

Web Data Models

XPath: Evaluation

Silviu Maniu



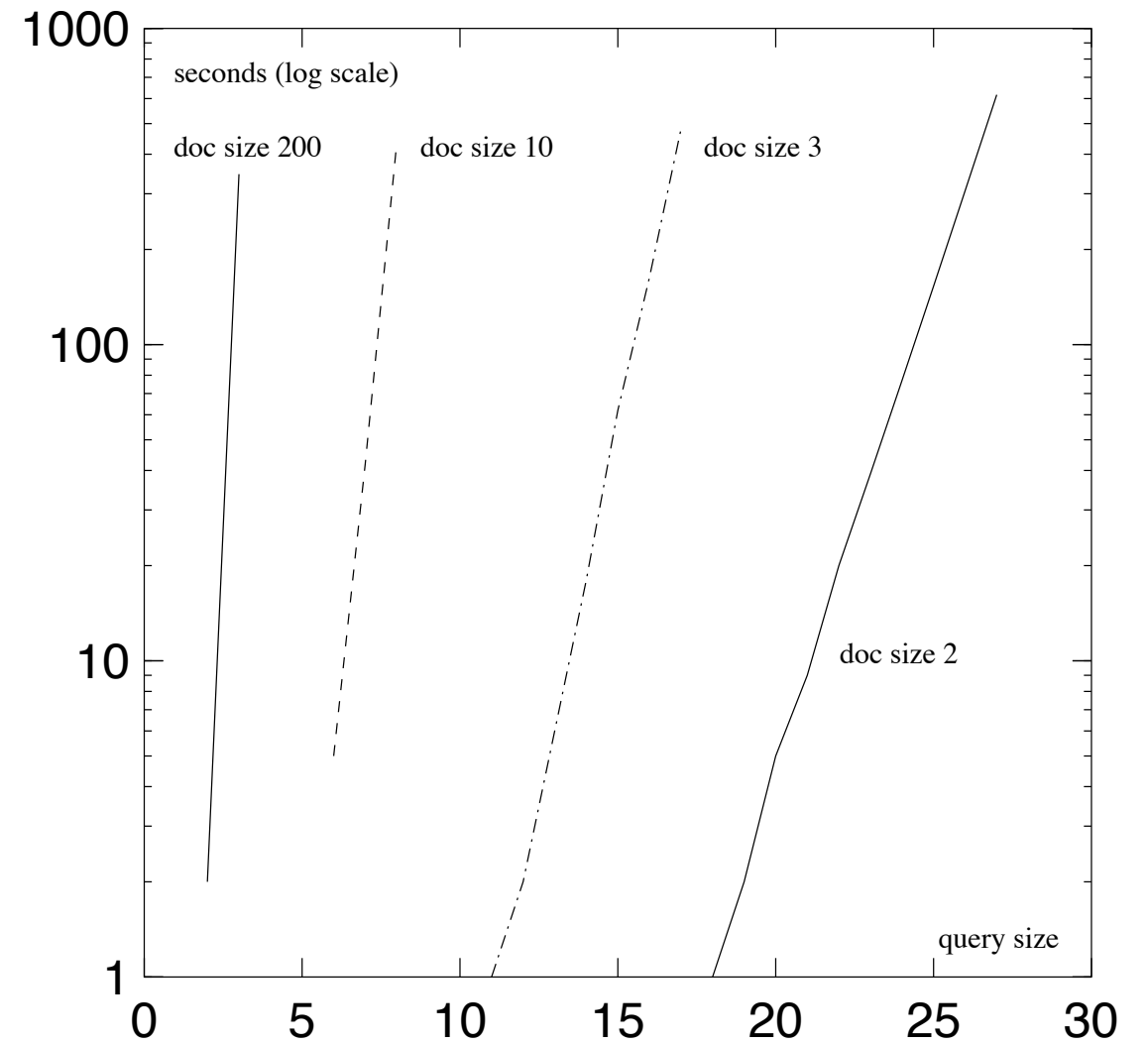
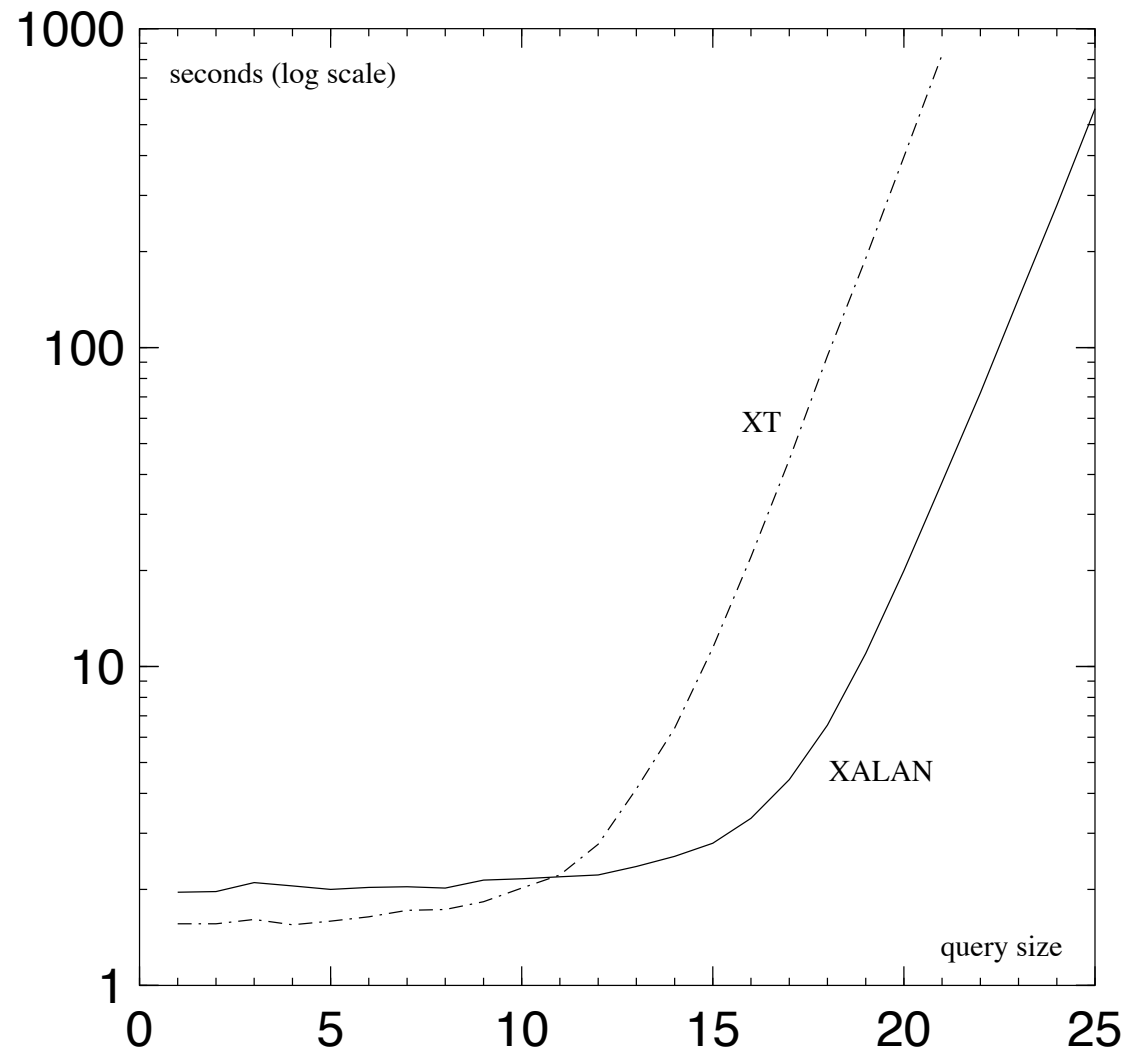
Comprendre le monde,
construire l'avenir



XPath: Performance

- XPath is a **navigational language** — specifies how the XML documents should be **traversed**
- **Main issue:** big volume of nodes can be extracted via XPath, so **efficient processing is still an ongoing challenge**

XPath: Performance

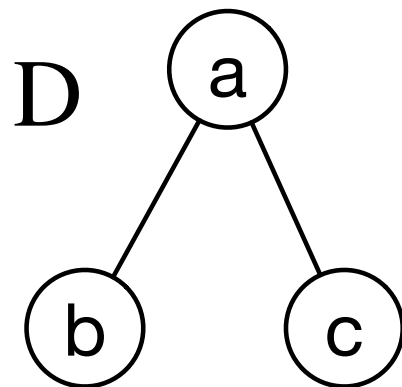


Gottlob, Koch, Pichler "Efficient Algorithms for Processing XPath Queries", VLDB 2002

XPath: Performance

Why?

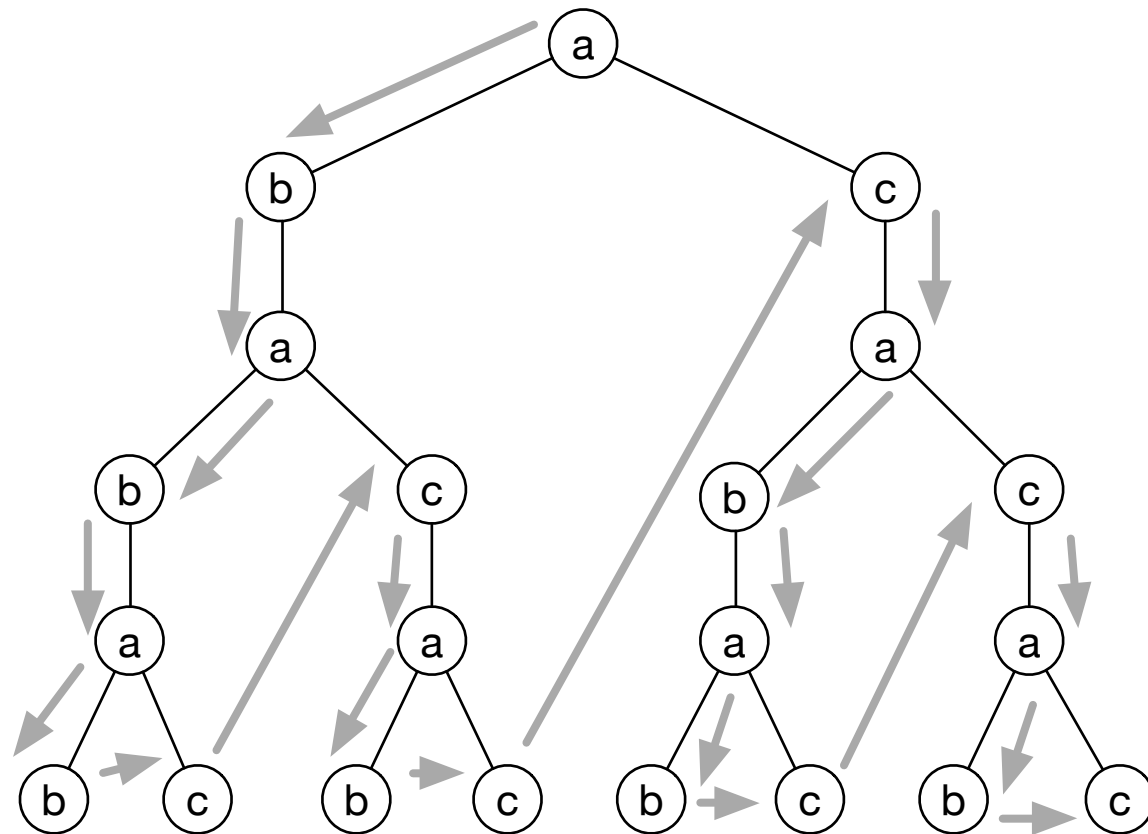
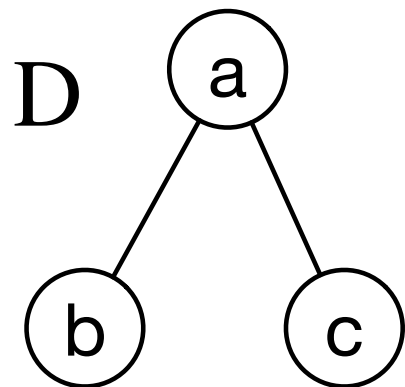
`Q := child::* / parent::* / child::* / parent::* / child::*`



XPath: Performance

Why?

$Q := \text{child}::*/\text{parent}::*/\text{child}::*/\text{parent}::*/\text{child}::*$



$$O(|D|^{|Q|})$$

XPath: Performance

Why?

```
procedure process-location-step( $n_0$ ,  $Q$ )  
  /*  $n_0$  is the context node;  
   query  $Q$  is a list of location steps */  
begin  
  node set  $S :=$  apply  $Q$ .first to node  $n_0$ ;  
  if ( $Q$ .tail is not empty) then  
    for each node  $n \in S$  do  
      process-location-step( $n$ ,  $Q$ .tail);  
     $O(|D|^{Q|})$   
end
```

Lecture Outline

- evaluating simple paths
- evaluating Core XPath
- evaluating Full XPath

XPath: Simple Paths

- Simple paths are of the form:

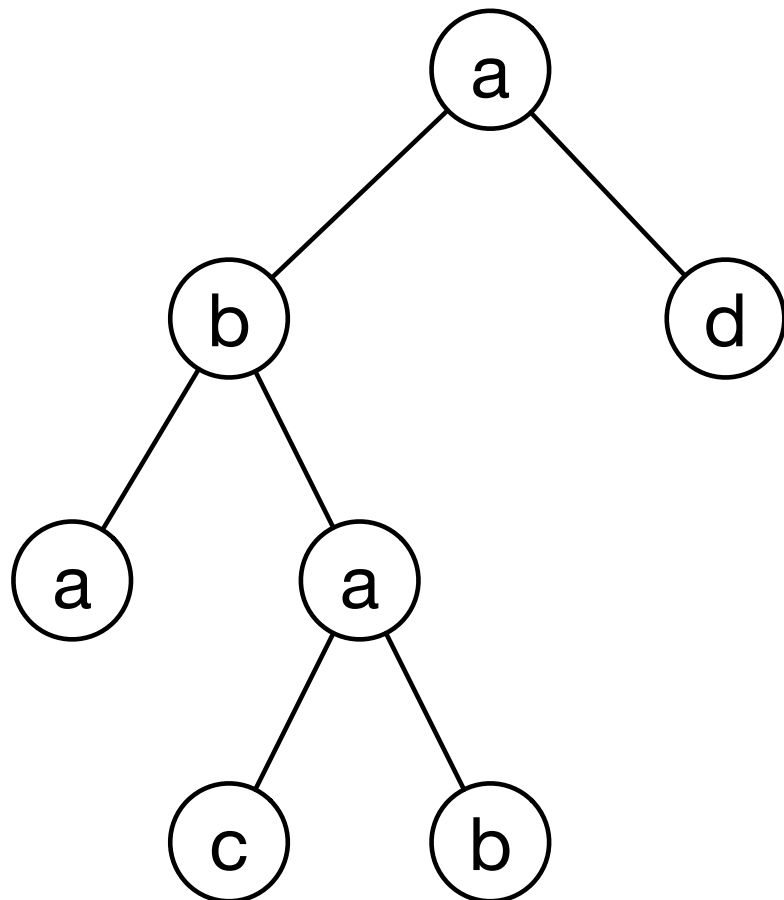
`//tag_1/tag_2/.../tag_n`

`//tag_1/tag_2/.../tag_n-1/text()`

- Can be evaluated in a single pre-order traversal (by using a stack)

XPath: Simple Paths

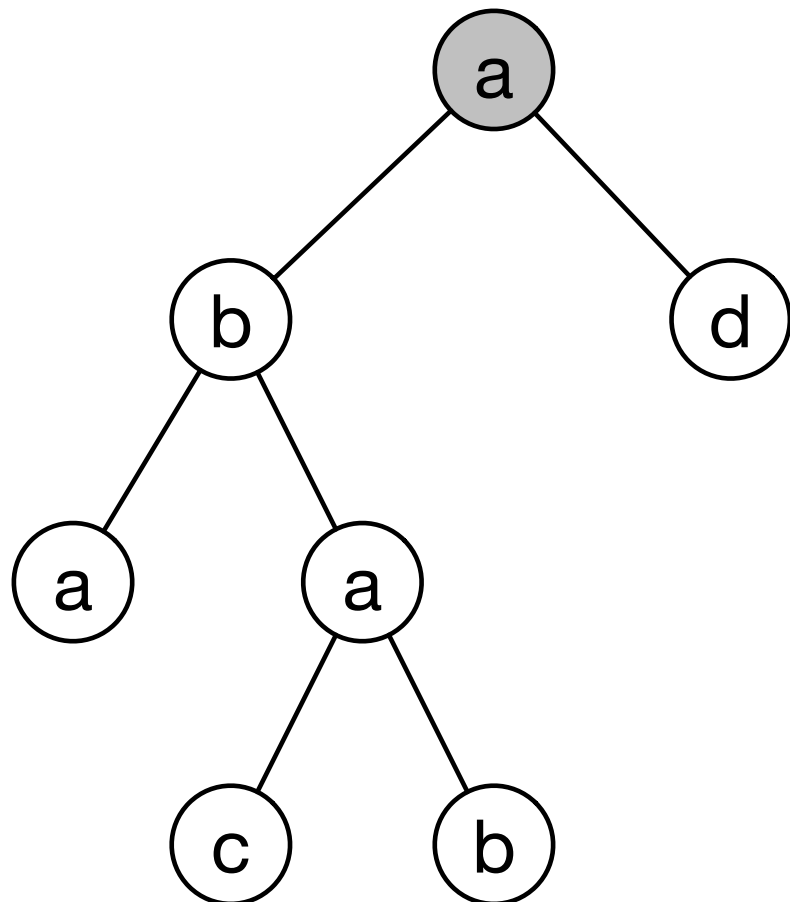
Q: *//a/b*



XPath: Simple Paths

Q: **//a/b**

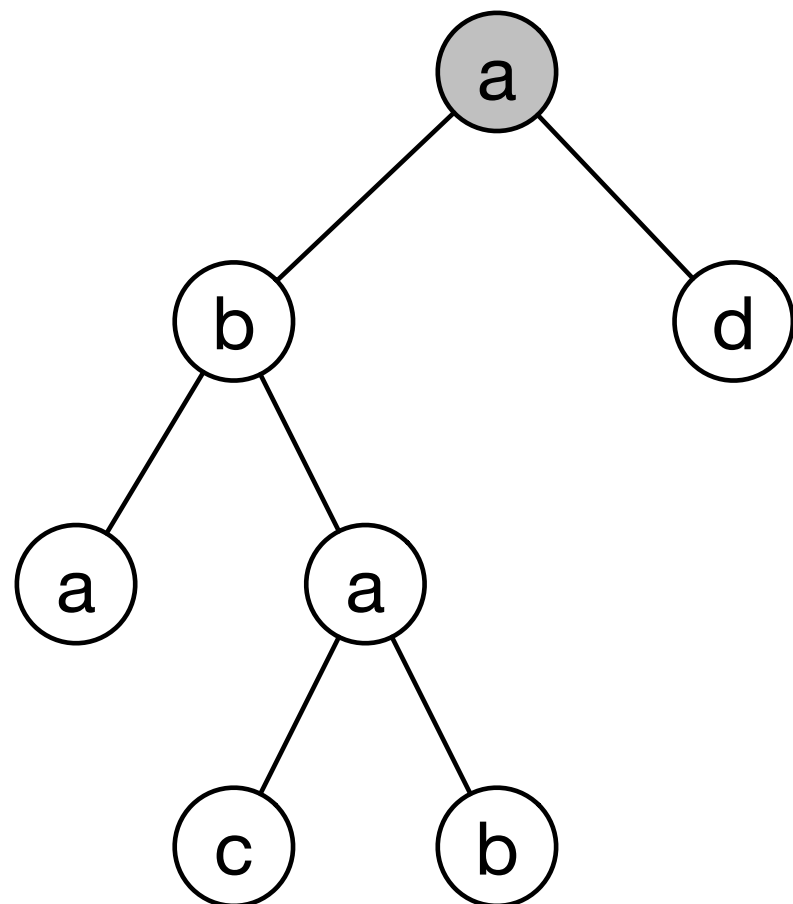
p = 1



XPath: Simple Paths

Q: **//a/b**

$p = p+1 = 2$



Seq

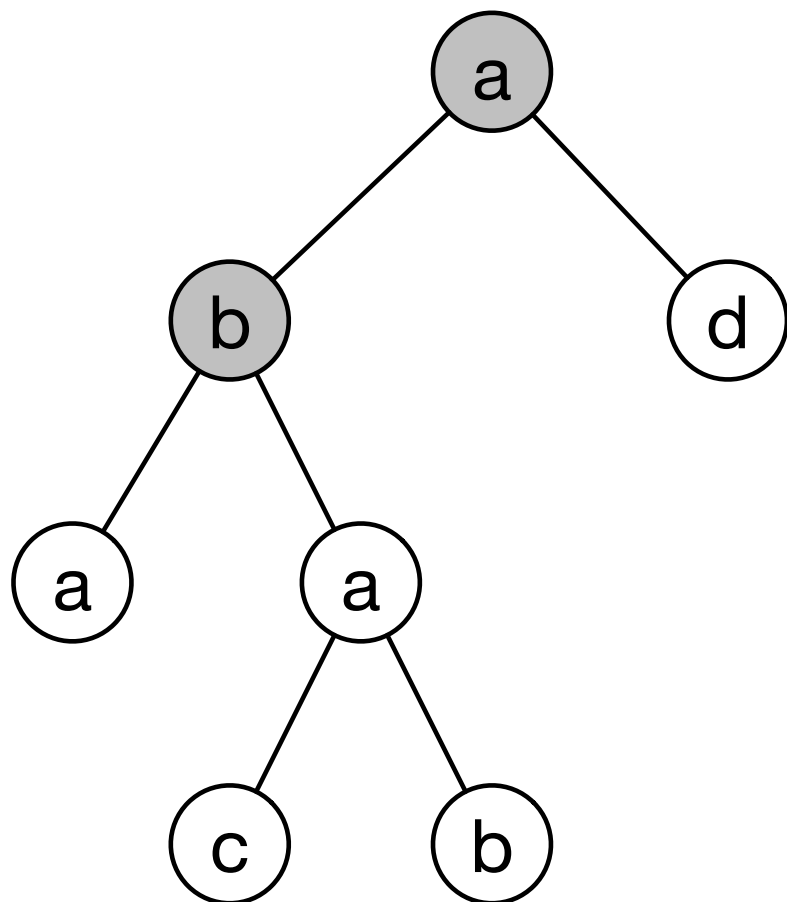
<a>

Stack

XPath: Simple Paths

Q: **//a/b**

p = 2



Seq

<a>

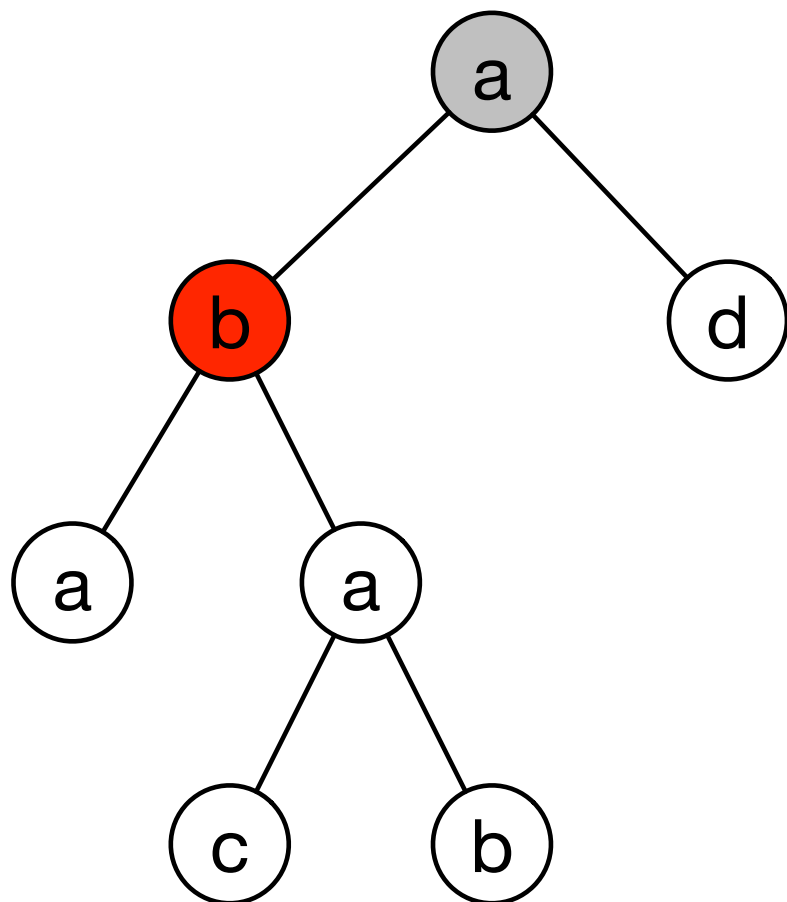
Stack

Match!

XPath: Simple Paths

Q: **//a/b**

p = 1



Seq

<a>

Stack

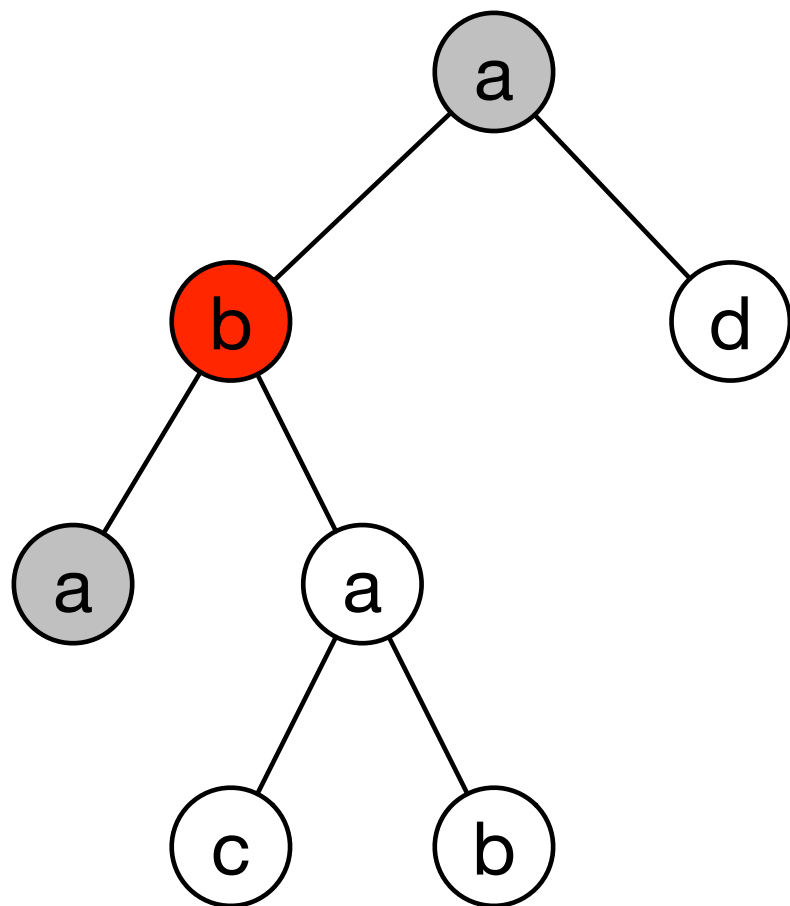
2

Match!

XPath: Simple Paths

Q: **//a/b**

p = 1



Seq

<a>

<a>

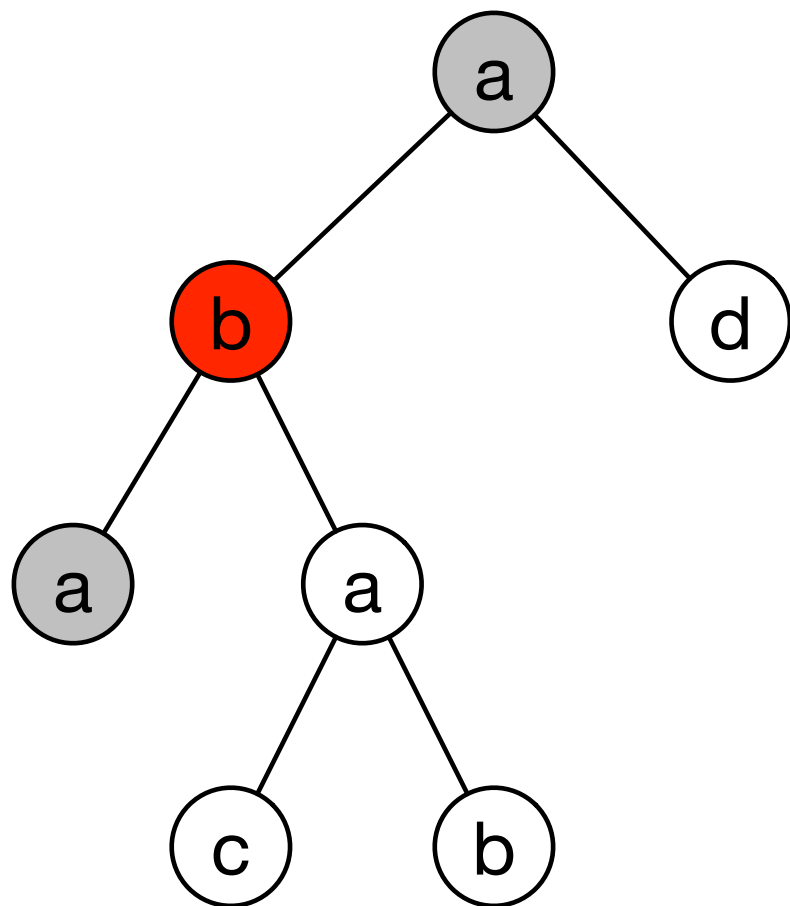
Stack

2

XPath: Simple Paths

Q: **//a/b**

$p = p+1 = 2$

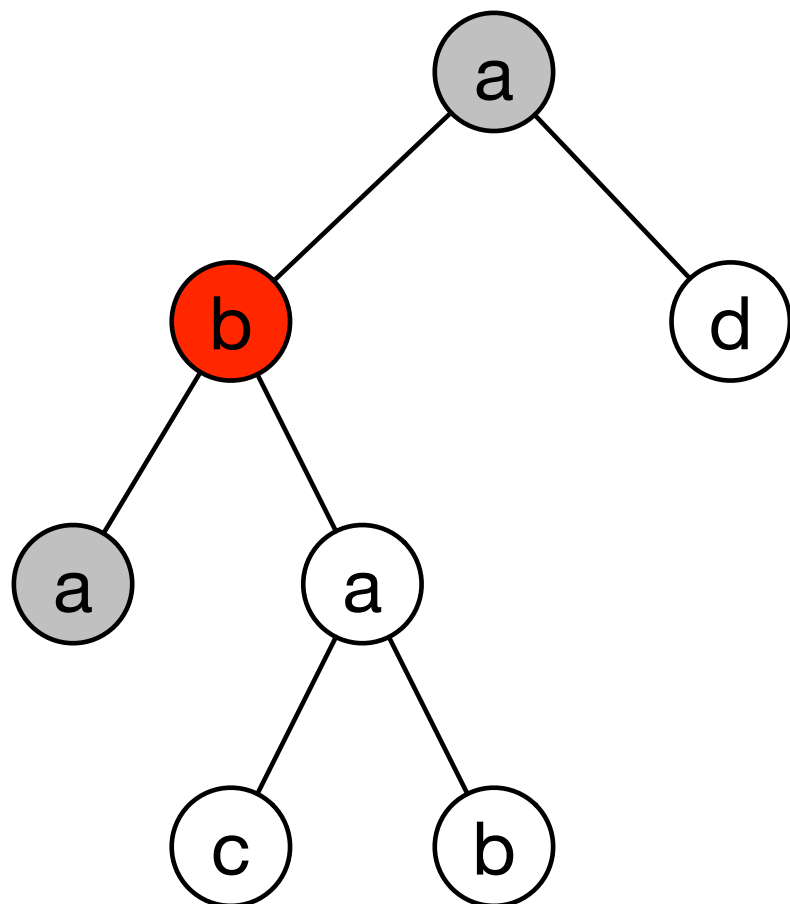


Seq	Stack
<a>	
	
<a>	
	1
	2

XPath: Simple Paths

Q: **//a/b**

p = 2



Seq

<a>

<a>

Stack

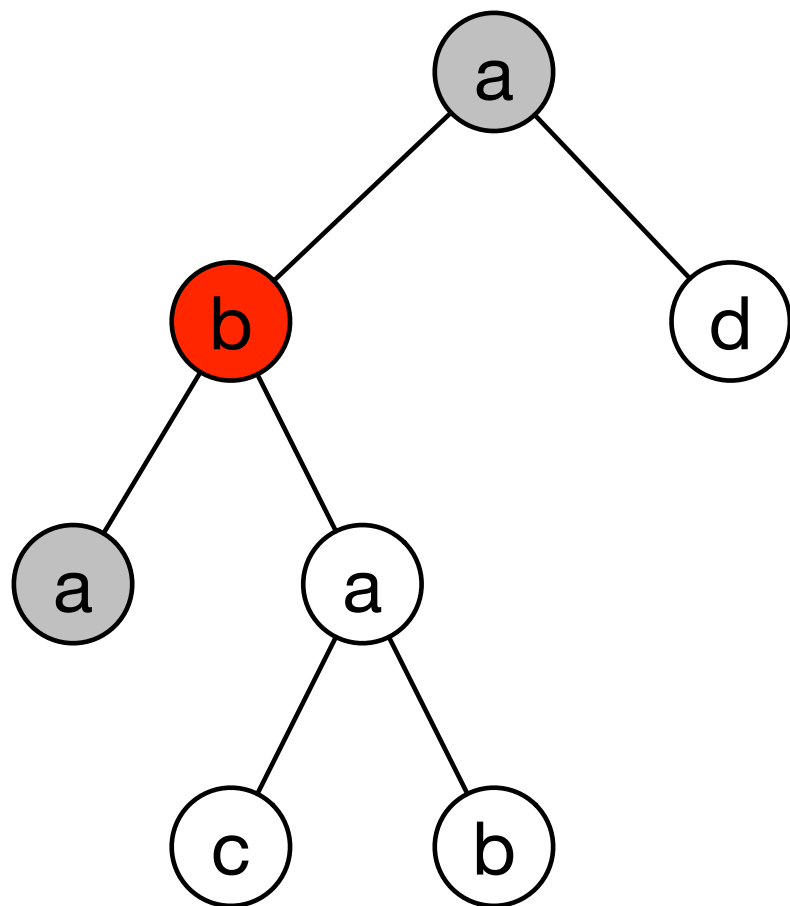
1

2

XPath: Simple Paths

Q: **//a/b**

p = pop() = 1

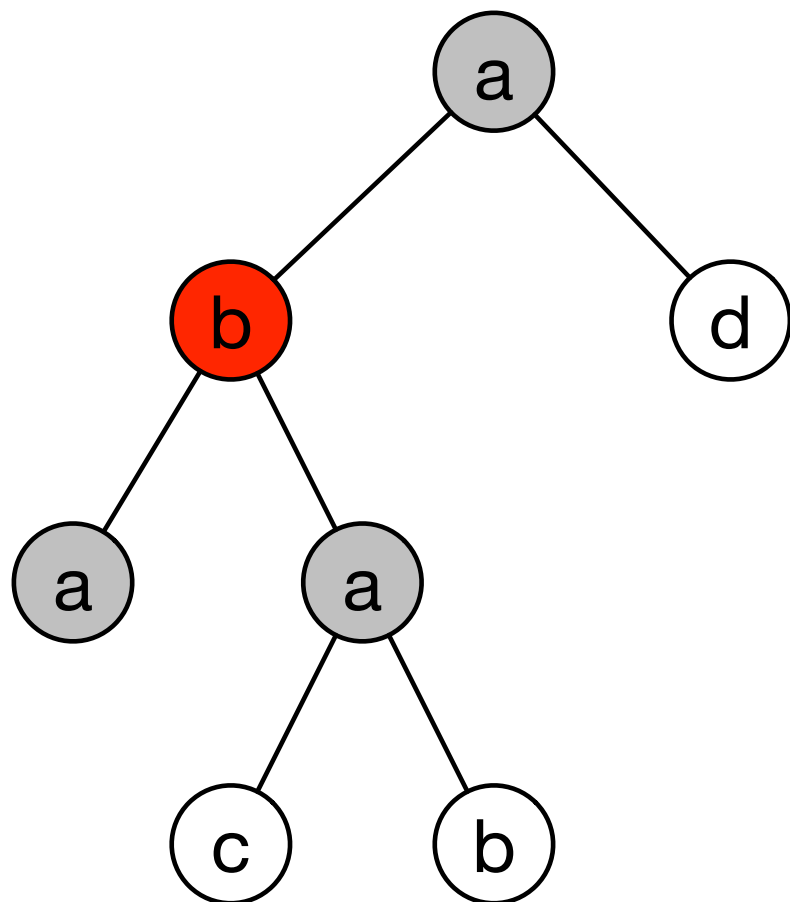


Seq	Stack
<a>	
	
<a>	
	
	2

XPath: Simple Paths

Q: **//a/b**

p = 1



Seq

<a>

<a>

<a>

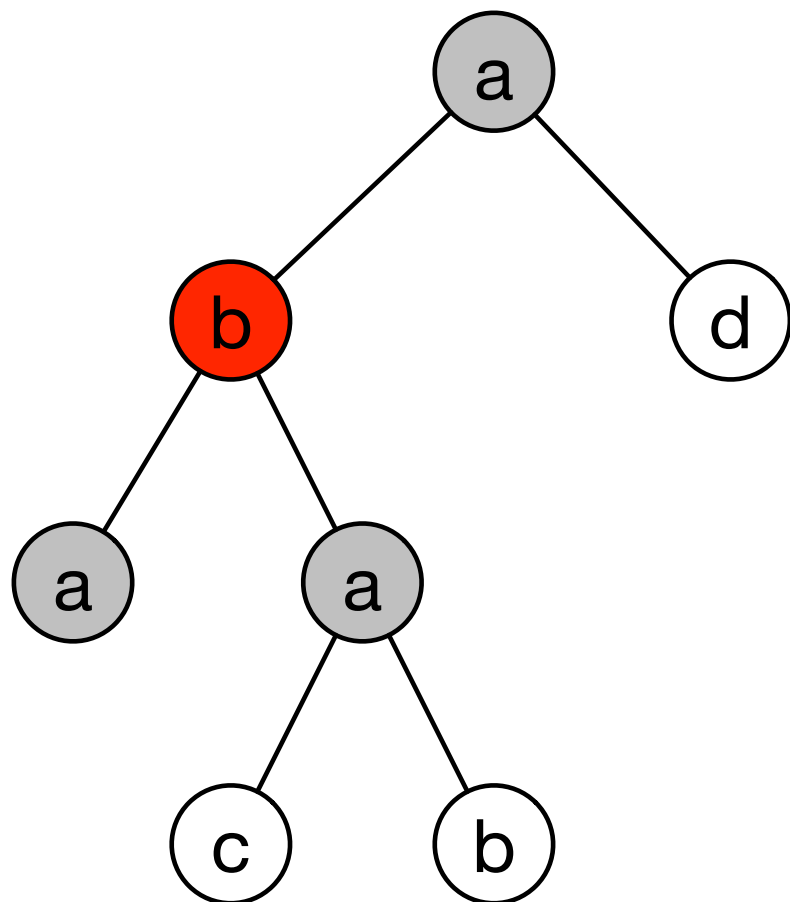
Stack

2

XPath: Simple Paths

Q: **//a/b**

$p = p+1 = 2$

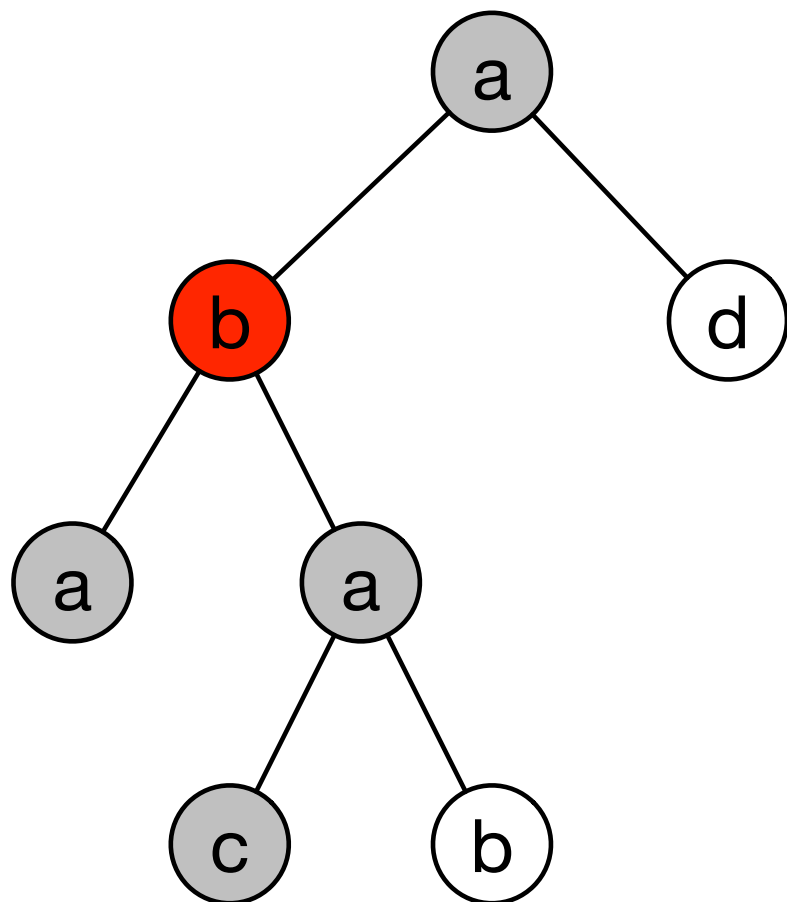


Seq	Stack
<a>	
	
<a>	
	
<a>	
	1
	2

XPath: Simple Paths

Q: **//a/b**

p = 2

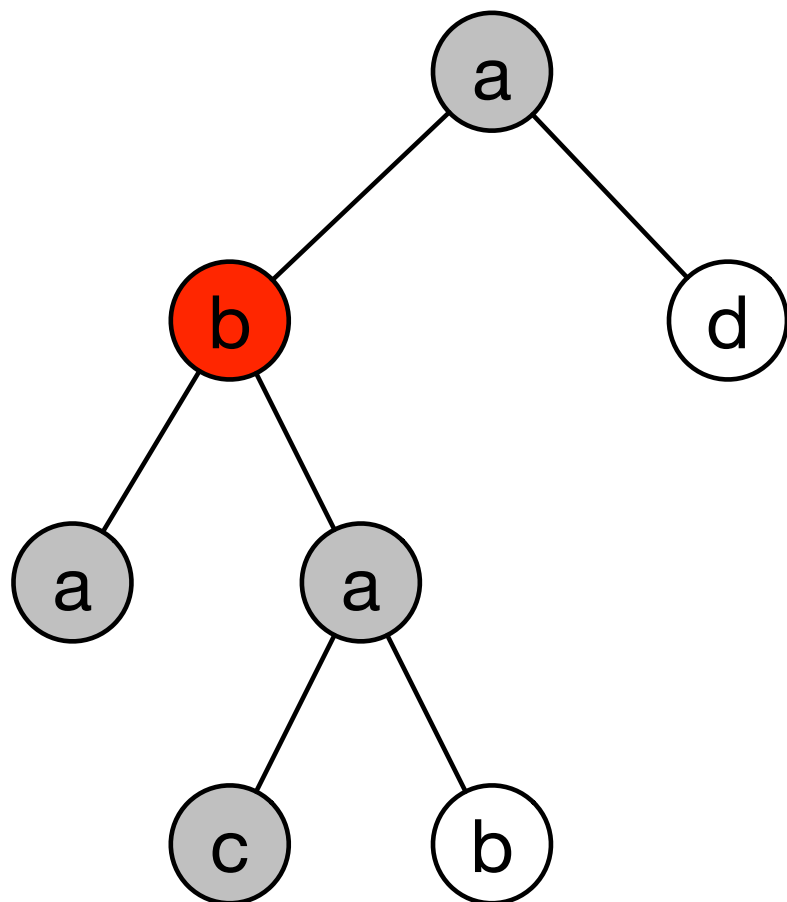


Seq	Stack
<a>	
	
<a>	
	
<a>	
<c>	
	1
	2

XPath: Simple Paths

Q: **//a/b**

$p = \text{pop}() = 1$

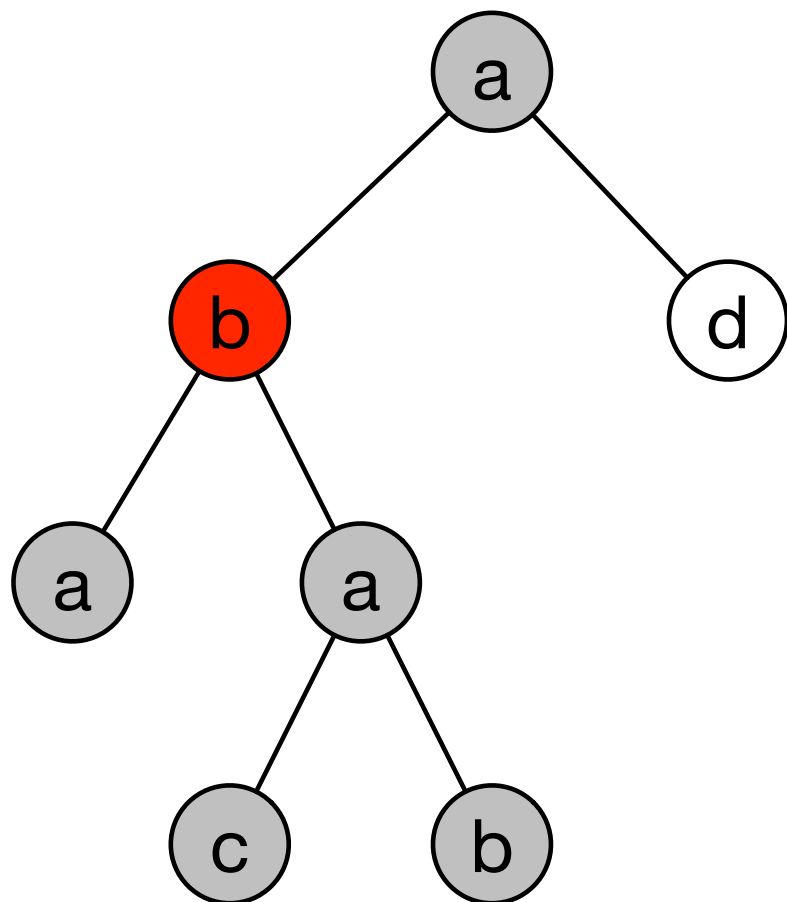


Seq	Stack
<a>	
	
<a>	
	
<a>	
<c>	
</c>	
	2

XPath: Simple Paths

Q: **//a/b**

p = 2



Seq

<a>

<a>

<a>

<c>

</c>

Stack

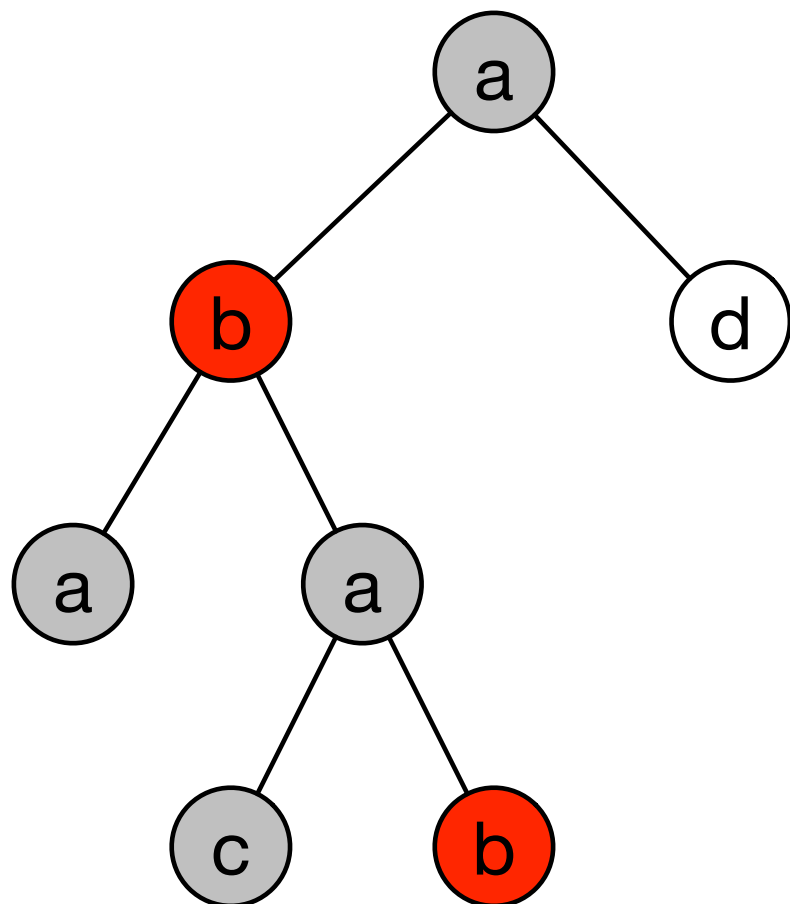
2

Match!

XPath: Simple Paths

Q: **//a/b**

p = 1



Match!

Seq	Stack
<a>	
	
<a>	
	
<a>	
<c>	
</c>	
	2

XPath: Simple Path Evaluation Complexity

- The algorithm is linear in the size of the document $O(|D|)$
- Moreover, it can be implemented as a streaming algorithm
- Simple path evaluation can be implemented on top of SAX (Simple API for XML)

XPath: Simple Path Evaluation In SAX

Algorithm (sketch):

1. **Initialization**: represent *path* query as an array for each step, maintain an array index *i* of the current step in the path, maintain a stack *S* of index positions
2. **startDocument**: empty stack *S*; *i*=1
3. **startElement**: if *path[i]* and element match, proceed to next step; otherwise, make a **failure transition**. Push *i* on *S*.
4. **endElement**: Pop old *i* from *S*.
5. **text**: If *path[i]*=*text*, we found a match. Otherwise, do nothing.

Failure Transitions

Example:

Q: //a/b/a/c/ but we have seen //a/b/a/**b**

- postfix of we have seen is prefix of the query!

Q: // **a/b/a/c/** //a/b/**a/b**

- this can be done via the [Knuth-Morris-Pratt algorithm](#) — linear string matching

XPath: Core XPath

Core XPath contains:

- all 12 axes
- all node tests (only element nodes)
- filters with logical operators: and, or, not

XPath: Bottom-up Evaluation of Core XPath

Set operations on nodes:

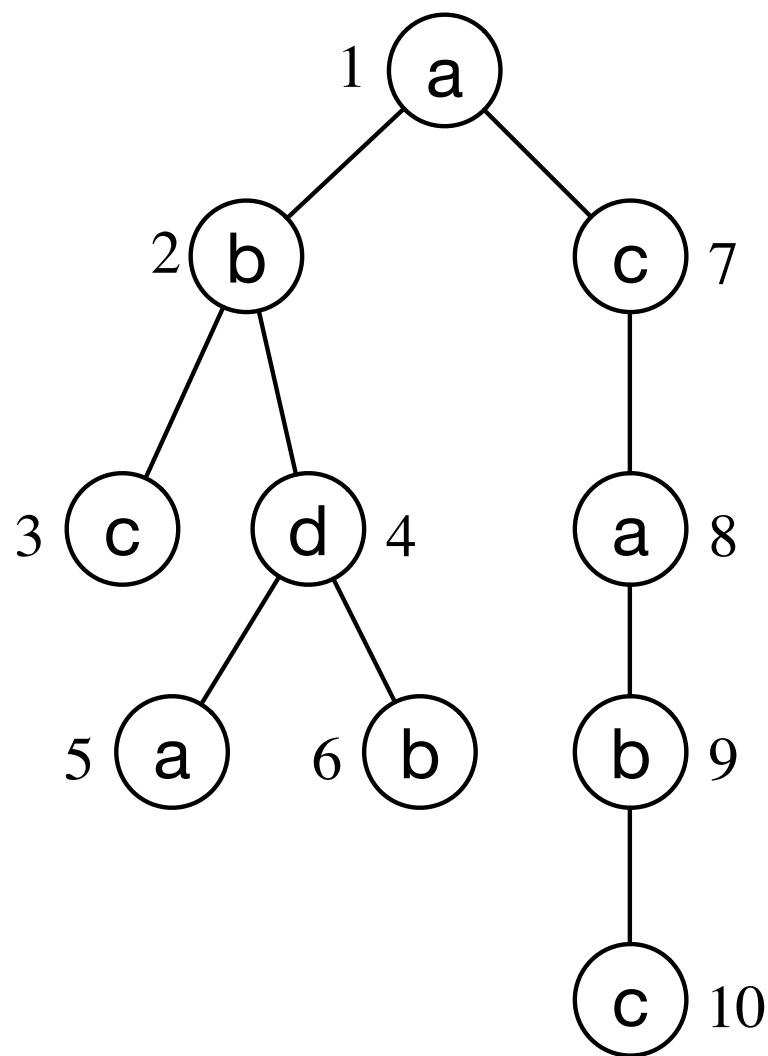
Operation	Objective
$\text{axis}(S_1) = S_2$	the node ids corresponding to the axis axis
$\cap(S_1, S_2) = S_3$	intersection of sets; for steps and and
$\cup(S_1, S_2) = S_3$	union of sets; for or
$-(S_1, S_2) = S_3$	difference of sets; for not
$T(\text{label}) = S_1$	set of node ids labelled label

XPath: Bottom-up Evaluation of Core XPath

Algorithm (sketch):

1. Transform the query into a tree composed of set operations;
2. Starting at the root (or at the filters); evaluate the set operations bottom-up;
3. The final results are the nodes corresponding to node ids.

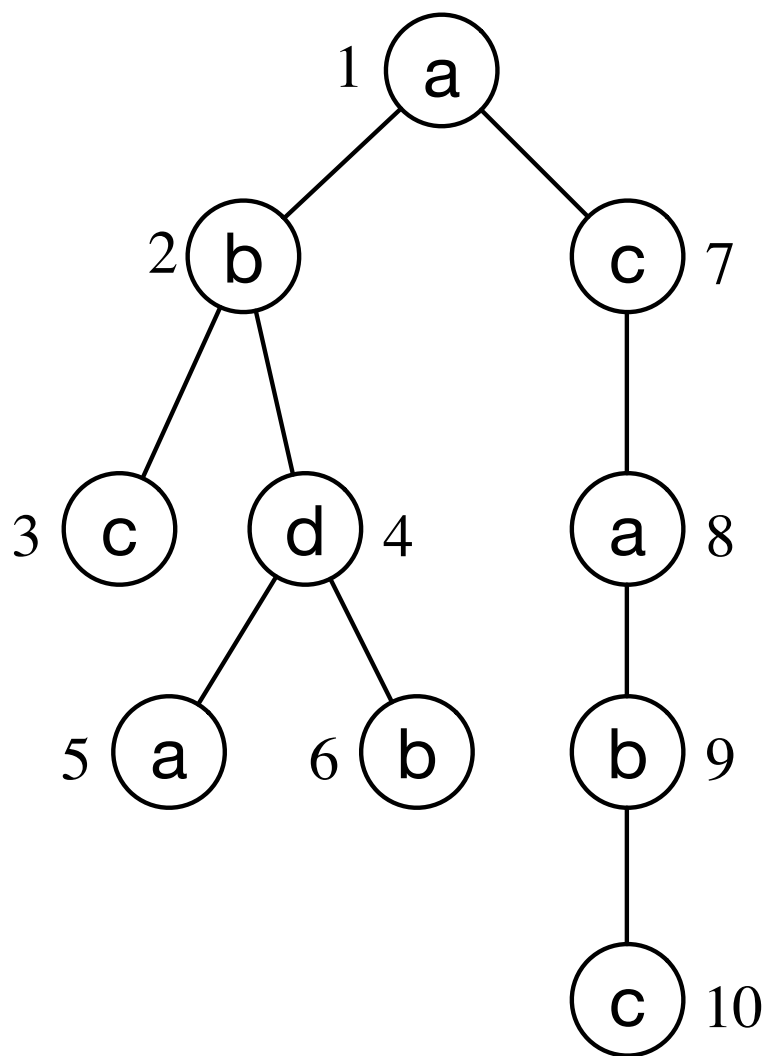
XPath: Bottom-up Evaluation of Core XPath



label	T(label)
a	{1,5,8}
b	{2,6,9}
c	{3,7,10}
d	{4}

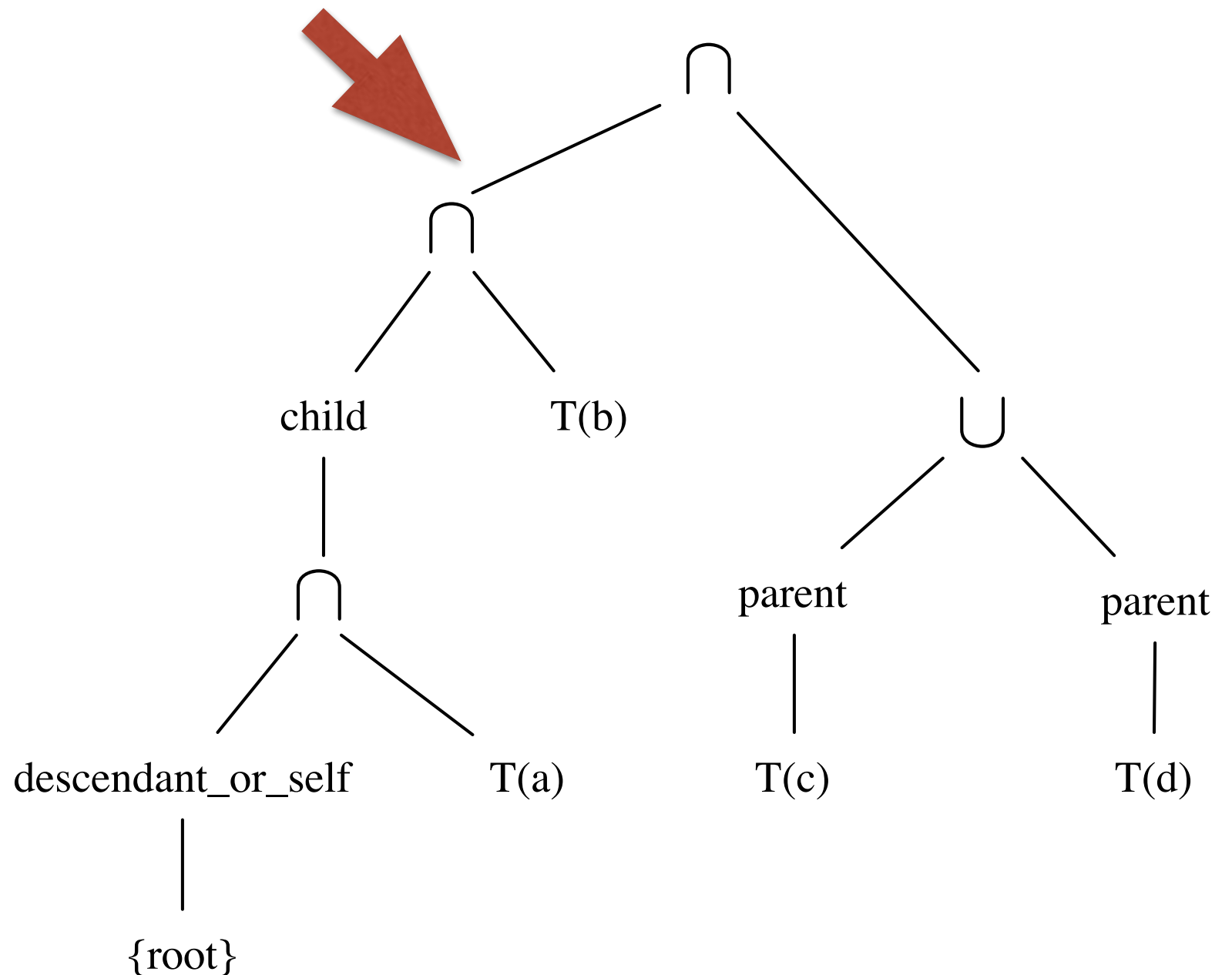
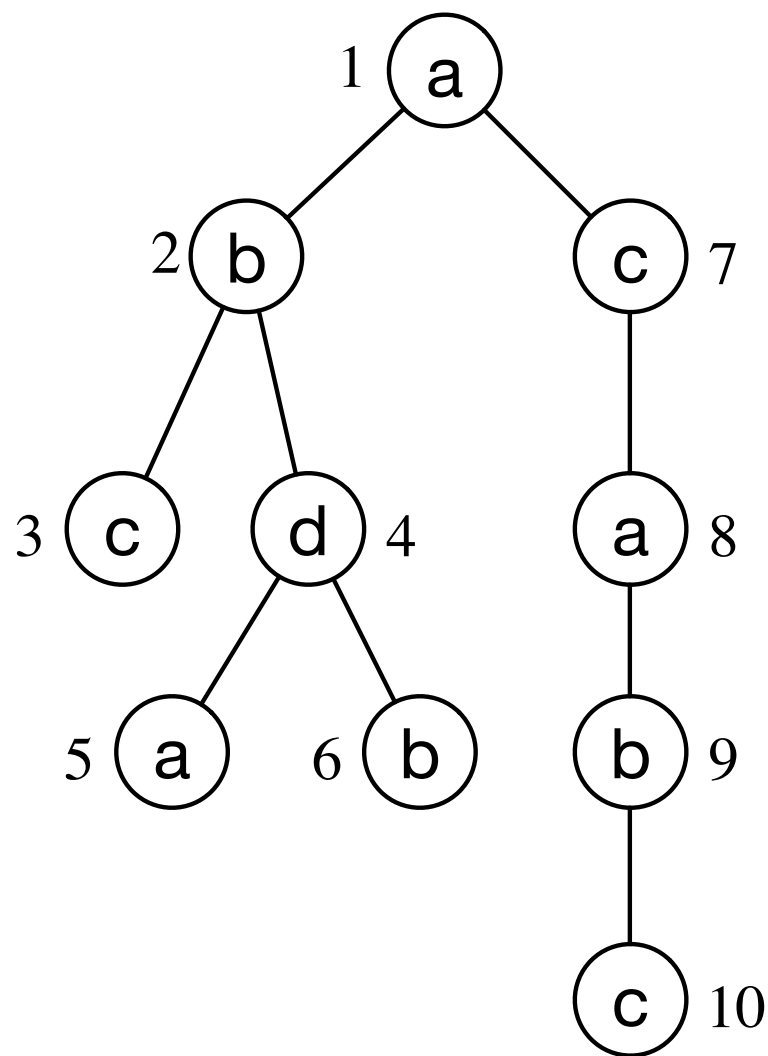
XPath: Bottom-up Evaluation of Core XPath

`/descendant_or_self::a/child::b[child::d or child::c]`



XPath: Bottom-up Evaluation of Core XPath

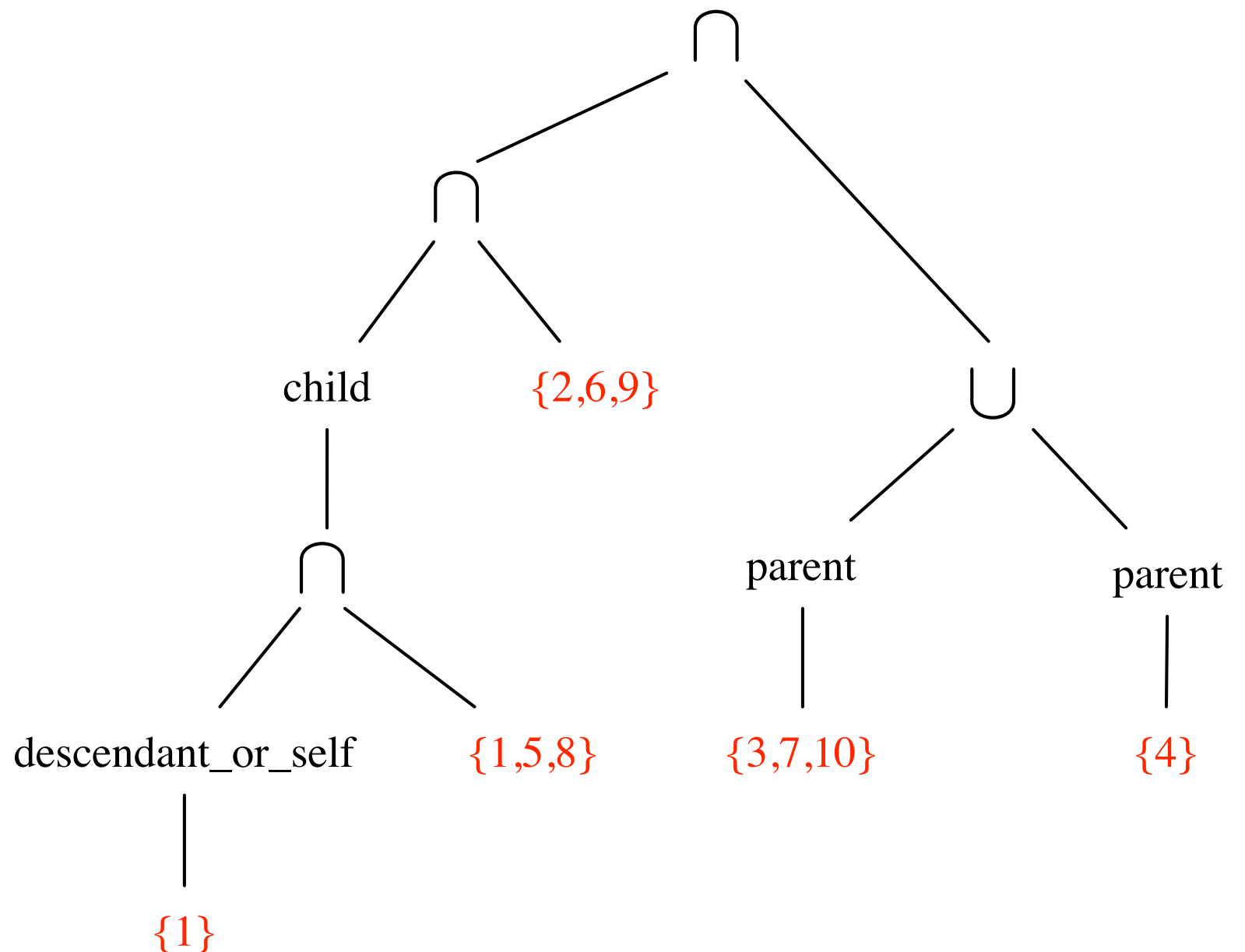
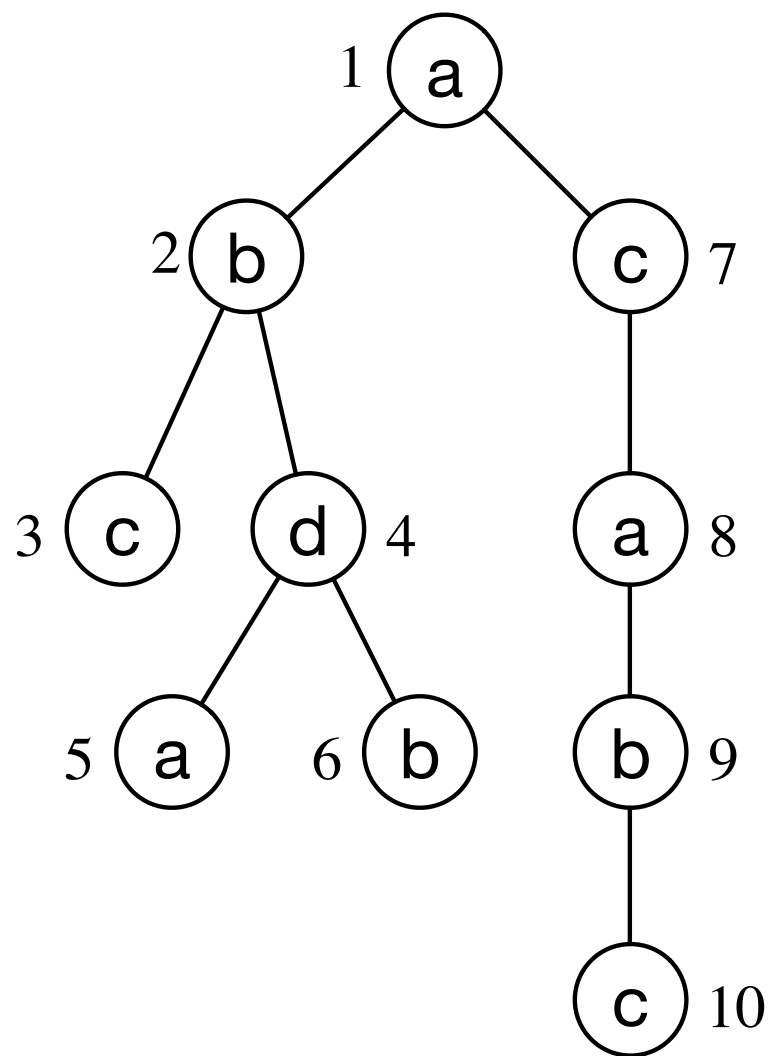
`/descendant_or_self::a/child::b[child::d or child::c]`



XPath: Bottom-up

Evaluation of Core XPath

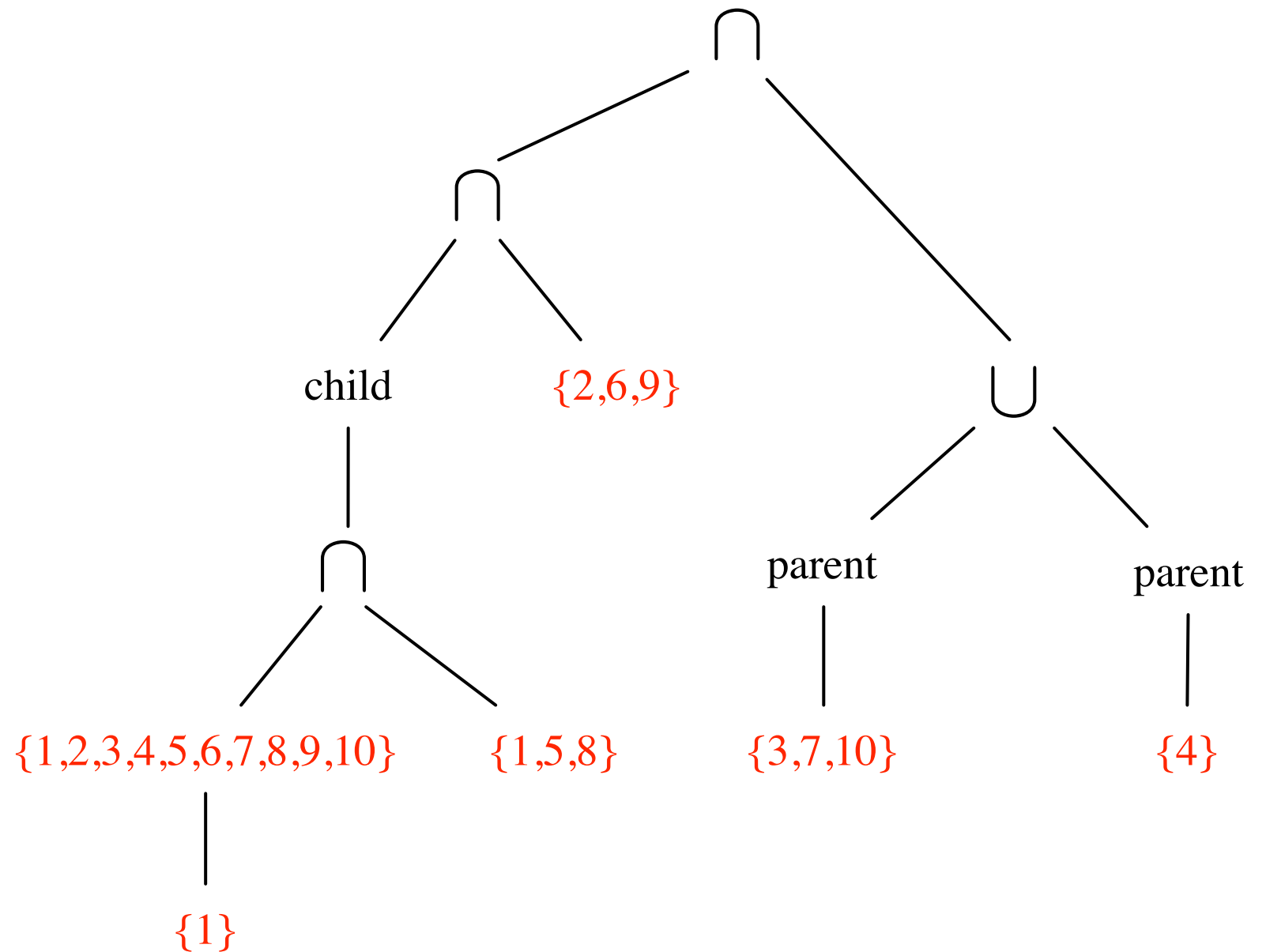
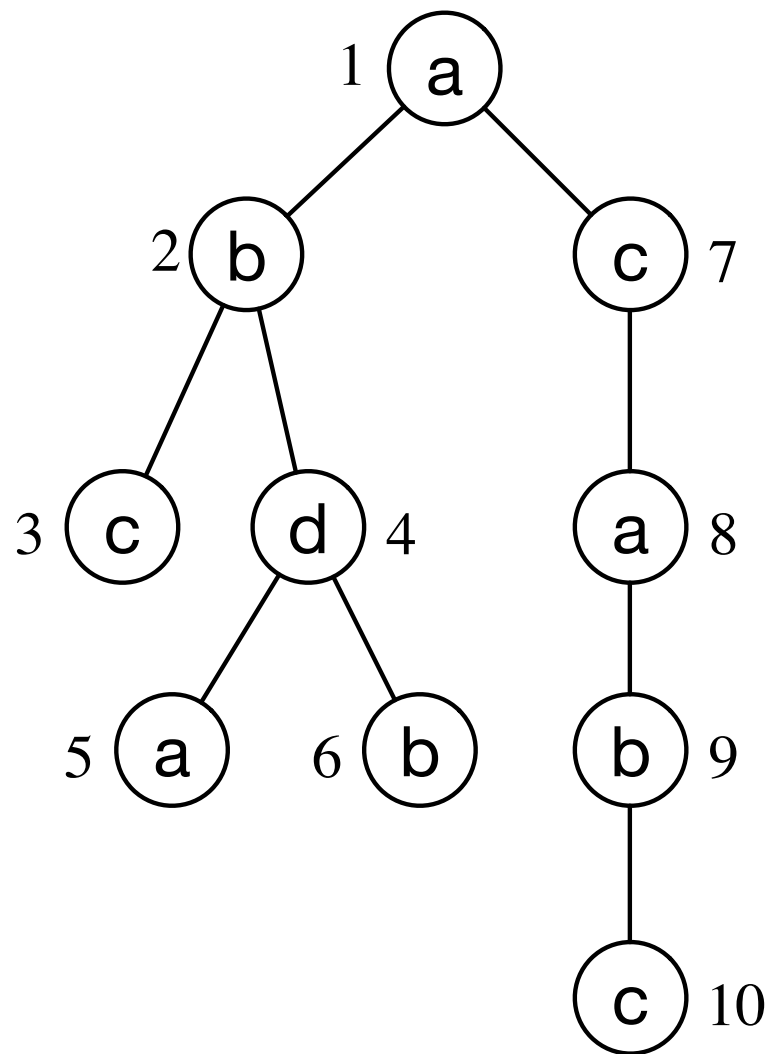
`/descendant_or_self::a/child::b[child::d or child::c]`



XPath: Bottom-up

Evaluation of Core XPath

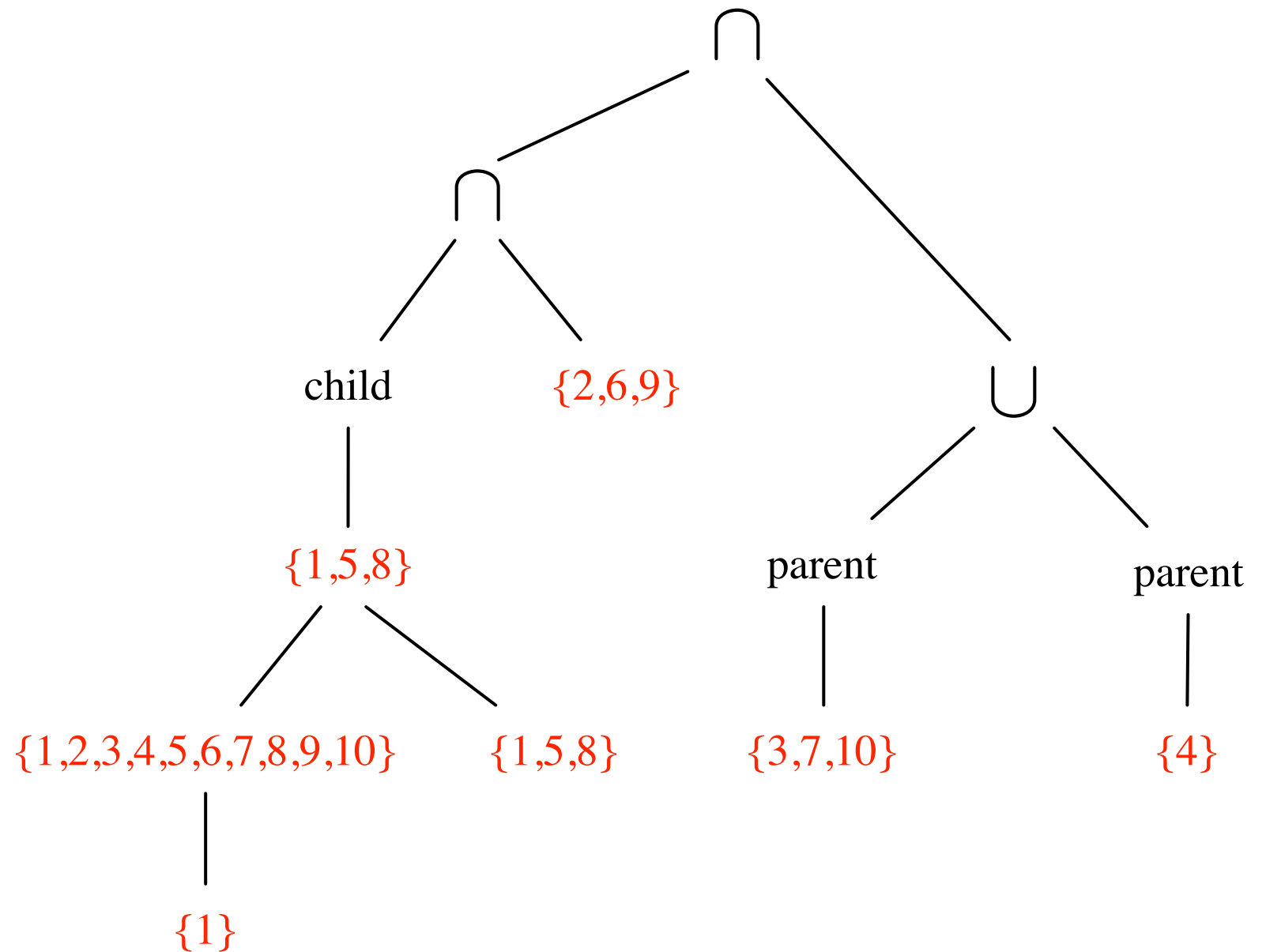
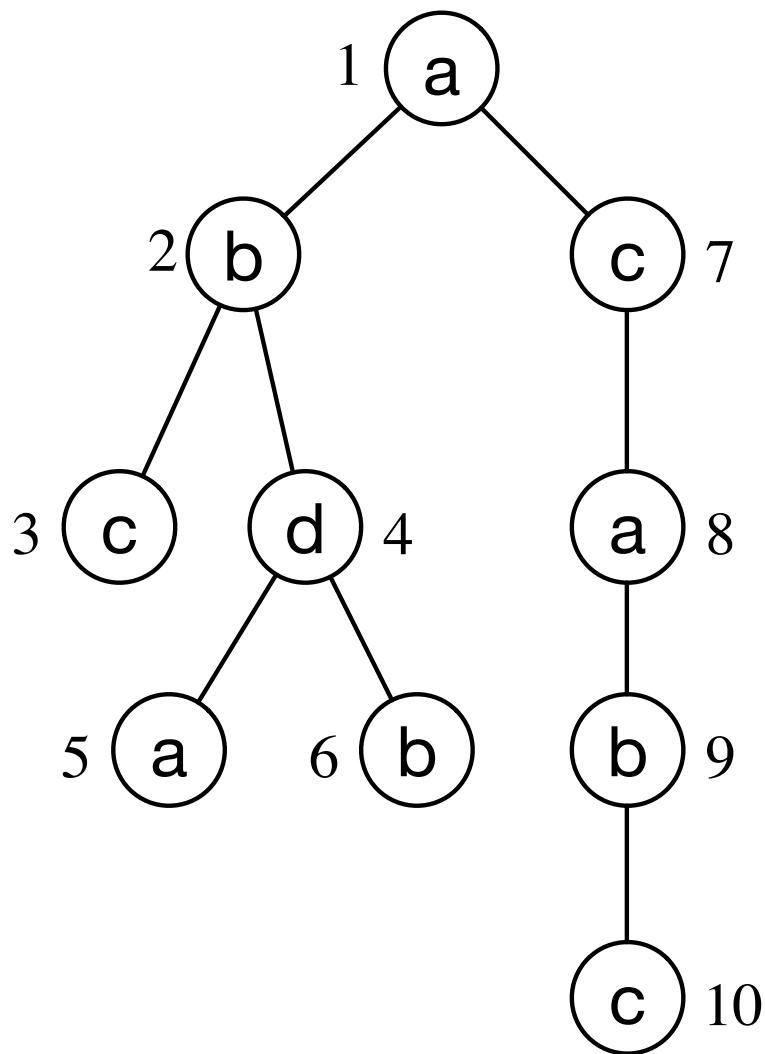
`/descendant_or_self::a/child::b[child::d or child::c]`



XPath: Bottom-up

Evaluation of Core XPath

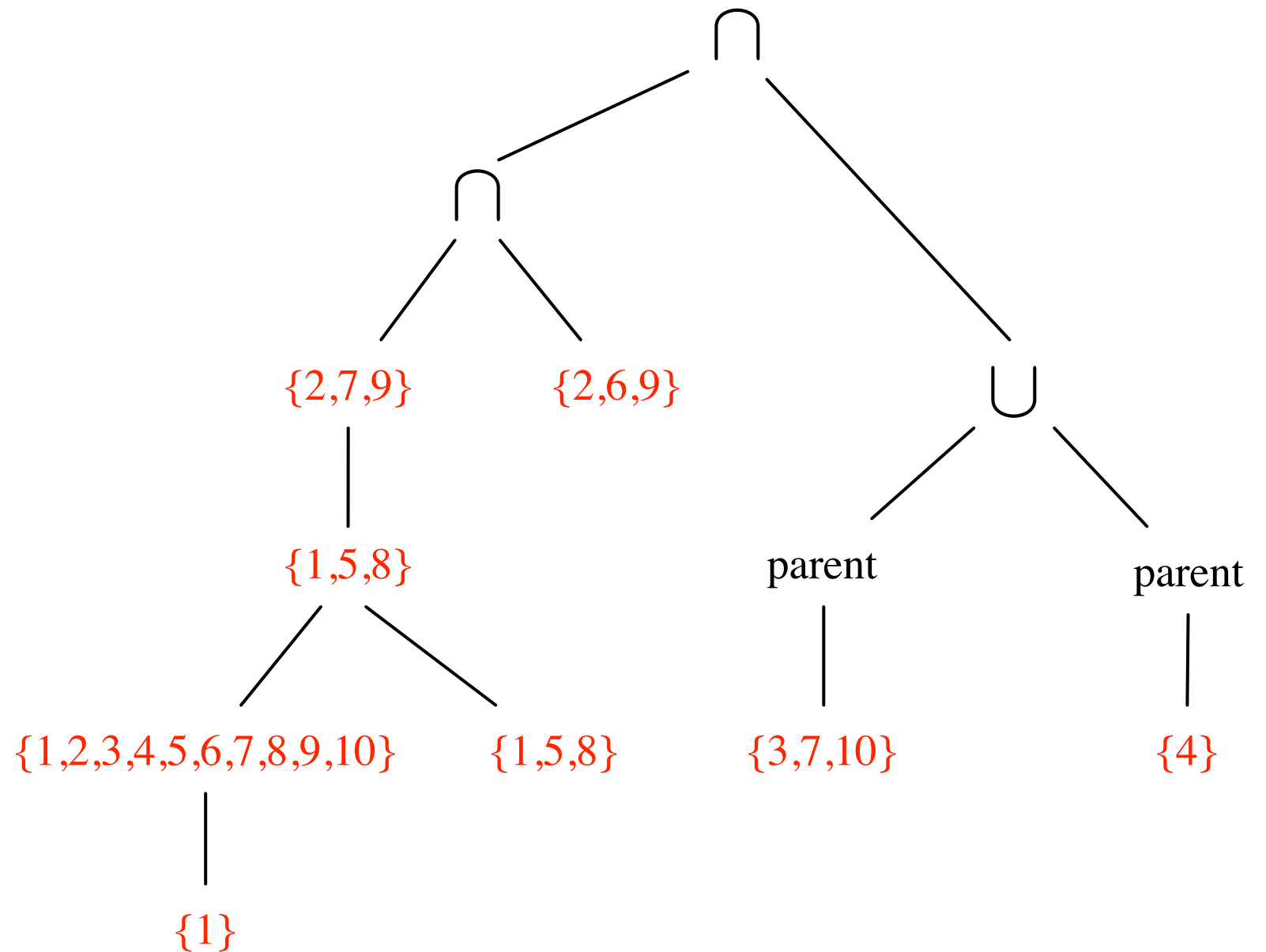
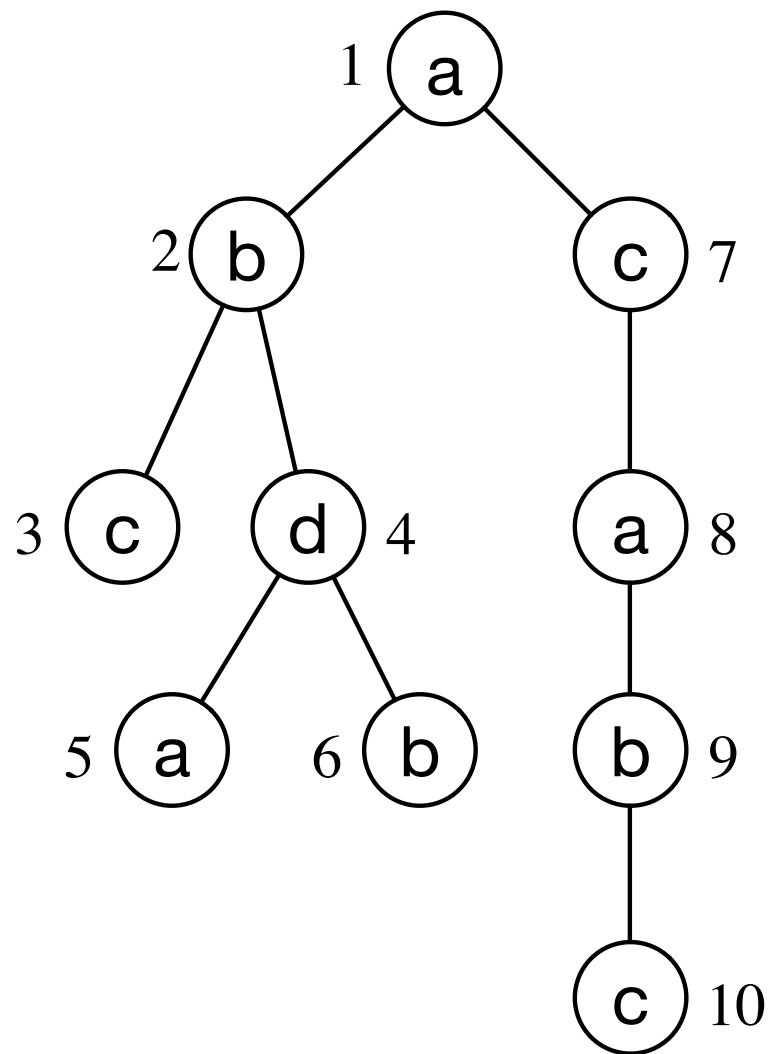
`/descendant_or_self::a/child::b[child::d or child::c]`



XPath: Bottom-up

Evaluation of Core XPath

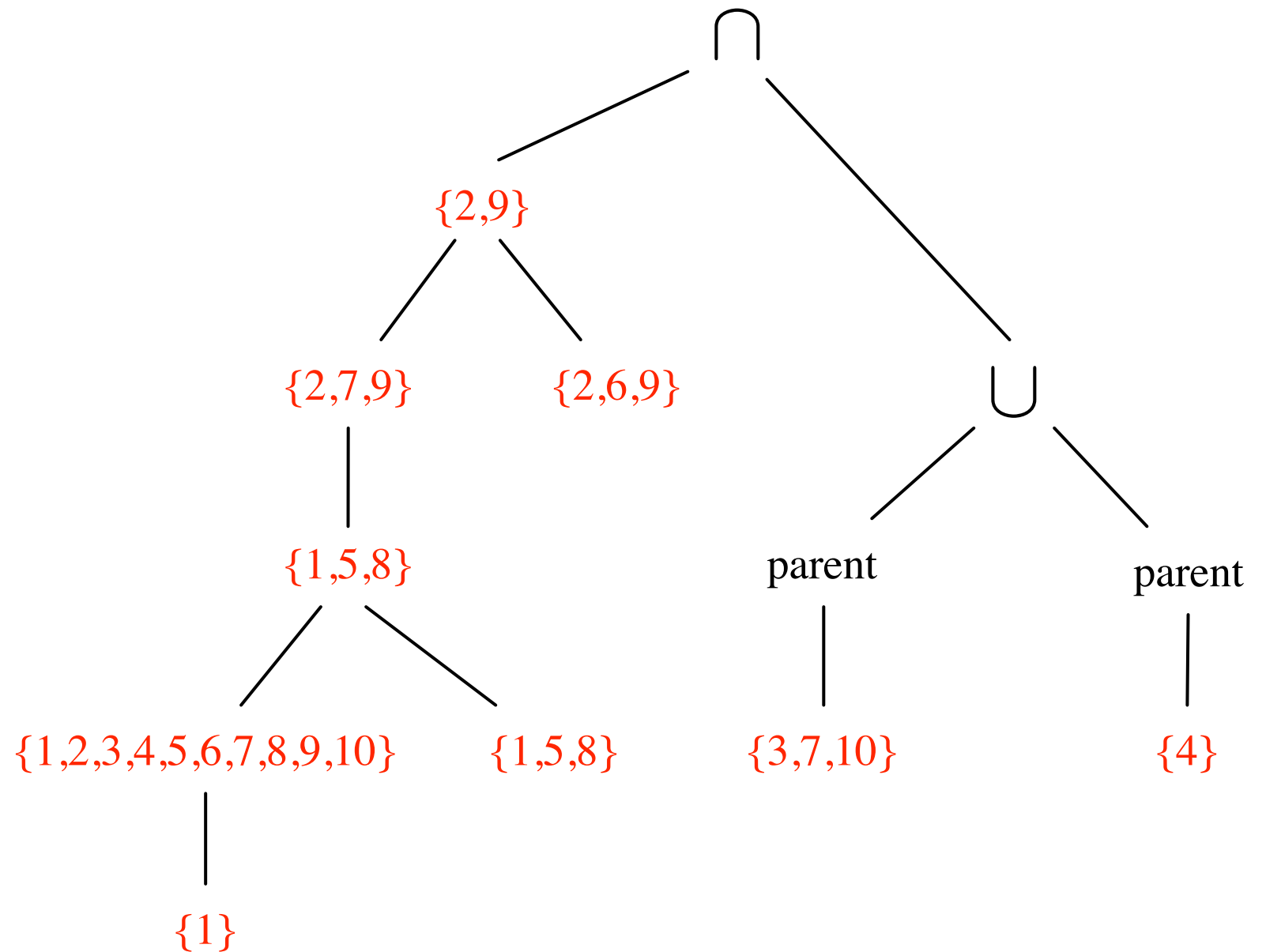
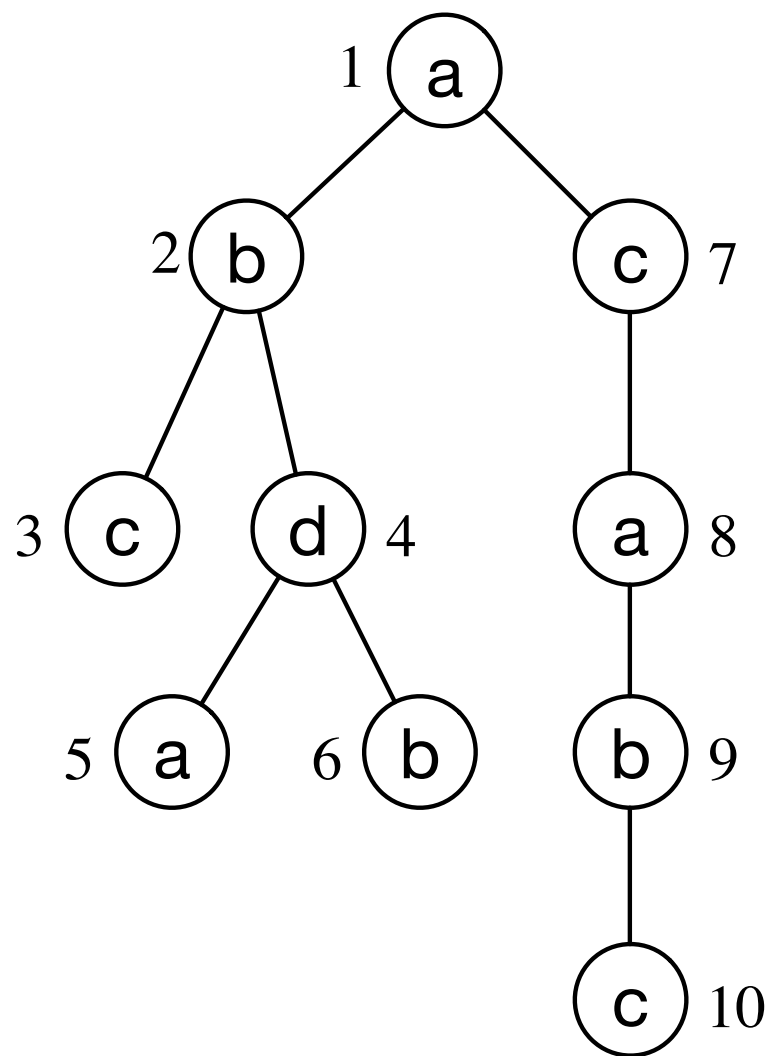
`/descendant_or_self::a/child::b[child::d or child::c]`



XPath: Bottom-up

Evaluation of Core XPath

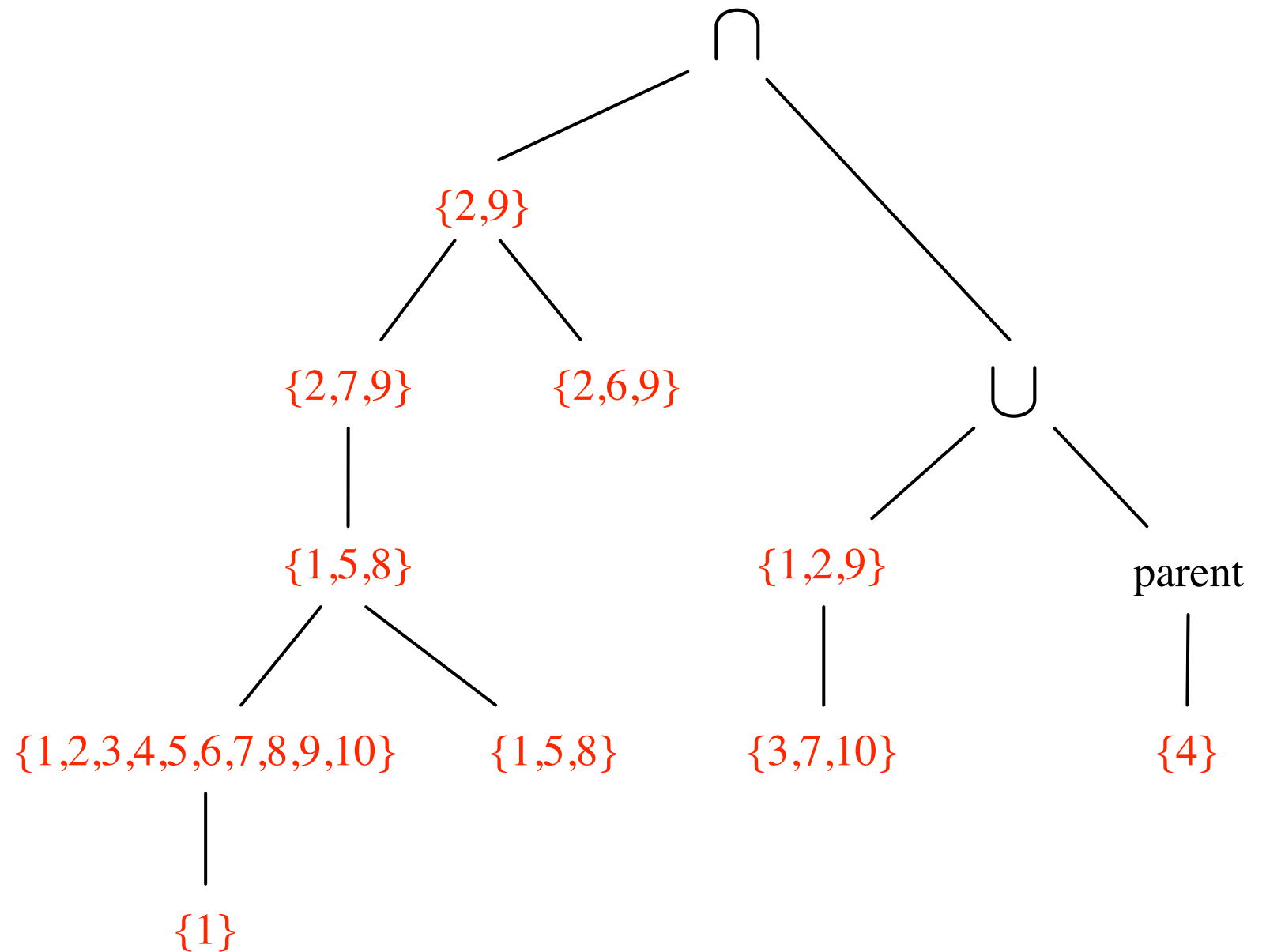
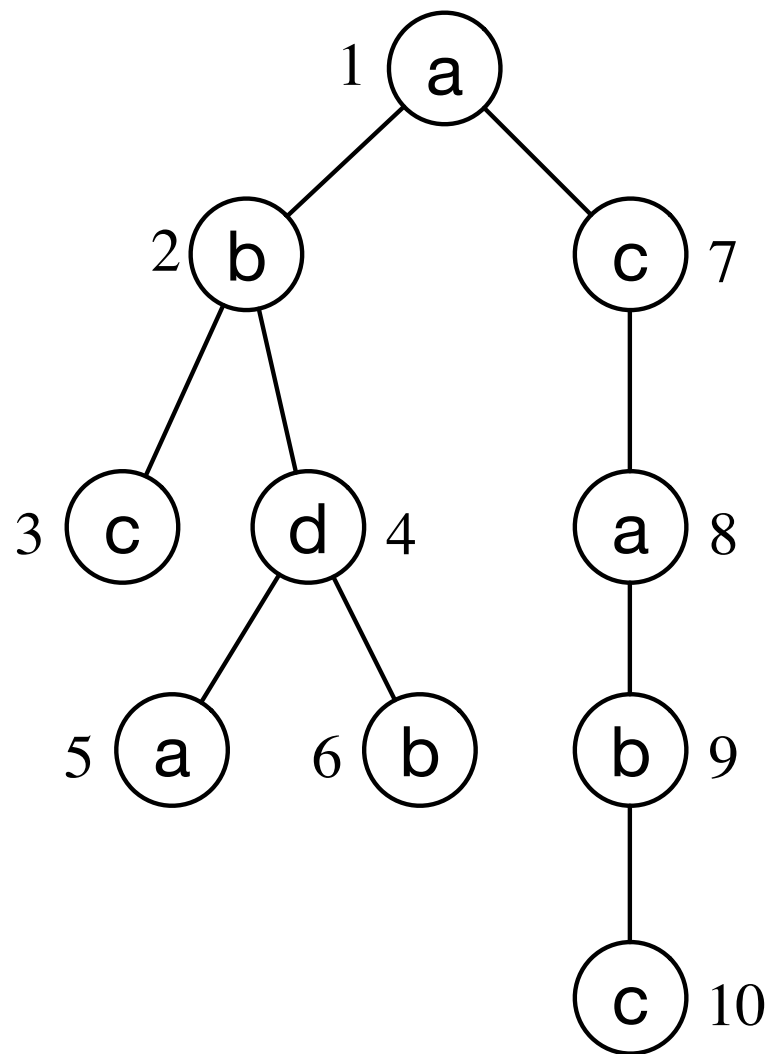
`/descendant_or_self::a/child::b[child::d or child::c]`



XPath: Bottom-up

Evaluation of Core XPath

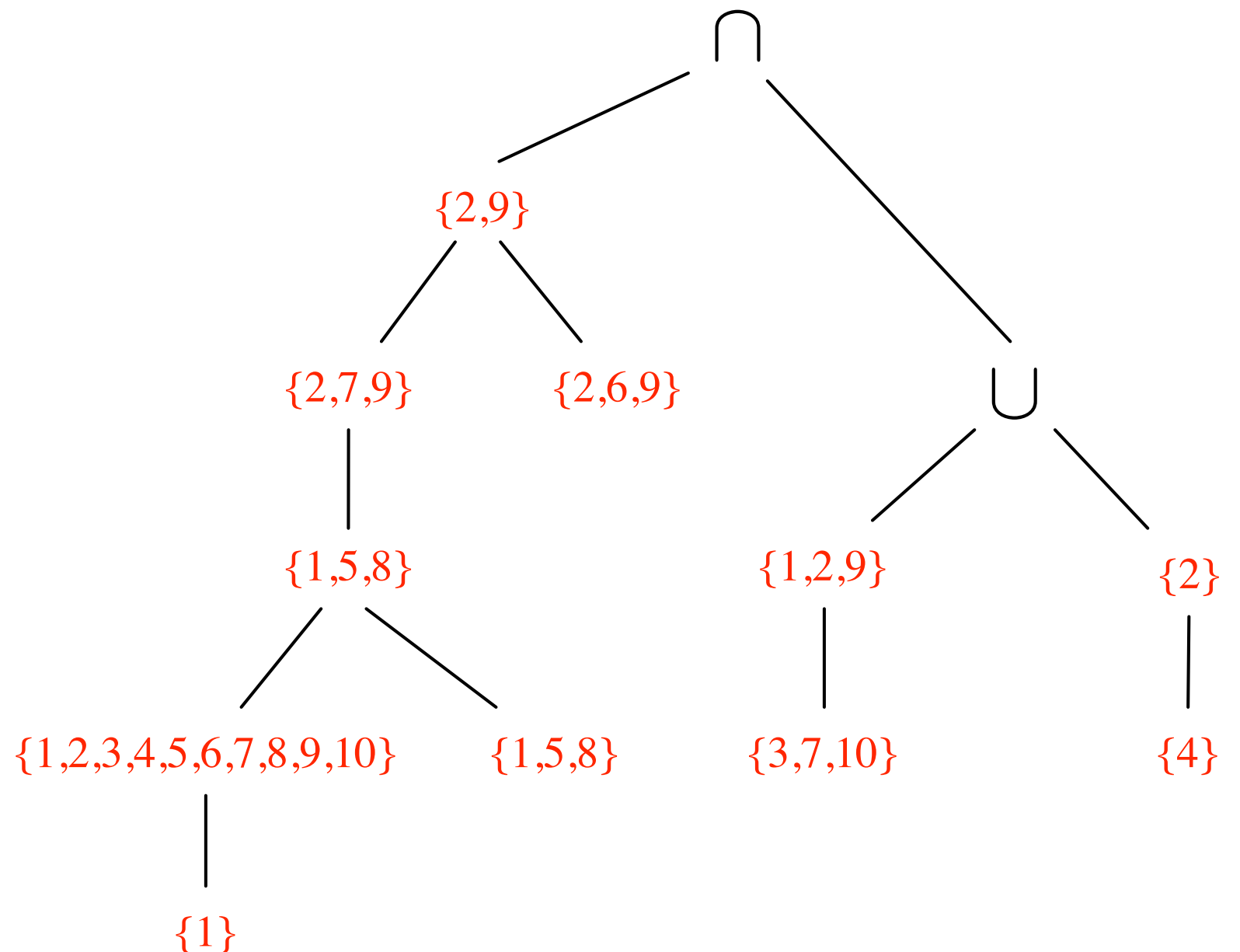
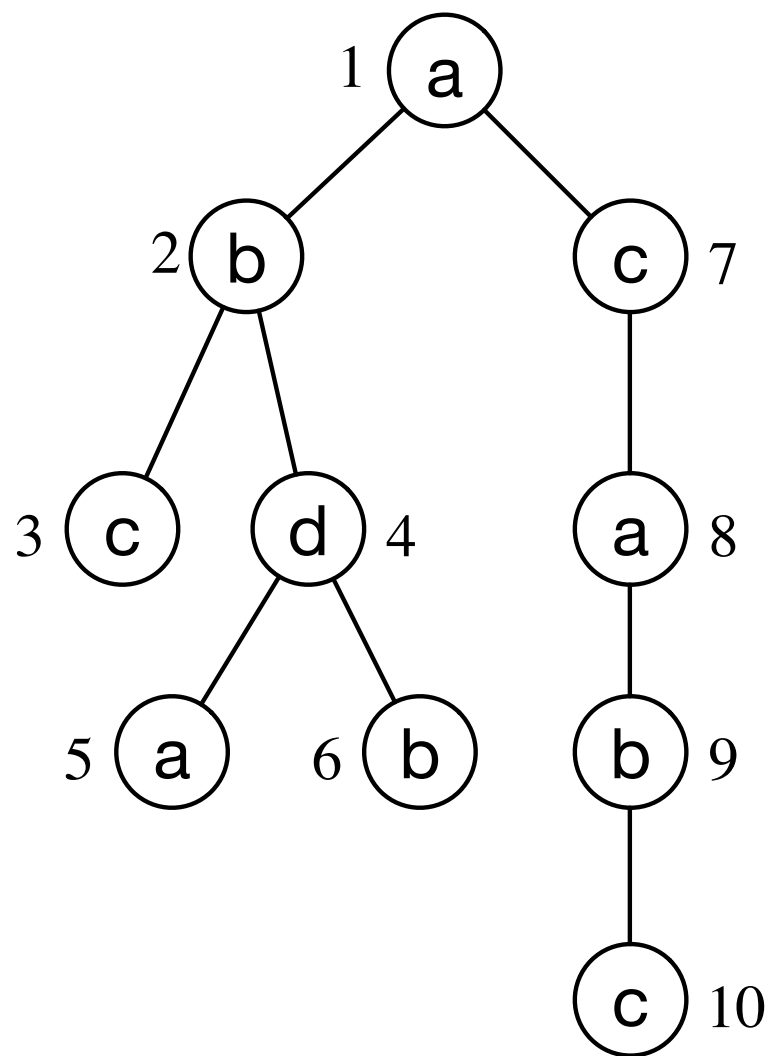
`/descendant_or_self::a/child::b[child::d or child::c]`



XPath: Bottom-up

Evaluation of Core XPath

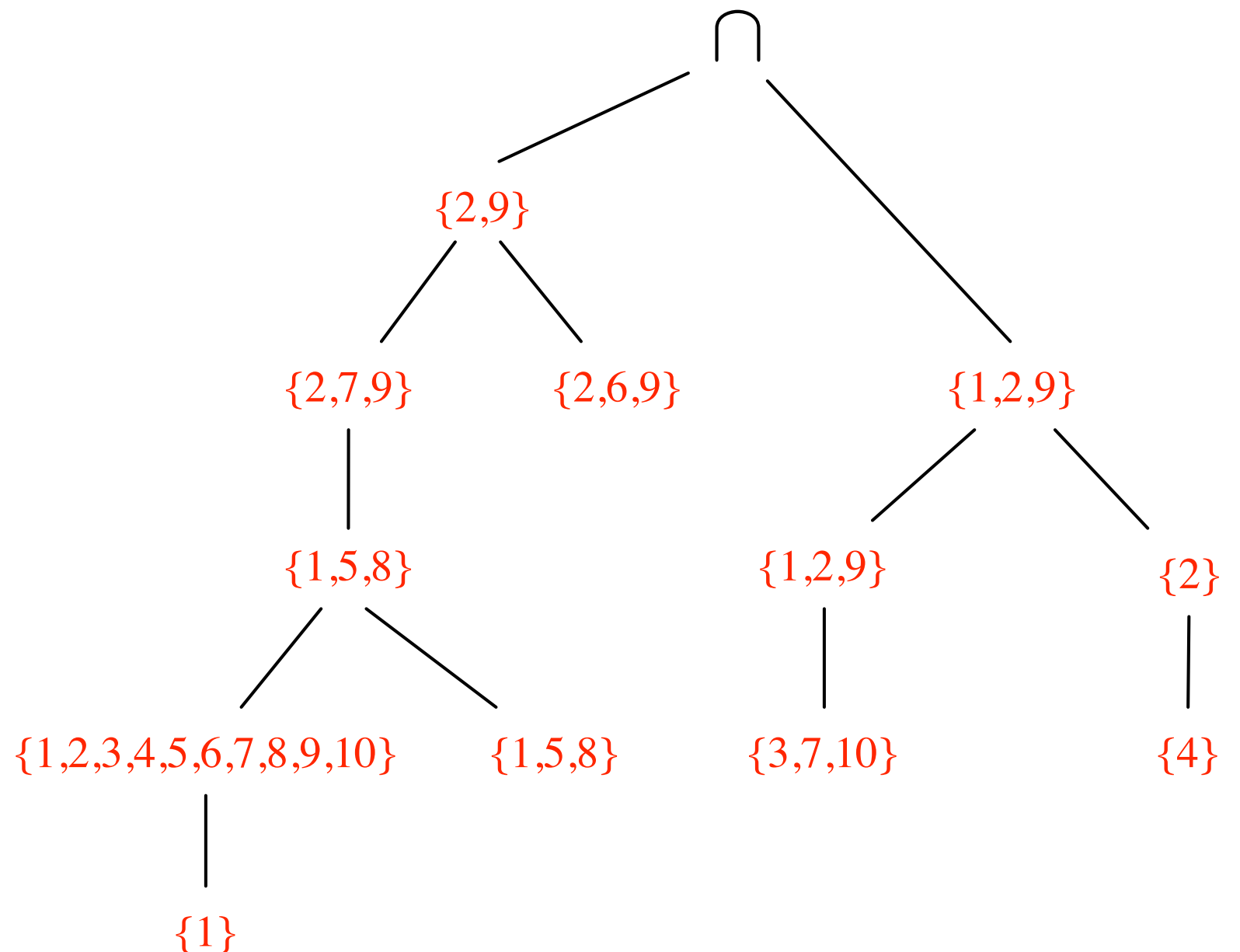
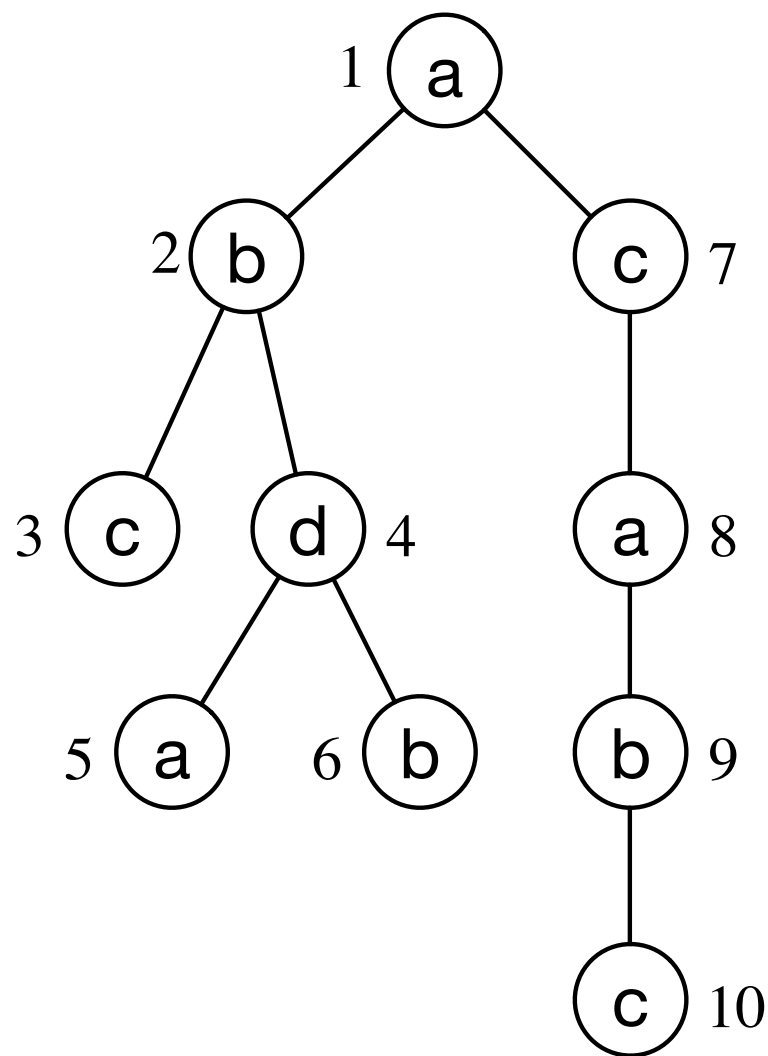
`/descendant_or_self::a/child::b[child::d or child::c]`



XPath: Bottom-up

Evaluation of Core XPath

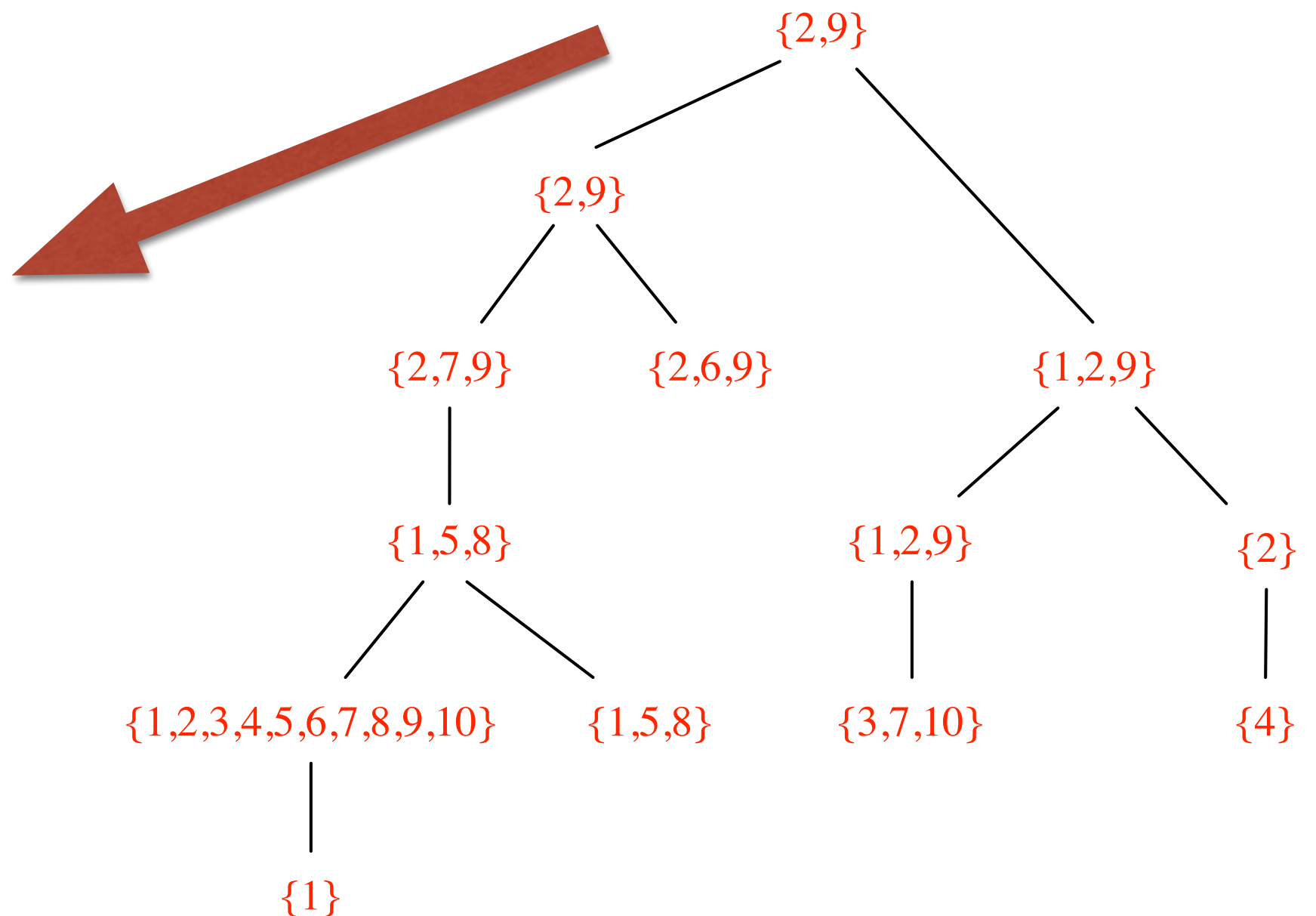
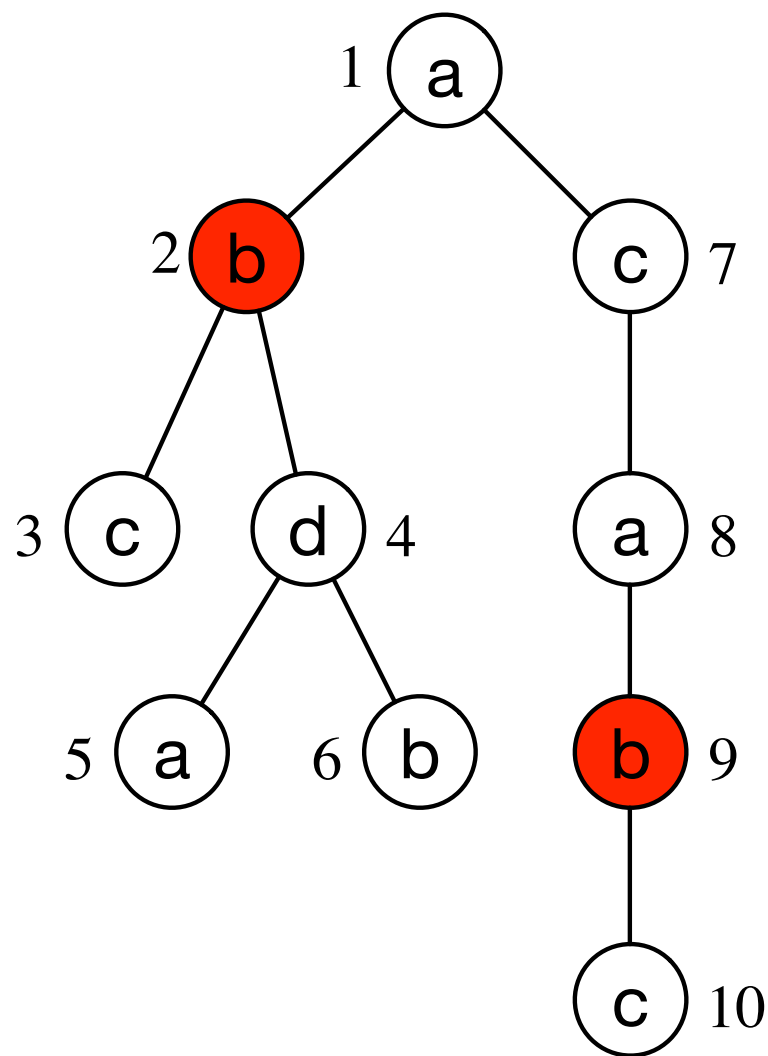
`/descendant_or_self::a/child::b[child::d or child::c]`



XPath: Bottom-up

Evaluation of Core XPath

`/descendant_or_self::a/child::b[child::d or child::c]`



XPath: Bottom-up Evaluation of Core XPath

- each **set operation** and the **bookkeeping** takes $O(|D|)$ time
- the **parse tree** is of size $O(|Q|)$
- hence, **linear processing**:

$$O(|D| \cdot |Q|)$$

Reading Assignment: Evaluation Using Automata

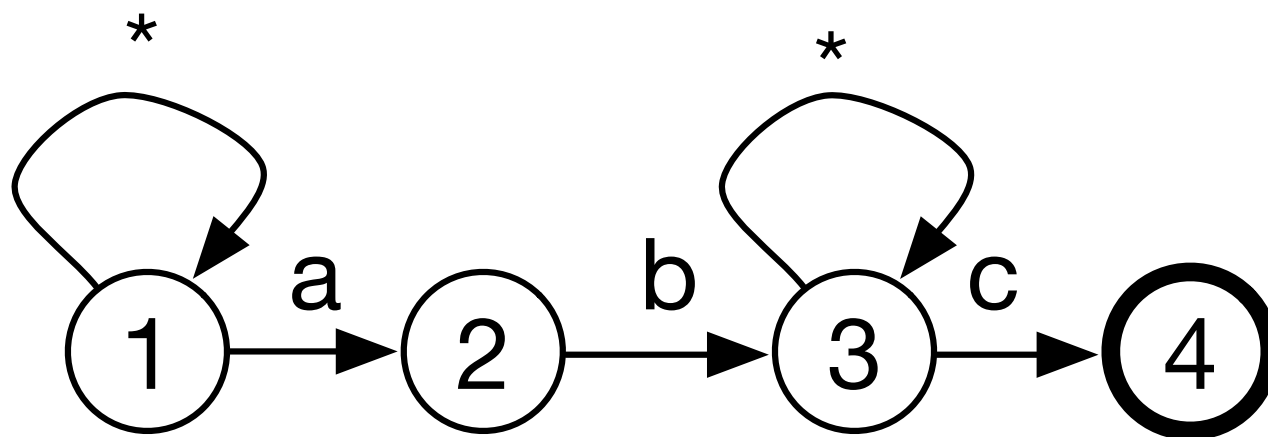
Principle: Use the XPath expression as a regular expression matching the paths of the tree.

Reading Assignment: Evaluation Using Automata

Principle: Use the XPath expression as a regular expression matching the paths of the tree.

Reading Assignment: Evaluation Using Automata

//a/b//c



Reading Assignment: Evaluation Using Automata

- dealing with $*$ transitions is quite tricky
- transforming the NFA into a DFA has exponential blow-up
- good news: do not need to transform into DFA (lazy DFA)

Reading Assignment: Evaluation Using Automata

- dealing with * transitions is quite tricky
- transforming the NFA into a DFA has exponential blow-up
- good news: do not need to transform into DFA (lazy DFA)

Green, Gupta, Miklau, Onizuka, Suciu. “**Processing XML Streams with Deterministic Automata and Stream Indexes**”
ACM TODS 29(4), 2004

XPath: Bottom-up Evaluation of Full XPath

- we can still have polynomial evaluation of full XPath (similar principle as Core XPath)

XPath: Bottom-up Evaluation of Full XPath

Context-value tables:

- each **context** in XPath can be represented using a **context-value table** (specifies situations in which a subquery should be evaluated):

$$\text{context} : < x, k, n >$$

- determined by **preceding XPath computations** —
bottom-up algorithm

XPath: Bottom-up Evaluation of Full XPath

$\mathcal{E}_\uparrow : \text{Expression} \rightarrow \text{nset} \cup \text{num} \cup \text{str} \cup \text{bool},$

Expr. E : Operator Signature Semantics $\mathcal{E}_\uparrow \llbracket E \rrbracket$
location step $\chi::t : \rightarrow \text{nset}$ $\{\langle x_0, k_0, n_0, \{x \mid x_0 \chi x, x \in T(t)\} \rangle \mid \langle x_0, k_0, n_0 \rangle \in \mathbf{C}\}$
location step $E[e]$ over axis χ : $\text{nset} \times \text{bool} \rightarrow \text{nset}$ $\{\langle x_0, k_0, n_0, \{x \in S \mid \langle x, \text{idx}_\chi(x, S), S , \text{true} \rangle \in \mathcal{E}_\uparrow \llbracket e \rrbracket \} \rangle \mid \langle x_0, k_0, n_0, S \rangle \in \mathcal{E}_\uparrow \llbracket E \rrbracket\}$
location path $\pi : \text{nset} \rightarrow \text{nset}$ $\mathbf{C} \times \{S \mid \exists k, n : \langle \text{root}, k, n, S \rangle \in \mathcal{E}_\uparrow \llbracket \pi \rrbracket\}$
location path $\pi_1 / \pi_2 : \text{nset} \times \text{nset} \rightarrow \text{nset}$ $\{\langle x, k, n, z \rangle \mid 1 \leq k \leq n \leq \text{dom} ,$ $\langle x, k_1, n_1, Y \rangle \in \mathcal{E}_\uparrow \llbracket \pi_1 \rrbracket,$ $\bigcup_{y \in Y} \langle y, k_2, n_2, z \rangle \in \mathcal{E}_\uparrow \llbracket \pi_2 \rrbracket\}$
location path $\pi_1 \mid \pi_2 : \text{nset} \times \text{nset} \rightarrow \text{nset}$ $\mathcal{E}_\uparrow \llbracket \pi_1 \rrbracket \cup \mathcal{E}_\uparrow \llbracket \pi_2 \rrbracket$
position() : $\rightarrow \text{num}$ $\{\langle x, k, n, k \rangle \mid \langle x, k, n \rangle \in \mathbf{C}\}$
last() : $\rightarrow \text{num}$ $\{\langle x, k, n, n \rangle \mid \langle x, k, n \rangle \in \mathbf{C}\}$

XPath: Bottom-up Evaluation of Full XPath

$\mathcal{E}_\uparrow : \text{Expression} \rightarrow \text{nset} \cup \text{num} \cup \text{str} \cup \text{bool},$

$$\begin{aligned} \mathcal{E}_\uparrow \llbracket Op(e_1, \dots, e_m) \rrbracket := \\ \{ \langle \vec{c}, \mathcal{F} \llbracket Op \rrbracket(v_1, \dots, v_m) \rangle \mid \vec{c} \in \mathbf{C}, \langle \vec{c}, v_1 \rangle \in \mathcal{E}_\uparrow \llbracket e_1 \rrbracket, \dots, \\ \langle \vec{c}, v_m \rangle \in \mathcal{E}_\uparrow \llbracket e_m \rrbracket \} \end{aligned}$$

XPath: Bottom-up Evaluation of Full XPath

Context-Value Principle (CVT):

- the size of each of the context-value tables is polynomial
- computing each combination step of the expression is polynomial
- hence, the computation is polynomial

XPath: Bottom-up Evaluation of Full XPath

GOTTLOB EXAMPLE SLIDES

XPath: Bottom-up Evaluation of Full XPath

Space Complexity:

- $O(|Q|)$ relations are created,
- nset are bounded by $O(|D|^4)$, bool are bounded by $O(|D|^3)$
- numbers and string computable in XPath are of size $O(|D||Q|)$

$$O(|D|^4 \cdot |Q|^2)$$

XPath: Bottom-up Evaluation of Full XPath

Time Complexity:

- $O(|Q|)$ computations are needed (parse tree size is linear in the query size),
- $O(|D|^5|Q|)$ for each expression relation

$$O(|D|^5 \cdot |Q|^2)$$

Useful Reading

- Gottlob, Koch, Pichler. “**Efficient Algorithms for Processing XPath Queries**”, VLDB 2002.
- Green, Gupta, Miklau, Onizuka, Suciu.
“**Processing XML Streams with Deterministic Automata and Stream Indexes**”, ACM TODS 29(4), 2004.
- Benedikt, Koch. “**XPath Leashed**”, ACM Computing Surveys 41(1), 2009.