Circuits for Querying trees: a little survey

Pierre Bourhis

SPIRALS Team, CRIStAL, CNRS, University of Lille, INRIA Lille

Thanks

Thanks to my collaborators with whom I worked on this topics

Antoine Amarilli

Alejandro Grez

Louis Jachiet

Stefen Mengel

Matthias Niewerth

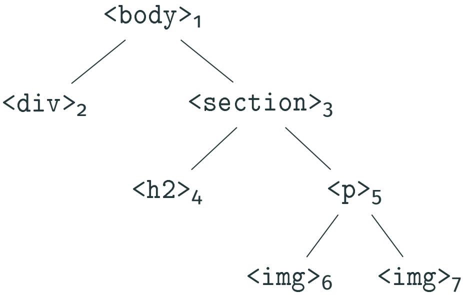
Cristian Riveros

Thanks to Antoine Amarill for part of the slides.

Querying Trees

Tree as representation of data

Tree is a classical data structure to represent data into different contexts.



MSO is the classical language to express Boolean queries over trees.

The other classical formalism for express Boolean queries is tree automaton.

More Complex Queries over trees

General MSO queries:

- MSO with **first order free variables**: returning tuples of nodes
- MSO with **second order free variables**: returning tuples of **sets** of nodes

Extension of MSO queries and trees:

- **Counting** number of solutions
- Query over **probabilistic** tree representation [Cohen et al., 2009]
- **Enumeration** of solutions for a MSO formula with first order variables [Bagan, 2006, Kazana and Segoufin, 2013]

Maintaining an answer through **updates** of the tree

Complex Queries Evaluation over trees are simple

MSO evaluation is in linear time in the size of the tree

- Counting number of solutions is in linear time in the size of the tree
- Query over probabilistic tree representation is in linear time in the size of the tree
- Enumeration of solutions for a MSO formula with first order variables can be done with a linear time preprocessing and a constant delay

Why MSO complex query evaluation over trees is simpler than conjunctive query complex query evaluation over relational database ?

Representing the solutions of a MSO query

How to represent the solutions of a MSO evaluation?

A partial answer through the notion of provenance [Amarilli et al., 2015].

Theorem

Provenance of a MSO query over tree can be computed in linear time by a circuit with a bounded tree width

Boolean circuits

A Boolean circuit represents a set of answers to a pattern $P(\alpha, \beta)$

Boolean circuits

A Boolean circuit represents a set of answers to a pattern $P(\alpha, \beta)$

- Singleton α:6 $\rightarrow$ *"the variable α is mapped to node* 6*"*

Boolean circuits

A Boolean circuit represents a set of answers to a pattern $P(\alpha, \beta)$

- Singleton $\alpha{:}6 \rightarrow$ *"the variable α is mapped to node 6"*
- Tuple $\langle \alpha{:}4, \beta{:}6 \rangle$: tuple of singletons

Boolean circuits

A **Boolean circuit** represents a **set of answers** to a pattern $P(\alpha, \beta)$

- **Singleton** $\alpha{:}6 \rightarrow$ *"the variable α is mapped to node 6"*
- **Tuple** $\langle \alpha{:}4, \beta{:}6 \rangle$: tuple of singletons
- The circuit captures a **set** of tuples, e.g., $\{\langle \alpha{:}4, \beta{:}6 \rangle, \langle \alpha{:}4, \beta{:}7 \rangle\}$

Approach

- The answers of the query are the satisfying assignments

Approach

- The answers of the query are the satisfying assignments
- These circuits fall in restricted circuit classes
  that allow for efficient complex operations

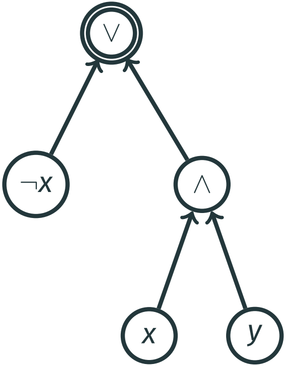
Approach

- The answers of the query are the **satisfying assignments**
- These circuits fall in **restricted circuit classes** that allow for efficient complex operations

$\rightarrow$ **Task:** Given a **Boolean circuit**, how to efficiently operate the complex operation?

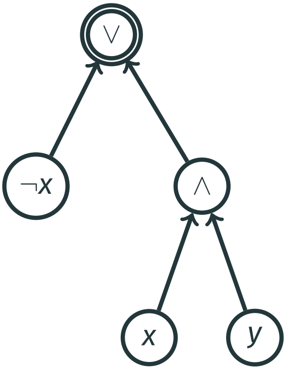
Boolean circuits

Boolean circuits



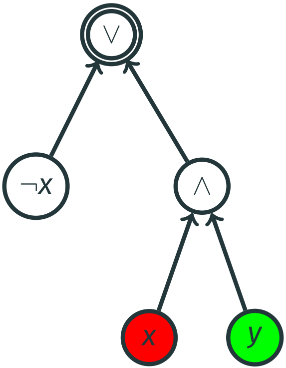
- Directed acyclic graph of **gates**
- **Output** gate: 
- **Literal** gates: x , $\neg x$
- **Internal** gates: $\lor$ $\land$ $D(x)$

Boolean circuits



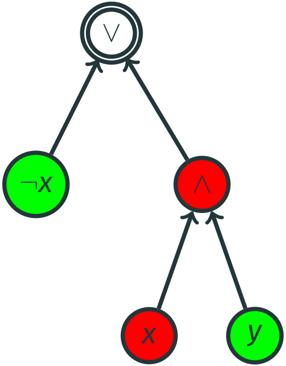
- Directed acyclic graph of **gates**
- **Output** gate: ◎
- **Literal** gates: x , $\neg x$
- **Internal** gates: $\vee$ $\wedge$ $D(x)$
- **Valuation**: function from variables to $\{0, 1\}$
  Example: $\nu = \{x \mapsto 0,\ y \mapsto 1\}$...

Boolean circuits



- Directed acyclic graph of **gates**
- **Output** gate: 
- **Literal** gates: x , $\neg x$
- **Internal** gates: $\vee$ $\wedge$ $D(x)$
- **Valuation**: function from variables to $\{0, 1\}$
  Example: $\nu = \{x \mapsto 0,\ y \mapsto 1\}$...

Boolean circuits



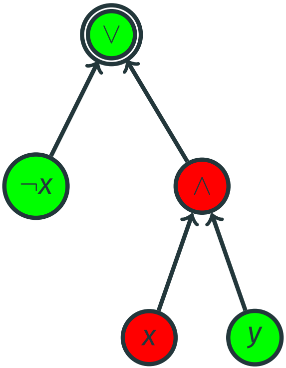
- Directed acyclic graph of **gates**
- **Output** gate: 
- **Literal** gates: 

, 

- **Internal** gates: 

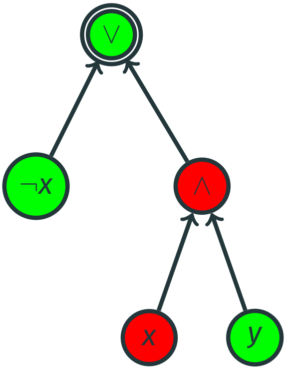
- **Valuation**: function from variables to $\{0, 1\}$
  Example: $\nu = \{x \mapsto 0,\ y \mapsto 1\}$...

Boolean circuits



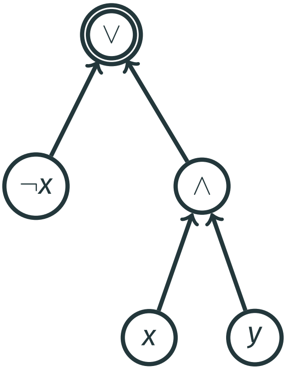
- Directed acyclic graph of **gates**
- **Output** gate: 
- **Literal** gates: x , $\neg x$
- **Internal** gates: $\vee$ $\wedge$ $D(x)$
- **Valuation**: function from variables to $\{0, 1\}$
  Example: $\nu = \{x \mapsto 0,\ y \mapsto 1\}$... mapped to 1

Boolean circuits



- Directed acyclic graph of **gates**
- **Output** gate: 
- **Literal** gates: x , $\neg x$
- **Internal** gates: $\vee$ $\wedge$ $D(x)$
- **Valuation**: function from variables to $\{0, 1\}$
  Example: $\nu = \{x \mapsto 0,\ y \mapsto 1\}$... mapped to 1
- **Assignment**: set of variables mapped to 1
  Example: $S_\nu = \{y\}$; more concise than ν

Boolean circuits



- Directed acyclic graph of **gates**
- **Output** gate: ◎
- **Literal** gates: x , $\neg x$
- **Internal** gates: $\lor$ $\land$ $D(x)$
- **Valuation**: function from variables to $\{0, 1\}$
  Example: $\nu = \{x \mapsto 0,\ y \mapsto 1\}$... mapped to 1
- **Assignment**: set of variables mapped to 1
  Example: $S_\nu = \{y\}$; more concise than ν

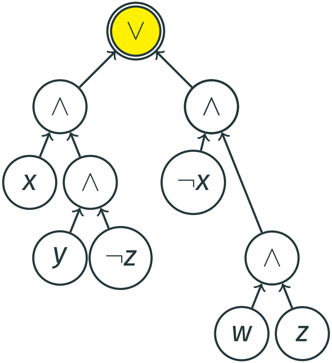
Our task: Enumerate all **satisfying assignments** of an input circuit

Circuit restrictions

d-DNNF:

- ∨ are all **deterministic**:

The inputs are **mutually exclusive** (= no valuation ν makes two inputs simultaneously evaluate to 1)



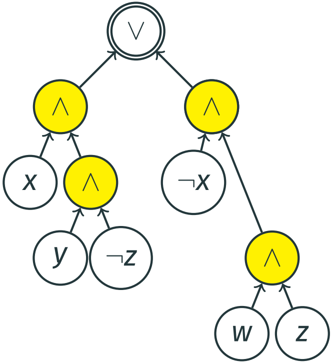
d-DNNF:

- (∨) are all **deterministic**:

The inputs are **mutually exclusive**
(= no valuation ν makes two inputs simultaneously evaluate to 1)

- (∧) are all **decomposable**:

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



Circuit restrictions

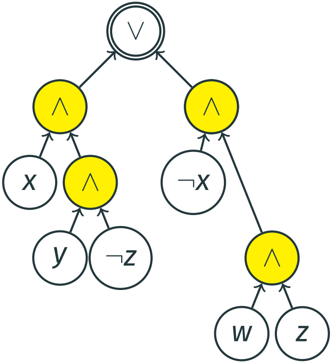
d-DNNF:

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

The inputs are **mutually exclusive** (= no valuation ν makes two inputs simultaneously evaluate to 1)

- $\land$ are all **decomposable**:

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



Smoothed and zero suppressed semantics

Smooth Circuit

- $\vee$ are all **smoothed**:

the valuations defined the subcircuits are defined on the same set of variables. To smooth, we need to consider all valuations over the missing variables.

Smoothed and zero suppressed semantics

Smooth Circuit

- $\bigcirc\!\!\!\!\!\vee$ are all **smoothed**:

the valuations defined the subcircuits are defined on the same set of variables. To smooth, we need to consider all valuations over the missing variables.

Zero Suppressed Semantics (ZSS) circuit

- Semantics for assignements and not valuations

Main Results for Querying circuits

Computing K semi-ring value over d-DNNF

Let K be a commutative semi-ring. Let S be a set of assignment over a set of variables $\mathcal{X}$. Let ν be a cost function from the literals of $\mathcal{X}$ to K. We can generalize ν to S such that

$\sum_{\rho \in S} (\prod_{x \text{ s.t } \rho(x)=1} \nu(x)).(\prod_{x \text{ s.t } \rho(x)=0} \nu(\neg x))$

Theorem

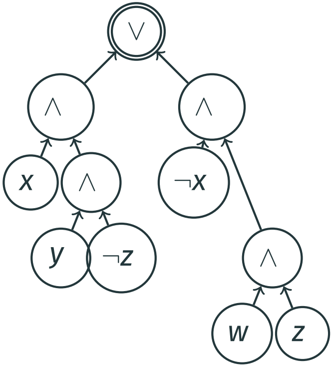
Given a smoothed d-DNNF circuit C over a set of variables $\mathcal{X}$, let K be a commutative semi-ring and ν be a cost function from the literals of $\mathcal{X}$ to K, $\nu(C)$ can be computed in linear time in $|C|$.

Smoothing is not needed for positive cost functions i.e when negative literals are associated with $\mathbf{1}$ and zss d-DNNF.

In particular, counting valuations and probabilistic evaluation can be done in linear time in $|C|$.

Example

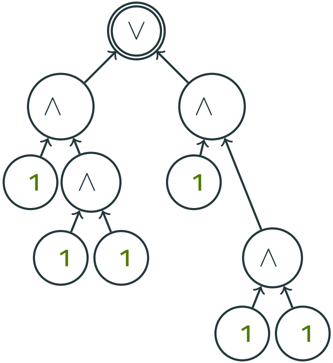
Counting the possibles valuations using $(\mathbb{N}, +, \cdot, 0, 1)$.



Example

Counting the possibles valuations using $(\mathbb{N}, +, \cdot, 0, 1)$.

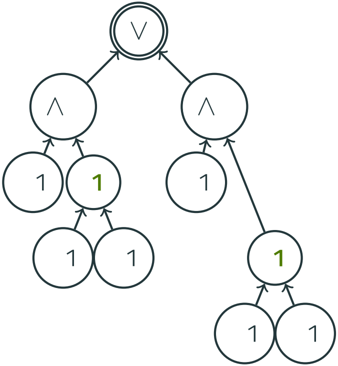
ν maps each literal to 1.



Example

Counting the possibles valuations using $(\mathbb{N}, +, \cdot, 0, 1)$.

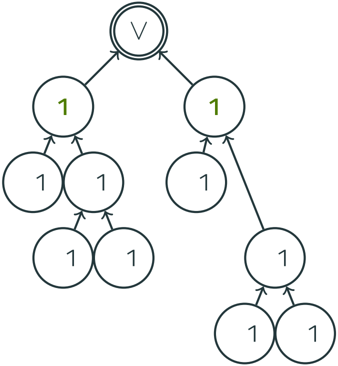
Propagate the values in bottom-up manner associating $\wedge$ to $\cdot$ and $\vee$ to $+$.



Example

Counting the possibles valuations using $(\mathbb{N}, +, \cdot, 0, 1)$.

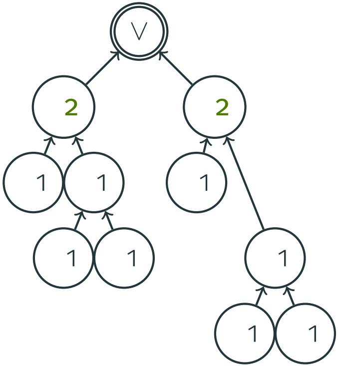
Propagate the values in bottom-up manner associating $\wedge$ to $\cdot$ and $\vee$ to $+$.



Example

Counting the possibles valuations using $(\mathbb{N}, +, \cdot, 0, 1)$.

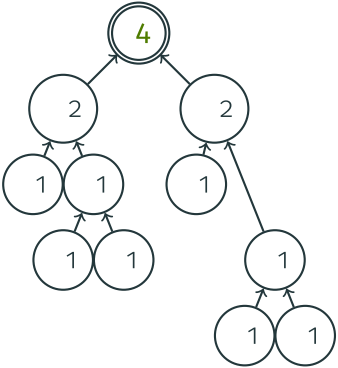
Partial valuations are not on the same variables $\{x, y, z\}$ on the left and $\{x, w, z\}$ on the right. We need to smooth them.



Example

Counting the possibles valuations using $(\mathbb{N}, +, \cdot, 0, 1)$.

Propagate the values in bottom-up manner associating $\wedge$ to $\cdot$ and $\vee$ to $+$.



Enumeration

Theorem

Given a zss d-DNNF circuit C, we can enumerate its satisfying assignements with preprocessing linear in $|C|$ and delay linear in the size of each assignment

Enumeration

Subtleties: Dealing with part of the circuits with empty partial assignements, memory usage problems

In practice: we enumerate the $\textcircled{x}$ and $\textcircled{\wedge}$ of the acceptance subtree tree of the partial assignments in C.

Key structure [Amarilli et al., 2017] : a persistent set structure for which the following operations are in $O(1)$

- adding an element
- giving an arbitrary element and deleting this element
- union of two sets

The preprocessing is a bottom-up evaluation.

Ranked Enumeration

Theorem

*Given a zss d-DNNF circuit C and a strong subset-monotone ranking function on the partial assignements, we can enumerate the assignments following the order given by their cost with preprocessing linear in $|C|$
and delay $O(\log(k+1) \cdot \max(|\alpha|))$, where $|\alpha|$ is the size of assignment and k is the number of solutions already enumerated.*

In practice: we ranked enumerate the $\times$ and $\wedge$ of the acceptance subtree of the partial assignments in C.

Key structure Brodal Queue [Brodal, 1996]: persistence priority queue with the following properties

- adding a pair (element, value) in $O(1)$
- giving an maximuml pair (element,value) respecting the order over the values in $O(1)$
- union of two sets in $O(1)$
- deleting a maximum pair in $O(\log |S|)$

Cost of Smoothing

In general smoothing is costly, the size of the new circuit is in $O(|C|^2)$.

Better cases

- structured d-DNNF [Shih et al., 2019], still above $O(|C|)$
- ordered d-DNNF [Amarilli et al., 2017], in $O(|C|)$ but with particular new gates

Construction of the circuit representing the answers of a MSO Query

Theorem

*For any **tree automaton** A with capture variables $\alpha_1, \ldots, \alpha_k$, given a **tree** T, we can build in $O(|T| \times |A|)$ a **smoothed circuit** capturing exactly the set of tuples $\{\langle \alpha_1 : n_1, \ldots, \alpha_k : n_k \rangle$ in the output of A on T*

Theorem

*For any **tree automaton** A with capture variables $\alpha_1, \ldots, \alpha_k$, given a **tree** T, we can build in $O(|T| \times |A|)$ a **zss circuit** capturing exactly the set of tuples $\{\langle \alpha_1 : n_1, \ldots, \alpha_k : n_k \rangle$ in the output of A on T*

Proof idea for trees: circuit construction (details)

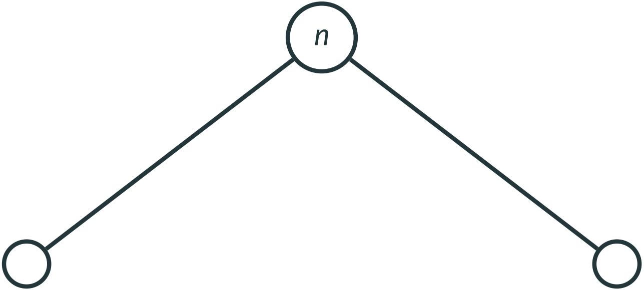
- **Automaton:** *"Select all node pairs* (α, β)*"*

- **States:** $\{\emptyset, \alpha, \beta, \alpha\beta\}$
- **Rules:** $\{\beta, \emptyset \longrightarrow \beta,$
  $\beta, \emptyset, \alpha : n \longrightarrow \alpha\beta$
  $\cdots\}$

Proof idea for trees: circuit construction (details)

- Automaton: *"Select all node pairs* (α, β)*"*

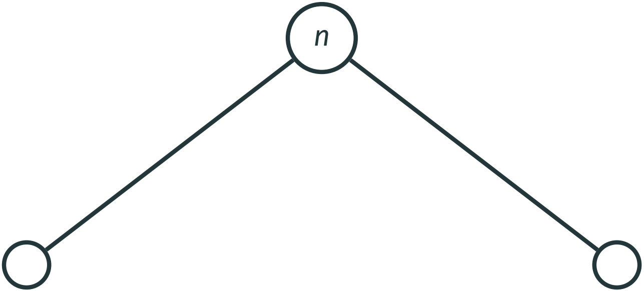
- States: $\{\emptyset, \alpha, \beta, \alpha\beta\}$
- Rules: $\{\beta, \emptyset \longrightarrow \beta,$
  $\beta, \emptyset, \alpha : n \longrightarrow \alpha\beta$
  $\dots\}$



Proof idea for trees: circuit construction (details)

- **Automaton:** *"Select all node pairs* (α, β)*"*

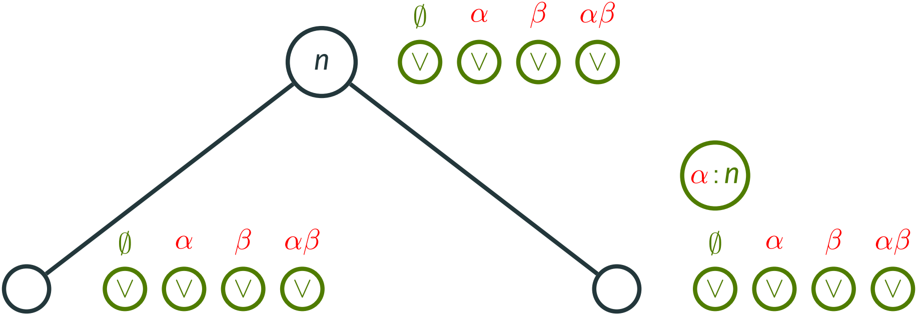
- **States:** $\{\emptyset, \alpha, \beta, \alpha\beta\}$
- **Rules:** $\{\beta, \emptyset \longrightarrow \beta,$
  $\beta, \emptyset, \alpha : n \longrightarrow \alpha\beta$
  $\dots\}$



Proof idea for trees: circuit construction (details)

- **Automaton:** *"Select all node pairs* (α, β)*"*

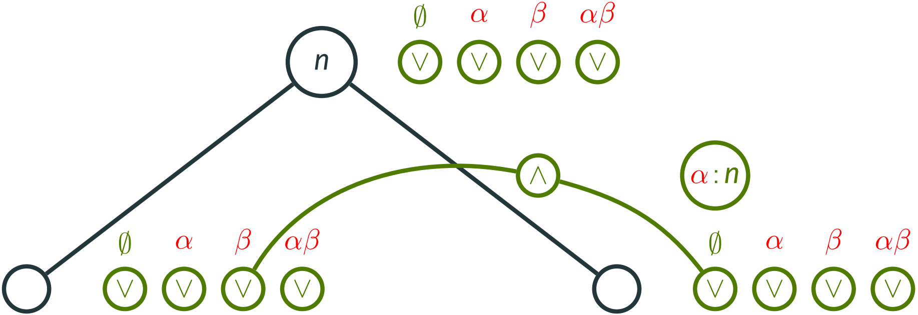
- **States:** $\{\emptyset, \alpha, \beta, \alpha\beta\}$
- **Rules:** $\{\beta, \emptyset \longrightarrow \beta,$
  $\beta, \emptyset, \alpha : n \longrightarrow \alpha\beta$
  $\cdots\}$



Proof idea for trees: circuit construction (details)

- **Automaton:** *"Select all node pairs* (α, β)*"*

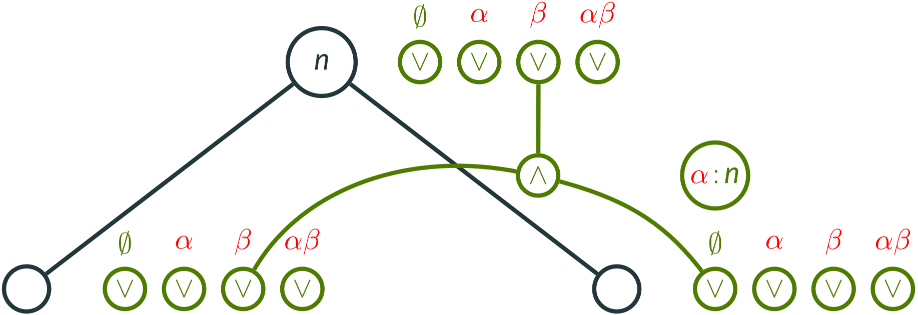
- **States:** $\{\emptyset, \alpha, \beta, \alpha\beta\}$
- **Rules:** $\{\beta, \emptyset \longrightarrow \beta,$
  $\beta, \emptyset, \alpha : n \longrightarrow \alpha\beta$
  $\dots\}$



Proof idea for trees: circuit construction (details)

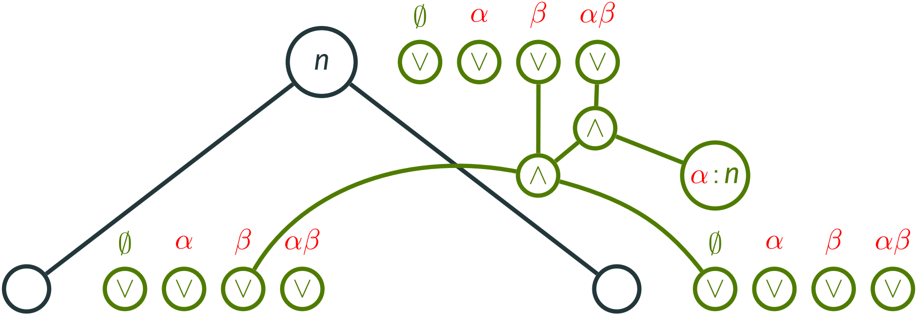
- **Automaton:** *"Select all node pairs* (α, β)*"*

- **States:** $\{\emptyset, \alpha, \beta, \alpha\beta\}$
- **Rules:** $\{\beta, \emptyset \longrightarrow \beta,$
  $\beta, \emptyset, \alpha : n \longrightarrow \alpha\beta$
  $\cdots\}$



Proof idea for trees: circuit construction (details)

- **Automaton:** *"Select all node pairs* (α, β)*"*
- **States:** $\{\emptyset, \alpha, \beta, \alpha\beta\}$
- **Rules:** $\{\beta, \emptyset \longrightarrow \beta,$ $\beta, \emptyset, \alpha : n \longrightarrow \alpha\beta$ $\ldots\}$



Summary

We can reproof the following complex MSO queries over trees:

- Counting number of solutions
- Query over probabilistic tree representation [Cohen et al., 2009]
- Enumeration of solutions
  [Bagan, 2006, Kazana and Segoufin, 2013]

For MSO with first order variables, we need to notice that the size of the corresponding assignments is bounded by the size of Q

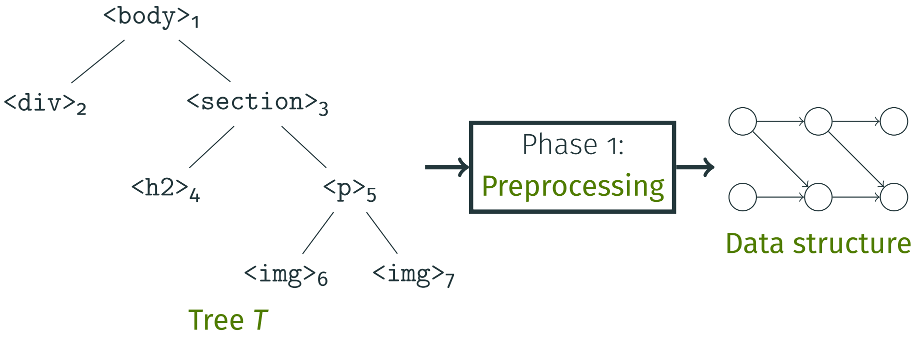
New Result

Theorem

For any fixed MSO query $Q(x_1, \ldots, x_n)$ *with free first-order variables, given as input a tree* T *and a subset-monotone ranking function* w *on the partial assignments of* $x_1, \ldots, x_n$ *to nodes of* T*, we can enumerate the answers to* Q *on* T *in nonincreasing order of scores according to* w *with a preprocessing time of* $O(|T|)$ *and a delay of* $O(\log(K+1))$*, where* K *is the number of answers produced so far enumerated.*

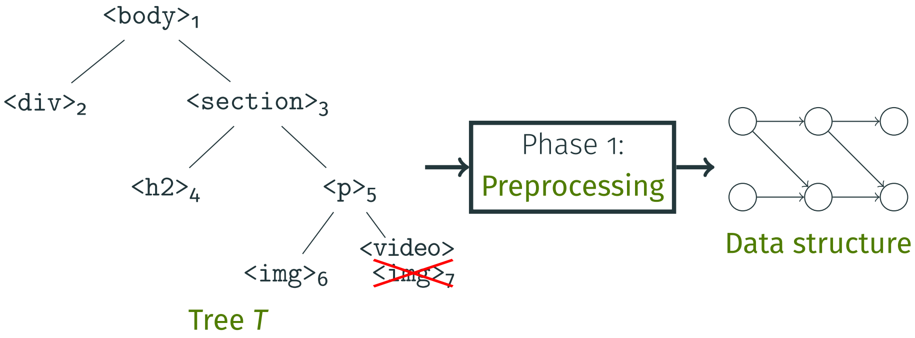
Extension: Handling Updates

Updates



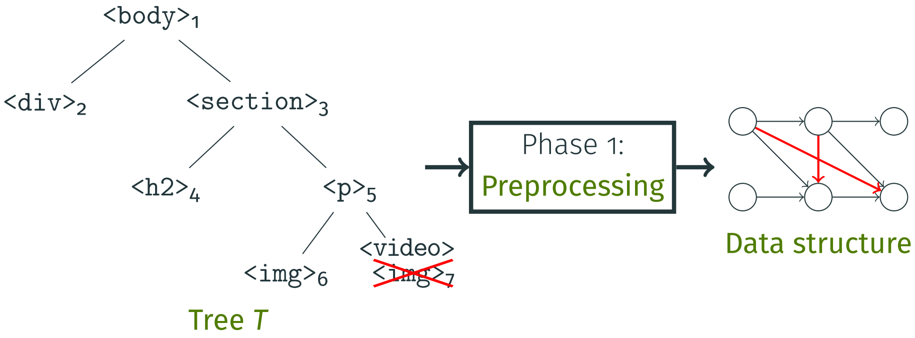
- The input data can be **modified** after the computation

Updates



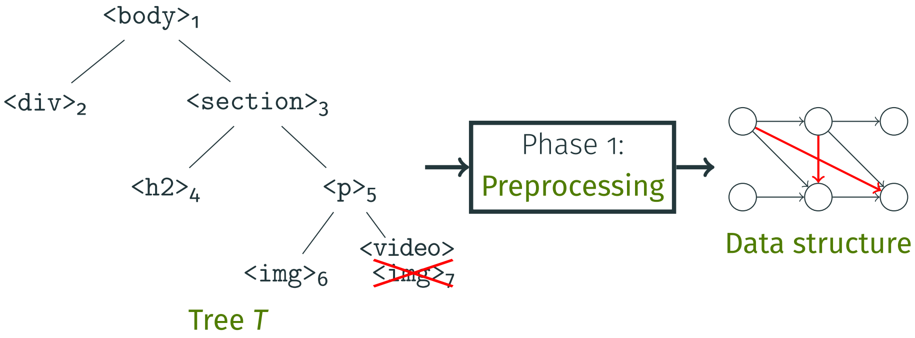
- The input data can be **modified** after the computation

Updates

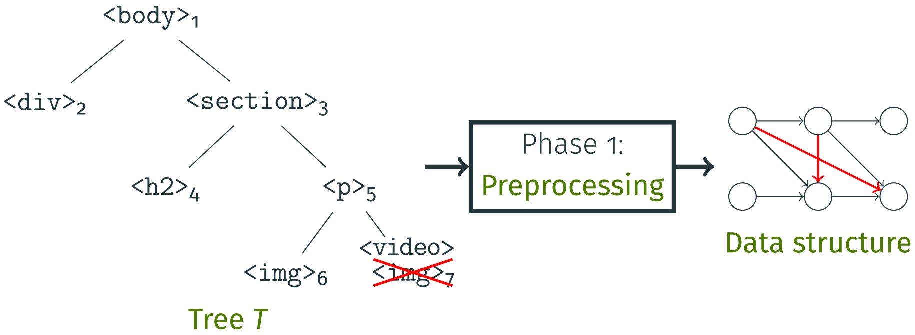


- The input data can be **modified** after the computation

Updates



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



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

→ Can we **do better**?

Known results on dynamic trees

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

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

Known results on dynamic trees

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

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

Known results on dynamic trees

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

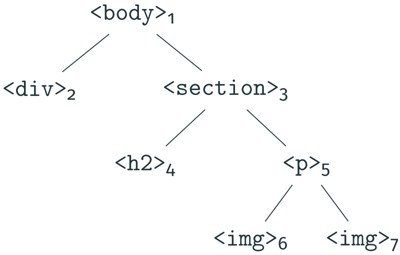
Work	Data	Preproc.	Delay	Updates
[Bagan, 2006], [Kazana and Segoufin, 2013]	trees	$O(T)$	$O(1)$	$O(T)$
[Losemann and Martens, 2014]	trees	$O(T)$	$O(\log^2 T)$	$O(\log^2 T)$
[Losemann and Martens, 2014]	text	$O(T)$	$O(\log T)$	$O(\log T)$

Known results on dynamic trees

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

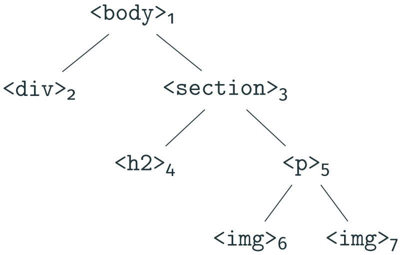
Work	Data	Preproc.	Delay	Updates
[Bagan, 2006], [Kazana and Segoufin, 2013]	trees	$O(T)$	$O(1)$	$O(T)$
[Losemann and Martens, 2014]	trees	$O(T)$	$O(\log^2 T)$	$O(\log^2 T)$
[Losemann and Martens, 2014]	text	$O(T)$	$O(\log T)$	$O(\log T)$
[Niewerth and Segoufin, 2018]	text	$O(T)$	$O(1)$	$O(\log T)$

Relabelings



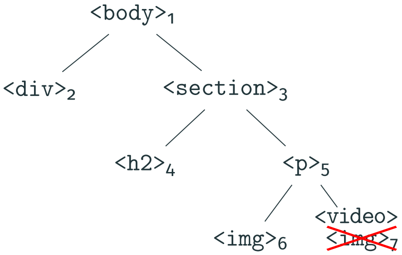
- Special kind of updates: **relabelings** that change the label of a node

Relabelings



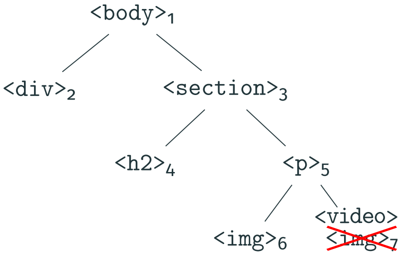
- Special kind of updates: relabelings that change the label of a node
- Example: relabel node 7 to `<video>`

Relabelings



- Special kind of updates: **relabelings** that change the label of a node
- **Example:** relabel node 7 to `<video>`

Relabelings



- Special kind of updates: **relabelings** that change the label of a node
- **Example:** relabel node `7` to `<video>`
- The tree's **structure** never changes

New results on dynamic trees

- If we allow only relabeling updates, we can show:

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

New results on dynamic trees

- If we allow only relabeling updates, we can show:

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

New results on dynamic trees

- If we allow only relabeling updates, we can show:

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

New results on dynamic trees

- If we allow only relabeling updates, we can show:

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

Idea of the technique

Theorem

Let Q be a MSO query and T be a tree. Let $C_{Q,T}$ be the circuit representing the set of answer $Q(T)$. Let U be an update on T, then the update of C can be done in the depth of C which in $O(\mathrm{depth}(T))$.

Problem: the depth of T of can be linear in $|T|$.

For relabeling, we need to balance the tree during the preprocessing. It can be done in $O(T)$ [Bodlaender and Hagerup, 1998].

In general, we need to **rebalance** the tree and to continue to balance the tree after an update.

[Balmin et al., 2004] ensure to maintain a representation of the tree ensuring a depth in $O(\log^2(T))$

[Kleest-Meißner et al., 2022] proposes to maintain a representation of a tree ensuring a depth in $O(\log(T))$.

Summary and Future Work

Summary

Complex evaluation of MSO queries over trees can be done efficiently

We present an unifying framework to reproof known results based on particular circuits : smoothed/zss d-DNNF

Our framework shows that the incremental maintenance through these circuits is efficient too

Future work

New types of queries to consider from databases:

- Direct Access
- Uniform Sampling
- Generalizing enumeration of weighted MSO on word [Bourhis et al., 2021] to trees
- ...

It is just sufficient to study these problems over our particular circuits

Future work

New types of queries to consider from databases:

- Direct Access
- Uniform Sampling
- Generalizing enumeration of weighted MSO on word [Bourhis et al., 2021] to trees
- ...

It is just sufficient to study these problems over our particular circuits

Thanks for your attention!

- Amarilli, A., Bourhis, P., Jachiet, L., and Mengel, S. (2017).
  A circuit-based approach to efficient enumeration.
  In *ICALP*.
- Amarilli, A., Bourhis, P., and Mengel, S. (2018).
  Enumeration on trees under relabelings.
  In *ICDT*.
- Amarilli, A., Bourhis, P., and Senellart, P. (2015).
  Provenance circuits for trees and treelike instances.
  In *ICALP*.
- Bagan, G. (2006).
  MSO queries on tree decomposable structures are computable with linear delay.
  In *CSL*.

Balmin, A., Papakonstantinou, Y., and Vianu, V. (2004).
Incremental validation of XML documents.
TODS.

Bodlaender, H. L. and Hagerup, T. (1998).
Parallel algorithms with optimal speedup for bounded treewidth.
SIAM Journal on Computing.

Bourhis, P., Grez, A., Jachiet, L., and Riveros, C. (2021).
Ranked enumeration of MSO logic on words.
In *ICDT*.

Brodal, G. S. (1996).
Worst-case efficient priority queues.
In *SODA*.

Cohen, S., Kimelfeld, B., and Sagiv, Y. (2009).
Running tree automata on probabilistic XML.
In *PODS*.

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

Kleest-Meißner, S., Marasus, J., and Niewerth, M. (2022).
MSO queries on trees: Enumerating answers under updates using forest algebras.
CoRR, abs/2208.04180.

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

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

Shih, A., den Broeck, G. V., Beame, P., and Amarilli, A. (2019).
Smoothing structured decomposable circuits.
In *NeurIPS*.