Efficient Enumeration Algorithms via Circuits

Antoine Amarilli

October 18, 2023

Télécom Paris

General motivation (for database people)

- Intensional query evaluation: given a query Q and a database D

General motivation (for database people)

- Intensional query evaluation: given a query Q and a database D
  - Compile Q on D to a circuit C in a tractable class (d-SDNNF, uOBDD, ...)
    - Other names: grounding Q on D, computing the provenance of Q on D...

General motivation (for database people)

- Intensional query evaluation: given a query Q and a database D
  - Compile Q on D to a circuit C in a tractable class (d-SDNNF, uOBDD, ...)
    - Other names: grounding Q on D, computing the provenance of Q on D...
  - Use the circuit C to retrieve the answer of Q on D

General motivation (for database people)

- Intensional query evaluation: given a query Q and a database D
  - Compile Q on D to a circuit C in a tractable class (d-SDNNF, uOBDD, ...)
    - Other names: grounding Q on D, computing the provenance of Q on D...
  - Use the circuit C to retrieve the answer of Q on D
- Monday: intensional query evaluation for counting and probability computation

General motivation (for database people)

- **Intensional query evaluation**: given a query Q and a database D
  - **Compile** Q on D to **a circuit** C in a tractable class (d-SDNNF, uOBDD, ...)
    - Other names: **grounding** Q on D, computing the **provenance** of Q on D...
  - **Use** the circuit C to retrieve the answer of Q on D
- **Monday:** intensional query evaluation for **counting** and **probability computation**
- **Today:** can we use the intensional approach to **enumerate** query answers?

General motivation (for database people)

- **Intensional query evaluation**: given a query Q and a database D
  - **Compile** Q on D to **a circuit** C in a tractable class (d-SDNNF, uOBDD, ...)
    - Other names: **grounding** Q on D, computing the **provenance** of Q on D...
  - **Use** the circuit C to retrieve the answer of Q on D
- **Monday:** intensional query evaluation for **counting** and **probability computation**
- **Today:** can we use the intensional approach to **enumerate** query answers?

Structure of the talk:

- Preliminaries and problem statement

General motivation (for database people)

- **Intensional query evaluation**: given a query Q and a database D
  - **Compile** Q on D to **a circuit** C in a tractable class (d-SDNNF, uOBDD, ...)
    - Other names: **grounding** Q on D, computing the **provenance** of Q on D...
  - **Use** the circuit C to retrieve the answer of Q on D
- **Monday:** intensional query evaluation for **counting** and **probability computation**
- **Today:** can we use the intensional approach to **enumerate** query answers?

Structure of the talk:

- Preliminaries and problem statement
- Efficient enumeration for **d-DNNF set circuits**

General motivation (for database people)

- **Intensional query evaluation**: given a query Q and a database D
  - **Compile** Q on D to **a circuit** C in a tractable class (d-SDNNF, uOBDD, ...)
    - Other names: **grounding** Q on D, computing the **provenance** of Q on D...
  - **Use** the circuit C to retrieve the answer of Q on D
- **Monday:** intensional query evaluation for **counting** and **probability computation**
- **Today:** can we use the intensional approach to **enumerate** query answers?

Structure of the talk:

- Preliminaries and problem statement
- Efficient enumeration for **d-DNNF set circuits**
- Applications: Using enumeration on circuits for **query evaluation**

Dramatis Personae


Antoine Amarilli


Pierre Bourhis


Florent Capelli


Louis Jachiet


Stefan Mengel


Mikaël Monet


Martín Muñoz


Matthias Niewerth


Cristian Riveros

- Amarilli, A., Bourhis, P., Jachiet, L., and Mengel, S.
  A Circuit-Based Approach to Efficient Enumeration. ICALP 2017.
- Amarilli, A., Bourhis, P., and Mengel, S.
  Enumeration on Trees under Relabelings. ICDT 2018.
- Amarilli, A., Bourhis, P., Mengel, S., and Niewerth, M.
  Constant-Delay Enumeration for Nondeterministic Document Spanners. ICDT 2019.
- Amarilli, A., Bourhis, P., Mengel, S., and Niewerth, M.
  Enumeration on Trees with Tractable Combined Complexity and Efficient Updates. PODS 2019.
- Amarilli, A., Bourhis, P., Mengel, S., and Niewerth, M.
  Constant-Delay Enumeration for Nondeterministic Document Spanners. TODS 2020.
- Amarilli, A., Jachiet, L., Muñoz, M., and Riveros, C.
  Efficient Enumeration for Annotated Grammars. PODS 2022.
- Amarilli, A., Bourhis, P., Capelli, F., Monet, M.
  Ranked Enumeration for MSO on Trees via Knowledge Compilation. Under review.

Preliminaries

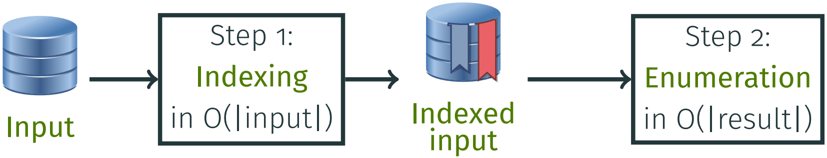
Enumeration algorithms (see Nofar's talks, September workshop)

Input

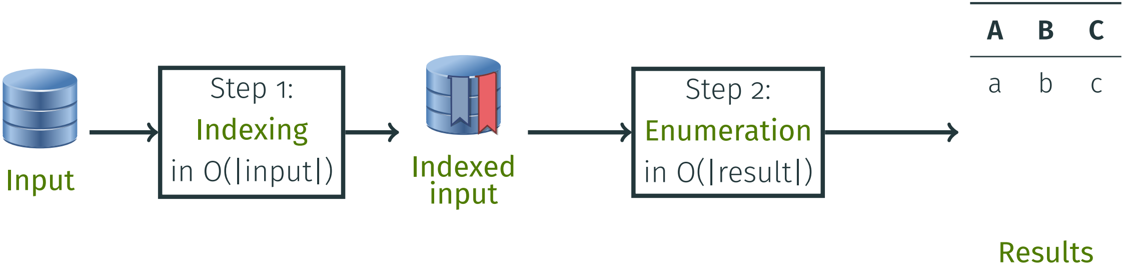
Enumeration algorithms (see Nofar’s talks, September workshop)

Enumeration algorithms (see Nofar's talks, September workshop)

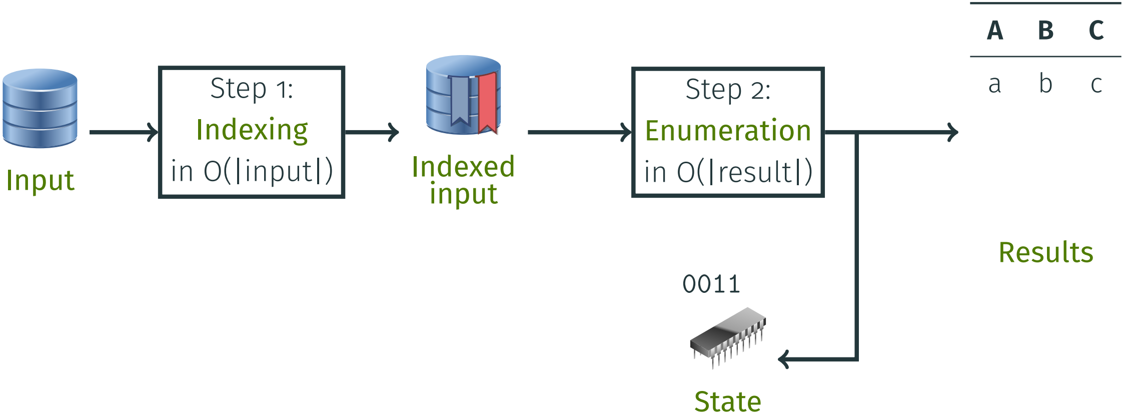
Enumeration algorithms (see Nofar's talks, September workshop)



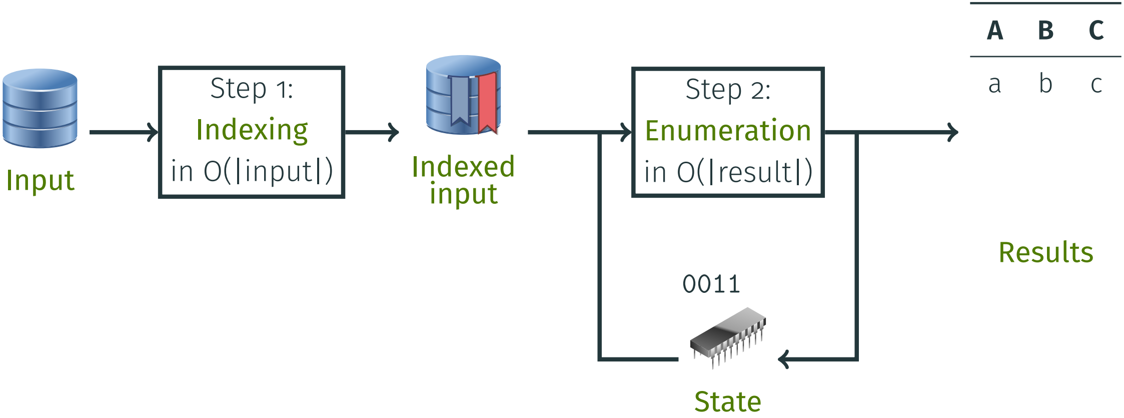
Enumeration algorithms (see Nofar's talks, September workshop)



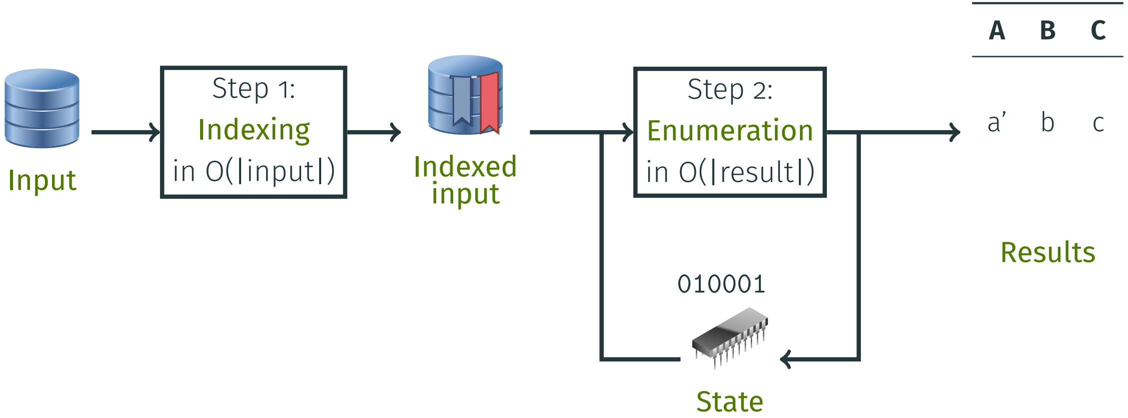
Enumeration algorithms (see Nofar's talks, September workshop)



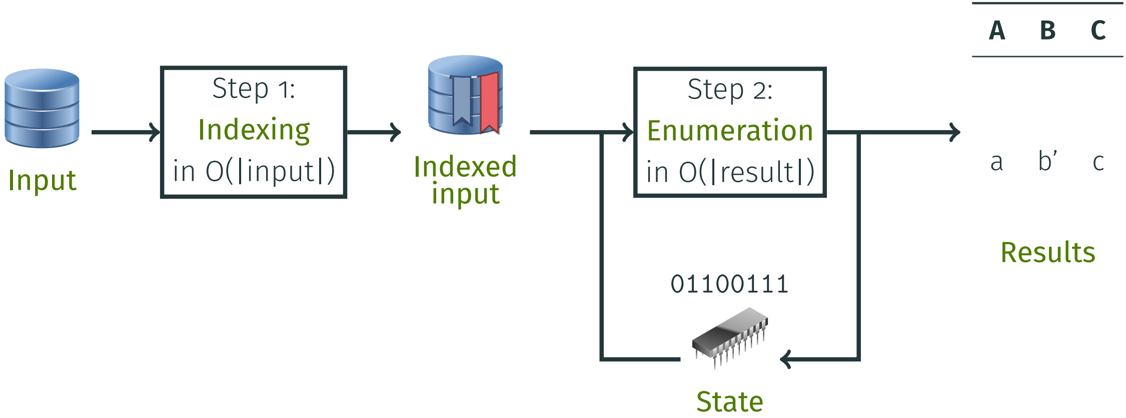
Enumeration algorithms (see Nofar's talks, September workshop)



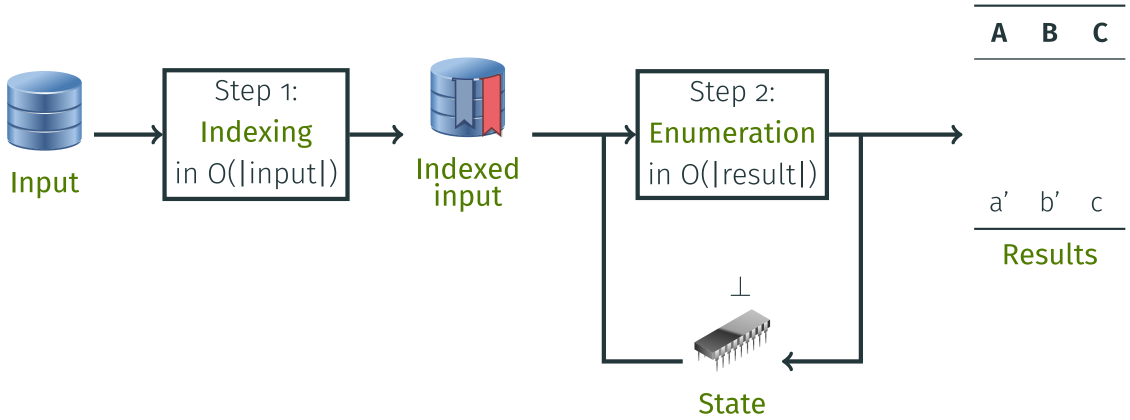
Enumeration algorithms (see Nofar's talks, September workshop)



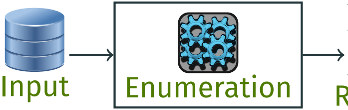
Enumeration algorithms (see Nofar's talks, September workshop)



Enumeration algorithms (see Nofar's talks, September workshop)



WITHOUT knowledge compilation:

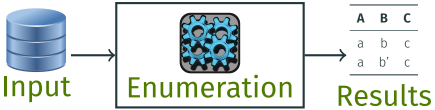


A	B	C
a	b	c
a	b'	c

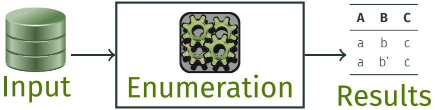
Results

Knowledge compilation (see Guy and YooJung's talks, Boot camp)

WITHOUT knowledge compilation:

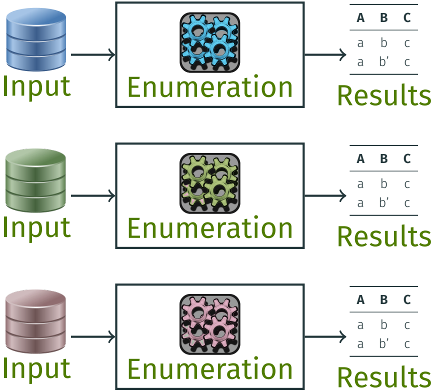


A	B	C
a	b	c
a	b'	c

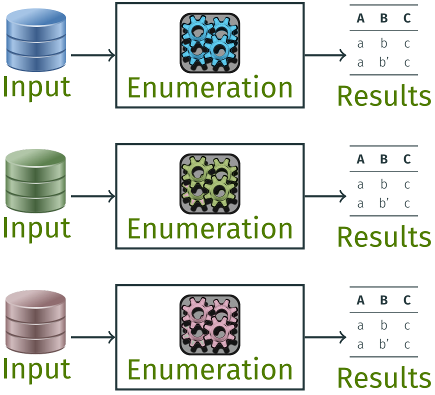


A	B	C
a	b	c
a	b'	c

WITHOUT knowledge compilation:



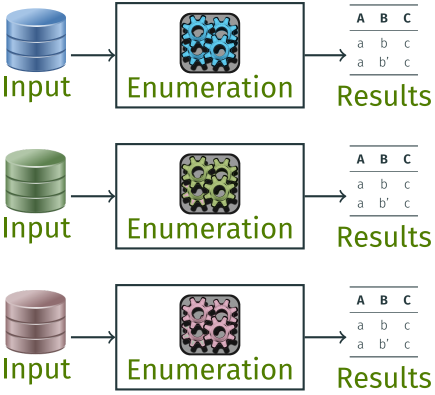
WITHOUT knowledge compilation:



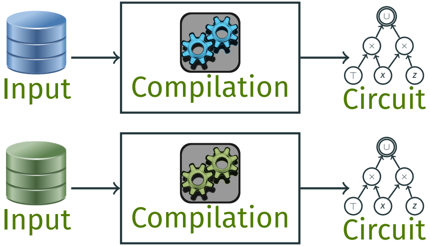
WITH knowledge compilation:

Knowledge compilation (see Guy and YooJung's talks, Boot camp)

WITHOUT knowledge compilation:

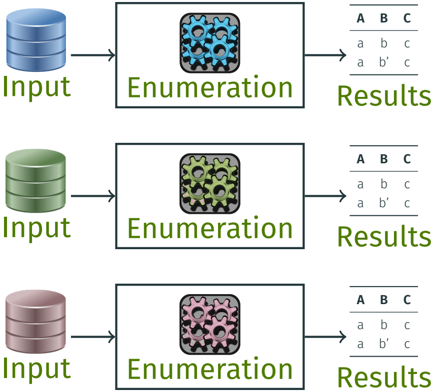


WITH knowledge compilation:

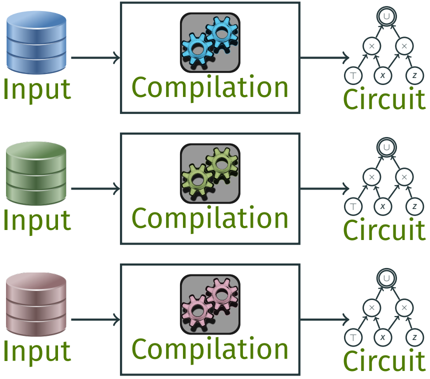


Knowledge compilation (see Guy and YooJung's talks, Boot camp)

WITHOUT knowledge compilation:

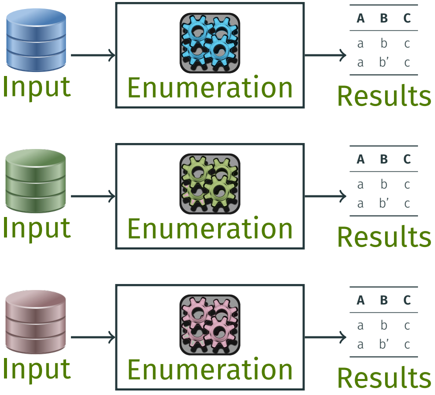


WITH knowledge compilation:

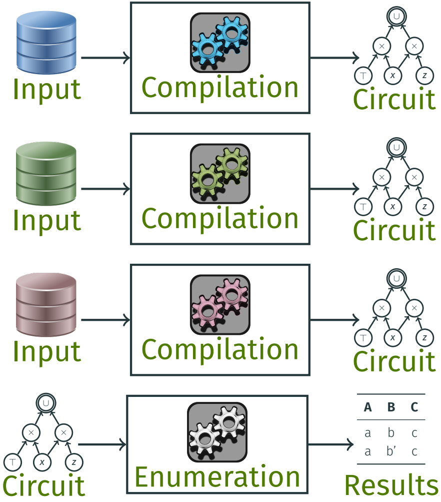


Knowledge compilation (see Guy and YooJung's talks, Boot camp)

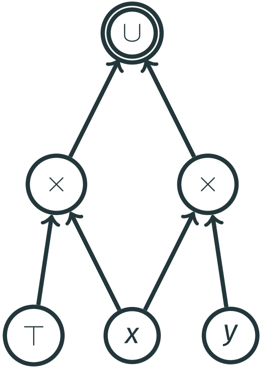
WITHOUT knowledge compilation:



WITH knowledge compilation:

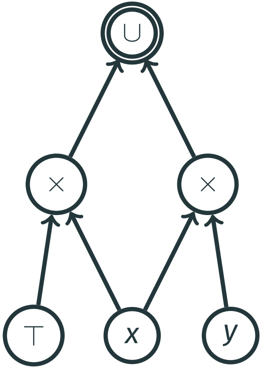


Set circuits



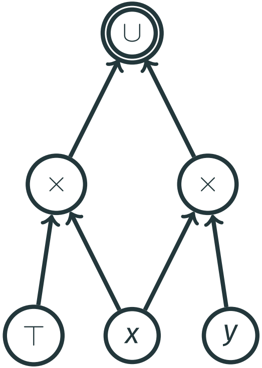
- Directed acyclic graph of **gates**

Set circuits



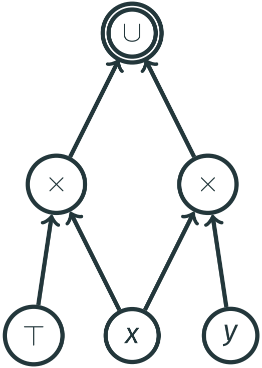
- Directed acyclic graph of **gates**
- **Output** gate: 

Set circuits



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

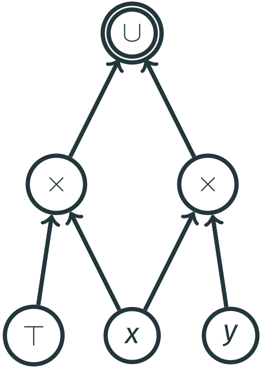
Set circuits



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

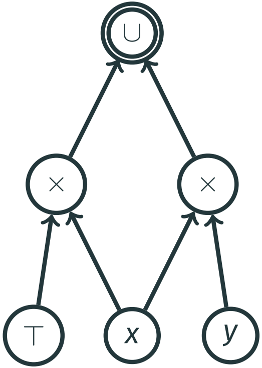
- **Constant** gates: 

Set circuits



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

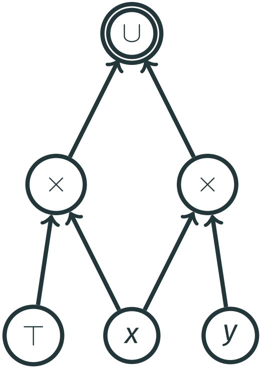
Set circuits



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

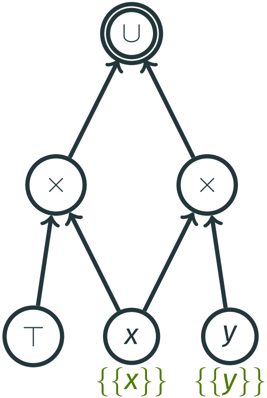
Factorized database fans may find these eerily familiar

Semantics of set circuits



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

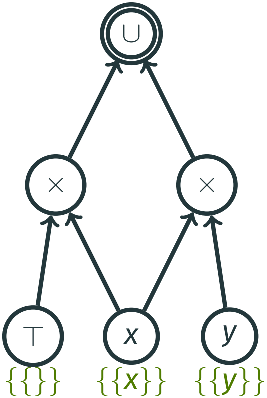
Semantics of set circuits



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

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

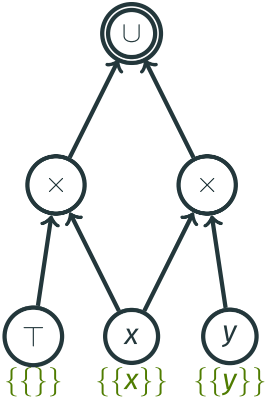
Semantics of set circuits



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

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

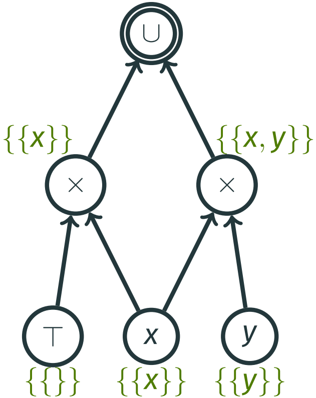
Semantics of set circuits



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: $S(g) = \emptyset$

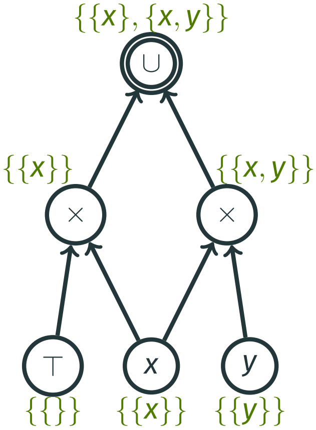
Semantics of set circuits



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:
  $S(g) := \{s_1 \cup s_2 \mid s_1 \in S(g_1), s_2 \in S(g_2)\}$

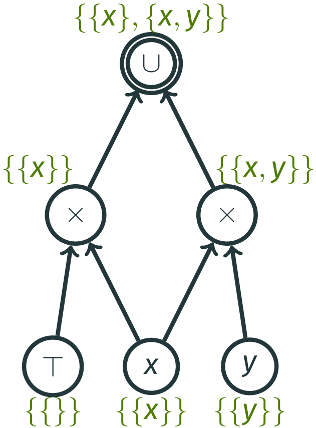
Semantics of set circuits



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

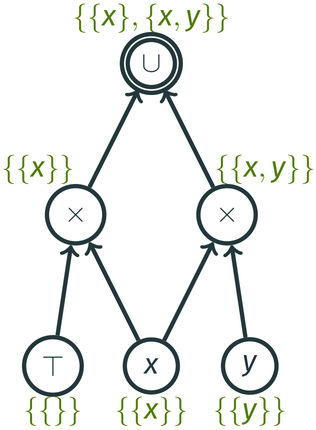


Every gate g **captures** a set $S(g)$ of sets (called **assignments**)

- **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: $S(g) := S(g_1) \cup S(g_2)$

Arithmetic circuit aficionados may see a connection

Semantics of set circuits



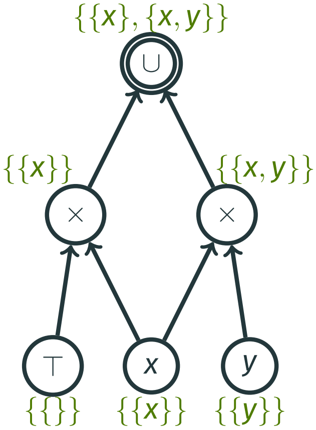
Every gate g **captures** a set $S(g)$ of sets (called **assignments**)

- **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:
  $S(g) := S(g_1) \cup S(g_2)$

Arithmetic circuit aficionados may see a connection

Semiring supporters may have recognized Why$[X]$

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: $S(g) = \{\{\}\}$
- $\bot$-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:
  $S(g) := S(g_1) \cup S(g_2)$

Arithmetic circuit aficionados may see a connection

Semiring supporters may have recognized Why[X]

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

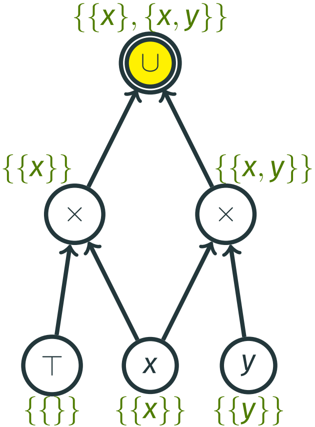
$\rightarrow$E.g., for $S(g) = \{\{x\},\{x,y\}\}$, enumerate $\{x\}$ and then $\{x,y\}$

Circuit restrictions

d-DNNF set circuit:

- ∪ are all **deterministic**:

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



Circuit restrictions

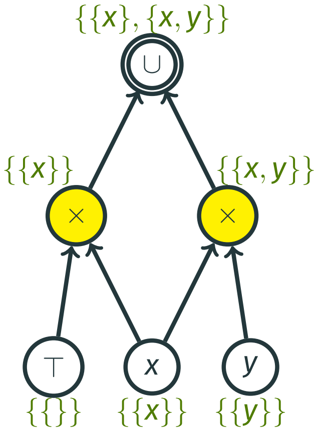
d-DNNF set circuit:

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

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

- $\times$ are all **decomposable**:

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



Main results

Theorem (A., Bourhis, Jachiet, Mengel, ICALP'17)

*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 (A., Bourhis, Jachiet, Mengel, ICALP'17)

*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***

Main results

Theorem (A., Bourhis, Jachiet, Mengel, ICALP'17)

*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***

But where do set circuits come from?

Main results

Theorem (A., Bourhis, Jachiet, Mengel, ICALP'17)

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

But where do set circuits come from?

- Directly when doing intensional query evaluation (see later)

Main results

Theorem (A., Bourhis, Jachiet, Mengel, ICALP'17)

*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***

But where do set circuits come from?

- **Directly** when doing intensional query evaluation (see later)
- **From Boolean circuits:** you can obtain a d-DNNF set circuit:

Main results

Theorem (A., Bourhis, Jachiet, Mengel, ICALP'17)

*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***

But where do set circuits come from?

- **Directly** when doing intensional query evaluation (see later)
- **From Boolean circuits:** you can obtain a d-DNNF set circuit:
  - From a **d-DNNF**, in **quadratic time** (smoothing)

Main results

Theorem (A., Bourhis, Jachiet, Mengel, ICALP'17)

*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***

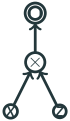
But where do set circuits come from?

- **Directly** when doing intensional query evaluation (see later)
- **From Boolean circuits:** you can obtain a d-DNNF set circuit:
    - From a **d-DNNF**, in **quadratic time** (smoothing)
    - From a **d-SDNNF**, in **linear time** when allowing special gates (implicit smoothing)

Proof techniques

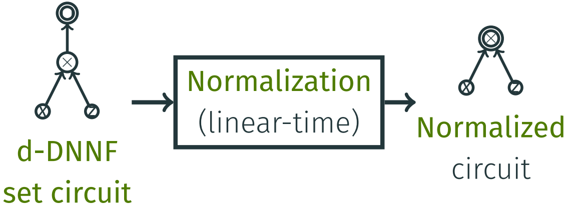
Proof overview

Preprocessing phase:



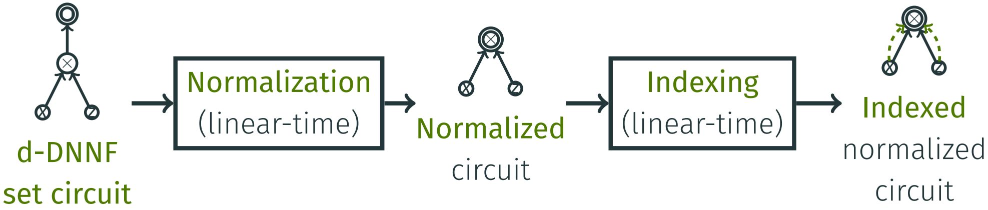
d-DNNF
set circuit

Preprocessing phase:



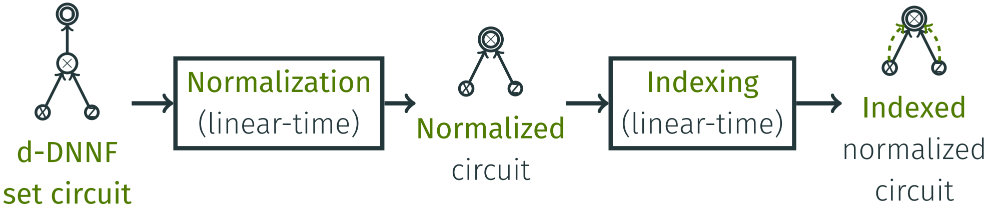
Proof overview

Preprocessing phase:

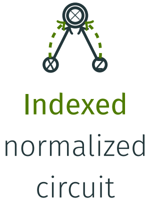


Proof overview

Preprocessing phase:

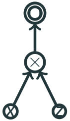


Enumeration phase:



Proof overview

Preprocessing phase:



d-DNNF set circuit → **Normalization** (linear-time) →

Normalized circuit → **Indexing** (linear-time) →

Indexed normalized circuit

Enumeration phase:

Indexed normalized circuit → **Enumeration** (linear delay in each result) →

A	B	C
a	b	c
a	b’	c

Results

Enumerating captured assignments of d-DNNF set circuits

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

→ 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 $\{x, y\}$

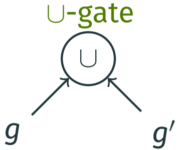
Base case: variable x : enumerate $\{x\}$ and stop

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 $S(g')$

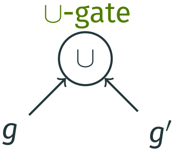
Enumerating captured assignments of d-DNNF set circuits

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 

: enumerate $\{x\}$ and stop



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

Determinism: no duplicates

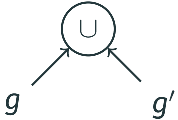
Enumerating captured assignments of d-DNNF set circuits

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

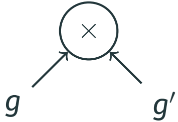
$\cup$-gate



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

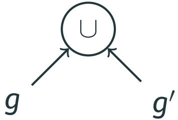
Enumerating captured assignments of d-DNNF set circuits

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

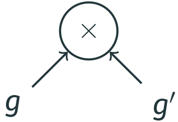
$\cup$-gate



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

Determinism: no duplicates

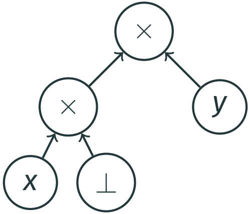
$\times$-gate



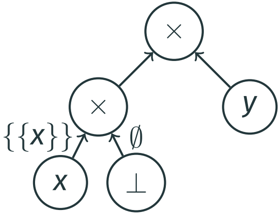
Lexicographic product: enumerate $S(g)$ and for each result t enumerate $S(g')$ and concatenate t with each result

Decomposability: no duplicates

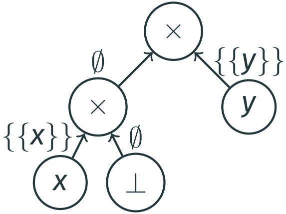
Normalization: handling ∅



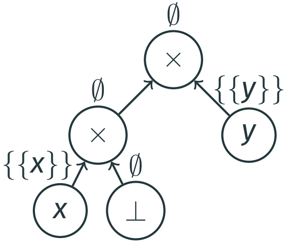
Normalization: handling $\emptyset$



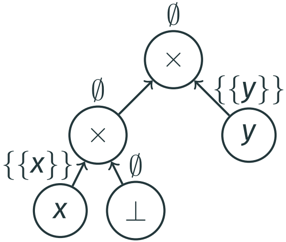
Normalization: handling $\emptyset$



Normalization: handling ∅

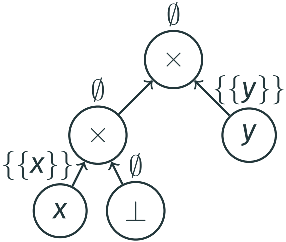


Normalization: handling $\emptyset$



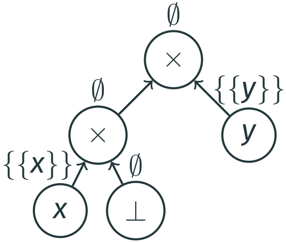
- **Problem:** if $S(g) = \emptyset$ we waste time

Normalization: handling $\emptyset$



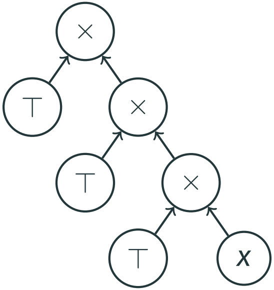
- **Problem:** if $S(g) = \emptyset$ we waste time
- **Solution:** in preprocessing
  - compute **bottom-up** if $S(g) = \emptyset$

Normalization: handling $\emptyset$

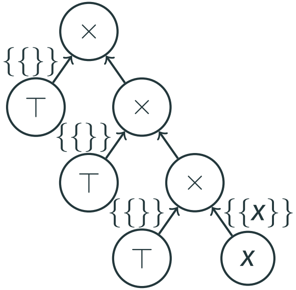


- **Problem:** if $S(g) = \emptyset$ we waste time
- **Solution:** in preprocessing
  - compute **bottom-up** if $S(g) = \emptyset$
  - then get rid of the gate

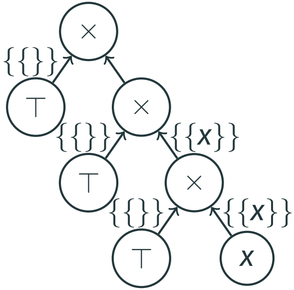
Normalization: handling empty assignments



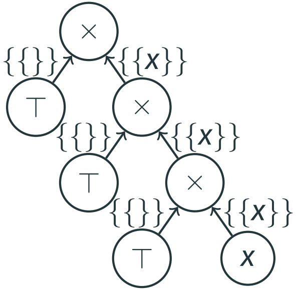
Normalization: handling empty assignments



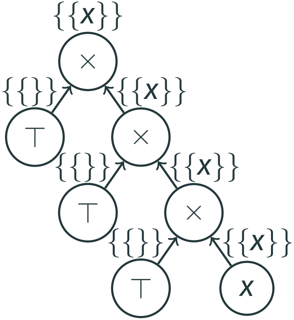
Normalization: handling empty assignments



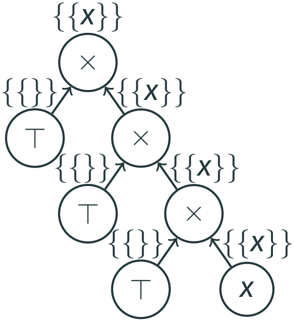
Normalization: handling empty assignments



Normalization: handling empty assignments

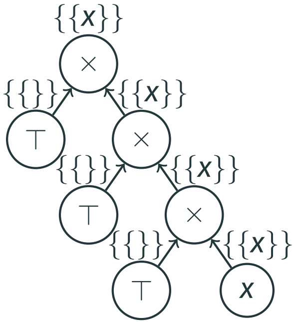


Normalization: handling empty assignments



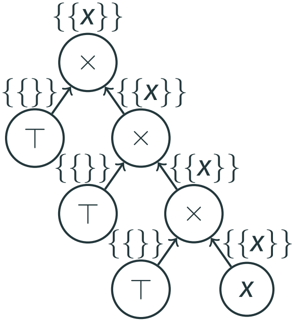
- **Problem:** if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates

Normalization: handling empty assignments



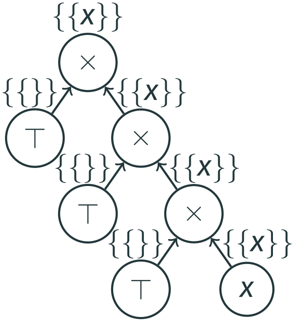
- **Problem:** if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- **Solution:**

Normalization: handling empty assignments



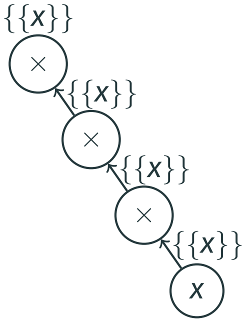
- **Problem:** if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- **Solution:**
  - split g between $S(g) \cap \{\{\}\}$ and $S(g) \setminus \{\{\}\}$ (homogenization)

Normalization: handling empty assignments



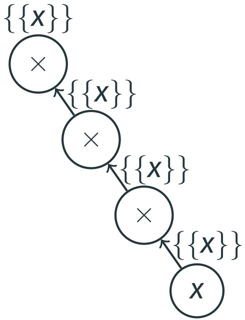
- **Problem:** if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- **Solution:**
  - **split** g between $S(g) \cap \{\{\}\}$ and $S(g) \setminus \{\{\}\}$ (homogenization)
  - **remove** inputs with $S(g) = \{\{\}\}$ for $\times$-gates

Normalization: handling empty assignments



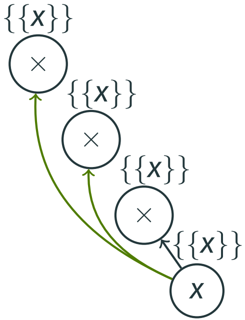
- **Problem:** if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- **Solution:**
  - **split** g between $S(g) \cap \{\{\}\}$ and $S(g) \setminus \{\{\}\}$ (homogenization)
  - **remove** inputs with $S(g) = \{\{\}\}$ for $\times$-gates

Normalization: handling empty assignments



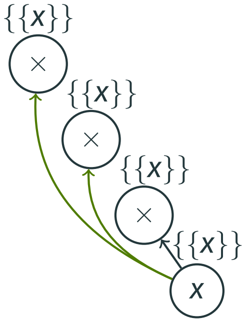
- **Problem:** if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- **Solution:**
  - **split** g between $S(g) \cap \{\{\}\}$ and $S(g) \setminus \{\{\}\}$ (homogenization)
  - **remove** inputs with $S(g) = \{\{\}\}$ for $\times$-gates
  - **collapse** $\times$-chains with fan-in 1

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

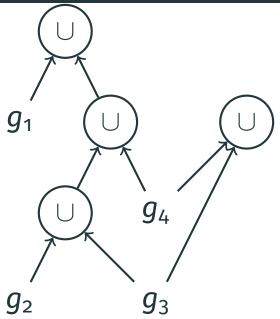
Normalization: handling empty assignments



- **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

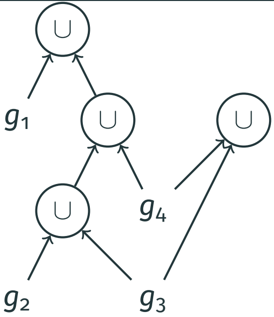
$\rightarrow$ Now, when traversing a **$\times$-gate** we make progress: **non-trivial split** of each set

Indexing: handling ∪-hierarchies

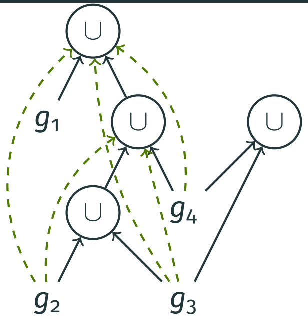


- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)

Indexing: handling ∪-hierarchies

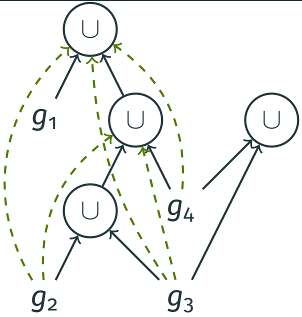


- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)
- **Solution:** compute **reachability index**



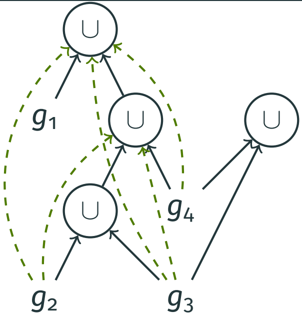
- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)
- **Solution:** compute **reachability index**

Indexing: handling ∪-hierarchies



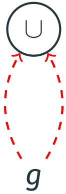
- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)
- **Solution:** compute **reachability index**
- **Problem:** must be done in **linear time**

Indexing: handling ∪-hierarchies

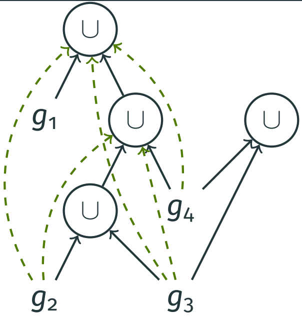


- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)
- **Solution:** compute **reachability index**
- **Problem:** must be done in **linear time**

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

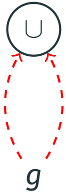


Indexing: handling ∪-hierarchies



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

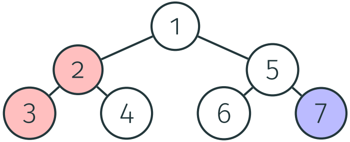
- **Solution:** Determinism ensures we have a **multitree** (we cannot have the pattern at the right)
- **Custom** constant-delay reachability index for multitrees



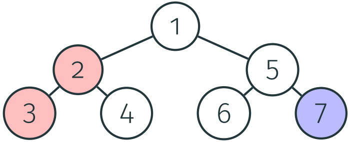
Applications

Application 1: MSO query evaluation on trees

Data: a **tree** T where nodes have a color from an alphabet



Application 1: MSO query evaluation on trees



Data: a **tree** T where nodes have a color from an alphabet ○ ○ ○

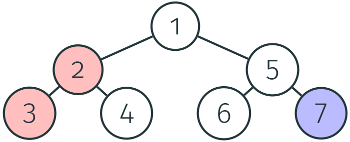
Query Q in monadic second-order logic (MSO)

- $P_{\bigcirc}(x)$ means "x is blue"
- $x \rightarrow y$ means "x is the parent of y"

"Find the pairs of a pink node and a blue node?"

$Q(x, y) := P_{\bigcirc}(x) \wedge P_{\bigcirc}(y)$

Application 1: MSO query evaluation on trees



Data: a **tree** T where nodes have a color from an alphabet $\bigcirc$ $\color{pink}\bullet$ $\color{blue}\bullet$

Query Q in monadic second-order logic (MSO)

- $P_{\color{blue}\bullet}(x)$ means "x is blue"
- $x \rightarrow y$ means "x is the parent of y"

"Find the pairs of a pink node and a blue node?"

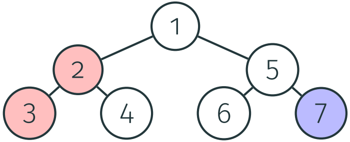
$Q(x, y) := P_{\color{pink}\bullet}(x) \wedge P_{\color{blue}\bullet}(y)$

Result: **Enumerate** all pairs (a, b) of nodes of T such that $Q(a, b)$ holds

results: $(2, 7), (3, 7)$

Application 1: MSO query evaluation on trees

Data: a **tree** T where nodes have a color from an alphabet ○ ○ ○



Query Q in monadic second-order logic (MSO)

- $P_{\bigcirc}(x)$ means "x is blue"
- $x \rightarrow y$ means "x is the parent of y"

"Find the pairs of a pink node and a blue node?"

$Q(x, y) := P_{\bigcirc}(x) \wedge P_{\bigcirc}(y)$

Result: **Enumerate** all pairs (a, b) of nodes of T such that $Q(a, b)$ holds

results: $(2, 7), (3, 7)$

Data complexity: Measure efficiency as a function of T (the query Q is **fixed**)

Application 1: Results

Theorem [Bagan, 2006, Kazana and Segoufin, 2013]

We can enumerate the answers of MSO queries on trees with linear-time preprocessing and constant delay.

Application 1: Results

Theorem [Bagan, 2006, Kazana and Segoufin, 2013]
We can enumerate the answers of MSO queries on trees with ***linear-time preprocessing*** *and* ***constant delay****.*

We can prove this with our methods:

Theorem (A., Bourhis, Jachiet, Mengel, ICALP'17)
For any bottom-up deterministic ***tree automaton*** A *and input* ***tree*** T*,*
we can build a ***d-DNNF set circuit*** *capturing the results of* A *on* T *in* $O(|A| \times |T|)$

Application 1: Results

Theorem [Bagan, 2006, Kazana and Segoufin, 2013]
We can enumerate the answers of MSO queries on trees with ***linear-time preprocessing*** *and* ***constant delay****.*

We can prove this with our methods:

Theorem (A., Bourhis, Jachiet, Mengel, ICALP'17)
For any bottom-up deterministic ***tree automaton*** A *and input* ***tree*** T*,*
we can build a ***d-DNNF set circuit*** *capturing the results of* A *on* T *in* $O(|A| \times |T|)$

- Can be extended to support **relabeling updates** to the tree in $O(\log n)$ time (A., Bourhis, Mengel, ICDT'18)

Application 1: Results

Theorem [Bagan, 2006, Kazana and Segoufin, 2013]
*We can enumerate the answers of MSO queries on trees with **linear-time preprocessing** and **constant delay**.*

We can prove this with our methods:

Theorem (A., Bourhis, Jachiet, Mengel, ICALP'17)
*For any bottom-up deterministic **tree automaton** A and input **tree** T,*
*we can build a **d-DNNF set circuit** capturing the results of A on T in $O(|A| \times |T|)$*

- Can be extended to support **relabeling updates** to the tree in $O(\log n)$ time (A., Bourhis, Mengel, ICDT'18)
- Same result for leaf **insertion/deletion** (A., Bourhis, Mengel, Niewerth, PODS'19) up to **fixing a buggy result** [Niewerth, 2018]

Application 2: Enumerating matches of nondeterministic document spanners

Data: a **text** T

```
Antoine Amarilli Description Name Antoine Amarilli.  Handle:  a3nm.  Identity Born 1990-02-07.  French national.  Appearance as
of 2017.  Auth OpenPGP. OpenId.  Bitcoin.  Contact Email and XMPP a3nm@a3nm.net Affiliation Associate professor of computer
science (office C201-4) in the DIG team of Télécom ParisTech, 46 rue Barrault, F-75634 Paris Cedex 13, France.  Studies
PhD in computer science awarded by Télécom ParisTech on March 14, 2016.  Former student of the École normale supérieure.
test@example.com More Résumé Location Other sites Blogging:  a3nm.net/blog Git:  a3nm.net/git ...
```

Application 2: Enumerating matches of nondeterministic document spanners

Data: a **text** T

```
Antoine Amarilli Description Name Antoine Amarilli.  Handle:  a3nm.  Identity Born 1990-02-07.  French national.  Appearance as
of 2017.  Auth OpenPGP. OpenId.  Bitcoin.  Contact Email and XMPP a3nm@a3nm.net Affiliation Associate professor of computer
science (office C201-4) in the DIG team of Télécom ParisTech, 46 rue Barrault, F-75634 Paris Cedex 13, France.  Studies
PhD in computer science awarded by Télécom ParisTech on March 14, 2016.  Former student of the École normale supérieure.
test@example.com More Résumé Location Other sites Blogging:  a3nm.net/blog Git:  a3nm.net/git ...
```

Query: a **pattern** P given as a regular expression

$$P := \text{␣ [a-z0-9.]* @ [a-z0-9.]* ␣}$$

Application 2: Enumerating matches of nondeterministic document spanners

Data: a **text** T

```
Antoine Amarilli Description Name Antoine Amarilli.  Handle:  a3nm.  Identity Born 1990-02-07.  French national.  Appearance as
of 2017.  Auth OpenPGP. OpenId.  Bitcoin.  Contact Email and XMPP a3nm@a3nm.net Affiliation Associate professor of computer
science (office C201-4) in the DIG team of Télécom ParisTech, 46 rue Barrault, F-75634 Paris Cedex 13, France.  Studies
PhD in computer science awarded by Télécom ParisTech on March 14, 2016.  Former student of the École normale supérieure.
test@example.com More Résumé Location Other sites Blogging:  a3nm.net/blog Git:  a3nm.net/git ...
```

Query: a **pattern** P given as a regular expression

$$P := \text{␣ [a-z0-9.]* @ [a-z0-9.]* ␣}$$

Output: the list of **substrings** of T that match P:

$$[186, 200\rangle, \quad [483, 500\rangle, \quad \dots$$

Application 2: Enumerating matches of nondeterministic document spanners

Data: a **text** T

```
Antoine Amarilli Description Name Antoine Amarilli. Handle: a3nm. Identity Born 1990-02-07. French national. Appearance as
of 2017. Auth OpenPGP. OpenId. Bitcoin. Contact Email and XMPP a3nm@a3nm.net Affiliation Associate professor of computer
science (office C201-4) in the DIG team of Télécom ParisTech, 46 rue Barrault, F-75634 Paris Cedex 13, France. Studies
PhD in computer science awarded by Télécom ParisTech on March 14, 2016. Former student of the École normale supérieure.
test@example.com More Résumé Location Other sites Blogging: a3nm.net/blog Git: a3nm.net/git ...
```

Query: a **pattern** P given as a regular expression

$$P := \text{␣ [a-z0-9.]* @ [a-z0-9.]* ␣}$$

Output: the list of **substrings** of T that match P:

$$[186, 200\rangle, \quad [483, 500\rangle, \quad \ldots$$

Goal:

- be **very efficient** in T (constant-delay)
- be **reasonably efficient** in P (polynomial-time)

Application 2: Results

Theorem (A., Bourhis, Mengel, Niewerth, ICDT'19)

*We can enumerate all matches of an input **nondeterministic automaton with captures** on an input **text** with*

- *Preprocessing **linear** in the text and **polynomial** in the automaton*
- *Delay **constant** in the text and **polynomial** in the automaton*

Application 2: Results

Theorem (A., Bourhis, Mengel, Niewerth, ICDT'19)
*We can enumerate all matches of an input **nondeterministic automaton with captures** on an input **text** with*

- *Preprocessing **linear** in the text and **polynomial** in the automaton*
- *Delay **constant** in the text and **polynomial** in the automaton*

→ Generalizes earlier result on **deterministic automata** [Florenzano et al., 2018]

Application 2: Results

Theorem (A., Bourhis, Mengel, Niewerth, ICDT'19)
*We can enumerate all matches of an input **nondeterministic automaton with captures** on an input **text** with*

- *Preprocessing **linear** in the text and **polynomial** in the automaton*
- *Delay **constant** in the text and **polynomial** in the automaton*

→ Generalizes earlier result on **deterministic automata** [Florenzano et al., 2018]

- Does not really use **d-DNNFs**, but **bounded-width nOBDDs**

Application 2: Results

Theorem (A., Bourhis, Mengel, Niewerth, ICDT'19)
*We can enumerate all matches of an input **nondeterministic automaton with captures** on an input **text** with*

- *Preprocessing **linear** in the text and **polynomial** in the automaton*
- *Delay **constant** in the text and **polynomial** in the automaton*

→ Generalizes earlier result on **deterministic automata** [Florenzano et al., 2018]

- Does not really use **d-DNNFs**, but **bounded-width nOBDDs**

→ Generalizes to **trees** with polynomial dependency in the **tree automaton**

Application 3: Enumerating matches of annotated grammars

Data: a text *T*, e.g., source code

```
long elt, prev, elt2, prev2=-1;
int ret = fscanf(fi, "%ld%ld", &elt, &prev);
if (ret != 2) {
 fprintf(stderr, "Bad offsets after position %ld in index!\n", pi);
 exit(1);
}
```

Application 3: Enumerating matches of annotated grammars

Data: a text T, e.g., **source code**

```
long elt, prev, elt2, prev2=-1;
int ret = fscanf(fi, "%ld%ld", &elt, &prev);
if (ret != 2) {
 fprintf(stderr, "Bad offsets after position %ld in index!\n", pi);
 exit(1);
}
```

Query: a **pattern** P given as a **context-free grammar with annotated terminals**

$$P := \text{“\textit{find all quoted strings in the program}”}$$

Application 3: Enumerating matches of annotated grammars

Data: a text T, e.g., **source code**

```
long elt, prev, elt2, prev2=-1;
int ret = fscanf(fi, "%ld%ld", &elt, &prev);
if (ret != 2) {
 fprintf(stderr, "Bad offsets after position %ld in index!\n", pi);
 exit(1);
}
```

Query: a **pattern** P given as a **context-free grammar with annotated terminals**

$$P := \text{“find all quoted strings in the program”}$$

Theorem (A., Jachiet, Muñoz, Riveros, PODS'22)

*Given an **unambiguous annotation grammar** $\mathcal{G}$ and input text w, we can enumerate the **matches** with preprocessing $O(|\mathcal{G}| \times |w|^3)$ and delay **linear in each assignment***

Application 3: Enumerating matches of annotated grammars

Data: a text T, e.g., source code

```
long elt, prev, elt2, prev2=-1;
int ret = fscanf(fi, "%ld%ld", &elt, &prev);
if (ret != 2) {
 fprintf(stderr, "Bad offsets after position %ld in index!\n", pi);
 exit(1);
}
```

Query: a pattern P given as a context-free grammar with annotated terminals

$$P := \text{“\textit{find all quoted strings in the program}”}$$

Theorem (A., Jachiet, Muñoz, Riveros, PODS'22)

Given an unambiguous annotation grammar $\mathcal{G}$ and input text w, we can enumerate the matches with preprocessing $O(|\mathcal{G}| \times |w|^3)$ and delay linear in each assignment

- Improves on an earlier quintic preprocessing result [Peterfreund, 2021]

Application 3: Enumerating matches of annotated grammars

Data: a text T, e.g., source code

```
long elt, prev, elt2, prev2=-1;
int ret = fscanf(fi, "%ld%ld", &elt, &prev);
if (ret != 2) {
 fprintf(stderr, "Bad offsets after position %ld in index!\n", pi);
 exit(1);
}
```

Query: a pattern P given as a context-free grammar with annotated terminals

$$P := \text{“}\textit{find all quoted strings in the program}\text{”}$$

Theorem (A., Jachiet, Muñoz, Riveros, PODS'22)

Given an unambiguous annotation grammar $\mathcal{G}$ and input text w, we can enumerate the matches with preprocessing $O(|\mathcal{G}| \times |w|^3)$ and delay linear in each assignment

- Improves on an earlier quintic preprocessing result [Peterfreund, 2021]
- Quadratic and linear preprocessing for subclasses (rigid grammars, deterministic pushdown annotators)

Other applications

- Using enumerable compact sets, a fully-persistent version of enumerable d-DNNFs:
    - For visibly pushdown transducers on nested documents in a streaming setting [Muñoz and Riveros, 2022]
    - For annotated automata on SLP-compressed documents, with updates [Muñoz and Riveros, 2023]

Other applications

- Using **enumerable compact sets**, a fully-persistent version of enumerable d-DNNFs:
    - For **visibly pushdown transducers** on **nested documents** in a streaming setting [Muñoz and Riveros, 2022]
    - For **annotated automata** on **SLP-compressed documents**, with updates [Muñoz and Riveros, 2023]
- Query evaluation beyond MSO and variants on words and trees:
    - For **first-order queries** on **bounded expansion** databases [Toruńczyk, 2020]
    - For **ranked direct access** for some **CQs with negation**, see Florent's talk this afternoon

Other applications

- Using **enumerable compact sets**, a fully-persistent version of enumerable d-DNNFs:
  - For **visibly pushdown transducers** on **nested documents** in a streaming setting [Muñoz and Riveros, 2022]
  - For **annotated automata** on **SLP-compressed documents**, with updates [Muñoz and Riveros, 2023]
- Query evaluation beyond MSO and variants on words and trees:
  - For **first-order queries** on **bounded expansion** databases [Toruńczyk, 2020]
  - For **ranked direct access** for some **CQs with negation**, see Florent's talk this afternoon
- Can also be used to enumerate **homomorphisms between structures** [Berkholz and Vinall-Smeeth, 2023]

What about ranked enumeration?

Enumeration algorithms typically give results in an arbitrary (non-controllable) order!

What about ranked enumeration?

Enumeration algorithms typically give results in an arbitrary (non-controllable) order!

- For MSO queries, ranked enumeration is possible with logarithmic delay:
    - First shown for queries on words [Bourhis et al., 2021]
    - Recent preprint (A., Bourhis, Capelli, Monet) for queries on trees under subset-monotone ranking functions
    - (Very) high-level idea: use one priority queue for each gate

What about ranked enumeration?

Enumeration algorithms typically give results in an **arbitrary (non-controllable) order**!

- For **MSO queries**, ranked enumeration is **possible** with **logarithmic** delay:
  - First shown for queries on **words** [Bourhis et al., 2021]
  - Recent preprint (A., Bourhis, Capelli, Monet) for queries on **trees** under subset-monotone ranking functions
  - (Very) high-level idea: use one **priority queue** for each gate
- For **CQs**, results for **ranked access**: [Tziavelis et al., 2022], [Deep et al., 2022], [Carmeli et al., 2023]
  - Also: see Florent's talk

Conclusion

Summary and conclusion

- We can **enumerate** the captured assignments of d-DNNF set circuits
  - → with preprocessing **linear** in the d-DNNF
  - → in delay **linear** in each assignment
  - → in **constant** delay for constant Hamming weight

→ Applies to **MSO enumeration** on **words** and **trees**

→ Applies to enumerate of the matches of **annotated context-free grammars** (with more expensive preprocessing)

→ Can be used for **other applications**

→ In particular: **incremental maintenance** under updates, **ranked enumeration**, etc.

Questions for future work

- What about **negation gates**?
- What can we do without **determinism**? (enumeration for DNNF?)
- Connect results on **updates** to finer bounds on **incremental maintenance** (A., Jachiet, Paperman, ICALP'21)
- Enumerate satisfying assignments via **edits on previous results** (A., Monet, STACS'23) to achieve **constant delay** even on **linear-sized assignments**
- For MSO queries: understand better the connection between **automata classes** and **circuit classes** (e.g., alternating automata, two-way automata...)
- More broadly, following the **intensional approach** for enumeration: classify enumeration tasks depending on the **circuit class** to which they can be compiled?

Questions for future work

- What about **negation gates**?
- What can we do without **determinism**? (enumeration for DNNF?)
- Connect results on **updates** to finer bounds on **incremental maintenance** (A., Jachiet, Paperman, ICALP'21)
- Enumerate satisfying assignments via **edits on previous results** (A., Monet, STACS'23) to achieve **constant delay** even on **linear-sized assignments**
- For MSO queries: understand better the connection between **automata classes** and **circuit classes** (e.g., alternating automata, two-way automata...)
- More broadly, following the **intensional approach** for enumeration: classify enumeration tasks depending on the **circuit class** to which they can be compiled?

Thanks for your attention!

Amarilli, A., Bourhis, P., Jachiet, L., and Mengel, S. (2017).
A circuit-based approach to efficient enumeration.
In *ICALP*.

Amarilli, A., Bourhis, P., and Mengel, S. (2018).
Enumeration on trees under relabelings.
In *ICDT*.

Amarilli, A., Bourhis, P., Mengel, S., and Niewerth, M. (2019a).
Constant-delay enumeration for nondeterministic document spanners.
In *ICDT*.

References ii

Amarilli, A., Bourhis, P., Mengel, S., and Niewerth, M. (2019b).
Enumeration on trees with tractable combined complexity and efficient updates.
In *PODS*.

Amarilli, A., Bourhis, P., and Senellart, P. (2015).
Provenance circuits for trees and treelike instances.
In *ICALP*.

Amarilli, A., Jachiet, L., Muñoz, M., and Riveros, C. (2022).
Efficient enumeration for annotated grammars.
In *PODS*.

Amarilli, A., Jachiet, L., and Paperman, C. (2021).
Dynamic membership for regular languages.
In *ICALP*.

Amarilli, A. and Monet, M. (2023).
Enumerating regular languages with bounded delay.
In *STACS*.

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

Berkholz, C. and Vinall-Smeeth, H. (2023).
A dichotomy for succinct representations of homomorphisms.
In *ICALP*.

Bourhis, P., Grez, A., Jachiet, L., and Riveros, C. (2021).
Ranked enumeration of MSO logic on words.
In *ICDT*.

Carmeli, N., Tziavelis, N., Gatterbauer, W., Kimelfeld, B., and Riedewald, M. (2023).
Tractable orders for direct access to ranked answers of conjunctive queries.
TODS, 48(1).

- Deep, S., Hu, X., and Koutris, P. (2022).
  Ranked enumeration of join queries with projections.
  PVLDB, 15(5).
- Florenzano, F., Riveros, C., Ugarte, M., Vansummeren, S., and Vrgoc, D. (2018).
  Constant delay algorithms for regular document spanners.
  In *PODS*.
- Kazana, W. and Segoufin, L. (2013).
  Enumeration of monadic second-order queries on trees.
  TOCL, 14(4).

Muñoz, M. and Riveros, C. (2022).
Streaming enumeration on nested documents.
In *ICDT*.

Muñoz, M. and Riveros, C. (2023).
Constant-delay enumeration for SLP-compressed documents.
In *ICDT*.

Niewerth, M. (2018).
MSO queries on trees: Enumerating answers under updates using forest algebras.
In *LICS*.

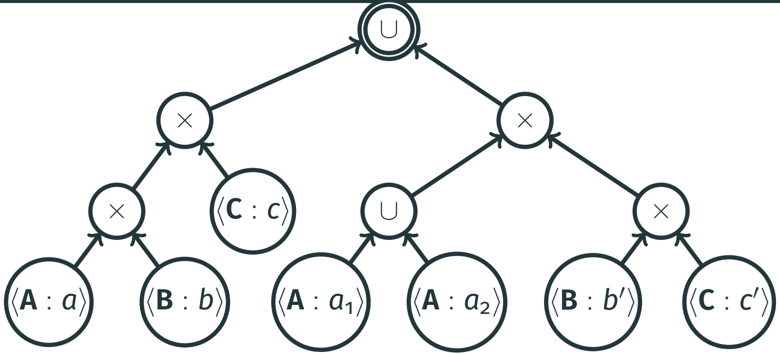
Peterfreund, L. (2021).
Grammars for document spanners.
In *ICDT*.

Toruńczyk, S. (2020).
Aggregate queries on sparse databases.
In *PODS*.

Tziavelis, N., Gatterbauer, W., and Riedewald, M. (2022).
Any-k algorithms for enumerating ranked answers to conjunctive queries.
arXiv preprint arXiv:2205.05649.

Set circuits vs factorized representations

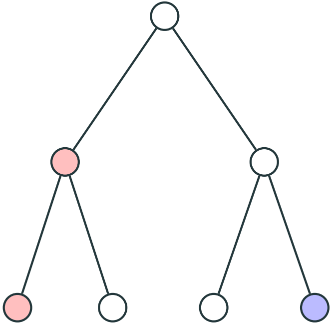
A	B	C
a	b	c
a_1	b'	c'
a_2	b'	c'



- Set circuits can be seen as **factorized representations**
  - → Not necessarily **well-typed**, height and/or assignment size may be **non-constant**
- **Determinism**: unions are disjoint
- **Decomposability**: no duplicate attribute names in products
- **Structuredness**: always the same decomposition of the attributes

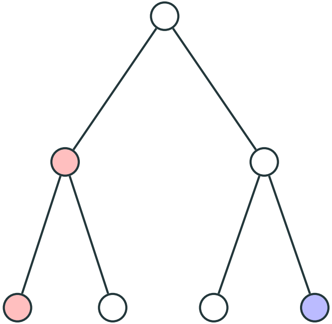
Tree automata

Tree alphabet: 



Tree automata

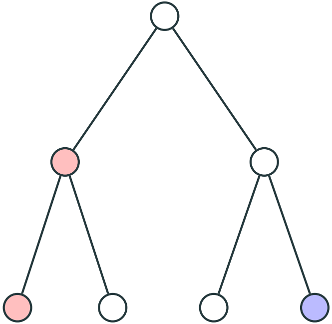
Tree alphabet: 



- Bottom-up deterministic **tree automaton**
- *“Is there both a pink and a blue node?”*

Tree automata

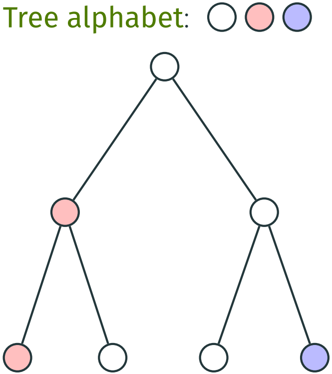
Tree alphabet:



- Bottom-up deterministic **tree automaton**
- *"Is there both a pink and a blue node?"*
- **States:** $\{\bot, B, P, \top\}$

Tree automata

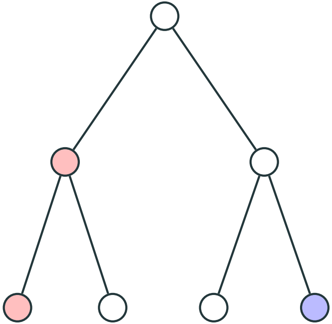
Tree alphabet:



- Bottom-up deterministic **tree automaton**
- *"Is there both a pink and a blue node?"*
- **States:** $\{\bot, B, P, \top\}$
- **Final states:** $\{\top\}$

Tree automata

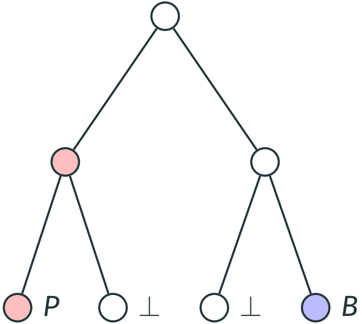
Tree alphabet:



- Bottom-up deterministic **tree automaton**
- *"Is there both a pink and a blue node?"*
- **States:** $\{\bot, B, P, \top\}$
- **Final states:** $\{\top\}$
- **Initial function:** $\bot$ P B

Tree automata

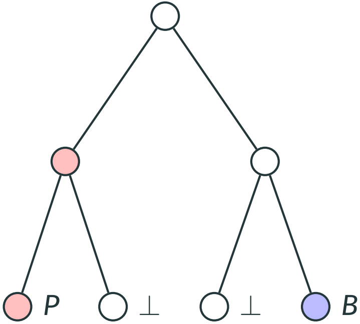
Tree alphabet:



- Bottom-up deterministic **tree automaton**
- *"Is there both a pink and a blue node?"*
- States: $\{\bot, B, P, \top\}$
- Final states: $\{\top\}$
- Initial function: ○ $\bot$ ● P ● B

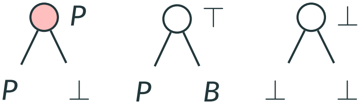
Tree automata

Tree alphabet: 



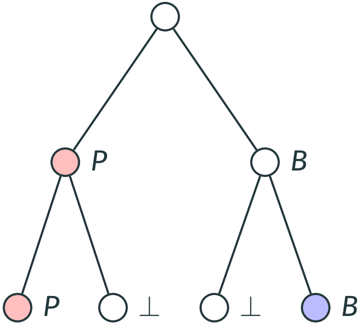
- Bottom-up deterministic **tree automaton**
- *"Is there both a pink and a blue node?"*
- **States:** $\{\bot, B, P, \top\}$
- **Final states:** $\{\top\}$
- **Initial function:** 

- **Transitions** (examples):



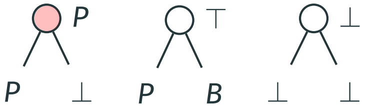
Tree automata

Tree alphabet: 



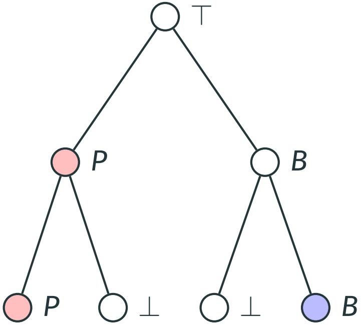
- Bottom-up deterministic **tree automaton**
- *"Is there both a pink and a blue node?"*
- **States:** $\{\bot, B, P, \top\}$
- **Final states:** $\{\top\}$
- **Initial function:** 

- **Transitions** (examples):



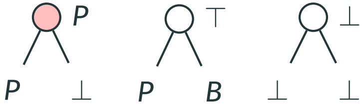
Tree automata

Tree alphabet: 



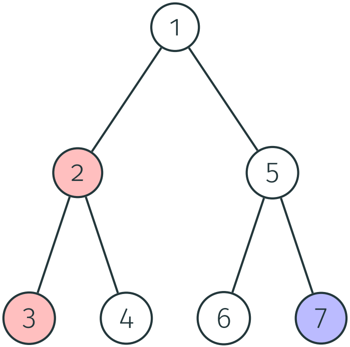
- Bottom-up deterministic **tree automaton**
- *"Is there both a pink and a blue node?"*
- **States:** $\{\bot, B, P, \top\}$
- **Final states:** $\{\top\}$
- **Initial function:** 

- **Transitions** (examples):



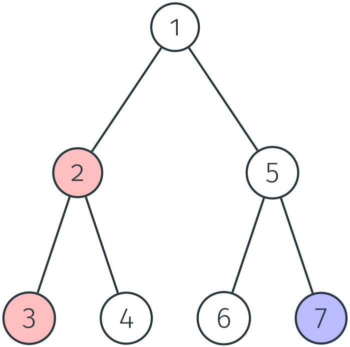
Uncertain trees

Now: Boolean query on a tree where the color of nodes is uncertain



Uncertain trees

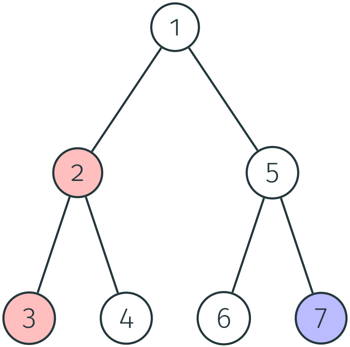
Now: Boolean query on a tree where the color of nodes is **uncertain**



A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

Uncertain trees

Now: Boolean query on a tree where the color of nodes is **uncertain**

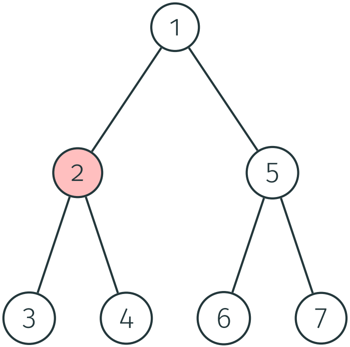


A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

Valuation: $\{2, 3, 7 \mapsto 1, \quad * \mapsto 0\}$

Uncertain trees

Now: Boolean query on a tree where the color of nodes is **uncertain**

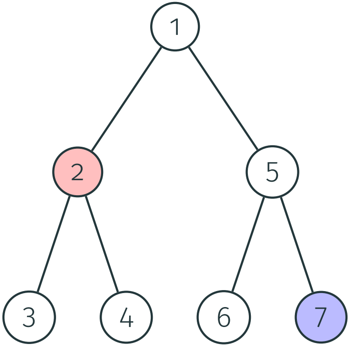


A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

Valuation: $\{2 \mapsto 1, \ * \mapsto 0\}$

Uncertain trees

Now: Boolean query on a tree where the color of nodes is **uncertain**

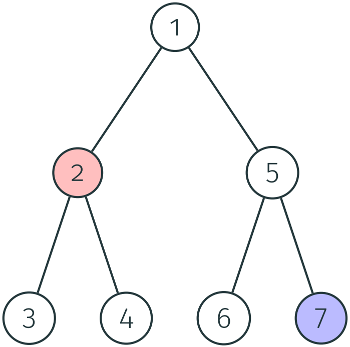


A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

Valuation: $\{2, 7 \mapsto 1, \quad * \mapsto 0\}$

Uncertain trees

Now: Boolean query on a tree where the color of nodes is **uncertain**



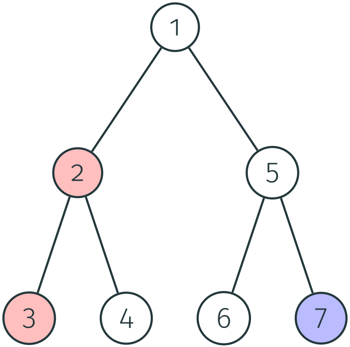
A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

Valuation: $\{\mathbf{2}, \mathbf{7} \mapsto \mathbf{1}, \ * \mapsto \mathbf{0}\}$

A: *"Is there both a pink and a blue node?"*

Uncertain trees

Now: Boolean query on a tree where the color of nodes is **uncertain**



A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

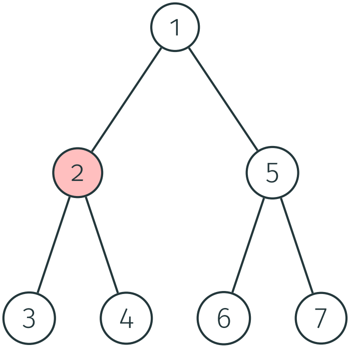
Valuation: $\{2, 3, 7 \mapsto 1, \quad * \mapsto 0\}$

A: *"Is there both a pink and a blue node?"*

The tree automaton A **accepts**

Uncertain trees

Now: Boolean query on a tree where the color of nodes is **uncertain**



A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

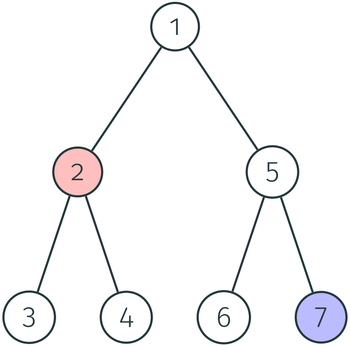
Valuation: $\{\mathbf{2} \mapsto \mathbf{1},\ \ * \mapsto \mathbf{0}\}$

A: *"Is there both a pink and a blue node?"*

The tree automaton A **rejects**

Uncertain trees

Now: Boolean query on a tree where the color of nodes is **uncertain**



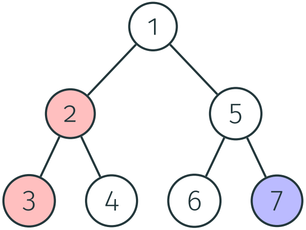
A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

Valuation: $\{2, 7 \mapsto 1, \; * \mapsto 0\}$

A: *"Is there both a pink and a blue node?"*

The tree automaton A **accepts**

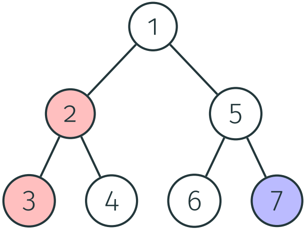
Set circuit



Set circuit:

- Tree automaton A, uncertain tree T, circuit C
- Variable gates of C: nodes of T

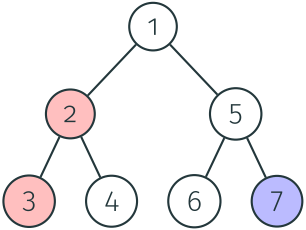
Set circuit



Set circuit:

- Tree automaton A, uncertain tree T, circuit C
- **Variable gates** of C: nodes of T
- **Condition:** Let ν be a valuation of T, then A accepts $\nu(T)$ iff the set $\mathrm{S}(g_0)$ of the output gate g_0 contains $\{g \in C \mid \nu(g) = 1\}$.

Set circuit

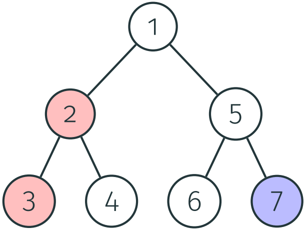


Set circuit:

- Tree automaton A, uncertain tree T, circuit C
- **Variable gates** of C: nodes of T
- **Condition:** Let ν be a valuation of T, then A accepts $\nu(T)$ iff the set $\mathrm{S}(g_0)$ of the output gate g_0 contains $\{g \in C \mid \nu(g) = 1\}$.

Query: *Is there both a pink and a blue node?*

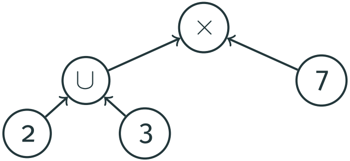
Set circuit



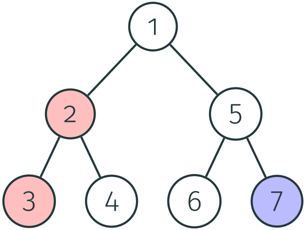
Set circuit:

- Tree automaton A, uncertain tree T, circuit C
- **Variable gates** of C: nodes of T
- **Condition:** Let ν be a valuation of T, then A accepts $\nu(T)$ iff the set $S(g_0)$ of the output gate g_0 contains $\{g \in C \mid \nu(g) = 1\}$.

Query: *Is there both a pink and a blue node?*



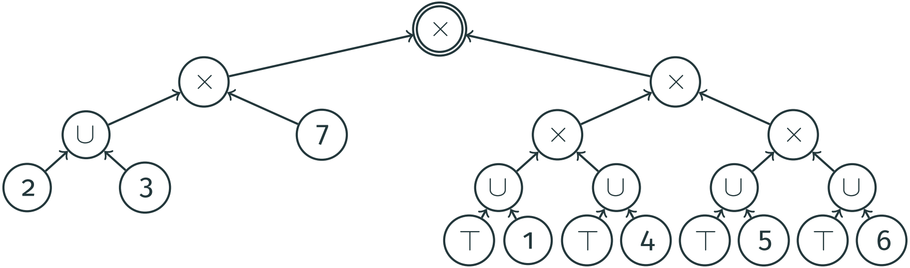
Set circuit



Set circuit:

- Tree automaton A, uncertain tree T, circuit C
- **Variable gates** of C: nodes of T
- **Condition:** Let ν be a valuation of T, then A accepts $\nu(T)$ iff the set $\mathrm{S}(g_0)$ of the output gate g_0 contains $\{g \in C \mid \nu(g) = 1\}$.

Query: *Is there both a pink and a blue node?*



Building provenance circuits on trees

Theorem

*For any bottom-up deterministic **tree automaton** A and input **tree** T, we can build a **d-DNNF set circuit** of A on T in $O(|A| \times |T|)$*

Building provenance circuits on trees

Theorem

For any bottom-up deterministic ***tree automaton*** *A and input* ***tree*** *T,*
we can build a ***d-DNNF set circuit*** *of A on T in $O(|A| \times |T|)$*

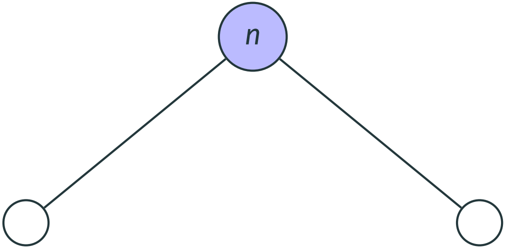
- **Alphabet:** 
- **Automaton:** *"Is there both a pink and a blue node?"*
- **States:** $\{\bot, B, P, \top\}$
- **Final:** $\{\top\}$
- **Transitions:**

Building provenance circuits on trees

Theorem

*For any bottom-up deterministic **tree automaton** A and input **tree** T, we can build a **d-DNNF set circuit** of A on T in $O(|A| \times |T|)$*

- **Alphabet:** 
- **Automaton:** *"Is there both a pink and a blue node?"*
- **States:** $\{\bot, B, P, \top\}$
- **Final:** $\{\top\}$
- **Transitions:**

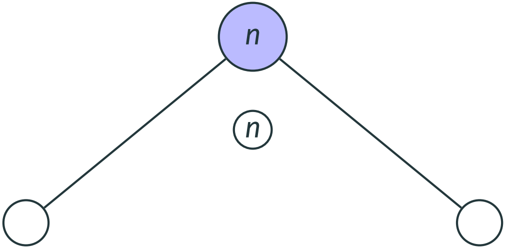


Building provenance circuits on trees

Theorem

For any bottom-up deterministic ***tree automaton*** A *and input* ***tree*** T*,*
we can build a ***d-DNNF set circuit*** *of* A *on* T *in* $O(|A| \times |T|)$

- **Alphabet:** 
- **Automaton:** *"Is there both a pink and a blue node?"*
- **States:** $\{\bot, B, P, \top\}$
- **Final:** $\{\top\}$
- **Transitions:**

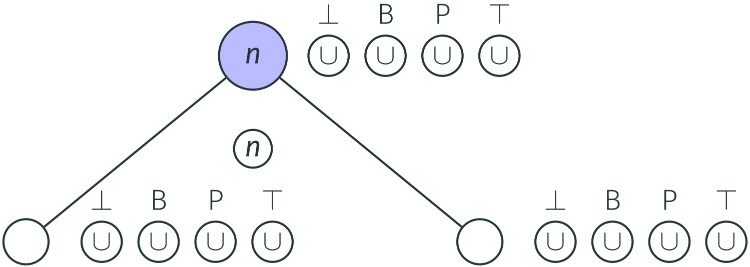


Building provenance circuits on trees

Theorem

For any bottom-up deterministic ***tree automaton*** A *and input* ***tree*** T*,*
we can build a ***d-DNNF set circuit*** *of* A *on* T *in* $O(|A| \times |T|)$

- **Alphabet:** 
- **Automaton:** *"Is there both a pink and a blue node?"*
- **States:** $\{\bot, B, P, \top\}$
- **Final:** $\{\top\}$
- **Transitions:**

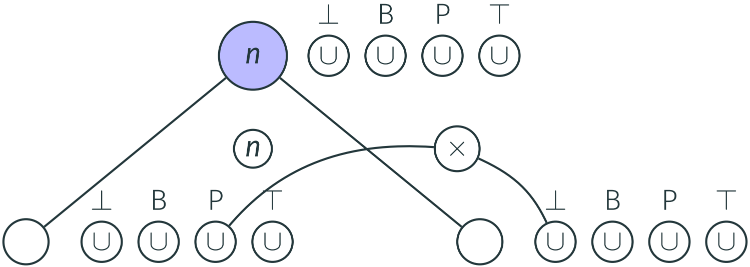


Building provenance circuits on trees

Theorem

For any bottom-up deterministic ***tree automaton*** A *and input* ***tree*** T*, we can build a* ***d-DNNF set circuit*** *of* A *on* T *in* $O(|A| \times |T|)$

- **Alphabet:** 
- **Automaton:** *"Is there both a pink and a blue node?"*
- **States:** $\{\bot, B, P, \top\}$
- **Final:** $\{\top\}$
- **Transitions:**

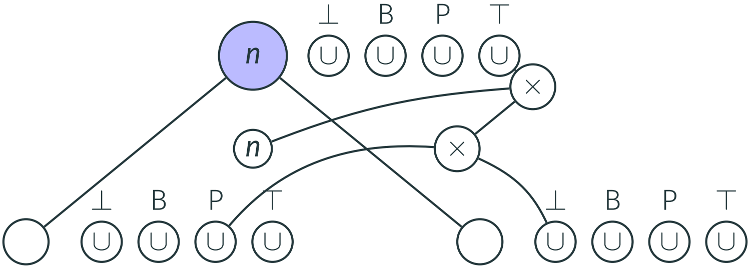


Building provenance circuits on trees

Theorem

*For any bottom-up deterministic **tree automaton** A and input **tree** T, we can build a **d-DNNF set circuit** of A on T in* $O(|A| \times |T|)$

- **Alphabet:** 
- **Automaton:** *"Is there both a pink and a blue node?"*
- **States:** $\{\bot, B, P, \top\}$
- **Final:** $\{\top\}$
- **Transitions:**

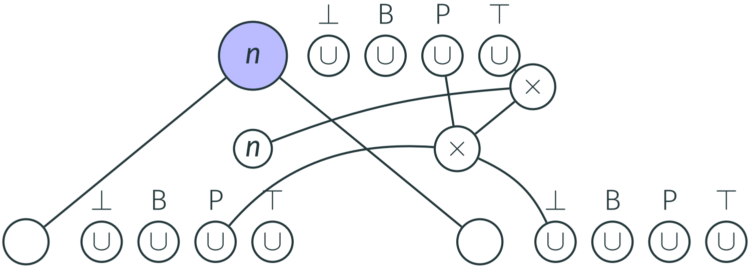


Building provenance circuits on trees

Theorem

*For any bottom-up deterministic **tree automaton** A and input **tree** T, we can build a **d-DNNF set circuit** of A on T in $O(|A| \times |T|)$*

- **Alphabet:** 
- **Automaton:** *“Is there both a pink and a blue node?”*
- **States:** $\{\bot, B, P, \top\}$
- **Final:** $\{\top\}$
- **Transitions:**



Circuits as factorized representations of query results

→ The set circuit of Q is now a factorized representation which describes all the tuples that make Q true

Circuits as factorized representations of query results

→ The **set circuit** of Q is now a **factorized representation** which describes all the tuples that make Q true

Example query:

$Q(X_1, X_2) : P_{\circ}(x) \land P_{\circ}(y)$

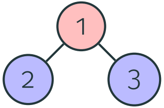
Circuits as factorized representations of query results

→ The **set circuit** of Q is now a **factorized representation** which describes all the tuples that make Q true

Example query:

$Q(X_1, X_2) : P_{\color{pink}\bullet}(x) \wedge P_{\color{blue}\bullet}(y)$

Data:



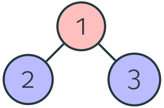
Circuits as factorized representations of query results

→ The **set circuit** of Q is now a **factorized representation** which describes all the tuples that make Q true

Example query:

$Q(X_1, X_2) : P_{\bigcirc}(x) \land P_{\bigcirc}(y)$

Data:



Results:

X_1	X_2
1	2
1	3

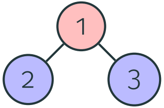
Circuits as factorized representations of query results

→ The set circuit of Q is now a factorized representation which describes all the tuples that make Q true

Example query:

$Q(X_1, X_2) : P_{\circ}(x) \wedge P_{\circ}(y)$

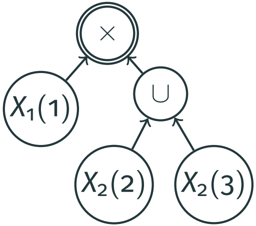
Data:



Results:

X_1	X_2
1	2
1	3

Provenance circuit:



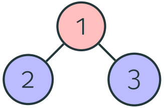
Circuits as factorized representations of query results

→ The set circuit of Q is now a factorized representation which describes all the tuples that make Q true

Example query:

$Q(X_1, X_2) : P_{\circ}(x) \land P_{\circ}(y)$

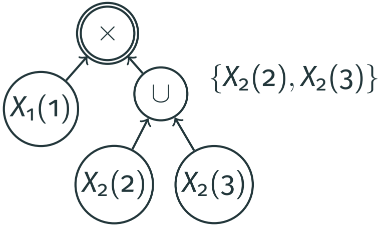
Data:



Results:

X_1	X_2
1	2
1	3

Provenance circuit:



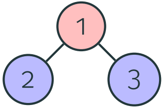
Circuits as factorized representations of query results

→ The **set circuit** of Q is now a **factorized representation** which describes all the tuples that make Q true

Example query:

$Q(X_1, X_2) : P_{\bigcirc}(x) \wedge P_{\bigcirc}(y)$

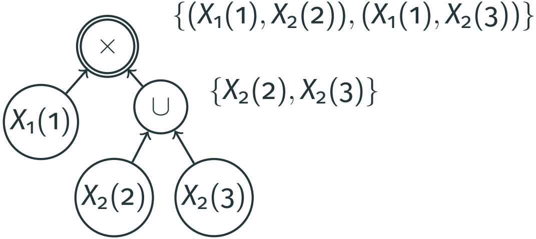
Data:



Results:

X_1	X_2
1	2
1	3

Provenance circuit:



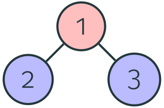
Circuits as factorized representations of query results

→ The **set circuit** of Q is now a **factorized representation** which describes all the tuples that make Q true

Example query:

$Q(X_1, X_2) : P_{\bigcirc}(x) \wedge P_{\bigcirc}(y)$

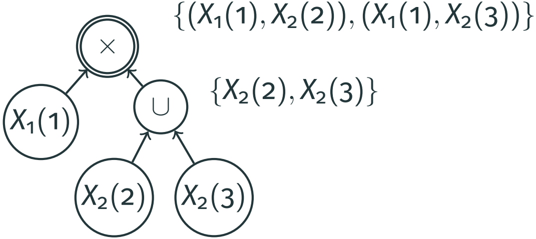
Data:



Results:

X_1	X_2
1	2
1	3

Provenance circuit:



Theorem [Bagan, 2006, Kazana and Segoufin, 2013]

We can enumerate the answers of MSO queries on trees with linear-time preprocessing and constant delay.

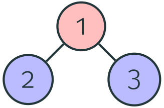
Circuits as factorized representations of query results

→ The **set circuit** of Q is now a **factorized representation** which describes all the tuples that make Q true

Example query:

$Q(X_1, X_2) : P_{\circ}(x) \wedge P_{\circ}(y)$

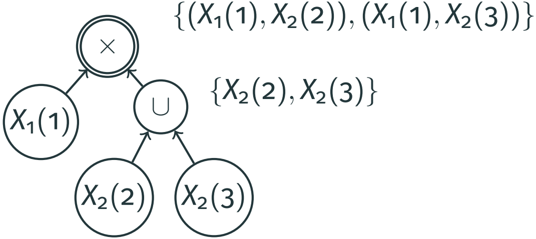
Data:



Results:

X_1	X_2
1	2
1	3

Provenance circuit:



Theorem [Bagan, 2006, Kazana and Segoufin, 2013]

We can enumerate the answers of MSO queries on trees with linear-time preprocessing and constant delay.

Semi-open question: what about memory usage?