Enumeration on Trees with Tractable Combined Complexity and Efficient Updates

Antoine Amarilli[1], Pierre Bourhis[2], Stefan Mengel[3], **Matthias Niewerth**[4]

May 20th, 2019

[1]Télécom ParisTech

[2]CNRS, CRIStAL, Lille

[3]CNRS, CRIL, Lens

[4]University of Bayreuth

Dramatis Personae

Antoine Amarilli

Pierre Bourhis

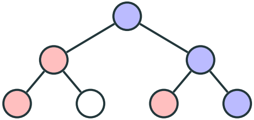
Stefan Mengel

Matthias Niewerth

Problem statement

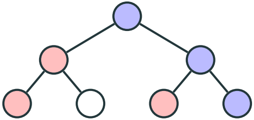
MSO query evaluation on trees

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



MSO query evaluation on trees

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



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

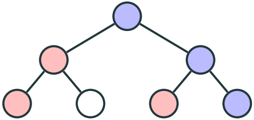
- $P_{\bigcirc}(x)$ means "x is blue"
- $x \rightarrow y$ means "x is the parent of y"

"Return all blue nodes that have a pink child"

$\exists y\ P_{\bigcirc}(x) \land P_{\bigcirc}(y) \land x \rightarrow y$

MSO query evaluation on trees

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



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

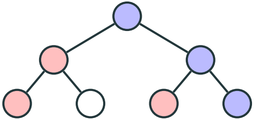
- $P_{\circ}(x)$ means "x is blue"
- $x \to y$ means "x is the parent of y"

"Return all blue nodes that have a pink child"

$\exists y \; P_{\circ}(x) \wedge P_{\circ}(y) \wedge x \to y$

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

MSO query evaluation on trees



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

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"

$\exists y \; P_{\circ}(x) \wedge P_{\circ}(y) \wedge x \to y$

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

Up to $|T|^k$ many answers

Enumeration algorithm

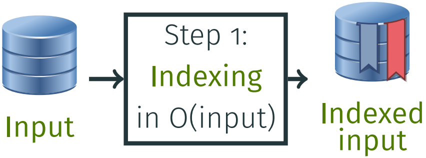
Input

Enumeration algorithm

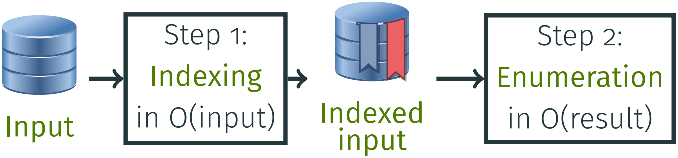
Input →

Step 1:
Indexing
in O(input)

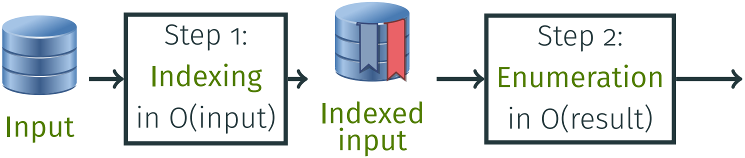
Enumeration algorithm



Enumeration algorithm



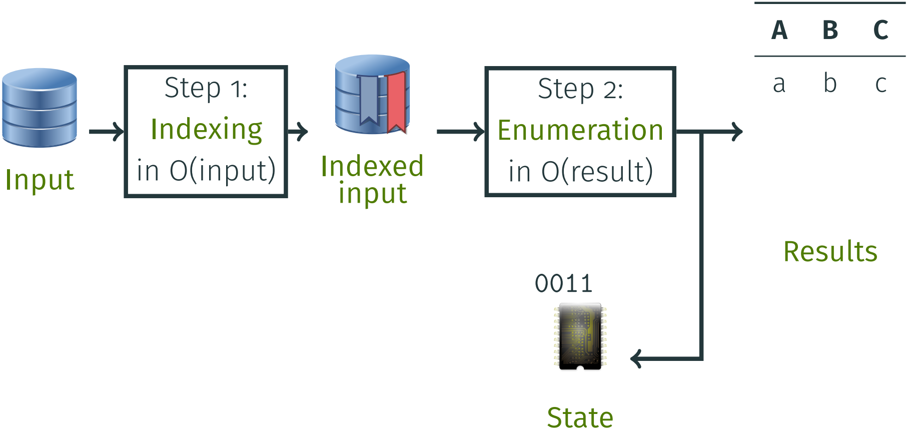
Enumeration algorithm



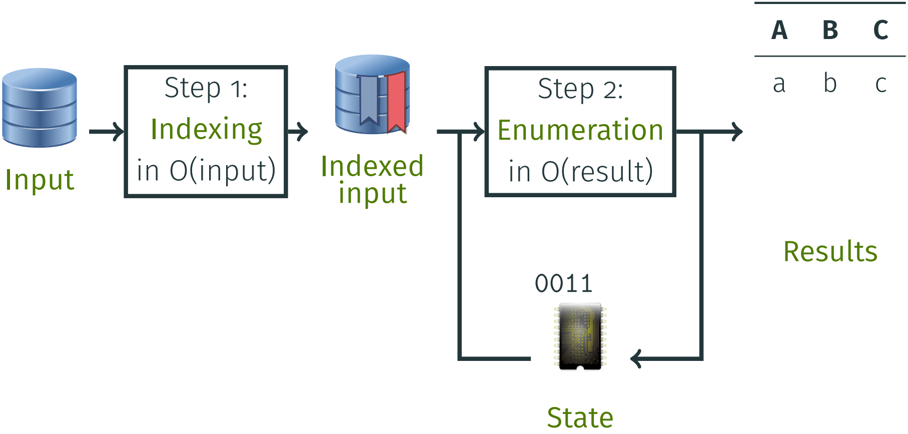
A	**B**	**C**
a	b	c

Results

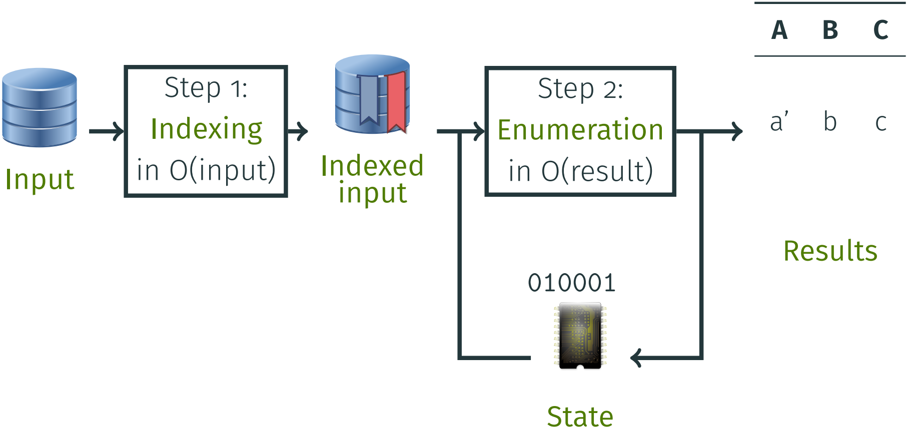
Enumeration algorithm



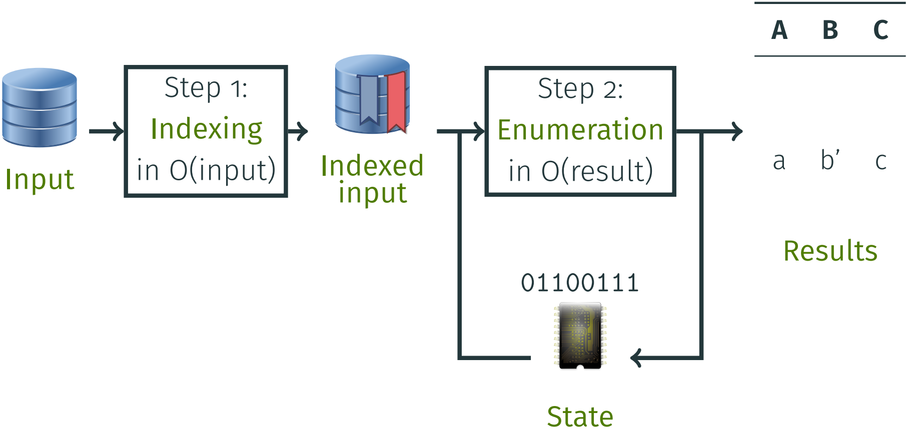
Enumeration algorithm



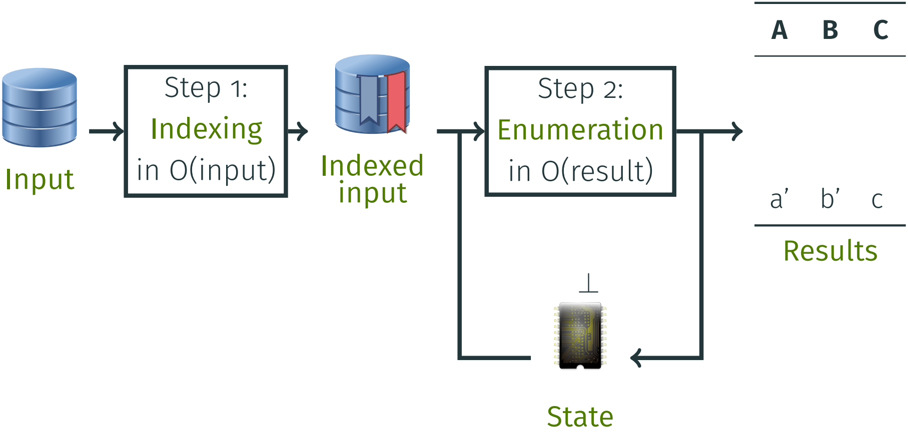
Enumeration algorithm



Enumeration algorithm



Enumeration algorithm



Known results on dynamic trees

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

Work	Data	Preproc.	Delay	Updates
[Bagan, 2006], [Kazana and Segoufin, 2013]	trees	$O(T)$	$O(1)$	$O(T)$

Known results on dynamic trees

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

Work	Data	Preproc.	Delay	Updates
[Bagan, 2006], [Kazana and Segoufin, 2013]	trees	$O(T)$	$O(1)$	$O(T)$
[Losemann and Martens, 2014]	trees	$O(T)$	$O(\log^2 T)$	$O(\log^2 T)$

Known results on dynamic trees

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

Work	Data	Preproc.	Delay	Updates
[Bagan, 2006], [Kazana and Segoufin, 2013]	trees	$O(T)$	$O(1)$	$O(T)$
[Losemann and Martens, 2014]	trees	$O(T)$	$O(\log^2 T)$	$O(\log^2 T)$
[Niewerth, 2018]	trees	$O(T)$	$O(\log T)$	$O(\log T)$

Known results on dynamic trees

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

Work	Data	Preproc.	Delay	Updates
[Bagan, 2006], [Kazana and Segoufin, 2013]	trees	$O(T)$	$O(1)$	$O(T)$
[Losemann and Martens, 2014]	trees	$O(T)$	$O(\log^2 T)$	$O(\log^2 T)$
[Niewerth, 2018]	trees	$O(T)$	$O(\log T)$	$O(\log T)$
[Niewerth and Segoufin, 2018]	text	$O(T)$	$O(1)$	$O(\log T)$

Known results on dynamic trees

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

Work	Data	Preproc.	Delay	Updates
[Bagan, 2006], [Kazana and Segoufin, 2013]	trees	$O(T)$	$O(1)$	$O(T)$
[Losemann and Martens, 2014]	trees	$O(T)$	$O(\log^2 T)$	$O(\log^2 T)$
[Niewerth, 2018]	trees	$O(T)$	$O(\log T)$	$O(\log T)$
[Niewerth and Segoufin, 2018]	text	$O(T)$	$O(1)$	$O(\log T)$
this paper	trees	$O(T)$	$O(1)$	$O(\log T)$

Tree Automata

- MSO query evaluation is non-elementary (if $P \neq NP$)

Tree Automata

- MSO query evaluation is non-elementary (if $P \neq NP$)
- Most queries are much simpler

Tree Automata

- 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

Automaton

Tree Automata

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

$\exists y \ldots$ $\longrightarrow$

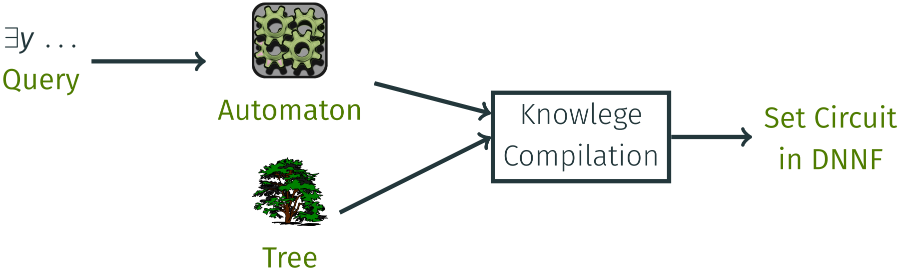
Query

Automaton

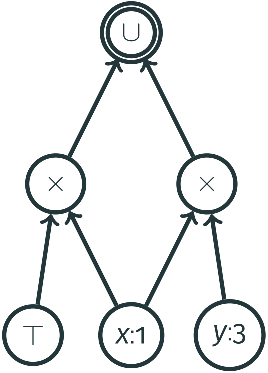
Tree

Tree Automata

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

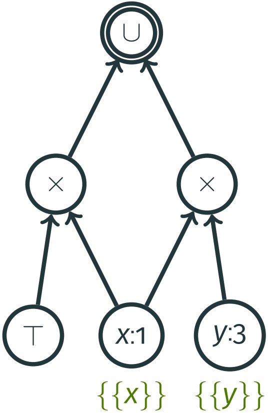


Semantics of set circuits



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

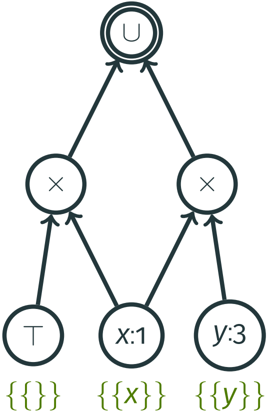
Semantics of set circuits



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

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

Semantics of set circuits

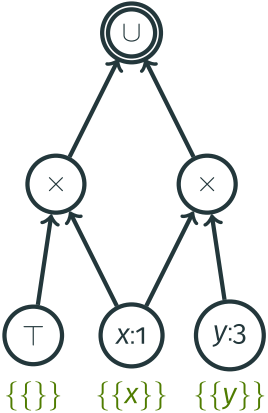


Every gate g **captures** set of sets $S(g)$

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

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

Semantics of set circuits



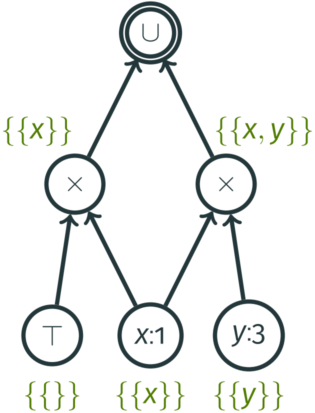
Every gate g captures set of sets $S(g)$

$$S(\,\boxed{x{:}1}\,) := \{\{x{:}1\}\}$$

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

$$S(\,\boxed{\bot}\,) := \emptyset$$

Semantics of set circuits



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

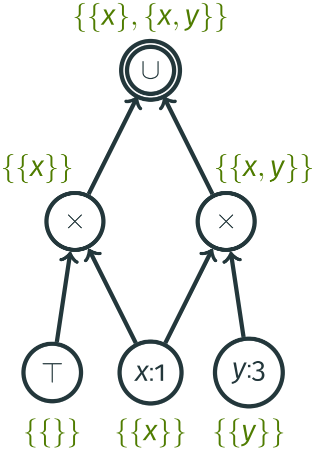
$S(\text{(}x{:}1\text{)}) := \{\{x{:}1\}\}$

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

$S(\text{(}\bot\text{)}) := \emptyset$

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

Semantics of set circuits



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

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

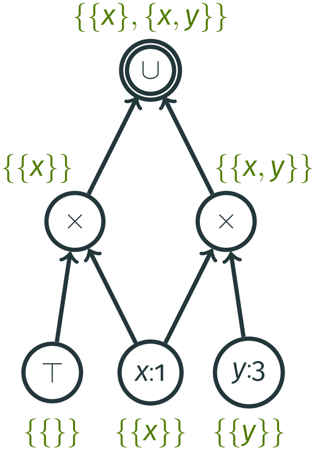
$$S(\,\top\,) := \{\{\}\}$$

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

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

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

Semantics of set circuits



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

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

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

$S(\text{(}\bot\text{)}) := \emptyset$

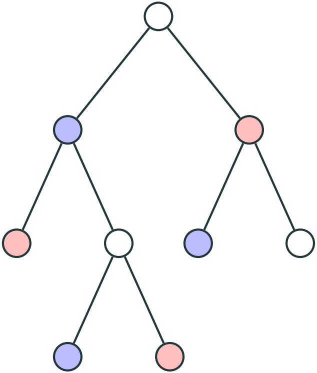
$S(\text{(}\times\text{)}) := \{s_1 \cup s_2 \mid s_1 \in S(g_1), s_2 \in S(g_2)\}$

$S(\text{(}\cup\text{)}) := S(g_1) \cup S(g_2)$

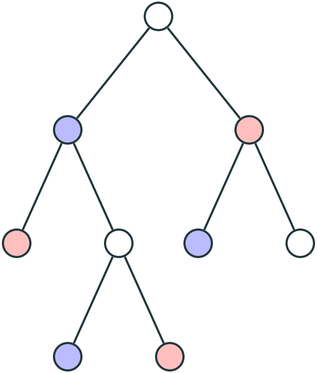
Task: Enumerate the elements of the set $S(g)$ captured by a gate g

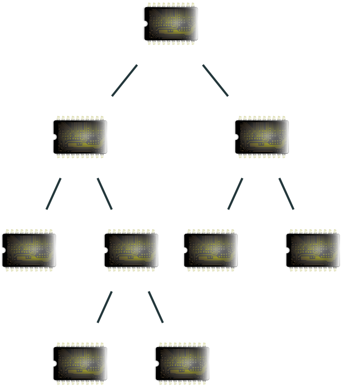
$\rightarrow$E.g., for $S(g) = \{\{x\}, \{x, y\}\}$, enumerate $\{x\}$ and then $\{x, y\}$

Compiling Trees in Set Circuits

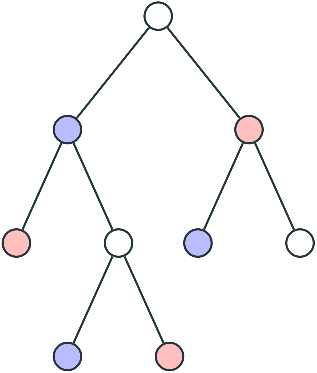


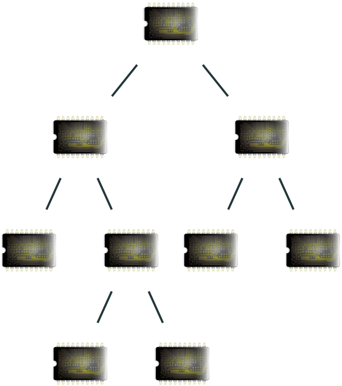
Compiling Trees in Set Circuits





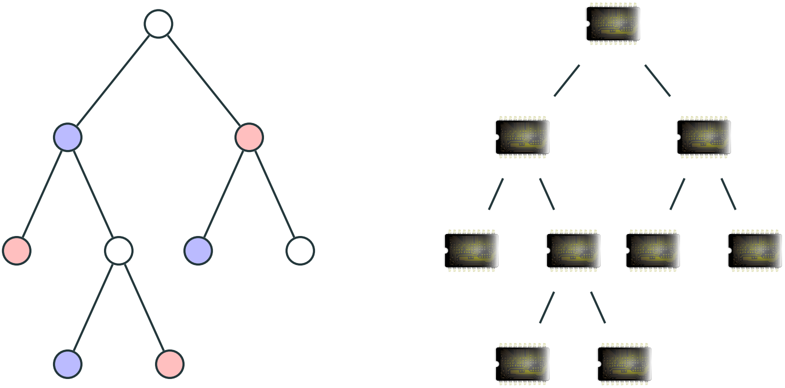
Compiling Trees in Set Circuits





- One box for each node of the tree

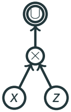
Compiling Trees in Set Circuits



- One box for each node of the tree
- In each box: one ∪-gate for each state q of the automaton
  - Captures partial runs that end in q

Enumerate Circuit Results

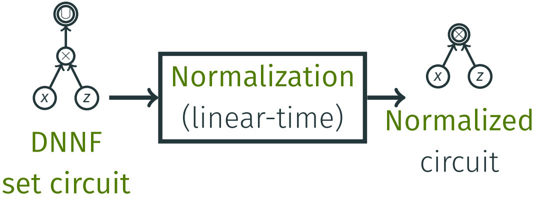
Preprocessing phase:



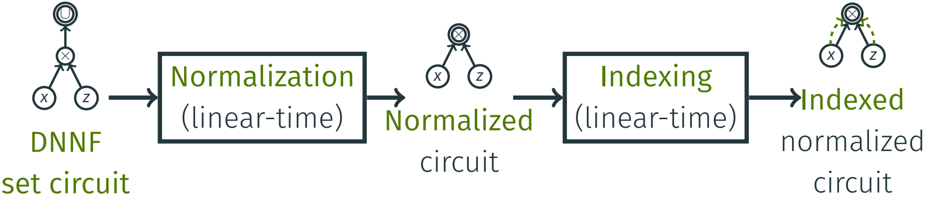
DNNF
set circuit

Enumerate Circuit Results

Preprocessing phase:

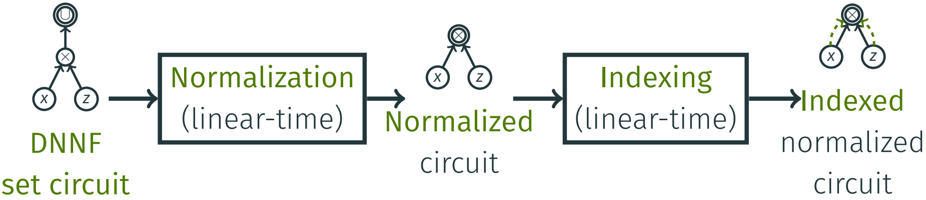


Preprocessing phase:

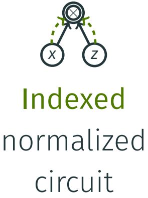


Enumerate Circuit Results

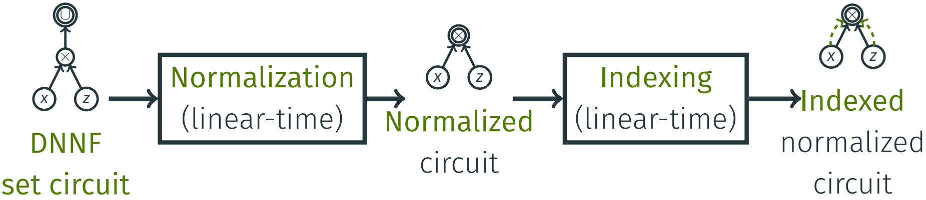
Preprocessing phase:



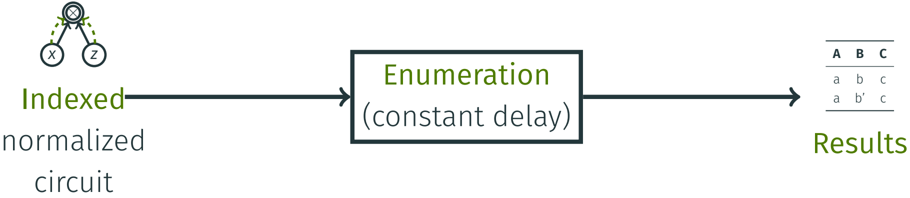
Enumeration phase:



Preprocessing phase:

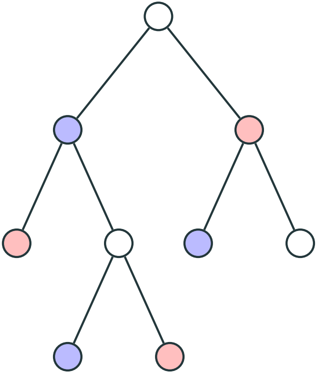


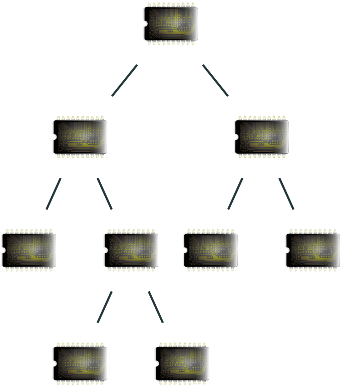
Enumeration phase:



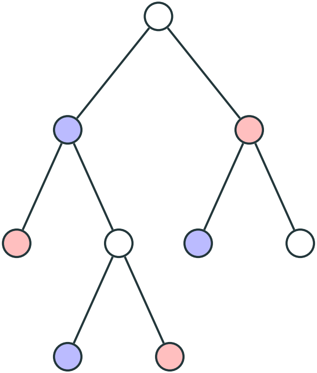
A	B	C
a	b	c
a	b′	c

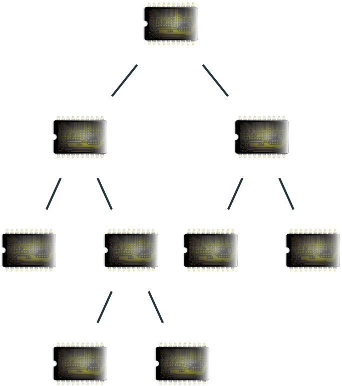
Compiling Trees in Set Circuits





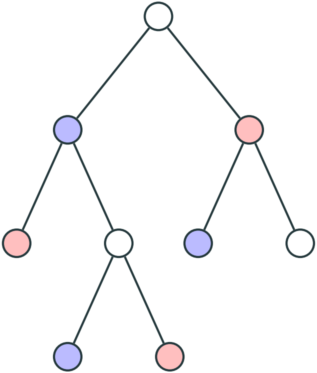
Compiling Trees in Set Circuits

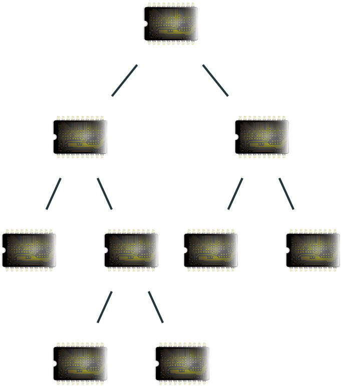




- Constructions are bottom-up

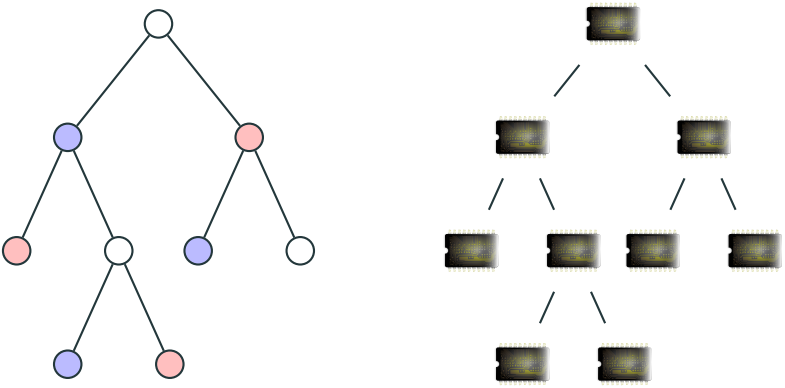
Compiling Trees in Set Circuits





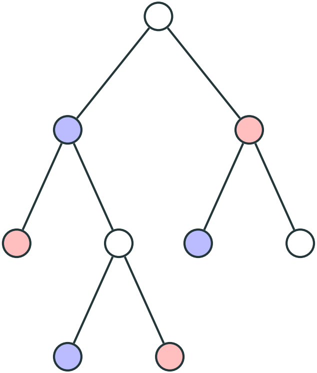
- Constructions are **bottom-up**
- **Updates** can be done in $\mathcal{O}(\mathrm{depth}(T))$

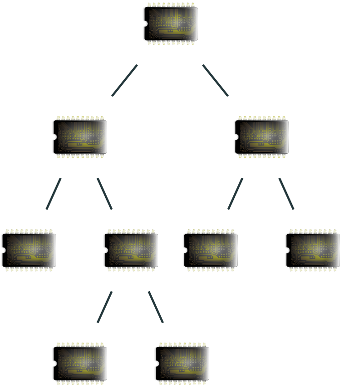
Compiling Trees in Set Circuits



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

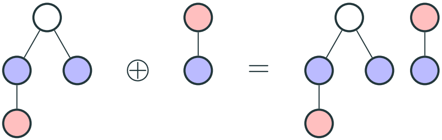
Compiling Trees in Set Circuits





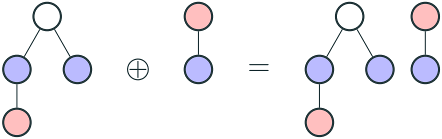
- Constructions are bottom-up
- Updates can be done in $\mathcal{O}(\text{depth}(T))$
- Problem: $\text{depth}(T)$ can be linear in T
- Solution: Depict trees by forest algebra terms

Free Forest Algebra in a Nutshell

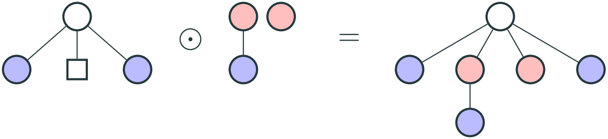


concatenation

Free Forest Algebra in a Nutshell

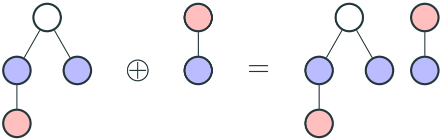


concatenation

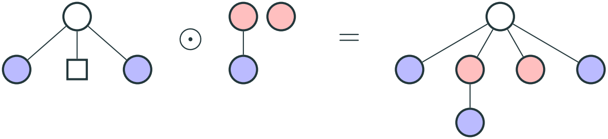


context
application

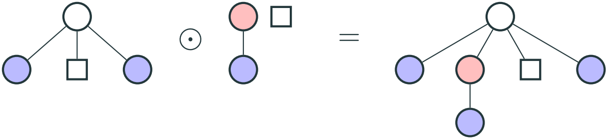
Free Forest Algebra in a Nutshell



concatenation



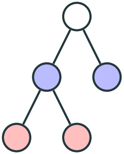
context application



context application

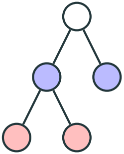
Free Forest Algebra in a Nutshell

tree

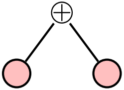


Free Forest Algebra in a Nutshell

tree

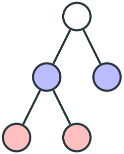


term

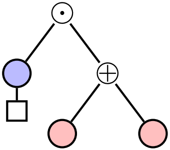


Free Forest Algebra in a Nutshell

tree

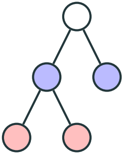


term

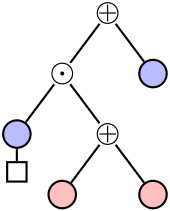


Free Forest Algebra in a Nutshell

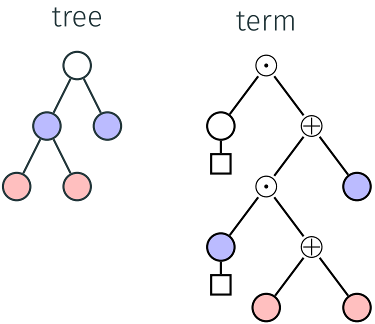
tree



term

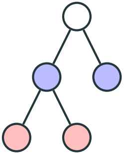


Free Forest Algebra in a Nutshell

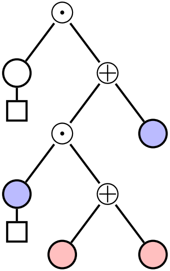


Free Forest Algebra in a Nutshell

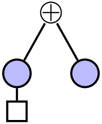
tree



term

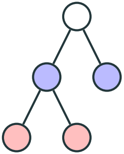


alternative term

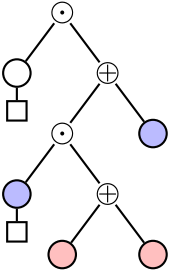


Free Forest Algebra in a Nutshell

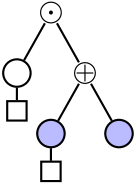
tree



term

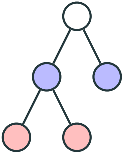


alternative term

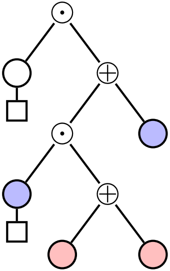


Free Forest Algebra in a Nutshell

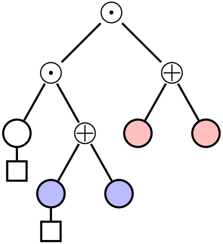
tree



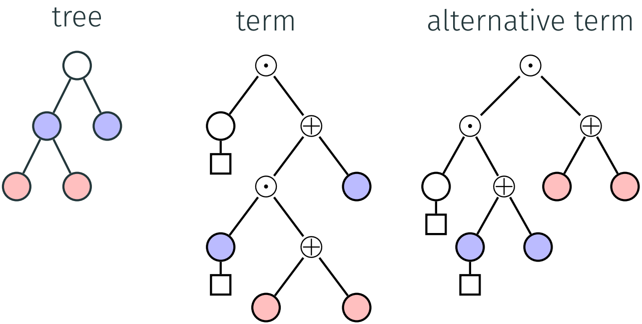
term



alternative term

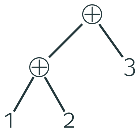


Free Forest Algebra in a Nutshell

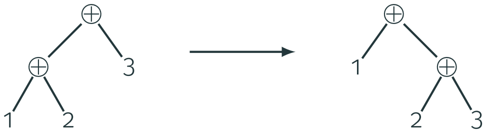


The leaves of the formula correspond to the nodes of the tree

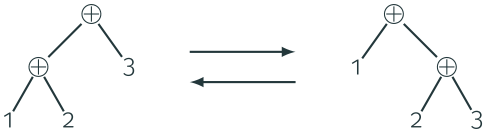
Rebalancing Forest Algebra Terms



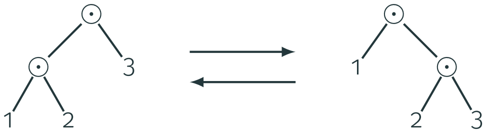
Rebalancing Forest Algebra Terms



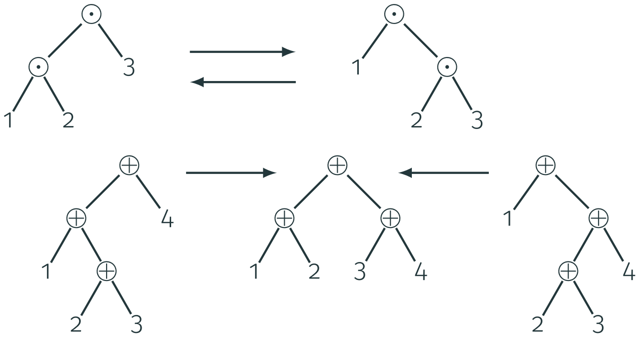
Rebalancing Forest Algebra Terms



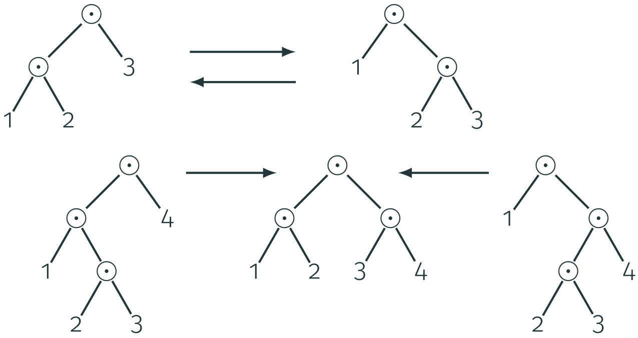
Rebalancing Forest Algebra Terms



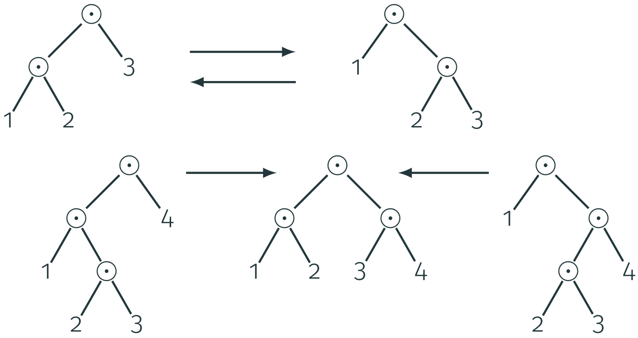
Rebalancing Forest Algebra Terms



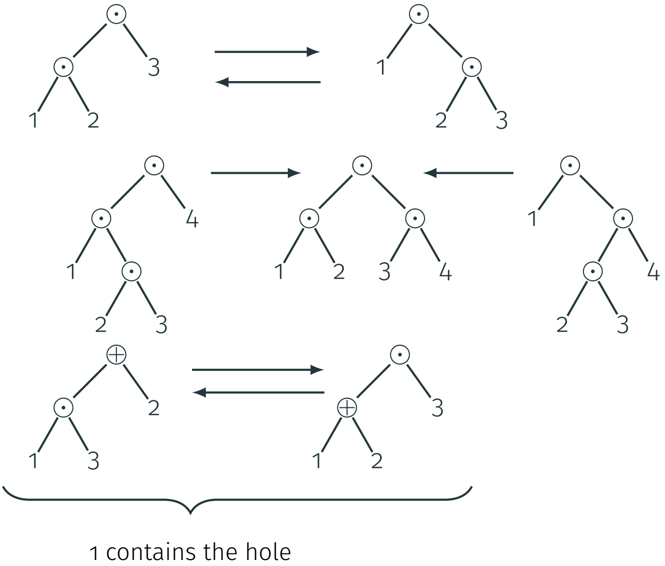
Rebalancing Forest Algebra Terms



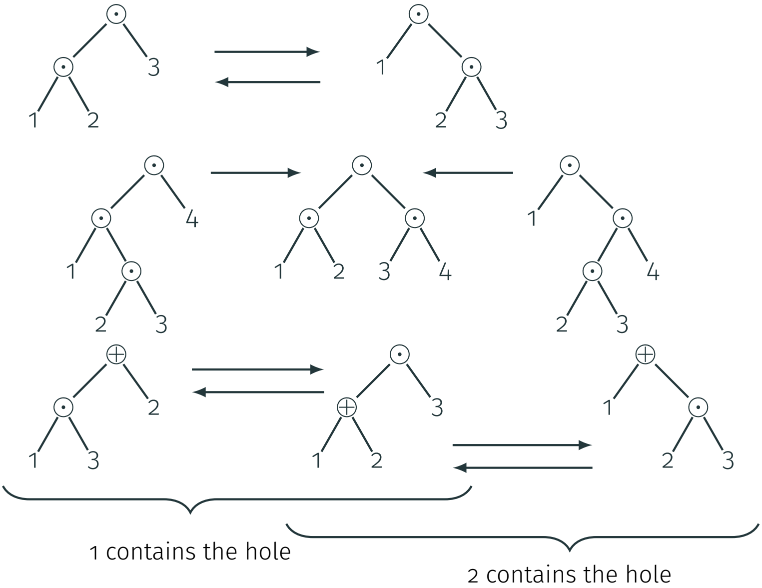
Rebalancing Forest Algebra Terms



Rebalancing Forest Algebra Terms



Rebalancing Forest Algebra Terms



Main Result

Theorem

Enumertion 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*
ω	*exponent for Boolean matrix multiplication*

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

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

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

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

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

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

Results

Theorem

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

Preprocessing	$O(\ \lvert T\rvert \times \lvert Q\rvert^{4\omega+1}\ )$
Delay	$O(\ \lvert Q\rvert^{4\omega} \times \lvert S\rvert\ )$
Updates	$O(\ \log(\lvert T\rvert) \times \lvert Q\rvert^{4\omega+1}\ )$

$\lvert T\rvert$	*size of tree*
$\lvert Q\rvert$	*number of states of a nondeterministic tree automaton*
$\lvert S\rvert$	*size of result*
ω	*exponent for Boolean matrix multiplication*

Theorem

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

Results

Theorem

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

Preprocessing	$O(\ \lvert T\rvert \times \lvert Q\rvert^{4\omega+1}\ )$
Delay	$O(\ \lvert Q\rvert^{4\omega} \times \lvert S\rvert\ )$
Updates	$O(\ \log(\lvert T\rvert) \times \lvert Q\rvert^{4\omega+1}\ )$

$\lvert T\rvert$	*size of tree*
$\lvert Q\rvert$	*number of states of a nondeterministic tree automaton*
$\lvert S\rvert$	*size of result*
ω	*exponent for Boolean matrix multiplication*

Theorem

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

Thank You

References i

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

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

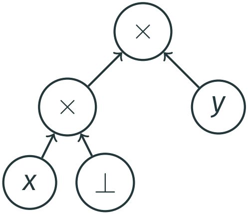
Losemann, K. and Martens, W. (2014).
MSO queries on trees: Enumerating answers under updates.
In *CSL-LICS*.

References ii

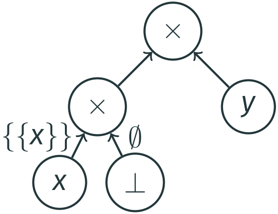
Niewerth, M. (2018).
Mso queries on trees: Enumerating answers under updates using forest algebras.
In *LICS*.

Niewerth, M. and Segoufin, L. (2018).
Enumeration of MSO queries on strings with constant delay and logarithmic updates.
In *PODS*.

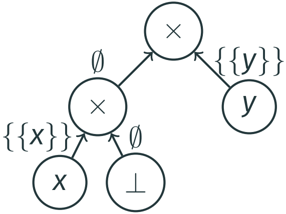
Normalization: handling $\emptyset$



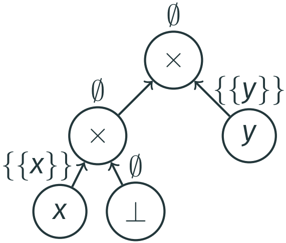
Normalization: handling $\emptyset$



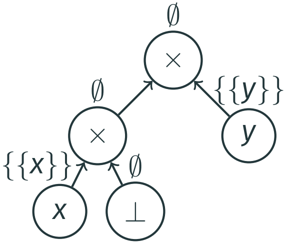
Normalization: handling $\emptyset$



Normalization: handling $\emptyset$

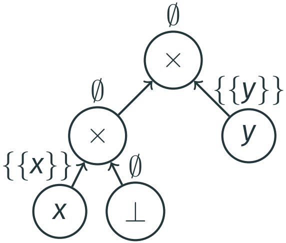


Normalization: handling $\emptyset$



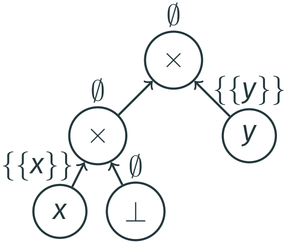
- Problem: if $S(g) = \emptyset$ we waste time

Normalization: handling $\emptyset$



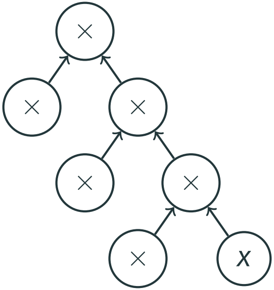
- Problem: if $S(g) = \emptyset$ we waste time
- Solution: in preprocessing
  - compute bottom-up if $S(g) = \emptyset$

Normalization: handling $\emptyset$

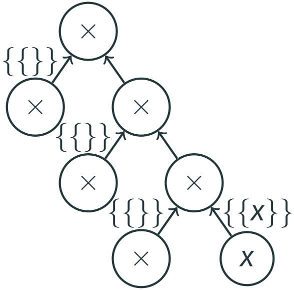


- Problem: if $S(g) = \emptyset$ we waste time
- Solution: in preprocessing
  - compute bottom-up if $S(g) = \emptyset$
  - then get rid of the gate

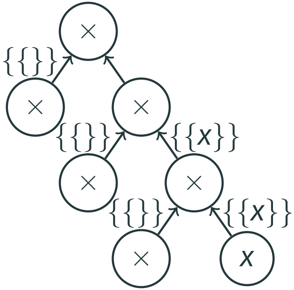
Normalization: handling empty sets



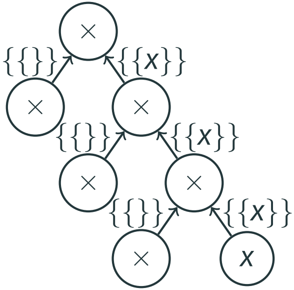
Normalization: handling empty sets



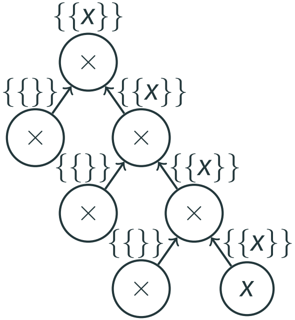
Normalization: handling empty sets



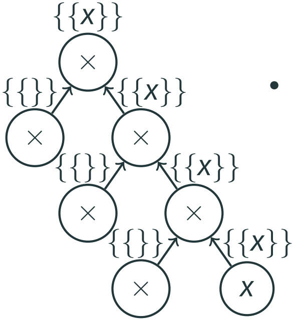
Normalization: handling empty sets



Normalization: handling empty sets

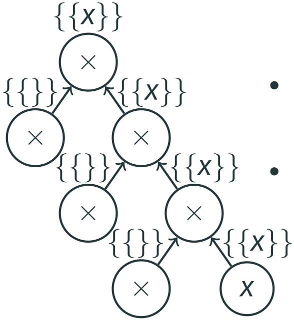


Normalization: handling empty sets



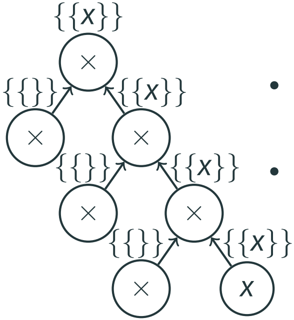
- Problem: if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates

Normalization: handling empty sets



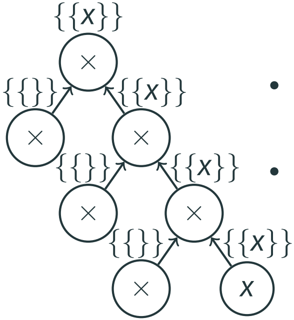
- Problem: if $S(g)$ contains {} we waste time in chains of ×-gates
- Solution:

Normalization: handling empty sets



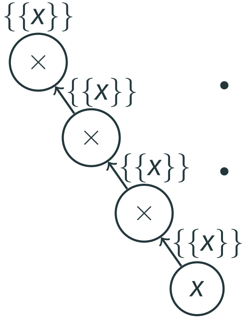
- Problem: if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- Solution:
  - remove inputs with $S(g) = \{\{\}\}$ for $\times$-gates

Normalization: handling empty sets



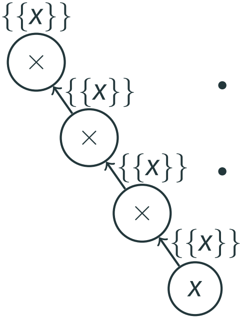
- Problem: if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- Solution:
  - remove inputs with $S(g) = \{\{\}\}$ for $\times$-gates

Normalization: handling empty sets



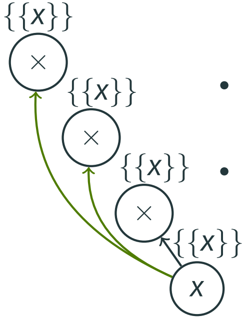
- Problem: if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- Solution:
  - remove inputs with $S(g) = \{\{\}\}$ for $\times$-gates
  - collapse $\times$-chains with fan-in 1

Normalization: handling empty sets



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

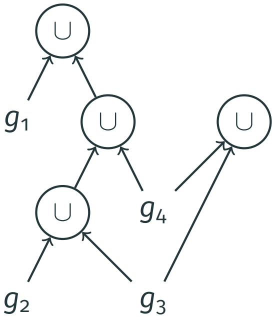
Normalization: handling empty sets



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

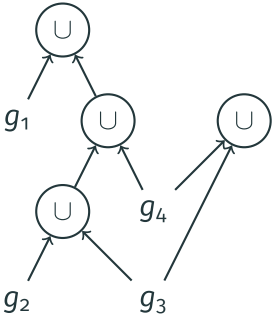
$\rightarrow$ Now, traversing a $\times$-gate ensures that we make progress: it splits the sets non-trivially

Indexing: handling ∪-hierarchies



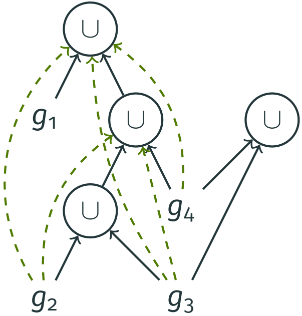
- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)

Indexing: handling ∪-hierarchies



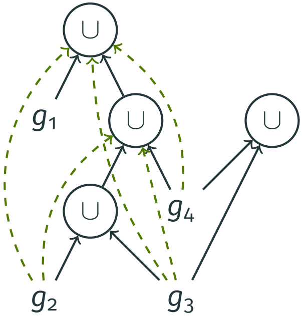
- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)
- **Solution:** compute **reachability index**

Indexing: handling ∪-hierarchies



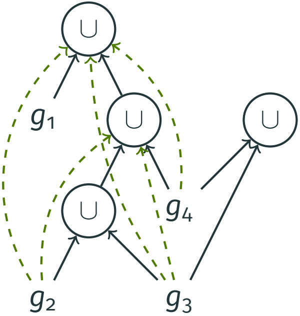
- **Problem:** we waste time in ∪-hierarchies to find a reachable exit (non-∪ gate)
- **Solution:** compute reachability index

Indexing: handling ∪-hierarchies



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

Indexing: handling ∪-hierarchies



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

- Solution: Compute reachability index with box-granularity
- Use matrix multiplication
- Circuit has bounded width (by the size of the automaton)