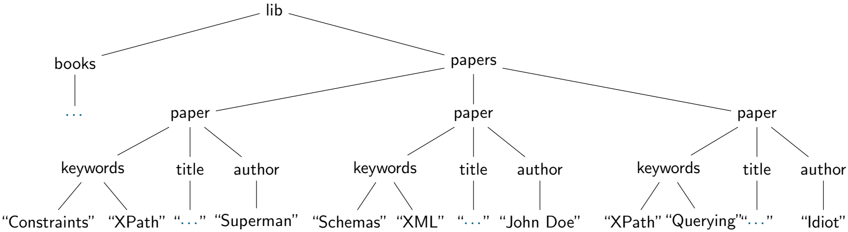
# Incremental XPath Evaluation

H. Björklund W. Gelade M. Marquardt W. Martens

Umeå University, Sweden
University of Bayreuth, Germany

# XML

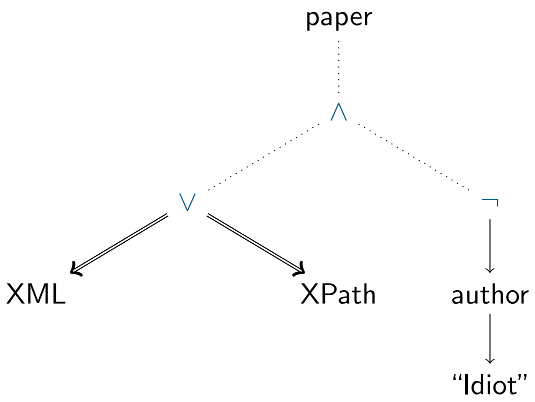

lib
books
papers
...
paper
paper
paper
keywords
title
author
keywords
title
author
keywords
title
author
"Constraints"
"XPath"
"..."
"Superman"
"Schemas"
"XML"
"..."
"John Doe"
"XPath"
"Querying"
"..."
"Idiot"

# XPath

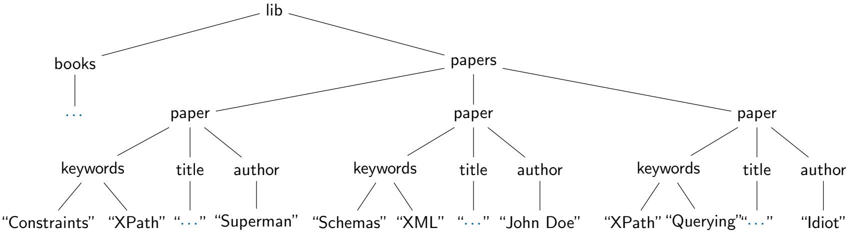
We are interested in:
"All papers about XML or XPath that are not written by Idiot."

# XPath

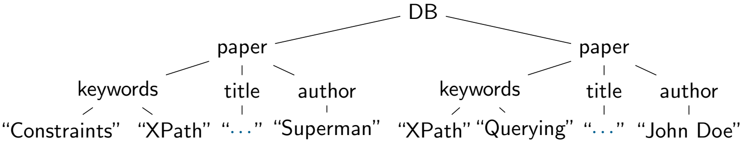
We are interested in:
"All papers about XML or XPath that are not written by Idiot."



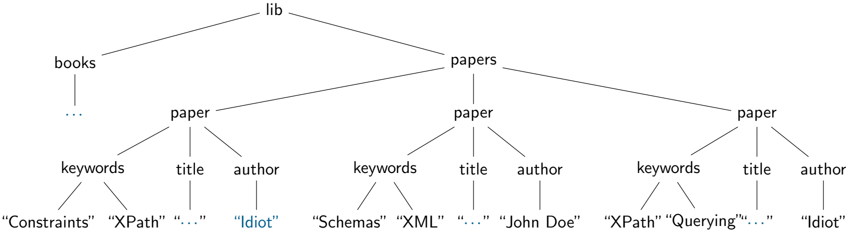
# Motivating example



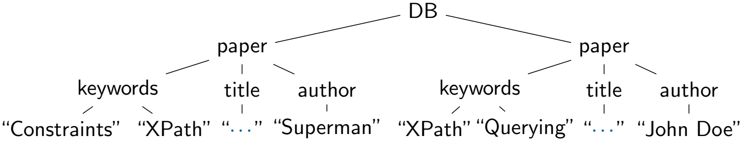

## Local Database



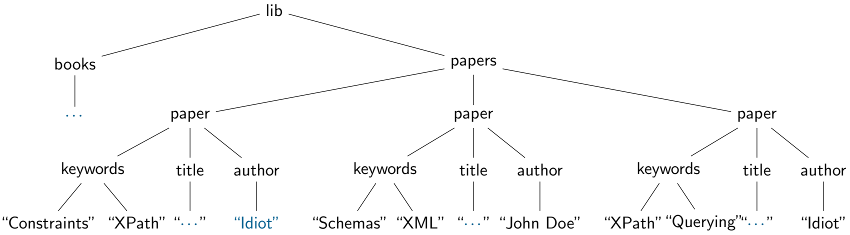
# Motivating example



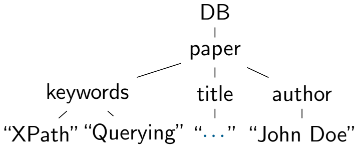

## Local Database



# Motivating example




## Local Database



So ...

# So ...

... what on Earth does this have to do with tree transducers?

# Incremental XPath Evaluation

We consider two versions of the problem:

- Incremental View Maintenance. We want to maintain a view.
- Incremental Boolean Evaluation. We want to maintain (non)-satisfaction of a trigger.

# Incremental Boolean Evaluation

Given:

- An XPath query $Q$
- An XML document $D$
- An update $u$ that changes $D$ into $D'$

# Incremental Boolean Evaluation

Given:

- An XPath query $Q$
- An XML document $D$
- An update $u$ that changes $D$ into $D'$

Question: Does $D'$ satisfy $Q$?

(Does $Q$ return a non-empty answer on $D'$?)

## Incremental Boolean Evaluation

Given:

- An XPath query $Q$
- An XML document $D$
- An update $u$ that changes $D$ into $D'$

Question: Does $D'$ satisfy $Q$?

(Does $Q$ return a non-empty answer on $D'$?)

We consider three aspects of the complexity.

- Time for performing re-evaluation.
- Size of auxiliary data structure.
- Time for update of auxiliary data.

# Updates

We allow the following atomic updates:

- Relabel(v,a): Overwrite the label of $v$ with $a$.
- InsertNS(v,a): Insert a leaf with label $a$ as the next sibling of $v$.
- InsertFC(v,a): Insert a leaf with label $a$ as the first child of $v$.
- Delete(v): Delete the subtree rooted at $v$.

In the best of all worlds ...

# In the best of all worlds ...

... we would be able to tell you that XPath View Maintenance is possible in

- time $\text{polylog}(D) \cdot \text{poly}(Q)$ and
- auxiliary space $\text{poly}(D, Q)$.

## In this world ...

# In this world ...

... we can offer you the following.

| | Fragment | Complexity |
|---|---|---|
| (1) | XPath<br>Patterns | Time: $O(\text{polylog}(\|D\|)) \cdot 2^{O(\|Q\|)}$<br>Size: $O(\|D\|) \cdot 2^{O(\|Q\|)}$ |
| (2) | XPath<br>Patterns | Time: $O(\text{depth}(D) \cdot \log(\text{width}(D))) \cdot 2^{O(\|Q\|)}$<br>Size: $O(\|D\|) \cdot 2^{O(\|Q\|)}$ |
| (3) | $\downarrow, \Downarrow, \wedge, \vee, \neg$ | Time: $O(\text{depth}(D) \cdot \|Q\|)$<br>Size: $O(\|D\| \cdot \|Q\|)$ |
| (4) | $\rightarrow, \Rightarrow, \wedge$ | Time: $O(\log(\|D\|) \cdot \text{poly}(\|Q\|))$<br>Size: $O(\|D\| \cdot \|Q\|^3)$ |
| (5) | $\downarrow, \Downarrow, \rightarrow, \Rightarrow, \wedge$ | Time $O(\text{depth}(D) \cdot \log(\text{width}(D)) \cdot \text{poly}(\|Q\|))$<br>Size: $O(\|D\| \cdot \|Q\|^3)$ |

# Full Core XPath

**Theorem (Balmin, Papakonstantinou, Vianu, 2005)**

*Incremental evaluation for an unranked tree automaton $A$ can be done in*

- *time* $\log(D) \cdot poly(A)$
- *auxspace* $D \cdot poly(A)$

# Full Core XPath

### Theorem (Balmin, Papakonstantinou, Vianu, 2005)

*Incremental evaluation for an unranked tree automaton $A$ can be done in*

- *time* $\log(D) \cdot poly(A)$
- *auxspace* $D \cdot poly(A)$

### Theorem

*An XPath query $Q$ can be transformed into an unranked tree automaton of size $2^{O(Q)}$ in time $2^{O(Q)}$.*

## Full Core XPath

### Corollary

*Incremental Boolean Evaluation for Core XPath is possible in*

- *time* $\log(D) \cdot 2^{O(Q)}$ *and*
- *auxspace* $D \cdot 2^{O(Q)}$.

# Full Core XPath

### Corollary

*Incremental Boolean Evaluation for Core XPath is possible in*

- *time* $\log(D) \cdot 2^{O(Q)}$ *and*
- *auxspace* $D \cdot 2^{O(Q)}$.

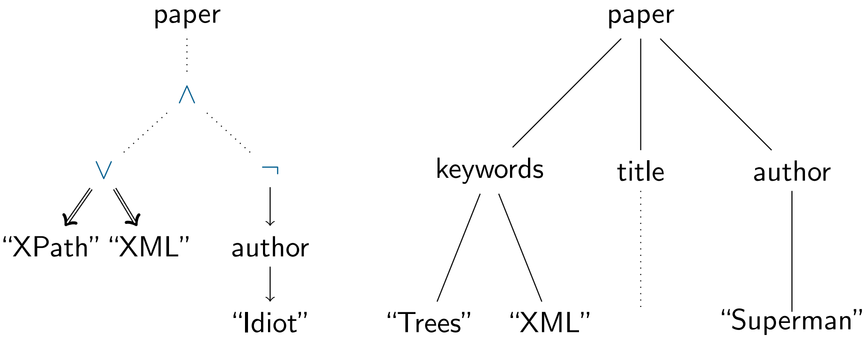
A different result by Balmin et al. gives us the following.

### Corollary

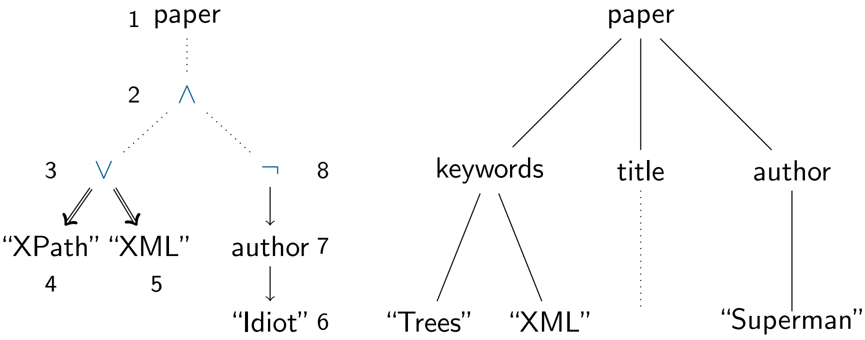
*Incremental Boolean Evaluation for Core XPath is possible in*

- *time* $depth(D) \cdot \log(width(D)) \cdot 2^{O(Q)}$ *and*
- *auxspace* $D \cdot 2^{O(Q)}$.

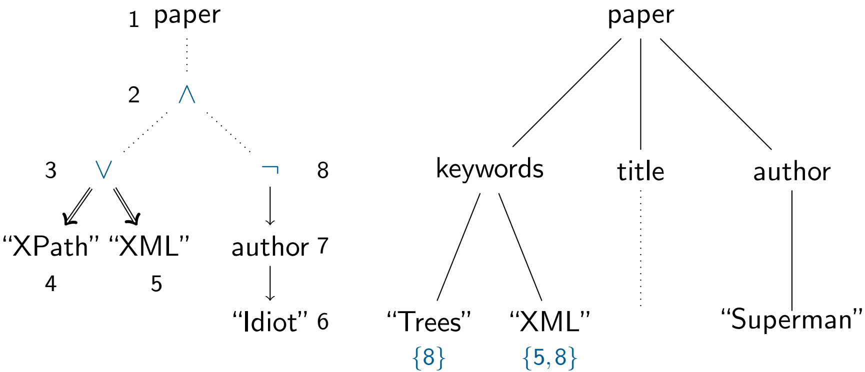
# Downward XPath



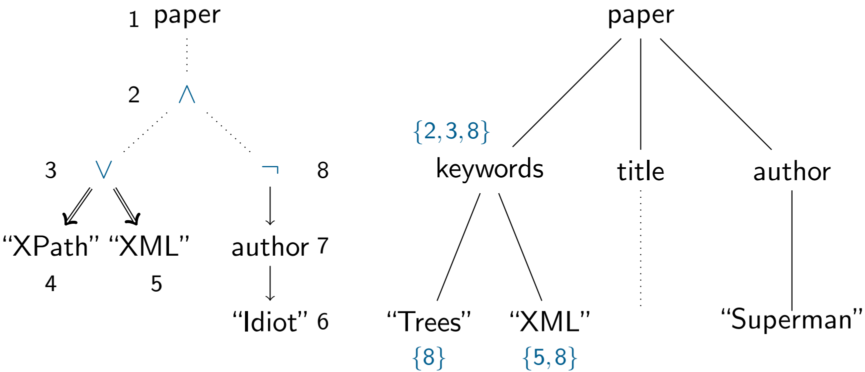
# Downward XPath



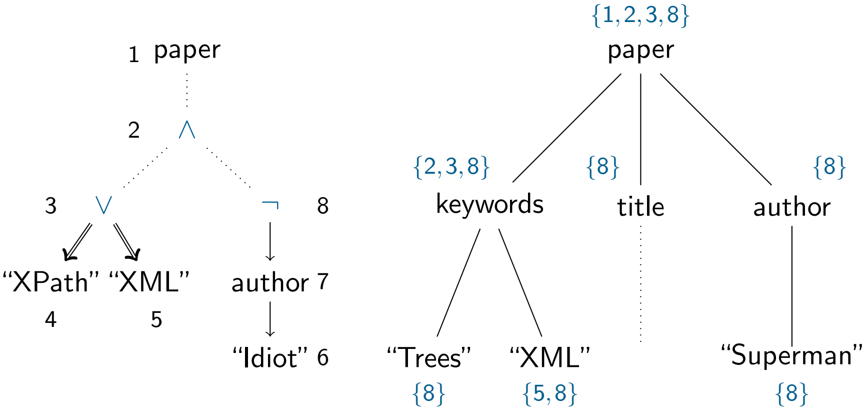
# Downward XPath



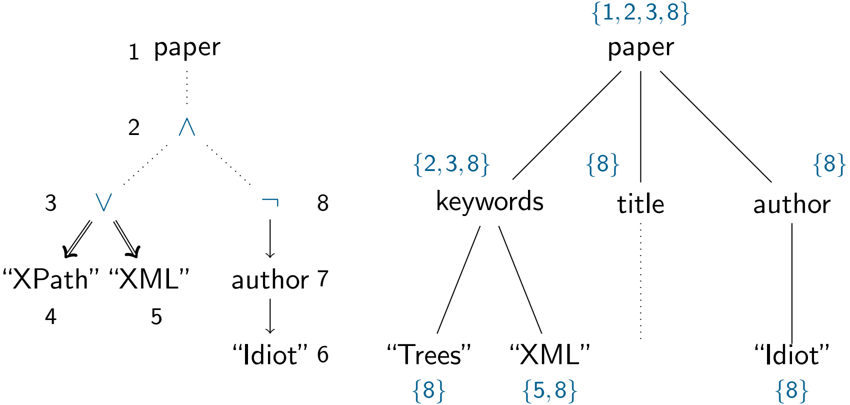
# Downward XPath



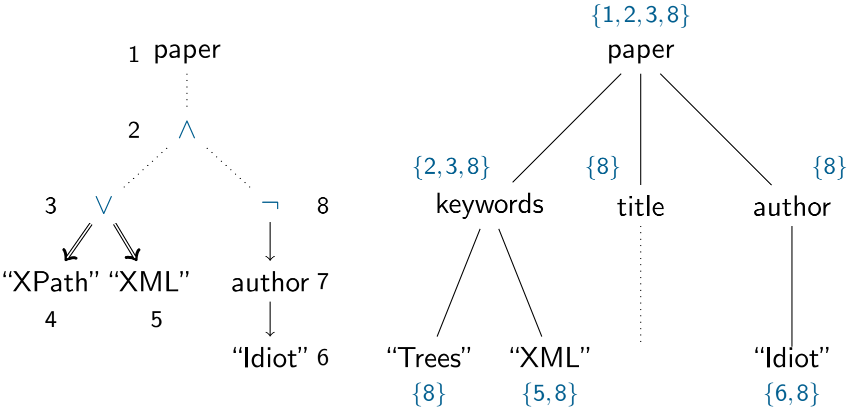
# Downward XPath



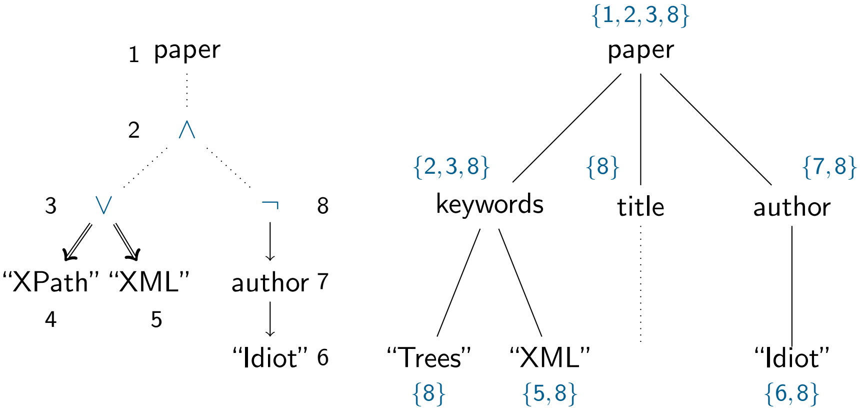
# Downward XPath



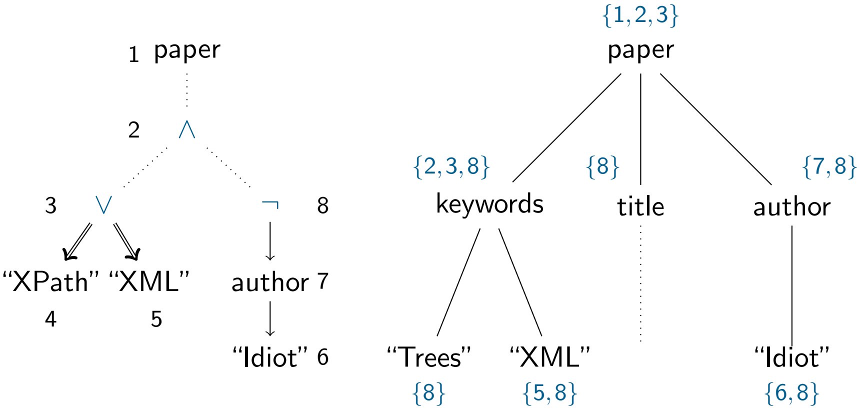
# Downward XPath



# Downward XPath



# Downward XPath



# Downward XPath

**Theorem**

*Incremental Boolean Evaluation and View Maintenance for downward XPath is possible in*

- *time* $O(depth(D) \cdot Q)$ *and*
- *auxspace* $O(D \cdot Q)$.

# Forward XPath

Forward XPath is the fragment that uses the operators

- child ($\downarrow$),
- descendant ($\Downarrow$),
- next sibling ($\rightarrow$),
- following sibling ($\Rightarrow$), and
- and ($\wedge$).

# Forward XPath on strings

We first need to figure out how to handle XPath($\rightarrow, \Rightarrow, \wedge$) over strings.

# Forward XPath on strings

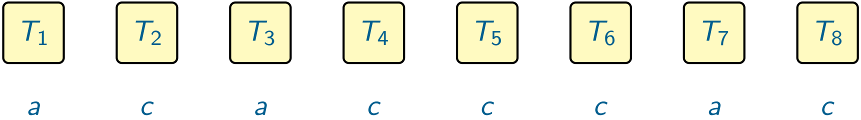
We first need to figure out how to handle XPath($\rightarrow, \Rightarrow, \wedge$) over strings.

Warmup: Incremental NFA evaluation.

# Incremental NFA evaluation

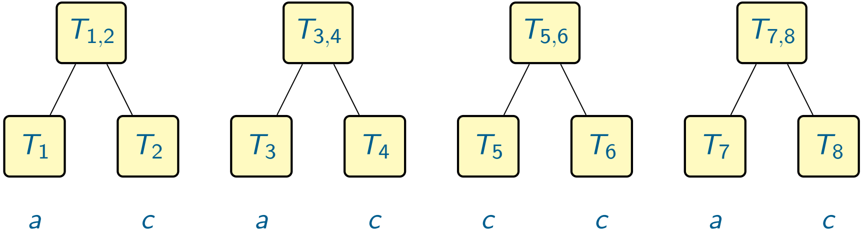
$a \quad c \quad a \quad c \quad c \quad c \quad a \quad c$

# Incremental NFA evaluation



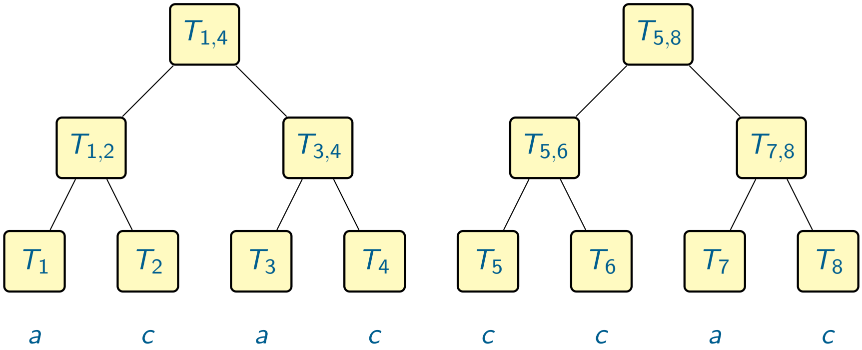

Example: $T_1 = \{(q_0, q_1), (q_0, q_3), (q_2, q_4)\}$

# Incremental NFA evaluation



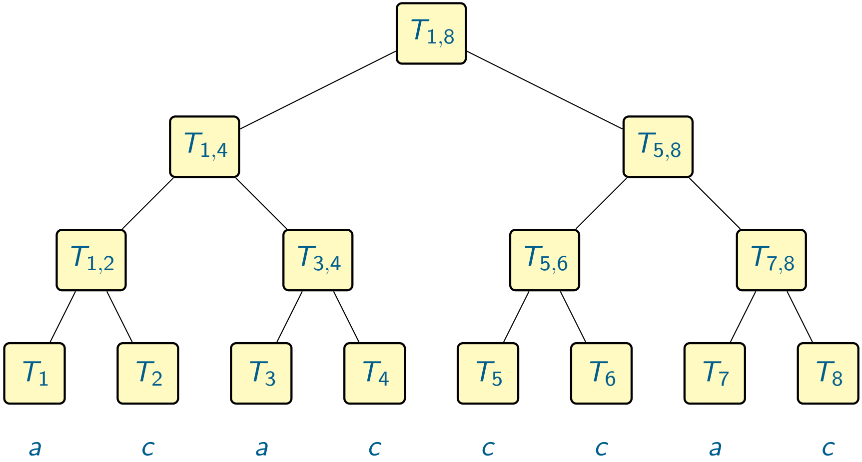

Example: $T_{1,2} = T_1 \circ T_2$

# Incremental NFA evaluation



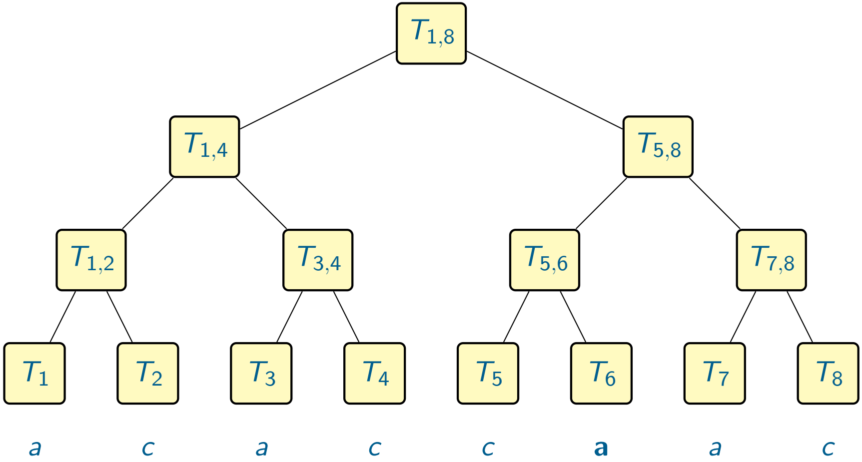

Example: $T_{1,2} = T_1 \circ T_2$

# Incremental NFA evaluation



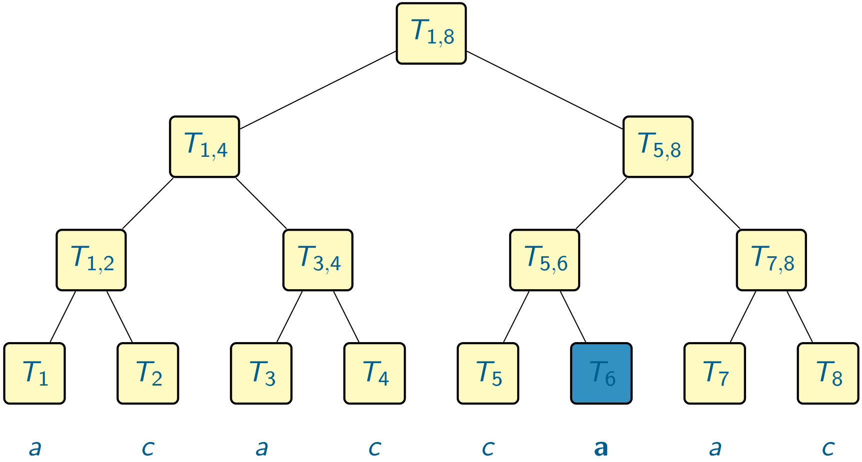

Example: $T_{1,2} = T_1 \circ T_2$

# Incremental NFA evaluation



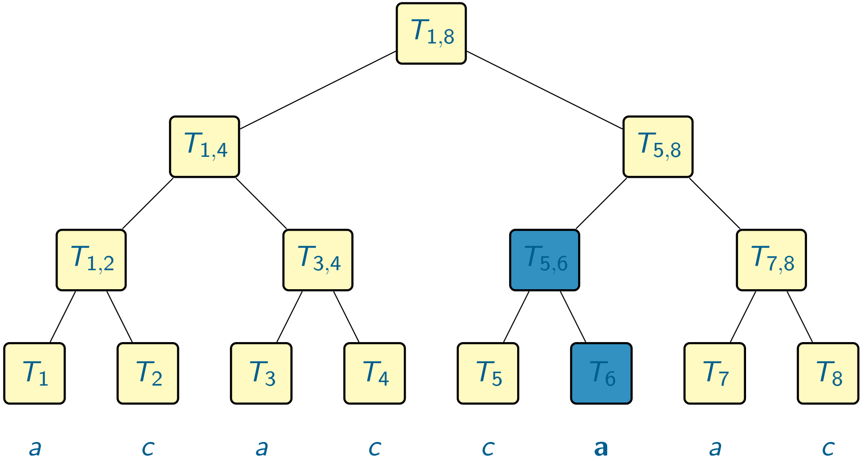

Example: $T_{1,2} = T_1 \circ T_2$

# Incremental NFA evaluation



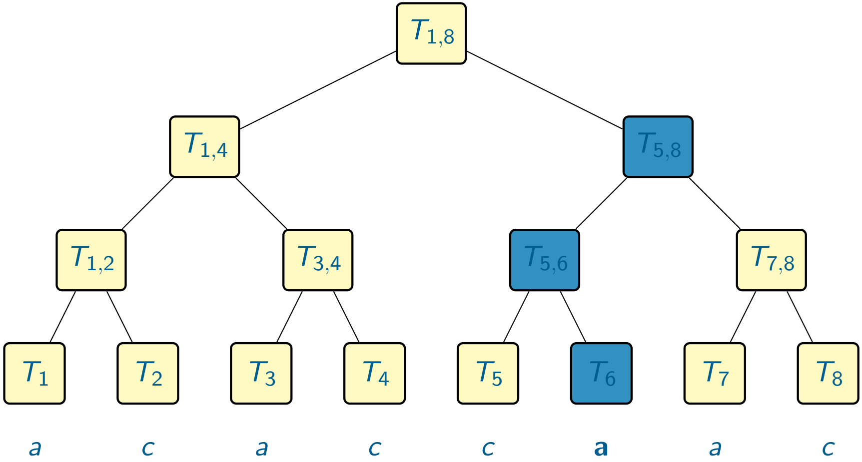

Example: $T_{1,2} = T_1 \circ T_2$

# Incremental NFA evaluation



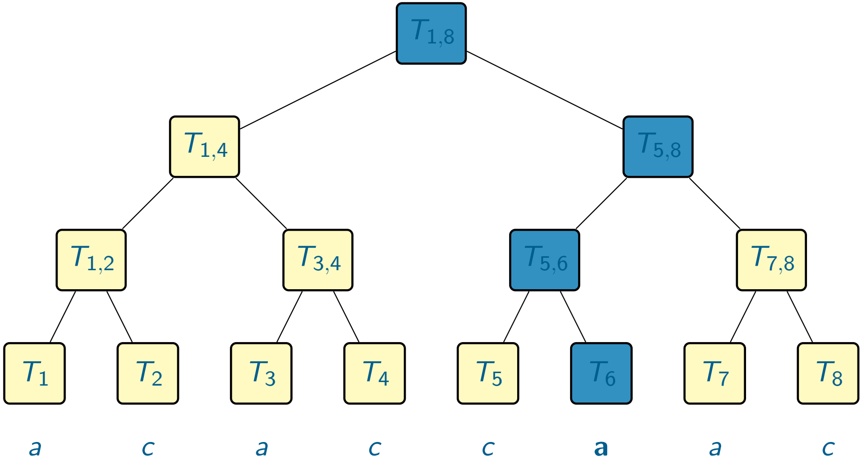

Example: $T_{1,2} = T_1 \circ T_2$

# Incremental NFA evaluation




Example: $T_{1,2} = T_1 \circ T_2$

# Incremental NFA evaluation




Example: $T_{1,2} = T_1 \circ T_2$

# Forward XPath on strings

Suggestion: Convert $Q$ into an equivalent NFA and use the technique we just saw.

# Forward XPath on strings

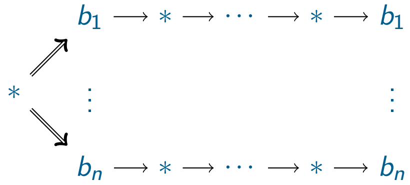
Suggestion: Convert $Q$ into an equivalent NFA and use the technique we just saw.

Problem: The NFA would have exponential size, making the computation and auxiliary data exponential in $Q$.

# Forward XPath on strings

Suggestion: Convert $Q$ into an equivalent NFA and use the technique we just saw.

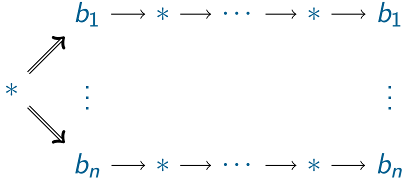
Problem: The NFA would have exponential size, making the computation and auxiliary data exponential in $Q$.



# Forward XPath on strings

Suggestion: Convert $Q$ into an equivalent NFA and use the technique we just saw.

Problem: The NFA would have exponential size, making the computation and auxiliary data exponential in $Q$.



The XPath query can express a kind of alternation.

# Forward XPath on strings

Suggestion: Convert $Q$ into an equivalent AFA.

# Forward XPath on strings

Suggestion: Convert $Q$ into an equivalent AFA.

Problem: It is not clear how to maintain AFA acceptance without using time and auxiliary space exponential in the AFA and thus in $Q$.

# Forward XPath on strings

Suggestion: Convert $Q$ into an equivalent AFA.

Problem: It is not clear how to maintain AFA acceptance without using time and auxiliary space exponential in the AFA and thus in $Q$.

Saving grace: Forward XPath queries over strings are not quite as efficient as AFAs.

# Forward XPath on strings

Suggestion: Convert $Q$ into an equivalent AFA.

Problem: It is not clear how to maintain AFA acceptance without using time and auxiliary space exponential in the AFA and thus in $Q$.

Saving grace: Forward XPath queries over strings are not quite as efficient as AFAs.

Our solution: A hand-crafted, special purpose algorithm.

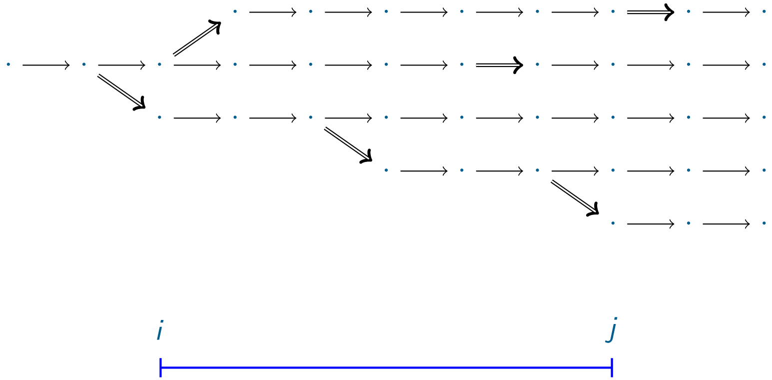
Basic idea: Treat each path from root to leaf in the query as an NFA.

# Forward XPath on strings

What are we going to store in the sets $T_{i,j}$?

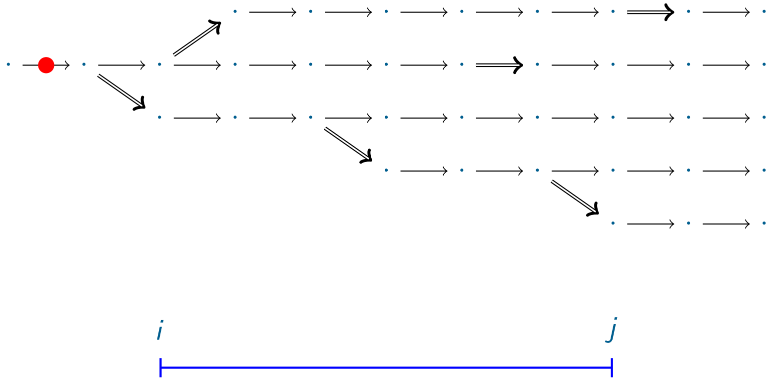
# Forward XPath on strings

What are we going to store in the sets $T_{i,j}$?



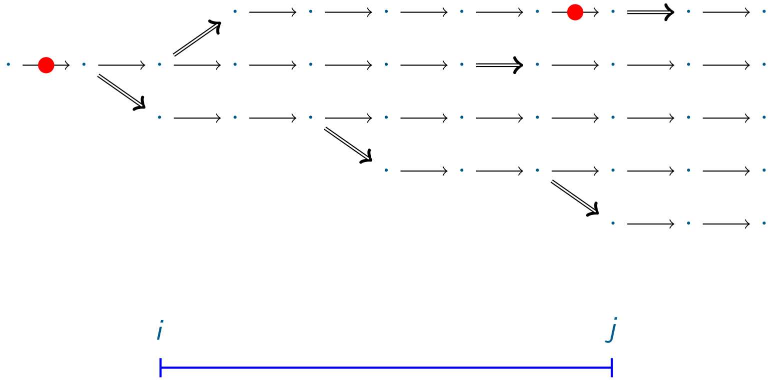
# Forward XPath on strings

What are we going to store in the sets $T_{i,j}$?



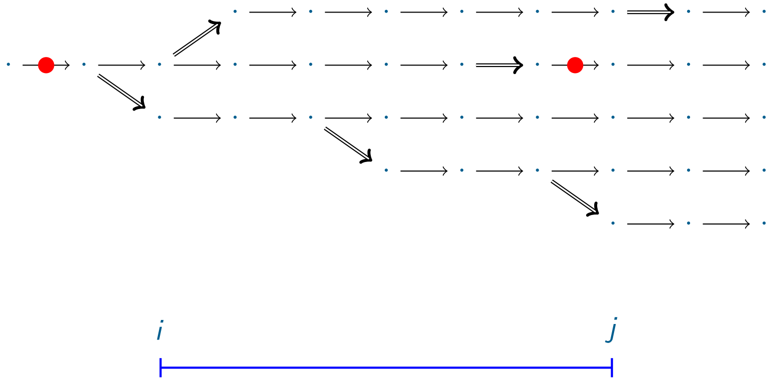
# Forward XPath on strings

What are we going to store in the sets $T_{i,j}$?



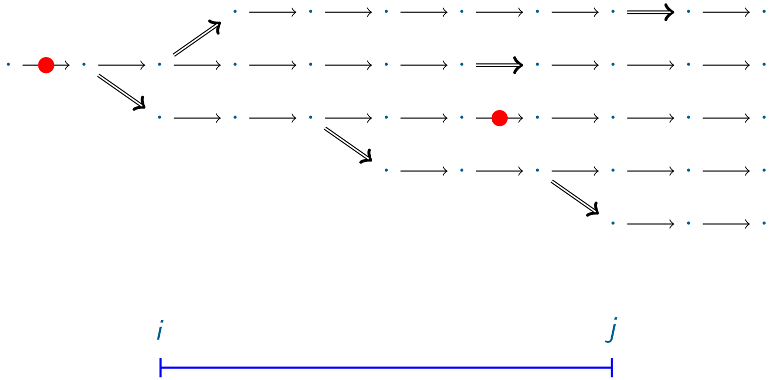
# Forward XPath on strings

What are we going to store in the sets $T_{i,j}$?



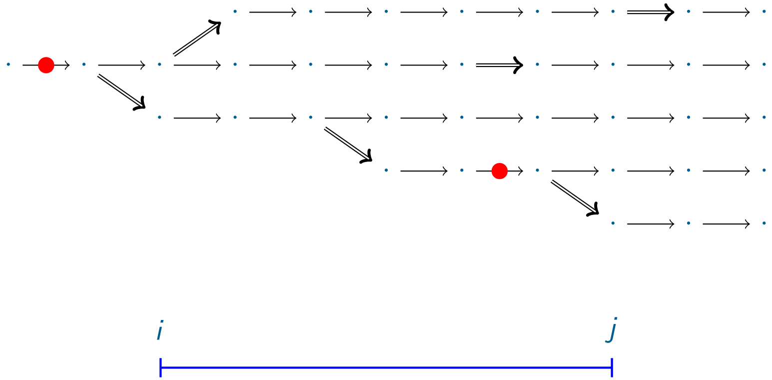
# Forward XPath on strings

What are we going to store in the sets $T_{i,j}$?



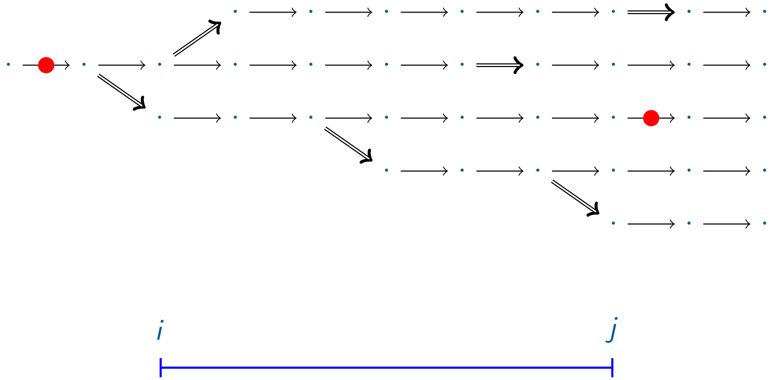
# Forward XPath on strings

What are we going to store in the sets $T_{i,j}$?



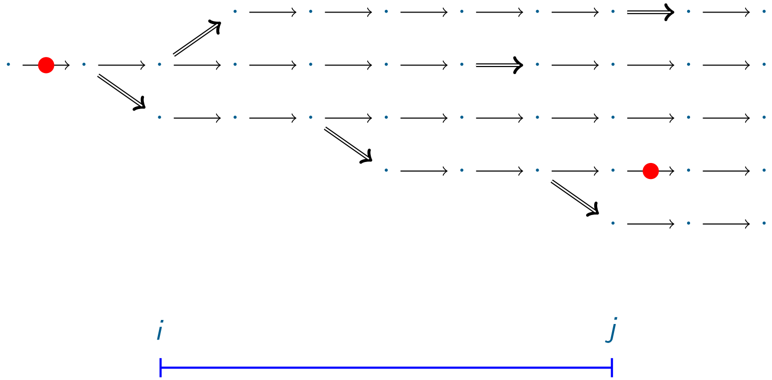
# Forward XPath on strings

What are we going to store in the sets $T_{i,j}$?



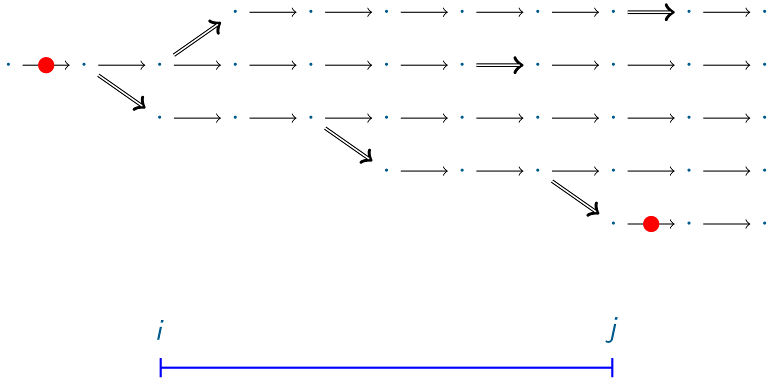
# Forward XPath on strings

What are we going to store in the sets $T_{i,j}$?



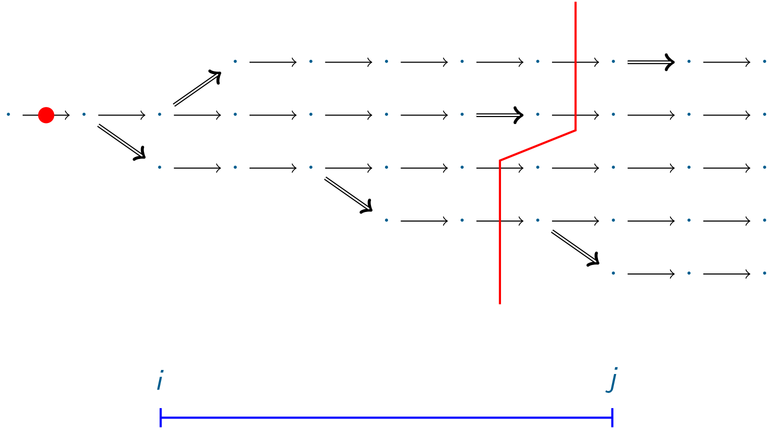
# Forward XPath on strings

What are we going to store in the sets $T_{i,j}$?



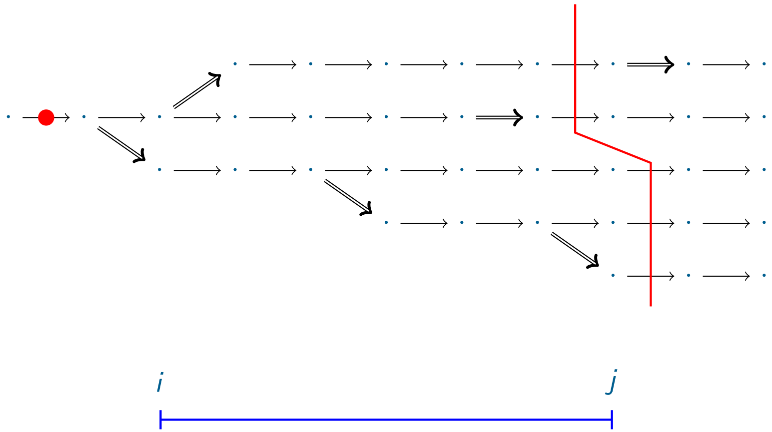
# Forward XPath on strings

What are we going to store in the sets $T_{i,j}$?



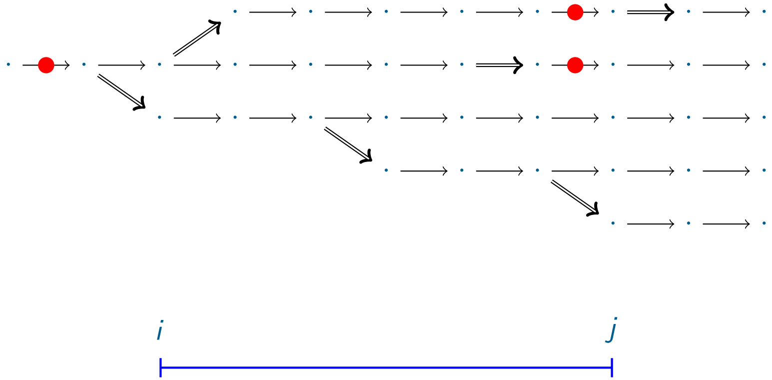
# Forward XPath on strings

What are we going to store in the sets $T_{i,j}$?



# Forward XPath on strings

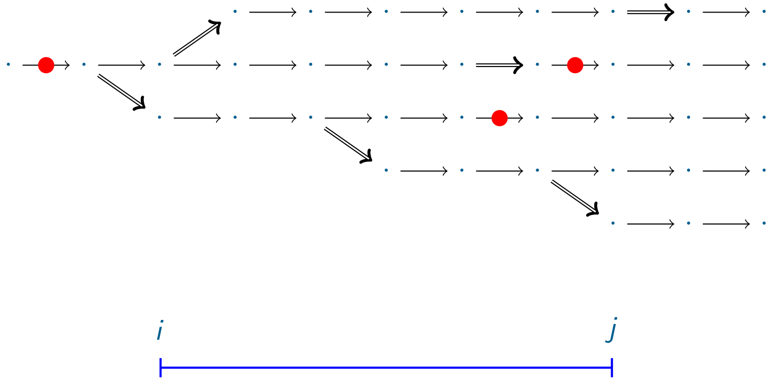
What are we going to store in the sets $T_{i,j}$?




We save matching triples

# Forward XPath on strings

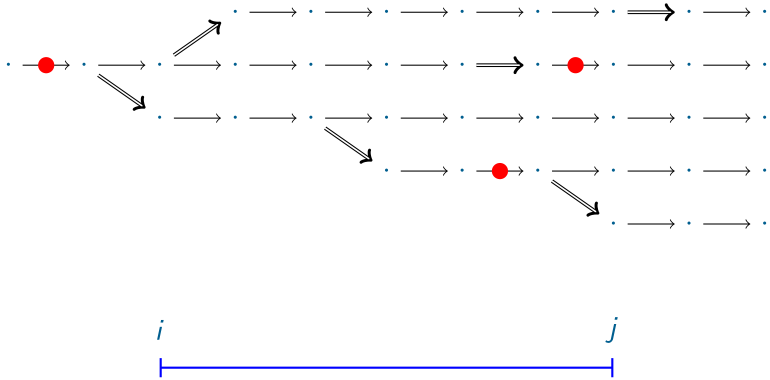
What are we going to store in the sets $T_{i,j}$?




We save matching triples

# Forward XPath on strings

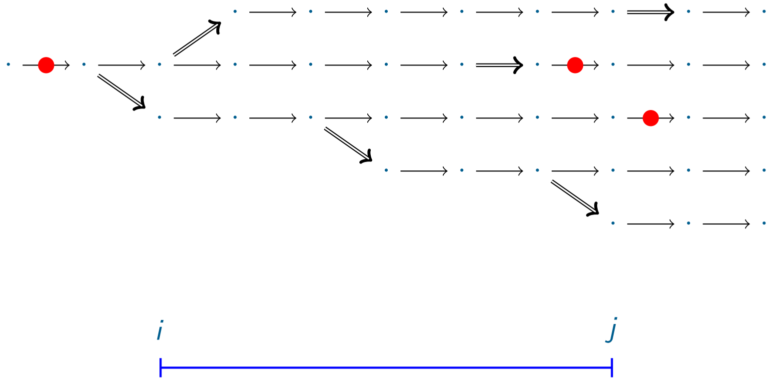
What are we going to store in the sets $T_{i,j}$?




We save matching triples

# Forward XPath on strings

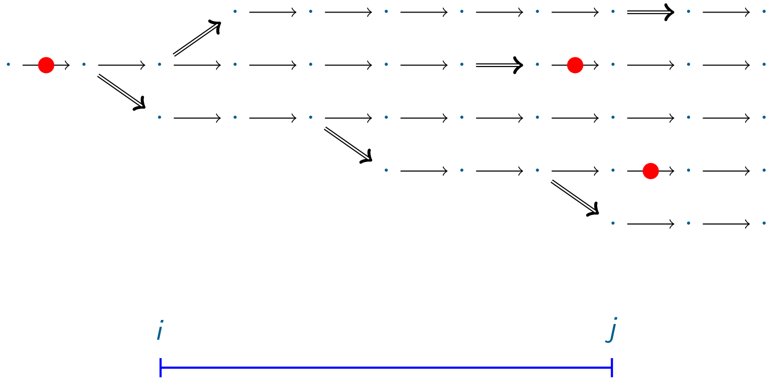
What are we going to store in the sets $T_{i,j}$?




We save matching triples

# Forward XPath on strings

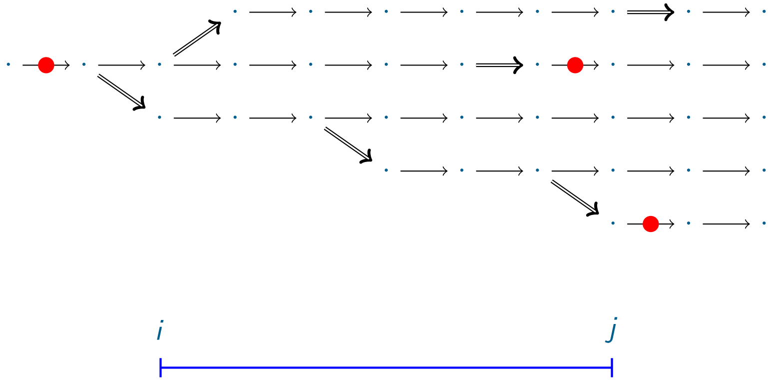
What are we going to store in the sets $T_{i,j}$?




We save matching triples

# Forward XPath on strings

What are we going to store in the sets $T_{i,j}$?




We save matching triples

# Forward XPath on strings

## Theorem

*Incremental Boolean evaluation of Forward XPath on strings is possible in*

- *time* $\log(D) \cdot poly(Q)$ *and*
- *auxspace* $D \cdot Q^3$.

# Forward XPath

Combining the results on Downward XPath on trees and Forward XPath on strings, we get the following.

Theorem

*Incremental Boolean evaluation of Forward XPath on trees is possible in*

- *time* $depth(D) \cdot \log(width(D)) \cdot poly(Q)$ *and*
- *auxspace* $D \cdot Q^3$.

# Final remarks

- Already the Boolean version of the incremental XPath evaluation problem is quite challenging.

# Final remarks

- Already the Boolean version of the incremental XPath evaluation problem is quite challenging.
- Our depth($D$) algorithm for Downward XPath is fairly simple and should perform well in practice.

# Final remarks

- Already the Boolean version of the incremental XPath evaluation problem is quite challenging.
- Our depth($D$) algorithm for Downward XPath is fairly simple and should perform well in practice.
- Our algorithm for Forward XPath is quite involved ...

# Final remarks

- Already the Boolean version of the incremental XPath evaluation problem is quite challenging.
- Our depth($D$) algorithm for Downward XPath is fairly simple and should perform well in practice.
- Our algorithm for Forward XPath is quite involved ...
- ... but becomes simple if next sibling is disallowed ...

# Final remarks

- Already the Boolean version of the incremental XPath evaluation problem is quite challenging.
- Our depth($D$) algorithm for Downward XPath is fairly simple and should perform well in practice.
- Our algorithm for Forward XPath is quite involved ...
- ... but becomes simple if next sibling is disallowed ...
- ... and there might be a less complicated method.

## The big open quesions

For which XPath fragments is incremental Boolean evaluation possible in

- time $\text{polylog}(D) \cdot \text{poly}(Q)$ and
- auxspace $\text{poly}(D) \cdot \text{poly}(Q)$?

For which XPath fragments is incremental view maintenance possible in

- time $\text{polylog}(D) \cdot \text{poly}(Q)$ and
- auxspace $\text{poly}(D) \cdot \text{poly}(Q)$?