

# Online Evaluation of Regular Tree Queries

Alexandru Berlea

*Technische Universität München*

Bonn, June 2006

# Overview

1. Online query evaluation
2. Regular tree queries (RTQs)
3. Online evaluation of RTQs

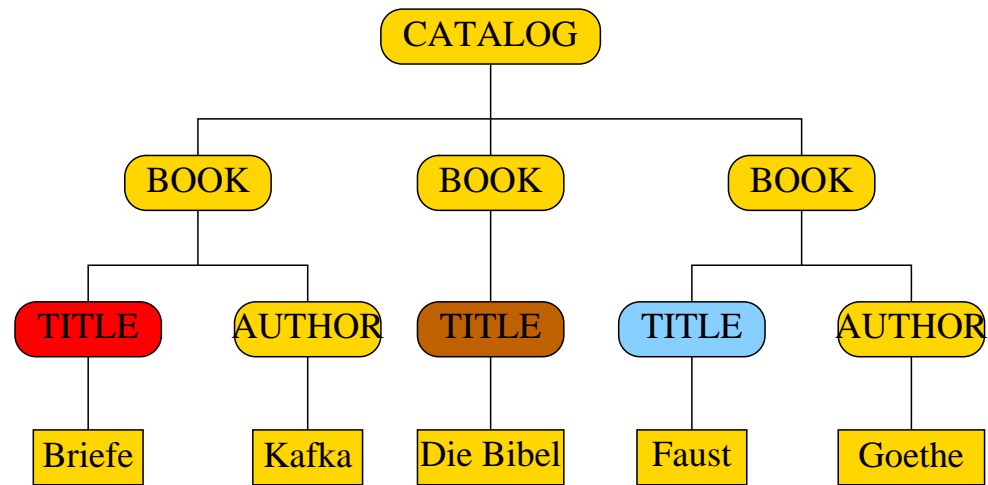
## Motivation

- Most XML applications build the tree representation of their XML input **in-memory**.
- Not suitable if:
  - memory space is insufficient
    - ▷ for very large documents
    - ▷ for devices with little memory
  - processing should preferably be performed while receiving data
    - ▷ monitoring applications
    - ▷ routing application
- Alternative: **event-based/online** processing
  - The input is seen as a sequence of small pieces of informations = **events**(e.g. start-tag, end-tag)
  - Processing is achieved by reacting to the events → **event-based**

## XML-Querying:

//BOOK/TITLE

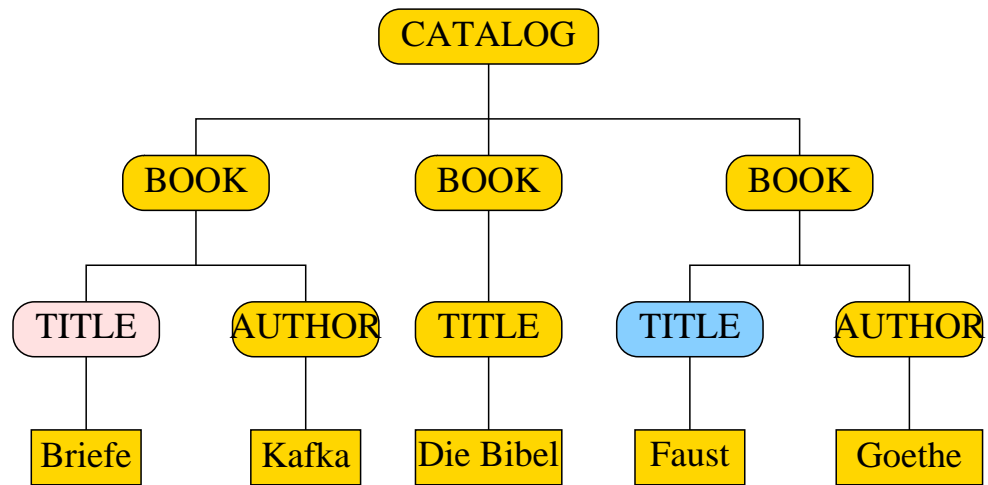
```
<KATALOG>
  <BOOK>
    <TITLE>Briefe</TITLE>
    <AUTHOR>Kafka</AUTHOR>
  </BOOK>
  <BOOK>
    <TITLE>Die Bibel</TITLE>
  </BOOK>
  <BOOK>
    <TITLE>Faust</TITLE>
    <AUTHOR>Goethe</AUTHOR>
  </BOOK>
</KATALOG>
```



## XML-Querying:

//BOOK[AUTHOR]/TITLE

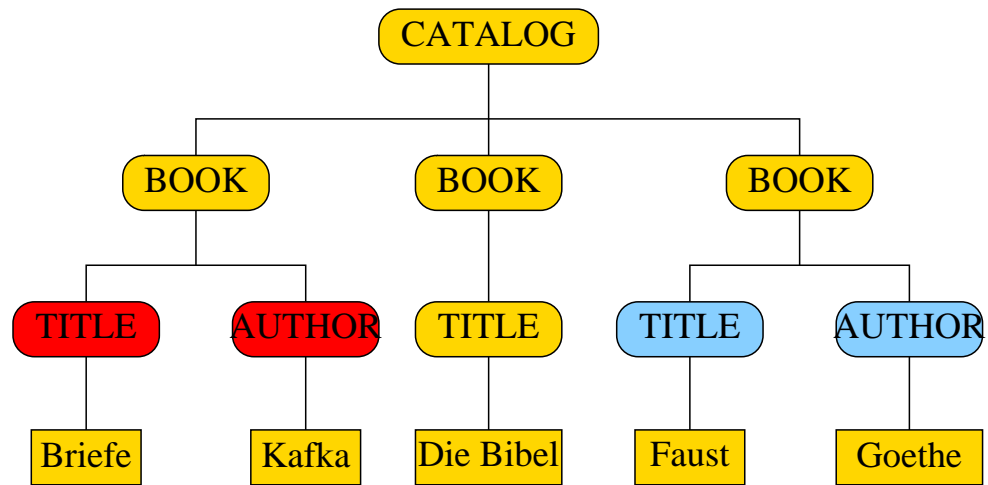
```
<KATALOG>
  <BOOK>
    <TITLE>Briefe</TITLE>
    <AUTHOR>Kafka</AUTHOR>
  </BOOK>
  <BOOK>
    <TITLE>Die Bibel</TITLE>
  </BOOK>
  <BOOK>
    <TITLE>Faust</TITLE>
    <AUTHOR>Goethe</AUTHOR>
  </BOOK>
</KATALOG>
```



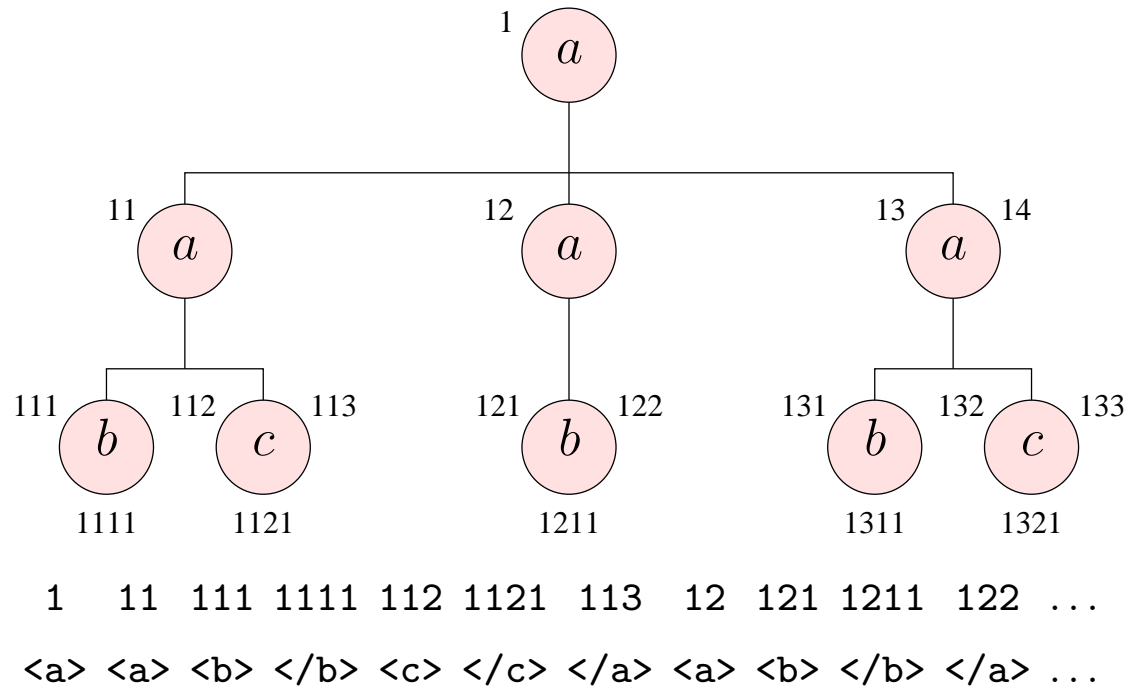
## Binary Query:

## *Titles+Authors*

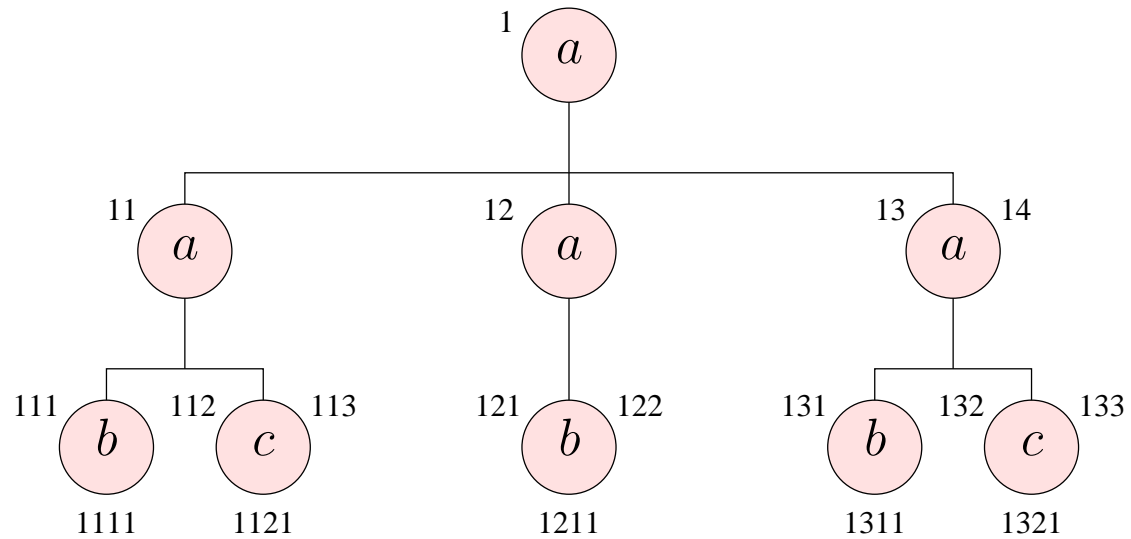
```
<KATALOG>
  <BOOK>
    <TITLE>Briefe</TITLE>
    <AUTHOR>Kafka</AUTHOR>
  </BOOK>
  <BOOK>
    <TITLE>Die Bibel</TITLE>
  </BOOK>
  <BOOK>
    <TITLE>Faust</TITLE>
    <AUTHOR>Goethe</AUTHOR>
  </BOOK>
</KATALOG>
```



# Online Query Evaluation: Requirements



# Online Query Evaluation: Requirements

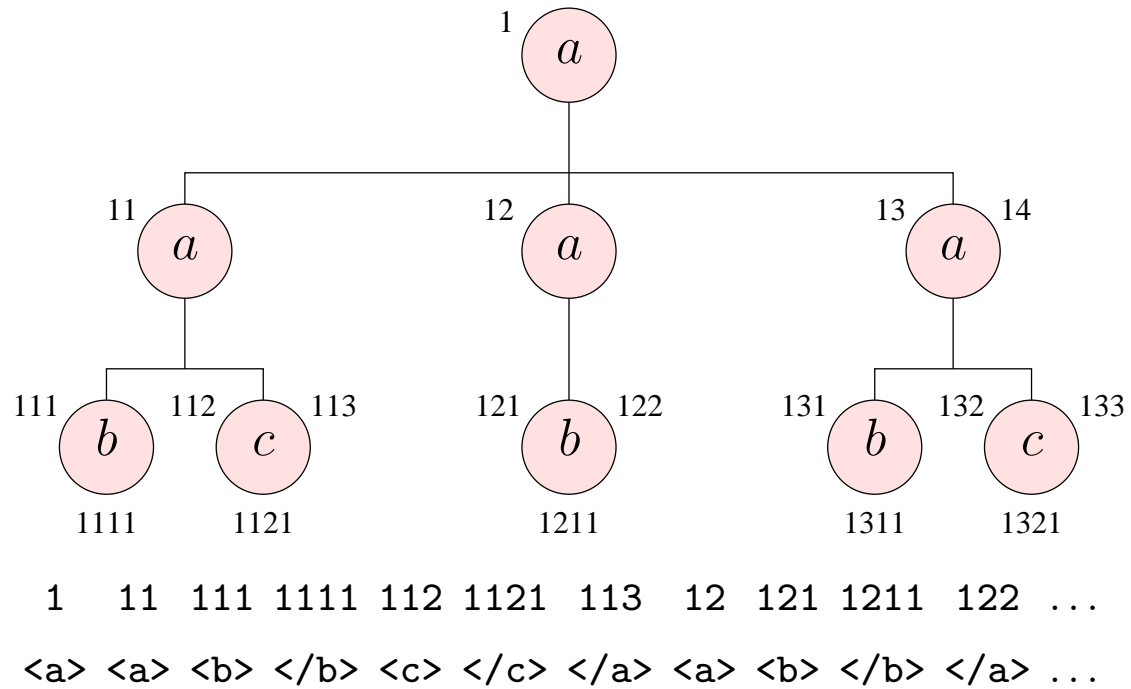


1 11 111 1111 112 1121 113 12 121 1211 122 ...  
 <a> <a> <b> </b> <c> </c> </a> <a> <b> </b> </a> ...

Buffering: depends on the query and the input at hand

//a/b                      not necessary

## Online Query Evaluation: Requirements

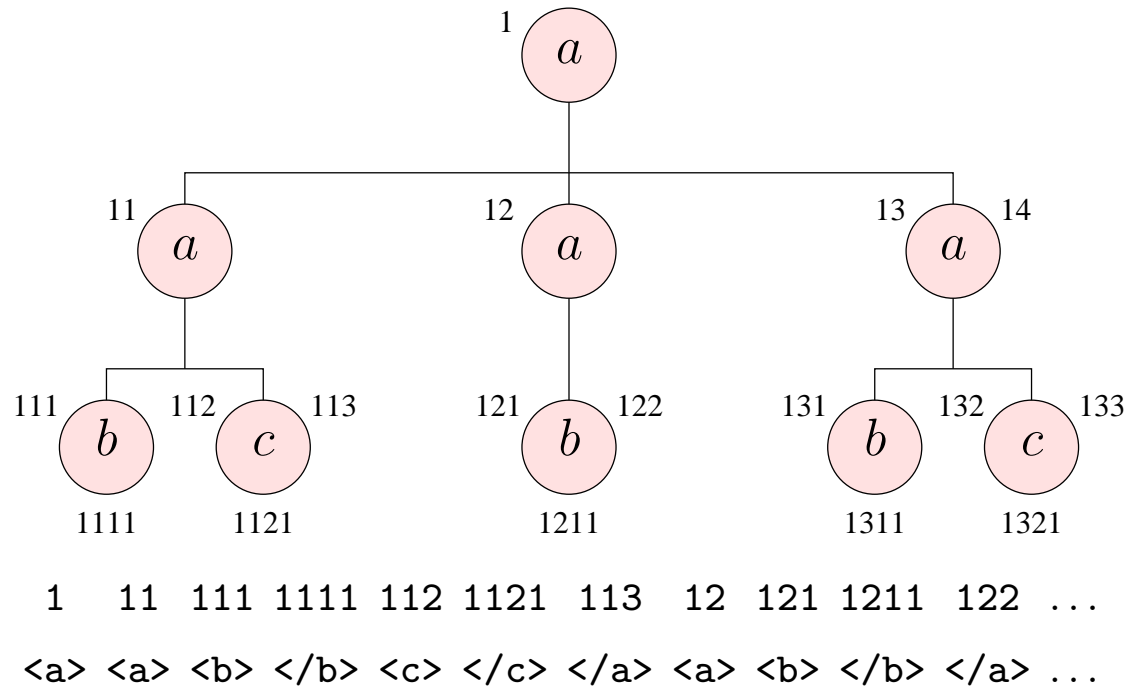


**Buffering:** depends on the query and the input at hand

//a/b            not necessary

//a[c]/b        not (statically) bounded

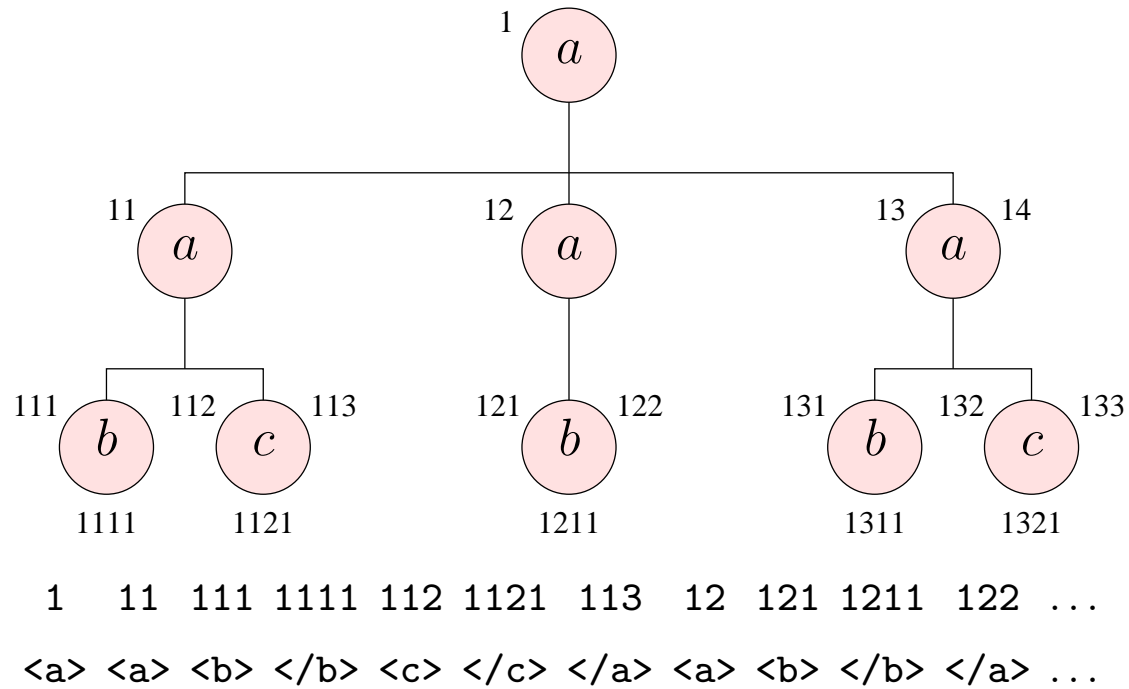
## Online Query Evaluation: Requirements



**Buffering:** depends on the query and the input at hand

- //a/b            not necessary
- //a[c]/b        not (statically) bounded
- /\*[not(d)]/\*    all input

## Online Query Evaluation: Requirements



**Buffering:** depends on the query and the input at hand

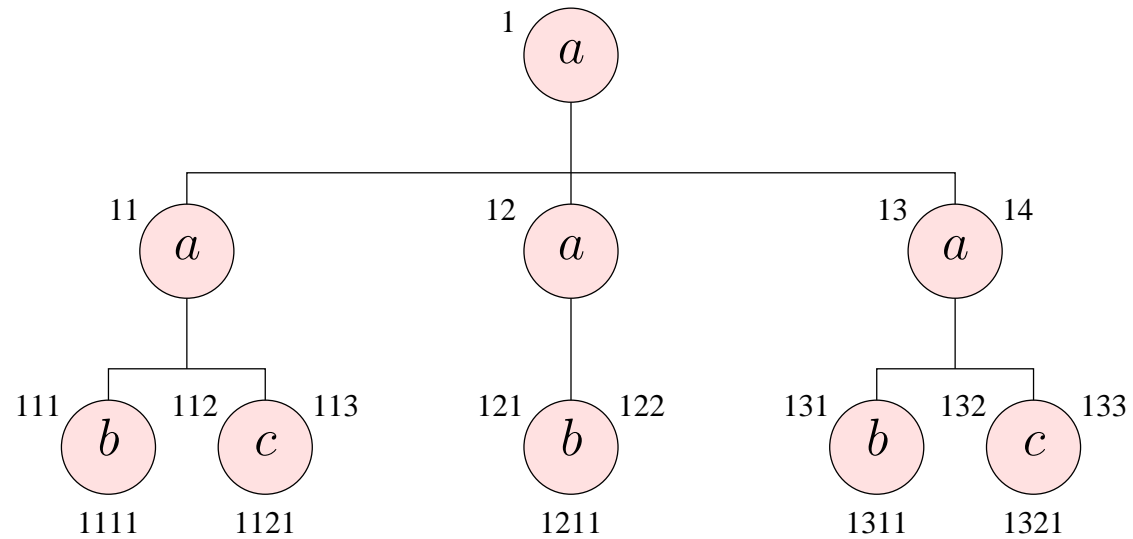
//a/b            not necessary

//a[c]/b        not (statically) bounded

/\*[not(d)]/\*    all input

→ should be **adaptive**

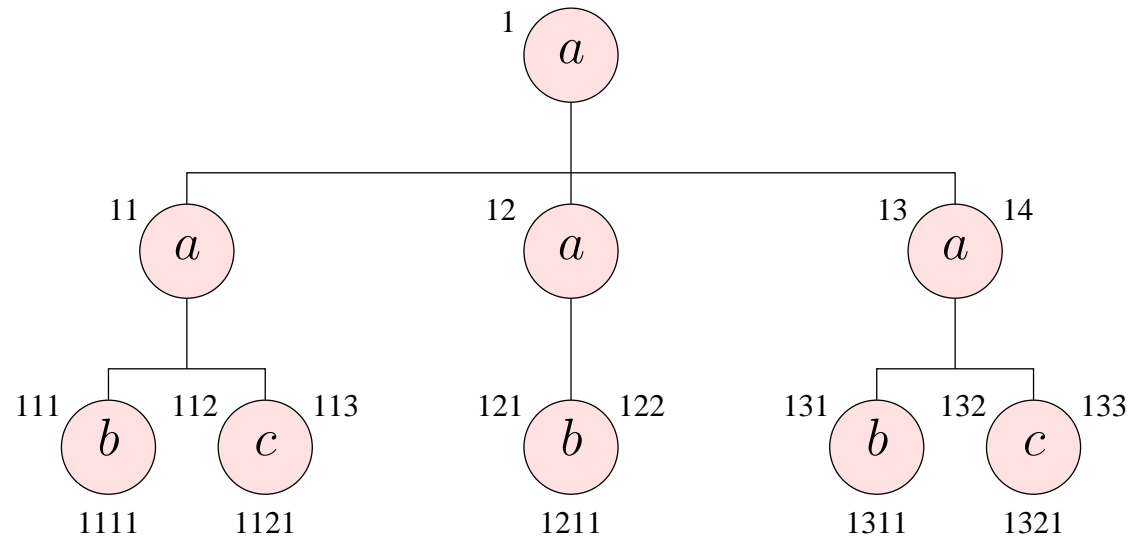
## Online Query Evaluation: Requirements



1 11 111 1111 112 1121 113 12 121 1211 122 ...  
<a> <a> <b> </b> <c> </c> </a> <a> <b> </b> </a> ...

Match reporting: should be progressive

## Online Query Evaluation: Requirements

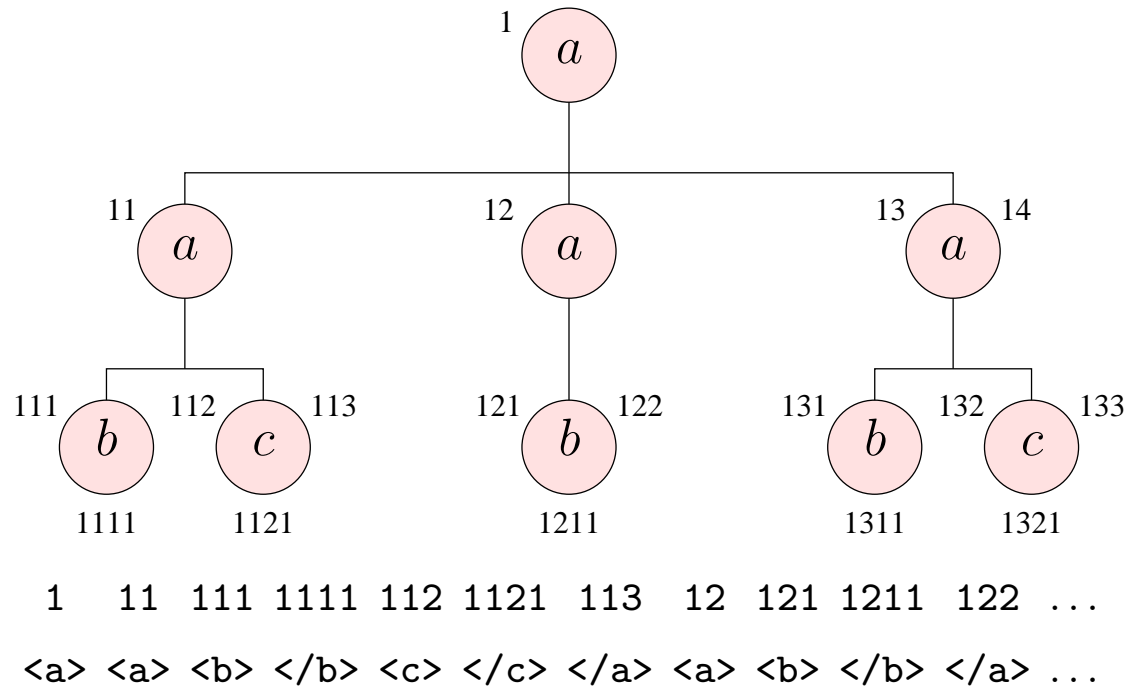


1 11 111 1111 112 1121 113 12 121 1211 122 ...  
 <a> <a> <b> </b> <c> </c> </a> <a> <b> </b> </a> ...

Match reporting: should be progressive

//a/b immediately

# Online Query Evaluation: Requirements

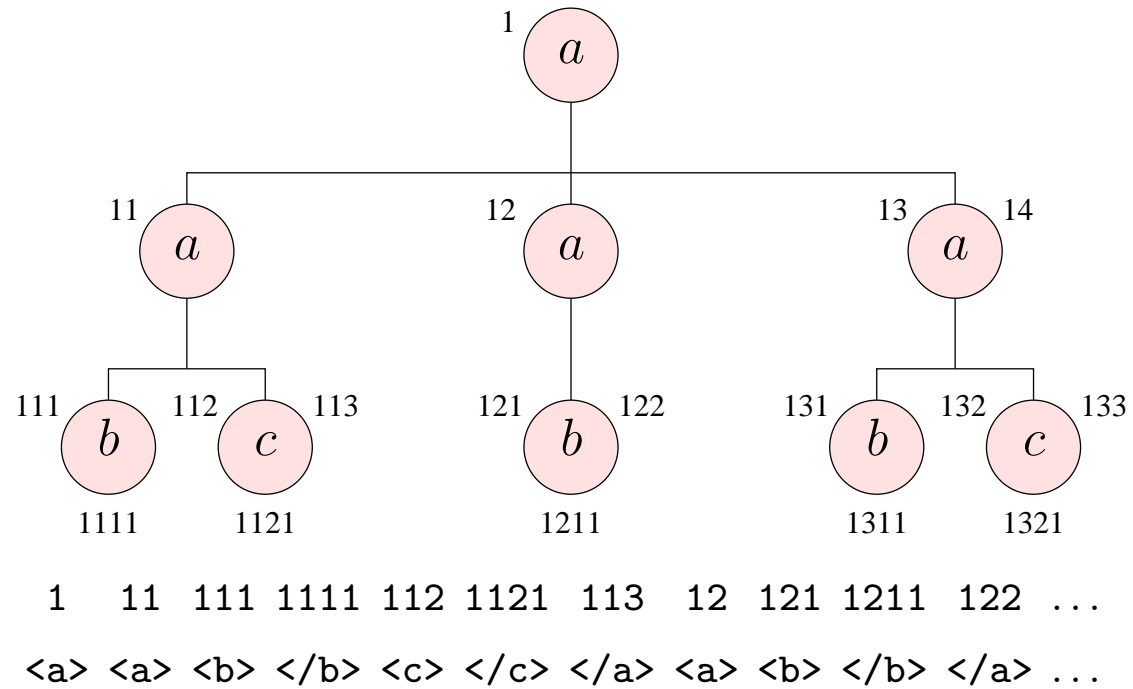


Match reporting: should be progressive

//a/b            immediately

//a[c]/b        delayed

## Online Query Evaluation: Requirements



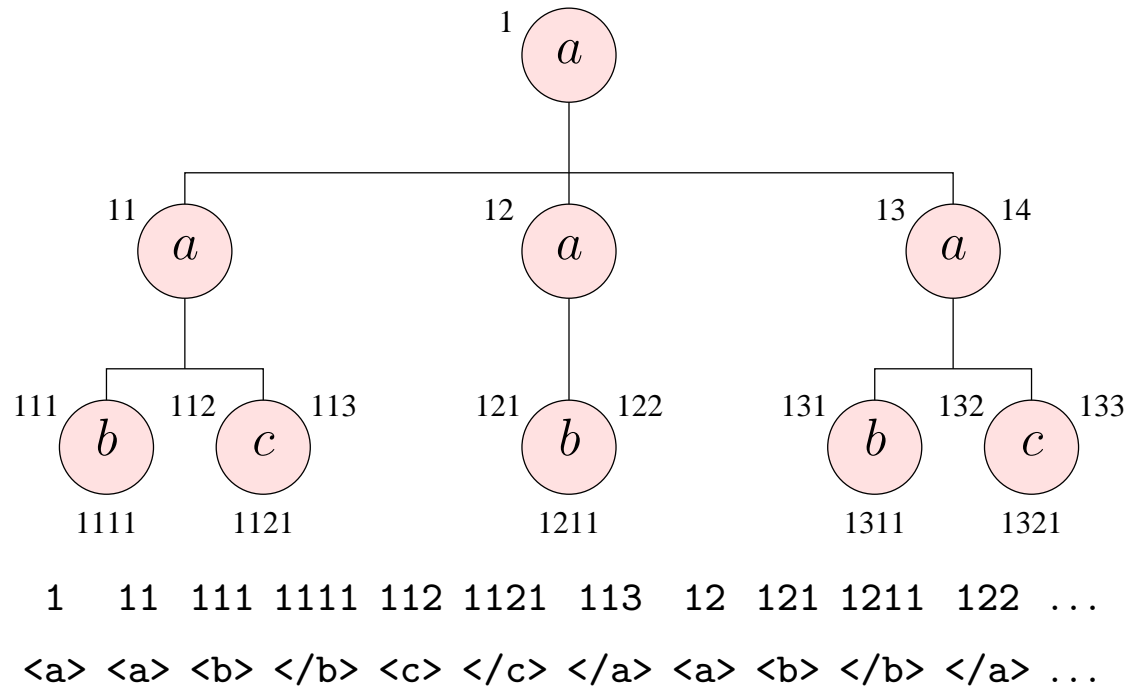
Match reporting: should be progressive

//a/b            immediately

//a[c]/b        delayed

/\*[not(d)]/\*    at the end

## Online Query Evaluation: Requirements



**Match reporting:** should be progressive

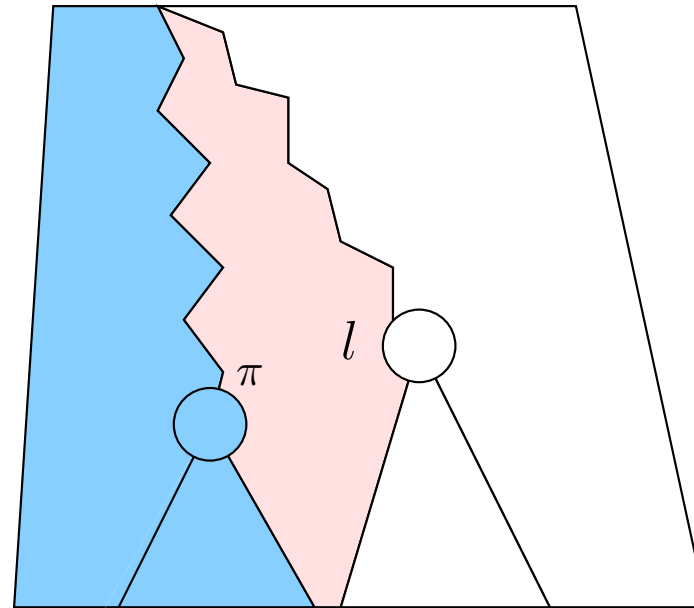
//a/b            immediately

//a[c]/b        delayed

/\*[not(d)]/\*    at the end

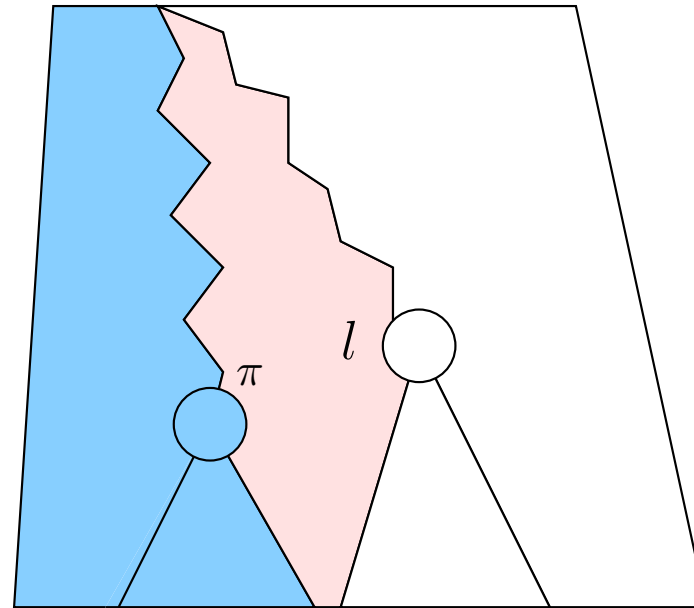
→ ideally, at the earliest possible time

## Earliest detection location



- $l =$  early detection location for  $\pi$   
iff  $\pi = \text{match}$  whatever follows after  $l$
- $l =$  earliest detection location for  $\pi$   
iff  $l =$  first early detection location in document order

## Earliest detection location



- $l =$  early detection location for  $\pi$   
iff  $\pi = \text{match}$  whatever follows after  $l$
- $l =$  earliest detection location for  $\pi$   
iff  $l =$  first early detection location in document order

Obs.:

For some matches (e.g. for `/*[not(d)]/*`) there is no earliest detection location.

# Overview

1. Online query evaluation
2. Regular tree queries (RTQs)
3. Online evaluation of RTQs

## (Extended) Regular Tree Grammars

▷ DTD-look

sample.dtd:

```
<!ELEMENT BOOK (TITLE, SUBTITLE?, CHAPTER+, APPENDIX?)>  
<!ELEMENT CHAPTER (TITLE, (CHAPTER|PAR)+)>  
<!ELEMENT APPENDIX (CHAPTER+)>
```

Start element:

```
<!DOCTYPE BOOK SYSTEM "sample.dtd">
```

▷ Grammar look:

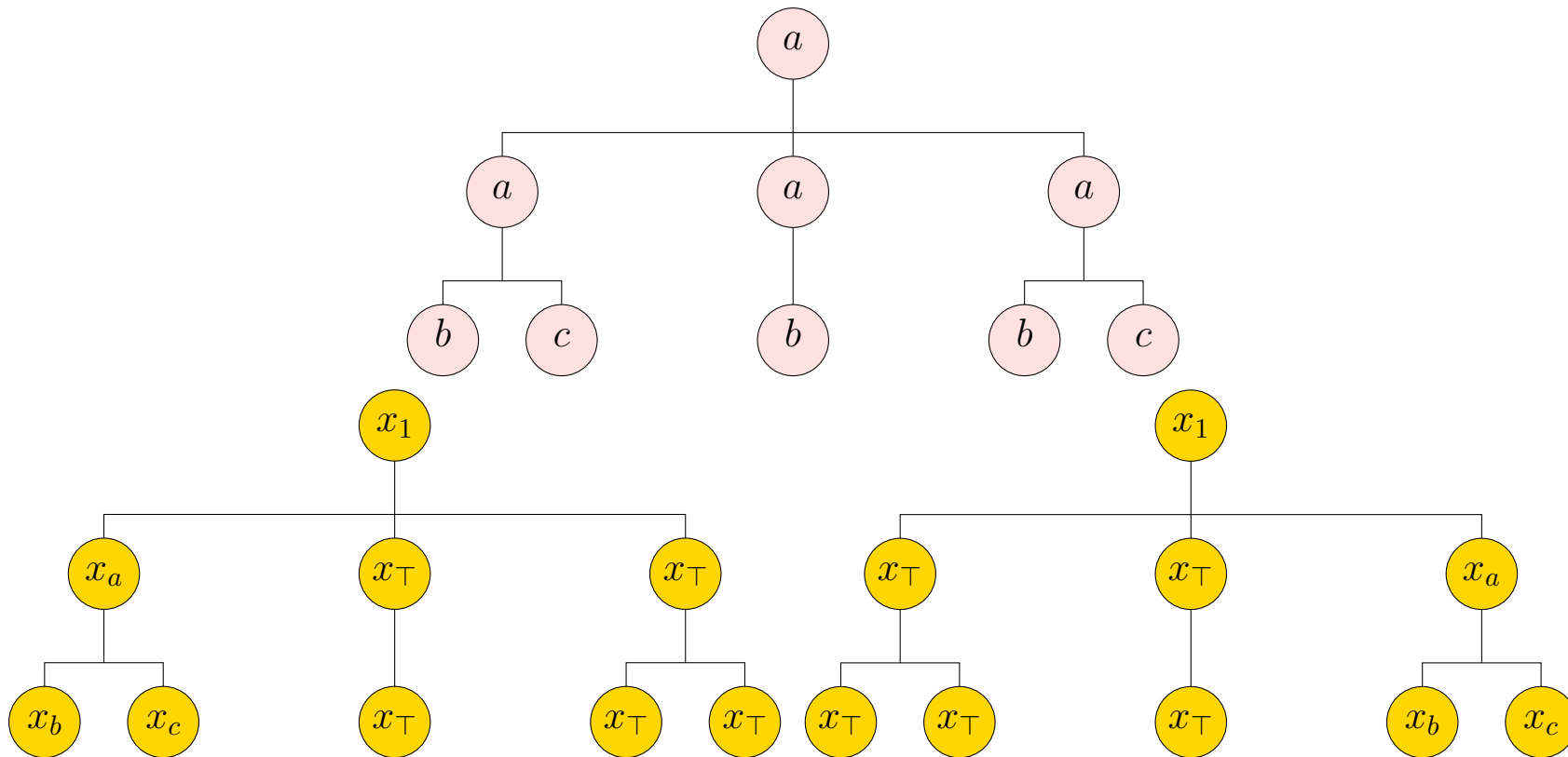
```
 $x_{book}$  → BOOK $\langle x_{title}, x_{subtitle}^?, x_{chapter}^+, x_{appendix}^? \rangle$   
 $x_{chapter}$  → CHAPTER $\langle x_{title}, (x_{chapter}|x_{par})^+ \rangle$   
 $x_{appendix}$  → APPENDIX $\langle x_{chapter}^+ \rangle$ 
```

Start element:  $x_{book}$

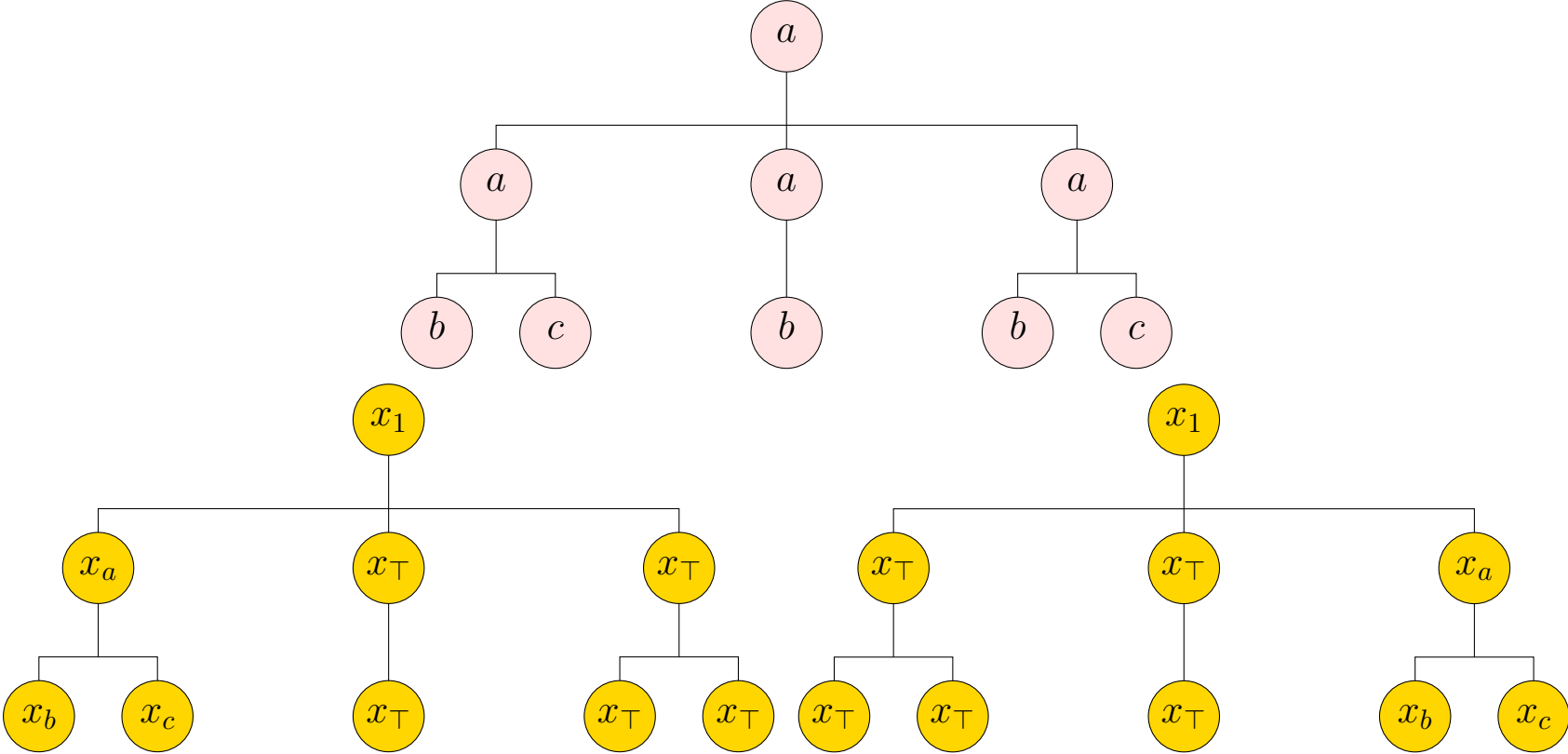
## Grammars specify possible derivations

Productions:  $x_{\top} \rightarrow \mathbf{a}\langle x_{\top}^* \rangle$        $x_1 \rightarrow \mathbf{a}\langle x_{\top}^*, (x_1|x_a), x_{\top}^* \rangle$   
 $x_{\top} \rightarrow \mathbf{b}\langle x_{\top}^* \rangle$        $x_a \rightarrow \mathbf{a}\langle x_b, x_c \rangle$   
 $x_{\top} \rightarrow \mathbf{c}\langle x_{\top}^* \rangle$        $x_b \rightarrow \mathbf{b}\langle x_{\top}^* \rangle$   
 $x_c \rightarrow \mathbf{c}\langle x_{\top}^* \rangle$

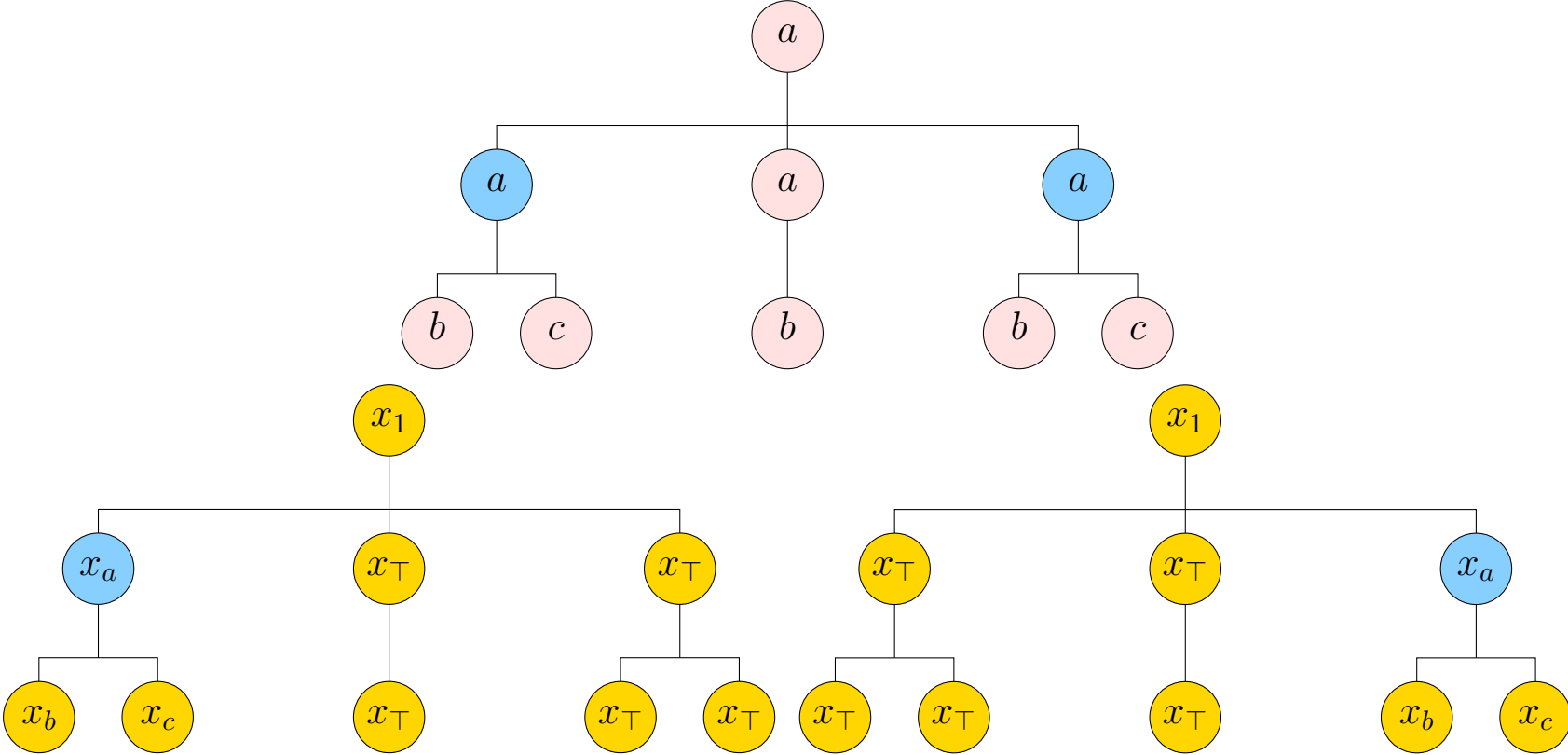
Start:  $x_1$



Non-terminals specify queries

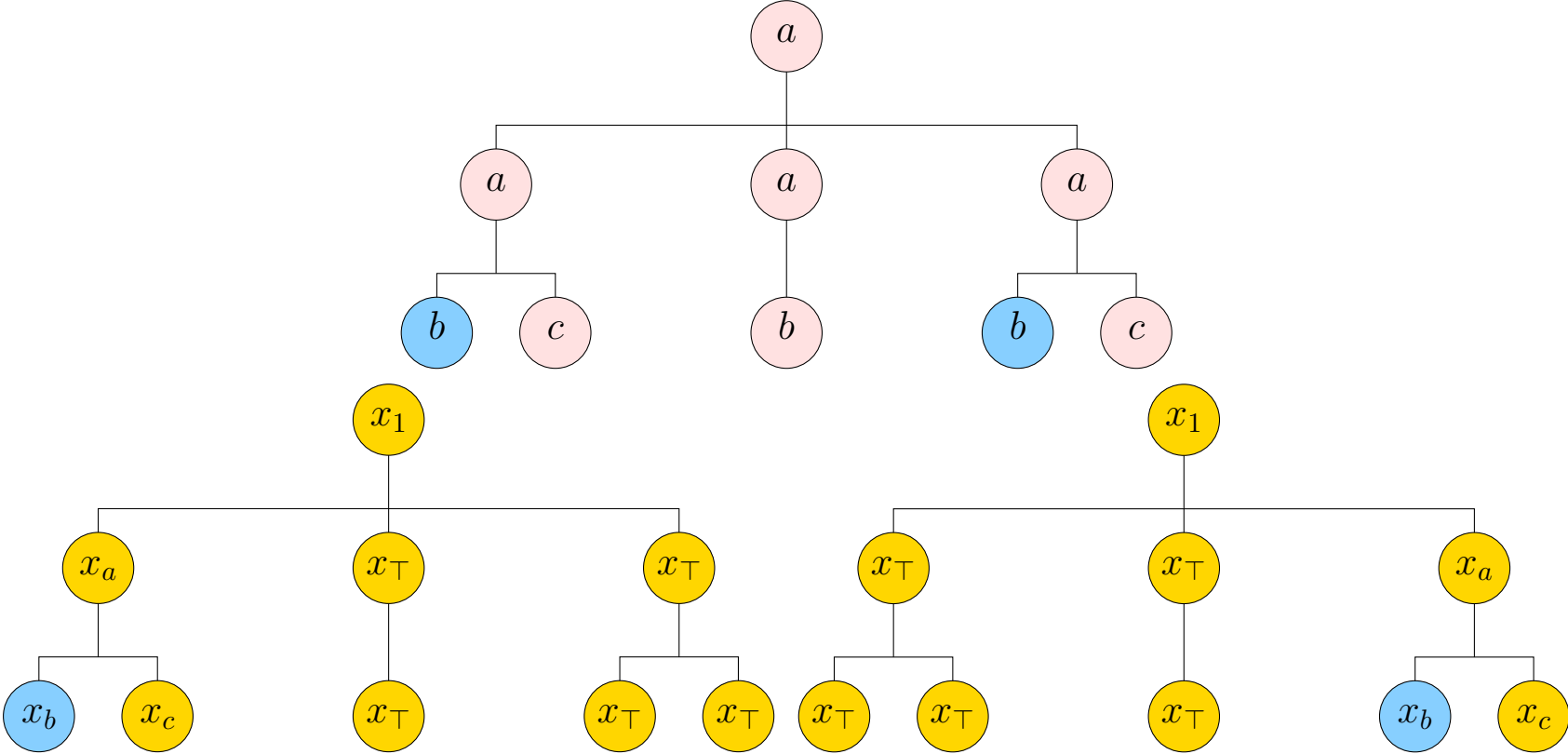


Non-terminals specify queries



◇  $(G, x_a)$

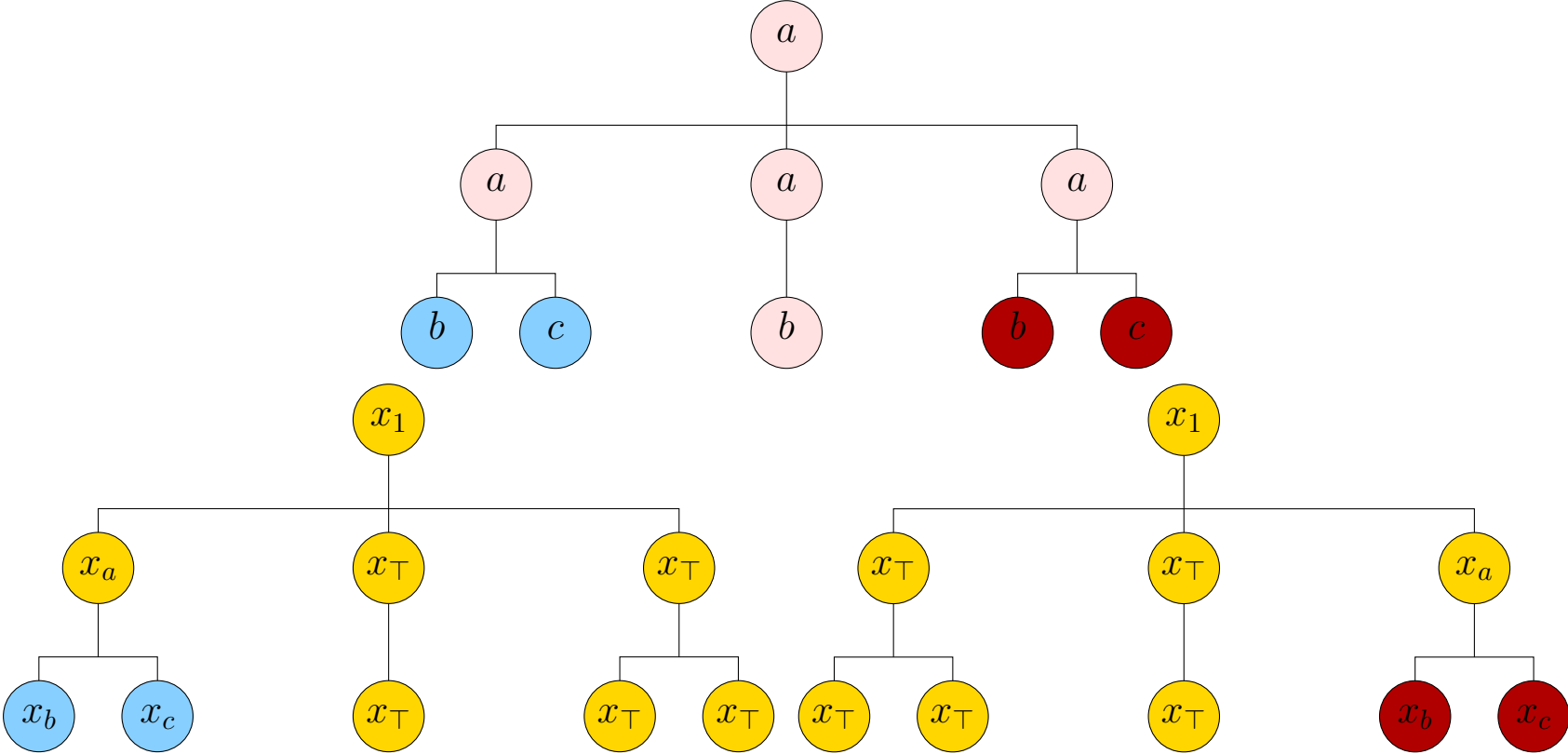
Non-terminals specify queries



◇  $(G, x_a)$

◇  $(G, x_b)$

Non-terminals specify queries



- ◇  $(G, x_a)$
- ◇  $(G, x_b)$
- ◇  $(G, (x_b, x_c))$

## RTQs' vs. XPath's expressiveness

- $RTQs \cap XPath = CoreXPath$
- $XPath \setminus RTQs$  :
  - data value comparisons `//BOOK[TITLE=AUTHOR]`
  - arithmic predicates `//BOOK[count(TITLE)=count(AUTHOR)]`
- $RTQs \setminus XPath$  : precise context specification
  - horizontal context `//BOOK[TITLE (AUTHOR|EDITOR)+]/PRICE`
  - vertical context `(SECTION/)+ TITLE`
- Equally expressive: Selecting tree automata, MSO queries

# Overview

1. Online query evaluation
2. Regular tree queries (RTQs)
3. Online evaluation of RTQs

## Binary Encoding

For clarity we use a first-child next-sibling encoding, which preserves the document order.

Binary Grammar  $G = (X, P, x_0)$

- Non-terminals  $X$
- Productions  $P$

Node productions  $x \rightarrow a\langle x_1, x_2 \rangle$

Leaf productions  $x \rightarrow \lambda$

**Wildcard productions:**  $x \rightarrow *\langle x_1, x_2 \rangle$

- to be able to generate all (not a priori known) Universe:

$x \in Univ_G$  iff  $x \rightarrow \lambda$ ,  $x \rightarrow *\langle x_1, x_2 \rangle$  and  $x_1, x_2 \in Univ_G$

Note:  $Univ_G$  can be computed in **linear** time

- for convenient specification of queries

- Start non-terminal  $x_0$

## Example

Productions:

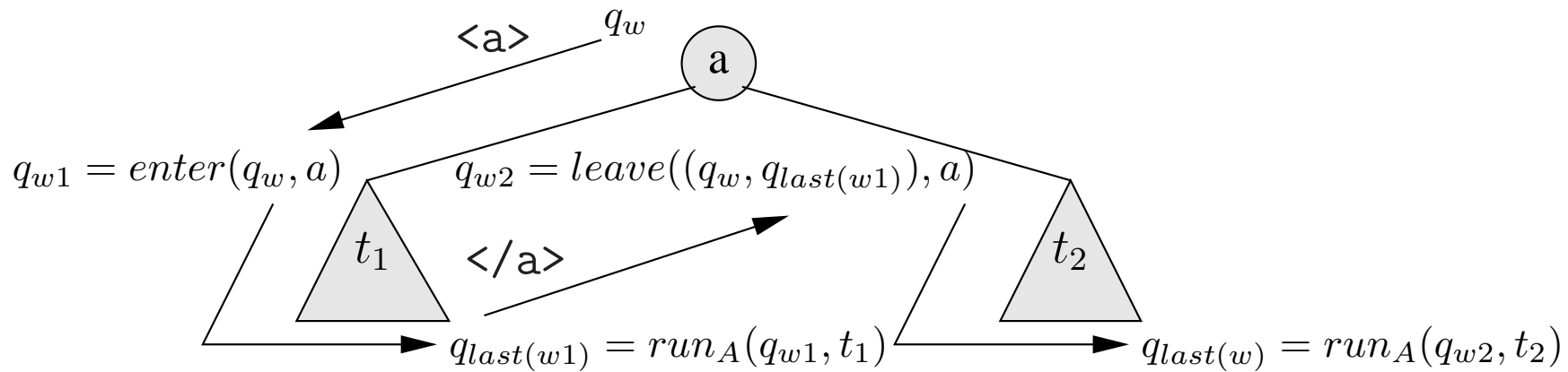
$$\begin{array}{ll} x_{\top} \rightarrow * \langle x_{\top}, x_{\top} \rangle & (1) & x_0 \rightarrow a \langle x_b, x_{\top} \rangle & (5) \\ x_{\top} \rightarrow \lambda & (2) & x_0 \rightarrow * \langle x_0, x_{\top} \rangle & (6) \\ x_c \rightarrow c \langle x_{\top}, x_{\top} \rangle & (3) & x_0 \rightarrow * \langle x_{\top}, x_0 \rangle & (7) \\ x_b \rightarrow b \langle x_{\top}, x_c \rangle & (4) & & \end{array}$$

Start:  $x_0$

Queries:

$$\begin{array}{ll} x_b & \equiv //a/b[\text{following-sibling}::c] \\ x_c & \equiv //a/b[\text{preceding-sibling}::c] \\ x_0 & \equiv //a/b[\text{following-sibling}::c]/\text{ancestor}::* \end{array}$$

**Pre-order automaton (POA)**  $A = (Q, enter, leave, F, q_0)$



- Run:

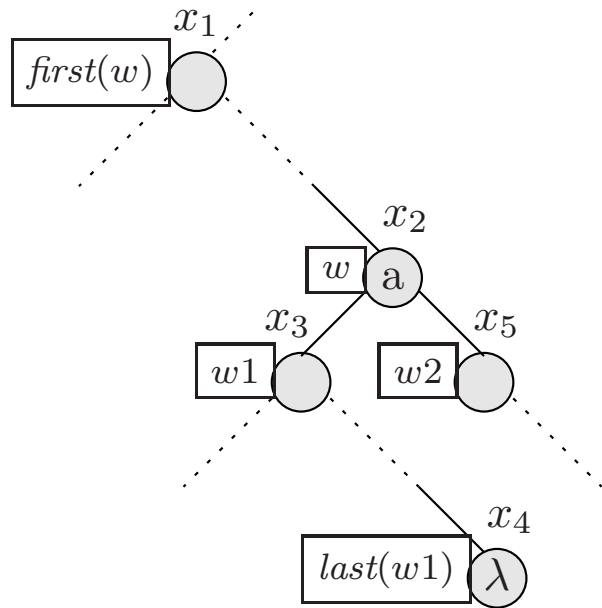
$$run_A(q, t) = \begin{cases} q & , \text{ if } t = \lambda \\ run_A(leave((q, run_A(enter(q, a), t_1)), a), t_2), & \text{ if } t = a\langle t_1, t_2 \rangle \end{cases}$$

- Accepted language:

$$\mathcal{L}_A = \{t \in \mathcal{T}_\Sigma \mid run_A(q_0, t) \in F\}$$

## POA for validation

Given  $G = (X, P, x_0)$ , construct POA  $A_G = (X^2, enter, leave, F, q_\epsilon)$  s.t.  $\mathcal{L}_G = \mathcal{L}_{A_G}$ .



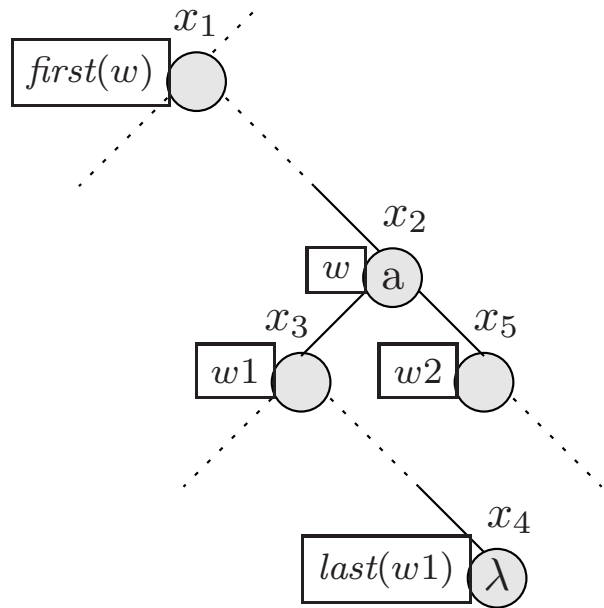
$$q_\epsilon = \{ x_0 \}$$

$$enter(q_w, a) = \{ x_3 \mid x_2 \in q_w, x_2 \rightarrow (a \mid *) \langle x_3, x_5 \rangle \}$$

$$leave((q_w, q_{last(w1)}), a) = \\ \{ x_5 \mid x_2 \in q_w, x_4 \in q_{last(w1)}, x_4 \rightarrow \lambda, \\ x_2 \rightarrow (a \mid *) \langle x_3, x_5 \rangle \}$$

## POA for validation

Given  $G = (X, P, x_0)$ , construct POA  $A_G = (X^2, enter, leave, F, q_\epsilon)$  s.t.  $\mathcal{L}_G = \mathcal{L}_{A_G}$ .



$$q_\epsilon = \{(x_0, \mathbf{x}_0)\}$$

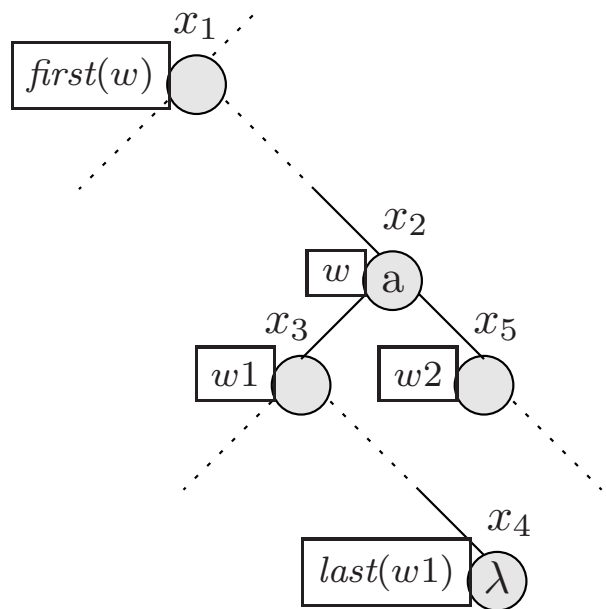
$$enter(q_w, a) = \{(x_3, \mathbf{x}_3) \mid (x_2, \mathbf{x}_1) \in q_w, x_2 \rightarrow (a \mid *) \langle x_3, x_5 \rangle\}$$

$$leave((q_w, q_{last(w1)}), a) =$$

$$\{(x_5, \mathbf{x}_1) \mid (x_2, \mathbf{x}_1) \in q_w, (x_4, \mathbf{x}_3) \in q_{last(w1)}, x_4 \rightarrow \lambda, \\ x_2 \rightarrow (a \mid *) \langle x_3, x_5 \rangle\}$$

## POA for validation

Given  $G = (X, P, x_0)$ , construct POA  $A_G = (X^2, enter, leave, F, q_\epsilon)$  s.t.  $\mathcal{L}_G = \mathcal{L}_{A_G}$ .



$$q_\epsilon = \{(x_0, x_0)\}$$

$$enter(q_w, a) = \{(x_3, x_3) \mid (x_2, x_1) \in q_w, x_2 \rightarrow (a \mid *) \langle x_3, x_5 \rangle\}$$

$$leave((q_w, q_{last(w1)}), a) = \\ \{(x_5, x_1) \mid (x_2, x_1) \in q_w, (x_4, x_3) \in q_{last(w1)}, x_4 \rightarrow \lambda, \\ x_2 \rightarrow (a \mid *) \langle x_3, x_5 \rangle\}$$

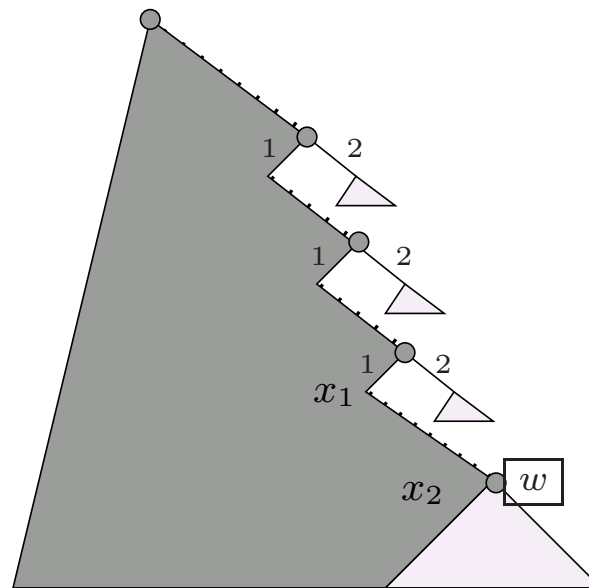
$$F = \{q \in Q \mid (x, x_0) \in q, x \rightarrow \lambda\}$$

## POA for query evaluation

Idea: associate  $(x_2, x_1)$  with

1. *compatible potential matches*  $M$
2. information on the *irrelevance of the upper right context*  $b$

$M$  and  $b$  can be computed locally, along the transitions

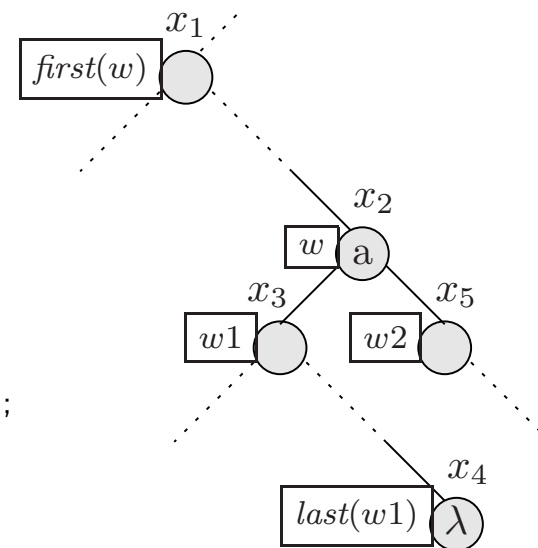


**Th.:**  $w$  is an early detection location for some match node  $w_1$  iff  
 $q_w(x_2, x_1) \mapsto (M, true)$ ,  $x_2 \in Univ_G$  and  $w_1 \in M$  for some  $x_2$  and  $x_1$ .

## Implementation

- Demand-driven construction of the automaton for a query  $(G, x_{\odot})$

```
1 startDocHandler() {  $q := \{(x_0, x_0) \mapsto (\emptyset, true)\};$  }
2
3 enterNodeHandler(Node  $w$ , Label  $a$ ) {
4    $s.push(q);$ 
5
6    $q_1 = \emptyset$ 
7   for  $(x_2, x_1) \in dom(q)$  with  $q(x_2, x_1) = (M, b)$ 
8     for  $x_2 \rightarrow (a | *) \langle x_3, x_5 \rangle$ 
9       if  $x_2 = x_{\odot}$  then  $M := M \cup \{w\};$ 
10       $b_1 := b \ \& \ (x_5 \in Univ_G);$ 
11      if  $b_1 \ \& \ (x_3 \in Univ_G)$  then reportMatches( $M$ );
12       $q_1 := q_1 \oplus [(x_3, x_3) \mapsto (b_1 \ \& \ (x_3 \notin Univ_G) ? M : \emptyset, b_1)];$ 
13
14    $q := q_1;$  }
```

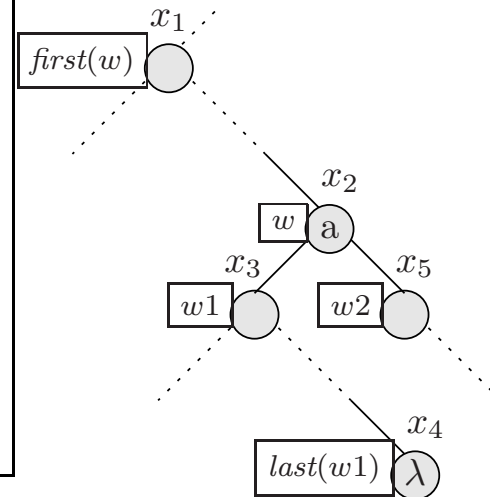


## Implementation

```

1  leaveNodeHandler(Node w, Label a){
2    qfather = s.pop();
3
4    q1 = ∅
5    for (x4, x3) ∈ dom(q) with q(x4, x3) = (M1, b1) ∧ x4 → λ
6      if b1 then reportMatches(M1);
7      for (x2, x1) ∈ dom(qfather) with qfather(x2, x1) = (M, b) ∧
8        x2 → (a | *)⟨x3, x5⟩
9        if (x2 = x⊙) & !b1 then M := M ∪ {w};
10       q1 := q1 ⊕ [(x5, x1) ↦ (b1 ? M : (M1 ∪ M), b)];
11
12     q := q1; }
13
14  endDocHandler() {
15    for (x, x0) ∈ dom(q) with q(x, x0) = (M, true) and x → λ
16      reportMatches(M) }

```



## Implementation

- True matches are reported as soon as possible.
- False matches are discarded as soon as possible.
- Algorithm is orthogonal w.r.t. universality test used.
- Has been implemented in Fxgrep.

## Complexity

Let

$|D|$  = size of the input data

$|X|$  = number of non-terminals (approx. the number of nodes referred by the query)

$|P|$  = number of productions (approx. the number of all node qualifiers)

$pot_{max}$  = maximum number of potential match nodes

$dom_{max}$  = the maximal size of the domain of all states ( $\leq |X|^2$ )

$d$  = maximal depth

Time complexity:  $O(|D| \cdot dom_{max}^2 \cdot |P| \cdot pot_{max})$

Space complexity:  $O(d \cdot dom_{max}^2 \cdot pot_{max})$