Enumeration on Trees with Tractable Combined Complexity and Efficient Updates

Antoine Amarilli¹, Pierre Bourhis², Stefan Mengel³, Matthias Niewerth⁴
May 20th, 2019

¹Télécom ParisTech
²CNRS, CRIStAL, Lille
³CNRS, CRIL, Lens
⁴University of Bayreuth
Dramatis Personae

Antoine Amarilli

Pierre Bourhis

Stefan Mengel

Matthias Niewerth
Problem statement
MSO query evaluation on trees

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

```
P(x) means "x is blue"

x → y means "x is the parent of y"
```

"Return all blue nodes that have a pink child"

$$\exists y \ P(x) \land P(y) \land x \rightarrow y$$

**Result:**

$$\{ (x_1, \ldots, x_k) \mid \exists y \ P(x_1) \land P(y) \land x_1 \rightarrow y \}$$

Up to $|T|$ many answers
**Data:** a tree $T$ where nodes have a color from an alphabet \( \circ \circ \circ \circ \)

**Query $Q$:** a formula in monadic second-order logic (MSO)

- $P_\circ(x)$ means “$x$ is blue”
- $x \rightarrow y$ means “$x$ is the parent of $y$”

“Return all blue nodes that have a pink child”

\[ \exists y \ P_\circ(x) \land P_\circ(y) \land x \rightarrow y \]
Data: a tree $T$ where nodes have a color from an alphabet $\bigcirc \bigcirc \bigcirc$

Query $Q$: a formula in monadic second-order logic (MSO)
- $P_\circ(x)$ means “$x$ is blue”
- $x \to y$ means “$x$ is the parent of $y$”

“Return all blue nodes that have a pink child”
$\exists y \; P_\circ(x) \land P_\circ(y) \land x \to y$

Result: $\{ (x_1, \ldots, x_k) \mid (x_1, \ldots, x_k) \models Q \}$
Data: a tree $T$ where nodes have a color from an alphabet $\circ\circ\circ$.

Query $Q$: a formula in monadic second-order logic (MSO)
- $P_\circ(x)$ means “$x$ is blue”
- $x \rightarrow y$ means “$x$ is the parent of $y$”

"Return all blue nodes that have a pink child"
\[ \exists y \; P_\circ(x) \land P_\circ(y) \land x \rightarrow y \]

Result: \( \{ (x_1, \ldots, x_k) \mid (x_1, \ldots, x_k) \models Q \} \)

Up to $|T|^k$ many answers.
Enumeration algorithm

Step /one.osf: Indexing in O(input)

Indexed input

Step /two.osf: Enumeration in O(result)

A B C

a a' a'
b b' b'
c c' c'

Results

State

/four.osf//one.osf/six.osf
Enumeration algorithm

Step 1: Indexing in $O(\text{input})$
Enumeration algorithm

Step 1: Indexing in $O(\text{input})$

Input $\rightarrow$ Indexed input
Enumeration algorithm

Input

Step 1: Indexing in $O(\text{input})$

Indexed input

Step 2: Enumeration in $O(\text{result})$
**Enumeration algorithm**

1. **Input**
   - **Step 1:** Indexing in $O(\text{input})$
   - **Indexed input**

2. **Step 2:** Enumeration in $O(\text{result})$

### Results

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
</tbody>
</table>
Enumeration algorithm

Input

Step 1: Indexing in $O(\text{input})$

Indexed input

Step 2: Enumeration in $O(\text{result})$

Results

State

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
</tbody>
</table>

0011
Enumeration algorithm

Input

Step 1: Indexing in O(input)

Indexed input

Step 2: Enumeration in O(result)

State

0011

Results

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
</tr>
</tbody>
</table>
Enumeration algorithm

1. **Input**

   - **Step 1:** Indexing in $O(\text{input})$
   - **Indexed input**

2. **Step 2:** Enumeration in $O(\text{result})$

   - **Results**
   - **State:** 010001
   - **A B C**
     - a’ b c
Enumeration algorithm

Input

Step 1: Indexing in $O(\text{input})$

Indexed input

Step 2: Enumeration in $O(\text{result})$

State

Results

01100111

A B C

a b' c
Enumeration algorithm

Input

Step 1: Indexing in $O(\text{input})$

Indexed input

Step 2: Enumeration in $O(\text{result})$

Results

State

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>a'</td>
<td>b'</td>
<td>c</td>
</tr>
</tbody>
</table>

State

⊥
**Known results on dynamic trees**

All these results are on data complexity in $T$ (for a fixed pattern):

<table>
<thead>
<tr>
<th>Work</th>
<th>Data</th>
<th>Preproc.</th>
<th>Delay</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Bagan, 2006], [Kazana and Segoufin, 2013]</td>
<td>trees</td>
<td>$O(T)$</td>
<td>$O(1)$</td>
<td>$O(T)$</td>
</tr>
</tbody>
</table>
Known results on dynamic trees

All these results are on data complexity in $T$ (for a fixed pattern):

<table>
<thead>
<tr>
<th>Work</th>
<th>Data</th>
<th>Preproc.</th>
<th>Delay</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Bagan, 2006],</td>
<td>trees</td>
<td>$O(T)$</td>
<td>$O(1)$</td>
<td>$O(T)$</td>
</tr>
<tr>
<td>[Kazana and Segoufin, 2013]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Losemann and Martens, 2014]</td>
<td>trees</td>
<td>$O(T)$</td>
<td>$O(\log^2 T)$</td>
<td>$O(\log^2 T)$</td>
</tr>
</tbody>
</table>
Known results on dynamic trees

All these results are on **data complexity** in $T$ (for a fixed pattern):

<table>
<thead>
<tr>
<th>Work</th>
<th>Data</th>
<th>Preproc.</th>
<th>Delay</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Bagan, 2006],</td>
<td>trees</td>
<td>$O(T)$</td>
<td>$O(1)$</td>
<td>$O(T)$</td>
</tr>
<tr>
<td>[Kazana and Segoufin, 2013]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Losemann and Martens, 2014]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Niewerth, 2018]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Known results on dynamic trees

All these results are on data complexity in $T$ (for a fixed pattern):

<table>
<thead>
<tr>
<th>Work</th>
<th>Data</th>
<th>Preproc.</th>
<th>Delay</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Bagan, 2006], [Kazana and Segoufin, 2013]</td>
<td>trees</td>
<td>$O(T)$</td>
<td>$O(1)$</td>
<td>$O(T)$</td>
</tr>
<tr>
<td>[Losemann and Martens, 2014]</td>
<td>trees</td>
<td>$O(T)$</td>
<td>$O(\log^2 T)$</td>
<td>$O(\log^2 T)$</td>
</tr>
<tr>
<td>[Niewerth, 2018]</td>
<td>trees</td>
<td>$O(T)$</td>
<td>$O(\log T)$</td>
<td>$O(\log T)$</td>
</tr>
<tr>
<td>[Niewerth and Segoufin, 2018]</td>
<td>text</td>
<td>$O(T)$</td>
<td>$O(1)$</td>
<td>$O(\log T)$</td>
</tr>
</tbody>
</table>
Known results on dynamic trees

All these results are on data complexity in $T$ (for a fixed pattern):

<table>
<thead>
<tr>
<th>Work</th>
<th>Data</th>
<th>Preproc.</th>
<th>Delay</th>
<th>Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Bagan, 2006],</td>
<td>trees</td>
<td>$O(T)$</td>
<td>$O(1)$</td>
<td>$O(T)$</td>
</tr>
<tr>
<td>[Kazana and Segoufin, 2013]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Losemann and Martens, 2014]</td>
<td>trees</td>
<td>$O(T)$</td>
<td>$O(\log^2 T)$</td>
<td>$O(\log^2 T)$</td>
</tr>
<tr>
<td>[Niewerth, 2018]</td>
<td>trees</td>
<td>$O(T)$</td>
<td>$O(\log T)$</td>
<td>$O(\log T)$</td>
</tr>
<tr>
<td>[Niewerth and Segoufin, 2018]</td>
<td>text</td>
<td>$O(T)$</td>
<td>$O(1)$</td>
<td>$O(\log T)$</td>
</tr>
<tr>
<td>this paper</td>
<td>trees</td>
<td>$O(T)$</td>
<td>$O(1)$</td>
<td>$O(\log T)$</td>
</tr>
</tbody>
</table>
Tree Automata

• MSO query evaluation is non-elementary (if $P \neq NP$)
• MSO query evaluation is **non-elementary** (if $P \neq NP$)
• Most queries are much simpler
• MSO query evaluation is non-elementary (if $P \neq NP$)
• Most queries are much simpler
• We use bottom-up (binary) tree-automata
Tree Automata

- MSO query evaluation is non-elementary (if $P \neq NP$)
- Most queries are much simpler
- We use bottom-up (binary) tree-automata

$\exists y \ldots$

Query
Tree Automata

- MSO query evaluation is non-elementary (if \( P \neq NP \))
- Most queries are much simpler
- We use bottom-up (binary) tree-automata

\[ \exists y \ldots \]  
Query \hspace{1cm} Automaton
• MSO query evaluation is **non-elementary** (if $P \neq NP$)
• Most queries are much simpler
• We use bottom-up (binary) tree-automata
• MSO query evaluation is non-elementary (if $P \neq NP$)
• Most queries are much simpler
• We use bottom-up (binary) tree-automata
Every gate $g$ captures set of sets $S(g)$.
Semantics of set circuits

Every gate $g$ captures set of sets $S(g)$

$$S(\{x:1\}) := \{\{x:1\}\}$$
Semantics of set circuits

Every gate $g$ captures set of sets $S(g)$

- $S(\top) := \{\{\}\}$
- $S(\times_{\text{x:1}}) := \{\{\text{x:1}\}\}$
- $S(\times_{\text{y:3}}) := \{\{\}\}$

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

- For $S(g) = \{\{\text{x}\}\}, \{\text{x, y}\}\}$, enumerate $\{\text{x}\}$ and then $\{\text{x, y}\}$. 

\[\begin{array}{c}
\text{U} \\
\times \\
\times \\
\top \\
\times_{\text{x:1}} \\
\times_{\text{y:3}} \\
\text{\{\{\}\}, \{\text{x}\}, \{\text{y}\}\}}
\end{array}\]
Semantics of set circuits

Every gate $g$ captures set of sets $S(g)$

$S(\begin{array}{c} x:1 \end{array}) := \{\{x:1\}\}$

$S(\begin{array}{c} \top \end{array}) := \{\{\}\}$

$S(\begin{array}{c} \bot \end{array}) := \emptyset$

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

E.g., for $S(g) = \{\{x\}\}$, enumerate $\{x\}$ and then $\{x, y\}$.
Every gate $g$ captures set of sets $S(g)$

- $S(\text{x:1}) := \{\{\text{x:1}\}\}$
- $S(\top) := \{\{\}\}$
- $S(\bot) := \emptyset$
- $S(\times) := \{s_1 \cup s_2 \mid s_1 \in S(g_1), s_2 \in S(g_2)\}$

Semantics of set circuits
Semantics of set circuits

Every gate $g$ captures set of sets $S(g)$

- $S(\texttt{x:1}) := \{{\{x:1\}\}}$
- $S(\top) := \{\{\}\}$
- $S(\bot) := \emptyset$
- $S(\texttt{x:1} \times \texttt{y:3}) := \{s_1 \cup s_2 \mid s_1 \in S(g_1), s_2 \in S(g_2)\}$
- $S(\bigcup) := S(g_1) \cup S(g_2)$
Semantics of set circuits

Every gate $g$ captures set of sets $S(g)$

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

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

$$S(\times:1) := \{\{x:1\}\}$$

$$S(\top) := \{\{}\}$$

$$S(\bot) := \emptyset$$

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

E.g., for $S(g) = \{\{x\}, \{x, y\}\}$, enumerate $\{x\}$ and then $\{x, y\}$.
Compiling Trees in Set Circuits

• One box for each node of the tree
• In each box: one $\cup$-gate for each state $q$
• Captures partial runs that end in $q$
Compiling Trees in Set Circuits

- One box for each node of the tree
- In each box: one $\cup$-gate for each state $q$
- Captures partial runs that end in $q$
Compiling Trees in Set Circuits

- One box for each node of the tree
Compiling Trees in Set Circuits

- One box for each node of the tree
- In each box: one $\cup$-gate for each state $q$ of the automaton
  - Captures partial runs that end in $q$
Enumerate Circuit Results

Preprocessing phase:

DNNF

set circuit
Enumerate Circuit Results

Preprocessing phase:

DNNF set circuit → Normalization (linear-time) → Normalized circuit

Results: /nine.osf//one.osf/six.osf
Enumerate Circuit Results

Preprocessing phase:

1. **DNNF set circuit**
2. **Normalization (linear-time)**
3. **Normalized circuit**
4. **Indexing (linear-time)**
5. **Indexed normalized circuit**

Results:

```
A B C
a b ca b' c
```

Paths:

```
/nine.osf//one.osf/six.osf
```
Enumerate Circuit Results

Preprocessing phase:

DNNF set circuit

Normalization (linear-time)

Indexed normalized circuit

Enumeration phase:

Indexed normalized circuit

Results
Enumerate Circuit Results

Preprocessing phase:

DNNF set circuit

Normalization (linear-time) → Normalized circuit

Indexing (linear-time) → Indexed normalized circuit

Enumeration phase:

Indexed normalized circuit

Enumeration (constant delay) → Results

A B C
a b c a b’ c
Compiling Trees in Set Circuits

- Constructions are bottom-up
- Updates can be done in $O(\text{depth}(T))$
- Problem: $\text{depth}(T)$ can be linear in $T$
- Solution: Depict trees by forest algebra
Compiling Trees in Set Circuits

- Constructions are **bottom-up**

- Updates can be done in $O(\text{depth}(T))$

- Problem: depth($T$) can be linear in $T$

- Solution: Depict trees by forest algebra terms
Constructions are **bottom-up**

Updates can be done in $O(\text{depth}(T))$
Constructions are **bottom-up**

**Updates** can be done in $O(\text{depth}(T))$

Problem: $\text{depth}(T)$ can be linear in $T$
Constructions are bottom-up
Updates can be done in $O(\text{depth}(T))$
Problem: $\text{depth}(T)$ can be linear in $T$
Solution: Depict trees by forest algebra terms
Free Forest Algebra in a Nutshell

concatenation
Free Forest Algebra in a Nutshell

\[ \begin{align*} 
\text{concatenation} & \quad \mathcal{F} \oplus \mathcal{F} = \mathcal{F} \\
\text{context application} & \quad \mathcal{F} \circ \mathcal{F} = \mathcal{F} 
\end{align*} \]
Free Forest Algebra in a Nutshell

concatenation

context

application
The leaves of the formula correspond to the nodes of the tree.
Free Forest Algebra in a Nutshell

The leaves of the formula correspond to the nodes of the tree.
The leaves of the formula correspond to the nodes of the tree.
The leaves of the formula correspond to the nodes of the tree.
The leaves of the formula correspond to the nodes of the tree.
The leaves of the formula correspond to the nodes of the tree.
The leaves of the formula correspond to the nodes of the tree.
The leaves of the formula correspond to the nodes of the tree.
The leaves of the formula correspond to the nodes of the tree.
Rebalancing Forest Algebra Terms
Rebalancing Forest Algebra Terms
Rebalancing Forest Algebra Terms

\[
\begin{array}{c}
\begin{array}{c}
\top \quad \top \quad 3 \\
1 \quad 2 \\
\end{array} \\
\begin{array}{c}
\top \\
1 \\
\end{array}
\end{array}
\quad \leftrightarrow 
\begin{array}{c}
\begin{array}{c}
\top \quad \top \\
1 \quad 2 \quad 3 \\
\end{array} \\
\begin{array}{c}
\top \\
2 \quad 3 \\
\end{array}
\end{array}
\]
Rebalancing Forest Algebra Terms
Rebalancing Forest Algebra Terms
Rebalancing Forest Algebra Terms
Rebalancing Forest Algebra Terms

1 contains the hole
Rebalancing Forest Algebra Terms

1 contains the hole

2 contains the hole
Main Result

Theorem

Enumeration of MSO formulas on trees can be done in time:

Preprocessing \( O(|T| \times |Q|^{4\omega+1}) \)

Delay \( O(|Q|^{4\omega} \times |S|) \)

Updates \( O(\log(|T|) \times |Q|^{4\omega+1}) \)

|\(|T| \quad \) size of tree |
|\(|Q| \quad \) number of states of a nondeterministic tree automaton |
|\(|S| \quad \) size of result |
|\(\omega \quad \) exponent for Boolean matrix multiplication |
Lower Bound
Lower Bound

Existencial Marked Ancestor Queries

**Input:** Tree $t$ with some marked nodes

**Query:** Does node $v$ have a marked ancestor?

**Updates:** Mark or unmark a node

---

**Theorem:** $t_{\text{query}} \in \Omega(\log(n) \log(t_{\text{update}} \log(n)))$

**Reduction to Query Enumeration with Updates**

**Fixed Query $Q$:** Return all special nodes with a marked ancestor

For every marked ancestor query $v$:

1. Mark node $v$ special
2. Enumerate $Q$ and return “yes”, iff $Q$ produces some result
3. Mark $v$ as non-special again

**Theorem:** $\max(t_{\text{delay}}, t_{\text{update}}) \in \Omega(\log(n) \log\log(n))$
Lower Bound

Existencial Marked Ancestor Queries

Input: Tree $t$ with some marked nodes
Query: Does node $v$ have a marked ancestor?
Updates: Mark or unmark a node

Theorem: $t_{\text{query}} \in \Omega \left( \frac{\log(n)}{\log(t_{\text{update}} \log(n))} \right)$
**Lower Bound**

**Existential Marked Ancestor Queries**

**Input:** Tree $t$ with some marked nodes  
**Query:** Does node $v$ have a marked ancestor?  
**Updates:** Mark or unmark a node

**Theorem:** $t_{\text{query}} \in \Omega \left( \frac{\log(n)}{\log(t_{\text{update}} \cdot \log(n))} \right)$

**Reduction to Query Enumeration with Updates**

**Fixed Query $Q$:** Return all special nodes with a marked ancestor  
For every marked ancestor query $v$:

1. Mark node $v$ special  
2. Enumerate $Q$ and return “yes”, iff $Q$ produces some result  
3. Mark $v$ as non-special again
## Lower Bound

### Existencial Marked Ancestor Queries

**Input:** Tree $t$ with some marked nodes  
**Query:** Does node $v$ have a marked ancestor?  
**Updates:** Mark or unmark a node

**Theorem:** $t_{query} \in \Omega \left( \frac{\log(n)}{\log(t_{update} \log(n))} \right)$

### Reduction to Query Enumeration with Updates

**Fixed Query $Q$:** Return all special nodes with a marked ancestor

For every marked ancestor query $v$:

1. Mark node $v$ special
2. Enumerate $Q$ and return “yes”, iff $Q$ produces some result
3. Mark $v$ as non-special again

**Theorem:** $\max(t_{delay}, t_{update}) \in \Omega \left( \frac{\log(n)}{\log \log(n)} \right)$
Results

Theorem

Enumeration of MSO formulas on trees can be done in time:

- **Preprocessing**: $O(|T| \times |Q|^{4\omega+1})$
- **Delay**: $O(|Q|^{4\omega} \times |S|)$
- **Updates**: $O(\log(|T|) \times |Q|^{4\omega+1})$

- $|T|$ *size of tree*
- $|Q|$ *number of states* of a nondeterministic tree automaton
- $|S|$ *size of result*
- $\omega$ *exponent for Boolean matrix multiplication*

Theorem

$$\max(t_{\text{delay}}, t_{\text{update}}) \in \Omega\left(\frac{\log(n)}{\log \log(n)}\right)$$
Results

Theorem

Enumeration of MSO formulas on trees can be done in time:

- **Preprocessing**: $O(|T| \times |Q|^{4\omega + 1})$
- **Delay**: $O(|Q|^{4\omega} \times |S|)$
- **Updates**: $O(\log(|T|) \times |Q|^{4\omega + 1})$

- $|T|$: **size of tree**
- $|Q|$: **number of states** of a nondeterministic tree automaton
- $|S|$: **size of result**
- $\omega$: exponent for Boolean matrix multiplication

Theorem

$$\max(t_{\text{delay}}, t_{\text{update}}) \in \Omega\left(\frac{\log(n)}{\log\log(n)}\right)$$

Thank You


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

In *PODS*. 
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 $\emptyset$

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 $\emptyset$

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 $\emptyset$

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 $\emptyset$

- **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 sets

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

Solution:
- remove inputs with $S(g) = \emptyset$ for $\times$-gates
- collapse $\times$-chains with fan-in / one.osf

Now, traversing a $\times$-gate ensures that we make progress: it splits the sets non-trivially
Normalization: handling empty sets

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

Solution:
- remove inputs with $S(g) = \emptyset$ for $\times$-gates
- collapse $\times$-chains with fan-in /one.osf

Now, traversing a $\times$-gate ensures that we make progress: it splits the sets non-trivially
Problem: if \( S(g) \) contains \( \{\} \) we waste time in chains of \( \times \)-gates

Solution:

• remove inputs with \( S(g) = \{\} \) for \( \times \)-gates
• collapse \( \times \)-chains with fan-in /one.osf

Now, traversing a \( \times \)-gate ensures that we make progress: it splits the sets non-trivially
Normalization: handling empty sets

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

Solution:
- remove inputs with $S(g) = \emptyset$ for $\times$-gates
- collapse $\times$-chains with fan-in / one.osf

Now, traversing a $\times$-gate ensures that we make progress: it splits the sets non-trivially
Normalization: handling empty sets

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

Solution:
- remove inputs with $S(g) = \emptyset$ for $\times$-gates
- collapse $\times$-chains with fan-in /one.osf

Now, traversing a $\times$-gate ensures that we make progress: it splits the sets non-trivially
Normalization: handling empty sets

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

Now, traversing a $\times$-gate ensures that we make progress: it splits the sets non-trivially.
Problem: if $S(g)$ contains $\emptyset$ we waste time in chains of $\times$-gates

Solution:
Normalization: handling empty sets

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

- **Solution:**
  - remove inputs with $S(g) = \{\emptyset\}$ for $\times$-gates
Normalization: handling empty sets

• Problem: if $S(g)$ contains $\emptyset$ we waste time in chains of $\times$-gates

• Solution:
  • remove inputs with $S(g) = \emptyset$ for $\times$-gates
**Problem:** if $S(g)$ contains {} we waste time in chains of $\times$-gates

**Solution:**
- remove inputs with $S(g) = \{\{\}\}$ for $\times$-gates
- collapse $\times$-chains with fan-in 1
Normalization: handling empty sets

- **Problem:** if $S(g)$ contains $\emptyset$ we waste time in chains of $\times$-gates
- **Solution:**
  - remove inputs with $S(g) = \{\emptyset\}$ for $\times$-gates
  - collapse $\times$-chains with fan-in 1
Normalization: handling empty sets

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

- **Solution:**
  - remove inputs with $S(g) = \emptyset$ for $\times$-gates
  - collapse $\times$-chains with fan-in 1

→ Now, traversing a $\times$-gate ensures that we make progress: it *splits* the sets non-trivially
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

Solution: Compute reachability index with box granularity

Use matrix multiplication

Circuit has bounded width (by the size of the automaton)
Indexing: handling $\cup$-hierarchies

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

Circuit has bounded width (by the size of the automaton)
• **Problem:** we waste time in $\cup$-hierarchies to find a **reachable exit** (non-$\cup$ gate)

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

Solution: Compute reachability index with box-granularity

- Use matrix multiplication
- Circuit has bounded width (by the size of the automaton)