Enumeration on Trees under Relabelings

Antoine Amarilli[1], Pierre Bourhis[2], Stefan Mengel[3]

March 27th, 2018

[1]Télécom ParisTech

[2]CNRS CRIStAL

[3]CNRS CRIL

Problem statement

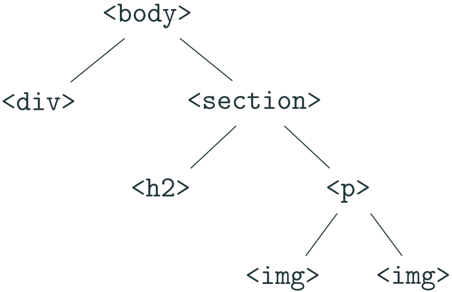
Problem: Query evaluation on trees

- Tree on a fixed alphabet
- Boolean query to test a property of the tree

Problem: Query evaluation on trees

- Tree on a fixed alphabet
- Boolean query to test a property of the tree

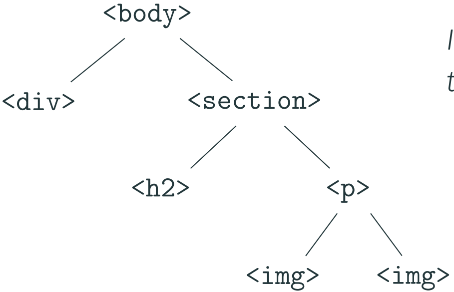
Example tree



Problem: Query evaluation on trees

- **Tree** on a fixed alphabet
- Boolean **query** to test a property of the tree

Example tree



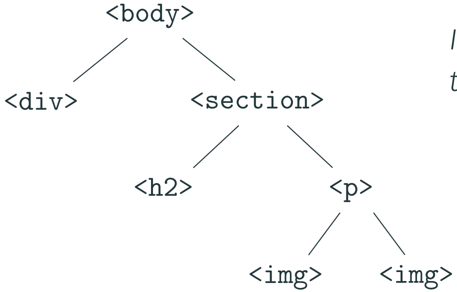
Example query

*Is there an **h2** header and an **image** that are in the same section?*

Problem: Query evaluation on trees

- **Tree** on a fixed alphabet
- Boolean **query** to test a property of the tree

Example tree



Example query

*Is there an **h2** header and an **image** that are in the same section?*

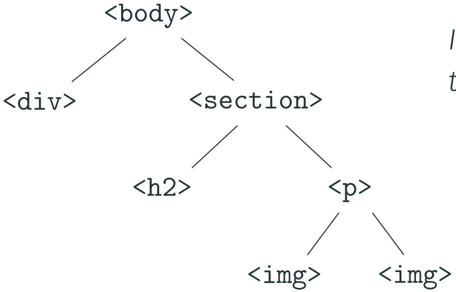
Example answer

$\rightarrow$YES

Problem: Query evaluation on trees

- **Tree** on a fixed alphabet
- Boolean **query** to test a property of the tree

Example tree



Example query

*Is there an **h2** header and an **image** that are in the same section?*

Example answer

→**YES**

→ **Theorem:** For **monadic second-order** (MSO) queries, we can check if the query is true or not in **linear time** in the input tree

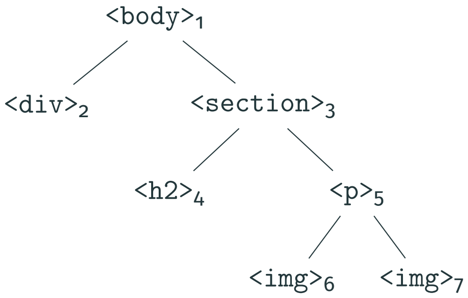
Problem: Non-Boolean query evaluation on trees

- **Tree** on a fixed alphabet
- **Non-**Boolean **query** to find tuples of nodes satisfying a property

Problem: Non-Boolean query evaluation on trees

- Tree on a fixed alphabet
- Non-Boolean query to find tuples of nodes satisfying a property

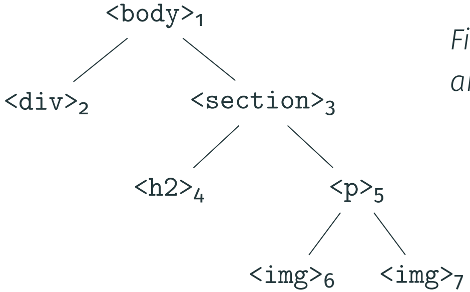
Example tree



Problem: Non-Boolean query evaluation on trees

- Tree on a fixed alphabet
- Non-Boolean query to find tuples of nodes satisfying a property

Example tree



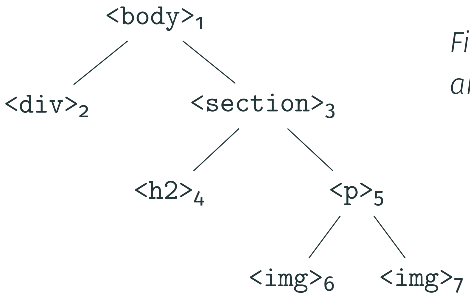
Example query

*Find all pairs of an **h2** header and an **image** in the same section*

Problem: Non-Boolean query evaluation on trees

- Tree on a fixed alphabet
- Non-Boolean query to find tuples of nodes satisfying a property

Example tree



Example query

Find all pairs of an h2 header and an image in the same section

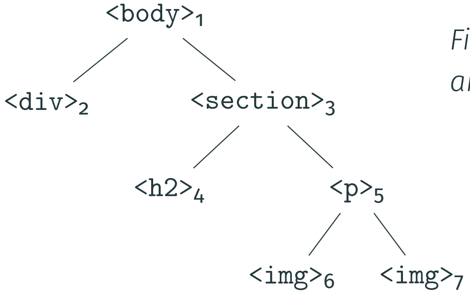
Example answer

$\rightarrow \ \{\langle 4,6\rangle,\langle 4,7\rangle\}$

Problem: Non-Boolean query evaluation on trees

- **Tree** on a fixed alphabet
- **Non-**Boolean **query** to find tuples of nodes satisfying a property

Example tree



Example query

*Find all pairs of an **h2** header and an **image** in the same section*

Example answer

$\rightarrow \{\langle 4, 6\rangle, \langle 4, 7\rangle\}$

$\rightarrow$ **Corollary:** For each possible tuple, we can check in **linear time** if it is an answer to the query

Enumerating all answers

$\rightarrow$ There can be lots of answers!

- *"Find all pairs of …"*: output size can be $O(|T|^2)$

Enumerating all answers

→ There can be lots of answers!

- *"Find all pairs of ..."*: output size can be $O(|T|^2)$

Enumerating all answers

→ There can be lots of answers!

- *"Find all pairs of ..."*: output size can be $O(|T|^2)$

Enumerating all answers

→ There can be lots of answers!

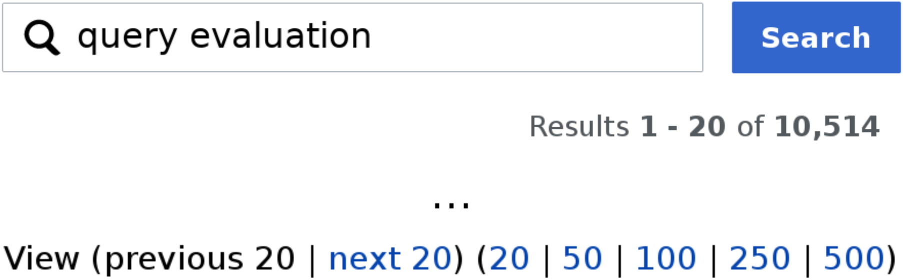
- *"Find all pairs of ..."*: output size can be $O(|T|^2)$

...

Enumerating all answers

→ There can be lots of answers!

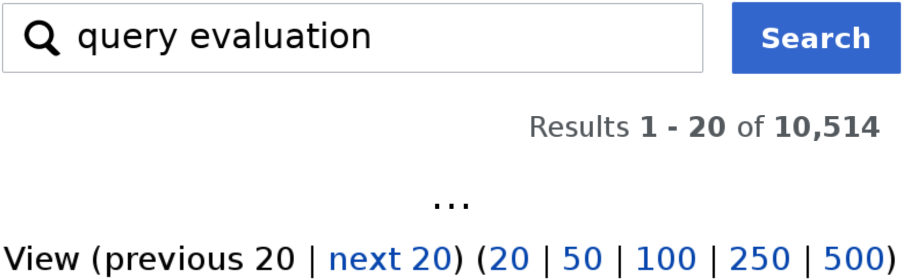
- *"Find all pairs of ..."*: output size can be $O(|T|^2)$



Enumerating all answers

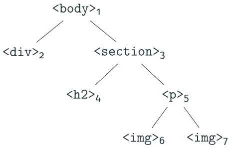
→ There can be **lots of answers**!

- *"Find all pairs of ..."*: output size can be $O(|T|^2)$



→ **Solution:** Enumerate answers **one after the other**

Enumeration

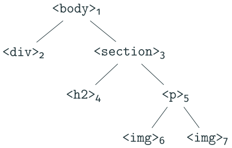


Tree

$$\exists s\ \mathtt{section}(s) \wedge$$
$$s \rightsquigarrow x \wedge s \rightsquigarrow y \wedge$$
$$\mathtt{h2}(x) \wedge \mathtt{img}(y)$$

Query

Enumeration



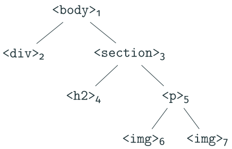
Tree

$$\exists s\ \mathtt{section}(s) \wedge$$
$$s \rightsquigarrow x \wedge s \rightsquigarrow y \wedge$$
$$\mathtt{h2}(x) \wedge \mathtt{img}(y)$$

Query

Phase 1:
Preprocessing

Enumeration



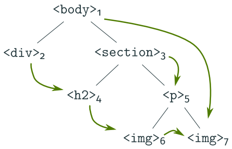
Tree

$$\exists s\ \texttt{section}(s) \land s \rightsquigarrow x \land s \rightsquigarrow y \land \texttt{h2}(x) \land \texttt{img}(y)$$

Query

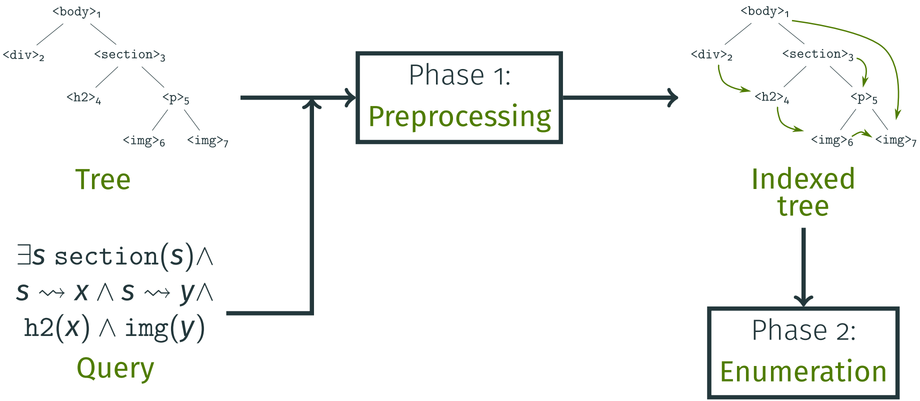
Phase 1:

Preprocessing

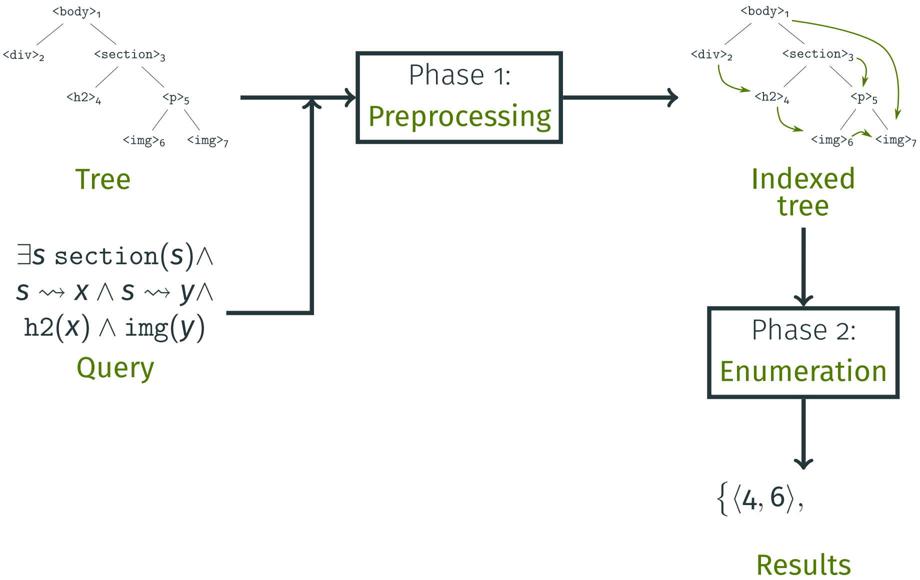


Indexed tree

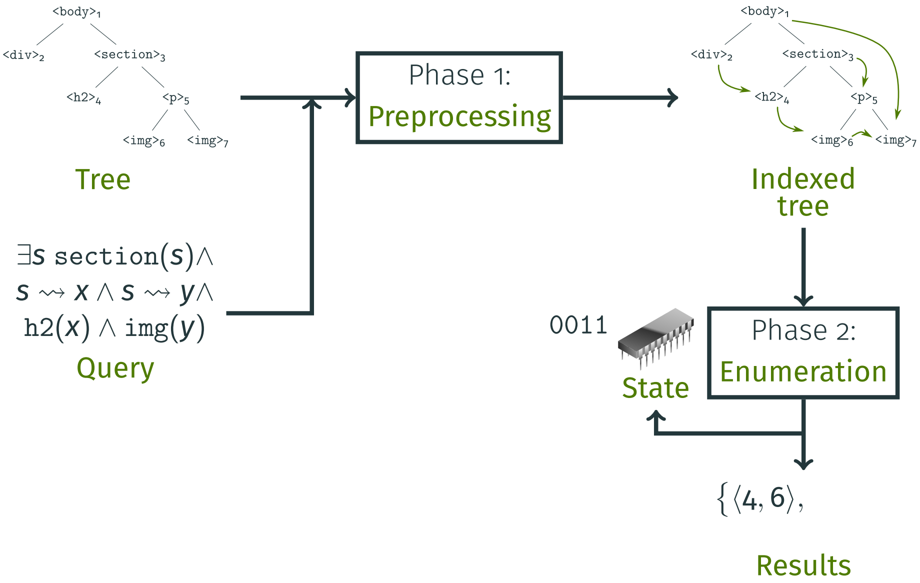
Enumeration



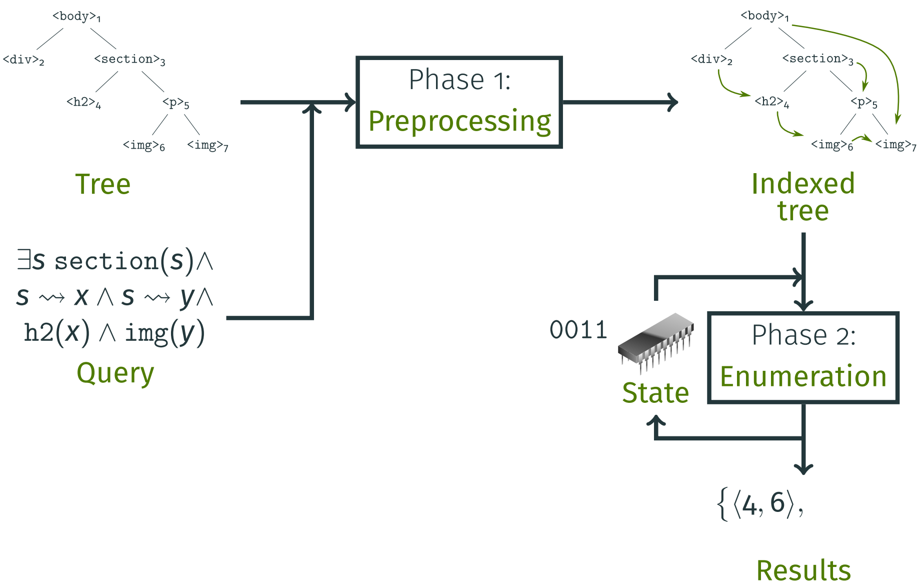
Enumeration



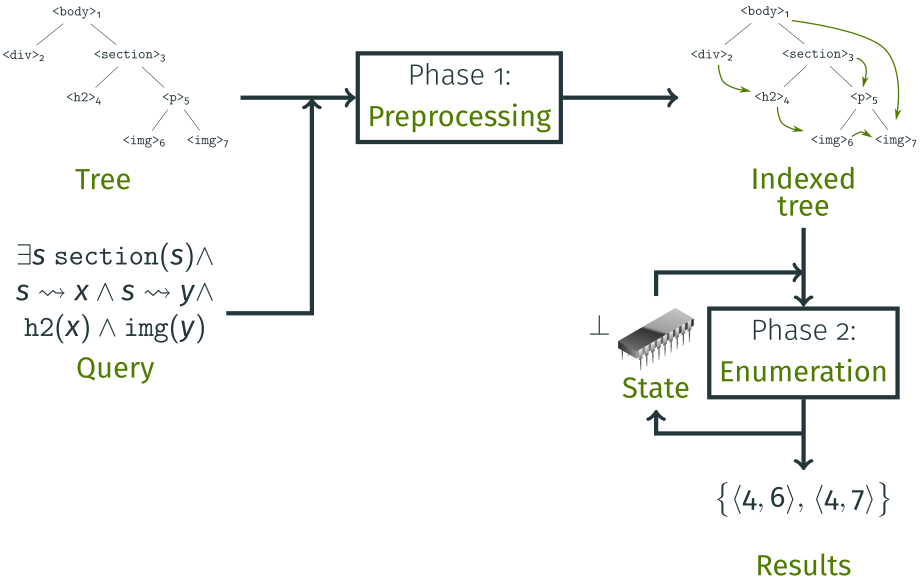
Enumeration



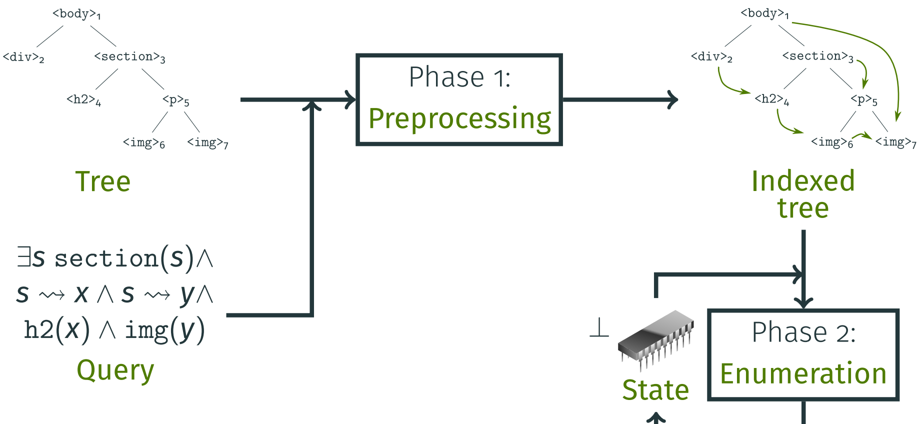
Enumeration



Enumeration

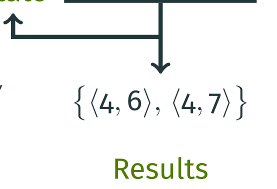


Enumeration



Theorem (Bagan'06; Kazana & Segoufin'13)

*We can enumerate the answers of any MSO query with preprocessing **linear** in the input tree and **constant** delay between each answer*



Handling updates

- Trees are often **updated**, even after we have preprocessed them

Handling updates

- Trees are often **updated**, even after we have preprocessed them

Handling updates

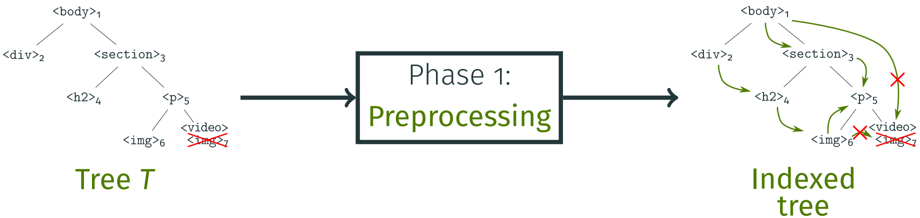
- Trees are often **updated**, even after we have preprocessed them

Handling updates

- Trees are often **updated**, even after we have preprocessed them

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

Handling updates



- Trees are often **updated**, even after we have preprocessed them

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

Handling updates

- Trees are often **updated**, even after we have preprocessed them

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

Handling updates

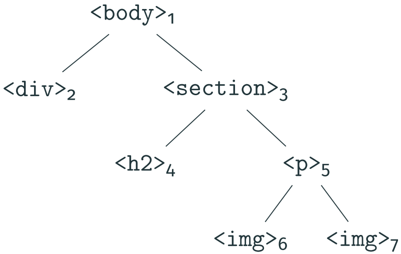
Tree T

Indexed tree

- Trees are often **updated**, even after we have preprocessed them

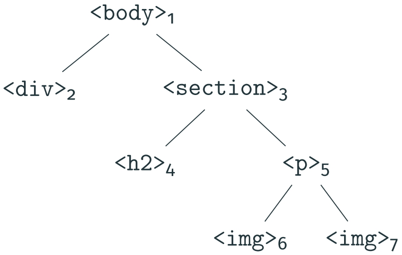
Work	**Data**	**Delay**	**Updates**
[Bagan, 2006], [Kazana and Segoufin, 2013]	trees	$O(1)$	$O(T)$ (from scratch)
[Losemann and Martens, 2014]	trees	$O(\log^2 T)$	$O(\log^2 T)$
[Losemann and Martens, 2014]	words	$O(\log T)$	$O(\log T)$
[Niewerth and Segoufin, 2018]	words	$O(1)$	$O(\log T)$

Relabelings



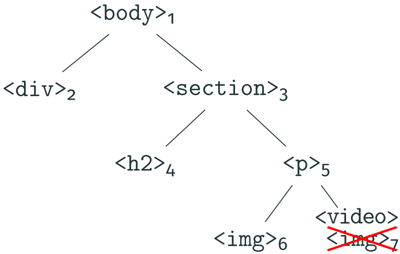
- We focus on relabeling updates: change the label of a node

Relabelings



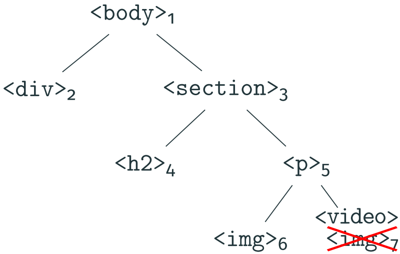
- We focus on **relabeling updates**: change the label of a node
- **Example:** relabel node 7 to `<video>`

Relabelings



- We focus on **relabeling updates**: change the label of a node
- **Example:** relabel node 7 to `<video>`

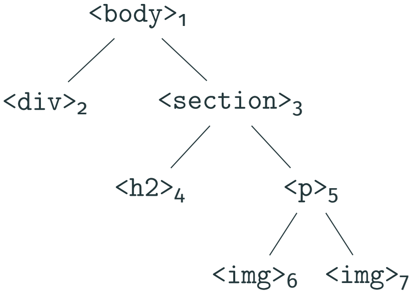
Relabelings



- We focus on **relabeling updates**: change the label of a node
- **Example:** relabel node `7` to `<video>`
- The tree's **structure** never changes

What are relabelings good for?

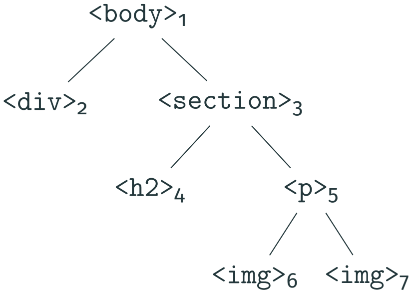
- Parameterized queries:
  - Example: *“Find all images in a* ***user-selected*** *section”*



What are relabelings good for?

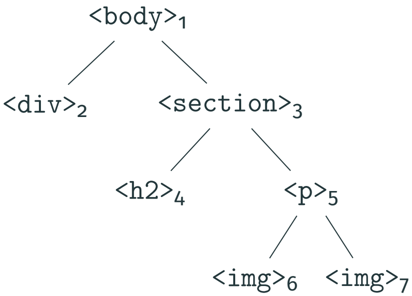
- Parameterized queries:
  - Example: *"Find all images in a* ***user-selected*** *section"*

  → Write down the user parameters as labels on the tree



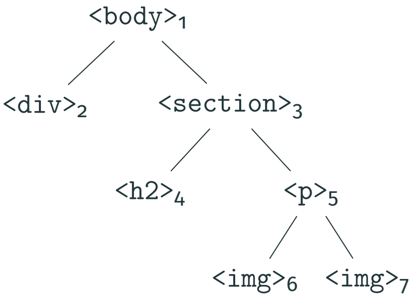
What are relabelings good for?

- Parameterized queries:
  - Example: *"Find all images in a* ***user-selected*** *section"*
  - → **Write down** the user parameters as labels on the tree
  - → **Relabel** when they change



What are relabelings good for?

- Parameterized queries:
  - Example: *“Find all images in a **user-selected** section”*
  - → Write down the user parameters as labels on the tree
  - → Relabel when they change

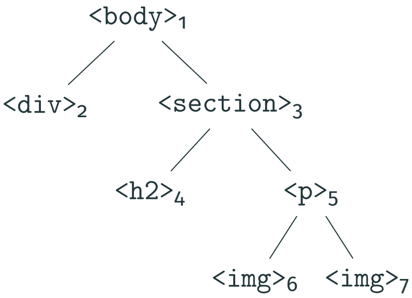


- Group-by with aggregation:
  - Example: *“For each section, what is the total size of images”*

What are relabelings good for?

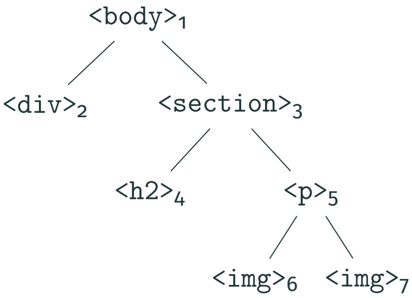
- Parameterized queries:
  - Example: *“Find all images in a **user-selected** section”*
  - → **Write down** the user parameters as labels on the tree
  - → **Relabel** when they change

- Group-by with aggregation:
  - Example: *“For each section, what is the total size of images”*
  - → **Enumerate** the groups and **write down** each group



What are relabelings good for?

- Parameterized queries:
  - Example: *“Find all images in a* ***user-selected*** *section”*
  - → **Write down** the user parameters as labels on the tree
  - → **Relabel** when they change



- Group-by with aggregation:
  - Example: *“For each section, what is the total size of images”*
  - → **Enumerate** the groups and **write down** each group
  - → **Relabel** when switching groups

Main result

Theorem (Main result)

*We can enumerate the answers of any MSO query with preprocessing **linear** in the input tree T and **constant** delay between each answer **and we can relabel any node and update the index in time $O(\log T)$***

Main result

Theorem (Main result)

We can enumerate the answers of any MSO query with preprocessing ***linear*** *in the input tree T and* ***constant*** *delay between each answer* ***and we can relabel any node and update the index in time $O(\log T)$***

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

Main result

Theorem (Main result)

We can enumerate the answers of any MSO query with preprocessing ***linear*** *in the input tree T and* ***constant*** *delay between each answer* ***and we can relabel any node and update the index in time $O(\log T)$***

Work	Data	Delay	Updates
[Bagan, 2006], [Kazana and Segoufin, 2013]	trees	$O(1)$	N/A
[Losemann and Martens, 2014]	trees	$O(\log^2 T)$	$O(\log^2 T)$
[Losemann and Martens, 2014]	words	$O(\log T)$	$O(\log T)$
[Niewerth and Segoufin, 2018]	words	$O(1)$	$O(\log T)$
[This work]	trees	$O(1)$	$O(\log T)$ (relabelings)

Main result

Theorem (Main result)

*We can enumerate the answers of any MSO query with preprocessing **linear** in the input tree T and **constant** delay between each answer **and we can relabel any node and update the index in time $O(\log T)$***

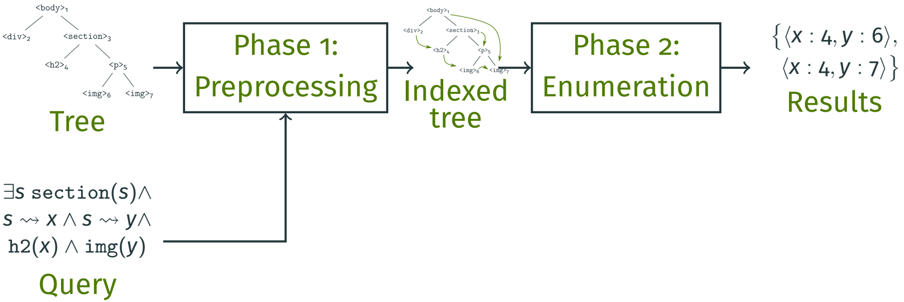
Work	Data	Delay	Updates
[Bagan, 2006], [Kazana and Segoufin, 2013]	trees	$O(1)$	N/A
[Losemann and Martens, 2014]	trees	$O(\log^2 T)$	$O(\log^2 T)$
[Losemann and Martens, 2014]	words	$O(\log T)$	$O(\log T)$
[Niewerth and Segoufin, 2018]	words	$O(1)$	$O(\log T)$
[This work]	trees	$O(1)$	$O(\log T)$ (relabelings)

$\rightarrow$ Consequences for group-by, aggregation, parameterized queries

Proof techniques

Knowledge compilation

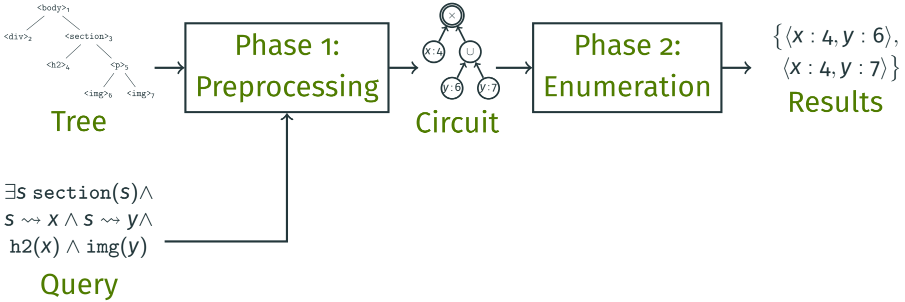
To make the proof modular, we follow knowledge compilation:



Knowledge compilation

To make the proof **modular**, we follow **knowledge compilation**:

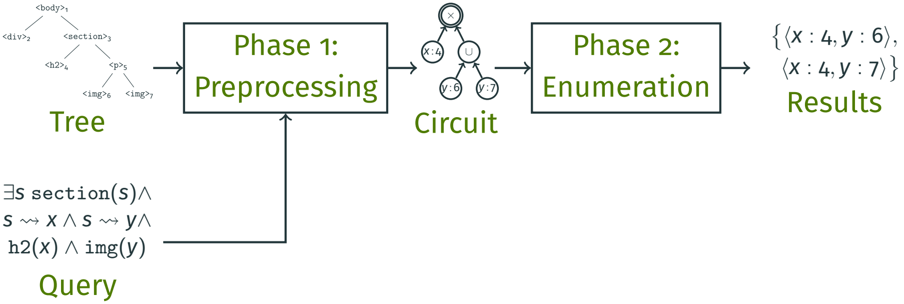
- **Preprocessing:** Compute a **circuit representation** of the answers
- **Enumeration:** Apply a **generic algorithm** on the circuit



Knowledge compilation

To make the proof **modular**, we follow **knowledge compilation**:

- **Preprocessing:** Compute a **circuit representation** of the answers
- **Enumeration:** Apply a **generic algorithm** on the circuit



First present approach **without relabelings** (as in our ICALP'17 paper) then extend the approach to **support relabelings**

Set circuits

A set circuit represents a set of answers to a query $Q(x, y)$

Set circuits

A set circuit represents a set of answers to a query $Q(x, y)$

- Singleton $x\!:\!6 \rightarrow$ *"the free variable* x *is mapped to node* 6*"*

Set circuits

A set circuit represents a set of answers to a query $Q(x, y)$

- Singleton $x : 6 \rightarrow$ *"the free variable x is mapped to node 6"*
- Tuple $\langle x : 4, y : 6 \rangle$: tuple of singletons

Set circuits

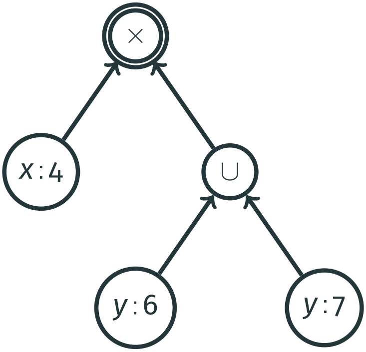
A set circuit represents a set of answers to a query $Q(x, y)$

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

Set circuits

A set circuit represents a set of answers to a query $Q(x, y)$

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

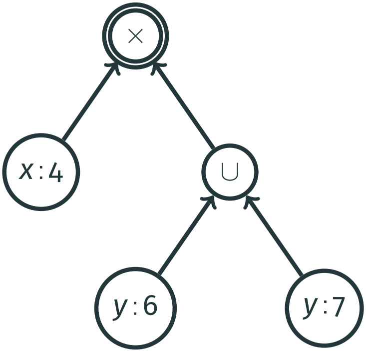


Three kinds of set-valued gates:

Set circuits

A set circuit represents a set of answers to a query $Q(x, y)$

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



Three kinds of set-valued gates:

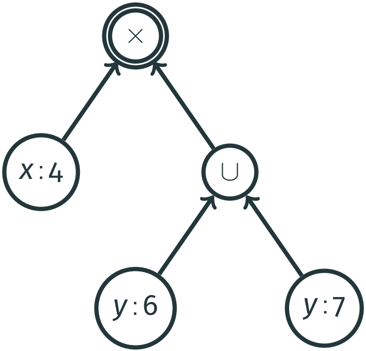
- Variable gate $x : 4$:

  $\rightarrow$ captures $\{\langle x : 4 \rangle\}$

Set circuits

A set circuit represents a set of answers to a query $Q(x, y)$

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



Three kinds of set-valued gates:

- Variable gate $x:4$:

  $\rightarrow$ captures $\{\langle x:4\rangle\}$

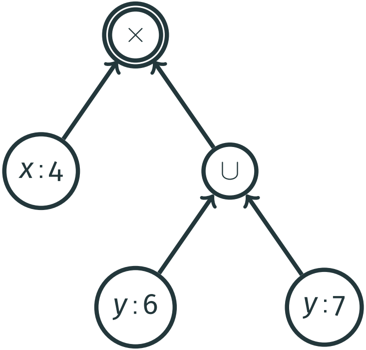
- Union gate $\cup$:

  $\rightarrow$ union of sets of tuples

Set circuits

A **set circuit** represents a **set of answers** to a query $Q(x, y)$

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



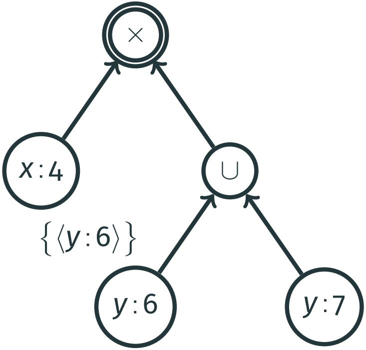
Three kinds of **set-valued gates**:

- **Variable gate** $x:4$:
  $\to$ captures $\{\langle x:4\rangle\}$
- **Union gate** $\cup$:
  $\to$ union of sets of tuples
- **Product gate** $\times$:
  $\to$ relational product

Set circuits

A **set circuit** represents a **set of answers** to a query $Q(x, y)$

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



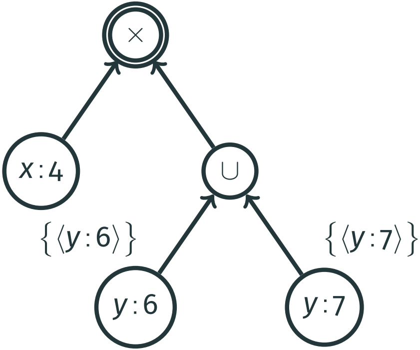
Three kinds of **set-valued gates**:

- **Variable gate** $x:4$:
  - $\rightarrow$ captures $\{\langle x:4\rangle\}$
- **Union gate** $\cup$:
  - $\rightarrow$ union of sets of tuples
- **Product gate** $\times$:
  - $\rightarrow$ relational product

Set circuits

A set circuit represents a set of answers to a query $Q(x, y)$

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



Three kinds of set-valued gates:

- Variable gate $x:4$:

  $\rightarrow$ captures $\{\langle x:4 \rangle\}$

- Union gate $\cup$:

  $\rightarrow$ union of sets of tuples

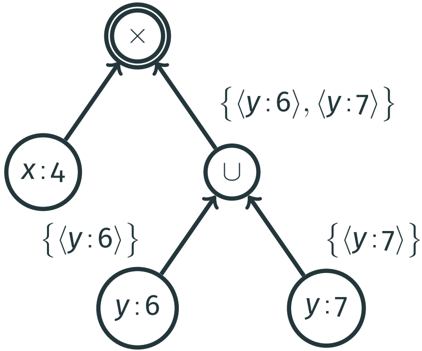
- Product gate $\times$:

  $\rightarrow$ relational product

Set circuits

A **set circuit** represents a **set of answers** to a query $Q(x, y)$

- **Singleton** $x:6$ $\rightarrow$ *“the free variable x is mapped to node 6”*
- **Tuple** $\langle x:4, y:6\rangle$: tuple of singletons
- The circuit captures a **set** of tuples, e.g., $\{\langle x:4, y:6\rangle, \langle x:4, y:7\rangle\}$



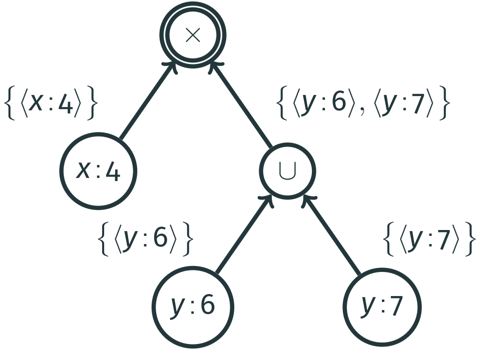
Three kinds of **set-valued gates**:

- **Variable gate** $x:4$:
  $\rightarrow$ captures $\{\langle x:4\rangle\}$
- **Union gate** $\cup$:
  $\rightarrow$ union of sets of tuples
- **Product gate** $\times$:
  $\rightarrow$ relational product

Set circuits

A set circuit represents a set of answers to a query $Q(x, y)$

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



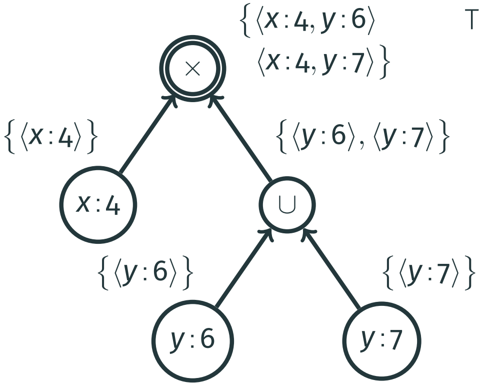
Three kinds of set-valued gates:

- Variable gate $x\!:\!4$:
  $\rightarrow$ captures $\{\langle x\!:\!4 \rangle\}$
- Union gate $\cup$:
  $\rightarrow$ union of sets of tuples
- Product gate $\times$:
  $\rightarrow$ relational product

Set circuits

A set circuit represents a set of answers to a query $Q(x, y)$

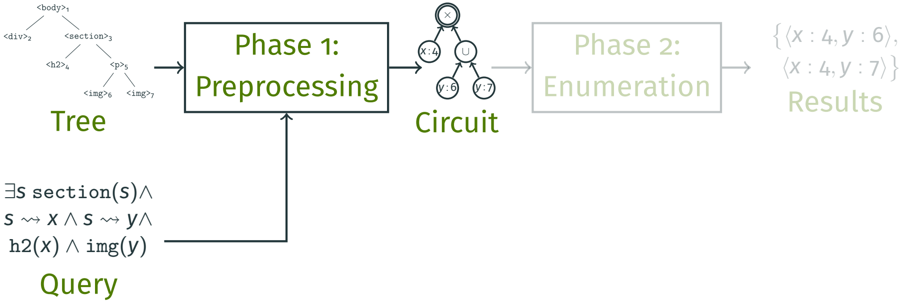
- Singleton $x{:}6$ → *"the free variable* x *is mapped to node* 6*"*
- Tuple $\langle x{:}4, y{:}6\rangle$: tuple of singletons
- The circuit captures a set of tuples, e.g., $\{\langle x{:}4, y{:}6\rangle, \langle x{:}4, y{:}7\rangle\}$



Three kinds of set-valued gates:

- Variable gate $x{:}4$:
  → captures $\{\langle x{:}4\rangle\}$
- Union gate $\cup$:
  → union of sets of tuples
- Product gate $\times$:
  → relational product

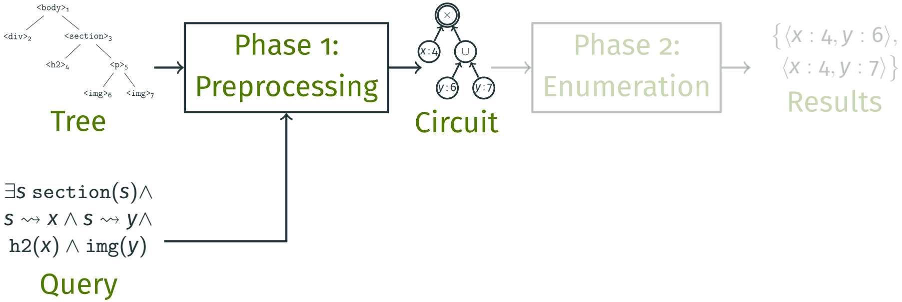
Preprocessing: Set circuit construction



Theorem

For any MSO query $Q(x_1, \ldots, x_k)$, given a tree T, we can build in $O(T)$ a set circuit capturing exactly the set of answers of Q on T:
$\{\langle x_1 : n_1, \ldots, x_k : n_k\rangle \mid (n_1, \ldots, n_k) \in T^k\}$

Preprocessing: Set circuit construction

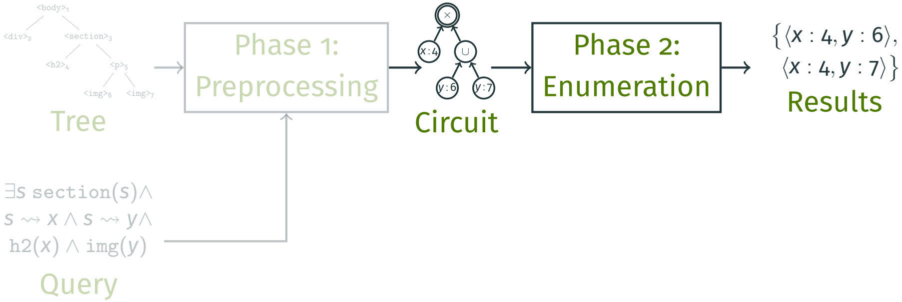


Theorem

For any **MSO query** $Q(x_1, \ldots, x_k)$, *given a* **tree** T, *we can build in* $O(T)$ *a* **set circuit** *capturing exactly the set of* **answers** *of* Q *on* T:
$\{\langle x_1 : n_1, \ldots, x_k : n_k \rangle \mid (n_1, \ldots, n_k) \in T^k\}$

- **Proof idea:** Translate query to **bottom-up tree automaton** and build a **provenance circuit** following the structure of the tree

Enumeration on set circuits



Theorem

Given a set circuit satisfying some conditions, we can enumerate all tuples that it captures with linear preprocessing and constant delay

E.g., for $\{\langle x:4, y:6\rangle, \langle x:4, y:7\rangle\}$: enumerate $\langle x:4, y:6\rangle$, then $\langle x:4, y:7\rangle$

General enumeration approach

- $\rightarrow$ Enumerate the set $T(g)$ captured by each gate g
- $\rightarrow$ Do it by top-down induction on the circuit

General enumeration approach

- $\rightarrow$ Enumerate the set $T(g)$ captured by each gate g
- $\rightarrow$ Do it by **top-down induction** on the circuit

Base case: variable $x : n$:

General enumeration approach

- $\rightarrow$ Enumerate the set $T(g)$ captured by each gate g
- $\rightarrow$ Do it by top-down induction on the circuit

Base case: variable $x : n$: enumerate $\langle x : n \rangle$ and stop

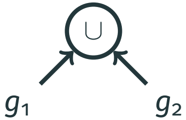
General enumeration approach

→ Enumerate the set $T(g)$ captured by each gate g

→ Do it by **top-down induction** on the circuit

Base case: variable $x : n$: enumerate $\langle x : n \rangle$ and stop

∪-gate



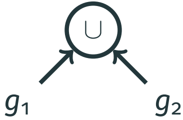
Concatenation: enumerate $T(g_1)$
and then enumerate $T(g_2)$

General enumeration approach

- $\rightarrow$ Enumerate the set $T(g)$ captured by each gate g
- $\rightarrow$ Do it by **top-down induction** on the circuit

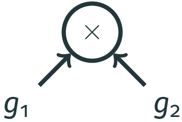
Base case: variable $x : n$: enumerate $\langle x : n \rangle$ and stop

$\cup$-gate



Concatenation: enumerate $T(g_1)$ and then enumerate $T(g_2)$

$\times$-gate

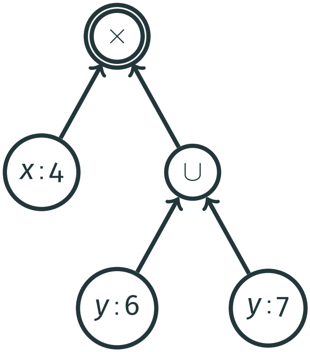


Lexicographic product:

for every t_1 in $T(g_1)$:
    for every t_2 in $T(g_2)$:
        output $t_1 + t_2$

Circuit conditions

Enumeration relies on some **conditions** on the input circuit (d-DNNF):



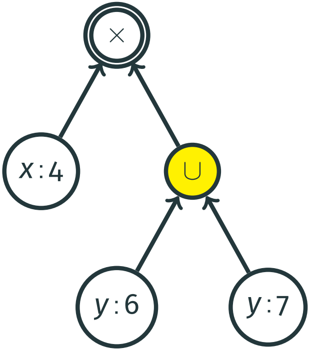
Circuit conditions

Enumeration relies on some **conditions** on the input circuit (d-DNNF):

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

For any two inputs g_1 and g_2 of a $\cup$-gate, the captured sets $T(g_1)$ and $T(g_2)$ are **disjoint** (they have no tuple in common)

$\rightarrow$ Avoids **duplicate tuples**



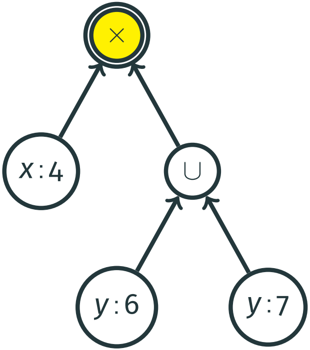
Circuit conditions

Enumeration relies on some conditions on the input circuit (d-DNNF):

- $\cup$ are all deterministic:

For any two inputs g_1 and g_2 of a $\cup$-gate, the captured sets $T(g_1)$ and $T(g_2)$ are disjoint (they have no tuple in common)

$\rightarrow$ Avoids duplicate tuples



- $\times$ are all decomposable:

For any two inputs g_1 and g_2 of a $\times$-gate, no variable has a path to both g_1 and g_2

$\rightarrow$ Avoids duplicate singletons

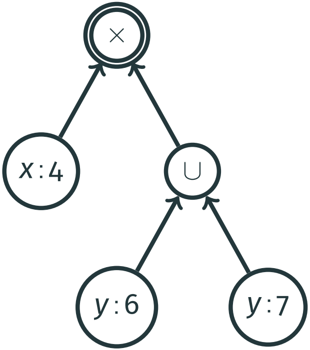
Circuit conditions

Enumeration relies on some **conditions** on the input circuit (d-DNNF):

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

For any two inputs g_1 and g_2 of a $\cup$-gate, the captured sets $T(g_1)$ and $T(g_2)$ are **disjoint** (they have no tuple in common)

$\rightarrow$ Avoids **duplicate tuples**



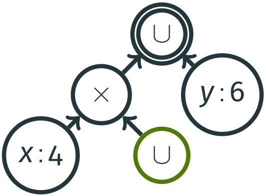
- $\times$ are all **decomposable**:

For any two inputs g_1 and g_2 of a $\times$-gate, no variable has a path to both g_1 and g_2

$\rightarrow$ Avoids **duplicate singletons**

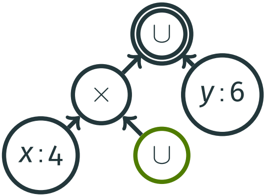
- Also an additional **upwards-determinism** condition

Enumeration subtleties



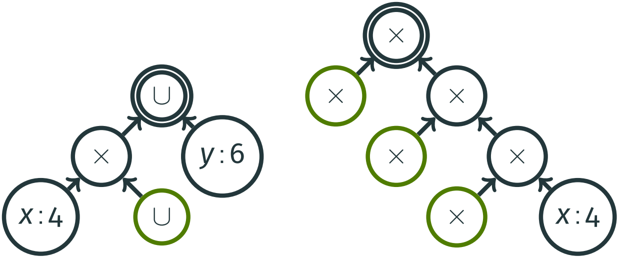
- We must not waste time in gates capturing $\emptyset$

Enumeration subtleties



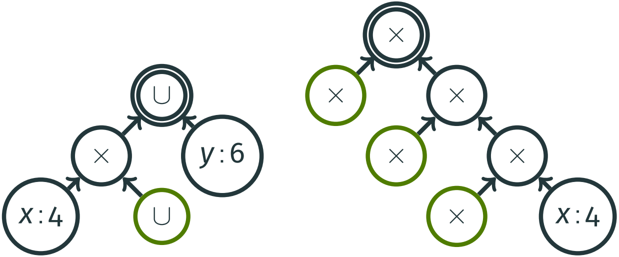
- We must not waste time in gates capturing $\emptyset$
  - $\rightarrow$ **Label** them during the preprocessing

Enumeration subtleties



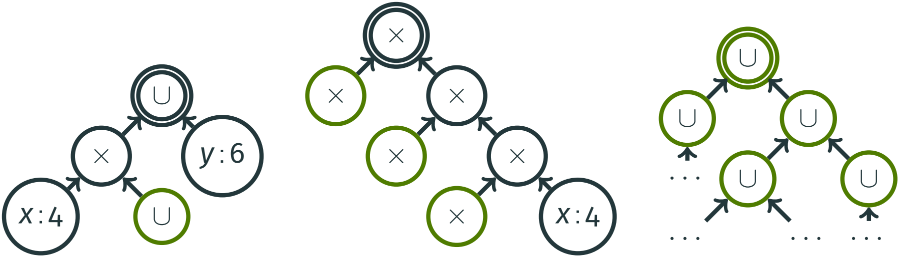
- We must not waste time in gates capturing $\emptyset$
  - $\rightarrow$ **Label** them during the preprocessing
- We must not waste time because of gates capturing $\{\langle\rangle\}$

Enumeration subtleties



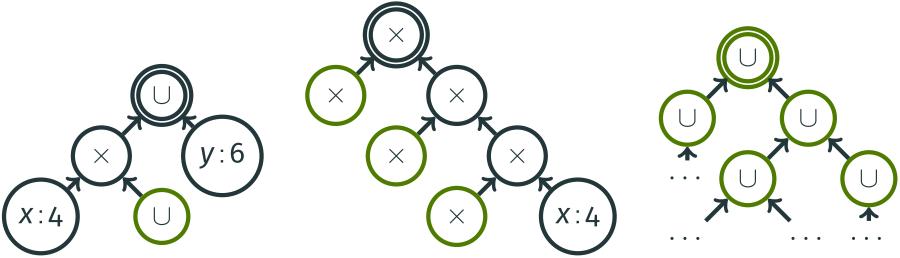
- We must not waste time in gates capturing $\emptyset$
  $\rightarrow$ **Label** them during the preprocessing
- We must not waste time because of gates capturing $\{\langle\rangle\}$
  $\rightarrow$ **Homogenization** to set them aside

Enumeration subtleties



- We must not waste time in gates capturing $\emptyset$
  - $\rightarrow$ **Label** them during the preprocessing
- We must not waste time because of gates capturing $\{\langle\rangle\}$
  - $\rightarrow$ **Homogenization** to set them aside
- We must not waste time in **hierarchies of $\cup$-gates**

Enumeration subtleties

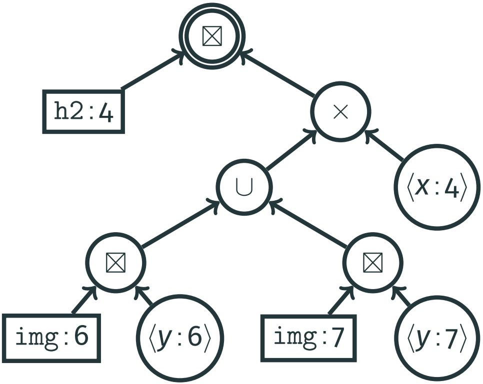


- We must not waste time in gates capturing $\emptyset$
  - $\rightarrow$ **Label** them during the preprocessing
- We must not waste time because of gates capturing $\{\langle\rangle\}$
  - $\rightarrow$ **Homogenization** to set them aside
- We must not waste time in **hierarchies of $\cup$-gates**
  - $\rightarrow$ Precompute a **reachability index** (uses **upwards-determinism**)

Hybrid circuits to support relabelings

To support relabelings, we use hybrid circuits that have:

- Boolean gates that depend only on the labeling
- Set gates that capture a set of tuples for each labeling

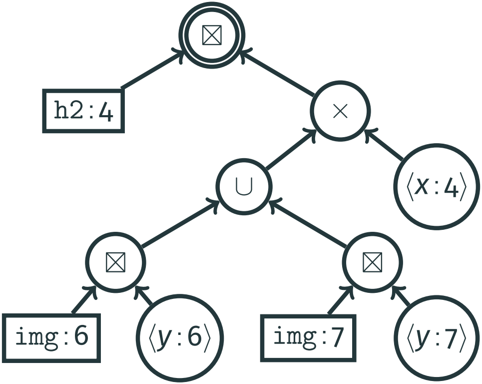


Hybrid circuits to support relabelings

To support relabelings, we use hybrid circuits that have:

-  Boolean gates that depend only on the labeling
-  Set gates that capture a set of tuples for each labeling

Four kinds of Boolean gates:



Hybrid circuits to support relabelings

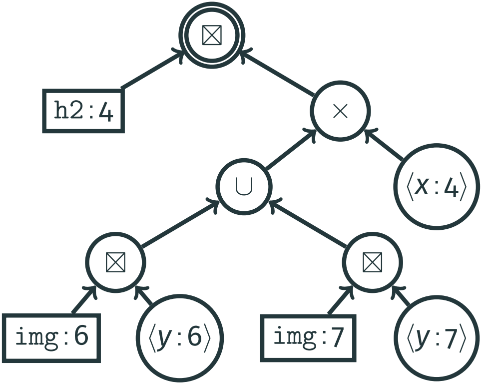
To support **relabelings**, we use **hybrid circuits** that have:

- **Boolean gates** that depend only on the **labeling**
- **Set gates** that capture a set of tuples for **each labeling**

Four kinds of **Boolean gates**:

- **Variable gate** $\boxed{\texttt{h2}:4}$:

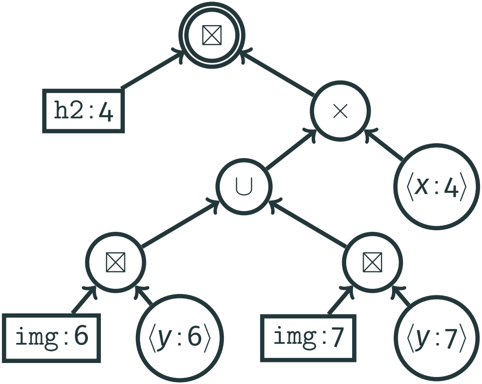
  $\rightarrow$ true iff node 4 is labeled $\texttt{h2}$



Hybrid circuits to support relabelings

To support **relabelings**, we use **hybrid circuits** that have:

- □ **Boolean gates** that depend only on the **labeling**
- ○ **Set gates** that capture a set of tuples for **each labeling**



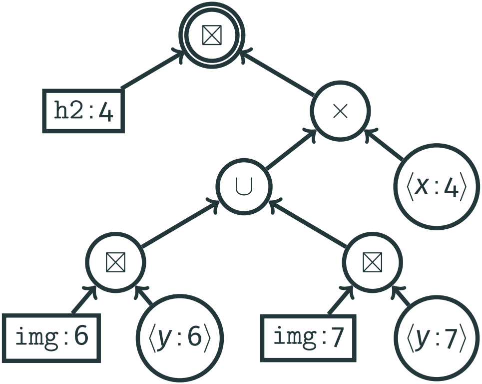
Four kinds of **Boolean gates**:

- **Variable gate** $\boxed{\texttt{h2}:4}$:
  - $\rightarrow$ true iff node 4 is labeled **h2**
- **AND, OR, NOT** $\boxed{\land}$ $\boxed{\lor}$ $\boxed{\neg}$:
  - $\rightarrow$ usual semantics

Hybrid circuits to support relabelings

To support **relabelings**, we use **hybrid circuits** that have:

- □ **Boolean gates** that depend only on the **labeling**
- ○ **Set gates** that capture a set of tuples for **each labeling**



Four kinds of **Boolean gates**:

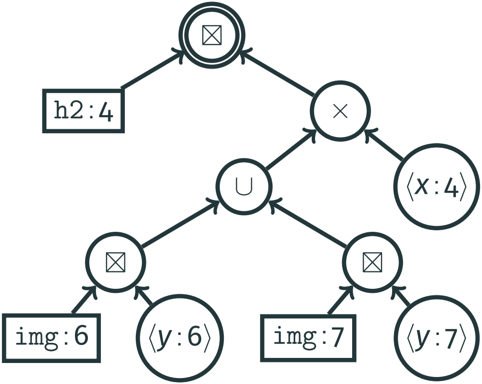
- **Variable gate** $\boxed{\texttt{h2:4}}$:
  → true iff node 4 is labeled `h2`
- **AND, OR, NOT** $\boxed{\wedge}$ $\boxed{\vee}$ $\boxed{\neg}$:
  → usual semantics

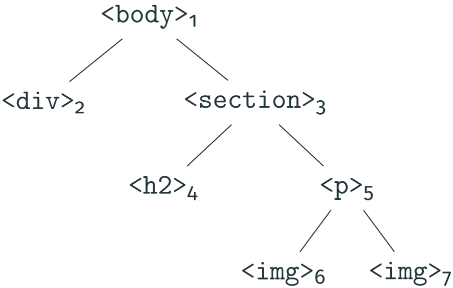
One new **set-valued gate**:

- $\boxtimes$: **Test gate**
  → $\emptyset$ if Boolean input is false
  → like the set input otherwise

Hybrid circuit semantics

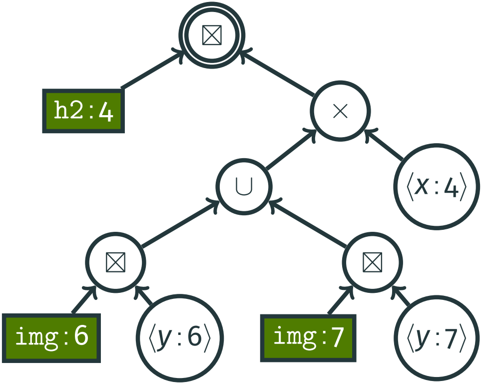
- For every **labeling**, the hybrid circuit captures a **set of tuples**

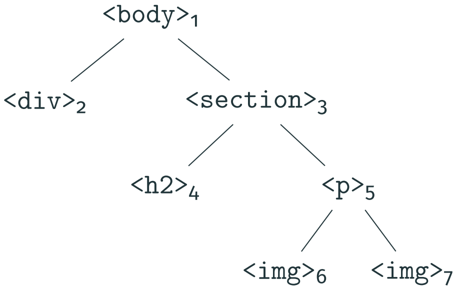




Hybrid circuit semantics

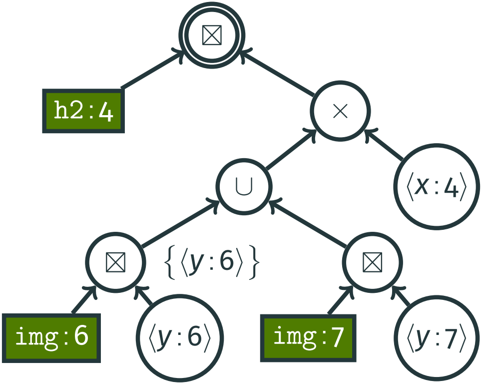
- For every **labeling**, the hybrid circuit captures a **set of tuples**
  - $\rightarrow$ Here, the captured set is $\{\langle x\!:\!4, y\!:\!6\rangle, \langle x\!:\!4, y\!:\!7\rangle\}$

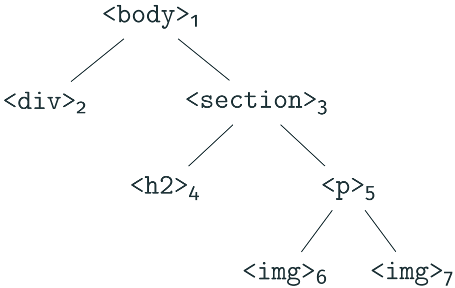




Hybrid circuit semantics

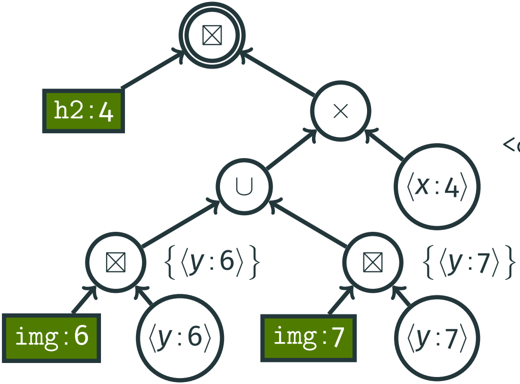
- For every labeling, the hybrid circuit captures a set of tuples
  - $\rightarrow$ Here, the captured set is $\{\langle x:4, y:6\rangle, \langle x:4, y:7\rangle\}$

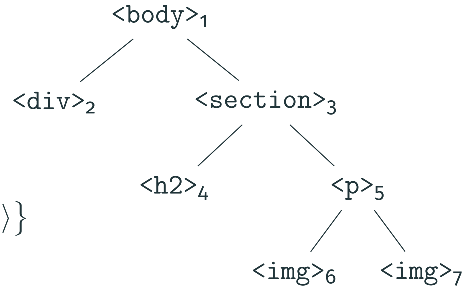




Hybrid circuit semantics

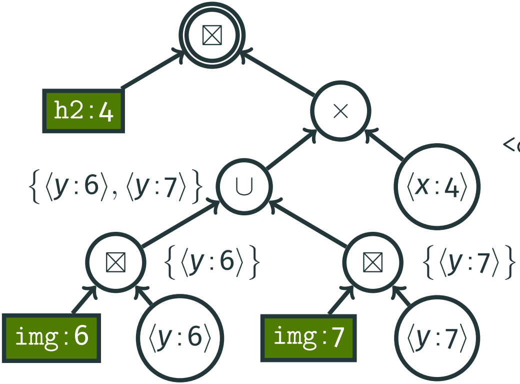
- For every labeling, the hybrid circuit captures a set of tuples
  → Here, the captured set is $\{\langle x{:}4, y{:}6\rangle, \langle x{:}4, y{:}7\rangle\}$

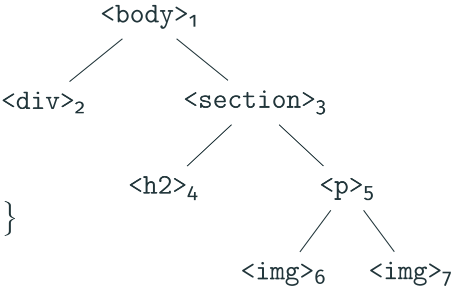




Hybrid circuit semantics

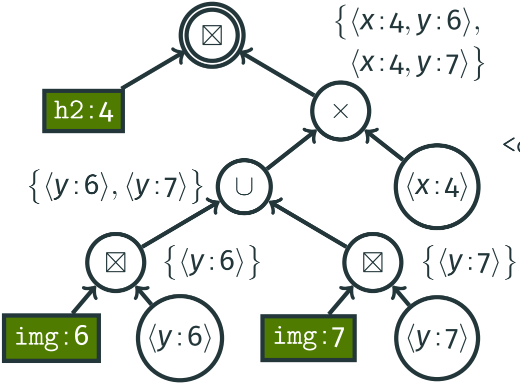
- For every **labeling**, the hybrid circuit captures a **set of tuples**
  - $\rightarrow$ Here, the captured set is $\{\langle x\!:\!4, y\!:\!6\rangle, \langle x\!:\!4, y\!:\!7\rangle\}$

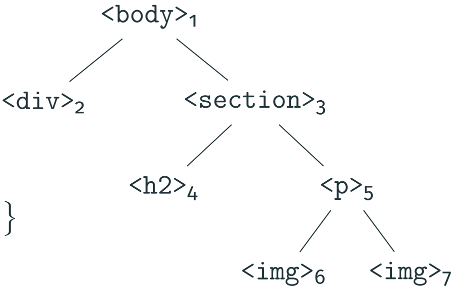




Hybrid circuit semantics

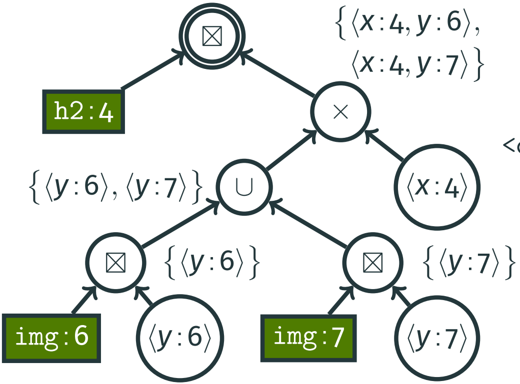
- For every **labeling**, the hybrid circuit captures a **set of tuples**
  $\rightarrow$ Here, the captured set is $\{\langle x{:}4, y{:}6\rangle, \langle x{:}4, y{:}7\rangle\}$

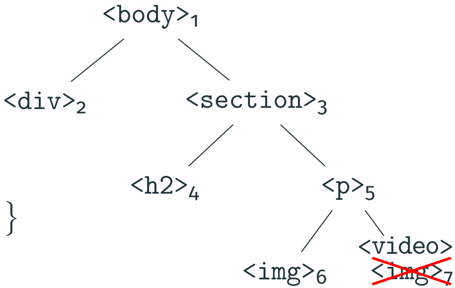




Hybrid circuit semantics

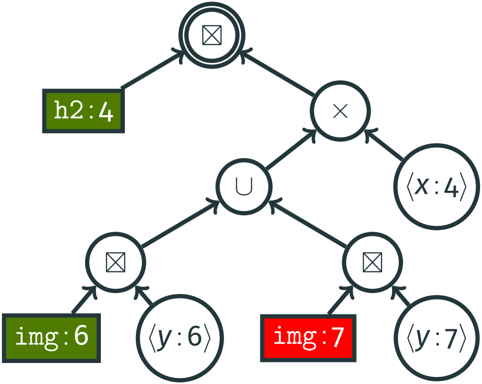
- For every **labeling**, the hybrid circuit captures a **set of tuples**
  - $\rightarrow$ Here, the captured set is $\{\langle x:4, y:6\rangle, \langle x:4, y:7\rangle\}$
- When the tree is **relabeled**, change the **Boolean variables**

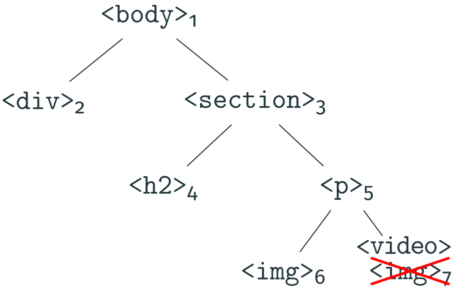




Hybrid circuit semantics

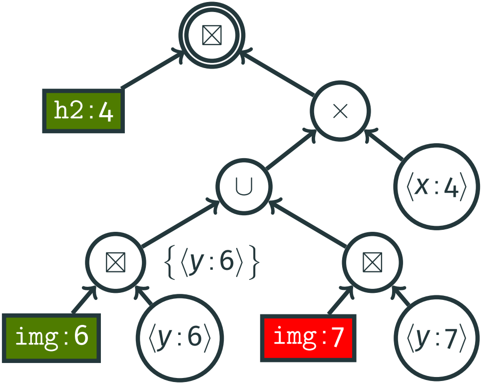
- For every **labeling**, the hybrid circuit captures a **set of tuples**
  - $\rightarrow$ Here, the captured set is $\{\langle x\!:\!4, y\!:\!6\rangle, \langle x\!:\!4, y\!:\!7\rangle\}$
- When the tree is **relabeled**, change the **Boolean variables**

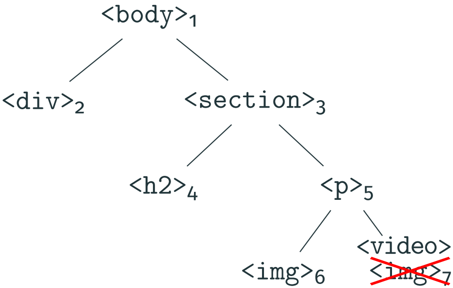




Hybrid circuit semantics

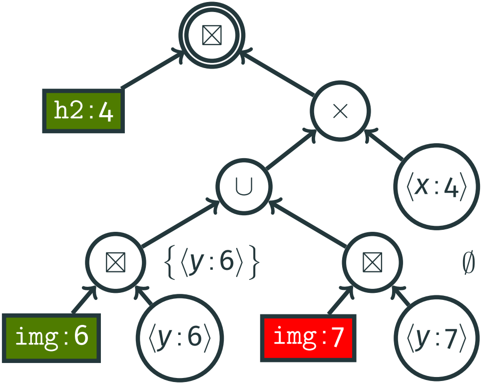
- For every labeling, the hybrid circuit captures a set of tuples
  - $\rightarrow$ Here, the captured set is $\{\langle x{:}4, y{:}6\rangle, \langle x{:}4, y{:}7\rangle\}$
- When the tree is relabeled, change the Boolean variables

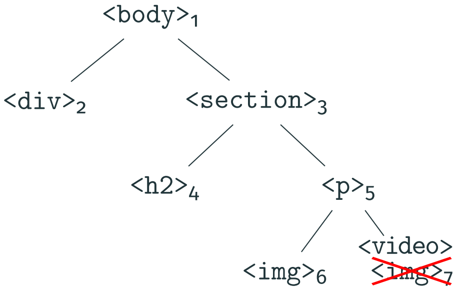




Hybrid circuit semantics

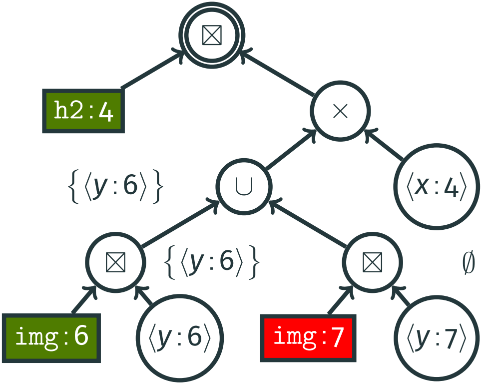
- For every **labeling**, the hybrid circuit captures a **set of tuples**
  - $\rightarrow$ Here, the captured set is $\{\langle x\!:\!4, y\!:\!6\rangle, \langle x\!:\!4, y\!:\!7\rangle\}$
- When the tree is **relabeled**, change the **Boolean variables**

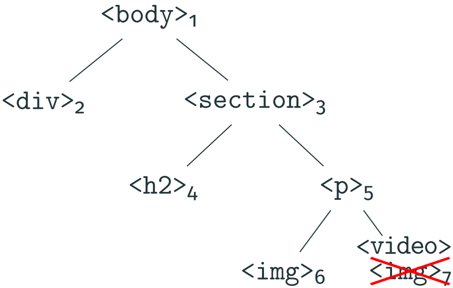




Hybrid circuit semantics

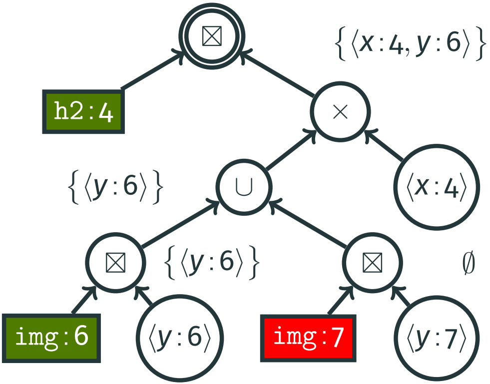
- For every **labeling**, the hybrid circuit captures a **set of tuples**
  - $\rightarrow$ Here, the captured set is $\{\langle x\!:\!4, y\!:\!6\rangle, \langle x\!:\!4, y\!:\!7\rangle\}$
- When the tree is **relabeled**, change the **Boolean variables**

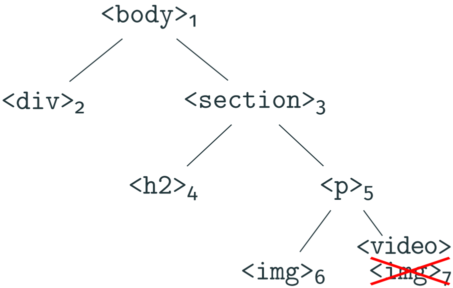




Hybrid circuit semantics

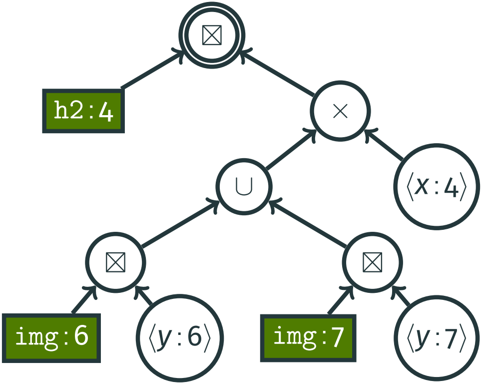
- For every **labeling**, the hybrid circuit captures a **set of tuples**
  - → Here, the captured set is $\{\langle x\!:\!4, y\!:\!6\rangle, \langle x\!:\!4, y\!:\!7\rangle\}$
- When the tree is **relabeled**, change the **Boolean variables**
  - → New captured set: $\{\langle x\!:\!4, y\!:\!6\rangle\}$

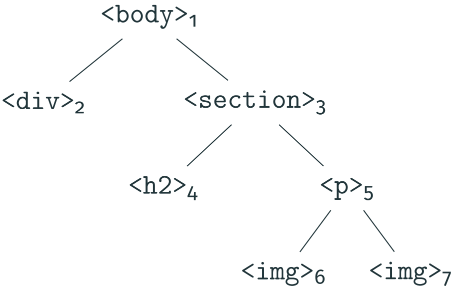




Complexity of relabelings

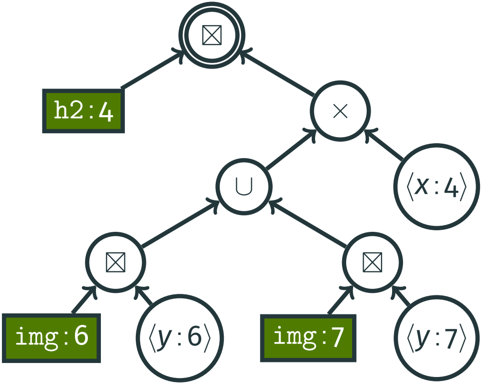
- When a label **changes**, update the circuit **bottom-up**

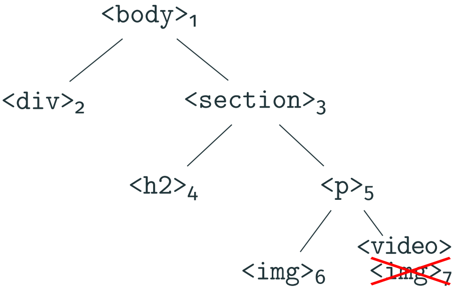




Complexity of relabelings

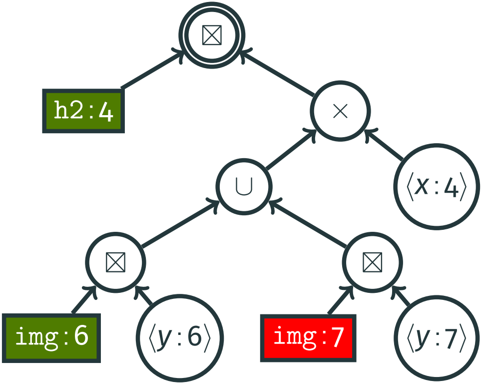
- When a label **changes**, update the circuit **bottom-up**

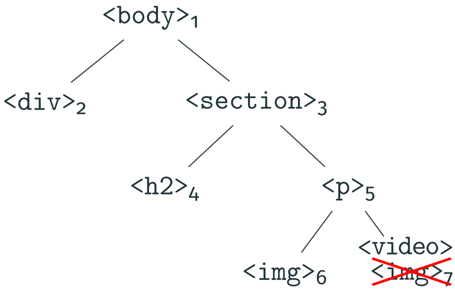




Complexity of relabelings

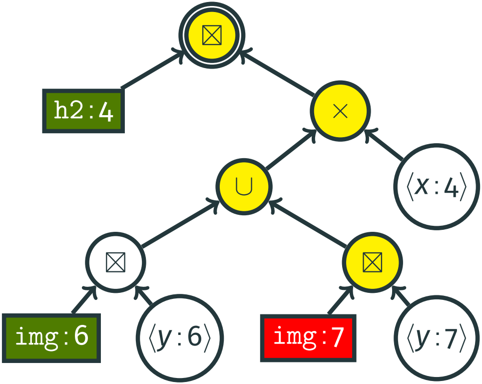
- When a label **changes**, update the circuit **bottom-up**

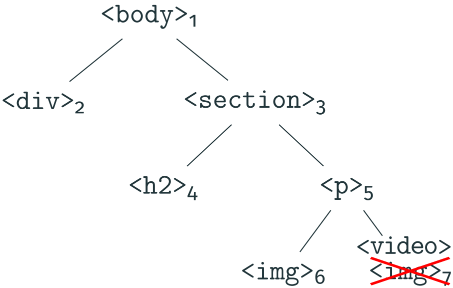




Complexity of relabelings

- When a label **changes**, update the circuit **bottom-up**
- The circuit follows the structure of the **input tree** T so updates are in $O(\text{height}(T))$

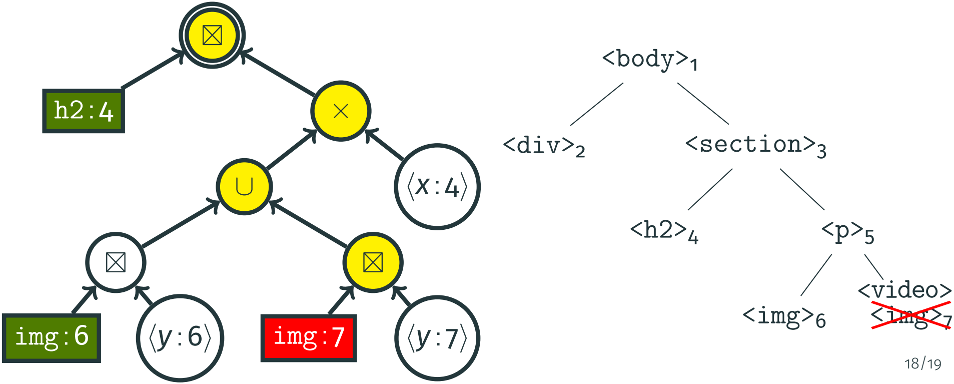




Complexity of relabelings

- When a label **changes**, update the circuit **bottom-up**
- The circuit follows the structure of the **input tree** T so updates are in $O(\text{height}(T))$

$\rightarrow$ **Balancing lemma:** Rewrite the **input tree** to make it **balanced**



Conclusion

Summary and future work

Summary:

- Problem: enumerate the answers of an MSO query on a tree with efficient support for relabeling updates on the tree

Summary and future work

Summary:

- Problem: enumerate the answers of an MSO query on a tree with efficient support for relabeling updates on the tree
- Main result: we can do this with linear preprocessing, constant delay between each answer, and log update time

Summary and future work

Summary:

- Problem: enumerate the answers of an MSO query on a tree with efficient support for relabeling updates on the tree
- Main result: we can do this with linear preprocessing, constant delay between each answer, and log update time
- Consequences: group-by, parameterized queries, aggregation

Summary and future work

Summary:

- Problem: enumerate the answers of an MSO query on a tree with efficient support for relabeling updates on the tree
- Main result: we can do this with linear preprocessing, constant delay between each answer, and log update time
- Consequences: group-by, parameterized queries, aggregation

Future work:

- Practice: implement the technique with automata
- Applications: text extraction? e.g., document spanners (ongoing)
- Updates: support insertions/deletions? (ongoing)

Summary and future work

Summary:

- **Problem:** enumerate the answers of an **MSO query** on a **tree** with efficient support for **relabeling updates** on the tree
- **Main result:** we can do this with **linear** preprocessing, **constant** delay between each answer, and **log** update time
- **Consequences:** group-by, parameterized queries, aggregation

Future work:

- **Practice:** implement the technique with **automata**
- **Applications:** text extraction? e.g., **document spanners** (ongoing)
- **Updates:** support insertions/deletions? (ongoing)

Thanks for your attention!

References i

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

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

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

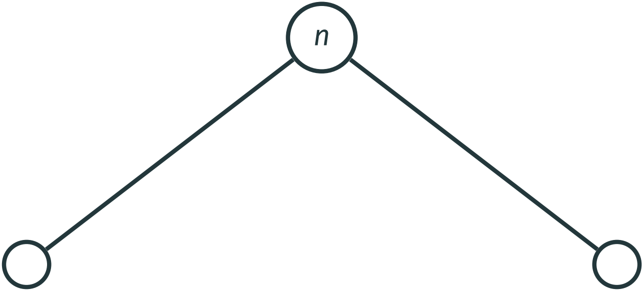
Niewerth, M. and Segoufin, L. (2018).
Enumeration of MSO queries on strings with constant delay and logarithmic updates.
In *PODS*.
To appear.

Set circuit construction

- **Automaton:** *"Select all node pairs* (x, y)*"*
- **States:** $\{\emptyset, x, y, xy\}$

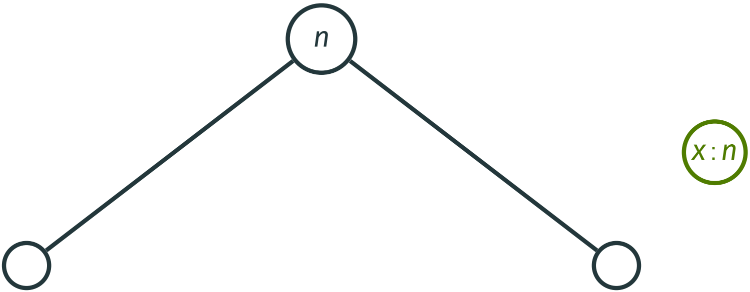
Set circuit construction

- **Automaton:** *"Select all node pairs* (x, y)*"*
- **States:** $\{\emptyset, x, y, xy\}$



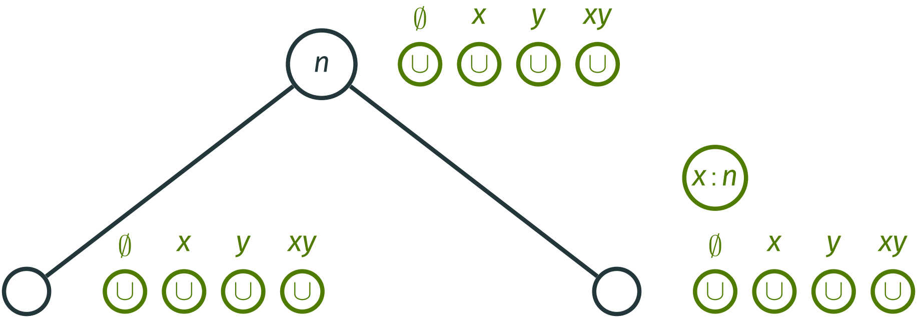
Set circuit construction

- **Automaton:** *"Select all node pairs (x, y)"*
- **States:** $\{\emptyset, x, y, xy\}$



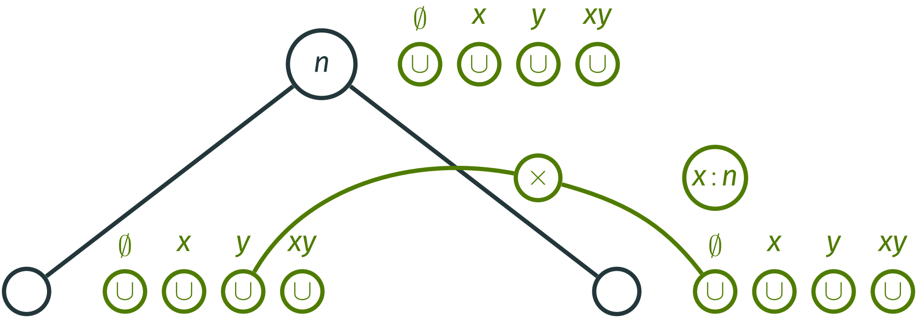
Set circuit construction

- **Automaton:** *"Select all node pairs* (x, y)*"*
- **States:** $\{\emptyset, x, y, xy\}$



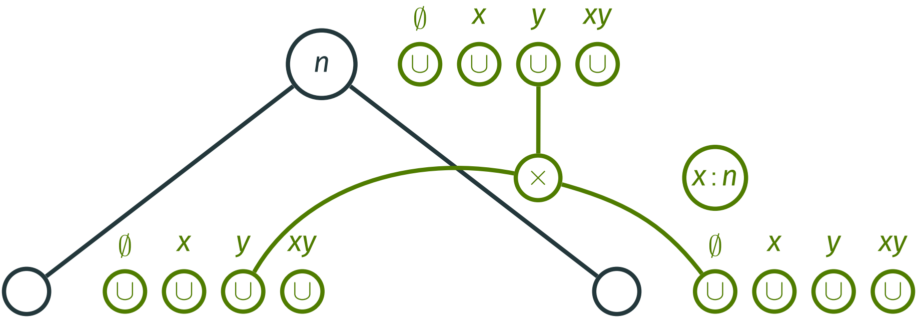
Set circuit construction

- **Automaton:** *"Select all node pairs* (x, y)*"*
- **States:** $\{\emptyset, x, y, xy\}$



Set circuit construction

- **Automaton:** *"Select all node pairs* (x, y)*"*
- **States:** $\{\emptyset, x, y, xy\}$



Set circuit construction

- **Automaton:** *"Select all node pairs (x, y)"*
- **States:** $\{\emptyset, x, y, xy\}$