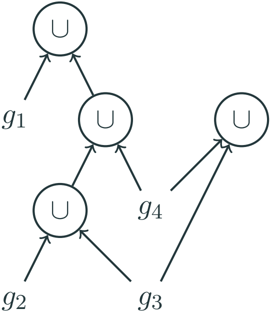
$\rightarrow$ Now, when traversing a **$\times$-gate** we make progress: **non-trivial split** of each set

Indexing: handling ∪-hierarchies



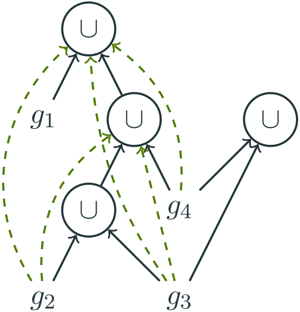
- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)

Indexing: handling ∪-hierarchies



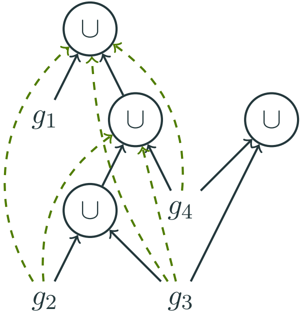
- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)
- **Solution:** compute **reachability index**

Indexing: handling ∪-hierarchies



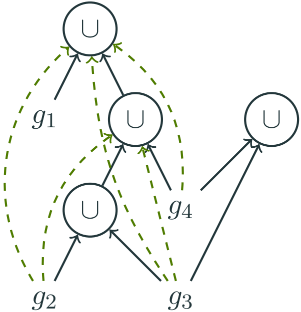
- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)
- **Solution:** compute **reachability index**

Indexing: handling ∪-hierarchies



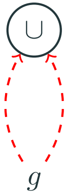
- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)
- **Solution:** compute **reachability index**
- **Problem:** must be done in **linear time**

Indexing: handling ∪-hierarchies

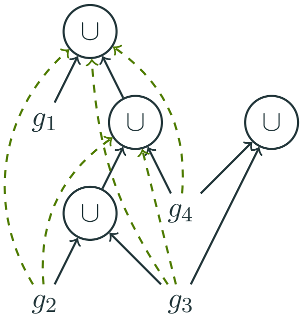


- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)
- **Solution:** compute **reachability index**
- **Problem:** must be done in **linear time**

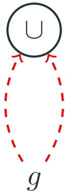
- **Solution:** Determinism ensures we have a **multitree** (we cannot have the pattern at the right)



Indexing: handling ∪-hierarchies



- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)
- **Solution:** compute **reachability index**
- **Problem:** must be done in **linear time**



- **Solution:** Determinism ensures we have a **multitree** (we cannot have the pattern at the right)
- **Custom** constant-delay reachability index for multitrees

Outline

Commutative semiring $(K, \mathbb{0}, \mathbb{1}, \oplus, \otimes)$

- Set K with distinguished elements $\mathbb{0}$, $\mathbb{1}$
- $\oplus$ associative, commutative operator, with identity $\mathbb{0}_K$:
  - $a \oplus (b \oplus c) = (a \oplus b) \oplus c$
  - $a \oplus b = b \oplus a$
  - $a \oplus \mathbb{0} = \mathbb{0} \oplus a = a$
- $\otimes$ associative, commutative operator, with identity $\mathbb{1}_K$:
  - $a \otimes (b \otimes c) = (a \otimes b) \otimes c$
  - $a \otimes b = b \otimes a$
  - $a \otimes \mathbb{1} = \mathbb{1} \otimes a = a$
- $\otimes$ distributes over $\oplus$:

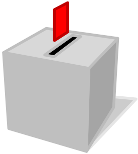
$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$$

- $\mathbb{0}$ is annihilating for $\otimes$:

$$a \otimes \mathbb{0} = \mathbb{0} \otimes a = \mathbb{0}$$

Commutative semiring examples

Which commutative semirings do you know about?



Example semirings

- $(\mathbb{N}, 0, 1, +, \times)$: **counting** semiring
- $(\{\bot, \top\}, \bot, \top, \vee, \wedge)$: **Boolean** semiring
- $(\{\textit{unclassified}, \textit{restricted}, \textit{confidential}, \textit{secret}, \textit{top secret}\},$ $\textit{top secret}, \textit{unclassified}, \min, \max)$: **security** semiring
- $(\mathbb{N} \cup \{\infty\}, \infty, 0, \min, +)$: **tropical** semiring
- $(\{\text{Boolean functions over } \mathcal{X}\}, \bot, \top, \vee, \wedge)$: semiring of **Boolean functions** over $\mathcal{X}$
- $(\mathbb{N}[\mathcal{X}], 0, 1, +, \times)$: semiring of integer-valued **polynomials** with variables in $\mathcal{X}$ (also called **How**-semiring or **universal** semiring)

Semiring provenance [Green et al., 2007]

- We **fix** a semiring $(K, \mathbb{0}, \mathbb{1}, \oplus, \otimes)$
- We assume provenance annotations are **in K**
- We consider a query Q from the **positive relational algebra** (selection, projection, renaming, product, union)
- We define a semantics for the provenance of a tuple $t \in Q(D)$ **inductively** on the structure of Q just like before

Selection, renaming

Provenance annotations of selected tuples are **unchanged**

Example $(\rho_{\mathbf{name}\to\mathbf{n}}(\sigma_{\mathbf{city}=\text{“New York”}}(R)))$

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

n	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2

Projection

Provenance annotations of identical, merged, tuples are $\oplus$-ed

Example ($\pi_{\textbf{city}}(R)$)

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

city	prov
New York	$x_1 \oplus x_2$
Paris	$x_3 \oplus x_5 \oplus x_6$
Berlin	$x_4 \oplus x_7$

Union

Provenance annotations of identical, merged, tuples are $\oplus$-ed

Example

$\pi_{\text{city}}(\sigma_{\textit{ends-with}(\text{position},\text{“agent”})}(R)) \cup \pi_{\text{city}}(\sigma_{\text{position}=\text{“Analyst”}}(R))$

name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

city	prov
Paris	$x_3 \oplus x_5$
Berlin	$x_4 \oplus x_7$

Cross product

Provenance annotations of combined tuples are $\otimes$-ed

Example

$\pi_{\text{city}}(\sigma_{\textit{ends-with}(\text{position},\text{“agent”})}(R)) \bowtie \pi_{\text{city}}(\sigma_{\text{position}=\text{“Analyst”}}(R))$

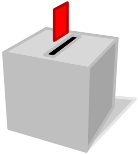
name	position	city	classification	prov
John	Director	New York	unclassified	x_1
Paul	Janitor	New York	restricted	x_2
Dave	Analyst	Paris	confidential	x_3
Ellen	Field agent	Berlin	secret	x_4
Magdalen	Double agent	Paris	top secret	x_5
Nancy	HR director	Paris	restricted	x_6
Susan	Analyst	Berlin	secret	x_7

city	prov
Paris	$x_3 \otimes x_5$
Berlin	$x_4 \otimes x_7$

Poll: counting semiring

Say we annotate each tuple of the input database by 1 and evaluate a query with provenance in $(\mathbb{N}, 0, 1, +, \times)$. What will the provenance of every result mean?

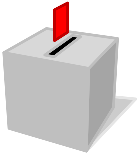
- **A**: The number of possible worlds giving the result
- **B**: The minimum number of tuples required to obtain the result
- **C**: The number of times the result is obtained
- **D**: The number of subqueries giving the result



Poll: counting semiring

Say we annotate each tuple of the input database by 1 and evaluate a query with provenance in $(\mathbb{N}, 0, 1, +, \times)$. What will the provenance of every result mean?

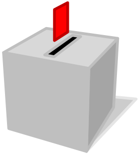
- **A**: The number of possible worlds giving the result
- **B**: The minimum number of tuples required to obtain the result
- **C**: **The number of times the result is obtained**
- **D**: The number of subqueries giving the result



Poll: universal provenance

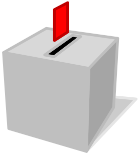
There is a semiring for which the provenance that we obtain is the most informative, i.e., we can recover provenance for any other semiring from it. Which one is it?

- **A**: The tropical semiring
- **B**: The semiring $\mathbb{N}[X]$
- **C**: The semiring of Boolean functions
- **D**: The security semiring



Poll: universal provenance

There is a semiring for which the provenance that we obtain is the most informative, i.e., we can recover provenance for any other semiring from it. Which one is it?



- **A**: The tropical semiring
- **B**: **The semiring** $\mathbb{N}[X]$
- **C**: The semiring of Boolean functions
- **D**: The security semiring

What can we do with semiring provenance?

counting semiring: count the number of times a tuple can be derived, multiset semantics

Boolean semiring: determines if a tuple exists when a subdatabase is selected

security semiring: determines the minimum clearance level required to get a tuple as a result

tropical semiring: minimum-weight way of deriving a tuple (think shortest path in a graph)

Boolean functions: Boolean provenance, as previously defined

integer polynomials: $\mathbb{N}[X]$, universal provenance, see further

Example of security provenance

$$\pi_{\text{city}}(\sigma_{\text{name}<\text{name2}}(\pi_{\text{name,city}}(R) \bowtie \rho_{\text{name}\to\text{name2}}(\pi_{\text{name,city}}(R))))$$

name	position	city	prov
John	Director	New York	unclassified
Paul	Janitor	New York	restricted
Dave	Analyst	Paris	confidential
Ellen	Field agent	Berlin	secret
Magdalen	Double agent	Paris	top secret
Nancy	HR director	Paris	restricted
Susan	Analyst	Berlin	secret

city	prov
New York	restricted
Paris	confidential
Berlin	secret

Properties [Green et al., 2007]

- Semiring provenance still has PTIME data overhead

Properties [Green et al., 2007]

- Semiring provenance still has PTIME data overhead
- Semiring homomorphisms commute with provenance computation: if $K \xrightarrow{\text{hom}} K'$, then one can compute the provenance in K, apply the homomorphism, and obtain the same result as when computing provenance in K'

Properties [Green et al., 2007]

- Semiring provenance still has **PTIME** data overhead
- Semiring homomorphisms **commute** with provenance computation: if $K \xrightarrow{\text{hom}} K'$, then one can compute the provenance in K, apply the homomorphism, and obtain the same result as when computing provenance in K'
- The integer polynomial semiring $\mathbb{N}[X]$ is **universal**: there is a unique homomorphism to any other commutative semiring that respects a given valuation of the variables

Properties [Green et al., 2007]

- Semiring provenance still has **PTIME** data overhead
- Semiring homomorphisms **commute** with provenance computation: if $K \xrightarrow{\text{hom}} K'$, then one can compute the provenance in K, apply the homomorphism, and obtain the same result as when computing provenance in K'
- The integer polynomial semiring $\mathbb{N}[X]$ is **universal**: there is a unique homomorphism to any other commutative semiring that respects a given valuation of the variables
- This means **all computations can be performed in the universal semiring**, and homomorphisms applied next

Properties [Green et al., 2007]

- Semiring provenance still has **PTIME** data overhead
- Semiring homomorphisms **commute** with provenance computation: if $K \xrightarrow{\text{hom}} K'$, then one can compute the provenance in K, apply the homomorphism, and obtain the same result as when computing provenance in K'
- The integer polynomial semiring $\mathbb{N}[X]$ is **universal**: there is a unique homomorphism to any other commutative semiring that respects a given valuation of the variables
- This means **all computations can be performed in the universal semiring**, and homomorphisms applied next
- Two **equivalent queries** can have two **different provenance annotations** on the same database, in some semirings

Outline

Outline

Desiderata for a provenance-aware DBMS

- Extends a **widely used** database management system
- **Easy to deploy**
- **Easy to use**, transparent for the user
- Provenance **automatically maintained** as the user interacts with the database management system
- Provenance computation **benefits from query optimization** within the DBMS
- Allow **probability computation** based on provenance
- **Any form of provenance** can be computed: Boolean provenance, semiring provenance in any semiring (possibly, with monus), aggregate provenance, **on demand**

ProvSQL: Provenance within PostgreSQL (1/2) [Senellart et al., 2018]

- Lightweight extension/plugin for PostgreSQL ≥ 9.5
- Provenance annotations stored as UUIDs, in an extra attribute of each provenance-aware relation
- A provenance circuit relating UUIDs of elementary provenance annotations and arithmetic gates stored as tables
- All computations done in the universal semiring (more precisely, with monus, in the free semiring with monus)

ProvSQL: Provenance within PostgreSQL (2/2) [Senellart et al., 2018]

- **Query rewriting** to automatically compute output provenance attributes in terms of the query and input provenance attributes:
  - Duplicate elimination (DISTINCT, set union) results in aggregation of provenance values with $\oplus$
  - Cross products, joins results in combination of provenance values with $\otimes$
  - Difference results in combination of provenance values with $\ominus$
- **Probability computation** from the provenance circuits, via various methods (naive, sampling, compilation to d-DNNFs)

Challenges

- **Low-level** access to PostgreSQL data structures in extensions
- No simple **query rewriting** mechanism
- SQL is much **less clean** than the relational algebra
- **Multiset semantics** by default in SQL
- SQL is a very **rich language**, with many different ways of expressing the same thing
- Inherent **limitations**: e.g., no aggregation within recursive queries
- Implementing provenance computation should **not slow down** the computation
- User-defined functions, updates, etc.: **unclear** how provenance should work

ProvSQL: Current status

- **Supported** SQL language features:
  - Regular SELECT-FROM-WHERE queries (aka conjunctive queries with multiset semantics)
  - JOIN queries (regular joins and outer joins; semijoins and antijoins are not currently supported)
  - SELECT queries with nested SELECT subqueries in the FROM clause
  - GROUP BY queries (without aggregation)
  - SELECT DISTINCT queries (i.e., set semantics)
  - UNION's or UNION ALL's of SELECT queries
  - EXCEPT queries
- Longer term project: aggregate computation
- Homepage: `https://github.com/PierreSenellart/provsql`

Provenance applications in practice

- How can we do probabilistic query evaluation via provenance?
    - ProvSQL is interfaced with c2d, d4, and dsharp

Provenance applications in practice

- How can we do probabilistic query evaluation via provenance?
  - ProvSQL is interfaced with c2d, d4, and dsharp
- How can we do enumeration via provenance?
  - Prototype: `https://github.com/PoDMR/enum-spanner-rs`

Provenance applications in practice

- How can we do probabilistic query evaluation via provenance?
  - ProvSQL is interfaced with c2d, d4, and dsharp
- How can we do enumeration via provenance?
  - Prototype: `https://github.com/PoDMR/enum-spanner-rs`
- Remark: missing studies of provenance notions used in the real world, e.g., “data lineage” used by Pachyderm

Provenance in theory

- Confession: as a theoretical topic, provenance feels definitional
  - → Recipe: take a complicated query language, define some complicated notion of provenance, appeal to scary algebraic structures, add one more paper to the pile...
- Which directions are less definitional?

Provenance in theory

- Confession: as a theoretical topic, provenance feels definitional
  → Recipe: take a complicated query language, define some complicated notion of provenance, appeal to scary algebraic structures, add one more paper to the pile...
- Which directions are less definitional?
  - Using provenance for computational tasks

Provenance in theory

- Confession: as a theoretical topic, provenance feels definitional
  → Recipe: take a complicated query language, define some complicated notion of provenance, appeal to scary algebraic structures, add one more paper to the pile...
- Which directions are less definitional?
  - Using provenance for computational tasks
    - We have seen two examples : probabilities and enumeration
    - In both cases, provenance competes against other approaches
    - Sometimes, provenance provides new insights

Provenance in theory

- Confession: as a theoretical topic, provenance feels definitional
  → Recipe: take a complicated query language, define some complicated notion of provenance, appeal to scary algebraic structures, add one more paper to the pile...
- Which directions are less definitional?
  - Using provenance for computational tasks
    - We have seen two examples : probabilities and enumeration
    - In both cases, provenance competes against other approaches
    - Sometimes, provenance provides new insights
  - Showing bounds on provenance representations

Provenance in theory

- Confession: as a theoretical topic, provenance feels definitional
  → Recipe: take a complicated query language, define some complicated notion of provenance, appeal to scary algebraic structures, add one more paper to the pile...
- Which directions are less definitional?
  - Using provenance for computational tasks
    - We have seen two examples : probabilities and enumeration
    - In both cases, provenance competes against other approaches
    - Sometimes, provenance provides new insights
  - Showing bounds on provenance representations
    - Connects to knowledge compilation work on circuit classes
    - Can be easier than computational complexity lower bounds

Provenance in theory

- Confession: as a theoretical topic, provenance feels definitional
  - → Recipe: take a complicated query language, define some complicated notion of provenance, appeal to scary algebraic structures, add one more paper to the pile...
- Which directions are less definitional?
  - Using provenance for computational tasks
    - We have seen two examples : probabilities and enumeration
    - In both cases, provenance competes against other approaches
    - Sometimes, provenance provides new insights
  - Showing bounds on provenance representations
    - Connects to knowledge compilation work on circuit classes
    - Can be easier than computational complexity lower bounds

Thanks for your attention!

Bibliography i

Antoine Amarilli, Pierre Bourhis, Louis Jachiet, and Stefan Mengel. A Circuit-Based Approach to Efficient Enumeration. In *ICALP*, 2017.

Daniel Deutch, Tova Milo, Sudeepa Roy, and Val Tannen. Circuits for Datalog provenance. In *ICDT*, 2014.

Todd J Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, 2007.

Pierre Senellart, Louis Jachiet, Silviu Maniu, and Yann Ramusat. ProvSQL: provenance and probability management in postgresql. 2018. Demonstration.

Credits

Original class material by Pierre Senellart