INFO-H-509
Exercises 2

XML Schema

Efficient processing of XPath

- Remember:

Q_1 = ancestor-or-self::*[not(parent::*)] / descendant-or-self::*

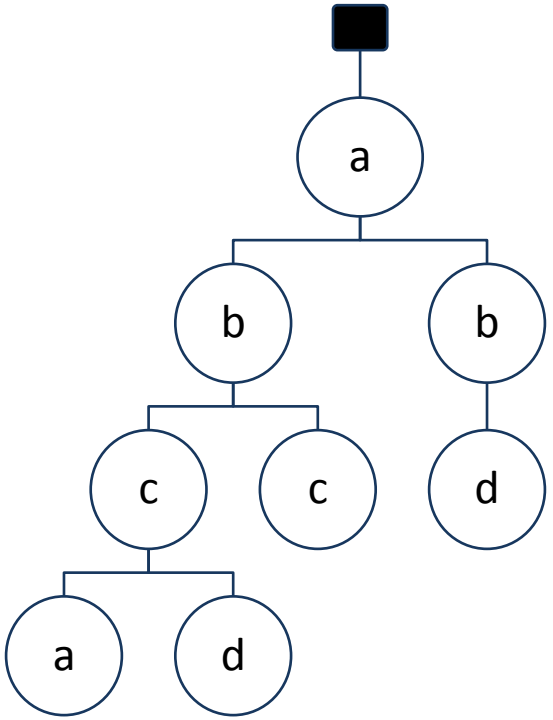
Q_i = ancestor-or-self::*[not(parent::*)] / descendant-or-self::* /Q_{i-1}

- Exponential time when using a naive algorithm

Efficient processing of XPath

Consider Q_2 with root element A as context:

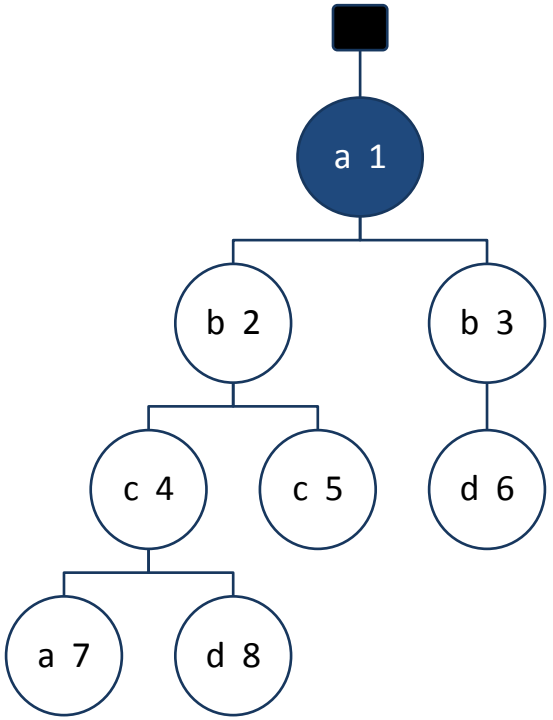
ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/
ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*



Efficient processing of XPath

ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/
ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/

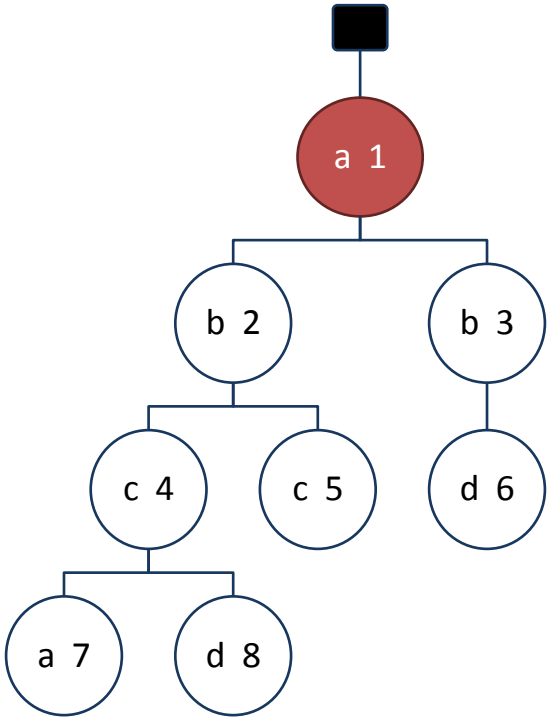
Call Naive on node 1



Efficient processing of XPath

ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/
ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/

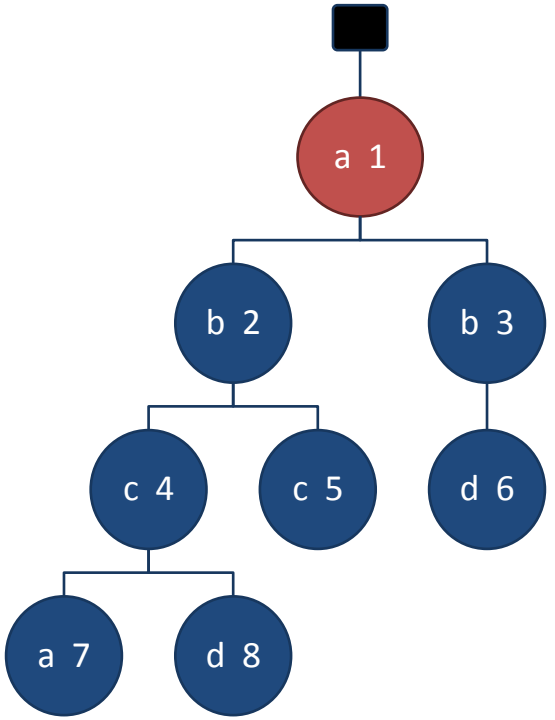
Call Naive on node 1



Efficient processing of XPath

ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/
ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/

Call Naive on node 1

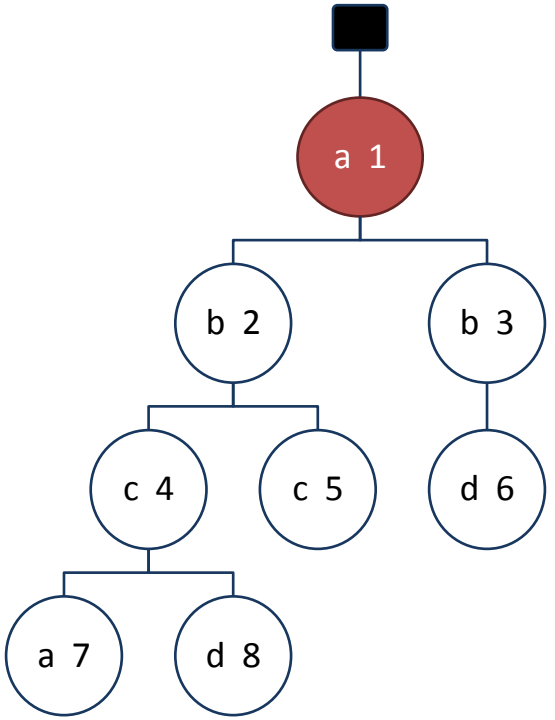


1
1

Efficient processing of XPath

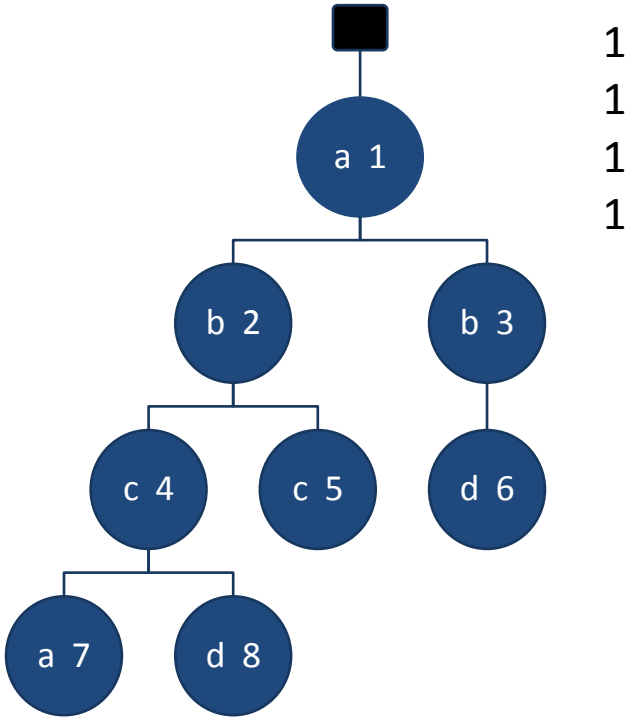
ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/
ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/

Call Naive on node 1



Efficient processing of XPath

ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/
ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/

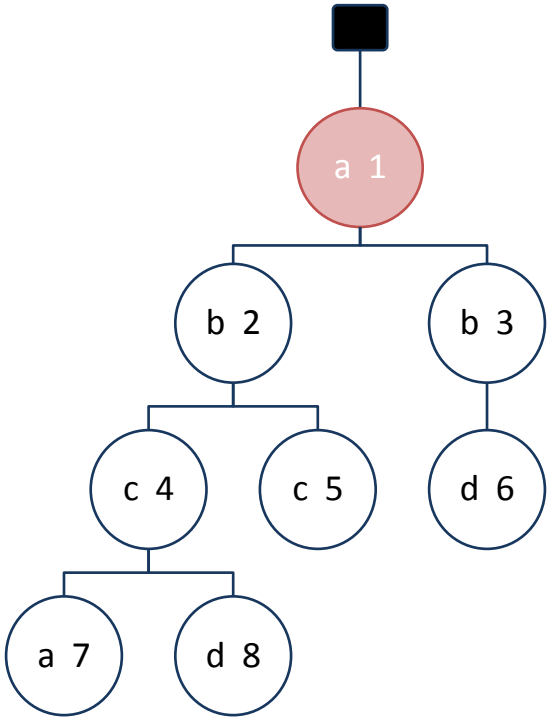


No more steps
Add { 1…8 } to the result

Efficient processing of XPath

ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/
ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/

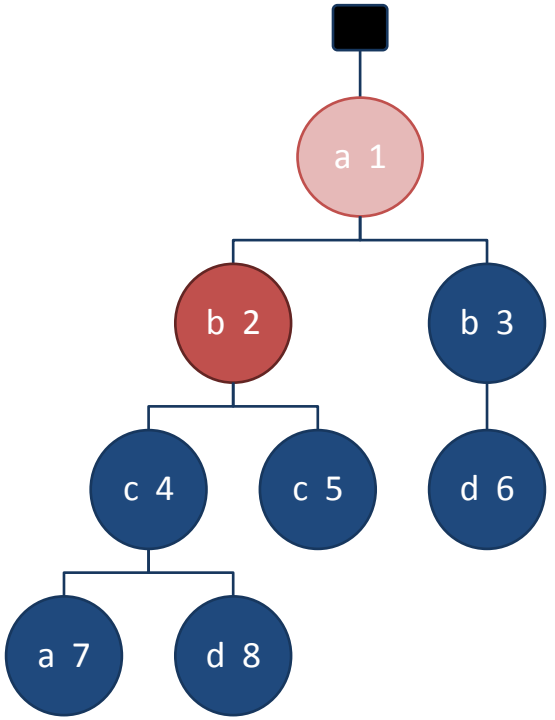
No more nodes



Efficient processing of XPath

ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/
ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/

Call Naive on node 2

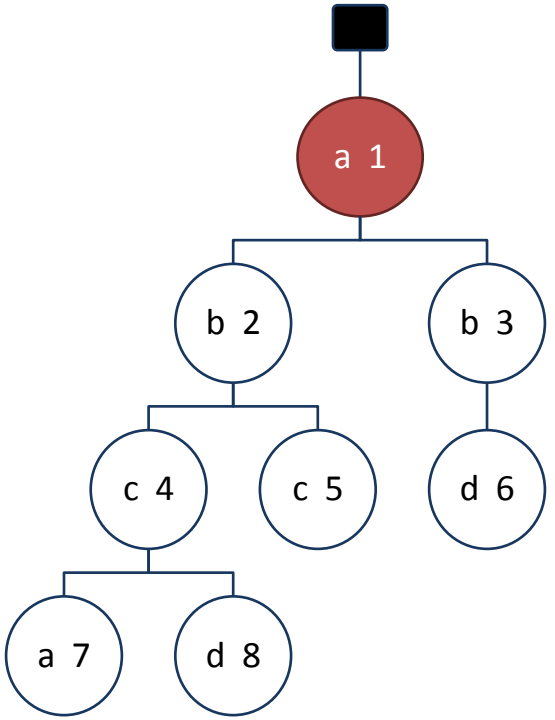


1
1

Efficient processing of XPath

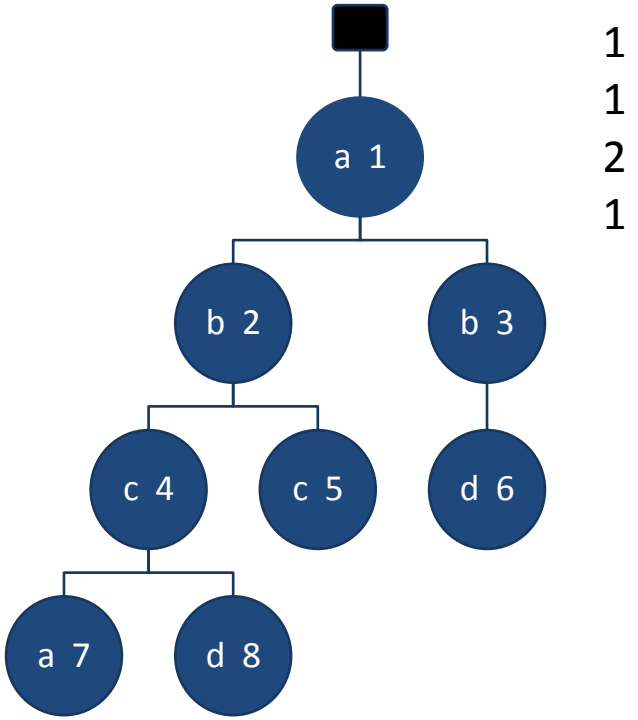
ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/
ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/

Call Naive on node 1



Efficient processing of XPath

ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/
ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/

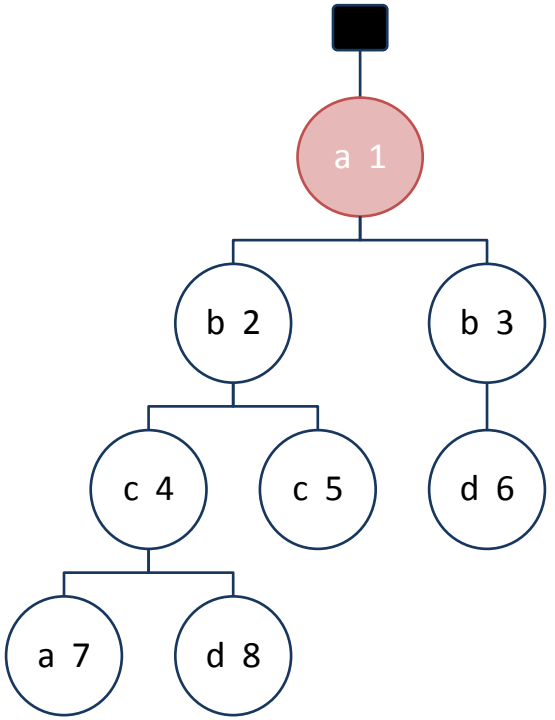


No more steps
Add { 1…8 } to the result, **again**

Efficient processing of XPath

ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/
ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/

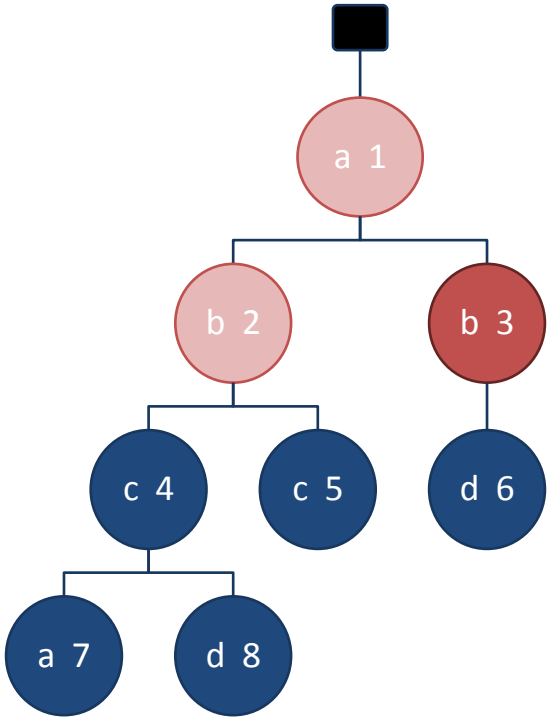
No more nodes



1
1
2

Efficient processing of XPath

ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/
ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/



1
1

Call Naive on node 3
…
And add { 1…8 } to the result, **again**

Efficient processing of XPath

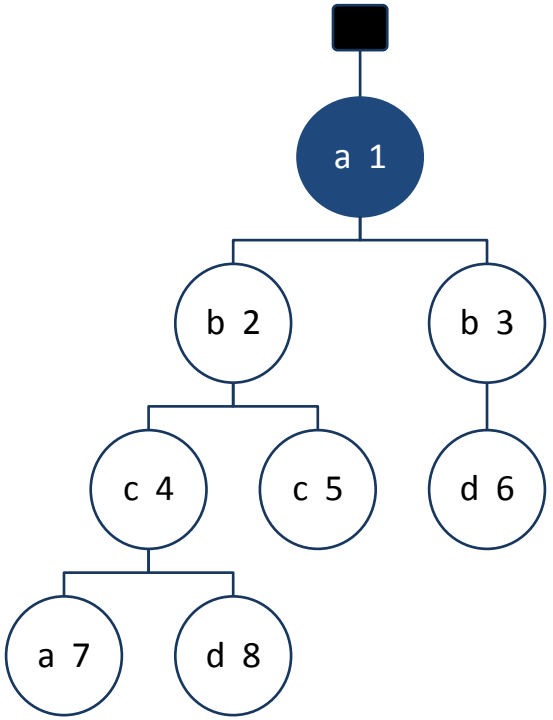
- Each for each node, every result is recomputed
  - Even when the context is the same!

- Solution: compute each result once.
- Read the expression from the left to the right
  - Never go back!

Efficient processing of XPath

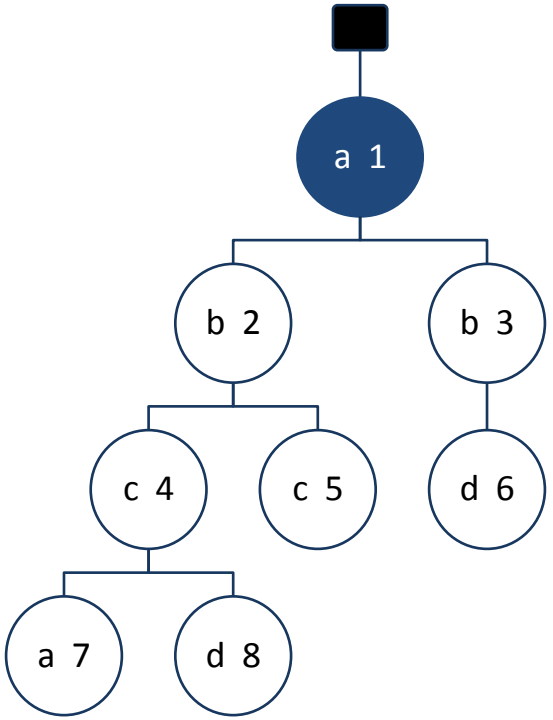
ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/
ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/

Set Temp to our context node
Now, Temp = { 1 }



Efficient processing of XPath

ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/
ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/

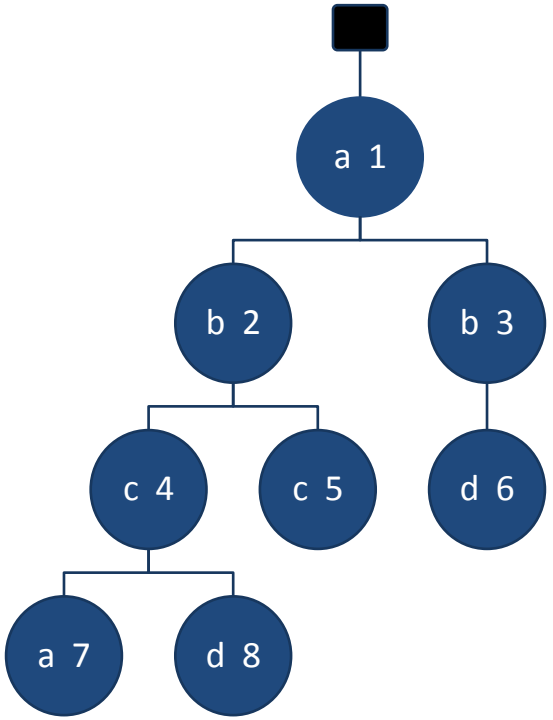


Temp = Call Eval-step with Temp
Now, Temp = { 1 }

Efficient processing of XPath

ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/
ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/

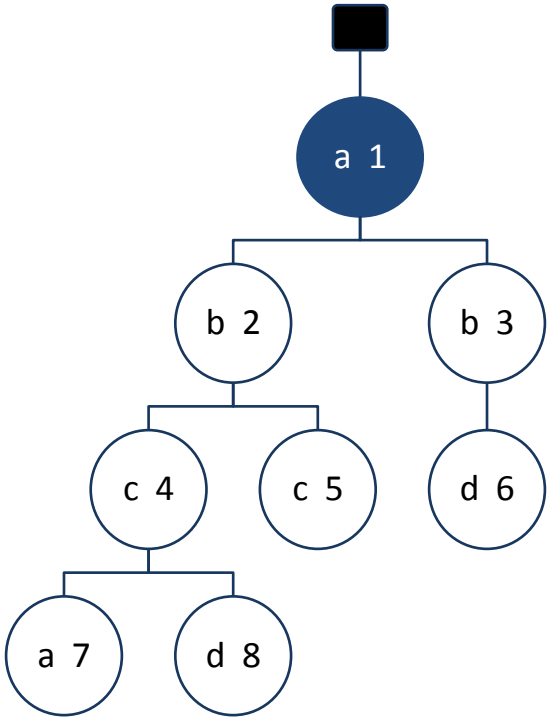
Temp = Call Eval-step with Temp
Now, Temp = { 1…8 }



Efficient processing of XPath

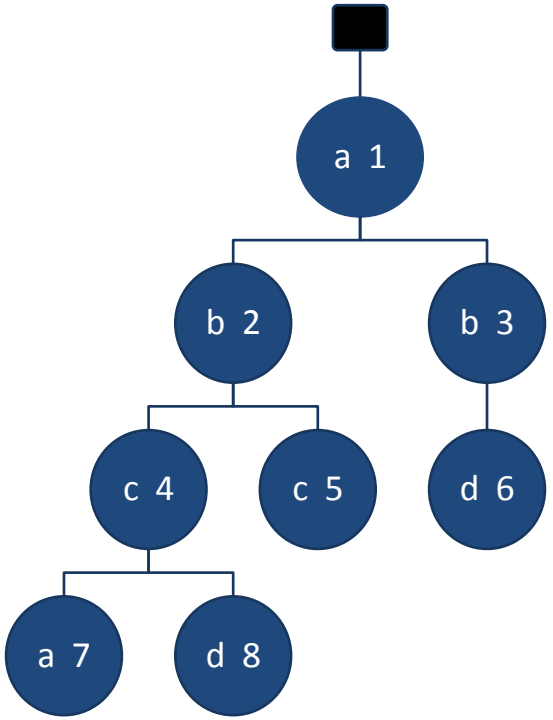
ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/
ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/

Temp = Call Eval-step with Temp
Now, Temp = { 1 }



Efficient processing of XPath

ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/
ancestor-or-self::*[not(parent::*)] /
descendant-or-self::*/



Temp = Call Eval-step with Temp
Now, Temp = { 1 }
No more steps

Efficient processing of XPath

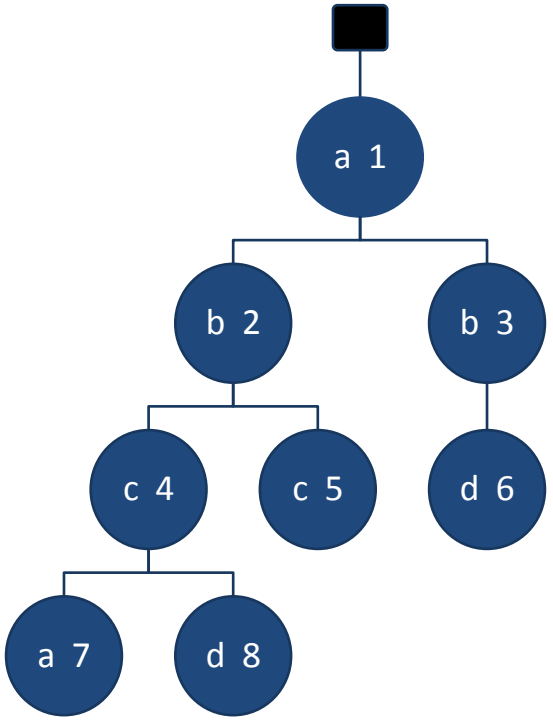
- For a single step, nodes are computed at once
- Read the expression from the left to the right
  - Never go back!

- The real algorithm is more complex
  - Filter expressions must also be handled

    …/ancestor-or-self::*[**not(parent::*)**]/…

Efficient processing of XPath

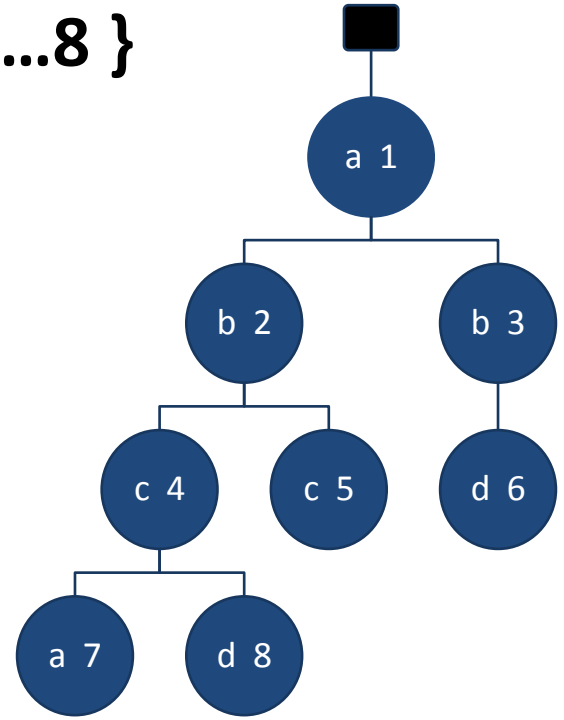
…/ancestor-or-self::*[2] / …



Efficient processing of XPath

…/ancestor-or-self::*[2] / …

The result of **ancestor-or-self::*** is **{ 1…8 }**



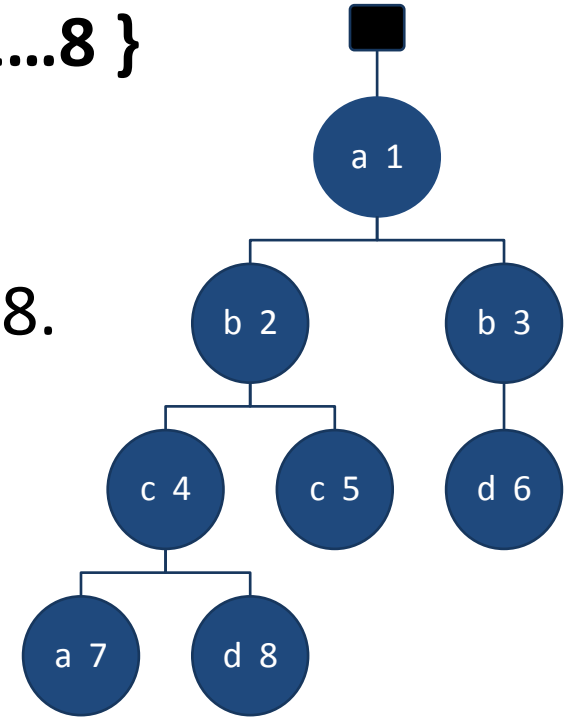
Efficient processing of XPath

…/ancestor-or-self::*[2] / …

The result of **ancestor-or-self::*** is **{ 1…8 }**

The context of node 2 is different if evaluated from node 4 or from node 8.

In the former case, **position** is 1.
In the latter case, **position** is 2.



Efficient processing of XPath

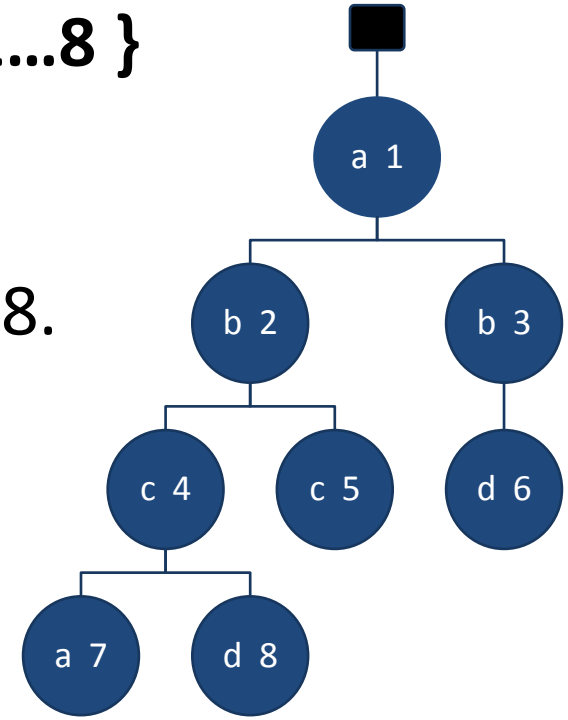
…/ancestor-or-self::*[2] / …

The result of **ancestor-or-self::*** is **{ 1…8 }**

The **context** of node 2 is **different** if evaluated from node 4 or from node 8.

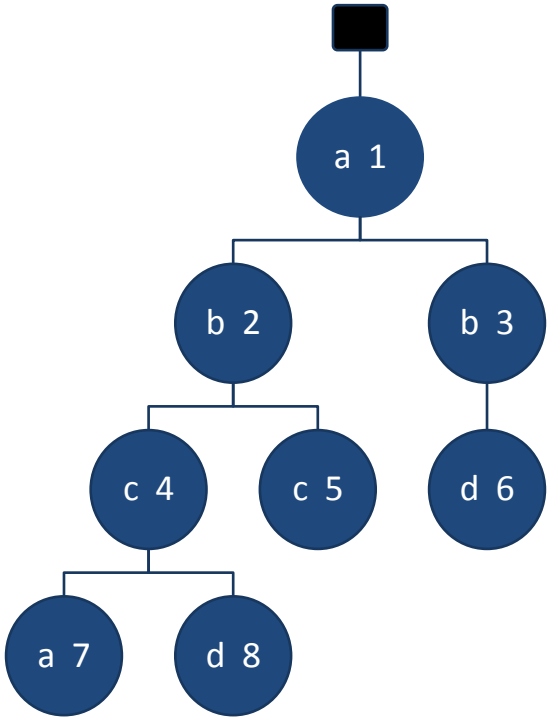
In the former case, **position** is 1.
In the latter case, **position** is 2.

The filter may keep or discard node 2 accordingly.



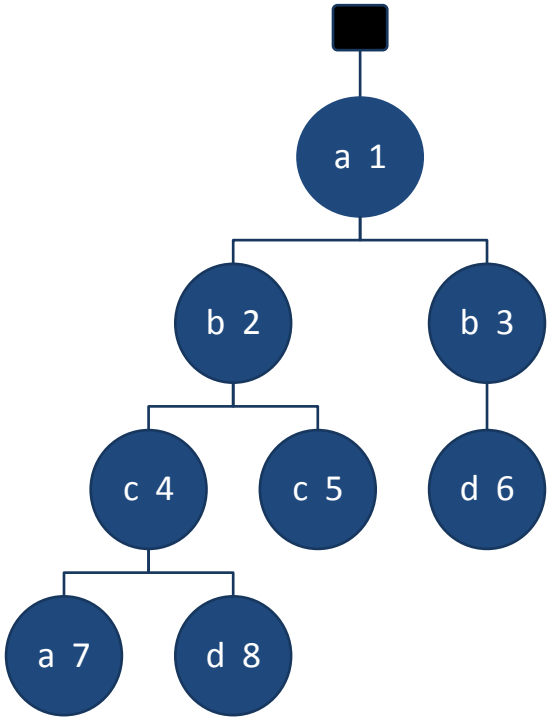
Efficient processing of XPath

…/ancestor-or-self::*[not(parent::*)] / …



Efficient processing of XPath

…/ancestor-or-self::*[empty(parent::*)] / …



Efficient processing of XPath

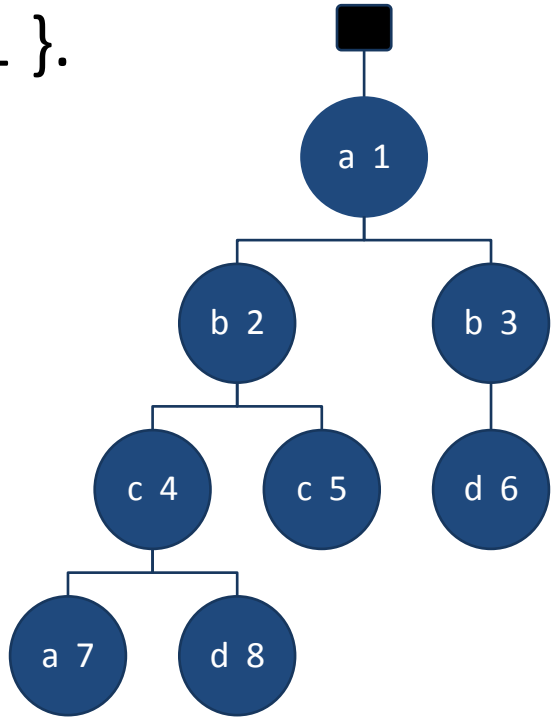
…/ancestor-or-self::*[empty(parent::*)] / …

For node 4, ancestor-or-self is { 4, 2, 1 }.

Add each context to a vector

(…, $\{4\}_4$, $\{2\}_4$, $\{1\}_4$, …)

Evaluate the filter for this vector



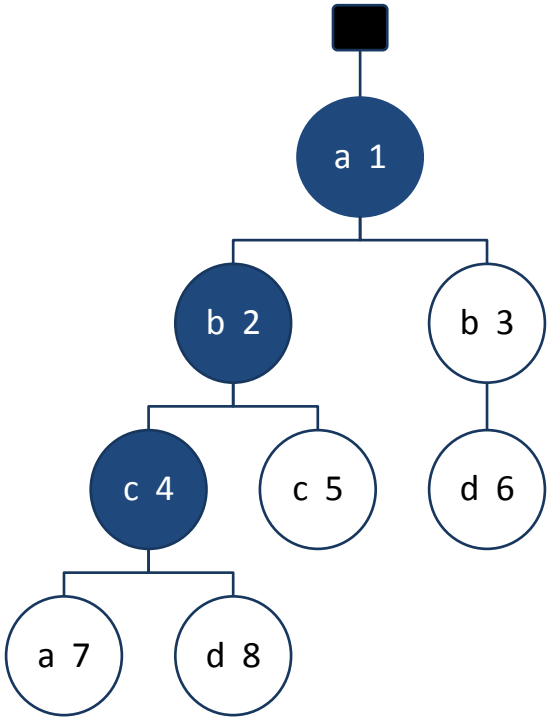
Efficient processing of XPath

…/ancestor-or-self::*[**empty**(parent::*)] / …

Compute **empty**(parent::*) for

(…, $\{4\}_4$, $\{2\}_4$, $\{1\}_4$, …)

Hence compute **parent::*** :



Efficient processing of XPath

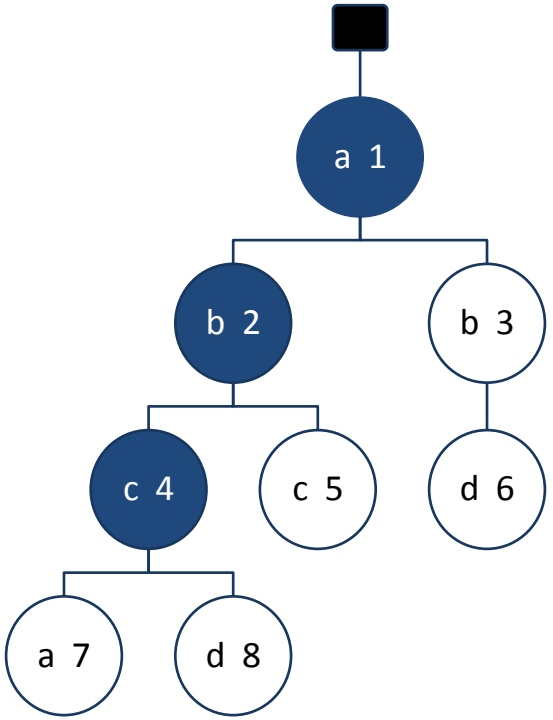
…/ancestor-or-self::*[**empty**(parent::*)] / …

Compute **empty**(parent::*) for

(…, $\{4\}_4$, $\{2\}_4$, $\{1\}_4$, …)

Hence compute **parent::*** :

(…, $\{2\}_4$, $\{1\}_4$, $\{\ \}_4$, …)



Efficient processing of XPath

…/ancestor-or-self::*[**empty**(parent::*)] / …

Compute **empty**(parent::*) for

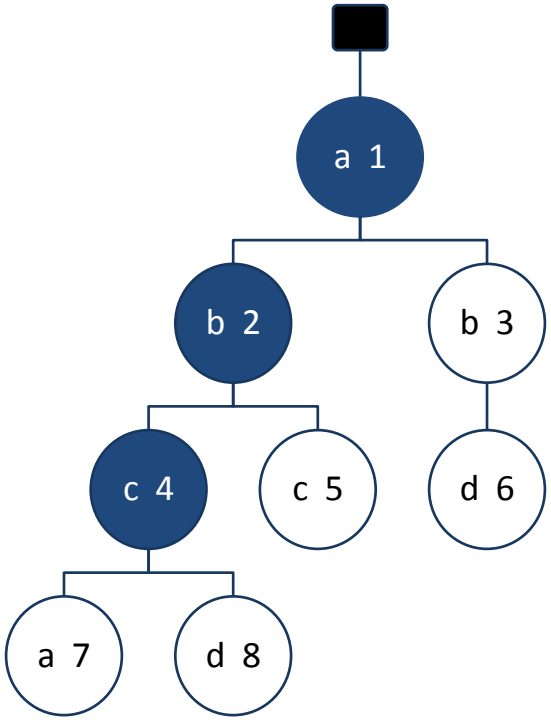
(…, $\{4\}_4$, $\{2\}_4$, $\{1\}_4$, …)

Hence compute **parent::*** :

(…, $\{2\}_4$, $\{1\}_4$, $\{\ \}_4$, …)

Resulting in

(…, F, F, T, …)



Efficient processing of XPath

…/ancestor-or-self::*[**empty**(parent::*)] / …

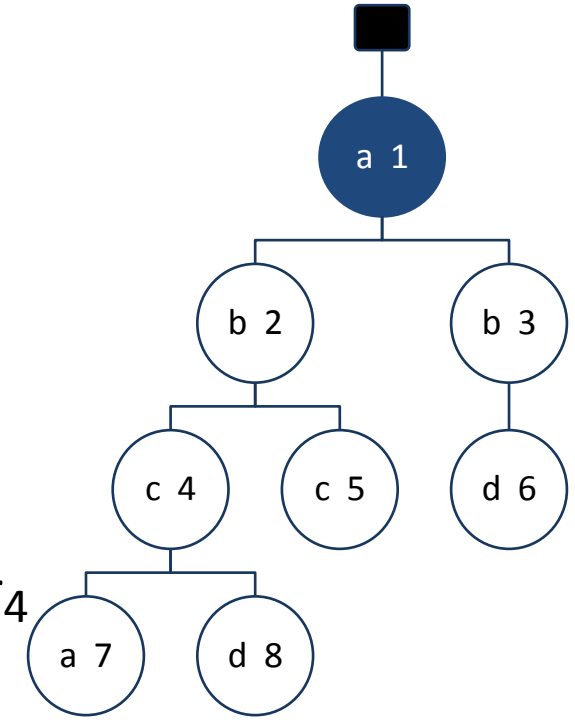
Compute **empty** for vector

(…, $\{4\}_4$, $\{2\}_4$, $\{1\}_4$, …)

Resulting in

(F, F, T)

We can therefore discard $\{4\}_4$ and $\{2\}_4$



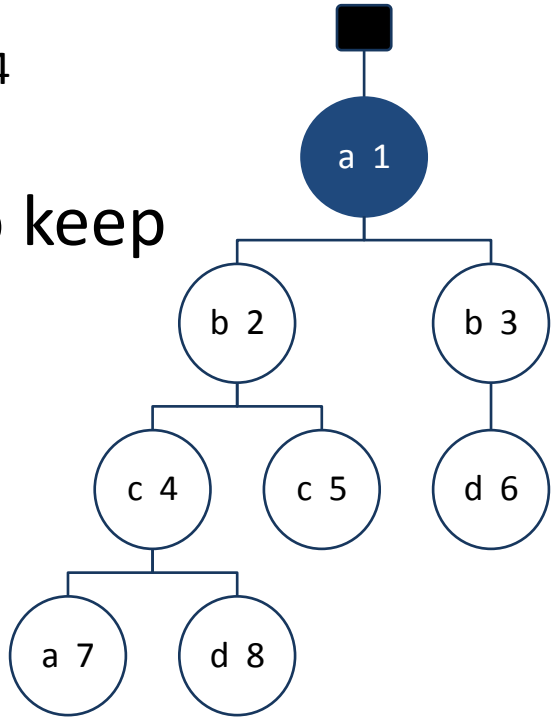
Efficient processing of XPath

…/ancestor-or-self::*[**empty**(parent::*)] / …

We can therefore discard $\{4\}_4$ and $\{2\}_4$

In other contexts we may have had to keep them.

In the end, we keep nodes that have survived in **at least one context**.



XML SCHEMA LANGUAGES

XML Schema

Cardinality

```
<xs:element name="a"
    minOccurs="4"
    maxOccurs="6"
 />
```

- Element a must appear between 4 and 6 times.

```
<xs:sequence
    minOccurs="0"
    maxOccurs="unbounded">
   ...
</xs:sequence>
```

- The whole sequence is optional. The number of occurences is however not restricted.

XML Schema

Sequence

```
<xs:sequence>
   <xs:element name=“a” />
   <xs:element name=“b” />
   <xs:element name=“c” />
</xs:sequence>
```

- a, b, and c in this precise order

XML Schema

Choice

```
<xs:choice>
   <xs:element name=“a” />
   <xs:element name=“b” />
   <xs:element name=“c” />
</xs:choice>
```

- One of a, b, or c

XML Schema

All

```
<xs:all>
   <xs:element name="a" />
   <xs:element name="b" />
   <xs:element name="c" />
</xs:all>
```

- a, b, and c, in any order

  maxOccurs must be ≤ 1

DTD

Binding XML to DTD

<!DOCTYPE *{root-element}*
SYSTEM '*{uri}*' [
{definitions}
]>

Element

<!ELEMENT *{name}* *{content-model}*>

Content models:

(#PCDATA | *{e1}* | *{e2}* | …)	**Mixed**
EMPTY	
ANY	
(*{e1}*, *{e2}*, *{e3}*, …)	**Sequence**
(*{e1}* | *{e2}* | *{e3}* | …)	**Choice**

Cardinality

?	0-1
*	0-inf
+	1-inf

Elements of sequence and choice can in turn be sequences or choices, with cardinality specifiers.

Mixed can be reduced to (#PCDATA) to only accept text.

Attribute list

<!ATTLIST {element}
{att1} {type1} {opt1}
{att2} {type2} {opt2} {def2}
…
{attn} {typen} {optn}
>

Type

CDATA	Any text
({v1}|{v2}|{v3})	List of values
NMTOKEN, NMTOKENS	(List of) XML names
ID	Unique identifier
IDREF	Reference to an ID
ENTITY, ENTITIES	

Options

#REQUIRED	
#IMPLIED	Optional
#FIXED	Cannot be changed

Exercices

- Validation tools

java -jar DTDValidation.jar *<xmldoc>*

java -jar XSDValidation.jar *<schema>* *<xmldoc>*

Deterministic Regular Expressions

1. For a regular expression *a*, define a' to be the regular expression obtained by replacing the i-th occurrence of symbol *s* by s_i
   - $a = (a \mid b)^+ cba^* (a \mid c)$
   - $a' = (a_1 \mid b_1)^+ c_1 b_2 a_2^* (a_3 \mid c_2)$
2. The regular expression is deterministic if there are no two strings wb_iv and wb_jz ($i \neq j$) in the regular language
   - Consider $a_1c_1b_2a_2a_3$ and $a_1c_1b_2a_3c_2$