Efficient Enumeration via Factorized Representations

Antoine Amarilli

August 2nd, 2022

Télécom Paris

Dramatis Personae

Antoine Amarilli

Pierre Bourhis

Louis Jachiet

Stefan Mengel

Matthias Niewerth

And our recent co-authors: Martín Muñoz, 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.

Niewerth, M.
MSO Queries on Trees: Enumerating Answers under Updates Using Forest Algebras. LICS 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

Problem statement

Enumeration algorithm

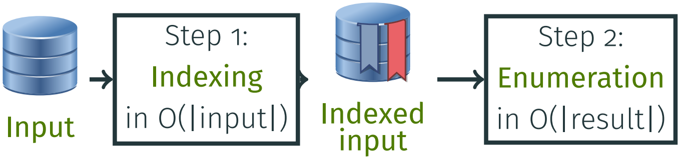
Input

Enumeration algorithm

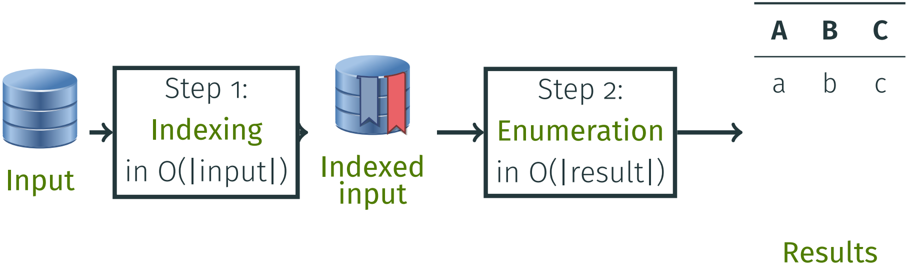
Input → Step 1: **Indexing** in O(|input|)

Enumeration algorithm

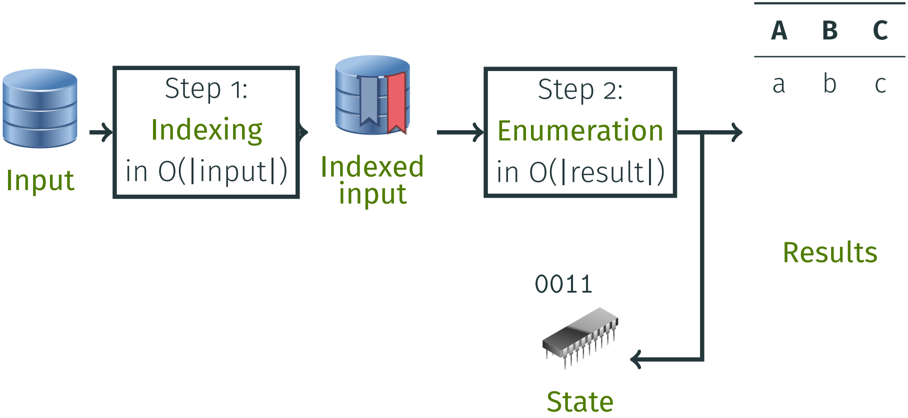
Enumeration algorithm



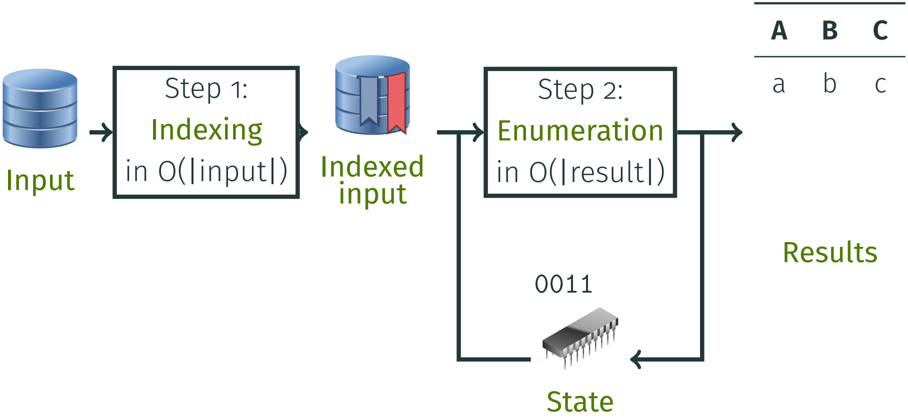
Enumeration algorithm



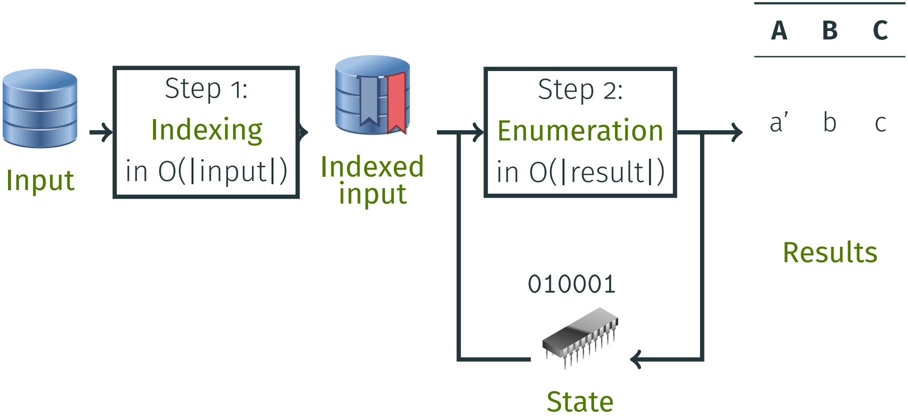
Enumeration algorithm



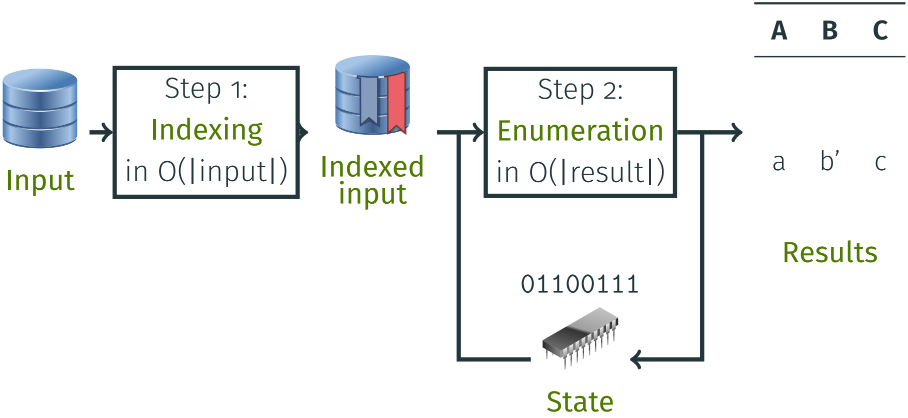
Enumeration algorithm



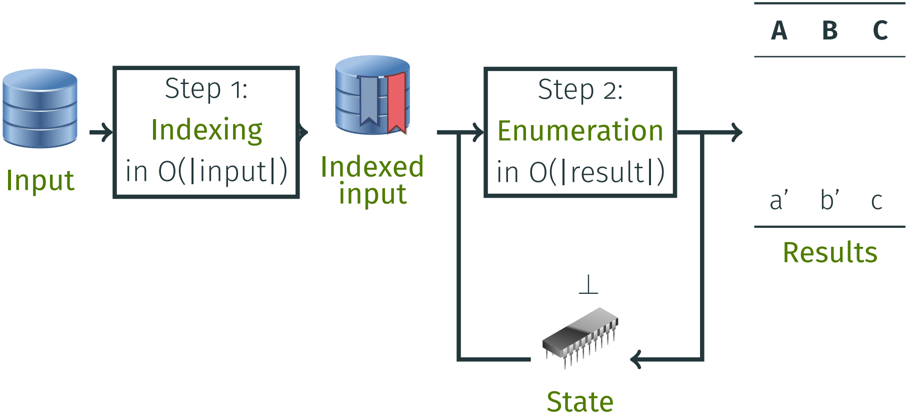
Enumeration algorithm



Enumeration algorithm

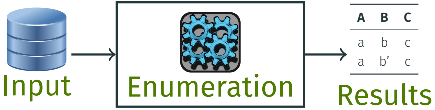


Enumeration algorithm



The knowledge compilation approach to enumeration

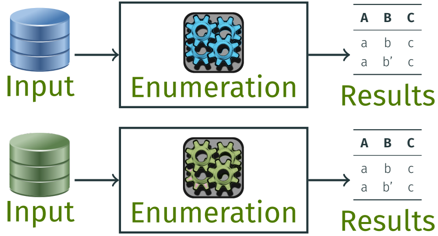
Currently:



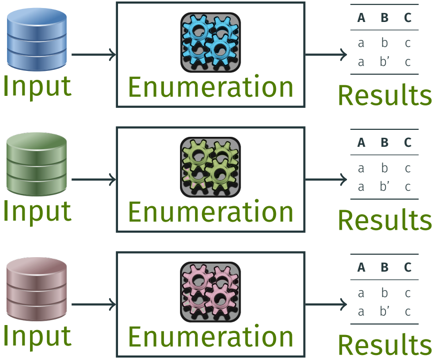
A	B	C
a	b	c
a	b’	c

The knowledge compilation approach to enumeration

Currently:

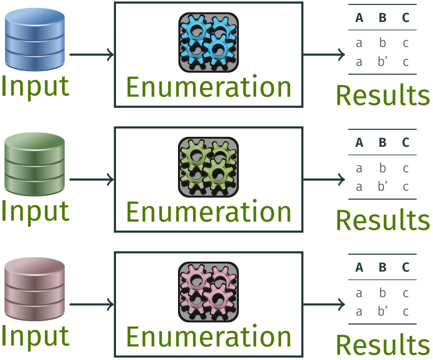


Currently:


Input
Enumeration
A B C
a b c
a b’ c
Results
Input
Enumeration
A B C
a b c
a b’ c
Results
Input
Enumeration
A B C
a b c
a b’ c
Results

The knowledge compilation approach to enumeration

Currently:



A	B	C
a	b	c
a	b’	c

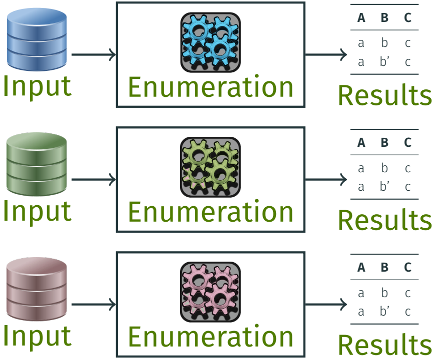
A	B	C
a	b	c
a	b’	c

A	B	C
a	b	c
a	b’	c

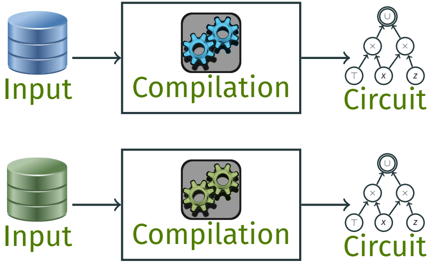
Our idea:

The knowledge compilation approach to enumeration

Currently:

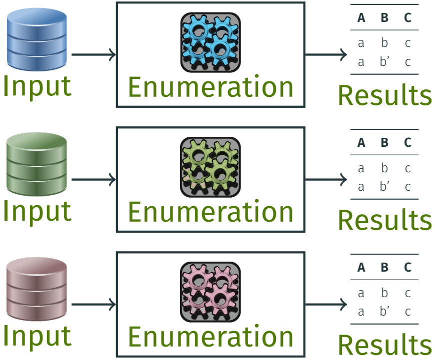


Our idea:

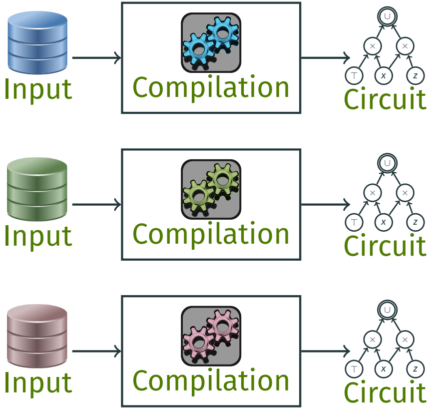


The knowledge compilation approach to enumeration

Currently:

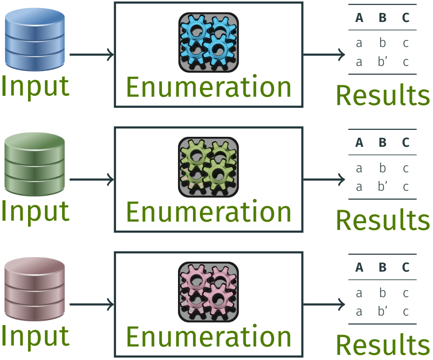


Our idea:

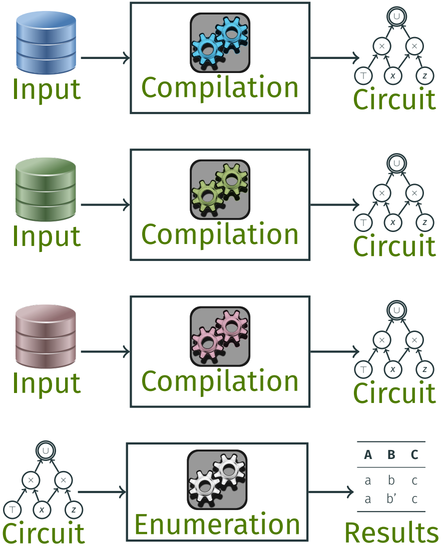


The knowledge compilation approach to enumeration

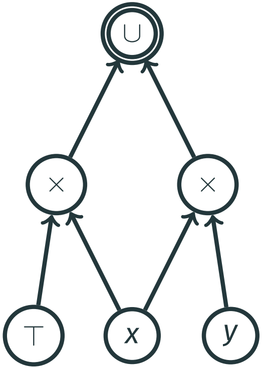
Currently:



Our idea:

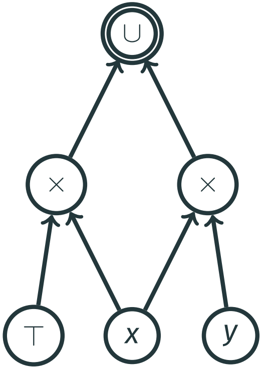


Set circuits

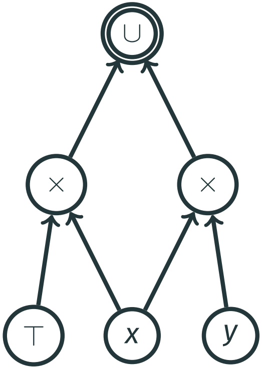


- Directed acyclic graph of **gates**

Set circuits

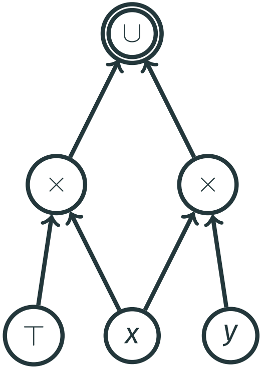


- Directed acyclic graph of **gates**
- **Output** gate:



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

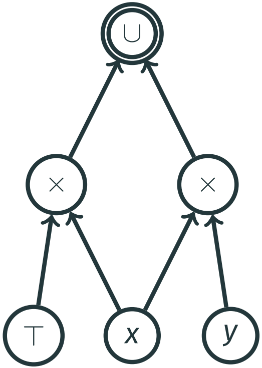
Set circuits



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

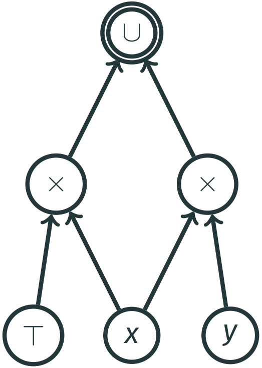
- **Constant** gates: 

Set circuits



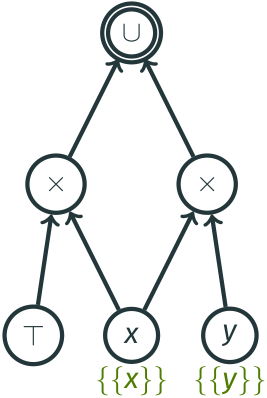
- Directed acyclic graph of **gates**
- **Output** gate:
- **Variable** gates: x
- **Constant** gates: $\top$ $\bot$
- **Internal** gates: $\times$ $\cup$

Semantics of set circuits



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

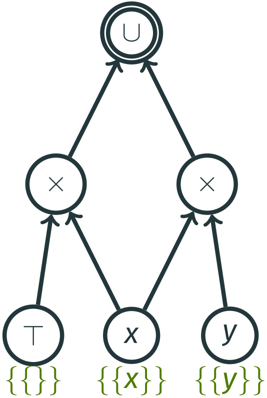
Semantics of set circuits



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

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

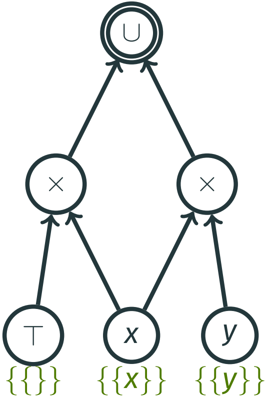
Semantics of set circuits



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

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

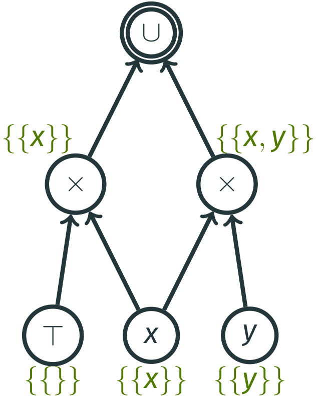
Semantics of set circuits



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: $S(g) = \emptyset$

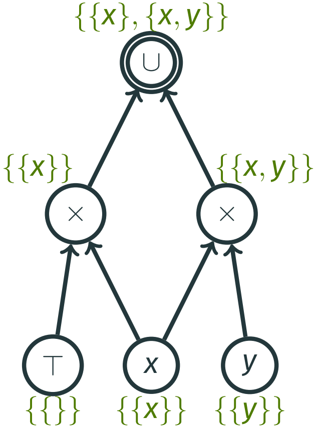
Semantics of set circuits



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

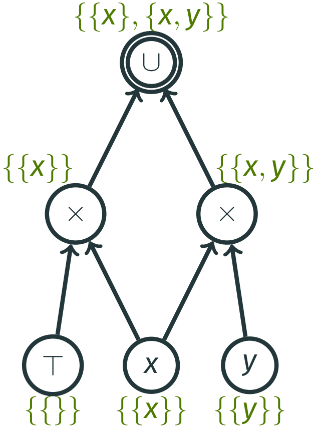
Semantics of set circuits



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

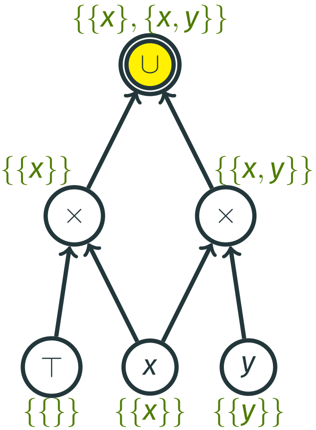
Task: Enumerate the assignments of the set $S(g)$ captured by a gate g
$\rightarrow$E.g., for $S(g) = \{\{x\},\{x,y\}\}$, enumerate $\{x\}$ and then $\{x,y\}$

d-DNNF set circuit:

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

The inputs are **disjoint**

(= no assignment is captured by two inputs)



d-DNNF set circuit:

- 

are all **deterministic**:

The inputs are **disjoint**

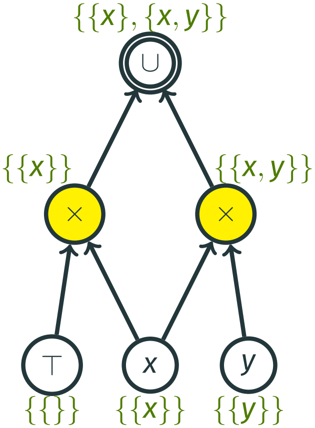
(= no assignment is captured by two inputs)

- 

are all **decomposable**:

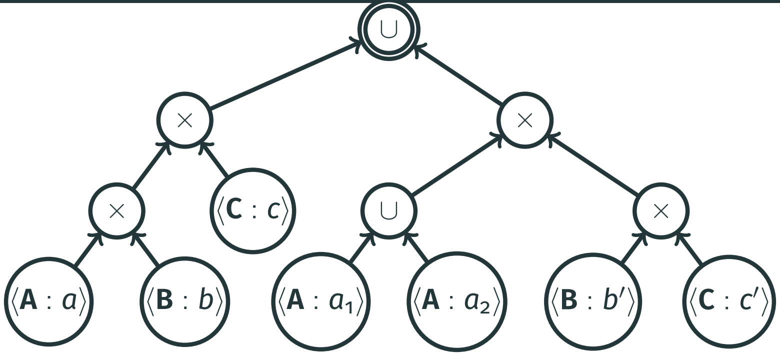
The inputs are **independent**

(= no variable x has a path to two different inputs)



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**
  - $\rightarrow$ 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

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** $k \in \mathbb{N}$

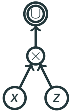
Theorem

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 techniques

Proof overview

Preprocessing phase:

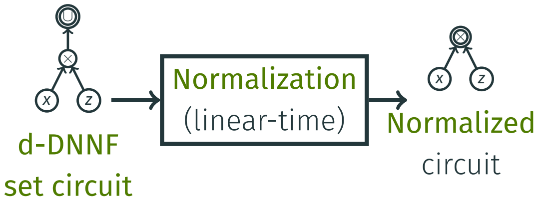


d-DNNF

set circuit

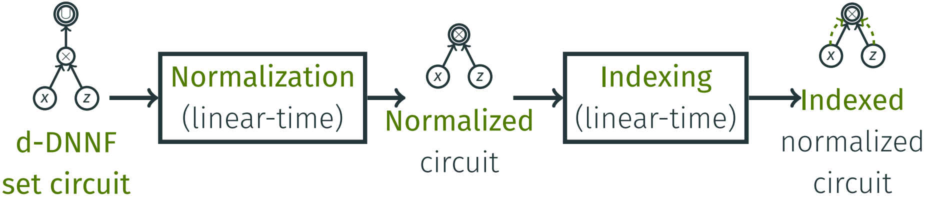
Proof overview

Preprocessing phase:



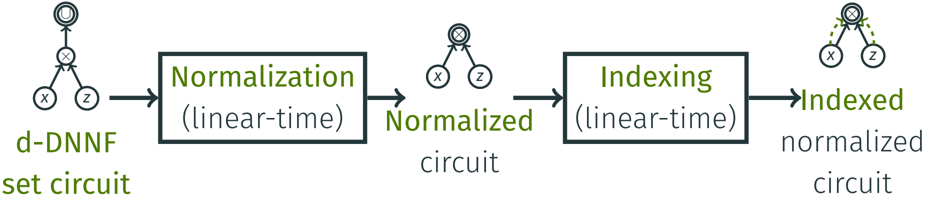
Proof overview

Preprocessing phase:

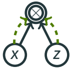


Proof overview

Preprocessing phase:



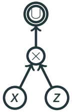
Enumeration phase:



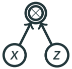
Indexed
normalized
circuit

Proof overview

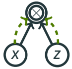
Preprocessing phase:



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

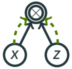


Normalized circuit → **Indexing** (linear-time) →



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

$\rightarrow$ 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

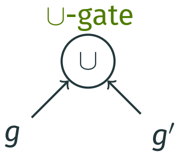
$\rightarrow$ E.g., for $S(g) = \{\{x\}, \{x, y\}\}$, enumerate $\{x\}$ and then $\{x, y\}$

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

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



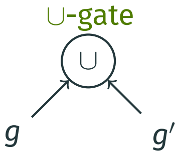
Concatenation: enumerate $S(g)$ and then enumerate $S(g')$

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



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

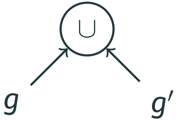
Determinism: no duplicates

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

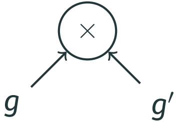
$\cup$-gate



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

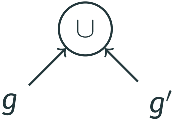
Enumerating captured assignments of d-DNNF set circuits

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

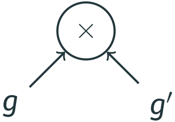
$\cup$-gate



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

Determinism: no duplicates

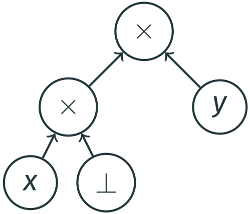
$\times$-gate



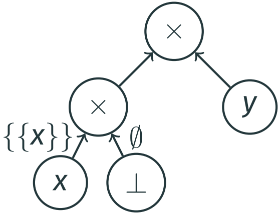
Lexicographic product: enumerate $S(g)$ and for each result t enumerate $S(g')$ and concatenate t with each result

Decomposability: no duplicates

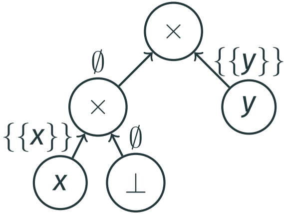
Normalization: handling $\emptyset$



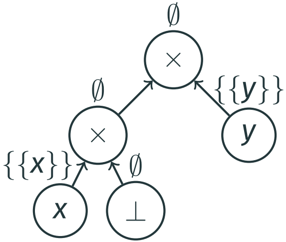
Normalization: handling $\emptyset$



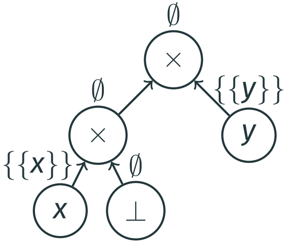
Normalization: handling $\emptyset$



Normalization: handling $\emptyset$

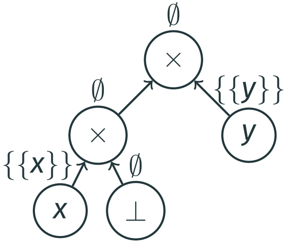


Normalization: handling $\emptyset$



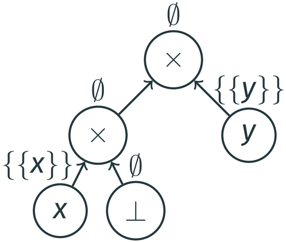
- **Problem:** if $S(g) = \emptyset$ we waste time

Normalization: handling $\emptyset$



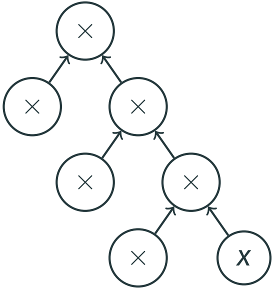
- **Problem:** if $S(g) = \emptyset$ we waste time
- **Solution:** in preprocessing
  - compute **bottom-up** if $S(g) = \emptyset$

Normalization: handling $\emptyset$

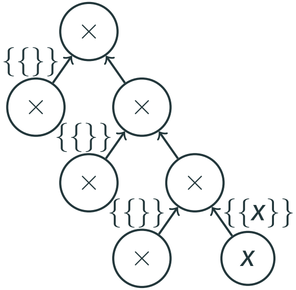


- **Problem:** if $S(g) = \emptyset$ we waste time
- **Solution:** in preprocessing
  - compute **bottom-up** if $S(g) = \emptyset$
  - then get rid of the gate

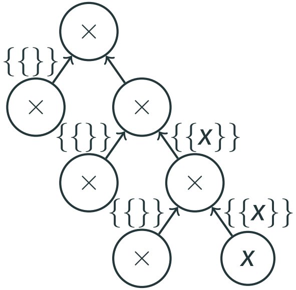
Normalization: handling empty assignments



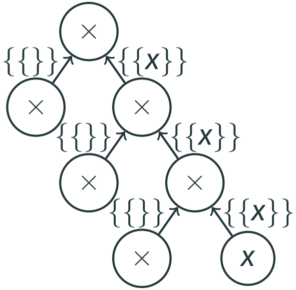
Normalization: handling empty assignments



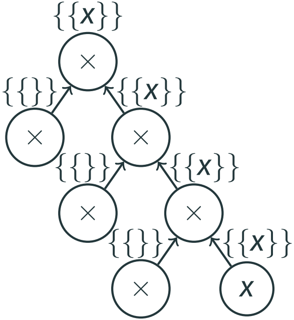
Normalization: handling empty assignments



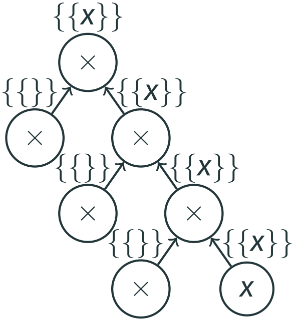
Normalization: handling empty assignments



Normalization: handling empty assignments

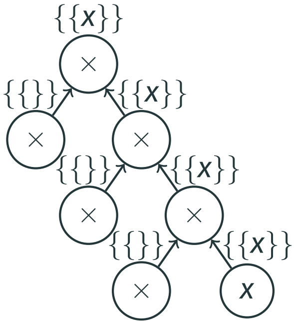


Normalization: handling empty assignments



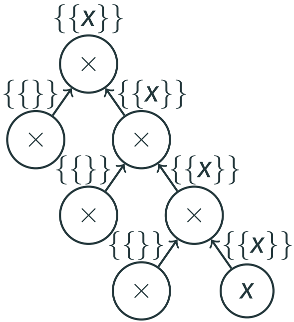
- **Problem:** if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates

Normalization: handling empty assignments



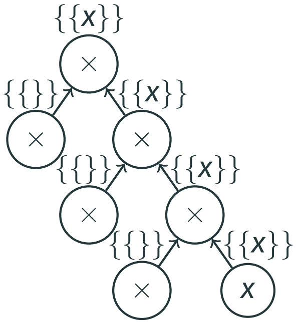
- **Problem:** if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- **Solution:**

Normalization: handling empty assignments



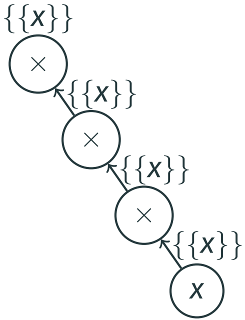
- **Problem:** if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- **Solution:**
  - split g between $S(g) \cap \{\{\}\}$ and $S(g) \setminus \{\{\}\}$ (homogenization)

Normalization: handling empty assignments



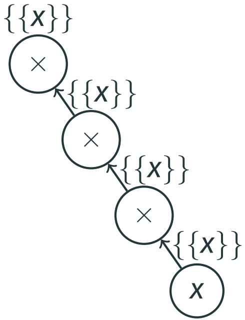
- **Problem:** if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- **Solution:**
  - **split** g between $S(g) \cap \{\{\}\}$ and $S(g) \setminus \{\{\}\}$ (homogenization)
  - **remove** inputs with $S(g) = \{\{\}\}$ for $\times$-gates

Normalization: handling empty assignments



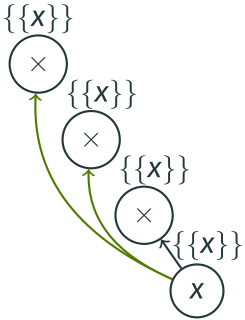
- **Problem:** if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- **Solution:**
  - **split** g between $S(g) \cap \{\{\}\}$ and $S(g) \setminus \{\{\}\}$ (homogenization)
  - **remove** inputs with $S(g) = \{\{\}\}$ for $\times$-gates

Normalization: handling empty assignments



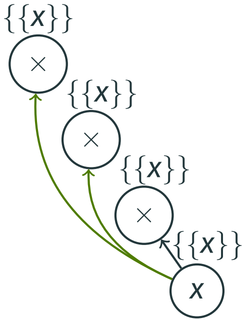
- **Problem:** if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- **Solution:**
  - **split** g between $S(g) \cap \{\{\}\}$ and $S(g) \setminus \{\{\}\}$ (homogenization)
  - **remove** inputs with $S(g) = \{\{\}\}$ for $\times$-gates
  - **collapse** $\times$-chains with fan-in 1

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

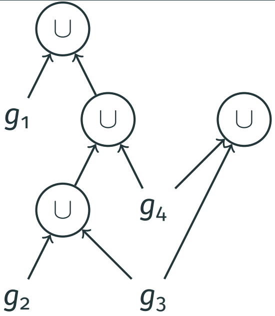
Normalization: handling empty assignments



- **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

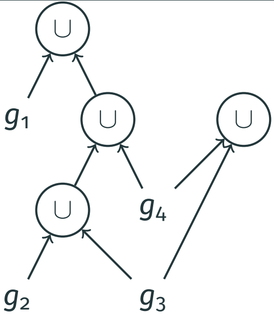
$\rightarrow$ Now, when traversing a $\times$**-gate** we make progress: **non-trivial split** of each set

Indexing: handling ∪-hierarchies



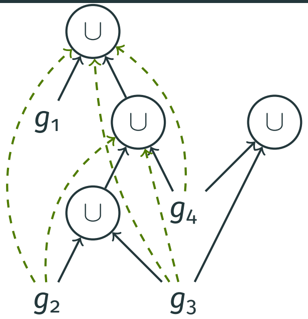
- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)

Indexing: handling ∪-hierarchies



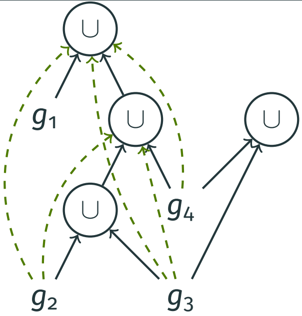
- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)
- **Solution:** compute **reachability index**

Indexing: handling ∪-hierarchies



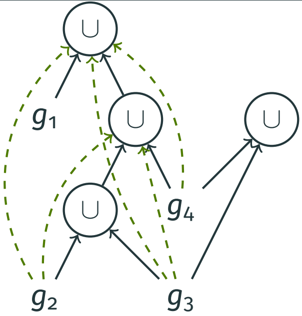
- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)
- **Solution:** compute **reachability index**

Indexing: handling $\cup$-hierarchies



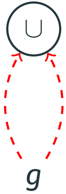
- **Problem:** we waste time in $\cup$-hierarchies to find a **reachable exit** (non-$\cup$ gate)
- **Solution:** compute **reachability index**
- **Problem:** must be done in **linear time**

Indexing: handling ∪-hierarchies

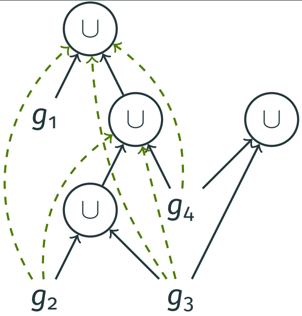


- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)
- **Solution:** compute **reachability index**
- **Problem:** must be done in **linear time**

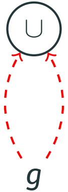
- **Solution:** Determinism ensures we have a **multitree** (we cannot have the pattern at the right)



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**

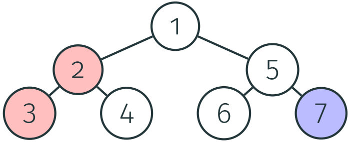


- **Solution:** Determinism ensures we have a **multitree** (we cannot have the pattern at the right)
- **Custom** constant-delay reachability index for multitrees

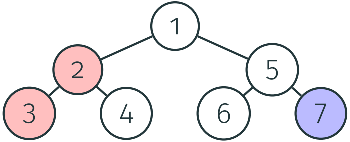
Applications

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)

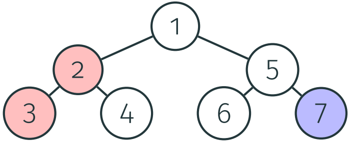
- $P_{\bigcirc}(x)$ means "x is blue"
- $x \rightarrow y$ means "x is the parent of y"

"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 $\bigcirc$ $\textcolor{pink}{\bullet}$ $\textcolor{blue}{\bullet}$



Query Q in monadic second-order logic (MSO)

- $P_{\textcolor{blue}{\bullet}}(x)$ 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?"

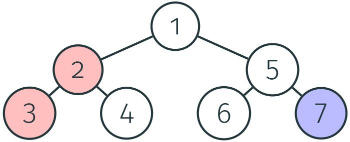
$Q(x, y) := P_{\textcolor{pink}{\bullet}}(x) \land P_{\textcolor{blue}{\bullet}}(y)$

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'15, 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'15, 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'15, 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

$P :=$ `␣ [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

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

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

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

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*

$\rightarrow$ 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 structured DNNFs**

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 structured DNNFs**

→ Actually equivalent to **MSO evaluation** on **text**; generalizes to **trees**

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**

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

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***

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 a 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**

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

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 a **quintic** preprocessing result [Peterfreund, 2021]
- **Quadratic** and **linear** preprocessing for **subclasses** (rigid grammars, deterministic pushdown annotators)

Conclusion

Summary and conclusion

- Enumerate the captured assignments of d-DNNF set circuits
  - $\rightarrow$ with preprocessing linear in the d-DNNF
  - $\rightarrow$ in delay linear in each assignment
  - $\rightarrow$ in constant delay for constant size

$\rightarrow$ Applies to MSO enumeration on words and trees

$\rightarrow$ Applies to enumeration of the matches of annotated context-free grammars (with more expensive preprocessing)

Summary and conclusion

- **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 size
- → Applies to **MSO enumeration** on **words** and **trees**
- → Applies to enumeration of the matches of **annotated context-free grammars** (with more expensive preprocessing)

Future work:

- **In-order** enumeration
- Linear-time preprocessing on **more general context-free grammar classes**
- Connect results on **updates** to **incremental maintenance for regular languages** (A., Jachiet, Paperman, ICALP'21)

Summary and conclusion

- Enumerate the captured assignments of d-DNNF set circuits
  - $\rightarrow$ with preprocessing linear in the d-DNNF
  - $\rightarrow$ in delay linear in each assignment
  - $\rightarrow$ in constant delay for constant size

$\rightarrow$ Applies to MSO enumeration on words and trees

$\rightarrow$ Applies to enumeration of the matches of annotated context-free grammars (with more expensive preprocessing)

Future work:

- In-order enumeration
- Linear-time preprocessing on more general context-free grammar classes
- Connect results on updates to incremental maintenance for regular languages (A., Jachiet, Paperman, ICALP'21)

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*.

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

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).

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

References iv

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