A Circuit-Based Approach to Efficient Enumeration: Enumerating MSO Query Results on Trees and Words

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

May 20, 2019

[1]Télécom ParisTech

[2]CNRS CRIStAL

[3]CNRS CRIL

[4]Universität Bayreuth

Roadmap of this talk

- Stefan has presented enumeration for d-DNNF set circuits

Roadmap of this talk

- Stefan has presented enumeration for d-DNNF set circuits
- This talk: introduce MSO query evaluation on trees and build a set circuit that represents the answers to enumerate

Roadmap of this talk

- Stefan has presented enumeration for d-DNNF set circuits
- This talk: introduce MSO query evaluation on trees and build a set circuit that represents the answers to enumerate
- Using the previous talk we can reprove the result:

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

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

Roadmap of this talk

- Stefan has presented enumeration for **d-DNNF set circuits**
- **This talk:** introduce MSO query evaluation on trees and build a **set circuit** that represents the answers to enumerate
- **Using the previous talk** we can reprove the result:

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

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

- **Also:** new results on **combined complexity**:

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

We can enumerate the results of an ***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*

Roadmap of this talk

- Stefan has presented enumeration for **d-DNNF set circuits**
- **This talk:** introduce MSO query evaluation on trees and build a **set circuit** that represents the answers to enumerate
- **Using the previous talk** we can reprove the result:

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

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

- **Also:** new results on **combined complexity**:

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

*We can enumerate the results of an **automaton** on a tree with:*

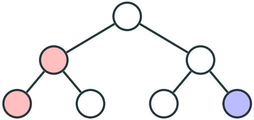
- *Preprocessing **linear** in the tree and **polynomial** in the automaton*
- *Delay **constant** in the tree and **polynomial** in the automaton*

- **Next talk:** how to support **updates** (and prove **new results**)

Boolean MSO on trees

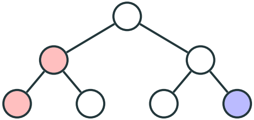
Boolean query evaluation on trees

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



Boolean query evaluation on trees

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



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

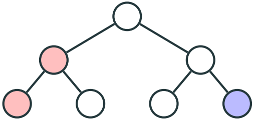
- $P_{\circ}(x)$ means "x is blue"
- $x \rightarrow y$ means "x is the parent of y"

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

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

Boolean query evaluation on trees

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



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

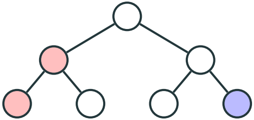
- $P_{\bullet}(x)$ means "x is blue"
- $x \rightarrow y$ means "x is the parent of y"

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

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

Result: TRUE/FALSE indicating if the tree T satisfies the query Q

Boolean query evaluation on trees



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

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

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

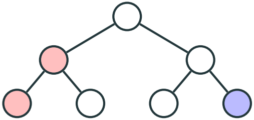
"Is there both a pink and a blue node?"

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

Result: TRUE/FALSE indicating if the tree T satisfies the query Q

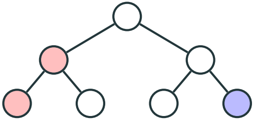
Computational complexity as a function of T
(the query Q is **fixed**)

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"

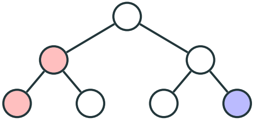
Monadic second-order logic (MSO)



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

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

Monadic second-order logic (MSO)

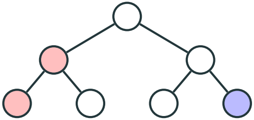


- $P_{\color{blue}\bullet}(x)$ means "x is blue"; also $P_{\color{pink}\bullet}(x)$, $P_{\bigcirc}(x)$
- $x \to y$ means "x is the parent of y"

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

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

Monadic second-order logic (MSO)

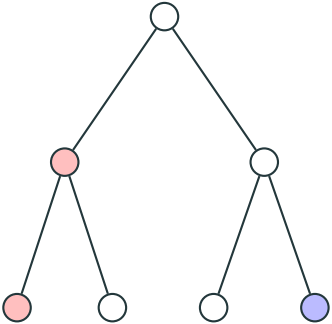


- $P_{\color{blue}\bullet}(x)$ means "x is blue"; also $P_{\color{pink}\bullet}(x)$, $P_{\circ}(x)$
- $x \to y$ means "x is the parent of y"

- **Propositional logic:** formulas with **AND** $\wedge$, **OR** $\vee$, **NOT** $\neg$
  - $P_{\color{pink}\bullet}(x) \wedge P_{\color{blue}\bullet}(y)$ means *"Node x is pink and node y is blue"*
- **First-order logic:** adds **existential quantifier** $\exists$ and **universal quantifier** $\forall$
  - $\exists x\, y\; P_{\color{pink}\bullet}(x) \wedge P_{\color{blue}\bullet}(y)$ 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"*
  - $\forall x\, P_{\color{pink}\bullet}(x) \Rightarrow \exists y\, P_{\color{blue}\bullet}(y) \wedge x \to^* y$
    means *"There is a blue node below every pink node"*

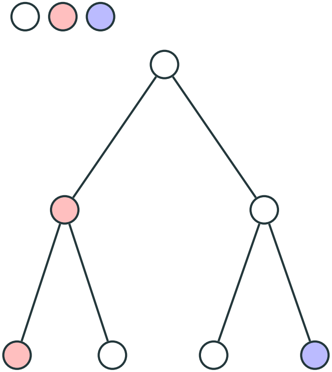
Tree automata

Tree alphabet:



Tree automata

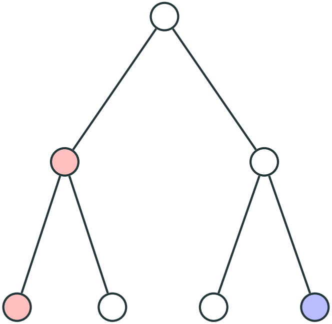
Tree alphabet:



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

Tree automata

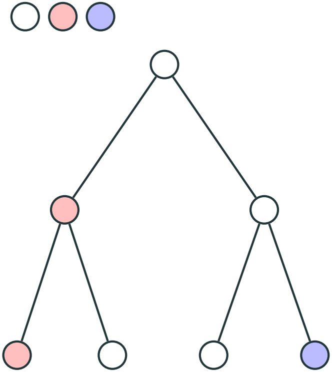
Tree alphabet:



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

Tree automata

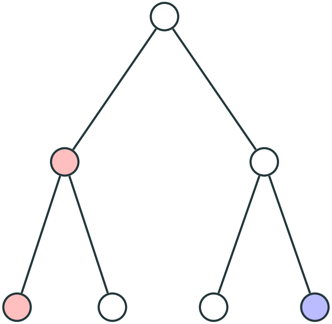
Tree alphabet:



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

Tree automata

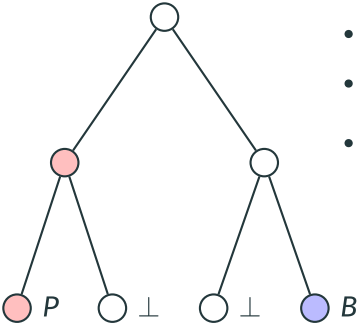
Tree alphabet:



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

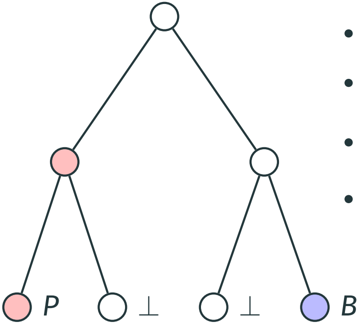
Tree automata

Tree alphabet:



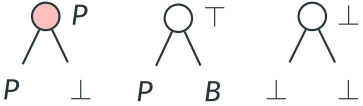
- Bottom-up deterministic tree automaton
- *"Is there both a pink and a blue node?"*
- States: $\{\perp, B, P, \top\}$
- Final states: $\{\top\}$
- Initial function: $\perp$ P B

Tree alphabet:



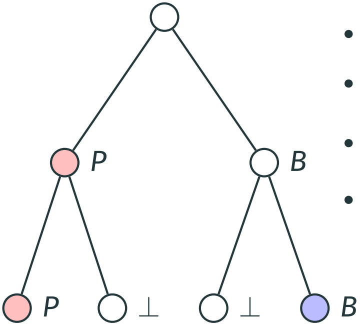
- Bottom-up deterministic tree automaton
- *"Is there both a pink and a blue node?"*
- States: $\{\perp, B, P, \top\}$
- Final states: $\{\top\}$
- Initial function:

- Transitions (examples):



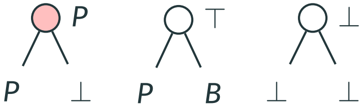
Tree automata

Tree alphabet:



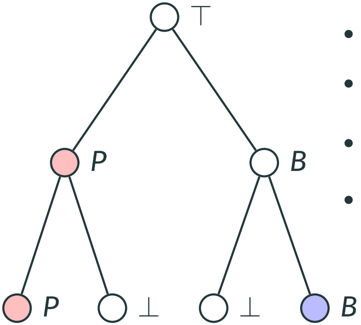
- Bottom-up deterministic tree automaton
- *"Is there both a pink and a blue node?"*
- States: $\{\bot, B, P, \top\}$
- Final states: $\{\top\}$
- Initial function: 

- Transitions (examples):

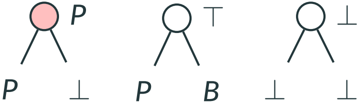


Tree automata

Tree alphabet:



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

$\rightarrow$ *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

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

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

$\rightarrow$ *Complexity (in the query) is generally nonelementary*

Corollary

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

Set Circuits for Non-Boolean MSO Queries

Overall idea

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

Overall idea

- 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

Overall idea

- **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

Overall idea

- **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

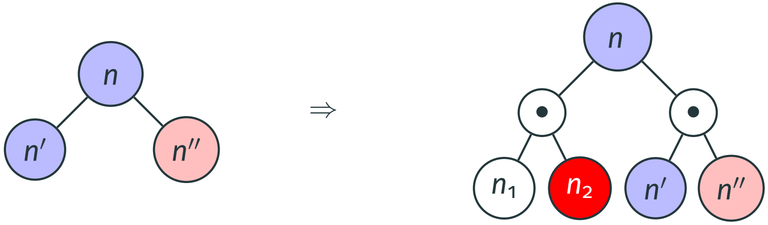
$\rightarrow$ Add **special nodes**: for each node n and variable x_i, add a node n_i which is colored red iff x_i is the node n

- 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

Overall idea

- **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

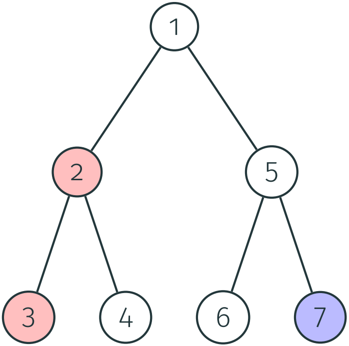
$\rightarrow$ Add **special nodes**: for each node n and variable x_i, add a node n_i which is colored red iff x_i is the node n



- 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
- Remark: same construction for free **second-order variables**

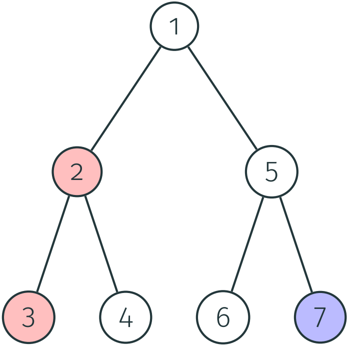
Uncertain trees

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



Uncertain trees

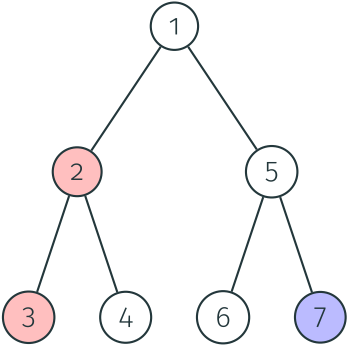
Now: Boolean query on a tree where the color of nodes is uncertain



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

Uncertain trees

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

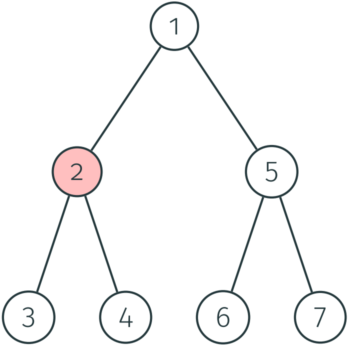


A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

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

Uncertain trees

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

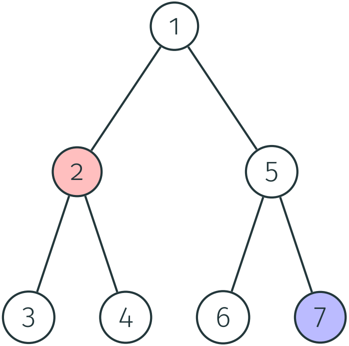


A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

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

Uncertain trees

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

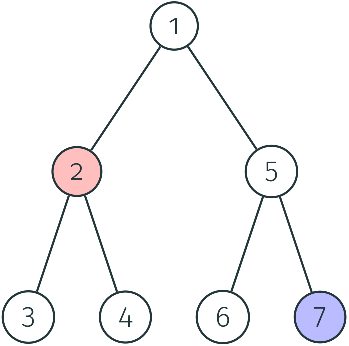


A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

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

Uncertain trees

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



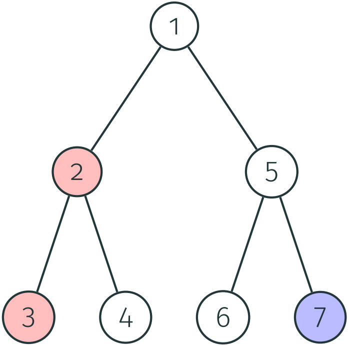
A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

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

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

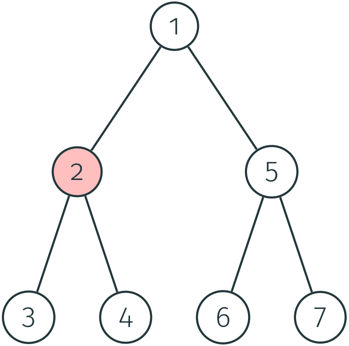
Valuation: $\{2, 3, 7 \mapsto 1, \quad * \mapsto 0\}$

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

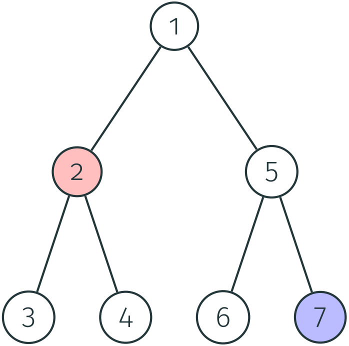
Valuation: $\{2 \mapsto 1, \ * \mapsto 0\}$

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**



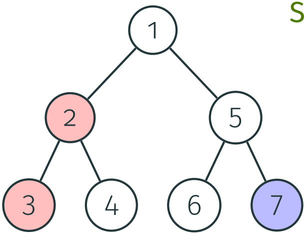
A **valuation** of a tree decides whether to **keep** (1) or **discard** (0) node labels

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

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

The tree automaton A **accepts**

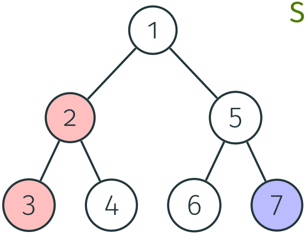
Set circuit



Set circuit:

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

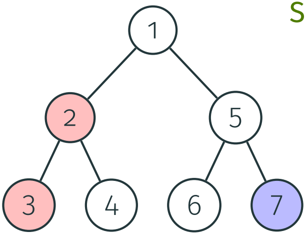
Set circuit



Set circuit:

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

Set circuit

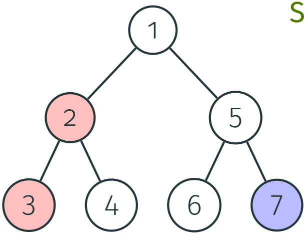


Set circuit:

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

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

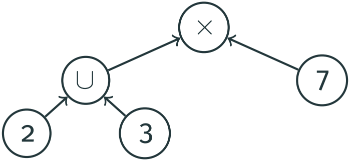
Set circuit



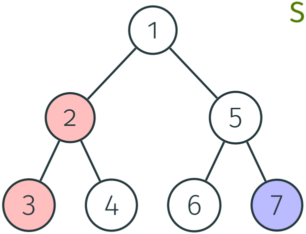
Set circuit:

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

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



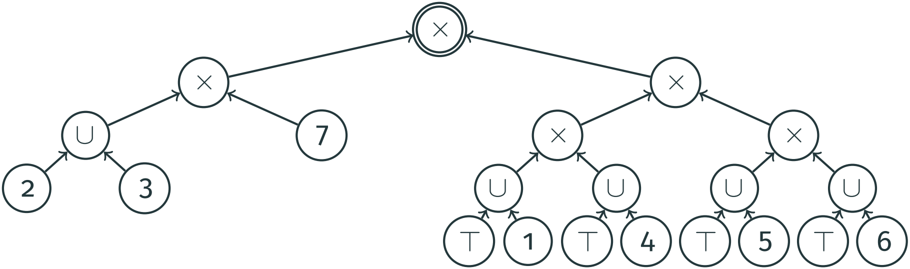
Set circuit



Set circuit:

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

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



Building provenance circuits on trees

Theorem

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

Building provenance circuits on trees

Theorem

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

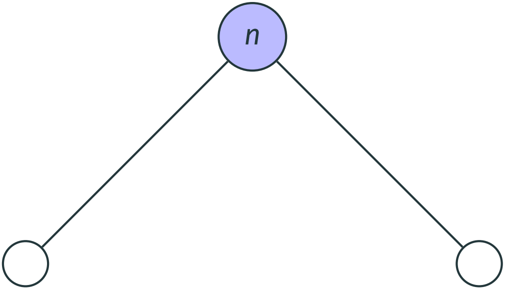
- **Alphabet:** 
- **Automaton:** *"Is there both a pink and a blue node?"*
- **States:** $\{\bot, B, P, \top\}$
- **Final:** $\{\top\}$
- **Transitions:**

Building provenance circuits on trees

Theorem

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

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

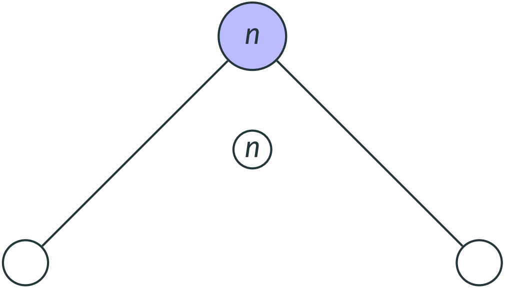


Building provenance circuits on trees

Theorem

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

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

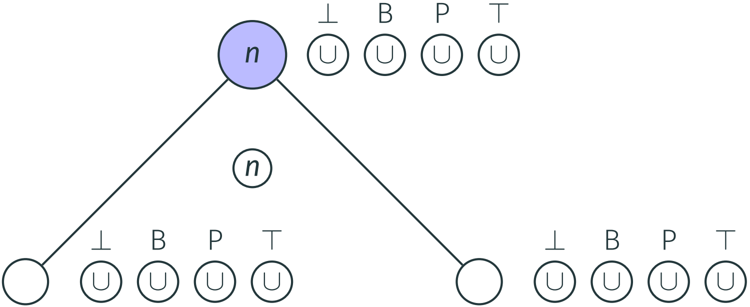


Building provenance circuits on trees

Theorem

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

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

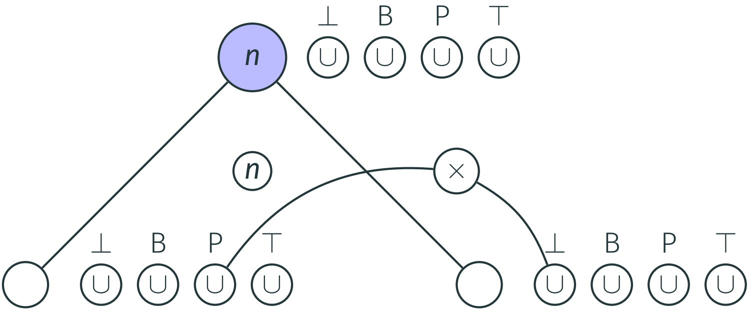


Building provenance circuits on trees

Theorem

*For any bottom-up **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:** ○ (white) ○ (pink) ○ (blue)
- **Automaton:** *"Is there both a pink and a blue node?"*
- **States:** $\{\bot, B, P, \top\}$
- **Final:** $\{\top\}$
- **Transitions:**

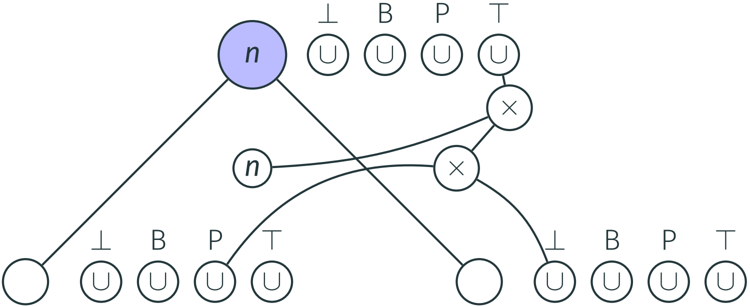


Building provenance circuits on trees

Theorem

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

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

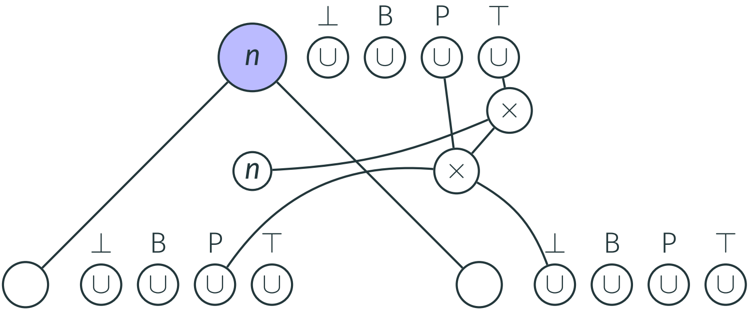


Building provenance circuits on trees

Theorem

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

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



Circuits as factorized representations of query results

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

Circuits as factorized representations of query results

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

Example query:

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

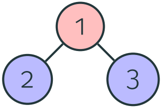
Circuits as factorized representations of query results

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

Example query:

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

Data:



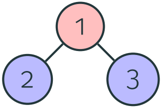
Circuits as factorized representations of query results

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

Example query:

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

Data:



Results:

X_1	X_2
1	2
1	3

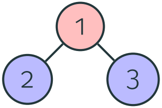
Circuits as factorized representations of query results

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

Example query:

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

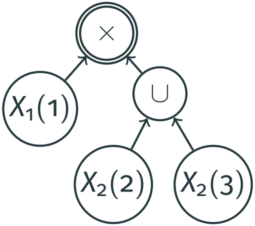
Data:



Results:

X_1	X_2
1	2
1	3

Provenance circuit:



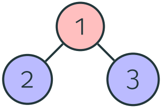
Circuits as factorized representations of query results

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

Example query:

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

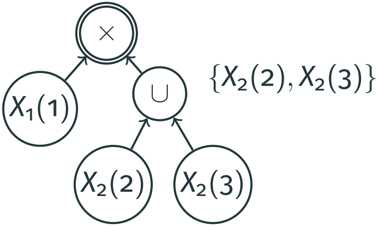
Data:



Results:

X_1	X_2
1	2
1	3

Provenance circuit:



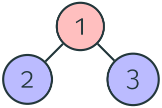
Circuits as factorized representations of query results

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

Example query:

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

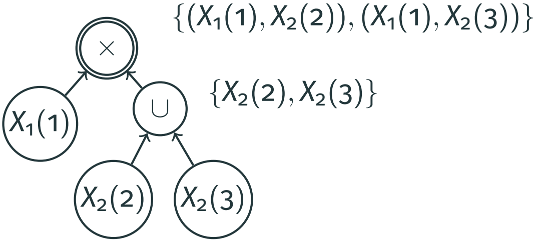
Data:



Results:

X_1	X_2
1	2
1	3

Provenance circuit:



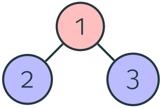
Circuits as factorized representations of query results

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

Example query:

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

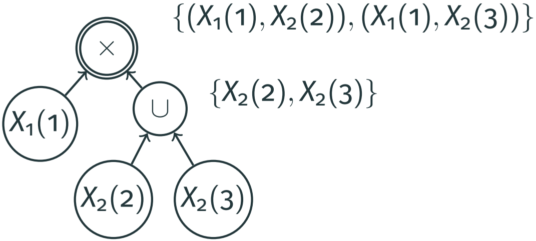
Data:



Results:

X_1	X_2
1	2
1	3

Provenance circuit:



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

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

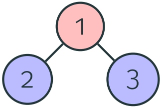
Circuits as factorized representations of query results

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

Example query:

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

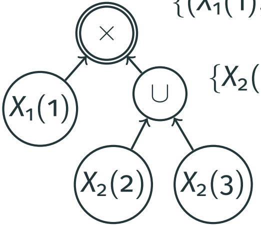
Data:



Results:

X_1	X_2
1	2
1	3

Provenance circuit:



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

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

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

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

Semi-open question: what about memory usage?

Application to Pattern Matching in Texts

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

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

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

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

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

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

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

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

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

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

Goal:

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

Reducing to MSO

- A text is just a tree with a simpler shape

Reducing to MSO

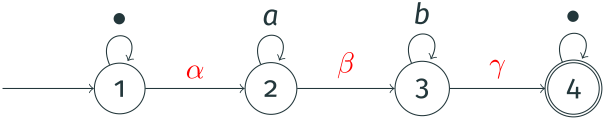
- A text is just a tree with a simpler shape
- A regular expression pattern can be expressed in MSO

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**)

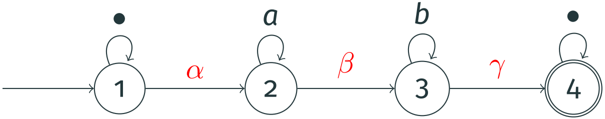
Reducing to MSO

- A text is just a tree with a simpler shape
- A regular expression pattern can be expressed in MSO
  - $\rightarrow$ 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**)



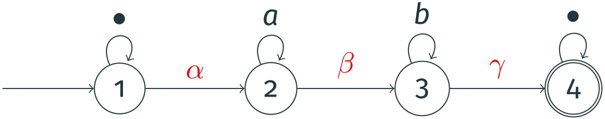
→ The MSO result implies:

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)
- However: our methods adapt to nondeterministic automata
  - Constant-delay enumeration for set circuits without assuming determinism but bounding some notion of width

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)
- However: our methods adapt to **nondeterministic automata**
  - Constant-delay enumeration for set circuits without assuming **determinism** but bounding some notion of **width**

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: enumeration for regular expression patterns on text

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?
    - $\rightarrow$ factors of maximal/minimal size
    - $\rightarrow$ distinct matching strings
    - $\rightarrow$ 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?
    - $\rightarrow$ factors of maximal/minimal size
    - $\rightarrow$ distinct matching strings
    - $\rightarrow$ etc.
  - Which application domains need this?
  - Are there good benchmarks?

Other problems?

- Counting the number of solutions:
  - Can be done with unambiguous automata and d-DNNF set circuits
  - With nondeterministic automata: hard [Florenzano et al., 2018]

Other problems?

- **Counting** the number of solutions:
  - Can be done with **unambiguous automata** and d-DNNF set circuits
  - With **nondeterministic automata**: hard [Florenzano et al., 2018]
- **Testing** if a tuple is a solution:
  - We **don't see how to do it** (unlike [Kazana and Segoufin, 2013])
  - **Open problem:** is there a good reason why?

Other problems?

- Counting the number of solutions:
  - Can be done with unambiguous automata and d-DNNF set circuits
  - With nondeterministic automata: hard [Florenzano et al., 2018]
- Testing if a tuple is a solution:
  - We don't see how to do it (unlike [Kazana and Segoufin, 2013])
  - Open problem: is there a good reason why?
- Updates: see next talk :)

Other problems?

- Counting the number of solutions:
  - Can be done with unambiguous automata and d-DNNF set circuits
  - With nondeterministic automata: hard [Florenzano et al., 2018]
- Testing if a tuple is a solution:
  - We don't see how to do it (unlike [Kazana and Segoufin, 2013])
  - Open problem: is there a good reason why?
- Updates: see next talk :)

Thanks for your attention!

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).
- 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.