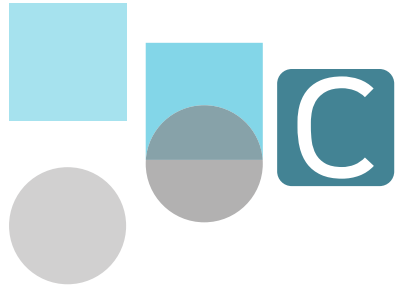
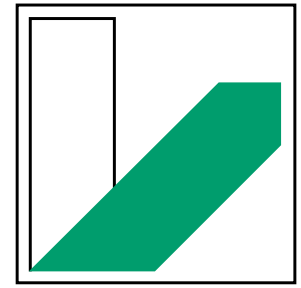


Enumeration for MSO Queries on Trees via Circuits

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

July 14, 2019

[1]Télécom Paris, Institut Polytechnique de Paris

[2]CNRS CRIStAL

[3]CNRS CRIL

[4]Universität Bayreuth

Enumeration for MSO Queries on Trees via Circuits

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

July 14, 2019

[1]Télécom Paris, Institut Polytechnique de Paris

[2]CNRS CRIStAL

[3]CNRS CRIL

[4]Universität Bayreuth

Circuits pour l'énumération de MSO sur des arbres

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

July 14, 2019

[1]Télécom Paris, Institut Polytechnique de Paris

[2]CNRS CRIStAL

[3]CNRS CRIL

[4]Universität Bayreuth

Query evaluation

The main database problem is query evaluation

Data D: what we know

Query evaluation

The main database problem is **query evaluation**

Data D: what we know

Query Q: question asked about the data

Query evaluation

The main database problem is **query evaluation**

Data D: what we know

Query Q: question asked about the data

Result: all results of the query Q on the data D

Query evaluation

The main database problem is **query evaluation**

Data D: what we know

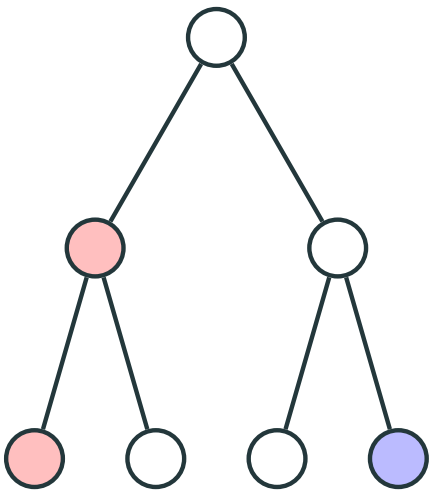
Query Q: question asked about the data

Result: all results of the query Q on the data D

Measure of efficiency: computational complexity:

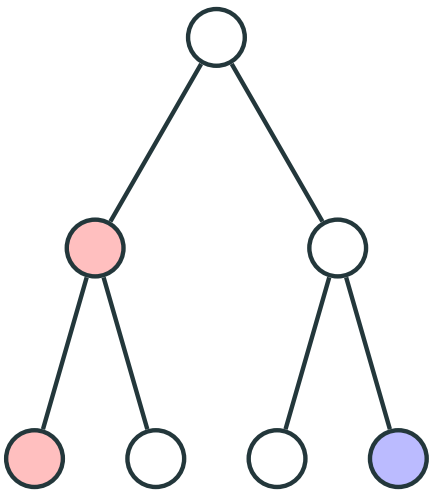
- Combined complexity: D and Q are **inputs**
- Data complexity: Q is **fixed**, D is the **input**

Structured data: trees



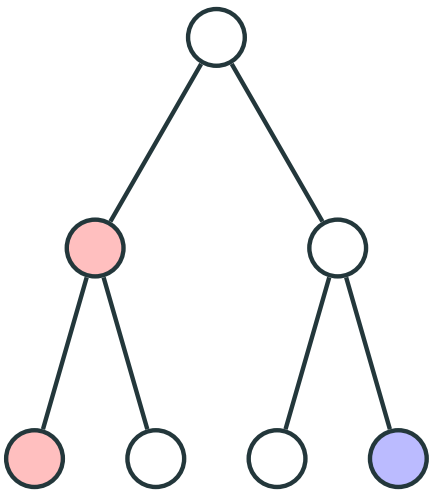
- Several kinds of data are structured as a tree (HTML pages, XML, folder hierarchies…)
  - Important special case: text

Structured data: trees



- Several kinds of data are structured as a tree (HTML pages, XML, folder hierarchies…)
  - Important special case: text
- On this kind of data, we can use more efficient query evaluation techniques

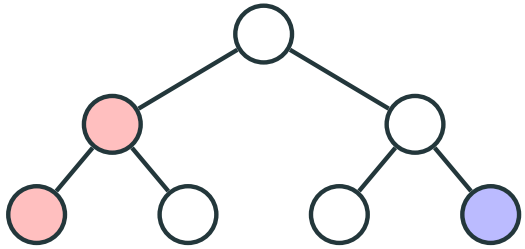
Structured data: trees



- Several kinds of data are structured as a **tree** (HTML pages, XML, folder hierarchies...)
  - Important special case: **text**
- On this kind of data, we can use **more efficient** query evaluation techniques
- Natural query language: **monadic second-order logic** (MSO)
  - Very expressive
  - Corresponds to **tree automata**
  - Data complexity is in **linear time**

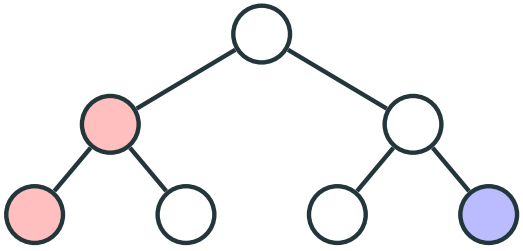
Query evaluation on trees

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



Query evaluation on trees

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



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

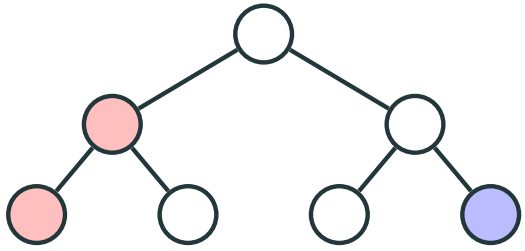
- $P_{●}(x)$ means "x is blue"
- $x \to y$ means "x is the parent of y"

"Is there both a pink and a blue node?"

$\exists x\, y\; P_{●}(x) \wedge P_{●}(y)$

Query evaluation on trees

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



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

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

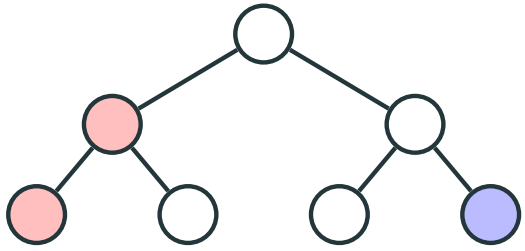
"Is there both a pink and a blue node?"

$\exists x\, y\; P_{\bullet}(x) \wedge P_{\bullet}(y)$

Result: YES/NO indicating if the tree T satisfies the query Q

Query evaluation on trees

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



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

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

"Is there both a pink and a blue node?"

$\exists x\, y\; P_{\bullet}(x) \wedge P_{\bullet}(y)$

Result: YES/NO indicating if the tree T satisfies the query Q

- Data complexity: linear in T
- Combined complexity: cannot be elementary

Beyond Boolean queries

- Boolean queries only gives YES/NO answers
  $\rightarrow$ YES, there is both a pink and a blue node

Beyond Boolean queries

- Boolean queries only gives YES/NO answers
  $\rightarrow$ YES, there is both a pink and a blue node

- Better idea: non-Boolean queries
  $\rightarrow$ e.g., $Q(x, y)$: "x is a pink node and y is a blue node"

Beyond Boolean queries

- Boolean queries only gives YES/NO answers
  $\rightarrow$ YES, there is both a pink and a blue node
- Better idea: non-Boolean queries
  $\rightarrow$ e.g., $Q(x, y)$: "x is a pink node and y is a blue node"
- Challenge: how to define complexity for non-Boolean queries?

Beyond Boolean queries

- Boolean queries only gives YES/NO answers
  $\rightarrow$ YES, there is both a pink and a blue node

- Better idea: non-Boolean queries
  $\rightarrow$ e.g., $Q(x, y)$: "x is a pink node and y is a blue node"

- Challenge: how to define complexity for non-Boolean queries?
  - Just writing the output is slow in the worst case

Beyond Boolean queries

- Boolean queries only gives YES/NO answers
  → YES, there is both a pink and a blue node

- Better idea: non-Boolean queries
  → e.g., $Q(x, y)$: "x is a pink node and y is a blue node"

- Challenge: how to define complexity for non-Boolean queries?
  - Just writing the output is slow in the worst case
  - Easy solution: the naive algorithm that tests all pairs

Beyond Boolean queries

- Boolean queries only gives YES/NO answers
  → YES, there is both a pink and a blue node

- Better idea: non-Boolean queries
  → e.g., $Q(x, y)$: "x is a pink node and y is a blue node"

- Challenge: how to define complexity for non-Boolean queries?
  - Just writing the output is slow in the worst case
  - Easy solution: the naive algorithm that tests all pairs

  → We need a new definition of complexity

Idea: Enumeration algorithms

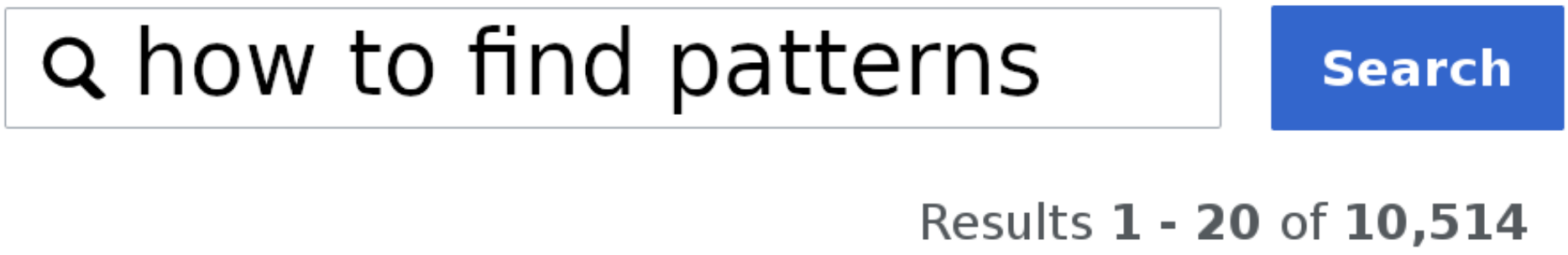
Idea: In real life, we do not want to compute all the matches
we just need to be able to enumerate matches quickly

Idea: Enumeration algorithms

Idea: In real life, we do not want to compute all the matches
we just need to be able to enumerate matches quickly

Idea: Enumeration algorithms

Idea: In real life, we do not want to compute all the matches
we just need to be able to enumerate matches quickly



Idea: Enumeration algorithms

Idea: In real life, we do not want to compute all the matches
we just need to be able to enumerate matches quickly

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

...

Idea: Enumeration algorithms

Idea: In real life, we do not want to compute all the matches
we just need to be able to enumerate matches quickly

how to find patterns

Search

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

...

View (previous 20 | next 20) (20 | 50 | 100 | 250 | 500)

Idea: Enumeration algorithms

Idea: In real life, we do not want to compute all the matches
we just need to be able to enumerate matches quickly

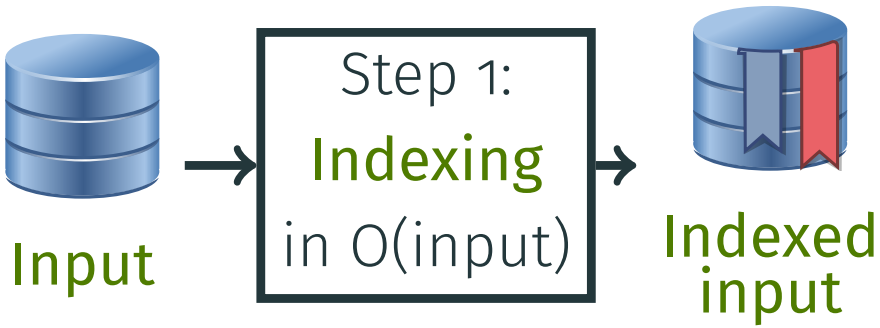
→ Formalization: enumeration algorithms

Formalizing an enumeration algorithm

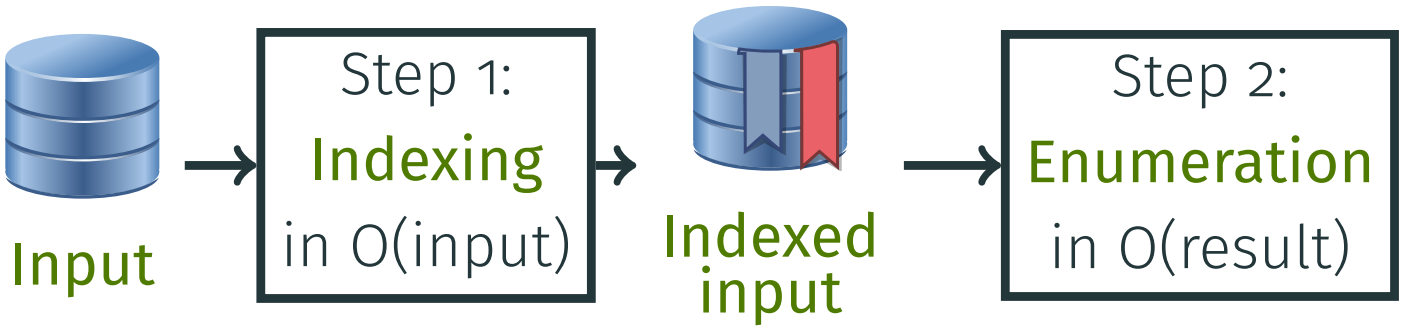
Input

Formalizing an enumeration algorithm

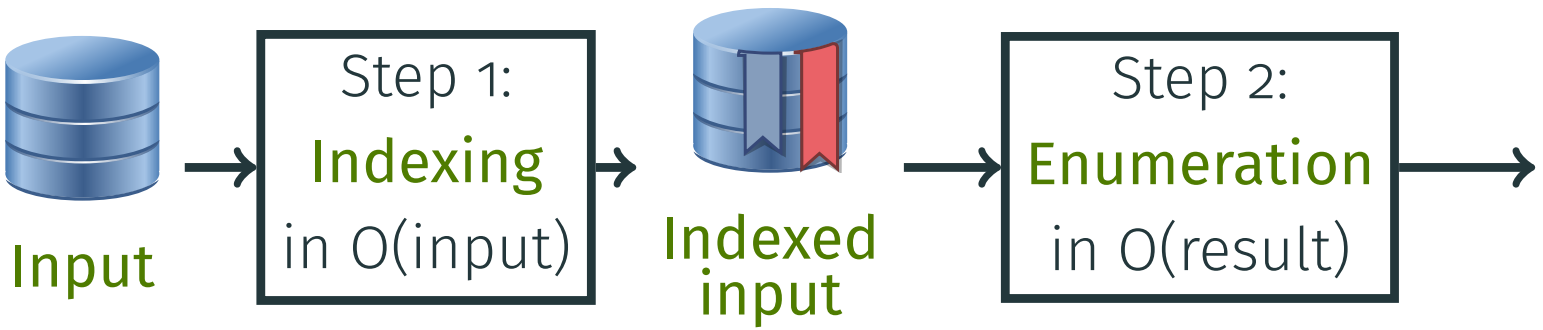
Formalizing an enumeration algorithm



Formalizing an enumeration algorithm



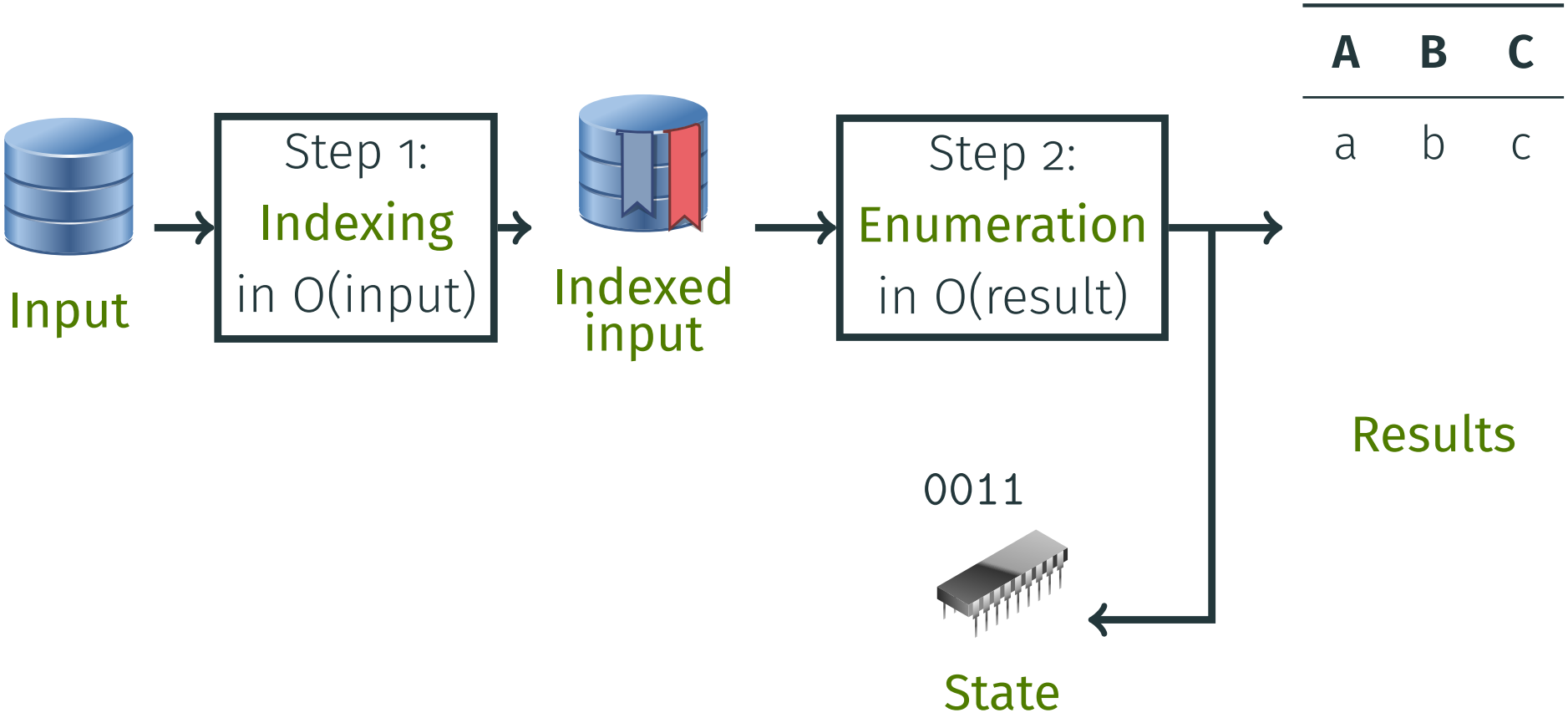
Formalizing an enumeration algorithm



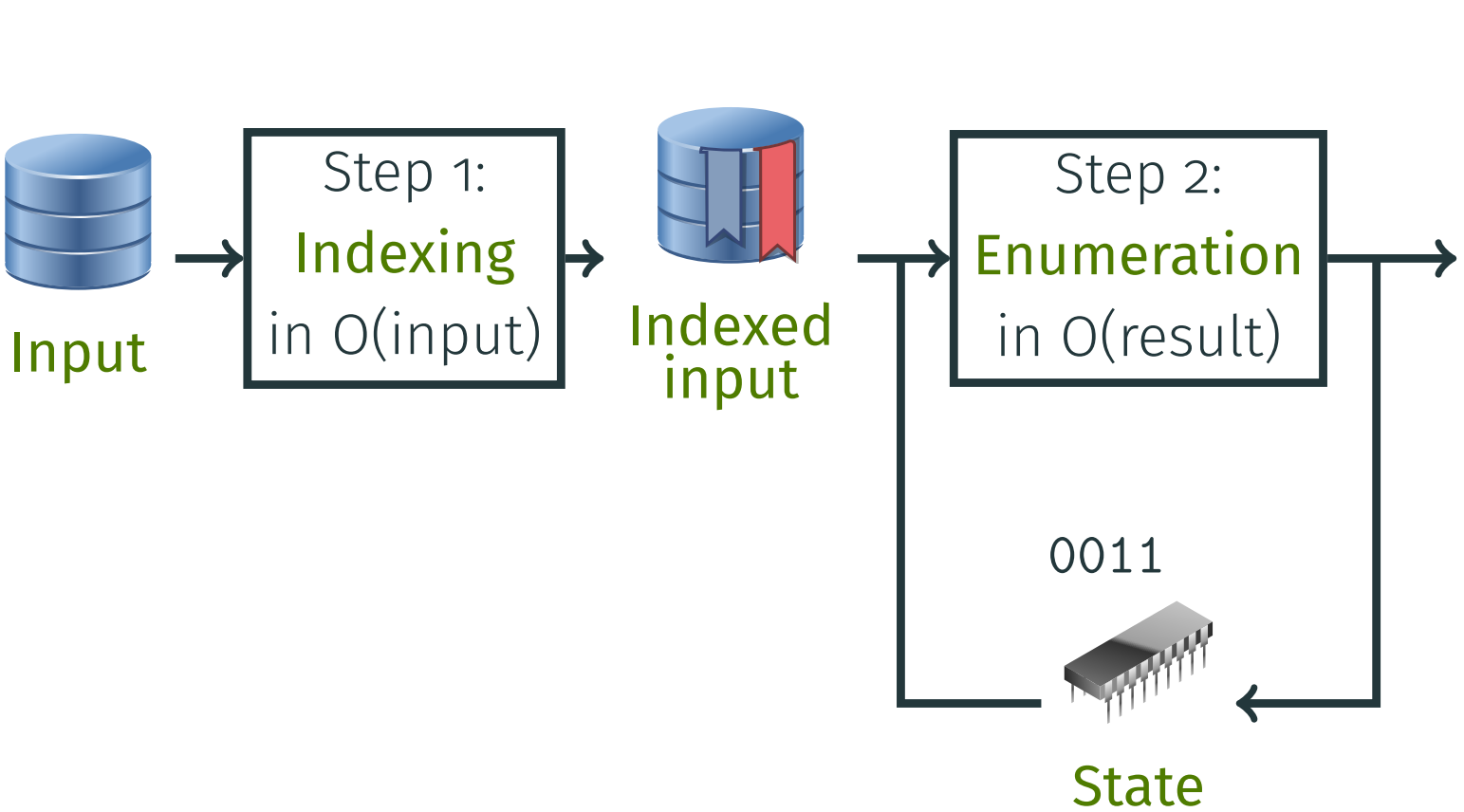
A	B	C
a	b	c

Results

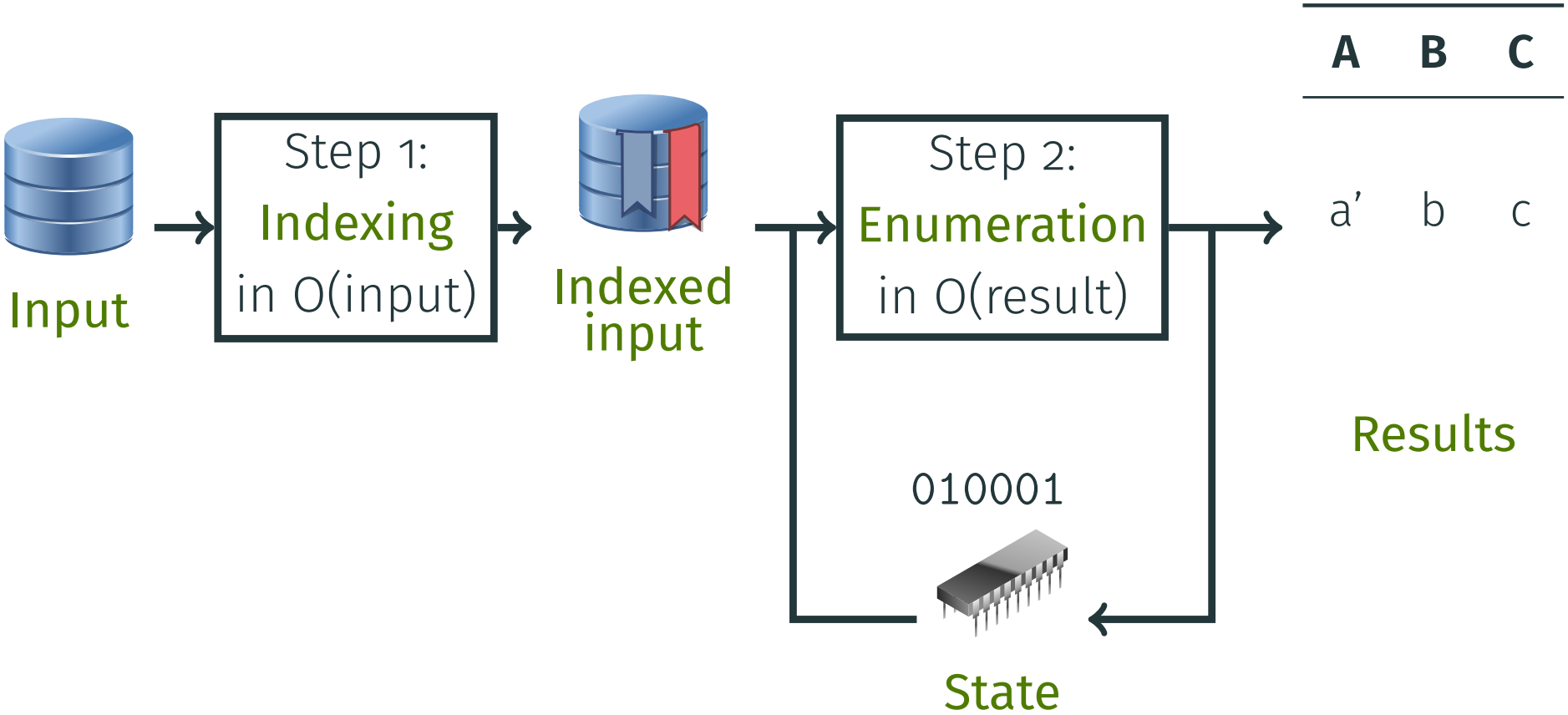
Formalizing an enumeration algorithm



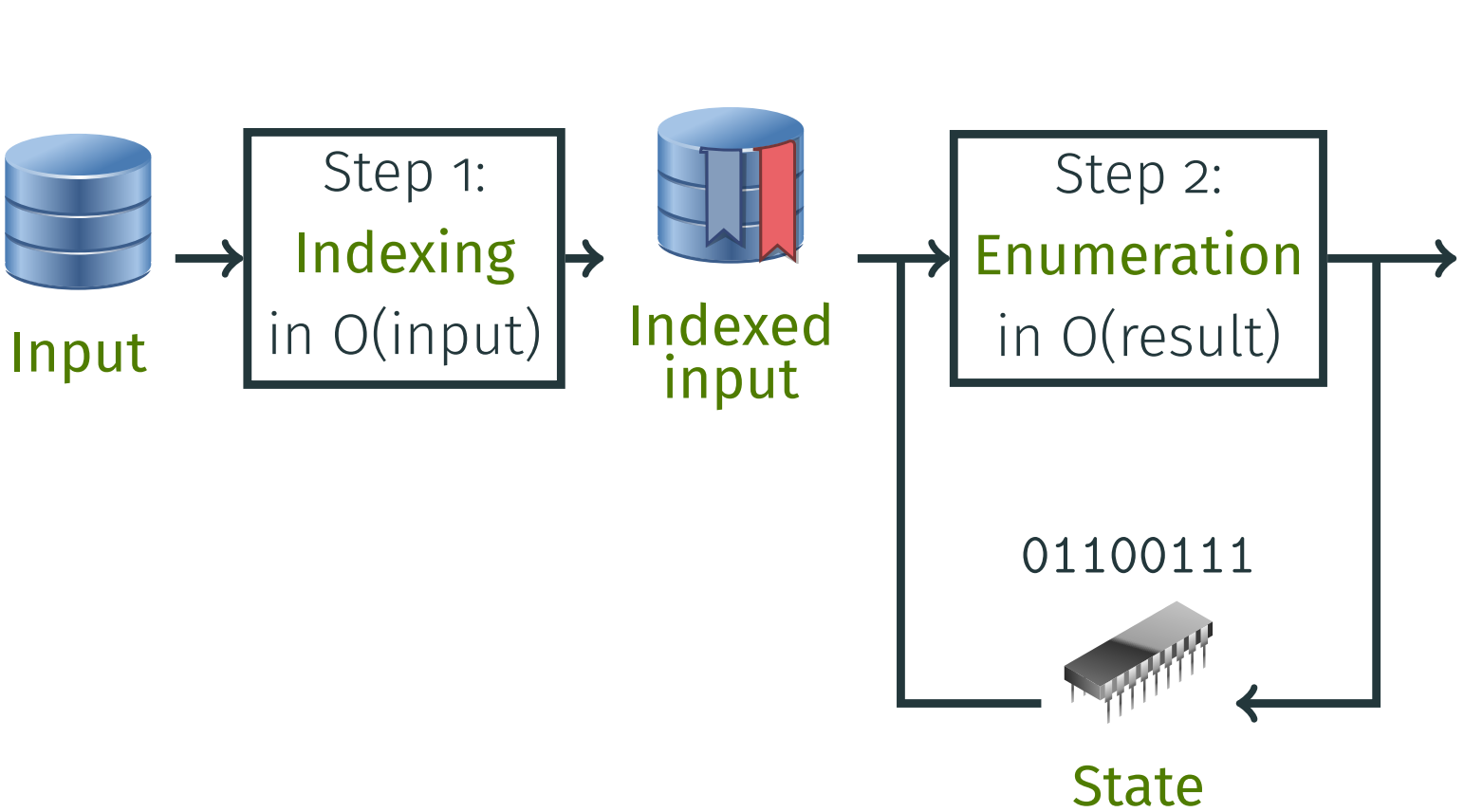
Formalizing an enumeration algorithm



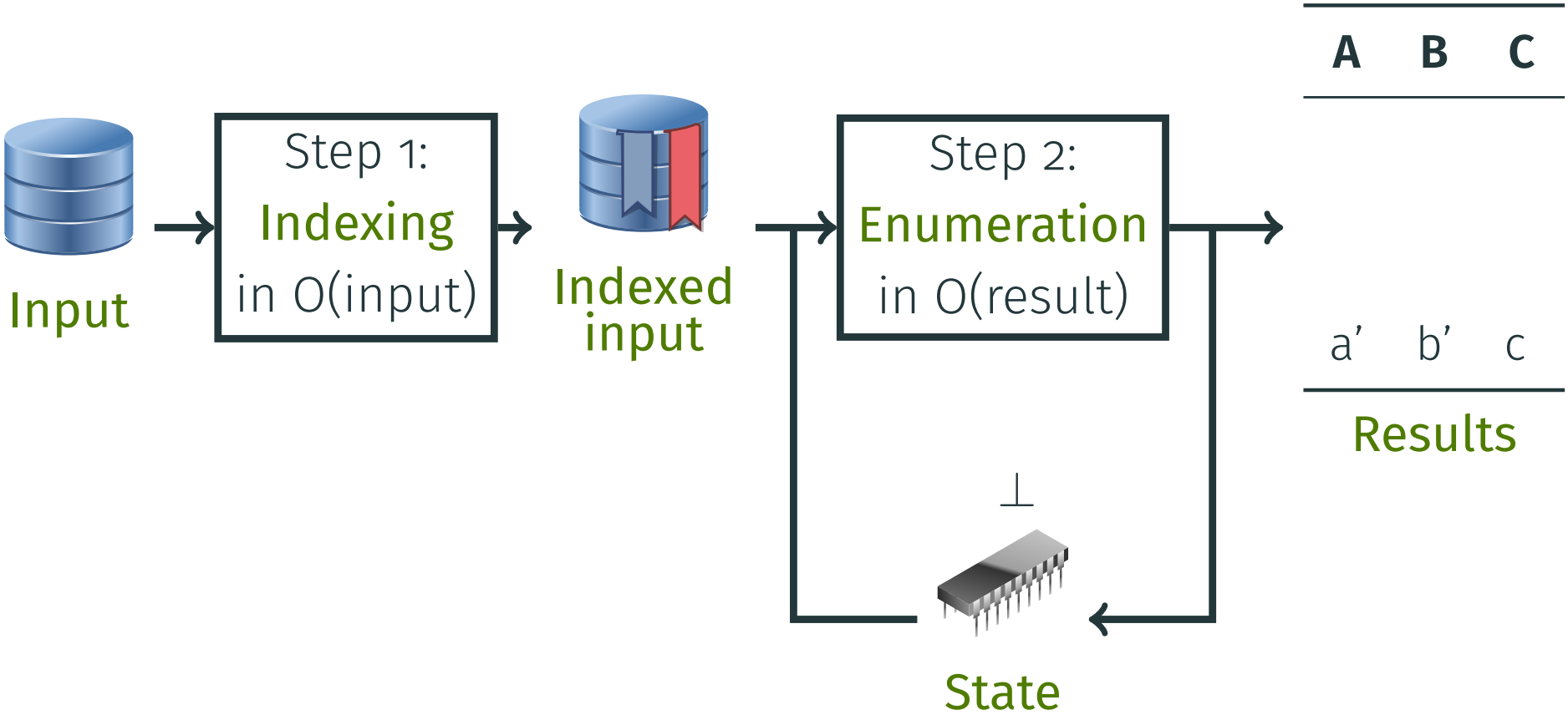
Formalizing an enumeration algorithm



Formalizing an enumeration algorithm



Formalizing an enumeration algorithm



Main results

- We are given: the tree T, a non-Boolean query Q in MSO
  $\rightarrow$ Example: $Q(x, y)$ asks for pairs of a blue node x and a pink node y

Main results

- We are given: the tree T, a non-Boolean query Q in MSO
  $\rightarrow$ Example: $Q(x, y)$ asks for pairs of a blue node x and a pink node y
- We want: enumerate the results of the query
  $\rightarrow$ Here, the pairs of a pink node and blue node

Main results

- We are given: the tree T, a non-Boolean query Q in MSO
  - $\rightarrow$ Example: $Q(x, y)$ asks for pairs of a blue node x and a pink node y
- We want: enumerate the results of the query
  - $\rightarrow$ Here, the pairs of a pink node and blue node

Theorem

*For any **fixed** MSO query Q, given a tree T,*
we can preprocess T in linear time in T and then
enumerate each result in linear time in the result

Main results

- We are given: the tree T, a non-Boolean query Q in MSO
  - $\rightarrow$ Example: $Q(x, y)$ asks for pairs of a blue node x and a pink node y
- We want: enumerate the results of the query
  - $\rightarrow$ Here, the pairs of a pink node and blue node

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

For any fixed MSO query Q, given a tree T,
we can preprocess T in linear time in T and then
enumerate each result in linear time in the result

This was already known, so what's new?

Main results

- We are given: the tree T, a non-Boolean query Q in MSO
  → Example: $Q(x, y)$ asks for pairs of a blue node x and a pink node y
- We want: enumerate the results of the query
  → Here, the pairs of a pink node and blue node

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

For any fixed MSO query Q, given a tree T,
we can preprocess T in linear time in T and then
enumerate each result in linear time in the result

This was already known, so what's new?

- Modular proof based on a notion of set circuits
- Tractable in combined complexity for Q given as an automaton
- Efficiently update the preprocessing when the tree changes

Structure of the talk

- Building a set circuit: given a tree T and automaton A, we can build a set circuit C that represents the results

Structure of the talk

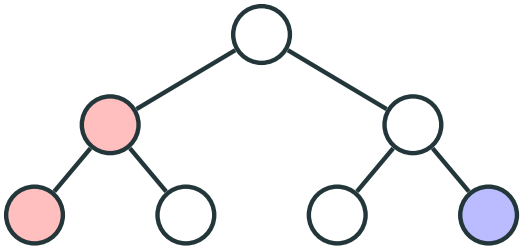
- Building a set circuit: given a tree T and automaton A, we can build a set circuit C that represents the results
- Enumeration on set circuits: given a set circuit C, we can enumerate efficiently the results that it captures (under some assumptions on C)

Structure of the talk

- Building a set circuit: given a tree T and automaton A, we can build a set circuit C that represents the results
- Enumeration on set circuits: given a set circuit C, we can enumerate efficiently the results that it captures (under some assumptions on C)
- New stuff:
  - Tractability in the automaton and application to text
  - Efficient updates of the index

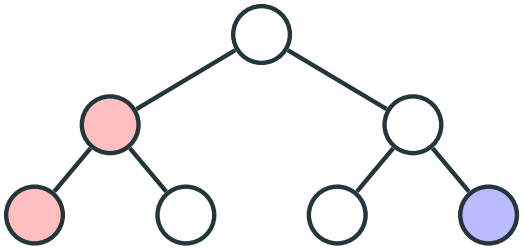
Building a set circuit

Monadic second-order logic (MSO)



- $P_{\bullet}(x)$ means "x is blue"; also $P_{\color{red}\bullet}(x)$, $P_{\circ}(x)$
- $x \to y$ means "x is the parent of y"

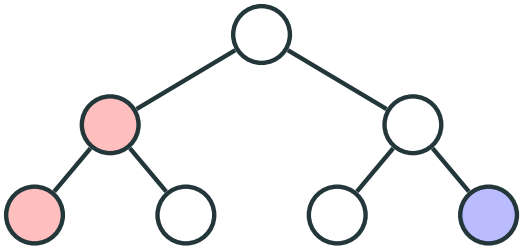
Monadic second-order logic (MSO)



- $P_{\bigcirc}(x)$ means "x is blue"; also $P_{\bigcirc}(x)$, $P_{\bigcirc}(x)$
- $x \to y$ means "x is the parent of y"

- Propositional logic: formulas with AND $\wedge$, OR $\vee$, NOT $\neg$
  - $P_{\bigcirc}(x) \wedge P_{\bigcirc}(y)$ means *"Node x is pink and node y is blue"*

Monadic second-order logic (MSO)

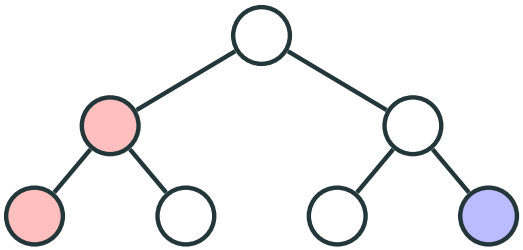


- $P_{\bullet}(x)$ means "x is blue"; also $P_{\bullet}(x)$, $P_{\circ}(x)$
- $x \to y$ means "x is the parent of y"

- Propositional logic: formulas with AND $\wedge$, OR $\vee$, NOT $\neg$
  - $P_{\bullet}(x) \wedge P_{\bullet}(y)$ means *"Node x is pink and node y is blue"*

- First-order logic: adds existential quantifier $\exists$ and universal quantifier $\forall$
  - $\exists x\, y\; P_{\bullet}(x) \wedge P_{\bullet}(y)$ means *"There is both a pink and a blue node"*

Monadic second-order logic (MSO)



- $P_{\color{blue}\bigcirc}(x)$ means "x is blue"; also $P_{\color{red}\bigcirc}(x)$, $P_{\bigcirc}(x)$
- $x \to y$ means "x is the parent of y"

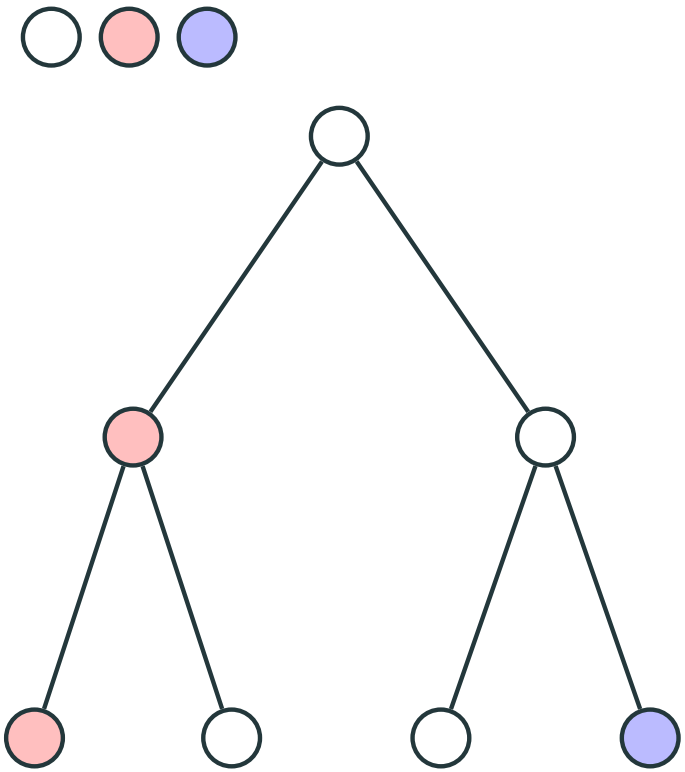
- Propositional logic: formulas with AND $\wedge$, OR $\vee$, NOT $\neg$
  - $P_{\color{red}\bigcirc}(x) \wedge P_{\color{blue}\bigcirc}(y)$ means *"Node x is pink and node y is blue"*

- First-order logic: adds existential quantifier $\exists$ and universal quantifier $\forall$
  - $\exists x\, y\; P_{\color{red}\bigcirc}(x) \wedge P_{\color{blue}\bigcirc}(y)$ means *"There is both a pink and a blue node"*

- Monadic second-order logic (MSO): adds quantifiers over sets
  - $\exists S\; \forall x\; S(x)$ means *"there is a set S containing every element x"*
  - Can express transitive closure $x \to^* y$, i.e., *"x is an ancestor of y"*
  - $\forall x\, P_{\color{red}\bigcirc}(x) \Rightarrow \exists y\, P_{\color{blue}\bigcirc}(y) \wedge x \to^* y$
    means *"There is a blue node below every pink node"*

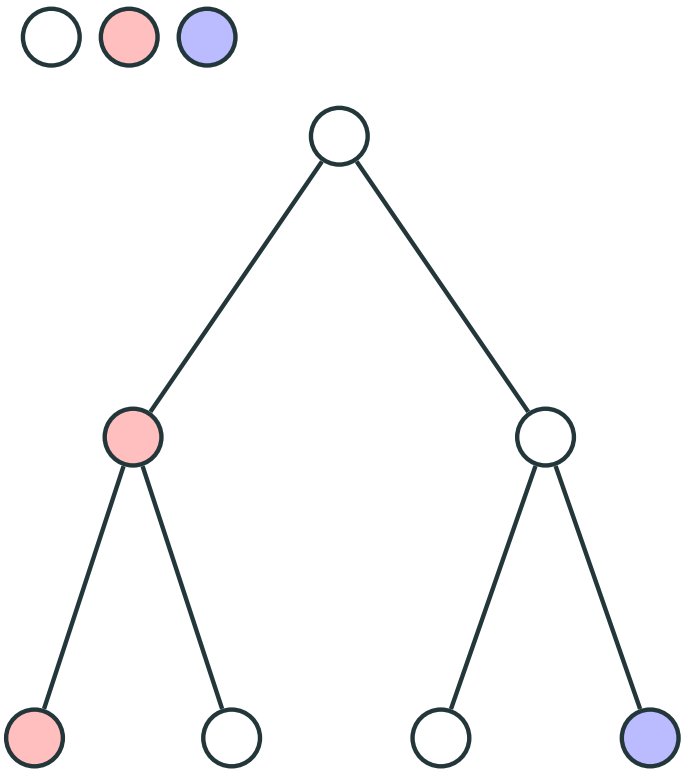
Tree automata

Tree alphabet:



Tree automata

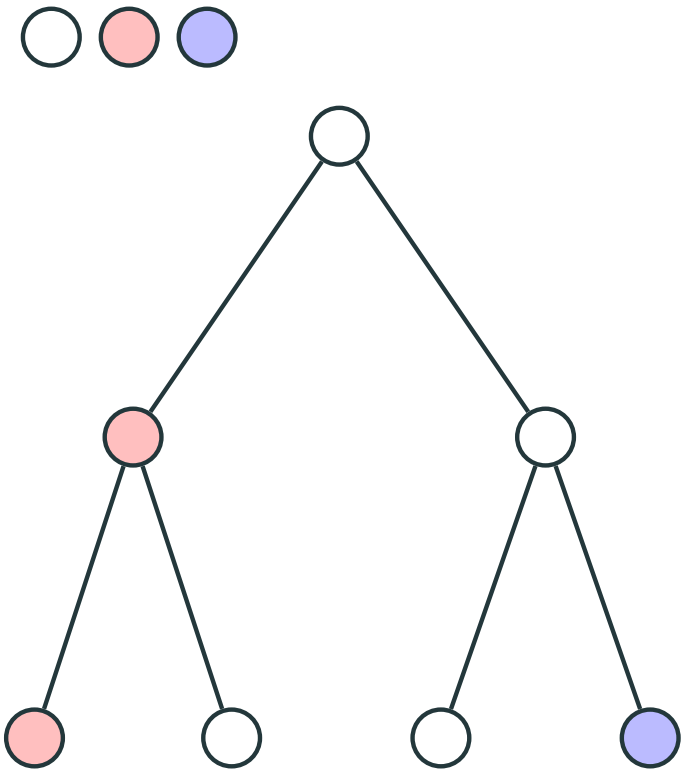
Tree alphabet:



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

Tree automata

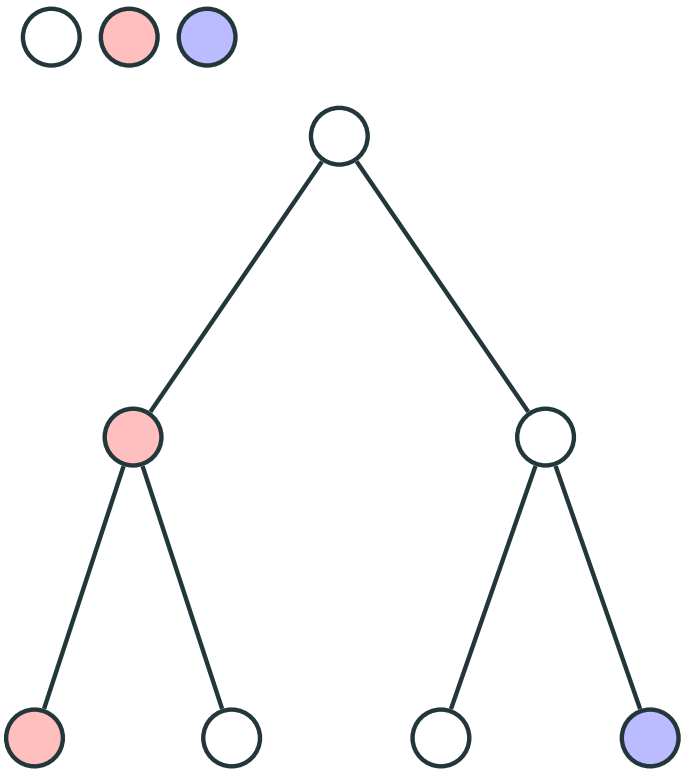
Tree alphabet:



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

Tree automata

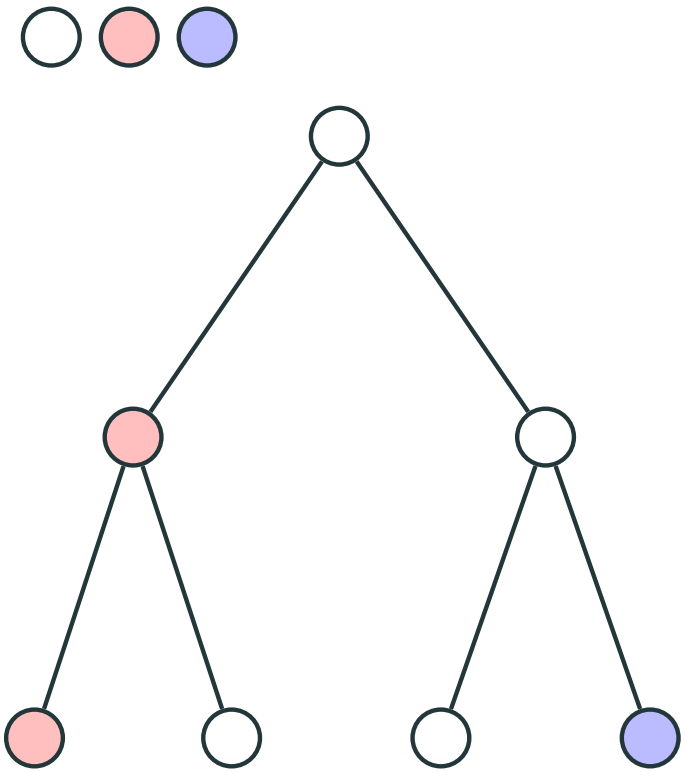
Tree alphabet:



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

Tree automata

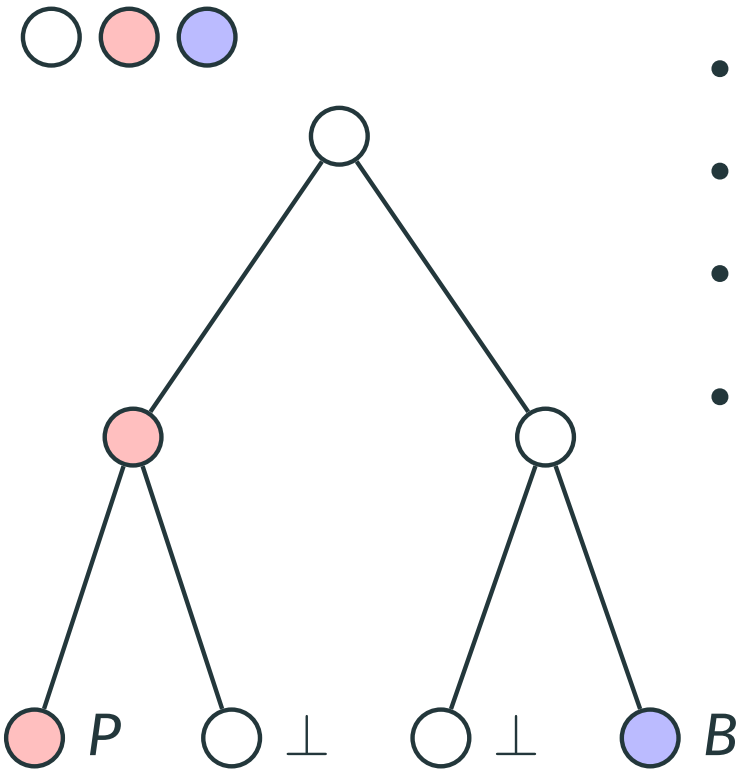
Tree alphabet:



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

Tree automata

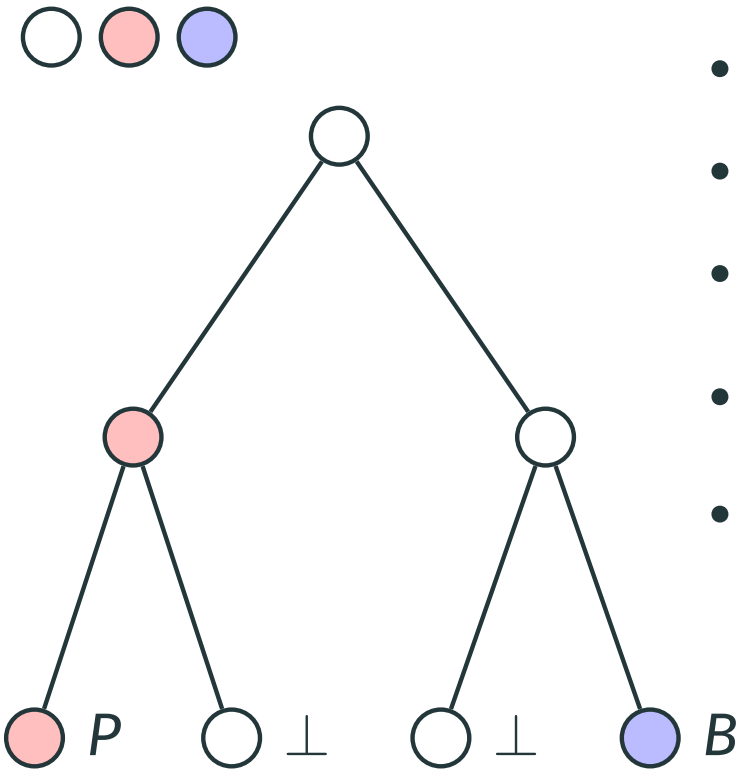
Tree alphabet:



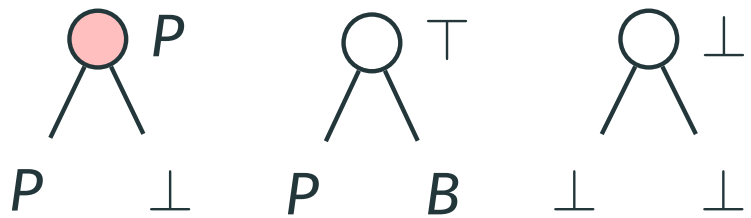
- Bottom-up deterministic tree automaton
- *"Is there both a pink and a blue node?"*
- States: $\{\bot, B, P, \top\}$
- Final states: $\{\top\}$
- Initial function: $\bot$ P B

Tree automata

Tree alphabet:

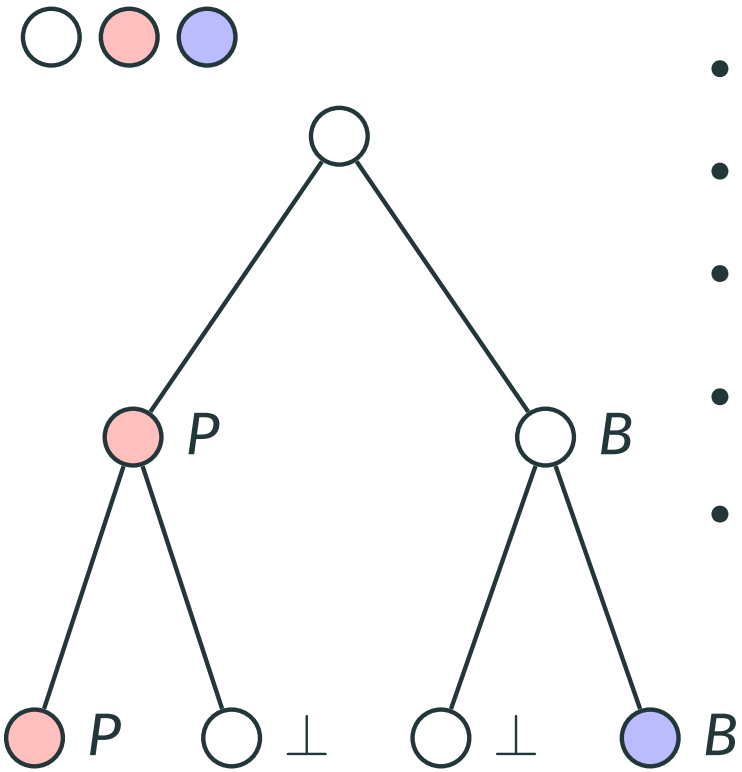


- Bottom-up deterministic tree automaton
- *"Is there both a pink and a blue node?"*
- States: $\{\bot, B, P, \top\}$
- Final states: $\{\top\}$
- Initial function: ○ $\bot$ ● P ● B
- Transitions (examples):

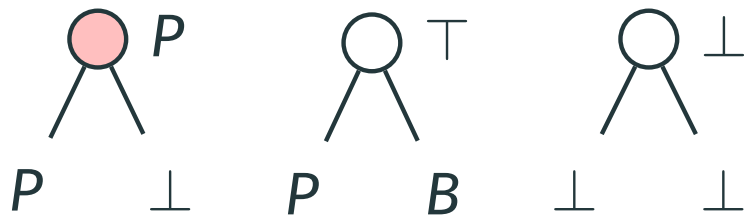


Tree automata

Tree alphabet:

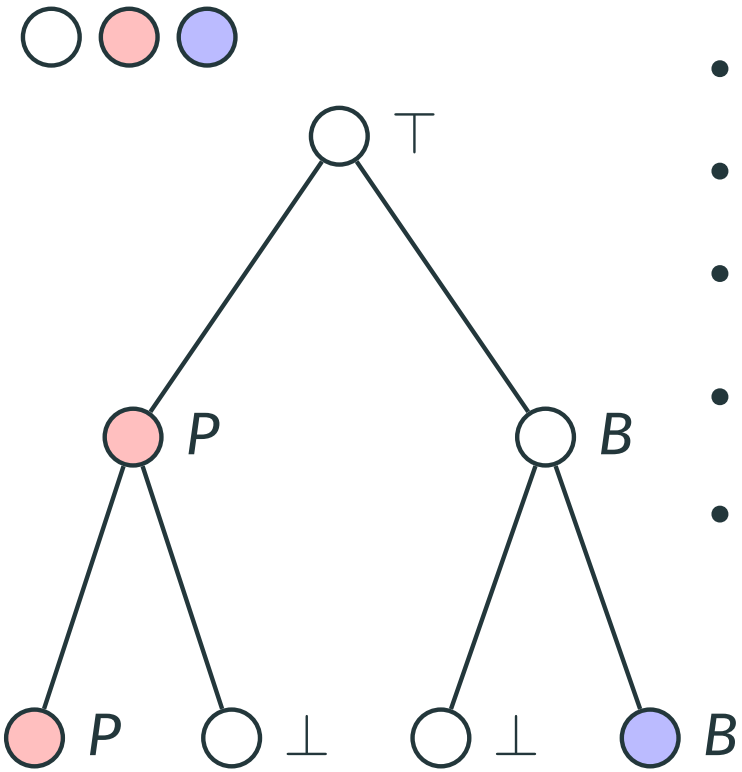


- Bottom-up deterministic **tree automaton**
- *"Is there both a pink and a blue node?"*
- **States:** $\{\bot, B, P, \top\}$
- **Final states:** $\{\top\}$
- **Initial function:** ○ $\bot$ ● P ● B
- **Transitions** (examples):

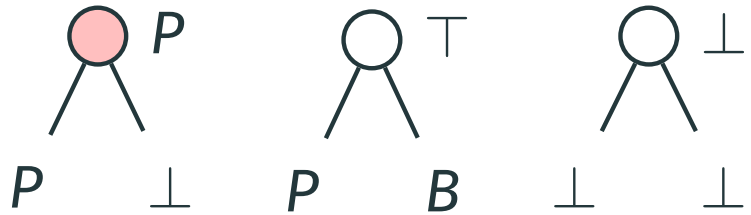


Tree automata

Tree alphabet:



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



Boolean MSO query evaluation via automata

Theorem [Thatcher and Wright, 1968]

MSO and tree automata have the same expressive power on trees

→ *Given a Boolean MSO query, we can compute a tree automaton that accepts precisely the trees on which the query holds*

Boolean MSO query evaluation via automata

Theorem [Thatcher and Wright, 1968]

MSO and tree automata have the same expressive power on trees

→ *Given a Boolean MSO query, we can compute a tree automaton that accepts precisely the trees on which the query holds*

→ *Complexity (in the query) is generally nonelementary*

Boolean MSO query evaluation via automata

Theorem [Thatcher and Wright, 1968]

MSO and tree automata have the same expressive power on trees

→ *Given a Boolean MSO query, we can compute a tree automaton that accepts precisely the trees on which the query holds*

→ *Complexity (in the query) is generally nonelementary*

Corollary

Evaluating a Boolean MSO query on a tree is in linear time in the tree

A small hack for non-Boolean queries

Remember our problem on non-Boolean queries:

Data: A tree T

A small hack for non-Boolean queries

Remember our problem on non-Boolean queries:

Data: A tree T

Query: A non-Boolean MSO $Q(\mathbf{x})$ formula

"What are the pairs of a pink and blue node?"

A small hack for non-Boolean queries

Remember our problem on non-Boolean queries:

Data: A tree T

Query: A non-Boolean MSO $Q(\mathbf{x})$ formula

"What are the pairs of a pink and blue node?"

Result: All the $\mathbf{a}$ such that $Q(\mathbf{a})$ holds

A small hack for non-Boolean queries

Data: A tree T

Query: A non-Boolean MSO $Q(\mathbf{x})$ formula

"What are the pairs of a pink and blue node?"

Result: All the $\mathbf{a}$ such that $Q(\mathbf{a})$ holds

For technical reasons it will be simpler to:

- Write the choice for variables as colors on the tree

A small hack for non-Boolean queries

Data: ~~A tree T~~

A tree T with more colors to select nodes

Query: A non-Boolean MSO $Q(\mathbf{x})$ formula

"What are the pairs of a pink and blue node?"

Result: All the $\mathbf{a}$ such that $Q(\mathbf{a})$ holds

For technical reasons it will be simpler to:

- Write the choice for variables as colors on the tree

A small hack for non-Boolean queries

Data: ~~A tree T~~

A tree T with more colors to select nodes

Query: A non-Boolean MSO $Q(\mathbf{x})$ formula

"What are the pairs of a pink and blue node?"

Result: All the $\mathbf{a}$ such that $Q(\mathbf{a})$ holds

For technical reasons it will be simpler to:

- Write the choice for variables as colors on the tree
- Replace the non-Boolean query Q by a Boolean query Q'

A small hack for non-Boolean queries

Data: ~~A tree T~~

A tree T with more colors to select nodes

Query: ~~A non-Boolean MSO $Q(\mathbf{x})$ formula~~

A Boolean MSO formula Q'

~~*"What are the pairs of a pink and blue node?"*~~

"Are the two selected nodes pink and blue?"

Result: All the $\mathbf{a}$ such that $Q(\mathbf{a})$ holds

For technical reasons it will be simpler to:

- Write the choice for variables as colors on the tree
- Replace the non-Boolean query Q by a Boolean query Q'

A small hack for non-Boolean queries

Data: ~~A tree T~~
A tree T with more colors to select nodes

Query: ~~A non-Boolean MSO $Q(\mathbf{x})$ formula~~
A Boolean MSO formula Q'

~~*"What are the pairs of a pink and blue node?"*~~
"Are the two selected nodes pink and blue?"

Result: All the **a** such that $Q(\mathbf{a})$ holds

For technical reasons it will be simpler to:

- Write the choice for variables as colors on the tree
- Replace the non-Boolean query Q by a Boolean query Q'
- Enumerate the ways to color the tree

A small hack for non-Boolean queries

Data: ~~A tree T~~
A tree T with more colors to select nodes

Query: ~~A non-Boolean MSO $Q(\mathbf{x})$ formula~~
A Boolean MSO formula Q'

~~*"What are the pairs of a pink and blue node?"*~~
"Are the two selected nodes pink and blue?"

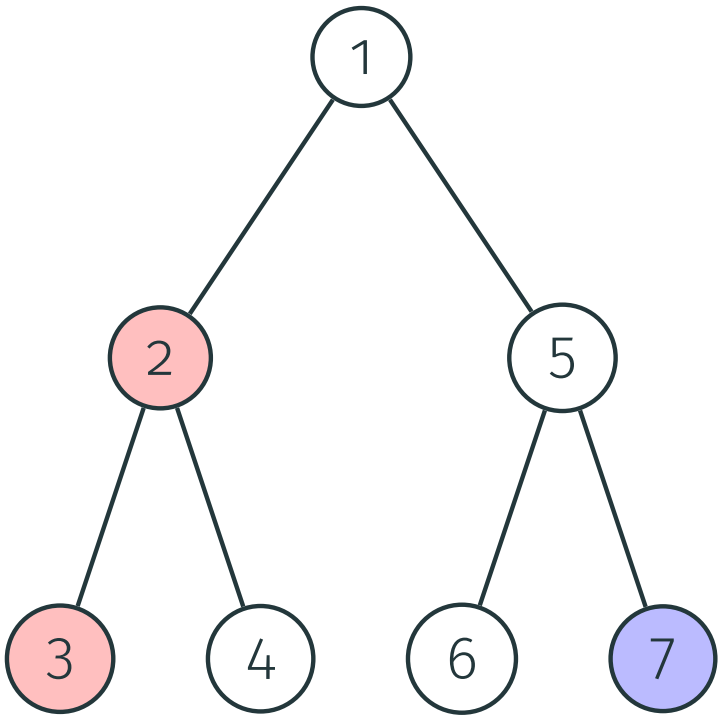
Result: ~~All the **a** such that $Q(\mathbf{a})$ holds~~
All the ways ν to color T such that Q' holds on $\nu(T)$

For technical reasons it will be simpler to:

- Write the choice for variables as colors on the tree
- Replace the non-Boolean query Q by a Boolean query Q'
- Enumerate the ways to color the tree

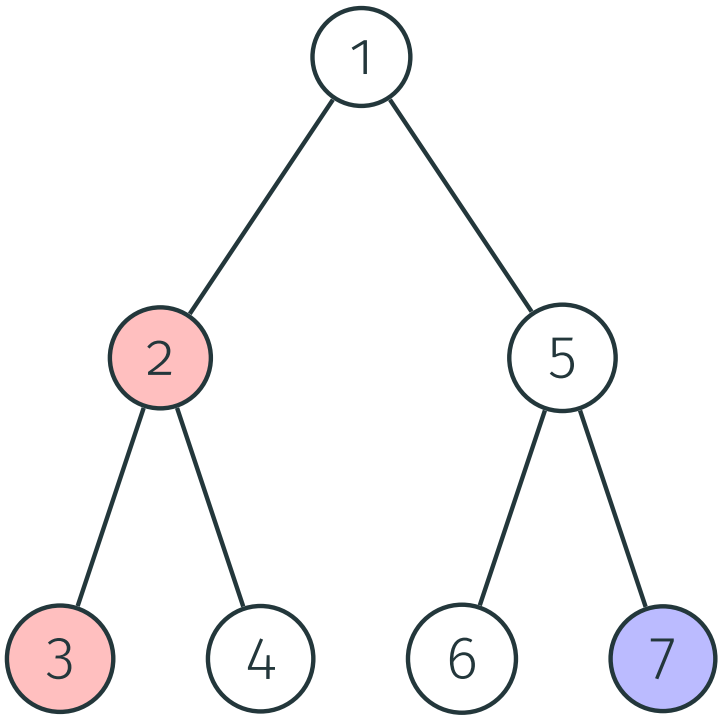
Uncertain trees

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



Uncertain trees

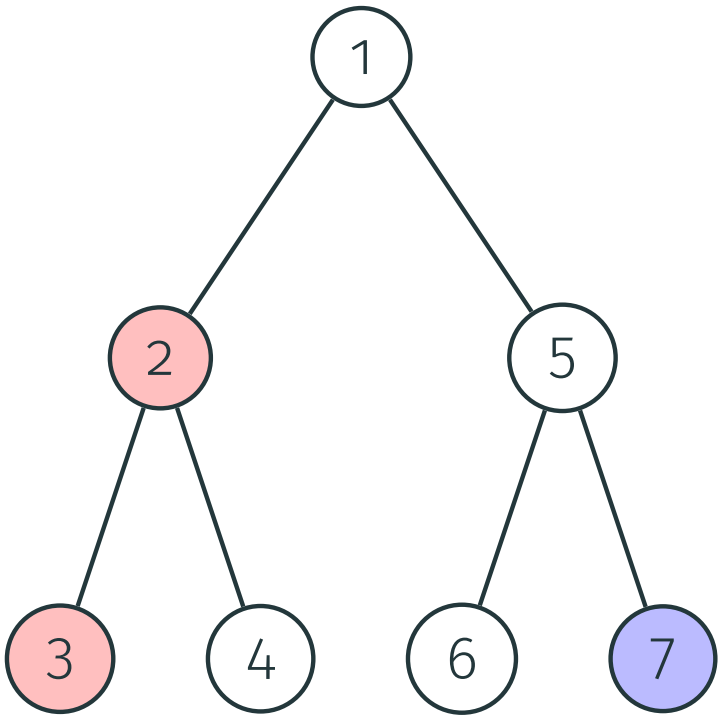
Now: Boolean query on a tree where the color of nodes is **uncertain**



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

Uncertain trees

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

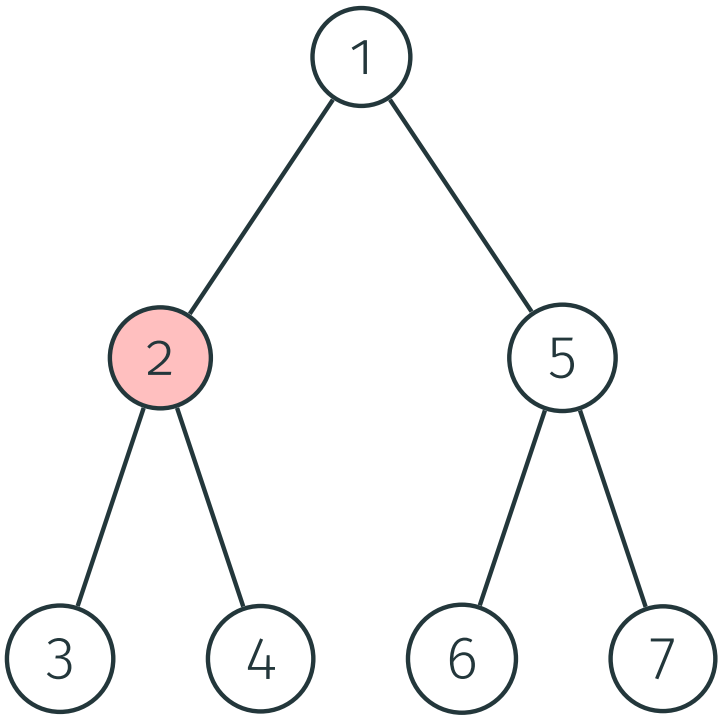


A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

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

Uncertain trees

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

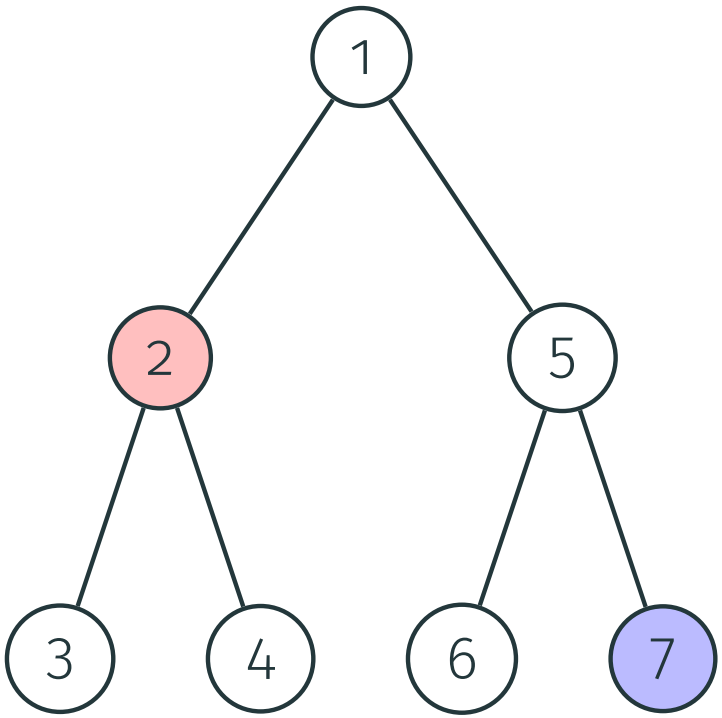


A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

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

Uncertain trees

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

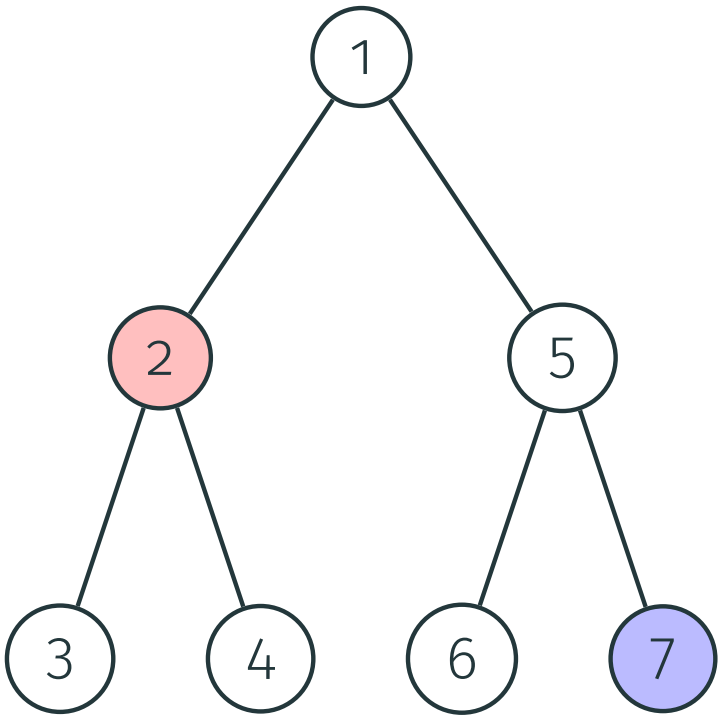


A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

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

Uncertain trees

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



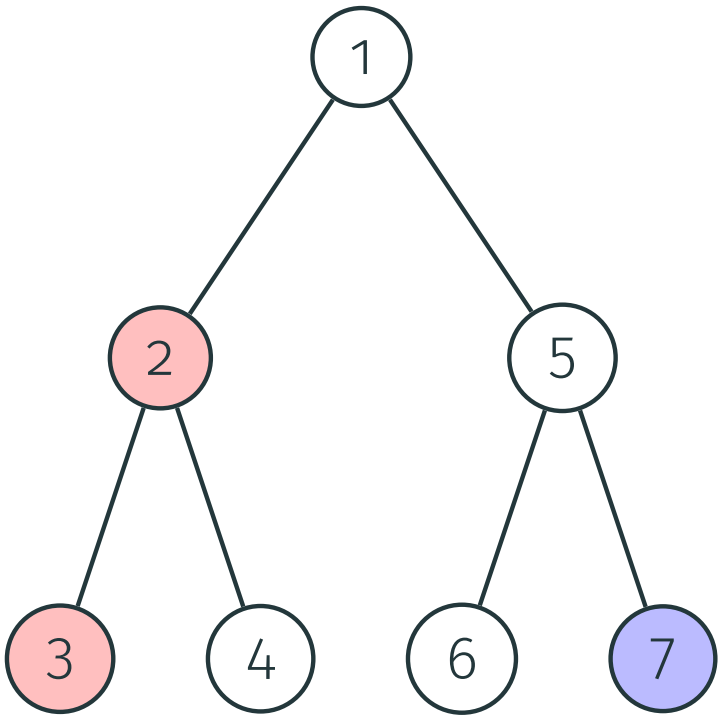
A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

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

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

Uncertain trees

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



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

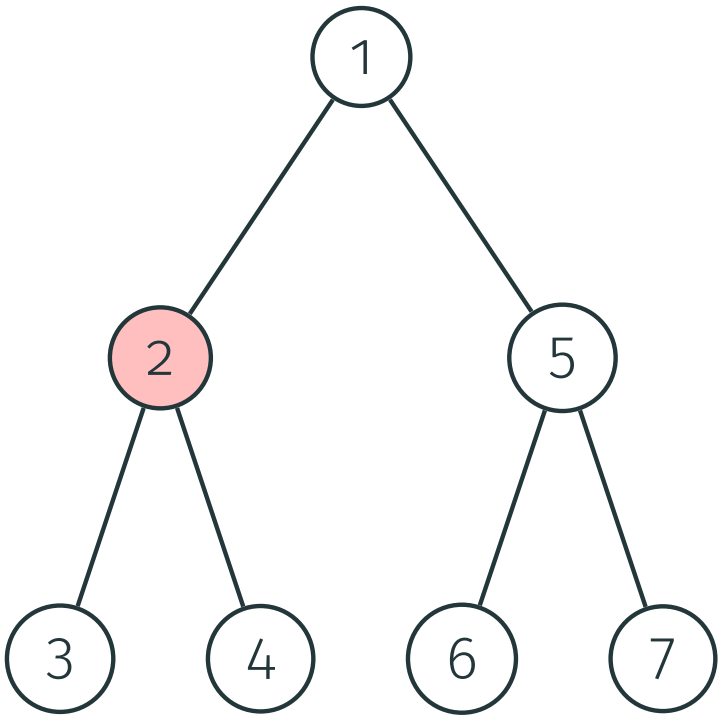
Valuation: $\{2, 3, 7 \mapsto 1, \quad * \mapsto 0\}$

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

The tree automaton A **accepts**

Uncertain trees

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



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

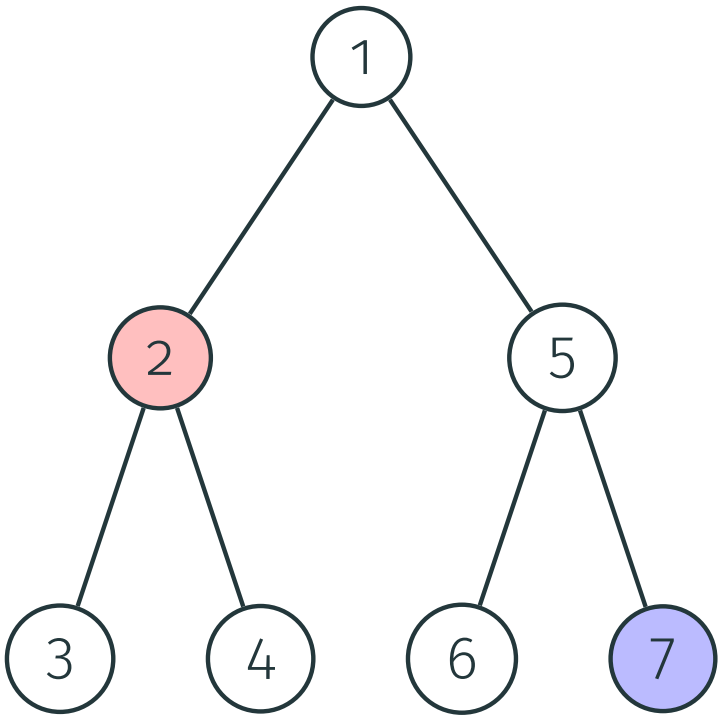
Valuation: $\{2 \mapsto 1, \quad * \mapsto 0\}$

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

The tree automaton A **rejects**

Uncertain trees

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



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

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

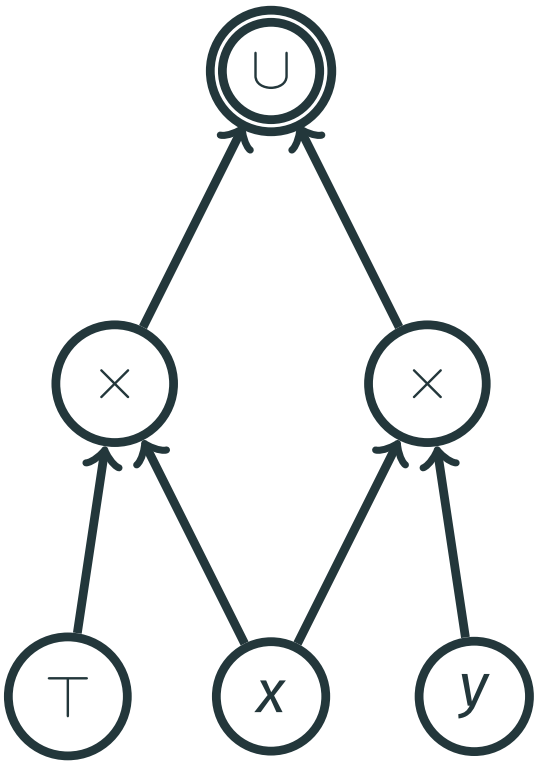
A: *"Is there both a pink and a blue node?"*

The tree automaton A **accepts**

$\rightarrow$ The **results** that we want to enumerate are all **valuations** of T that make A accept

Set circuits

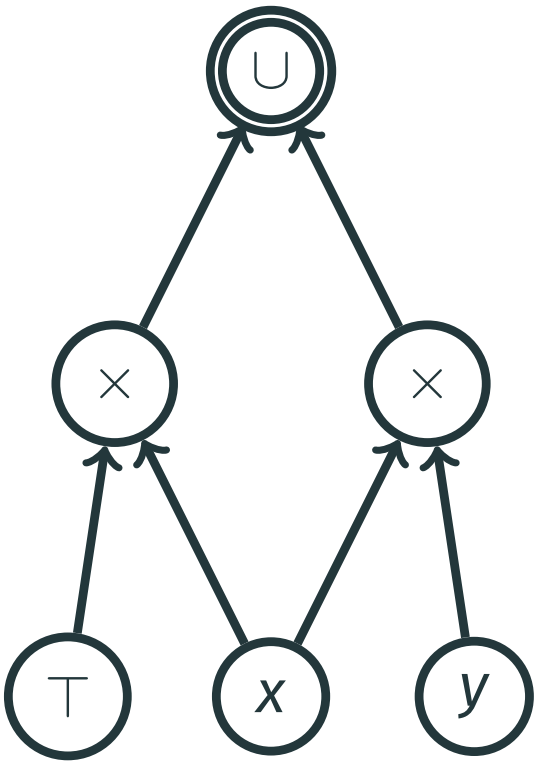
We want to represent these results as a set circuit:



- Directed acyclic graph of gates

Set circuits

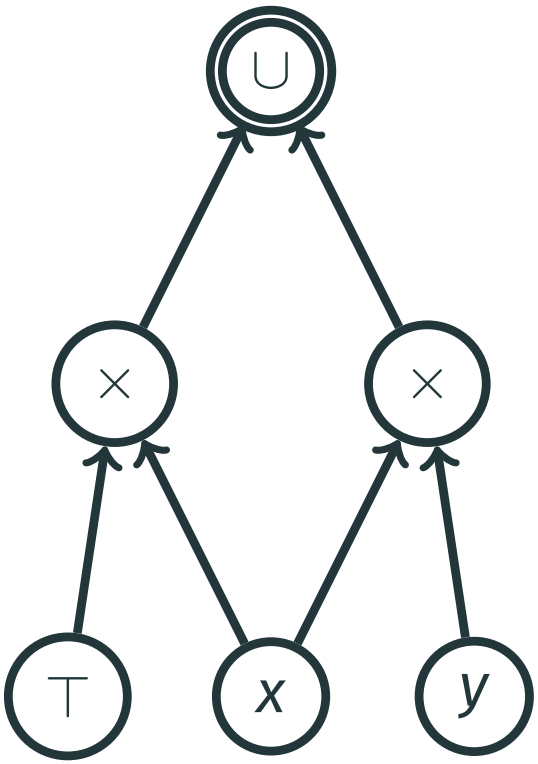
We want to represent these results as a set circuit:



- Directed acyclic graph of gates
- Output gate: 

Set circuits

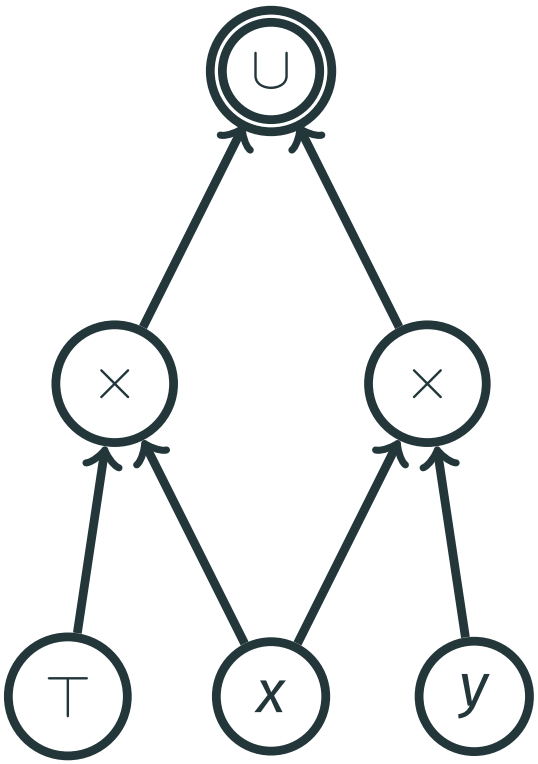
We want to represent these results as a set circuit:



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

Set circuits

We want to represent these results as a set circuit:

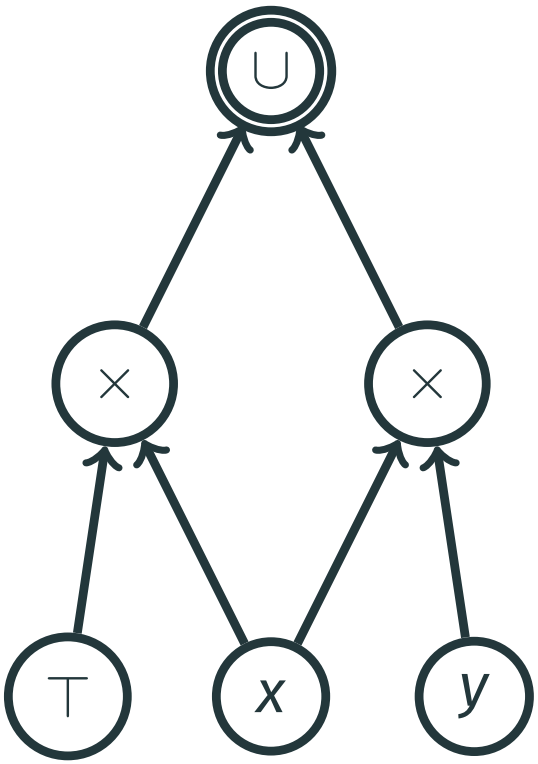


- Directed acyclic graph of gates
- Output gate: 
- Variable gates: 

- Constant gates: 

Set circuits

We want to represent these results as a set circuit:

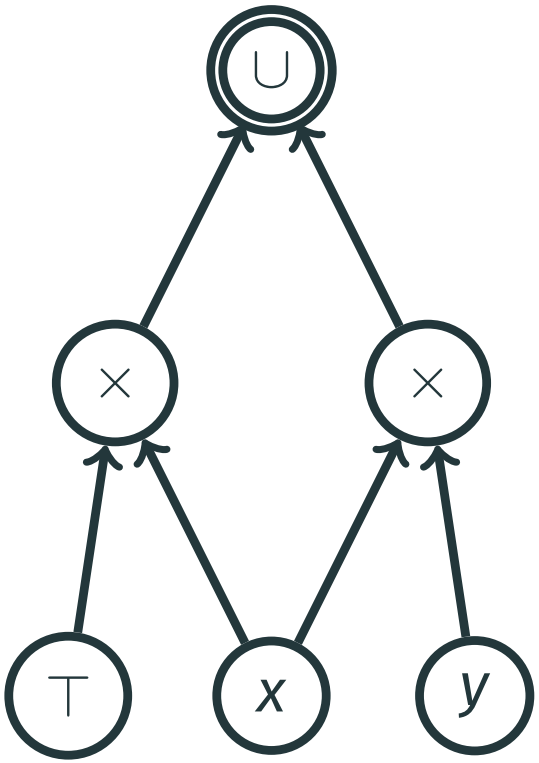


- Directed acyclic graph of gates
- Output gate: 
- Variable gates: 

- Constant gates: 

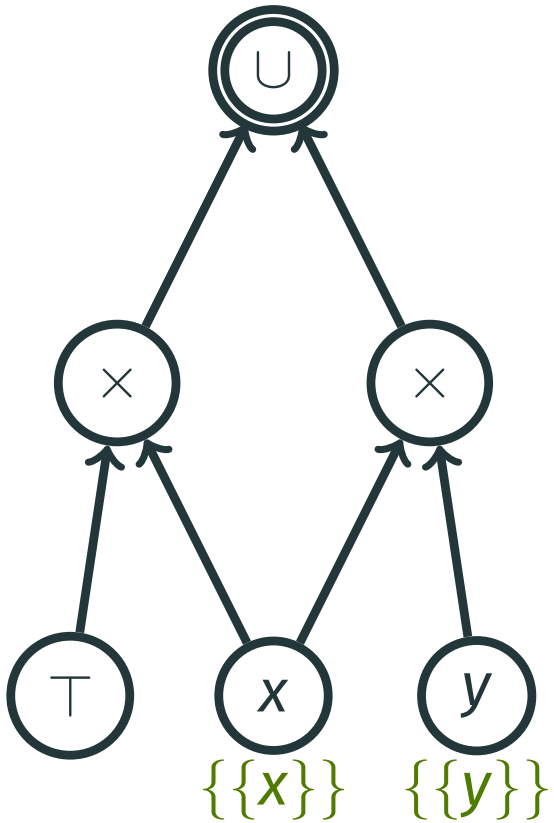
- Internal gates: 

Semantics of set circuits



Every gate g captures a set $S(g)$

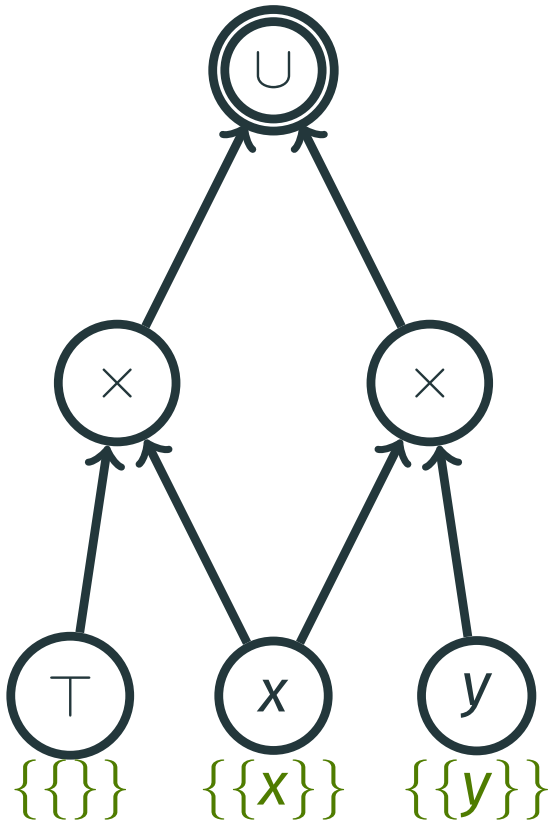
Semantics of set circuits



Every gate g captures a set $S(g)$

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

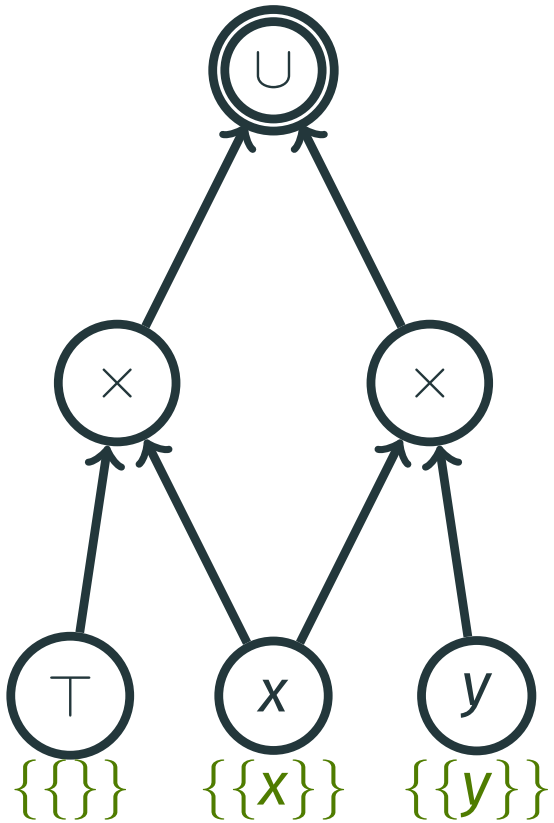
Semantics of set circuits



Every gate g **captures** a set $S(g)$

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

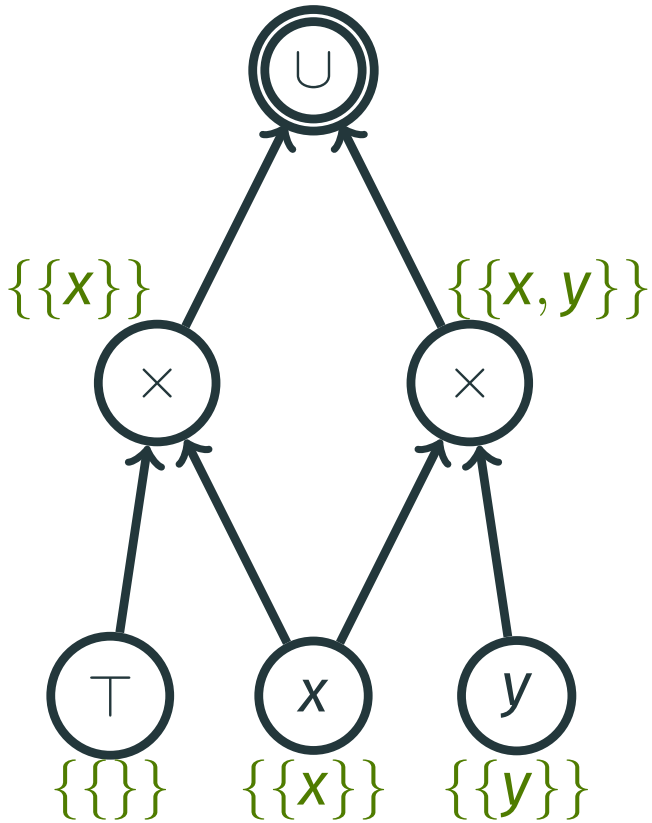
Semantics of set circuits



Every gate g **captures** a set $S(g)$

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

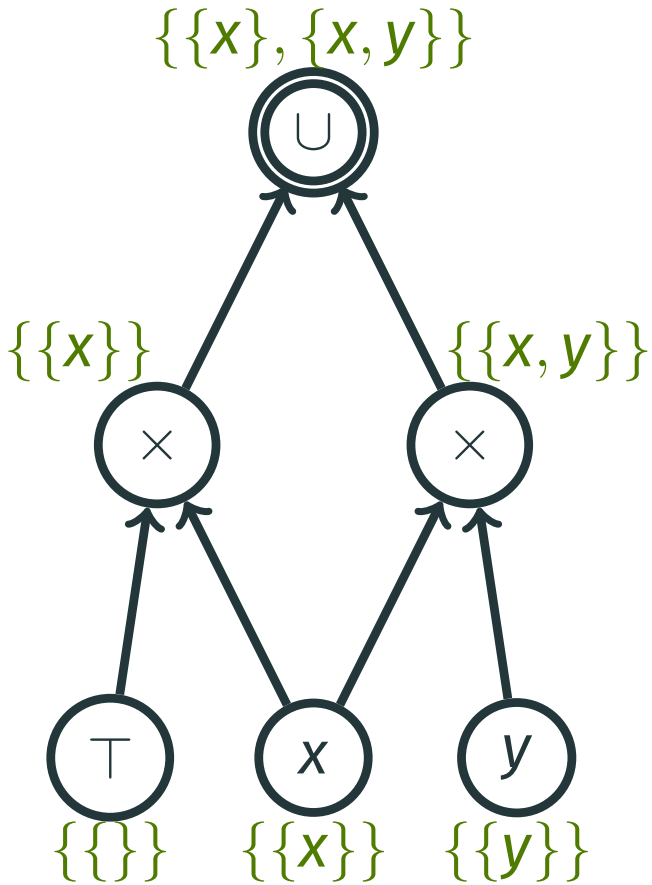
Semantics of set circuits



Every gate g **captures** a set $S(g)$

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

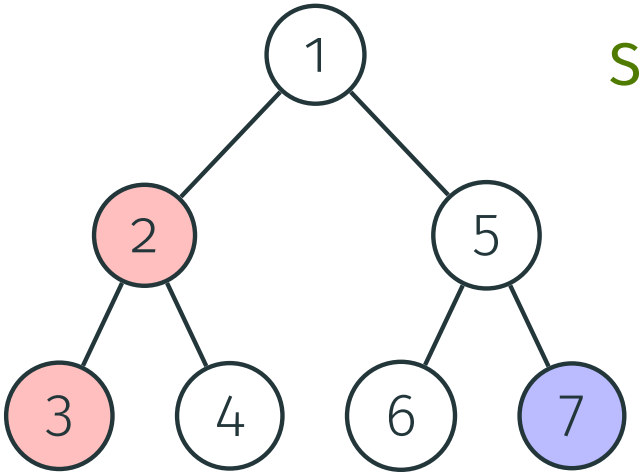
Semantics of set circuits



Every gate g **captures** a set $S(g)$

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

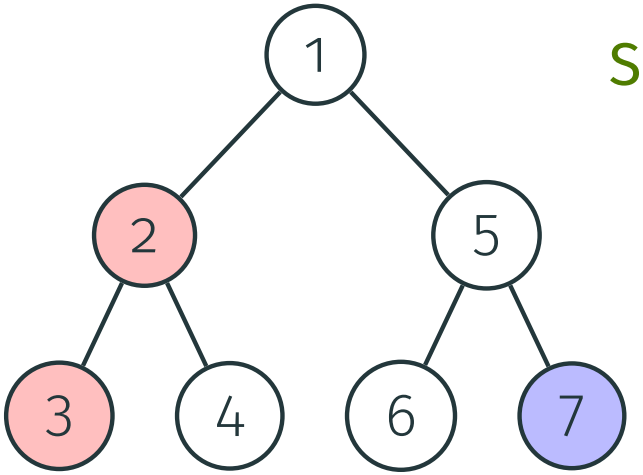
Set circuit for a query on a tree



Set circuit for automaton A on uncertain tree T:

- Variable gates: nodes of T

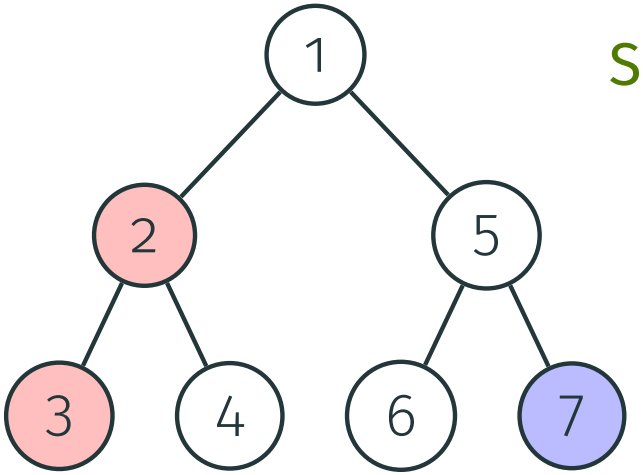
Set circuit for a query on a tree



Set circuit for automaton A on uncertain tree T:

- Variable gates: nodes of T
- Condition: Let ν be a valuation of T, then A accepts $\nu(T)$ iff the set $\mathrm{S}(g_o)$ of the output gate g_o contains $\{n \in T \mid \nu(n) = 1\}$.

Set circuit for a query on a tree

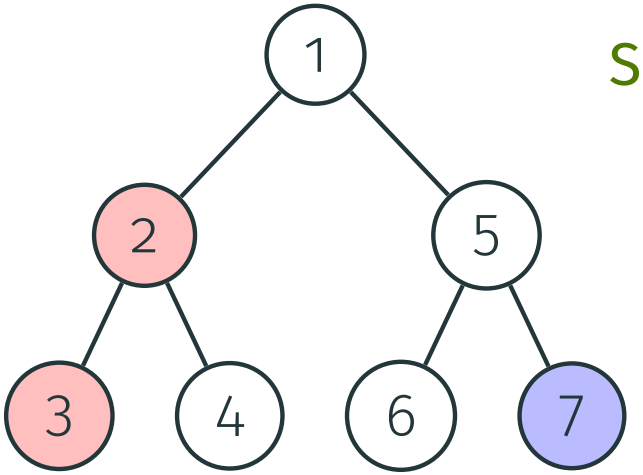


Set circuit for automaton A on uncertain tree T:

- Variable gates: nodes of T
- Condition: Let ν be a valuation of T, then A accepts $\nu(T)$ iff the set $\mathrm{S}(g_o)$ of the output gate g_o contains $\{n \in T \mid \nu(n) = 1\}$.

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

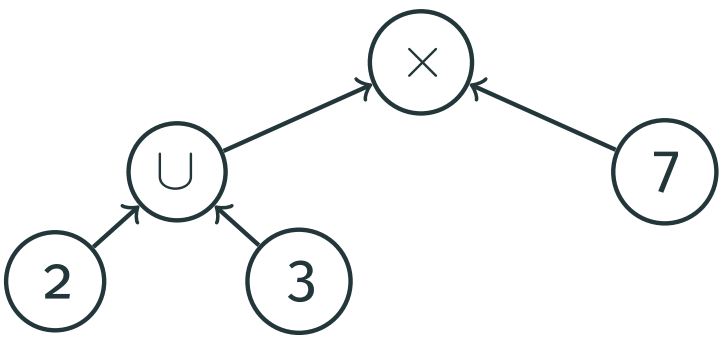
Set circuit for a query on a tree



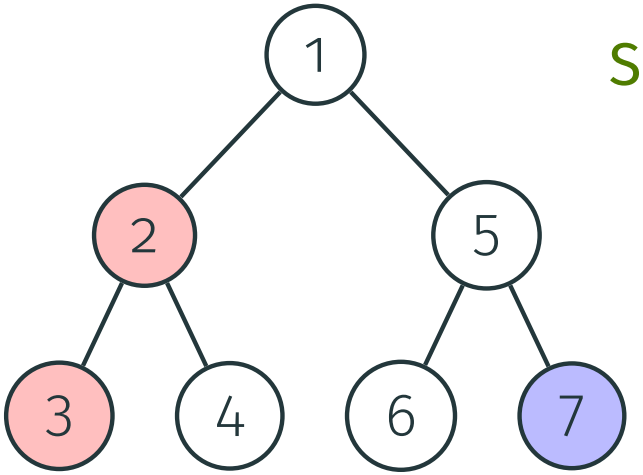
Set circuit for automaton A on uncertain tree T:

- Variable gates: nodes of T
- Condition: Let ν be a valuation of T, then A accepts $\nu(T)$ iff the set $\mathrm{S}(g_o)$ of the output gate g_o contains $\{n \in T \mid \nu(n) = 1\}$.

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



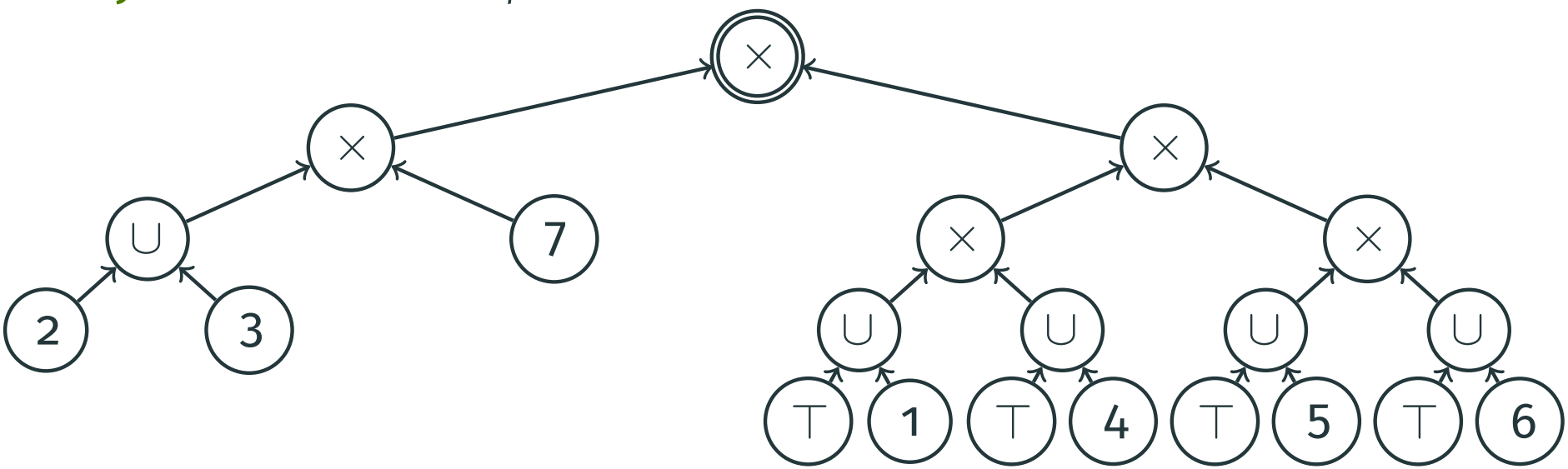
Set circuit for a query on a tree



Set circuit for automaton A on uncertain tree T:

- Variable gates: nodes of T
- Condition: Let ν be a valuation of T, then A accepts $\nu(T)$ iff the set $\mathrm{S}(g_o)$ of the output gate g_o contains $\{n \in T \mid \nu(n) = 1\}$.

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



Building set circuits

Theorem

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

Building set circuits

Theorem

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

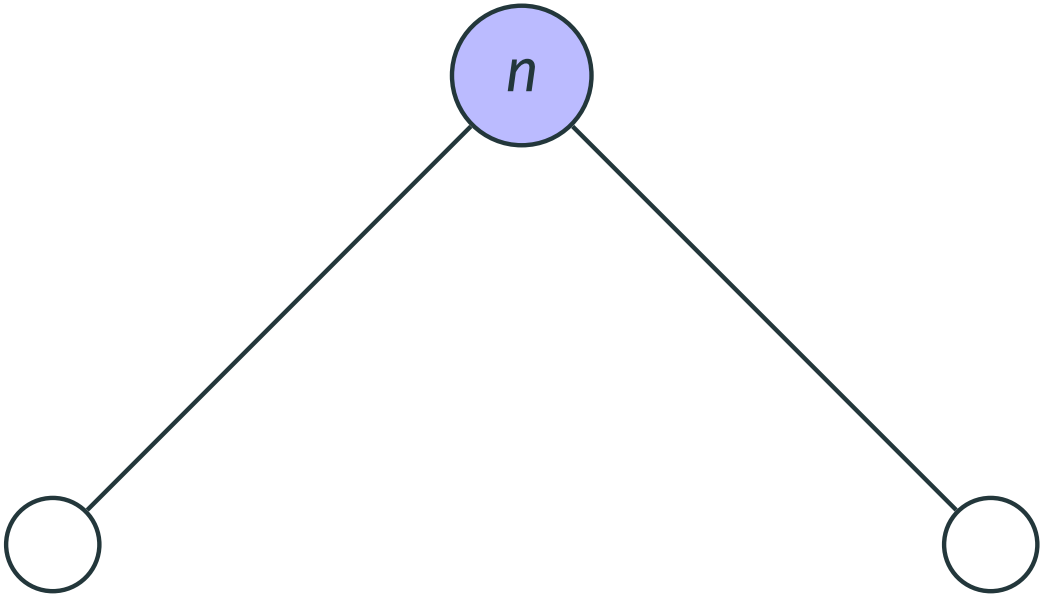
- Alphabet: 
- Automaton: *"Is there both a pink and a blue node?"*
- States: $\{\bot, B, P, \top\}$
- Final: $\{\top\}$
- Transitions:

Building set circuits

Theorem

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

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

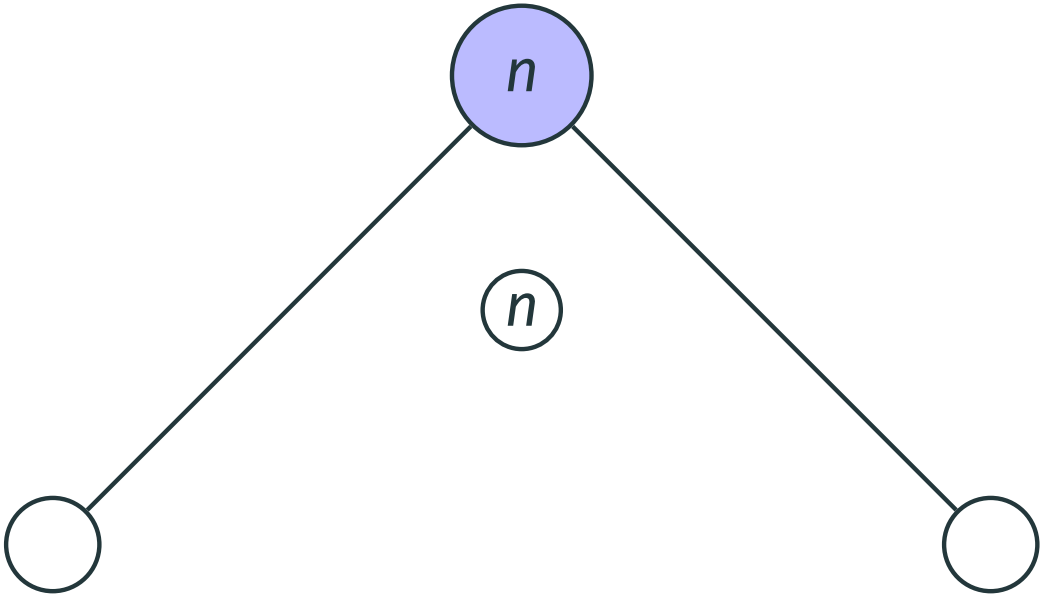


Building set circuits

Theorem

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

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

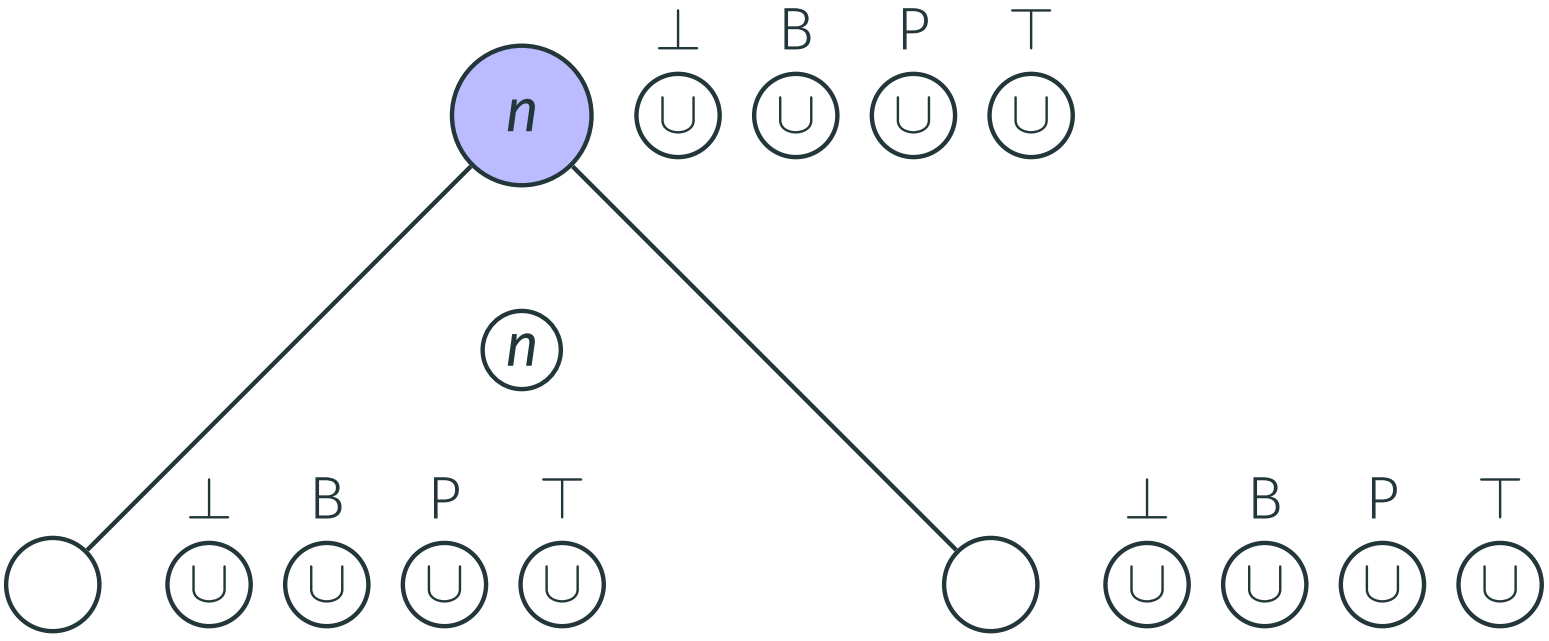


Building set circuits

Theorem

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

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

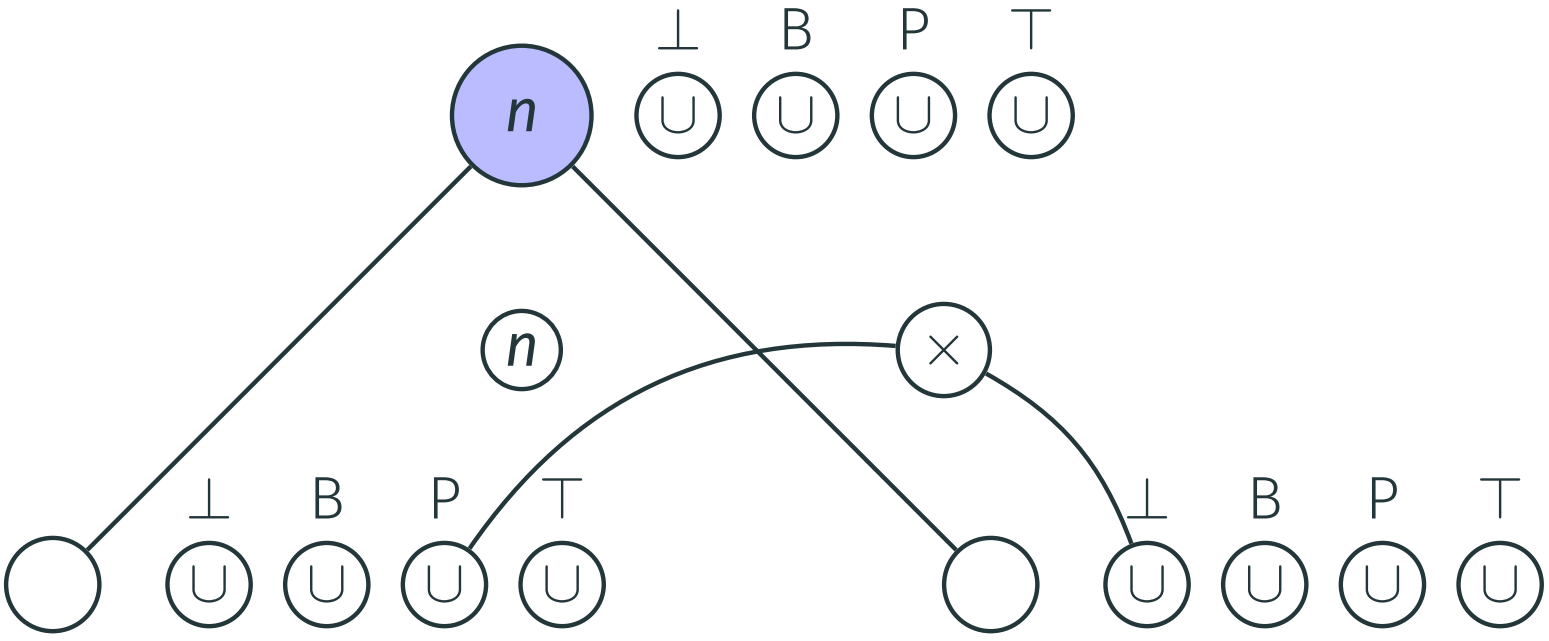


Building set circuits

Theorem

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

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

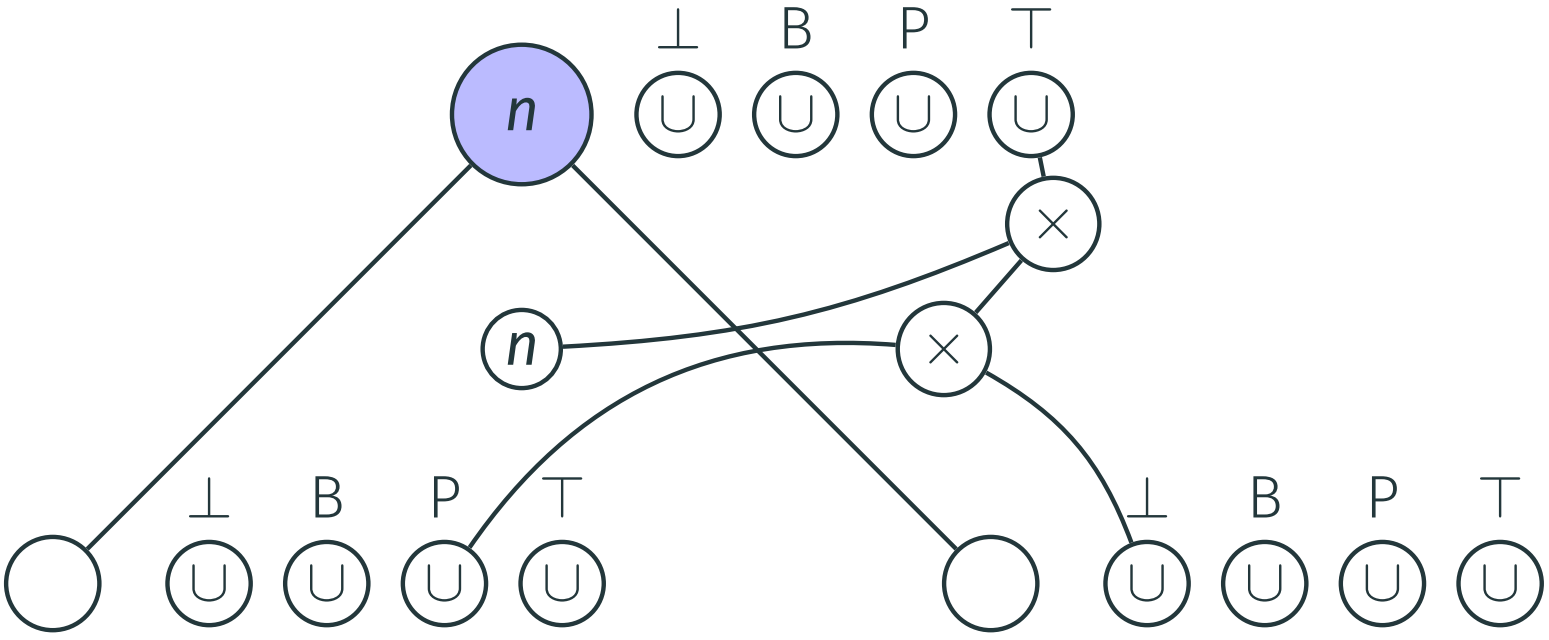


Building set circuits

Theorem

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

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

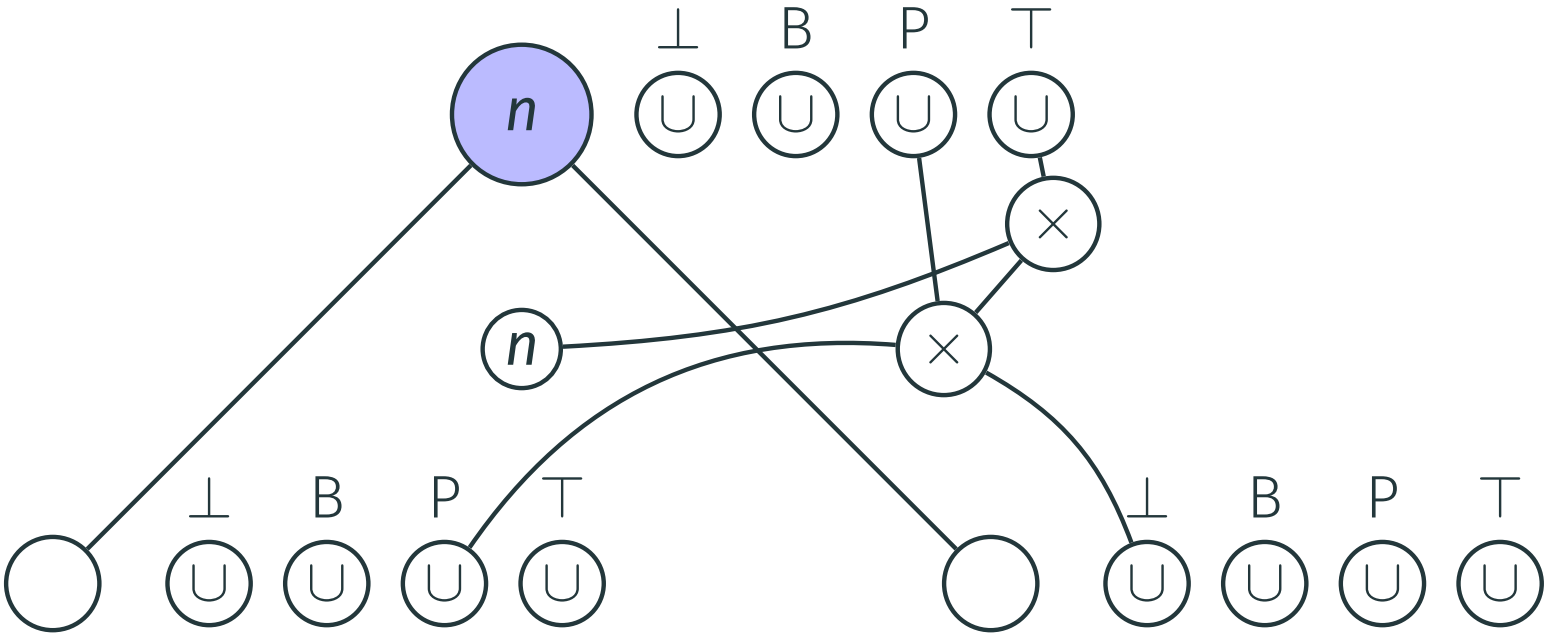


Building set circuits

Theorem

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

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



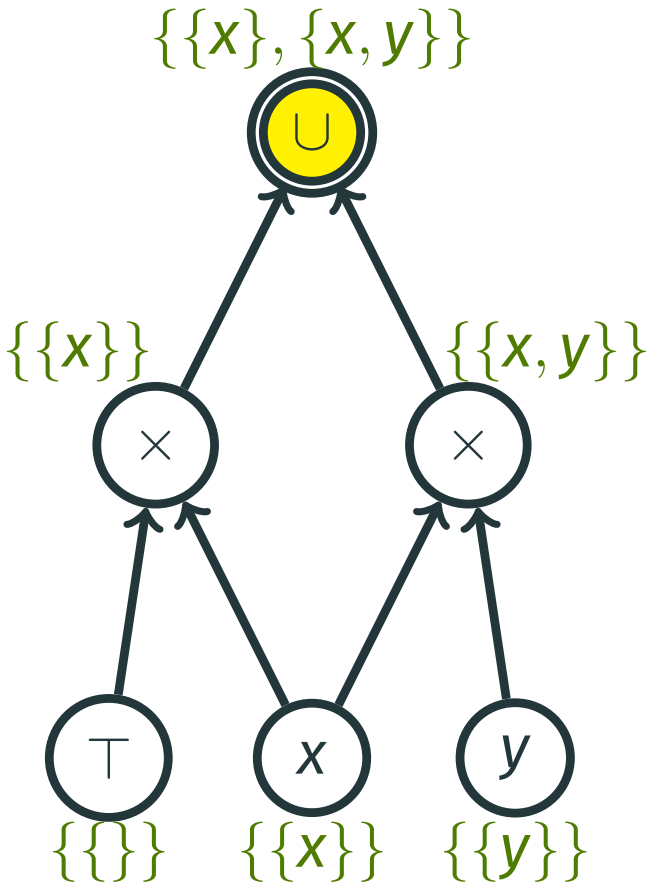
Set circuit restrictions

To have efficient enumeration, we need restrictions on the set circuit:

d-DNNF set circuit:

- ∪ are all deterministic:

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



Set circuit restrictions

To have **efficient enumeration**, we need **restrictions** on the set circuit:

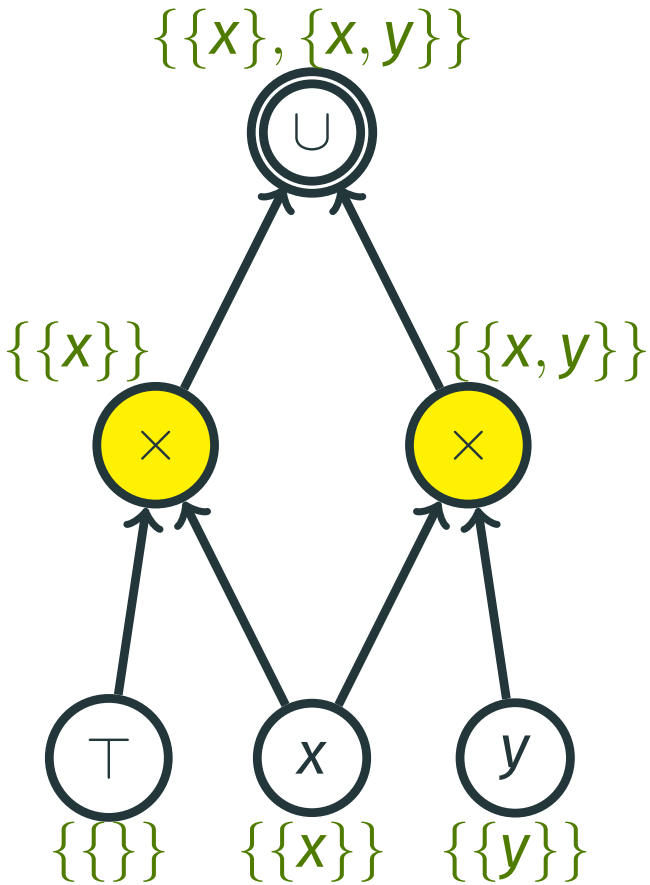
d-DNNF set circuit:

- ∪ are all **deterministic**:

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

- × are all **decomposable**:

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

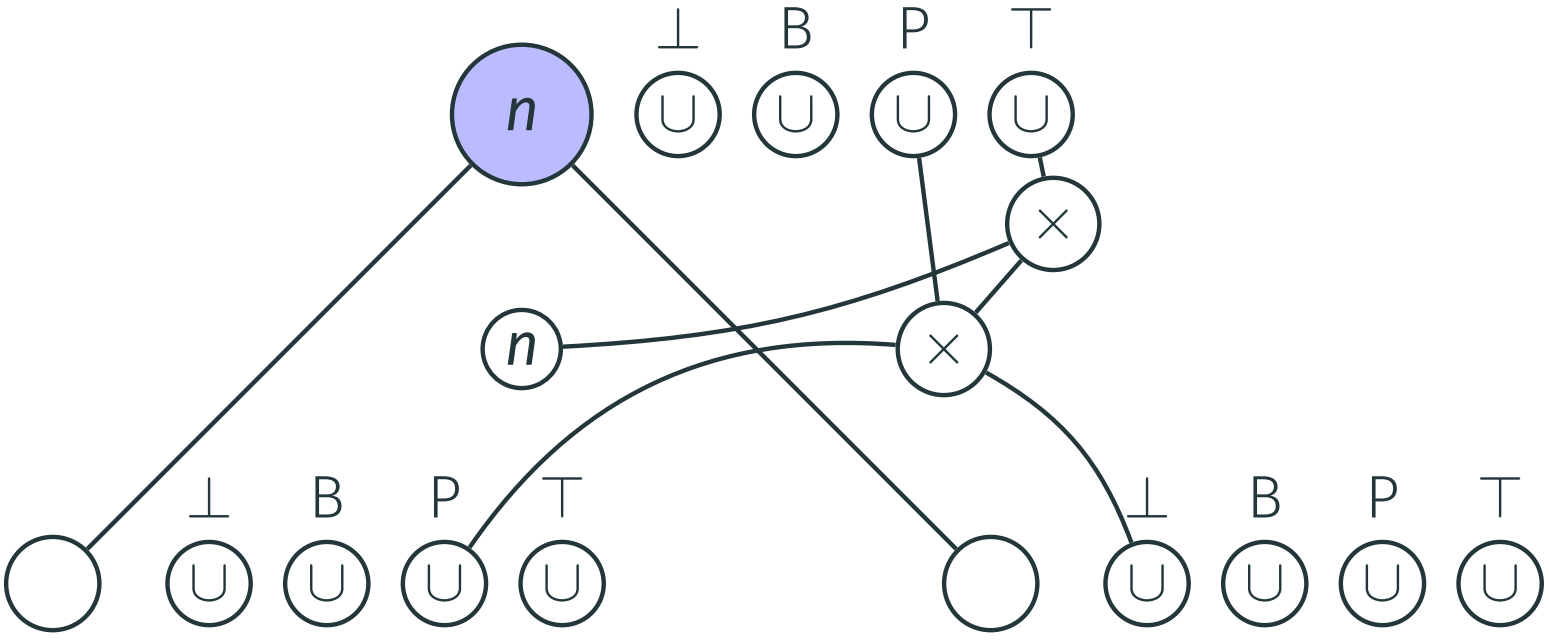


Building d-DNNF set circuits

Theorem

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

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



Meaning of the set circuit

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

Meaning of the set circuit

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

Example query:

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

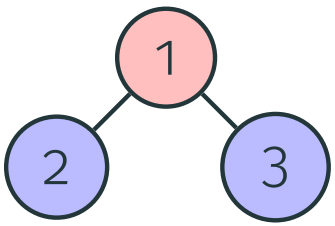
Meaning of the set circuit

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

Example query:

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

Data:



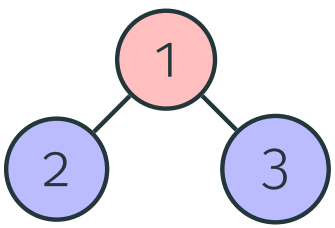
Meaning of the set circuit

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

Example query:

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

Data:



Results:

X_1	X_2
1	2
1	3

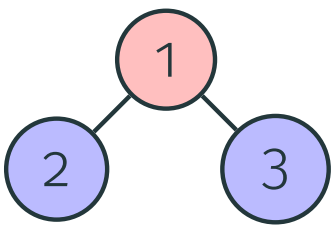
Meaning of the set circuit

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

Example query:

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

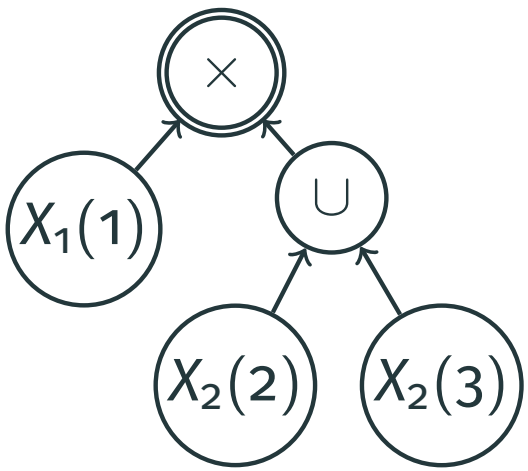
Data:



Results:

X_1	X_2
1	2
1	3

Set circuit:



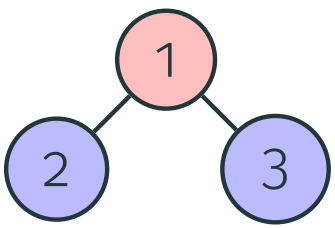
Meaning of the set circuit

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

Example query:

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

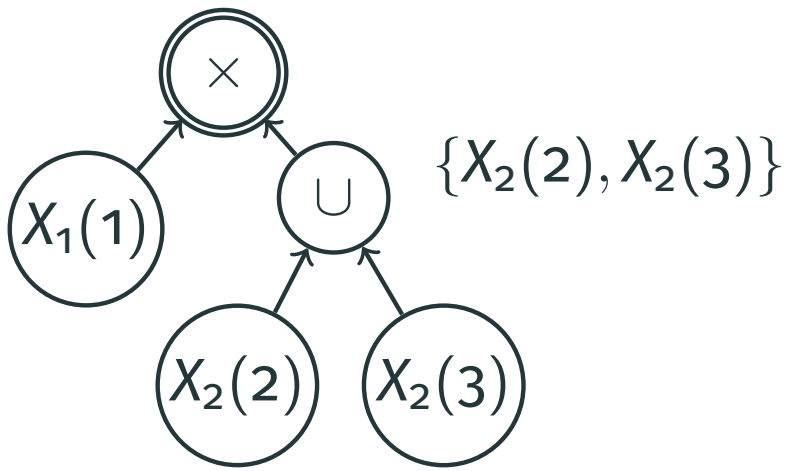
Data:



Results:

X_1	X_2
1	2
1	3

Set circuit:



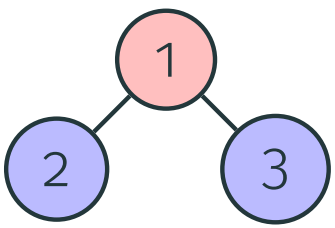
Meaning of the set circuit

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

Example query:

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

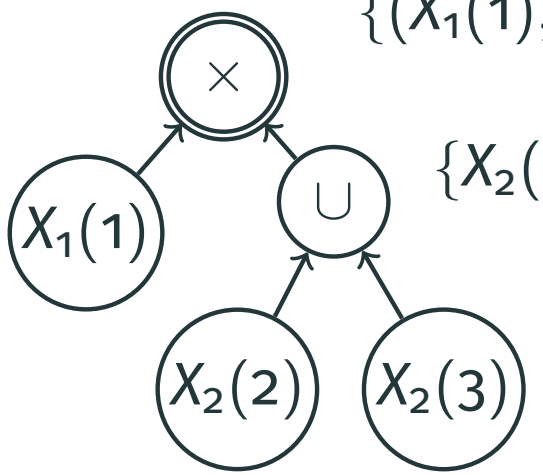
Data:



Results:

X_1	X_2
1	2
1	3

Set circuit:

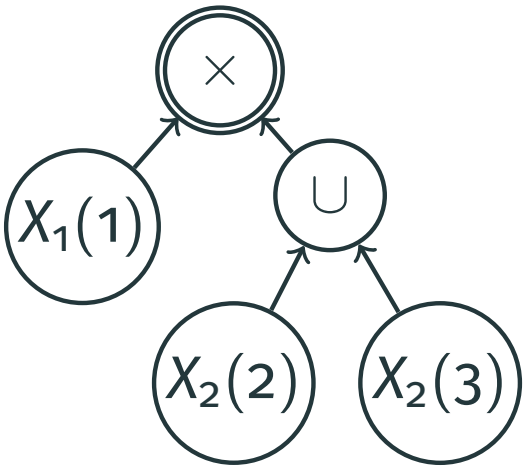


$\{(X_1(1), X_2(2)), (X_1(1), X_2(3))\}$

$\{X_2(2), X_2(3)\}$

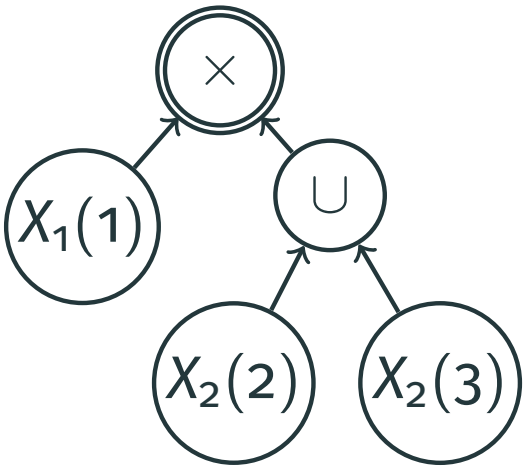
Enumeration for set circuits

The situation so far



- We started from our input tree T and query Q
- We have built a circuit C describing the results
- We know it is a d-DNNF
- We want to enumerate these results efficiently

The situation so far

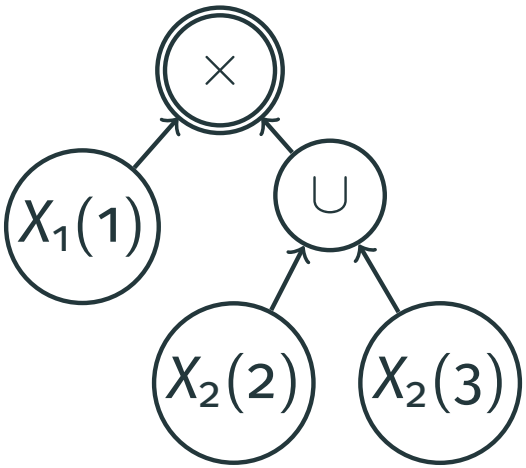


- We started from our input tree T and query Q
- We have built a **circuit** C describing the results
- We know it is a **d-DNNF**
- We want to **enumerate** these results efficiently

Theorem

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

The situation so far



- We started from our input tree T and query Q
- We have built a **circuit** C describing the results
- We know it is a **d-DNNF**
- We want to **enumerate** these results efficiently

Theorem

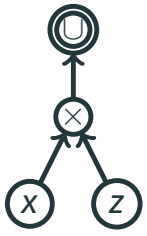
Given a d-DNNF set circuit C, we can enumerate its captured sets with preprocessing linear in $|C|$ and delay linear in each set

→ This is a **generic result** (does not talk about MSO or trees)

→ Any problems whose solutions can be coded as a d-DNNF can be efficiently enumerated via this method

Enumeration proof overview

Preprocessing phase:

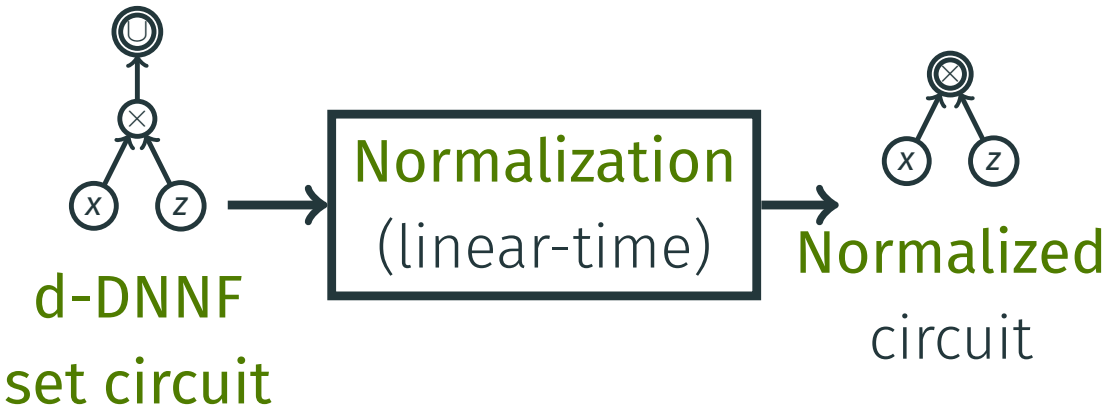


d-DNNF

set circuit

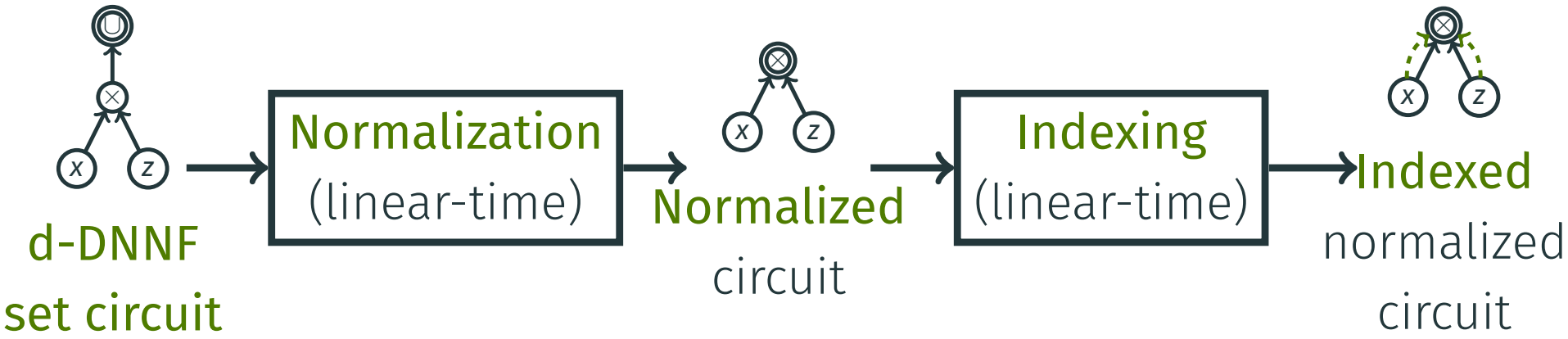
Enumeration proof overview

Preprocessing phase:



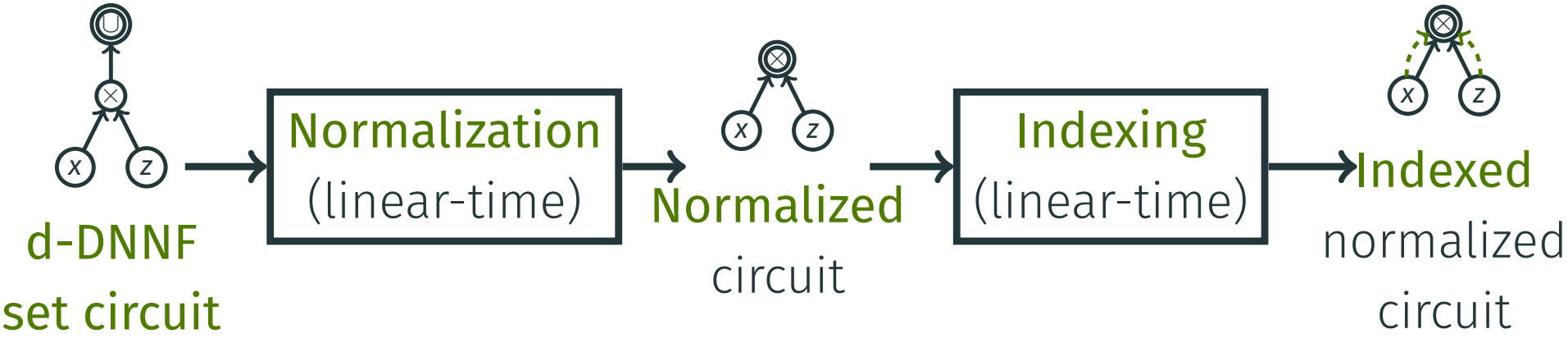
Enumeration proof overview

Preprocessing phase:

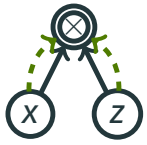


Enumeration proof overview

Preprocessing phase:



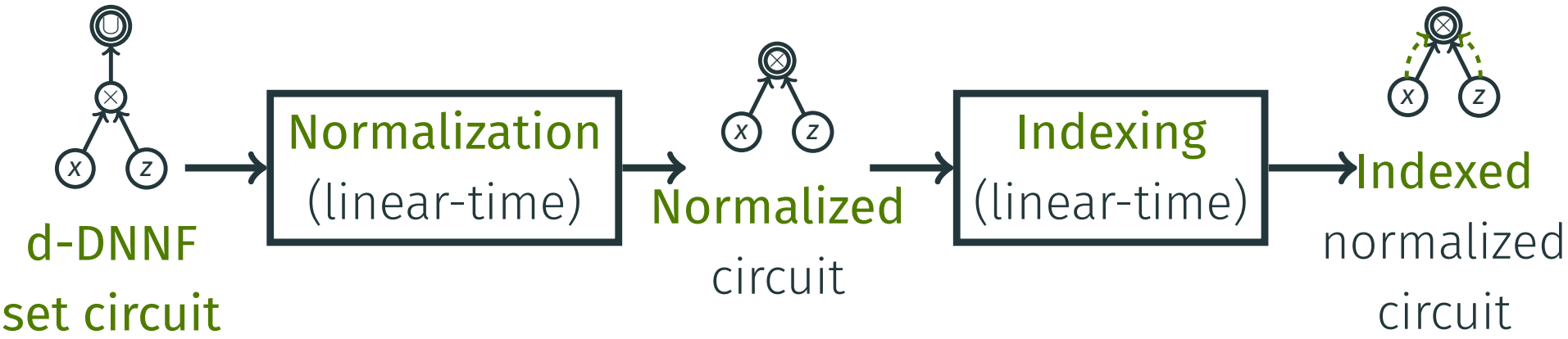
Enumeration phase:



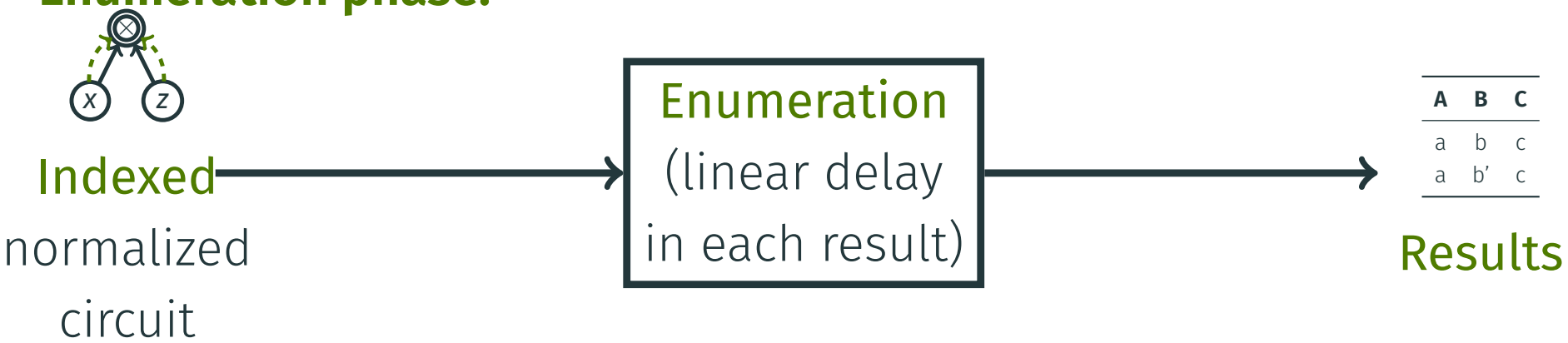
Indexed
normalized
circuit

Enumeration proof overview

Preprocessing phase:



Enumeration phase:



Enumerating captured sets of d-DNNF set circuits

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 $\{x, y\}$

Enumerating captured sets of d-DNNF set circuits

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 $\{x, y\}$

Base case: variable 

:

Enumerating captured sets of d-DNNF set circuits

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 $\{x, y\}$

Base case: variable 

: enumerate $\{x\}$ and stop

Enumerating captured sets of d-DNNF set circuits

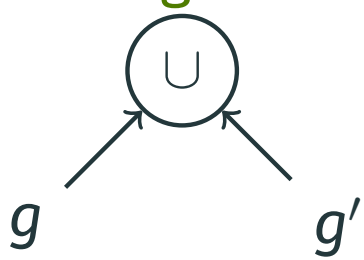
Task: Enumerate the elements of the set $S(g)$ captured by a gate g

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

Base case: variable 

: enumerate $\{x\}$ and stop

$\cup$-gate



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

Enumerating captured sets of d-DNNF set circuits

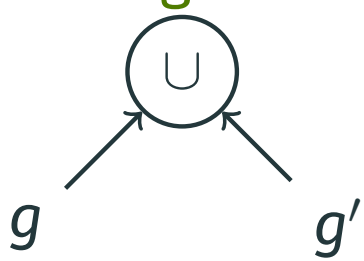
Task: Enumerate the elements of the set $S(g)$ captured by a gate g

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

Base case: variable 

: enumerate $\{x\}$ and stop

$\cup$-gate



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

Determinism: no duplicates

Enumerating captured sets of d-DNNF set circuits

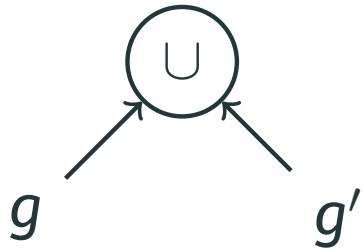
Task: Enumerate the elements of the set $S(g)$ captured by a gate g

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

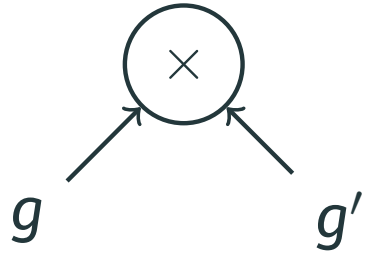
Base case: variable 

: enumerate $\{x\}$ and stop

$\cup$-gate



$\times$-gate



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

Determinism: no duplicates

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

Enumerating captured sets of d-DNNF set circuits

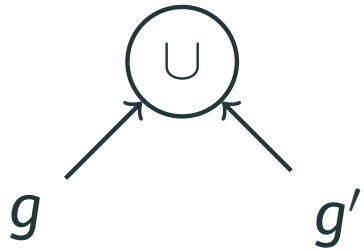
Task: Enumerate the elements of the set $S(g)$ captured by a gate g

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

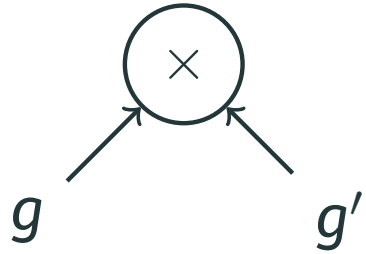
Base case: variable 

: enumerate $\{x\}$ and stop

$\cup$-gate



$\times$-gate



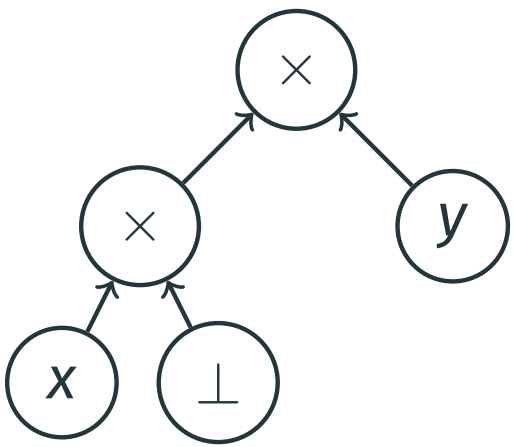
Concatenation: enumerate $S(g)$ and then enumerate $S(g')$

Determinism: no duplicates

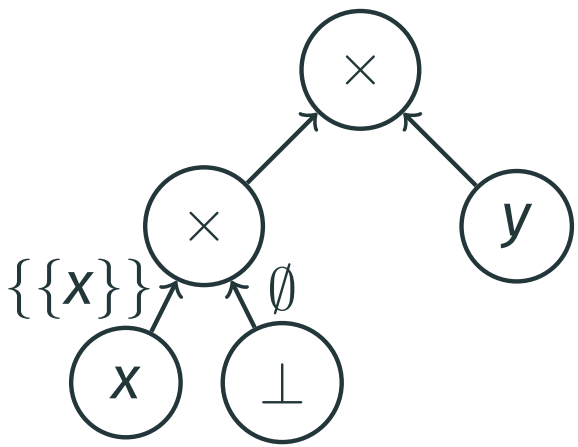
Lexicographic product: enumerate $S(g)$ and for each result t enumerate $S(g')$ and concatenate t with each result

Decomposability: no duplicates

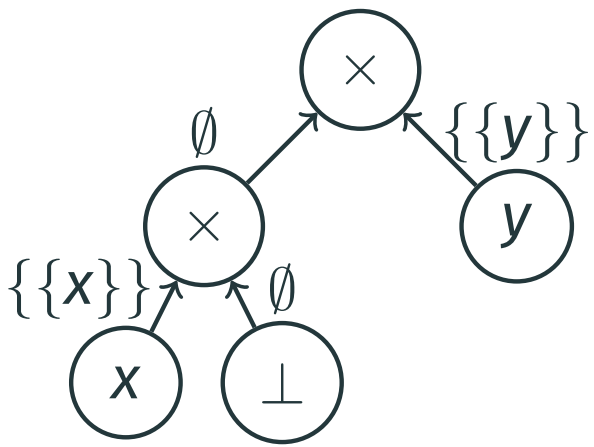
Normalization: handling $\emptyset$



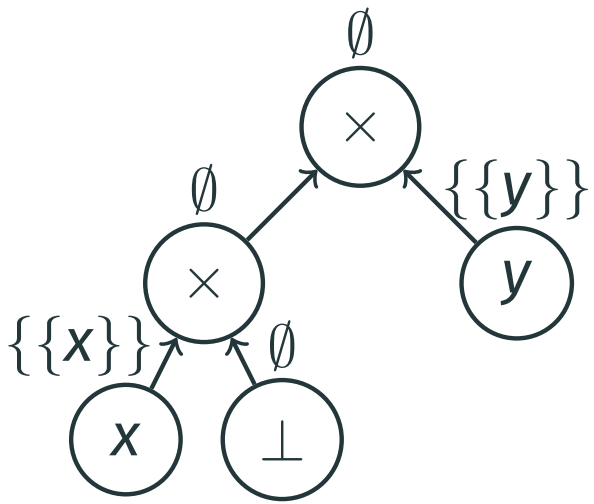
Normalization: handling $\emptyset$



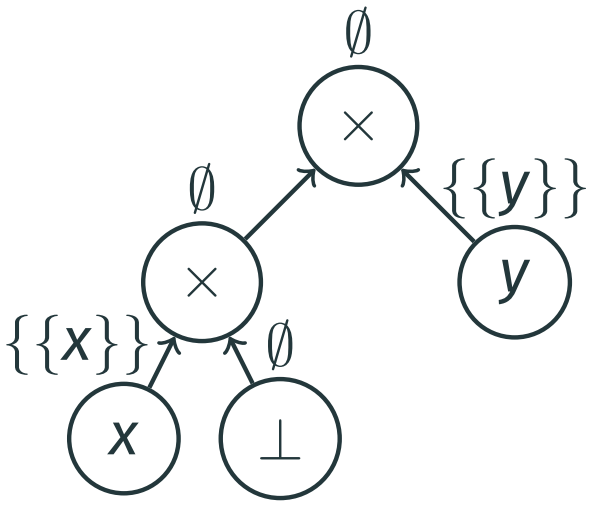
Normalization: handling $\emptyset$



Normalization: handling $\emptyset$

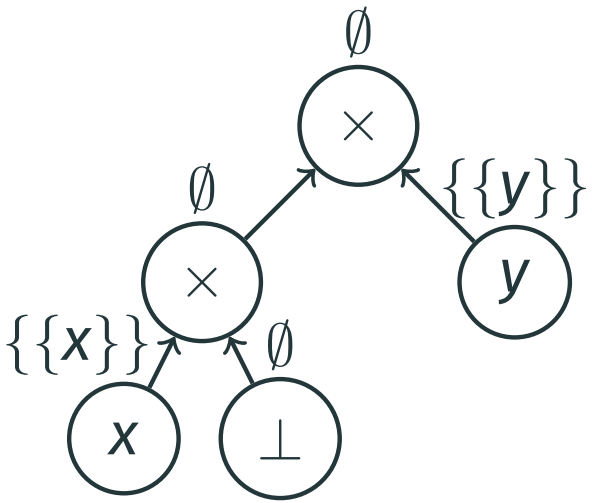


Normalization: handling $\emptyset$



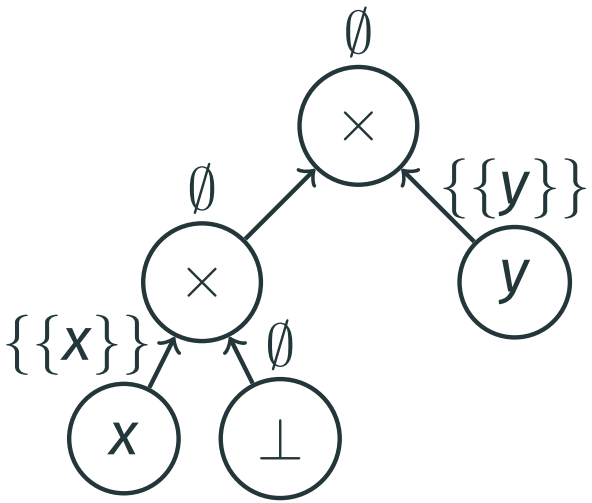
- Problem: if $S(g) = \emptyset$ we waste time

Normalization: handling $\emptyset$



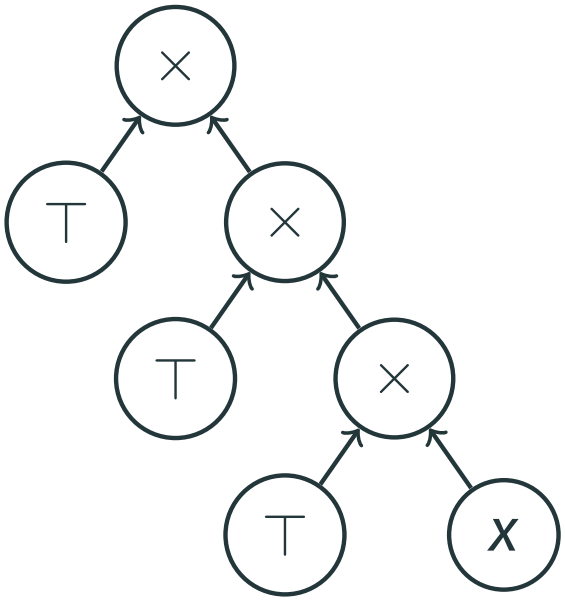
- Problem: if $S(g) = \emptyset$ we waste time
- Solution: in preprocessing
  - compute bottom-up if $S(g) = \emptyset$

Normalization: handling $\emptyset$

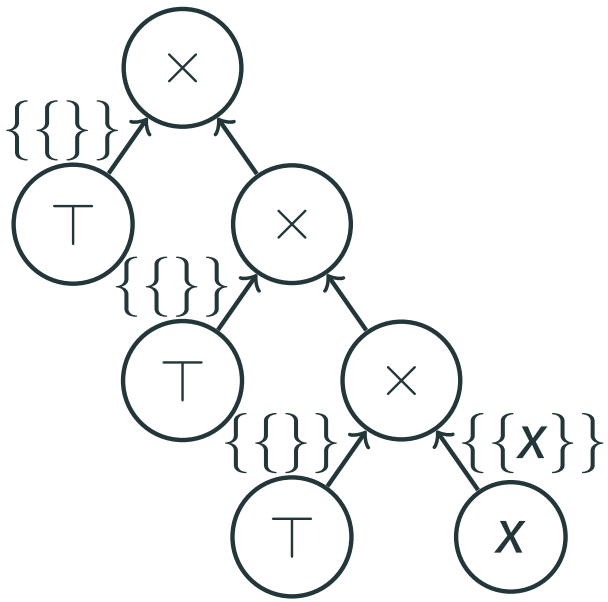


- Problem: if $S(g) = \emptyset$ we waste time
- Solution: in preprocessing
  - compute bottom-up if $S(g) = \emptyset$
  - then get rid of the gate

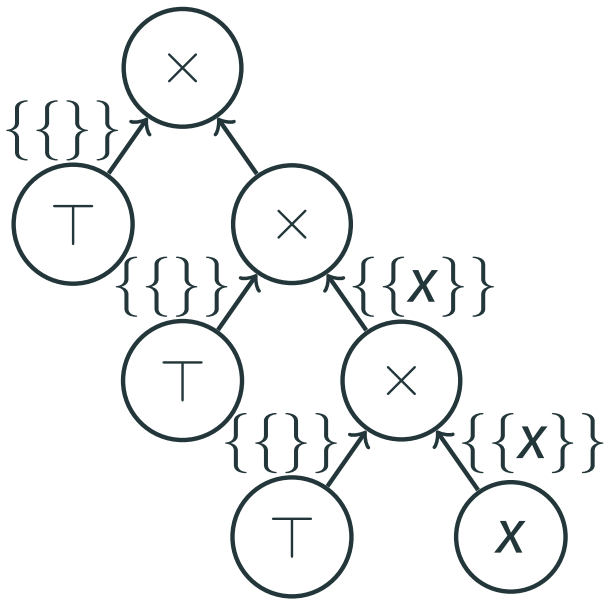
Normalization: handling empty sets



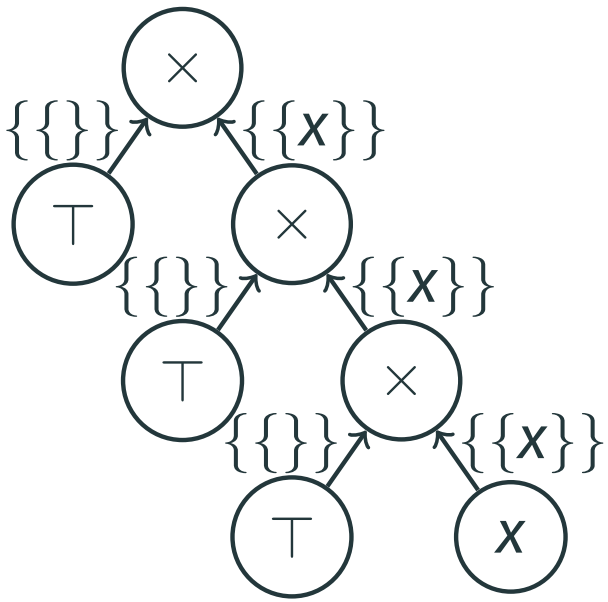
Normalization: handling empty sets



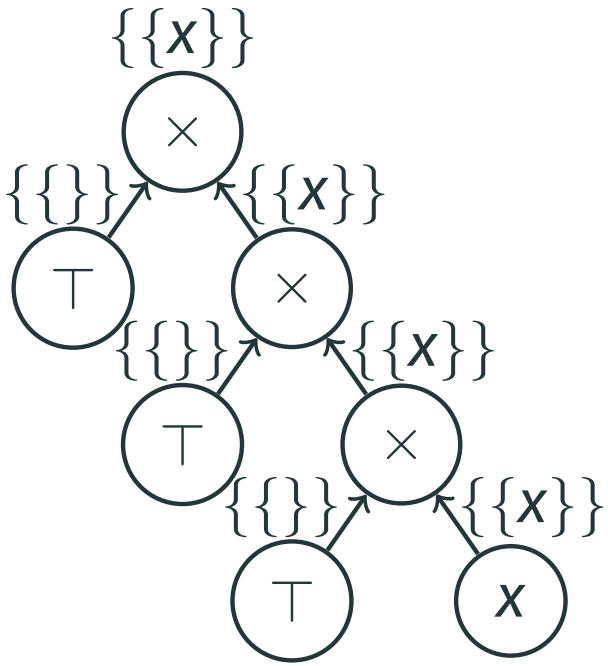
Normalization: handling empty sets



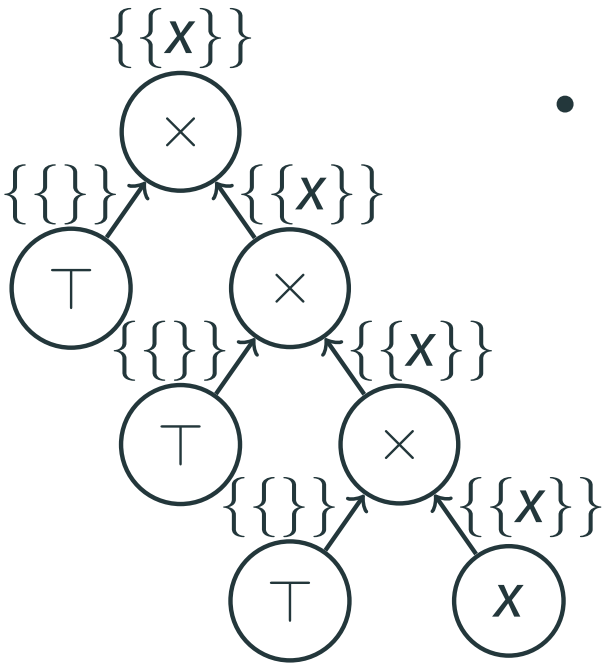
Normalization: handling empty sets



Normalization: handling empty sets

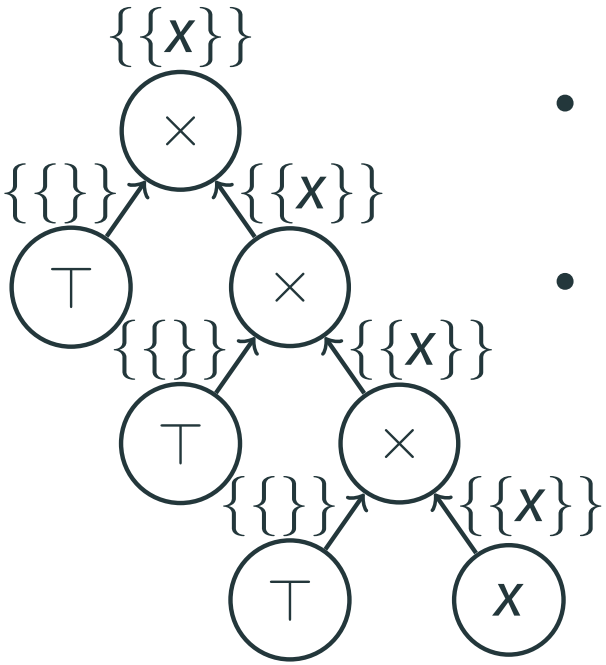


Normalization: handling empty sets



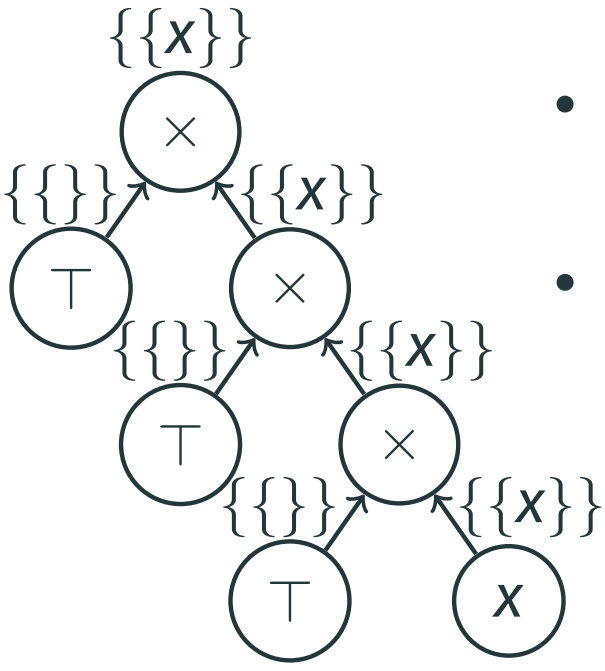
- Problem: if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates

Normalization: handling empty sets



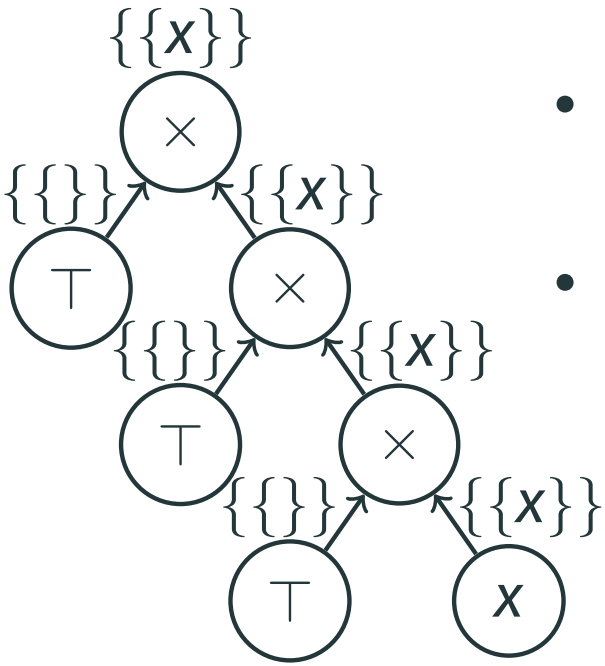
- Problem: if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- Solution:

Normalization: handling empty sets



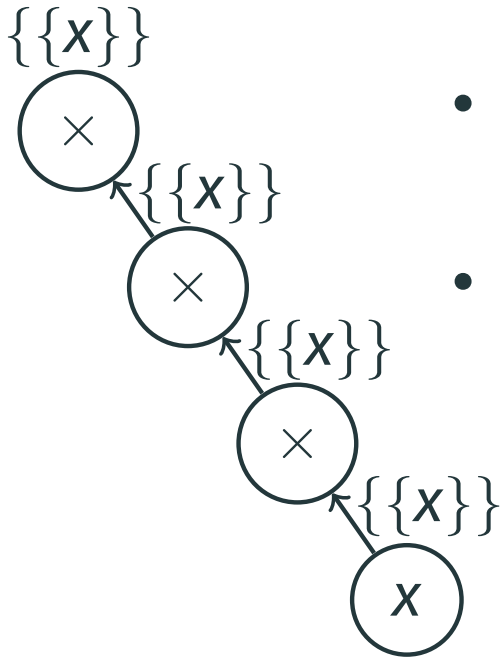
- Problem: if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- Solution:
  - split g between $S(g) \cap \{\{\}\}$ and $S(g) \setminus \{\{\}\}$ (homogenization)

Normalization: handling empty sets



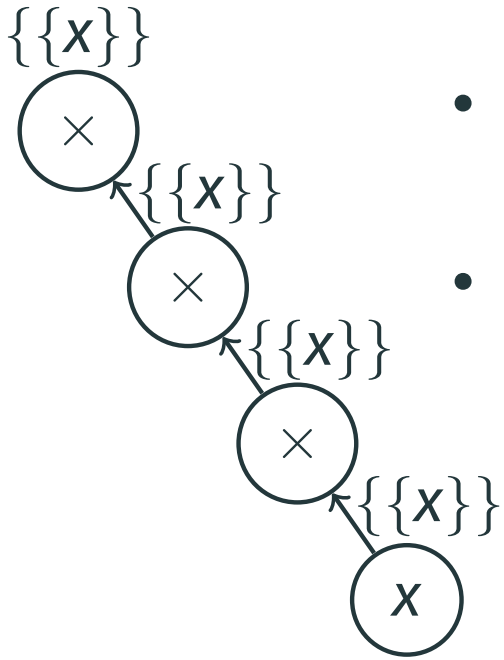
- Problem: if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- Solution:
  - split g between $S(g) \cap \{\}$ and $S(g) \setminus \{\}$ (homogenization)
  - remove inputs with $S(g) = \{\}$ for $\times$-gates

Normalization: handling empty sets



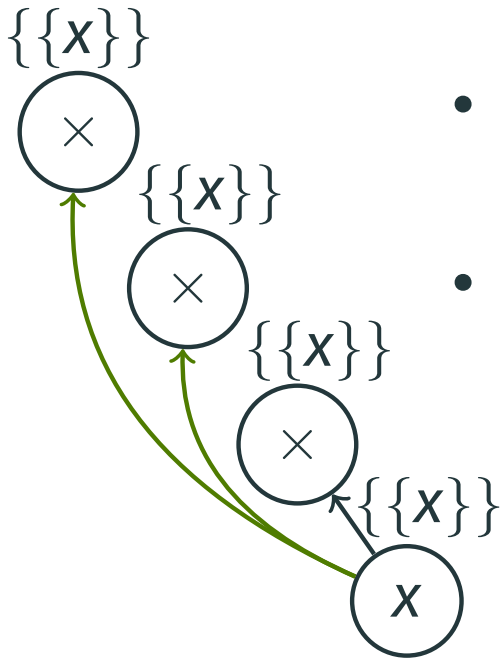
- Problem: if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- Solution:
  - split g between $S(g) \cap \{\{\}\}$ and $S(g) \setminus \{\{\}\}$ (homogenization)
  - remove inputs with $S(g) = \{\{\}\}$ for $\times$-gates

Normalization: handling empty sets



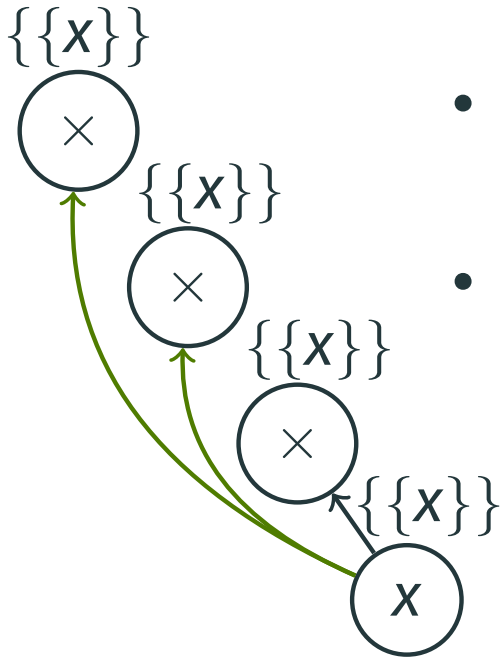
- Problem: if $S(g)$ contains $\{\}$ we waste time in chains of $\times$-gates
- Solution:
  - split g between $S(g) \cap \{\{\}\}$ and $S(g) \setminus \{\{\}\}$ (homogenization)
  - remove inputs with $S(g) = \{\{\}\}$ for $\times$-gates
  - collapse $\times$-chains with fan-in 1

Normalization: handling empty sets



- 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

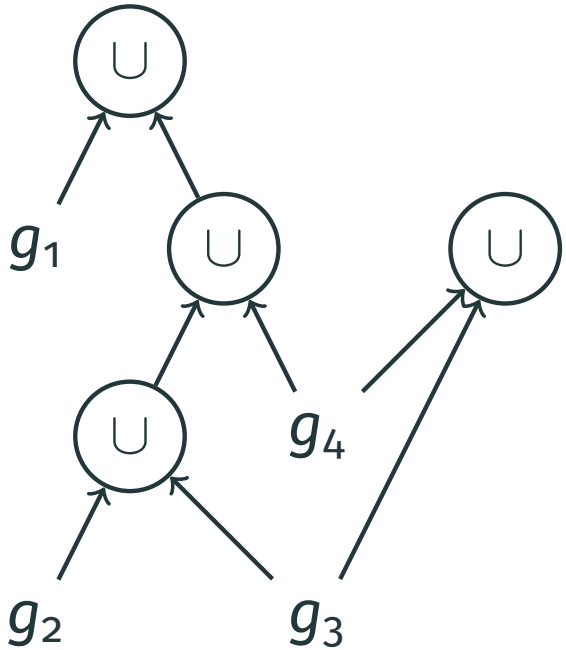
Normalization: handling empty sets



- **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

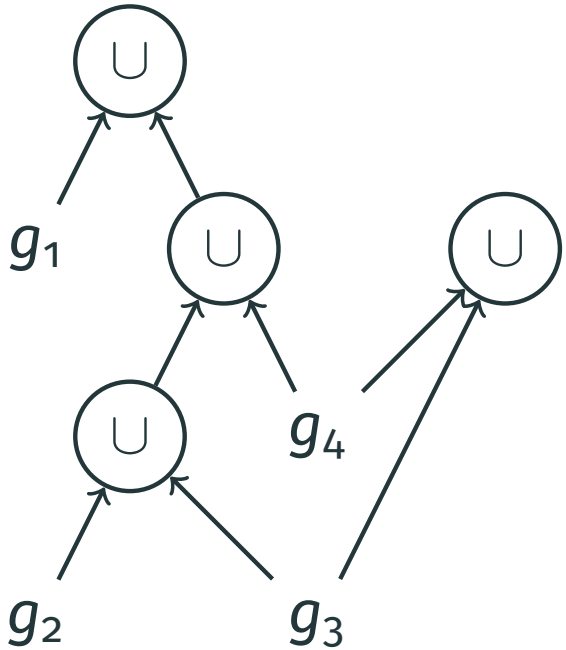
$\rightarrow$ Now, traversing a $\times$**-gate** ensures that we make progress: it **splits** the sets non-trivially

Indexing: handling ∪-hierarchies



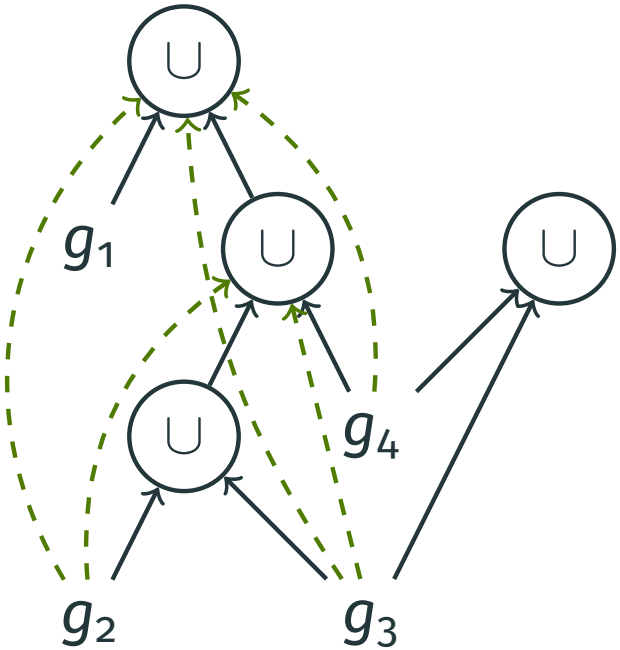
- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)

Indexing: handling ∪-hierarchies



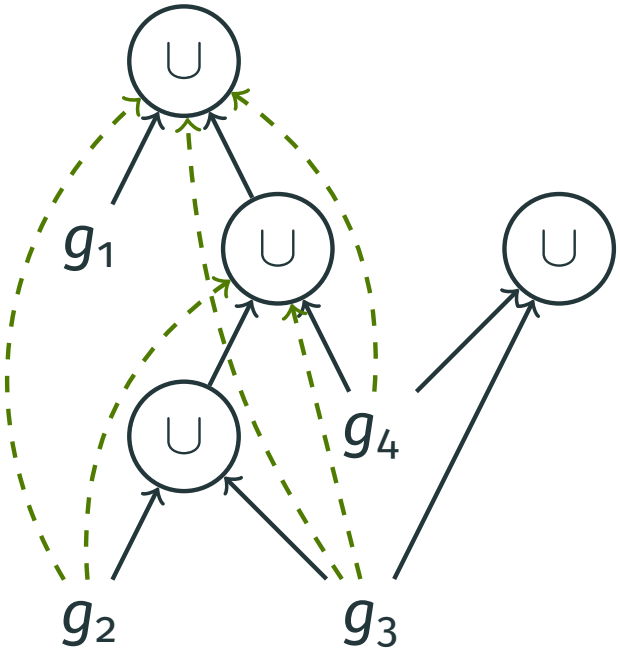
- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)
- **Solution:** compute **reachability index**

Indexing: handling ∪-hierarchies



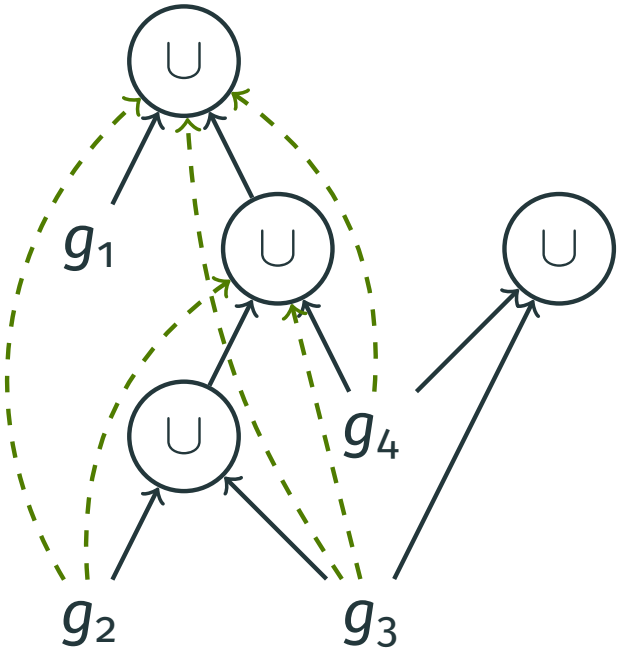
- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)
- **Solution:** compute **reachability index**

Indexing: handling ∪-hierarchies



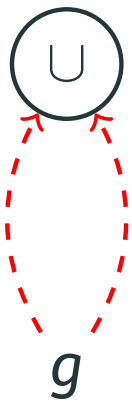
- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)
- **Solution:** compute **reachability index**
- **Problem:** must be done in **linear time**

Indexing: handling ∪-hierarchies

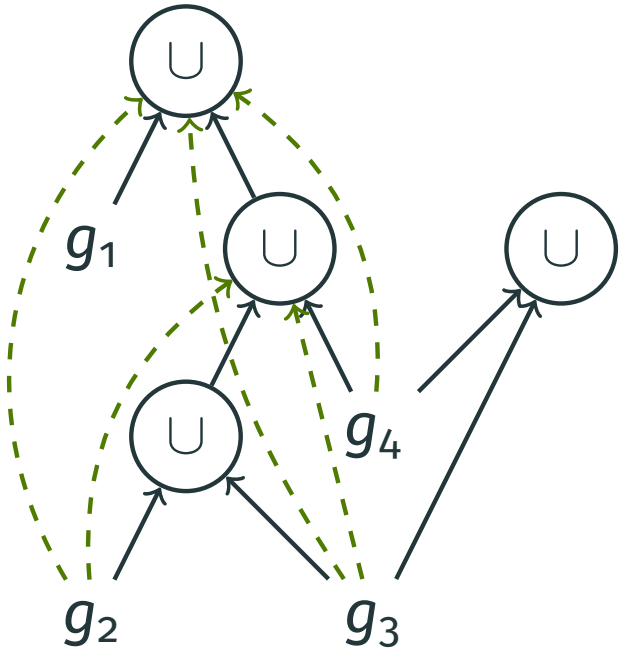


- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)
- **Solution:** compute **reachability index**
- **Problem:** must be done in **linear time**

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

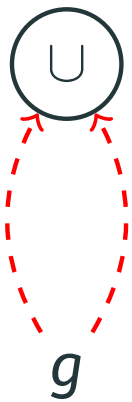


Indexing: handling ∪-hierarchies

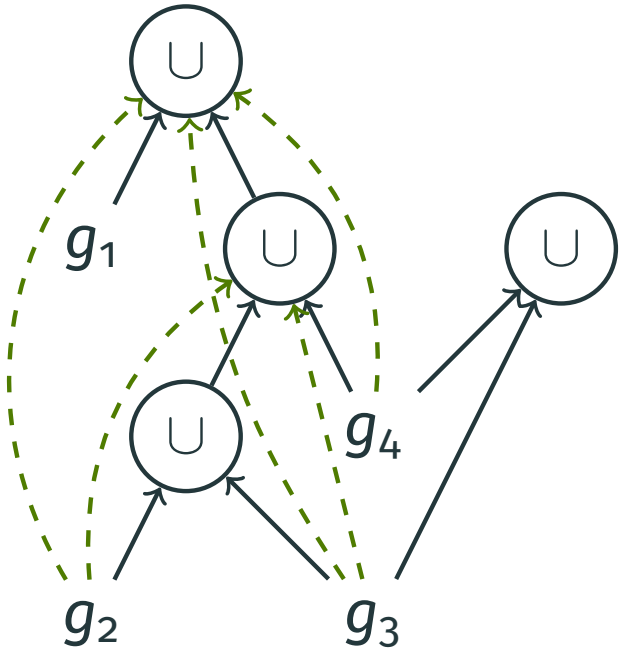


- **Problem:** we waste time in ∪-hierarchies to find a **reachable exit** (non-∪ gate)
- **Solution:** compute **reachability index**
- **Problem:** must be done in **linear time**

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

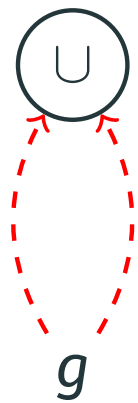


Indexing: handling ∪-hierarchies



- **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
- For **MSO query evaluation**: **upwards-deterministic** circuit so we have a **tree**: simpler constant-memory index



Summary of results

We have shown:

Theorem

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

Summary of results

We have shown:

Theorem

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

And for any **fixed** MSO query Q, given a tree T, we can...

- Construct a d-DNNF C representing the results in $O(T)$
- Apply to C the scheme above

Summary of results

We have shown:

Theorem

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

And for any fixed MSO query Q, given a tree T, we can...

- Construct a d-DNNF C representing the results in $O(T)$
- Apply to C the scheme above

So we have re-proved:

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

For any fixed MSO query Q, given a tree T, we can preprocess T in linear time in T and then enumerate each result in linear time in the result

Application to text and combined complexity

Problem statement: Pattern matching in texts

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 ...
```

Problem statement: Pattern matching in texts

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

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

Problem statement: Pattern matching in texts

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

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

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

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

Problem statement: Pattern matching in texts

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:

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

Goal:

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

Reducing to MSO

- A text is just a tree with a simpler shape

Reducing to MSO

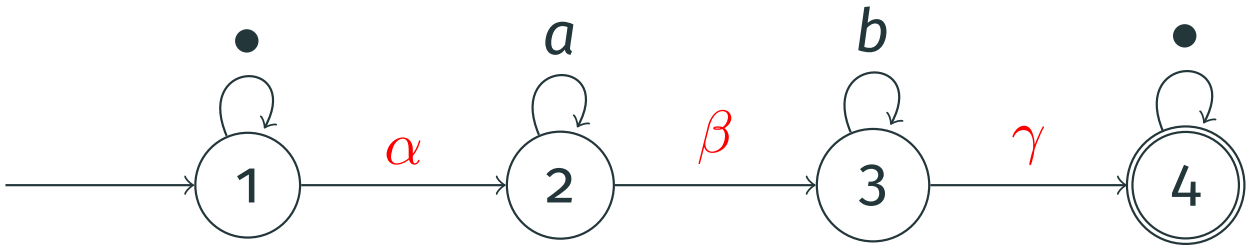
- A text is just a tree with a simpler shape
- A regular expression pattern can be expressed in MSO

Reducing to MSO

- A **text** is just a **tree** with a simpler shape
- A **regular expression pattern** can be expressed in MSO
  - $\rightarrow$ More generally: regular expressions with **variables**
  - $\rightarrow$ **Example:** $P := \bullet^* \ \alpha \ a^* \ \beta \ b^* \ \gamma \ \bullet^*$
- Translate to a **word automaton** (with **capture variables**)

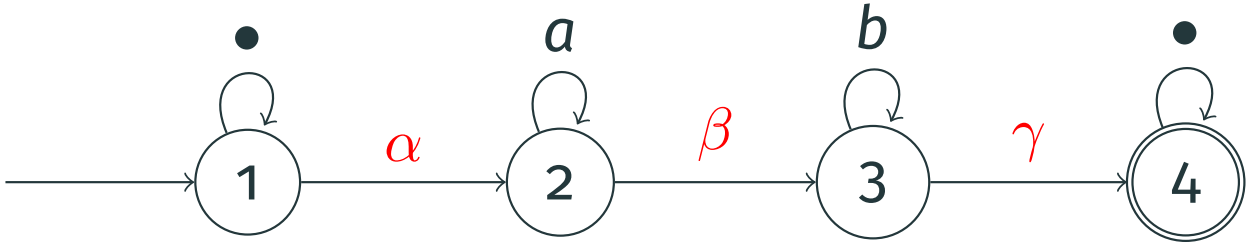
Reducing to MSO

- A text is just a tree with a simpler shape
- A regular expression pattern can be expressed in MSO
  - → More generally: regular expressions with variables
  - → Example: $P := \bullet^* \ \alpha \ a^* \ \beta \ b^* \ \gamma \ \bullet^*$
- Translate to a word automaton (with capture variables)



Reducing to MSO

- A **text** is just a **tree** with a simpler shape
- A **regular expression pattern** can be expressed in MSO
  - → More generally: regular expressions with **variables**
  - → **Example:** $P := \bullet^* \ \alpha \ a^* \ \beta \ b^* \ \gamma \ \bullet^*$
- Translate to a **word automaton** (with **capture variables**)



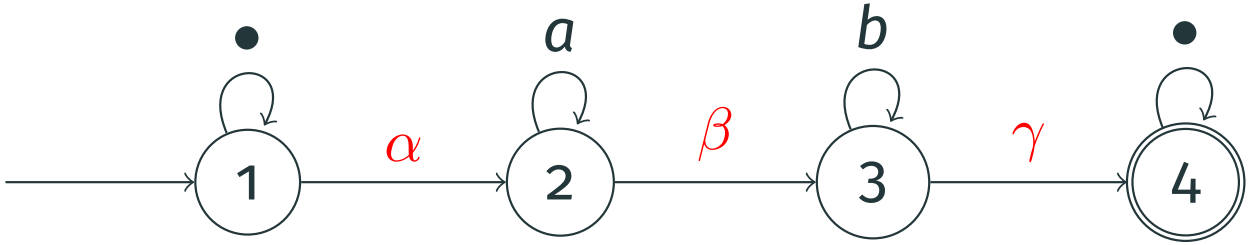
→ The MSO result implies:

Theorem [Florenzano et al., 2018]

We can enumerate all matches of a regular expression pattern on a tree with linear preprocessing and constant delay

Reducing to MSO

- A **text** is just a **tree** with a simpler shape
- A **regular expression pattern** can be expressed in MSO
  - → More generally: regular expressions with **variables**
  - → **Example:** $P := \bullet^* \ \alpha \ a^* \ \beta \ b^* \ \gamma \ \bullet^*$
- Translate to a **word automaton** (with **capture variables**)



→ The MSO result implies:

Theorem [Florenzano et al., 2018]

We can enumerate all matches of a regular expression pattern on a tree with linear preprocessing and constant delay

→ The resulting set circuit is a **binary decision diagram**, i.e., each $\times$-gate has only one input which is not a variable

Efficiency in the query

We have shown linear preprocessing and constant delay in the data; but what about the query?

- For general MSO queries: nonelementary complexity

Efficiency in the query

We have shown linear preprocessing and constant delay in the data; but what about the query?

- For general MSO queries: nonelementary complexity
- For regular expressions: exponential (determinization)

Efficiency in the query

We have shown linear preprocessing and constant delay in the data; but what about the query?

- For general MSO queries: nonelementary complexity
- For regular expressions: exponential (determinization)
- In fact: our methods adapt to nondeterministic automata

Efficiency in the query

We have shown linear preprocessing and constant delay in the data; but what about the query?

- For general MSO queries: nonelementary complexity
- For regular expressions: exponential (determinization)
- In fact: our methods adapt to nondeterministic automata

Theorem [Amarilli et al., 2019a, Amarilli et al., 2019b]

We can enumerate all matches of a nondeterministic tree automaton on a tree with

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

Efficiency in the query

We have shown linear preprocessing and constant delay in the data; but what about the query?

- For general MSO queries: nonelementary complexity
- For regular expressions: exponential (determinization)
- In fact: our methods adapt to nondeterministic automata

Theorem [Amarilli et al., 2019a, Amarilli et al., 2019b]

We can enumerate all matches of a nondeterministic tree automaton on a tree with

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

Corollary

Given a regular expression pattern P and text T, we can enumerate all matches of P on T with the complexity above

Implementation (ongoing internship by Rémi Dupré)

- Prototype to find matches of a regular expression in a text
- `https://github.com/remi-dupre/enum-spanner-rs`
- Work-in-progress

Implementation (ongoing internship by Rémi Dupré)

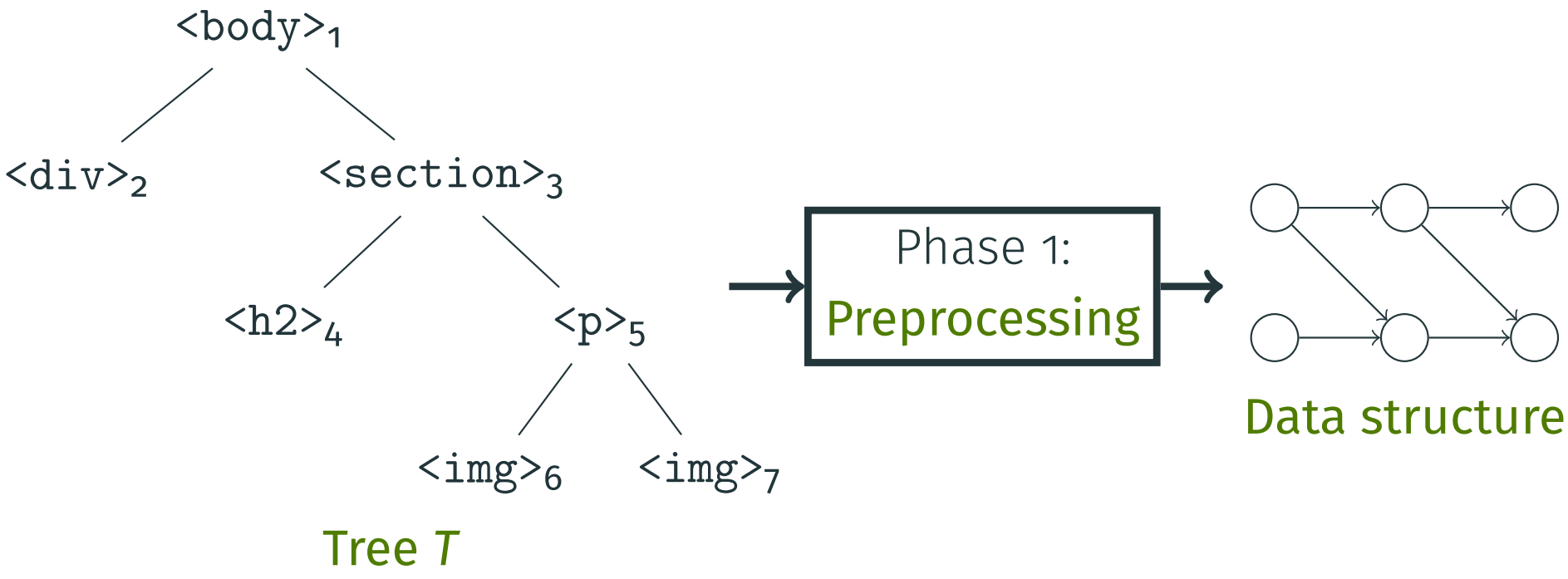
- Prototype to find matches of a regular expression in a text
- `https://github.com/remi-dupre/enum-spanner-rs`
- Work-in-progress
- Open questions / projects:
  - What about memory usage? (we cannot keep the whole index)
  - Output matches in streaming? (problem: duplicates)
  - Can we enumerate other notions of matches?
    - → factors of maximal/minimal size
    - → distinct matching strings
    - → etc.

Implementation (ongoing internship by Rémi Dupré)

- Prototype to find matches of a regular expression in a text
- `https://github.com/remi-dupre/enum-spanner-rs`
- Work-in-progress
- Open questions / projects:
  - What about memory usage? (we cannot keep the whole index)
  - Output matches in streaming? (problem: duplicates)
  - Can we enumerate other notions of matches?
    - → factors of maximal/minimal size
    - → distinct matching strings
    - → etc.
  - Which application domains need this?
  - Are there good benchmarks?

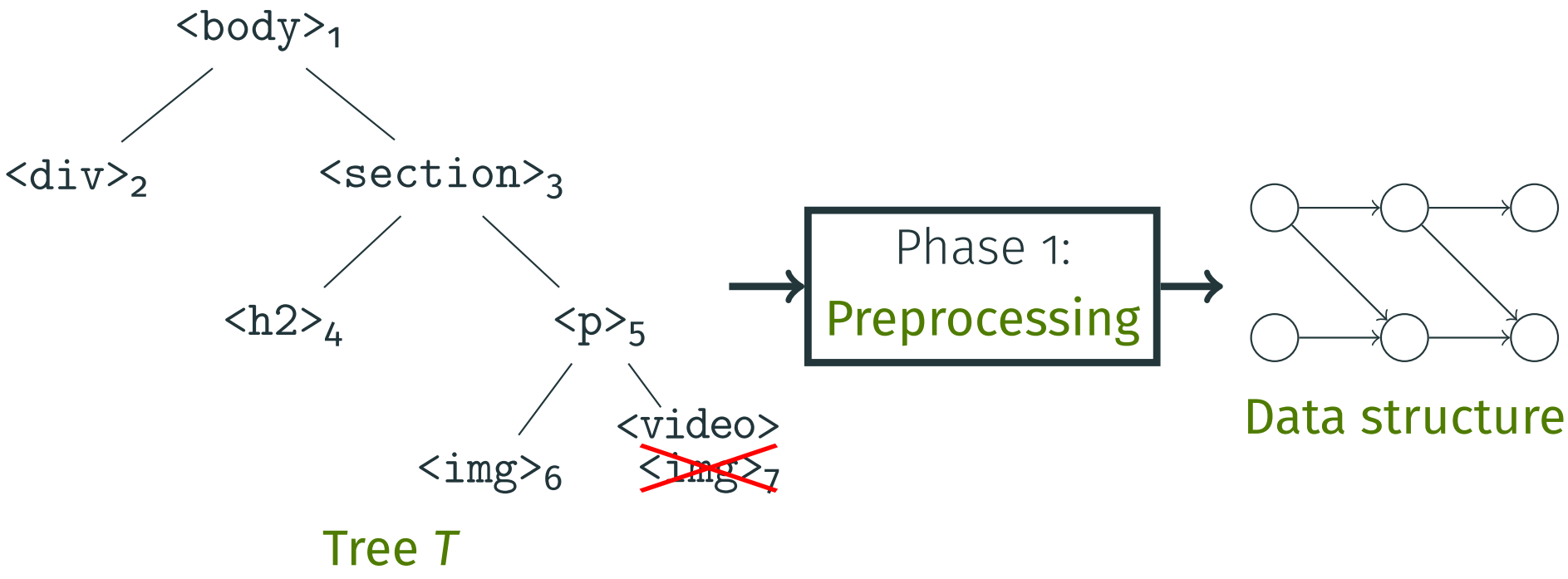
Handling updates

Updates



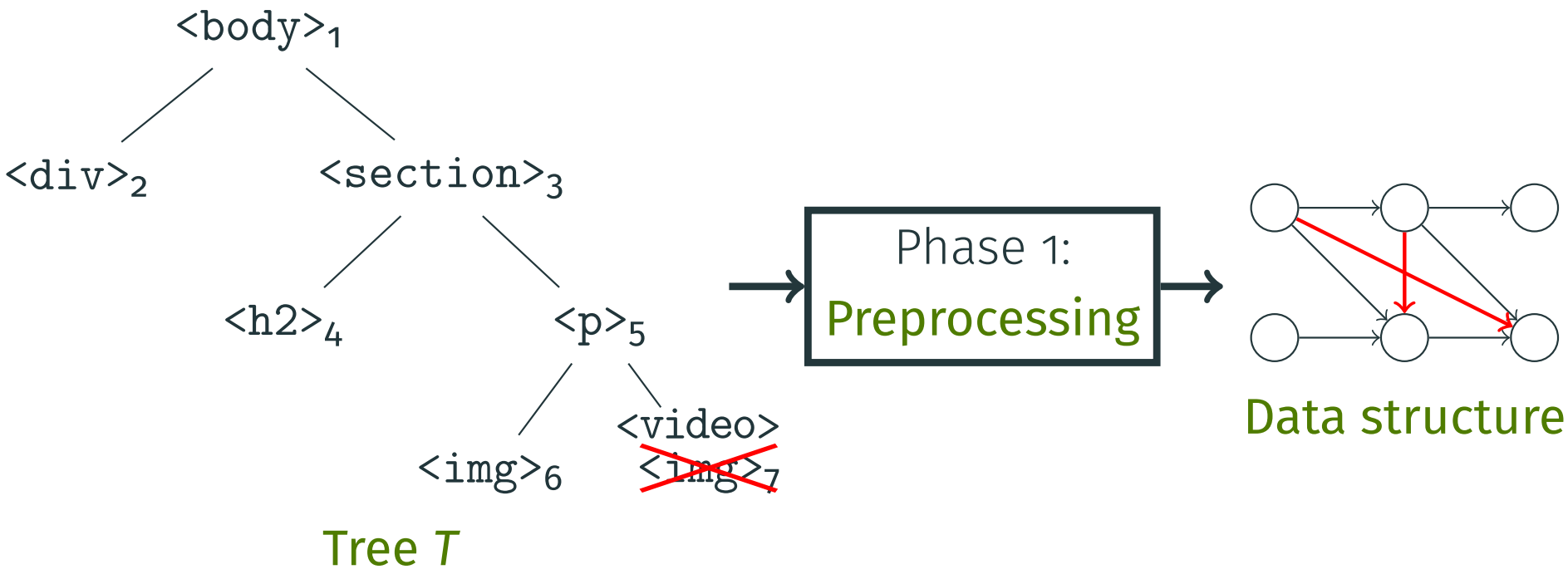
- The input data can be **modified** after the preprocessing

Updates



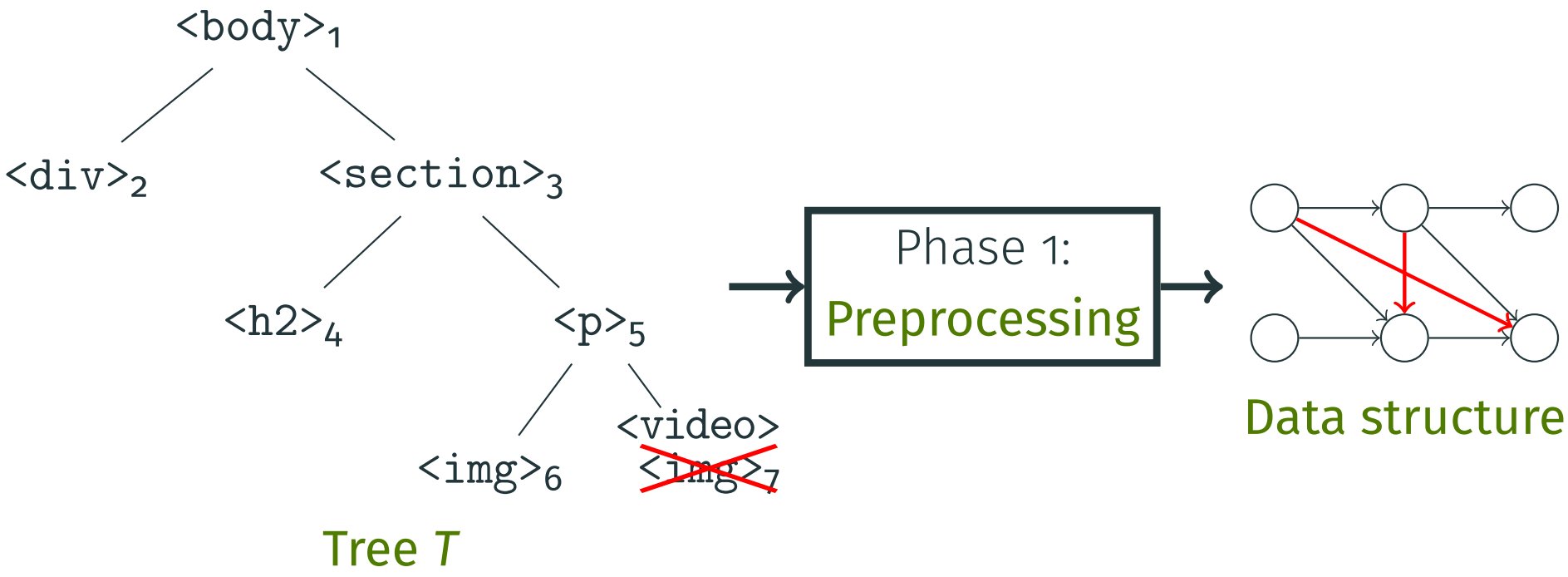
- The input data can be **modified** after the preprocessing

Updates



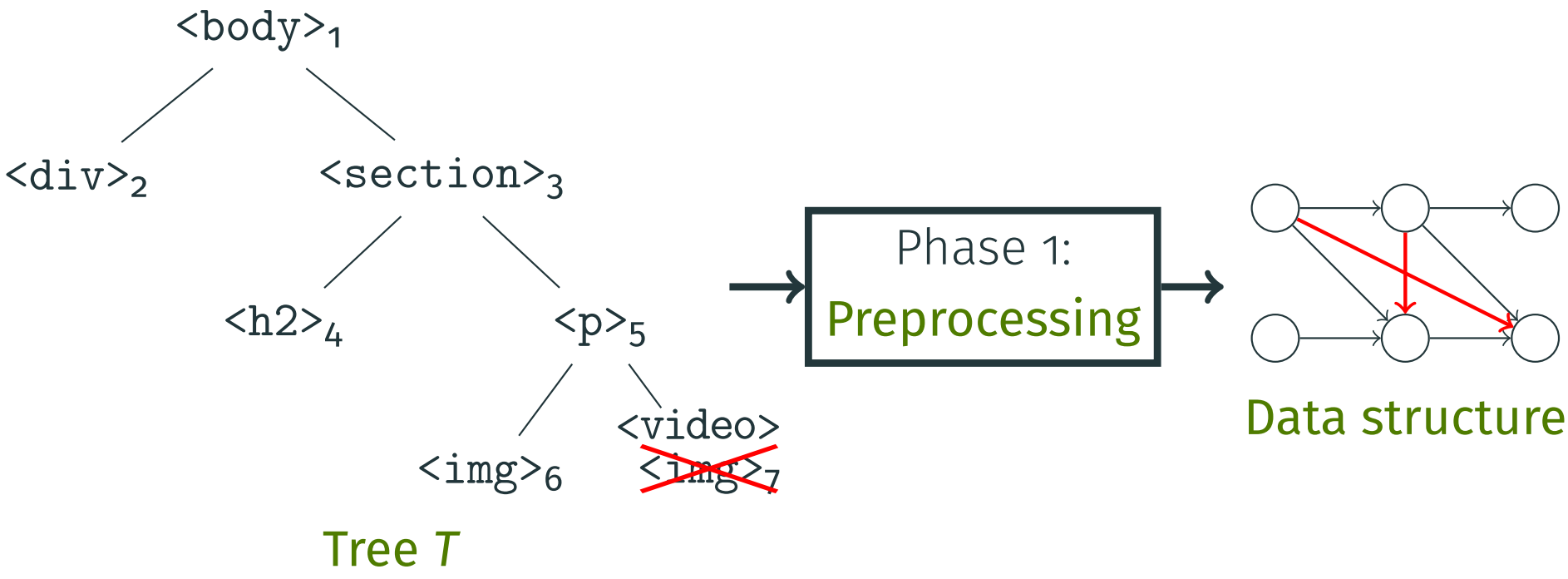
- The input data can be **modified** after the preprocessing

Updates



- The input data can be **modified** after the preprocessing
- If this happen, we must rerun the **preprocessing** from scratch

Updates



- The input data can be **modified** after the preprocessing
- If this happen, we must rerun the **preprocessing** from scratch

→ Can we **do better**?

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	$O(T)$	$O(1)$	$O(T)$

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	$O(T)$	$O(\log^2 T)$	$O(\log^2 T)$

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	$O(T)$	$O(\log^2 T)$	$O(\log^2 T)$
[Losemann and Martens, 2014]	text	$O(T)$	$O(\log T)$	$O(\log T)$

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	$O(T)$	$O(\log^2 T)$	$O(\log^2 T)$
[Losemann and Martens, 2014]	text	$O(T)$	$O(\log T)$	$O(\log T)$
[Niewerth and Segoufin, 2018]	text	$O(T)$	$O(1)$	$O(\log T)$

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	$O(T)$	$O(\log^2 T)$	$O(\log^2 T)$
[Losemann and Martens, 2014]	text	$O(T)$	$O(\log T)$	$O(\log T)$
[Niewerth and Segoufin, 2018]	text	$O(T)$	$O(1)$	$O(\log T)$
[Amarilli et al., 2019b]	trees	$O(T)$	$O(1)$	$O(\log T)$

Summary and open problems

Summary and conclusion

Theorem

Given a deterministic tree automaton A and a tree T, we can build in $O(|A| \times |T|)$ a d-DNNF set circuit capturing the results of A on T.

Theorem

Given a d-DNNF set circuit C, we can enumerate its results with linear preprocessing and delay linear in each result

Summary and conclusion

Theorem

Given a deterministic tree automaton A and a tree T, we can build in $O(|A| \times |T|)$ a d-DNNF set circuit capturing the results of A on T.

Theorem

Given a d-DNNF set circuit C, we can enumerate its results with linear preprocessing and delay linear in each result

- If A is nondeterministic, this still works with $O(\text{Poly}(A))$
- If T is updated, we can handle the change in $O(\log |T|)$

Summary and conclusion

Theorem

Given a deterministic tree automaton A and a tree T, we can build in $O(|A| \times |T|)$ a d-DNNF set circuit capturing the results of A on T.

Theorem

Given a d-DNNF set circuit C, we can enumerate its results with linear preprocessing and delay linear in each result

- If A is nondeterministic, this still works with $O(\text{Poly}(A))$
- If T is updated, we can handle the change in $O(\log |T|)$

Open problems:

- Implementation use cases?
- Lower bounds?
- Enumeration with order?
- Memory usage?
- Connection to tuple testing?
- Generic indexes?

References i


Amarilli, A., Bourhis, P., Mengel, S., and Niewerth, M. (2019a).
Constant-Delay Enumeration for Nondeterministic Document Spanners.
In *ICDT*.


Amarilli, A., Bourhis, P., Mengel, S., and Niewerth, M. (2019b).
Enumeration on Trees with Tractable Combined Complexity and Efficient Updates.
In *PODS*.


Bagan, G. (2006).
MSO queries on Tree Decomposable Structures Are Computable with Linear Delay.
In *CSL*.

References ii


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).


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

References iii

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

Thatcher, J. W. and Wright, J. B. (1968).
Generalized Finite Automata Theory with an Application to a Decision Problem of Second-Order Logic.
Mathematical systems theory, 2(1):57–81.

Hack: adding tree nodes to express the variable assignments

- Query: $Q(x_1, \ldots, x_n)$ with free variables $x_1, \ldots, x_n$

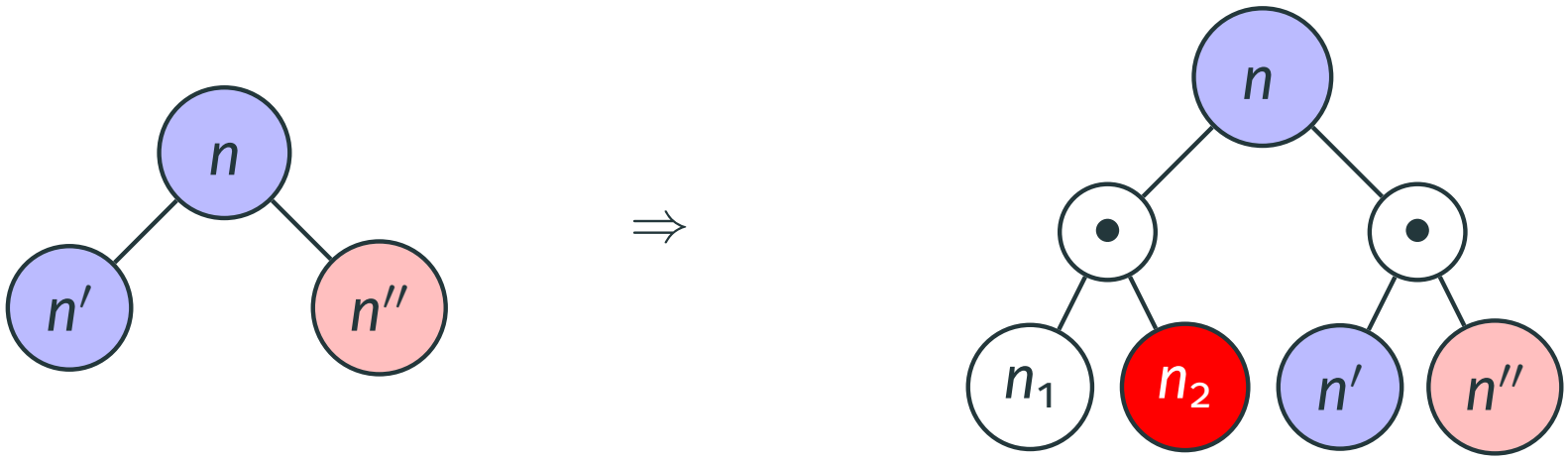
Hack: adding tree nodes to express the variable assignments

- Query: $Q(x_1, \ldots, x_n)$ with free variables $x_1, \ldots, x_n$
- Goal: find all tuples $a_1, \ldots, a_n$ such that $Q(a_1, \ldots, a_n)$ holds

Hack: adding tree nodes to express the variable assignments

- Query: $Q(x_1, \ldots, x_n)$ with free variables $x_1, \ldots, x_n$
- Goal: find all tuples $a_1, \ldots, a_n$ such that $Q(a_1, \ldots, a_n)$ holds

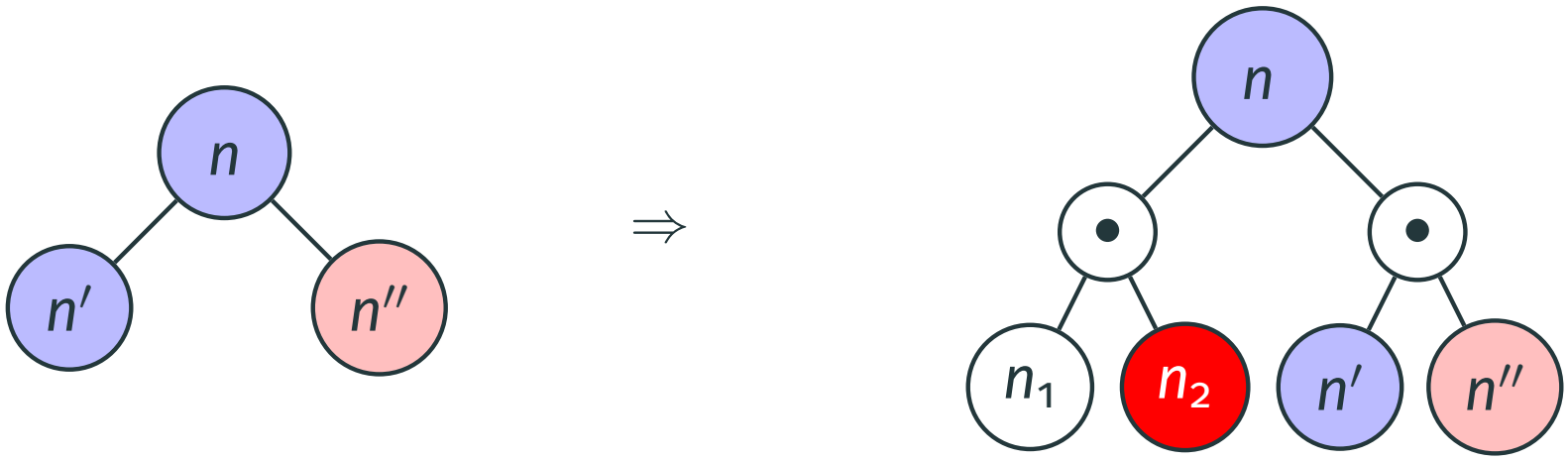
$\rightarrow$ Add special nodes: for each node n and variable x_i, add a node n_i which is colored red iff x_i is the node n



Hack: adding tree nodes to express the variable assignments

- Query: $Q(x_1, \ldots, x_n)$ with free variables $x_1, \ldots, x_n$
- Goal: find all tuples $a_1, \ldots, a_n$ such that $Q(a_1, \ldots, a_n)$ holds

→ Add special nodes: for each node n and variable x_i, add a node n_i which is colored red iff x_i is the node n

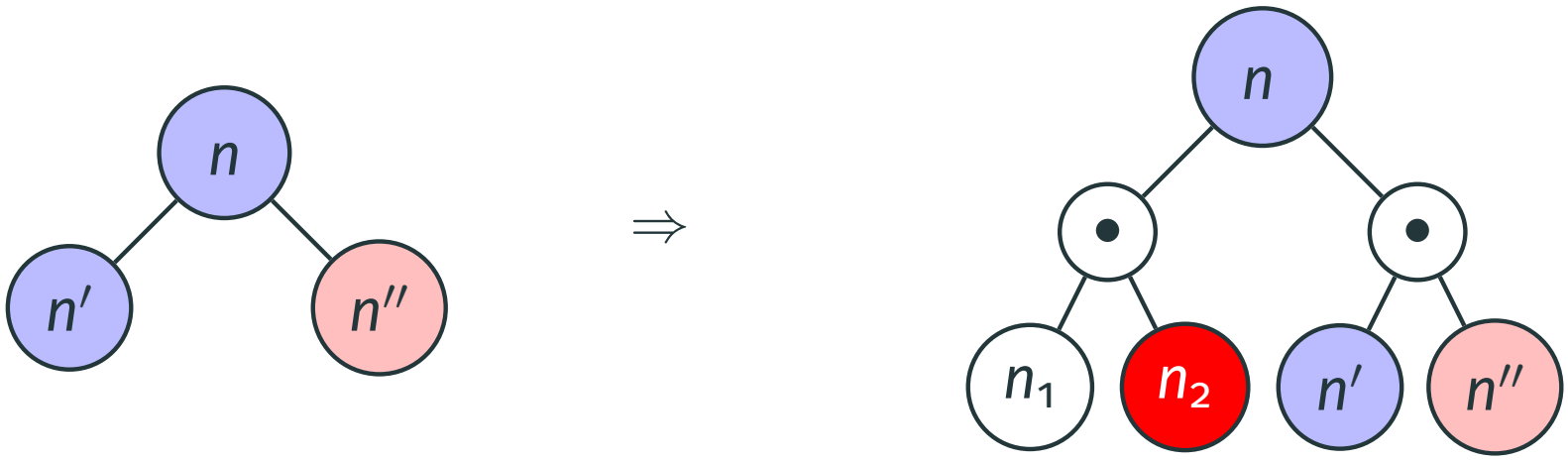


- Rewrite the query to a Boolean query which uses the new nodes n_i to read the valuation of x_i
- This can be done in linear time in the input tree

Hack: adding tree nodes to express the variable assignments

- Query: $Q(x_1, \ldots, x_n)$ with free variables $x_1, \ldots, x_n$
- Goal: find all tuples $a_1, \ldots, a_n$ such that $Q(a_1, \ldots, a_n)$ holds

→ Add special nodes: for each node n and variable x_i, add a node n_i which is colored red iff x_i is the node n



- Rewrite the query to a Boolean query which uses the new nodes n_i to read the valuation of x_i
- This can be done in linear time in the input tree

→ Now, the results are all ways to color the special nodes red and make the Boolean query true

Lower Bound

Existential Marked Ancestor Queries

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

Theorem

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

Lower Bound

Reduction to Query Enumeration

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

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

Image credits

Title slide: Eiffel tower image by Yann Caradec `https://www.flickr.com/photos/la_bretagne_a_paris/35118647963`, license CC-BY-SA 2.0.