

# XML and Databases

## **Exam Preparation (2)**

*Discuss Answers to last year's exam*

Sebastian Maneth  
NICTA and UNSW

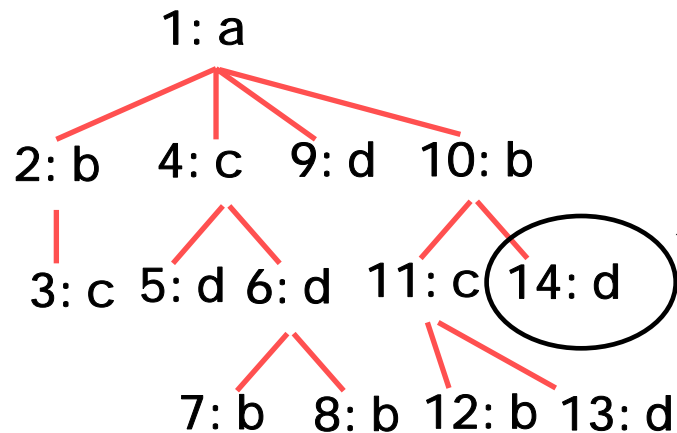
*CSE@UNSW -- Semester 1, 2008*

(9) For the tree given in 6, write XPath expressions that

- a) select all b nodes
- b) select all b nodes that have a c-child
- c) select all b nodes that have no c-children
- d) select the right most c-node
- e) select all nodes that have a c-parent

---

```
<a><b><c/></b><c><d/><d><b/><b/></d></c><d/><b><c><b/><d/></c><d/></b></a>
```



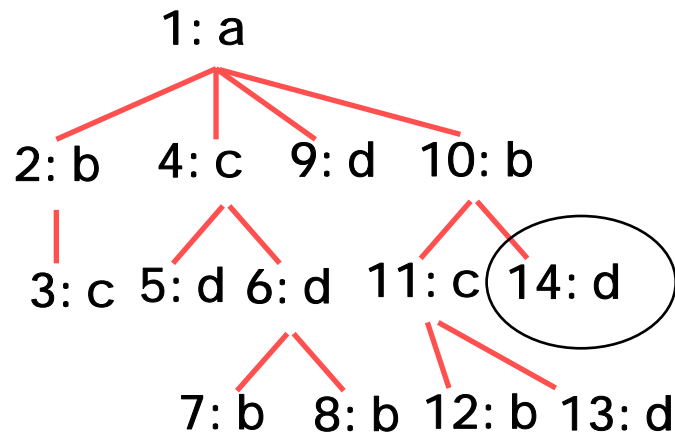
Watch out!  
This is a typo on your  
Exam printout... Sorry.

(9) For the tree given in 6, write XPath expressions that

- a) **select all b nodes**
- b) select all b nodes that have a c-child
- c) select all b nodes that have no c-children
- d) select the right most c-node
- e) select all nodes that have a c-parent

---

```
<a><b><c/></b><c><d/><d><b/><b/></d></c><d/><b><c><b/><d/></c><d/></b></a>
```



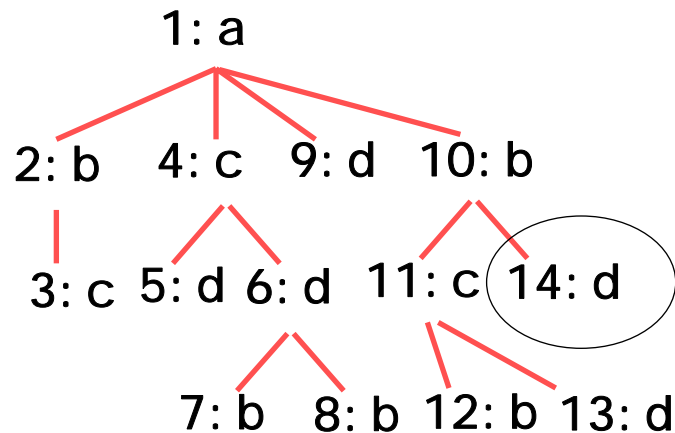
a) **//b**

(9) For the tree given in 6, write XPath expressions that

- a) select all b nodes
- b) select all b nodes that have a c-child**
- c) select all b nodes that have no c-children
- d) select the right most c-node
- e) select all nodes that have a c-parent

---

```
<a><b><c/></b><c><d/><d><b/><b/></d></c><d/><b><c><b/><d/></c><d/></b></a>
```



a) //b

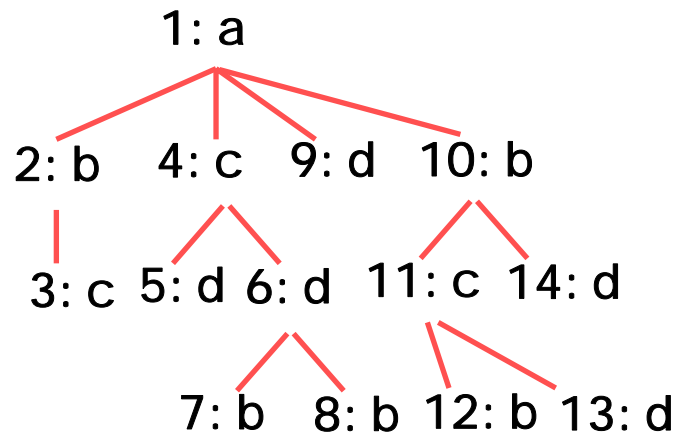
**b) //b[c]**

(9) For the tree given in 6, write XPath expressions that

- a) select all b nodes
- b) select all b nodes that have a c-child
- c) select all b nodes that have no c-children
- d) select the right most c-node
- e) select all nodes that have a c-parent

---

```
<a><b><c/></b><c><d/><d><b/><b/></d></c><d/><b><c><b/><d/></c><d/></b></a>
```



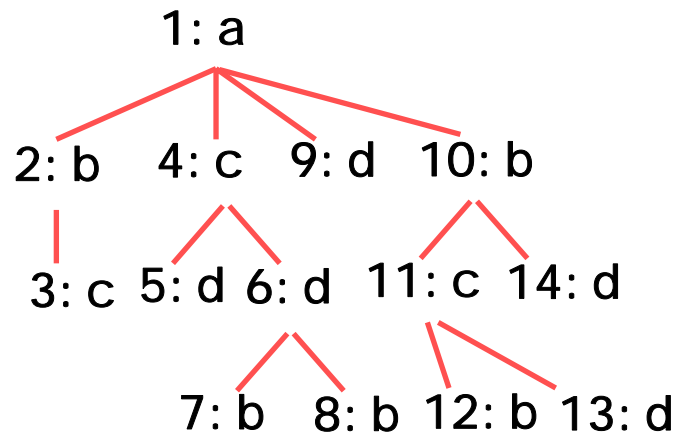
- a) //b
- b) //b[c]
- c) //b[not(c)]

(9) For the tree given in 6, write XPath expressions that

- a) select all b nodes
- b) select all b nodes that have a c-child
- c) select all b nodes that have no c-children
- d) select the right most c-node (node no 11)**
- e) select all nodes that have a c-parent

---

```
<a><b><c/></b><c><d/><d><b/><b/></d></c><d/><b><c><b/><d/></c><d/></b></a>
```



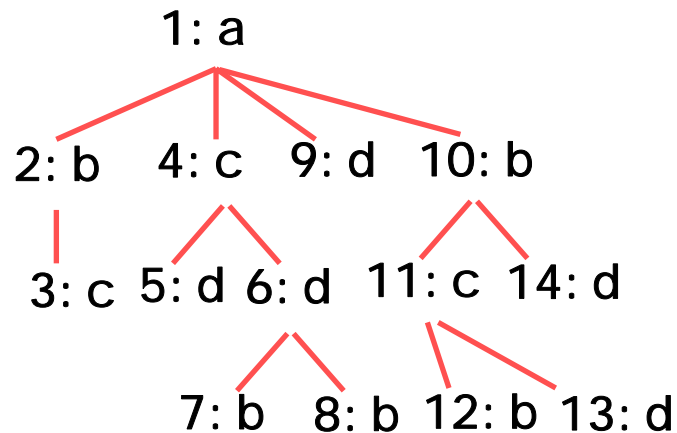
- a) //b
- b) //b[c]
- c) //b[not(c)]
- d) //c[b]**  
or **//c[b and d]**

(9) For the tree given in 6, write XPath expressions that

- a) select all b nodes
- b) select all b nodes that have a c-child
- c) select all b nodes that have no c-children
- d) select the right most c-node (node no 11)
- e) **select all nodes that have a c-parent**

---

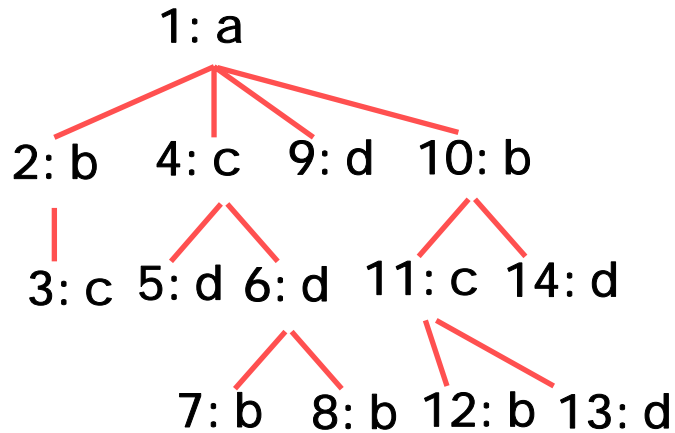
```
<a><b><c/></b><c><d/><d><b/><b/></d></c><d/><b><c><b/><d/></c><d/></b></a>
```



- a) //b
- b) //b[c]
- c) //b[not(c)]
- d) //c[b]
- or //c[b and d]
- e) **//\*[parent::c]**
- or **//c/\***

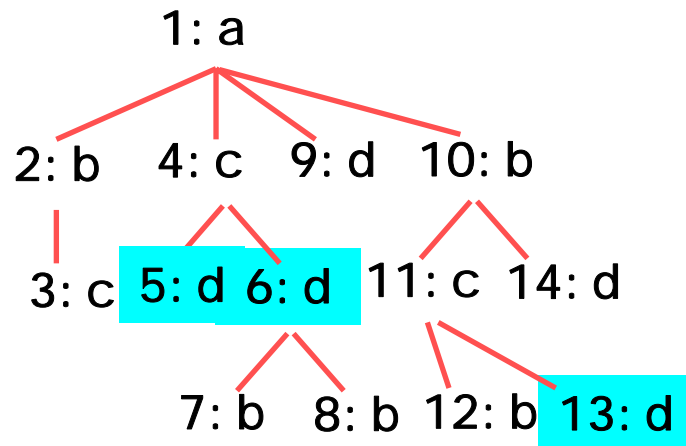
(10) Below is a tree corresponding to the document in (6). Show the sequences of node numbers that are selected by the following queries.

- a) //c//d
- b) //\*[a or b]
- c) //b/ancestor::d/following::d
- d) //\*[not(.//b | ./ancestor::c)]
- e) //c//d/preceding::\*//d



(10) Below is a tree corresponding to the document in (6).  
Show the sequences of node numbers that are selected by the following queries.

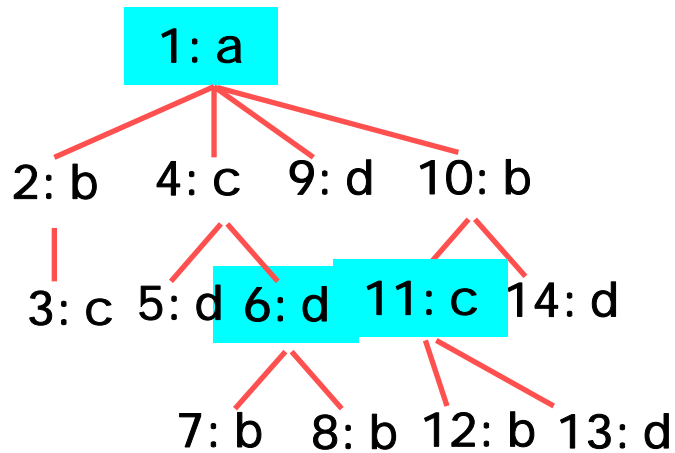
- a) `//c//d`
- b) `//*[a or b]`
- c) `//b/ancestor::d/following::d`
- d) `//*[not(.//b | ./ancestor::c)]`
- e) `//c//d/preceding::*//d`



a) 5, 6, 13

(10) Below is a tree corresponding to the document in (6).  
Show the sequences of node numbers that are selected by the following queries.

- a) //c//d
- b) **//\*[a or b]**
- c) //b/ancestor::d/following::d
- d) **//\*[not(.//b | ./ancestor::c)]**
- e) //c//d/preceding::\*//d

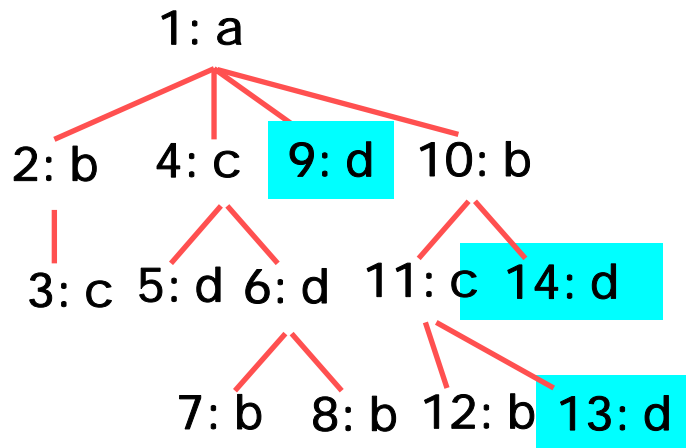


a) 5, 6, 13

b) **1, 6, 11**

(10) Below is a tree corresponding to the document in (6).  
Show the sequences of node numbers that are selected by the following queries.

- a) //c//d
- b) //\*[a or b]
- c) //b/ancestor::d/following::d
- d) //\*[not(.//b | ./ancestor::c)]
- e) //c//d/preceding::\*//d



a) 5, 6, 13

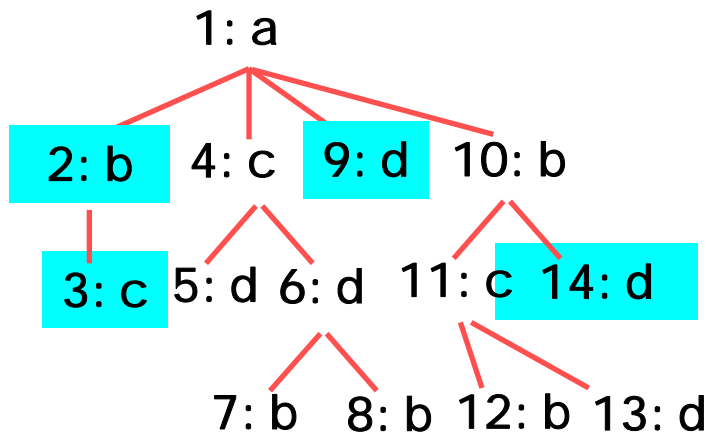
b) 1, 6, 11

c) 9, 13, 14

(10) This is a tree corresponding to the XML in (6).

Show the sequences of node numbers that are selected by the following queries.

- a) //c//d
- b) //\*[a or b]
- c) //b/ancestor::d/following::d
- d) //\*[not(.//b | ./ancestor::c)]
- e) //c//d/preceding::\*//d



a) 5, 6, 13

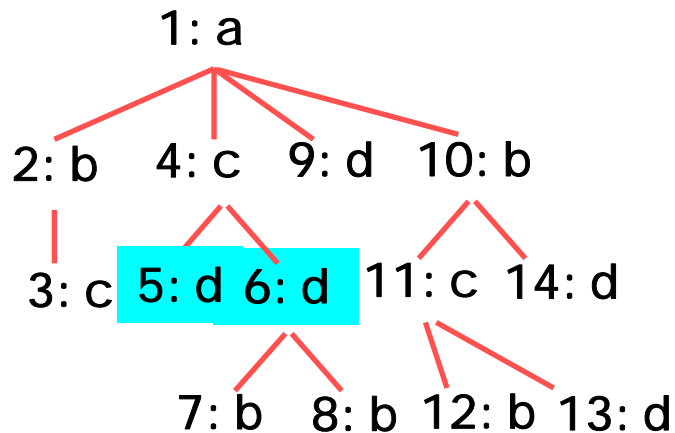
b) 1, 6, 11

c) 9, 13, 14

d) 2, 3, 9, 14

(10) This is a tree corresponding to the XML in (6).  
Show the sequences of node numbers that are selected by the following queries.

- a) `//c//d`
- b) `//*[a or b]`
- c) `//b/ancestor::d/following::d`
- d) `//*[not(.//b | ./ancestor::c)]`
- e) `//c//d/preceding::*//d`



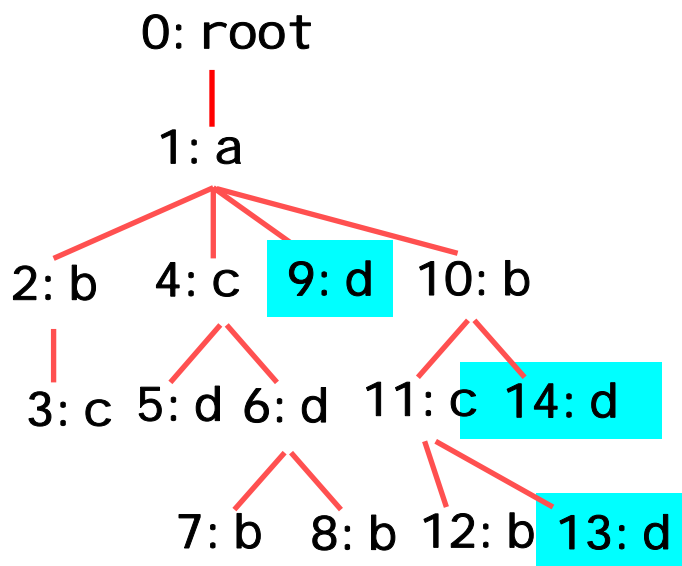
- a) 5, 6, 13
- b) 1, 6, 11
- c) 9, 13, 14
- d) 2, 3, 9, 14
- e) 5, 6

(10) This is a tree corresponding to the XML in (6).

c) `//b/ancestor::d/following::d`

d) `//*[not(.//b | ./ancestor::c)]`

For **query c)** show in detail how the Core-XPath evaluation algorithm computes the answer to this query. Do the same for query d).

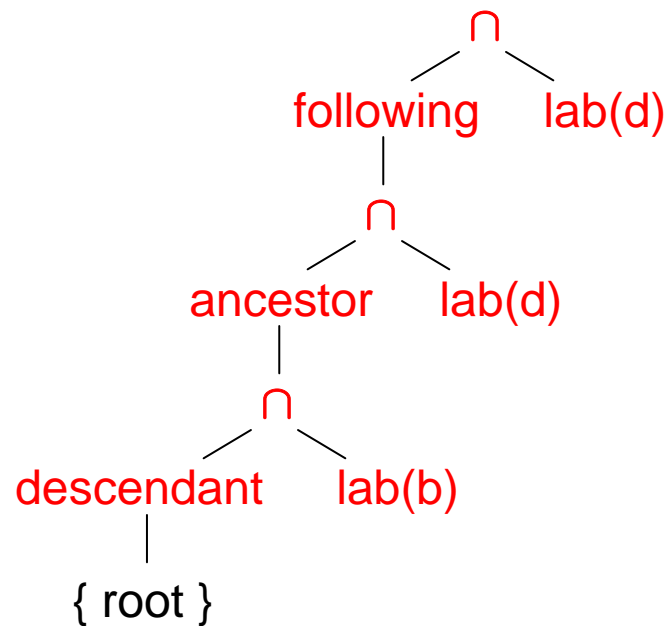


$\text{lab}(a) = \{ 1 \}$

$\text{lab}(b) = \{ 2, 7, 8, 10, 12 \}$

$\text{lab}(c) = \{ 3, 4, 11 \}$

$\text{lab}(d) = \{ 5, 6, 9, 13, 14 \}$

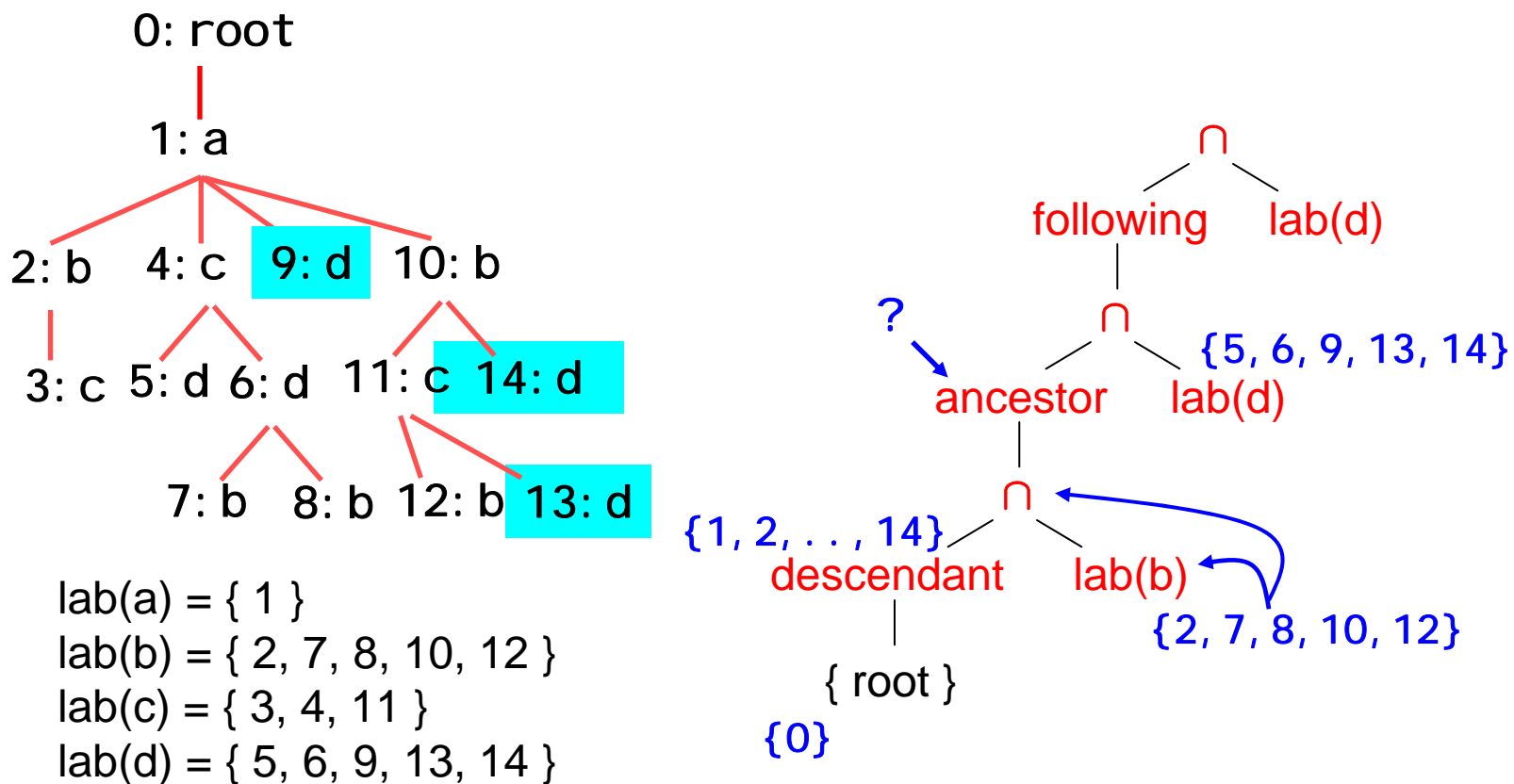


(10) This is a tree corresponding to the XML in (6).

c) `//b/ancestor::d/following::d`

d) `//*[not(.//b | ./ancestor::c)]`

For **query c)** show in detail how the Core-XPath evaluation algorithm computes the answer to this query. Do the same for query d).

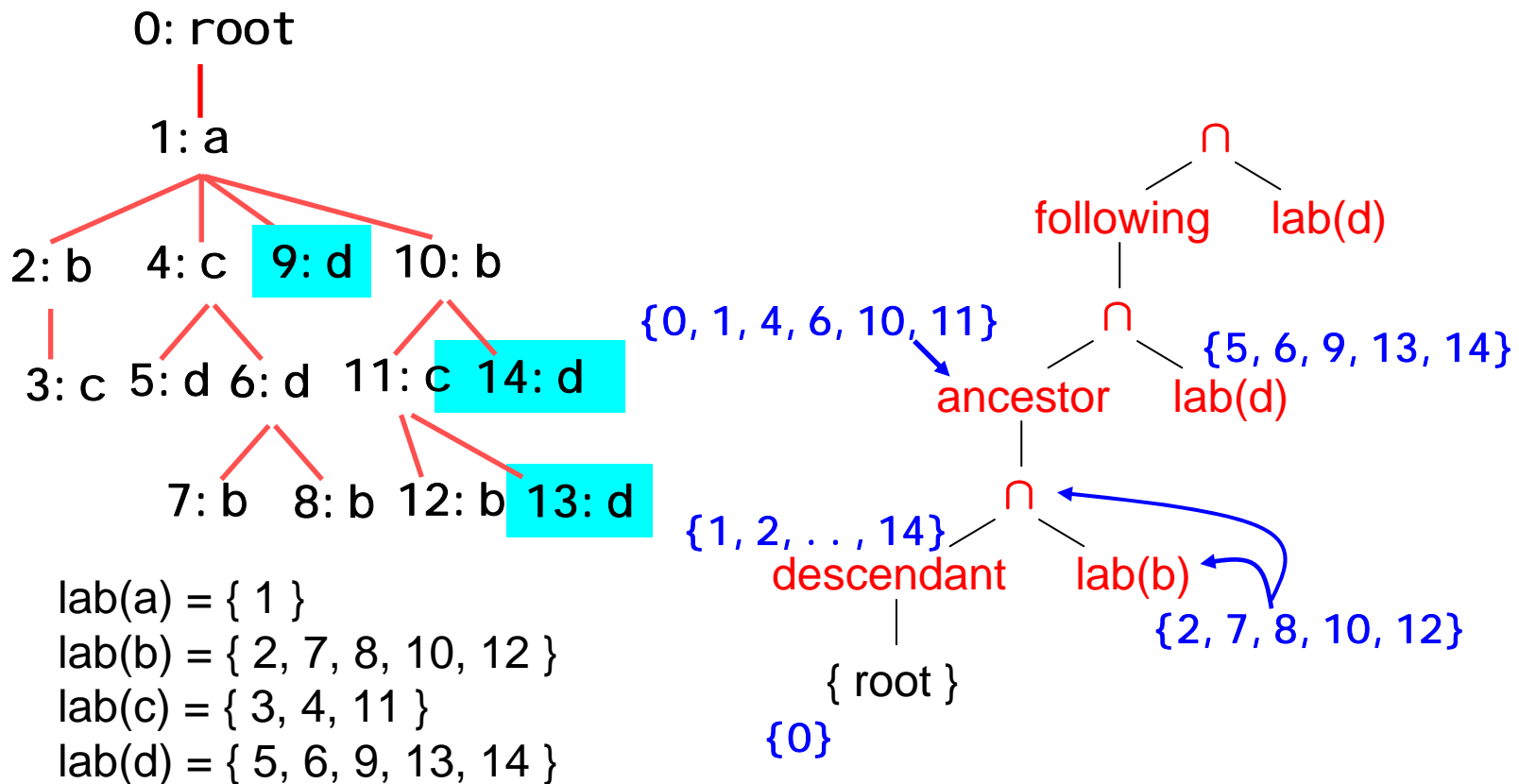


(10) This is a tree corresponding to the XML in (6).

c) `//b/ancestor::d/following::d`

d) `//*[not(.//b | ./ancestor::c)]`

For **query c)** show in detail how the Core-XPath evaluation algorithm computes the answer to this query. Do the same for query d).

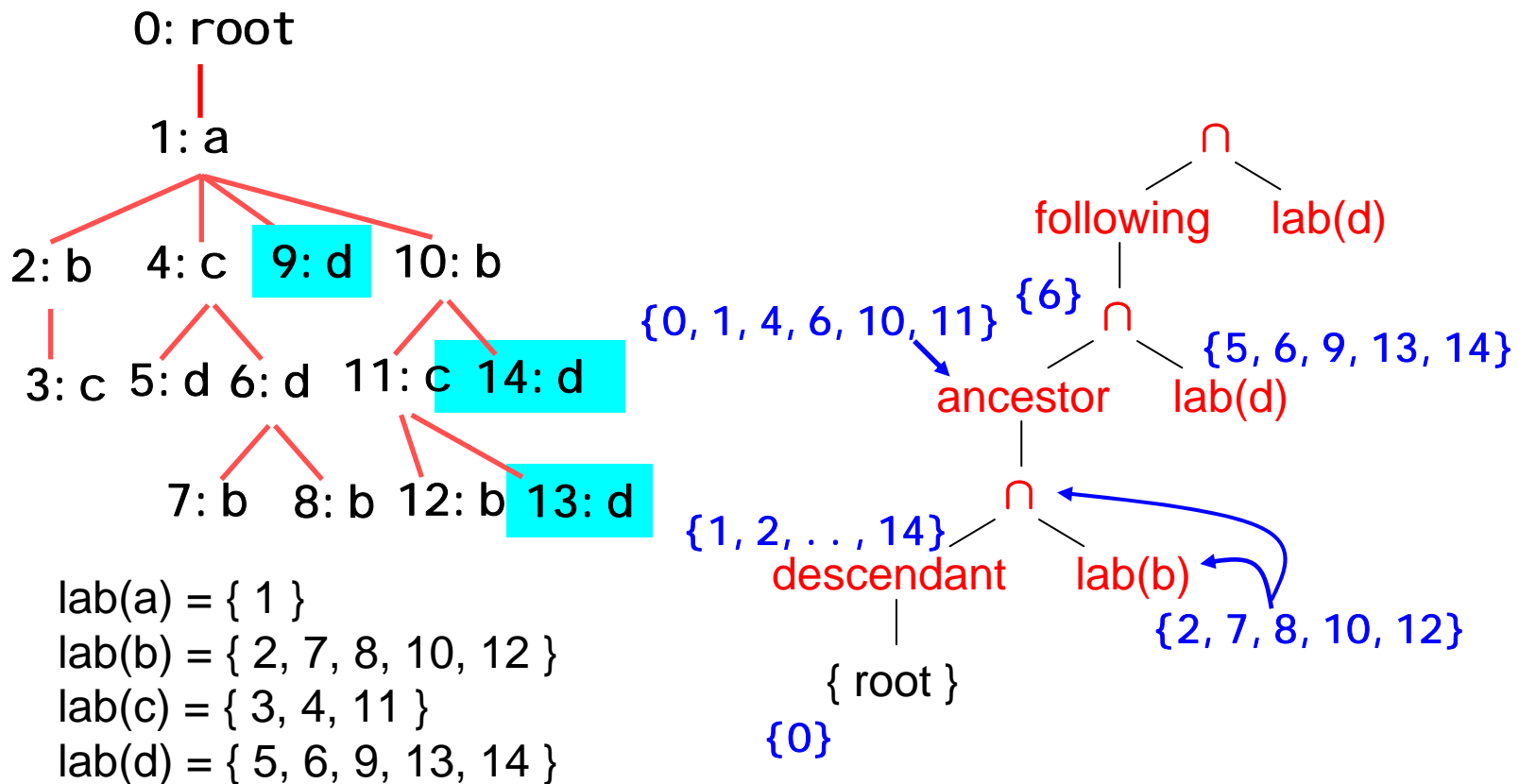


(10) This a tree corresponding to the XML in (6).

c) `//b/ancestor::d/following::d`

d) `//*[not(.//b | ./ancestor::c)]`

For **query c)** show in detail how the Core-XPath evaluation algorithm computes the answer to this query. Do the same for query d).

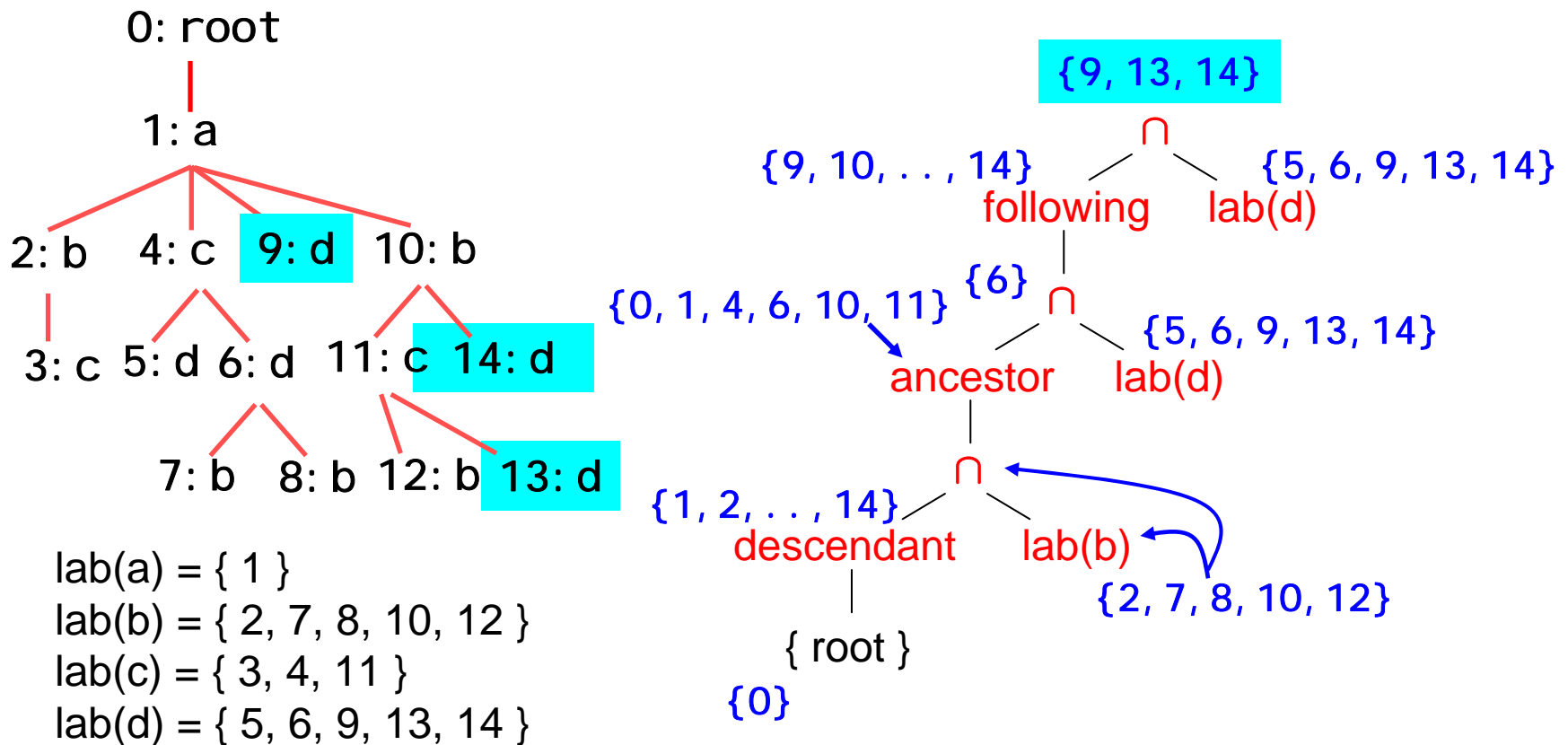


(10) This a tree corresponding to the XML in (6).

c) `//b/ancestor::d/following::d`

d) `//*[not(.//b | ./ancestor::c)]`

For **query c)** show in detail how the Core-XPath evaluation algorithm computes the answer to this query. Do the same for query d).



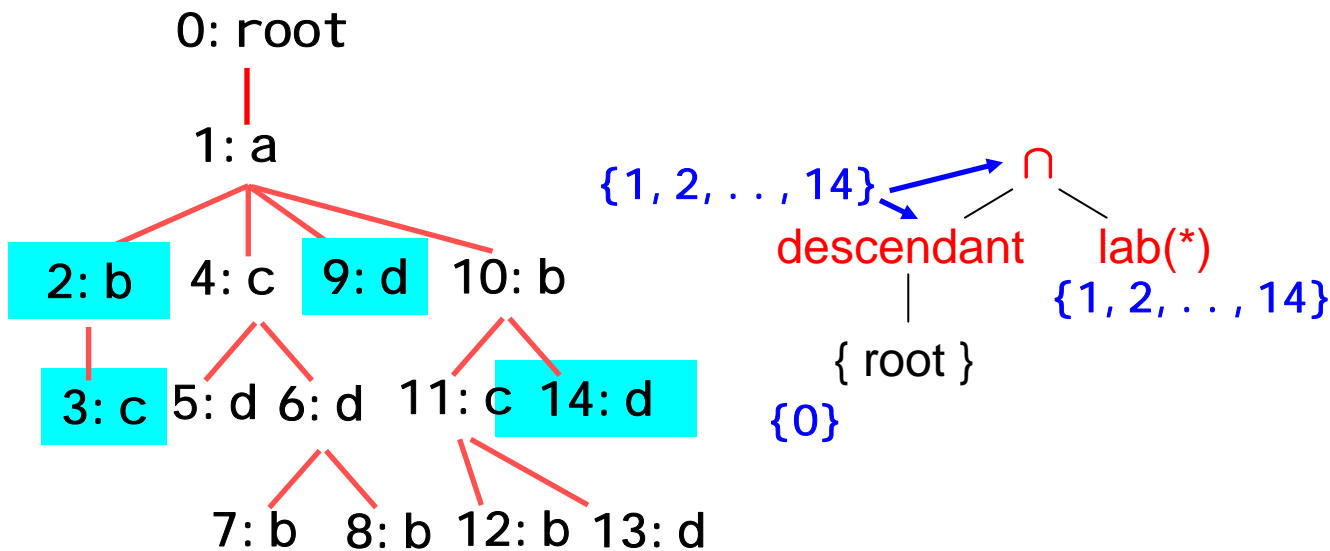
(10) This is a tree corresponding to the XML in (6).

c) `//b/ancestor::d/following::d`

d) `//*[not(.//b | ./ancestor::c)]`

For query c) show in detail how the Core-XPath evaluation algorithm computes the answer to this query.

Do the same for **query d**).



$\text{lab}(a) = \{ 1 \}$

$\text{lab}(b) = \{ 2, 7, 8, 10, 12 \}$

$\text{lab}(c) = \{ 3, 4, 11 \}$

$\text{lab}(d) = \{ 5, 6, 9, 13, 14 \}$

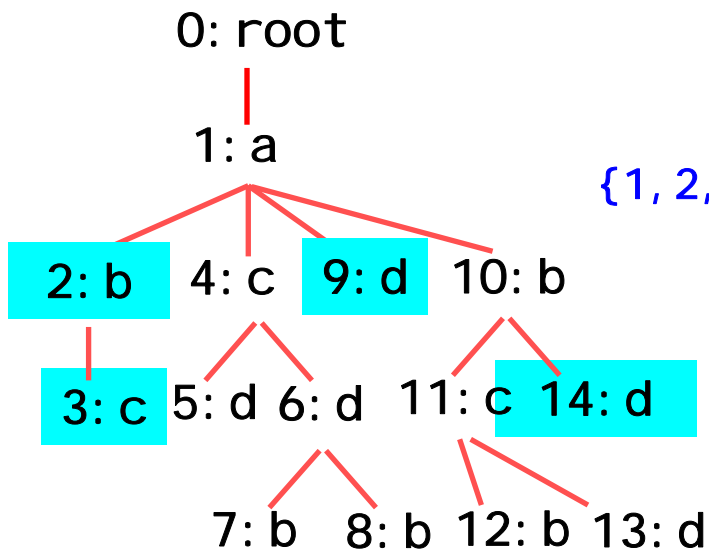
(10) This a tree corresponding to the XML in (6).

c) `//b/ancestor::d/following::d`

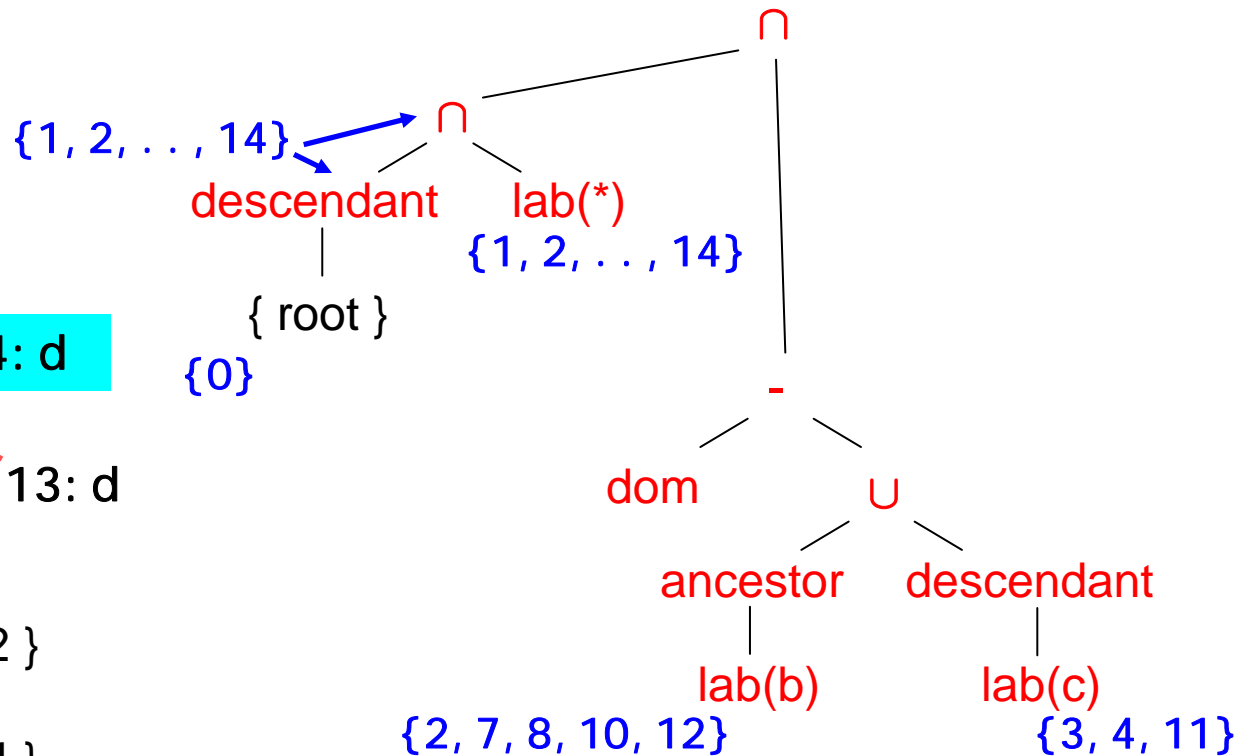
d) `//*[not(.//b | ./ancestor::c)]`

For query c) show in detail how the Core-XPath evaluation algorithm computes the answer to this query.

Do the same for **query d**).



$lab(a) = \{ 1 \}$   
 $lab(b) = \{ 2, 7, 8, 10, 12 \}$   
 $lab(c) = \{ 3, 4, 11 \}$   
 $lab(d) = \{ 5, 6, 9, 13, 14 \}$



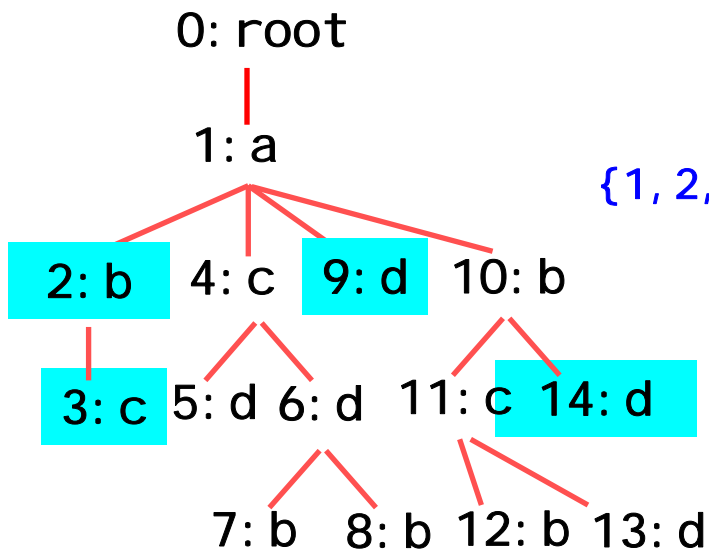
(10) This is a tree corresponding to the XML in (6).

c) `//b/ancestor::d/following::d`

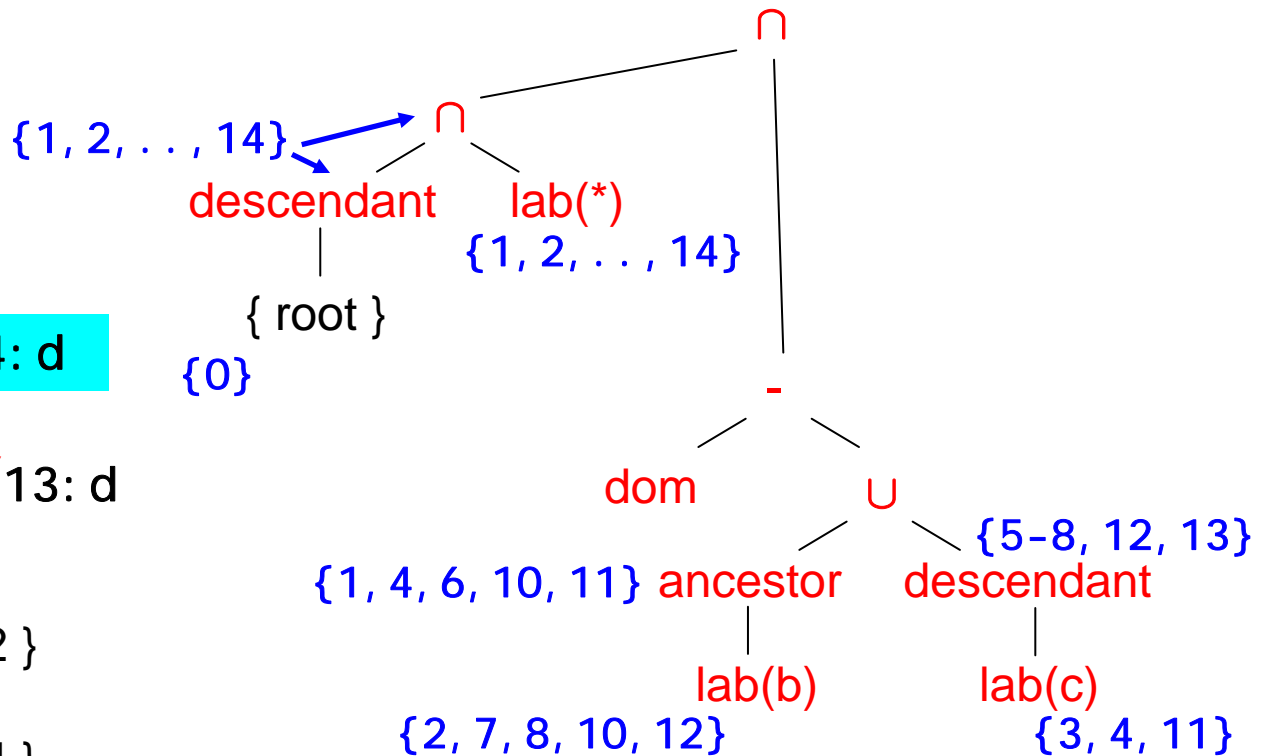
d) `//*[not(.//b | ./ancestor::c)]`

For query c) show in detail how the Core-XPath evaluation algorithm computes the answer to this query.

Do the same for query d).



$lab(a) = \{ 1 \}$   
 $lab(b) = \{ 2, 7, 8, 10, 12 \}$   
 $lab(c) = \{ 3, 4, 11 \}$   
 $lab(d) = \{ 5, 6, 9, 13, 14 \}$



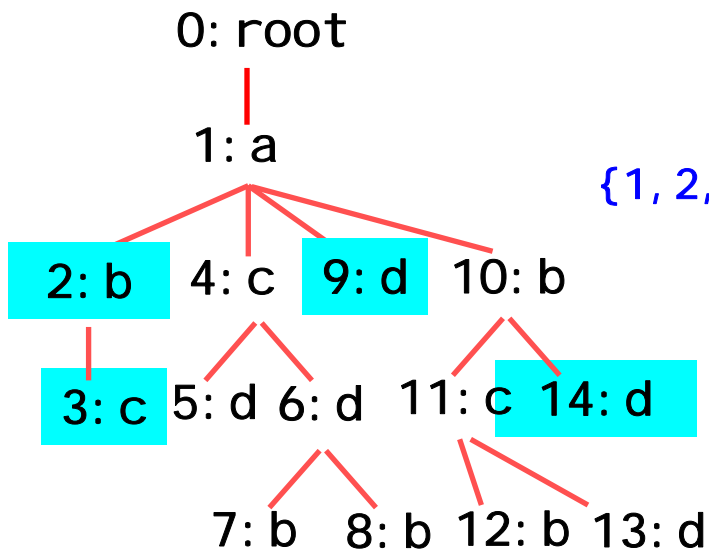
(10) This is a tree corresponding to the XML in (6).

c) `//b/ancestor::d/following::d`

d) `//*[not(.//b | ./ancestor::c)]`

For query c) show in detail how the Core-XPath evaluation algorithm computes the answer to this query.

Do the same for query d).

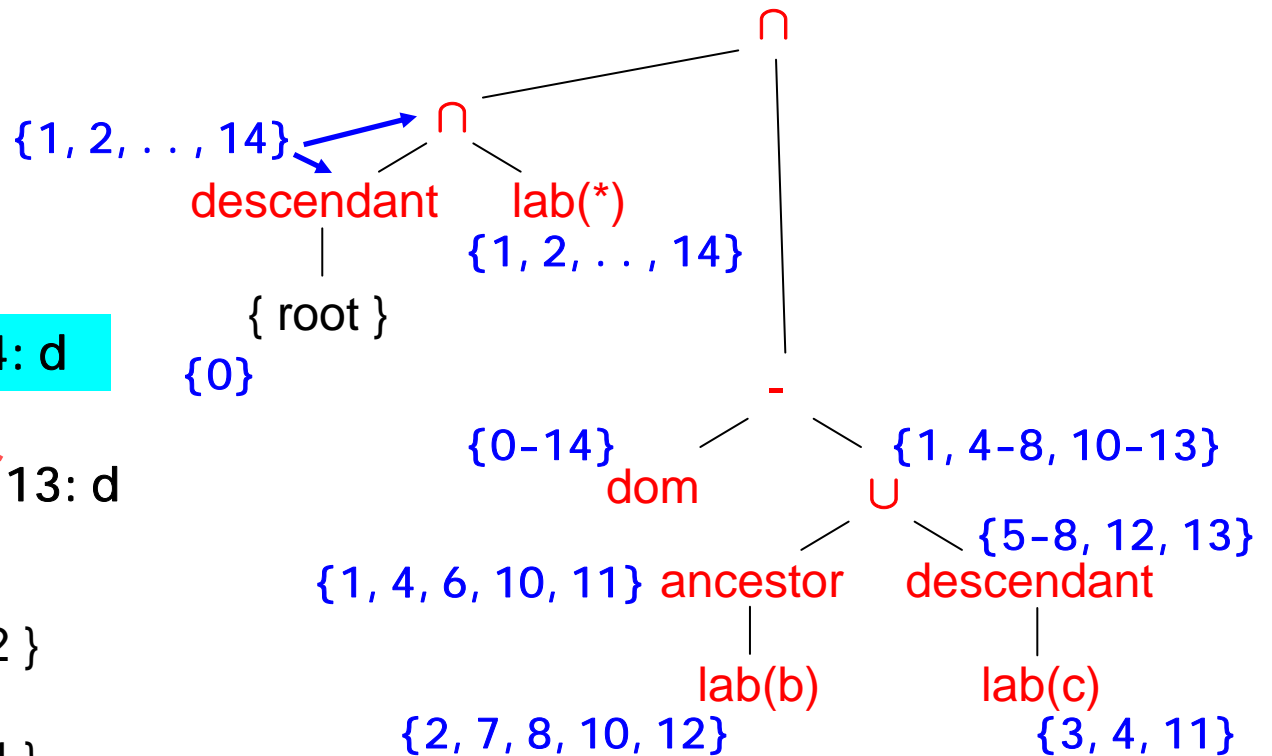


$lab(a) = \{ 1 \}$

$lab(b) = \{ 2, 7, 8, 10, 12 \}$

$lab(c) = \{ 3, 4, 11 \}$

$lab(d) = \{ 5, 6, 9, 13, 14 \}$



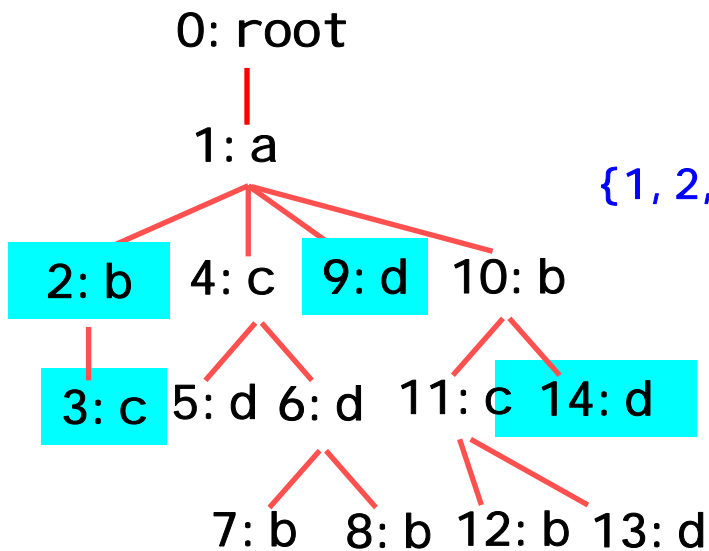
(10) This is a tree corresponding to the XML in (6).

c) `//b/ancestor::d/following::d`

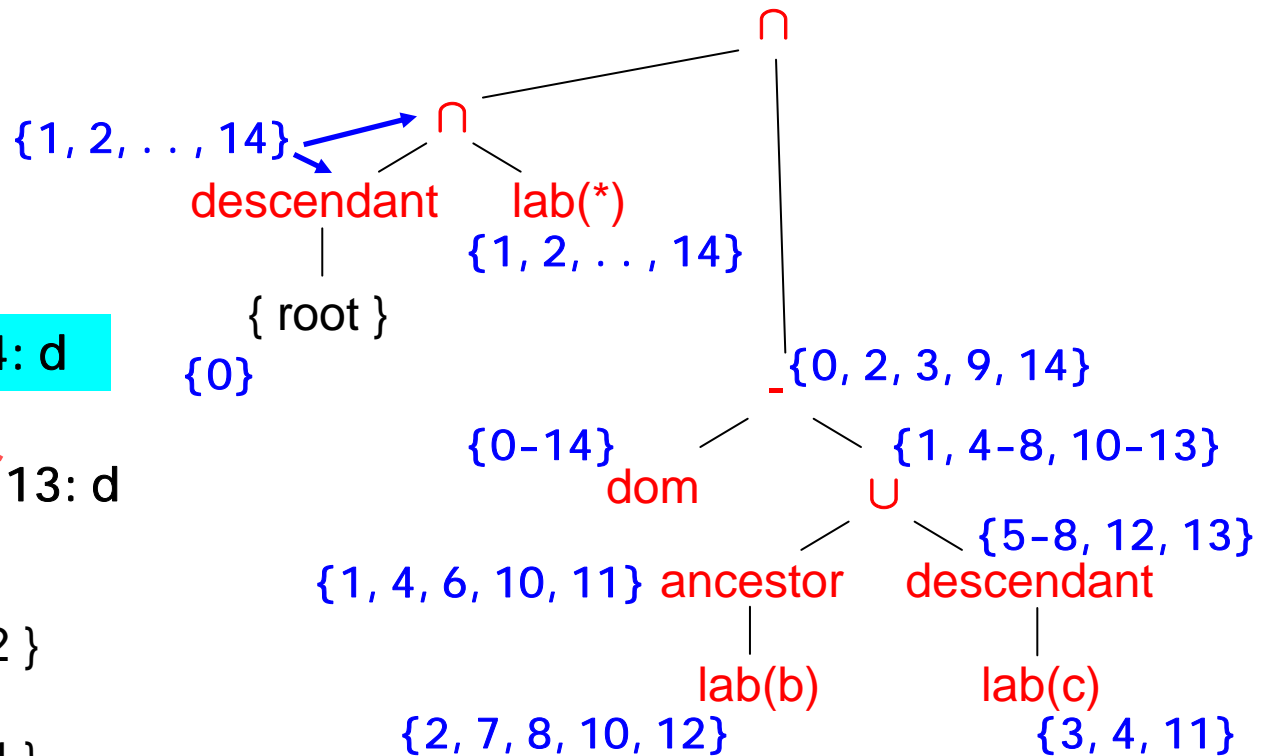
d) `//*[not(.//b | ./ancestor::c)]`

For query c) show in detail how the Core-XPath evaluation algorithm computes the answer to this query.

Do the same for query d).



$lab(a) = \{ 1 \}$   
 $lab(b) = \{ 2, 7, 8, 10, 12 \}$   
 $lab(c) = \{ 3, 4, 11 \}$   
 $lab(d) = \{ 5, 6, 9, 13, 14 \}$



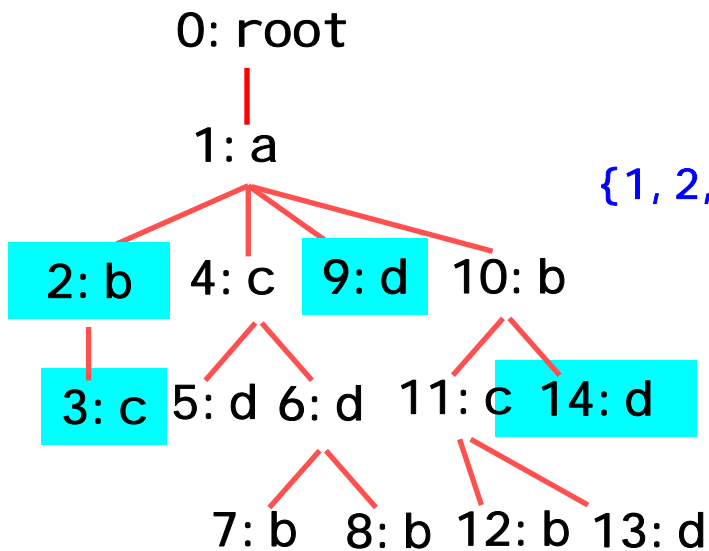
(10) This is a tree corresponding to the XML in (6).

c) `//b/ancestor::d/following::d`

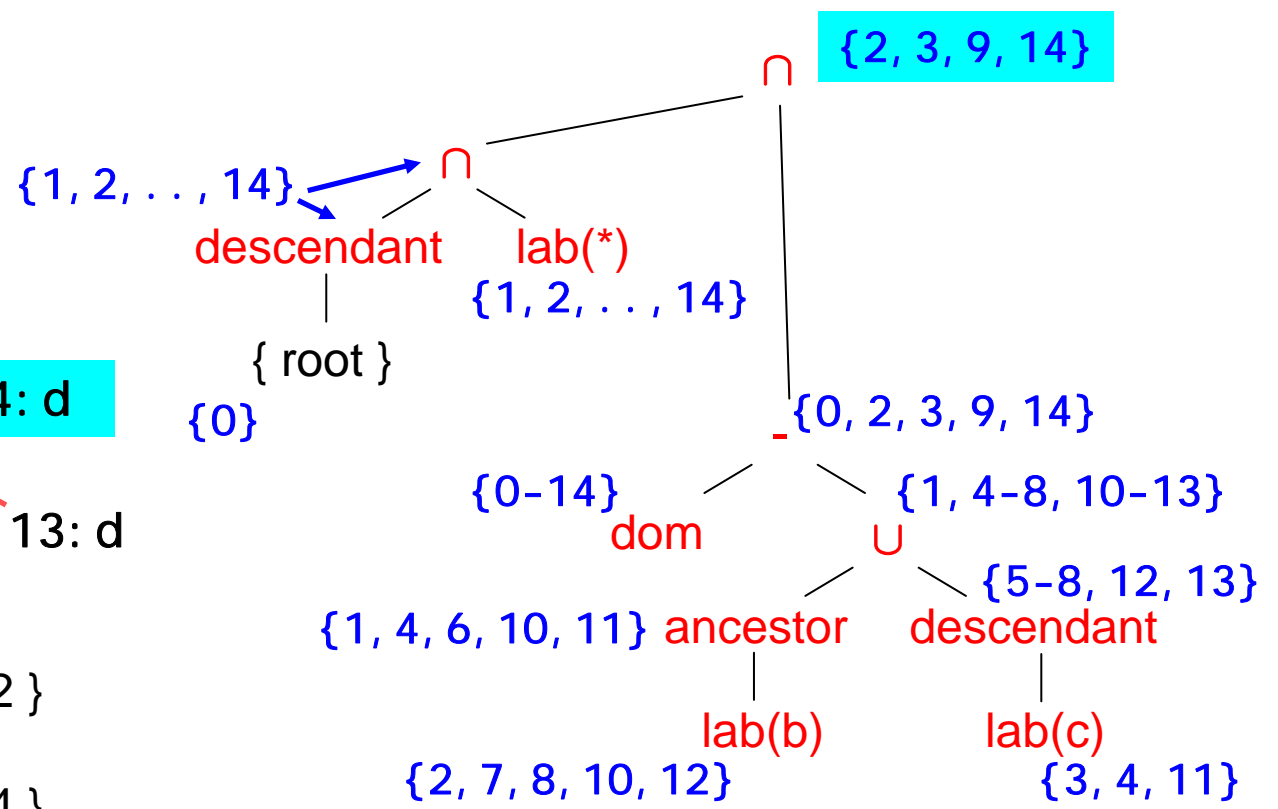
d) `//*[not(.//b | ./ancestor::c)]`

For query c) show in detail how the Core-XPath evaluation algorithm computes the answer to this query.

Do the same for query d).



- lab(a) = { 1 }
- lab(b) = { 2, 7, 8, 10, 12 }
- lab(c) = { 3, 4, 11 }
- lab(d) = { 5, 6, 9, 13, 14 }



(11) Show an example of XPath queries  $q_1, q_2$  such that they are not equivalent, but  $q_1$  is included in  $q_2$ .

Show that  $q_1$  is included in  $q_2$  using one of the methods discussed.

Use the homomorphism technique to test whether

$p = a[.//b[c/*//d]/b[c//d]/b[c/d]]$  is included in

$q = a[.//b[c//*//d]/*[c/d]]$

---

(11) Show an example of XPath queries  $q_1, q_2$  such that they are not equivalent, but  $q_1$  is included in  $q_2$ .

Show that  $q_1$  is included in  $q_2$  using one of the methods discussed.

Use the homomorphism technique to test whether

$p=a[.//b[c/*//d]/b[c//d]/b[c/d]]$  is included in

$q=a[.//b[c//*//d]/*[c/d]]$

---

$q_1 = /b//b$

$q_2 = //b$

or

$q_1 = //b$

$q_2 = //*$

(11) Show an example of XPath queries q1, q2 such that they are not equivalent, but q1 is included in q2.

Show that q1 is included in q2 using one of the methods discussed.

Use the homomorphism technique to test whether

p=a[.//b[c/\*//d]/b[c//d]/b[c/d]] is included in

q=a[.//b[c//\*//d]/\*[c/d]]

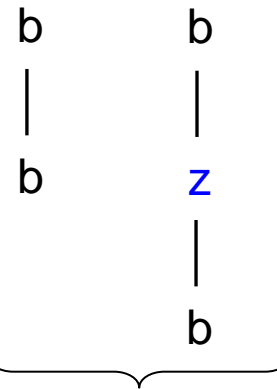
q1 = /b//b

q2 = //b

Canonical model :

Replace in q1 // by  
 $\leq N+1$  many /z/

0, here



Do they match q2? → yes!!

or

q1 = //b

q2 = //\*

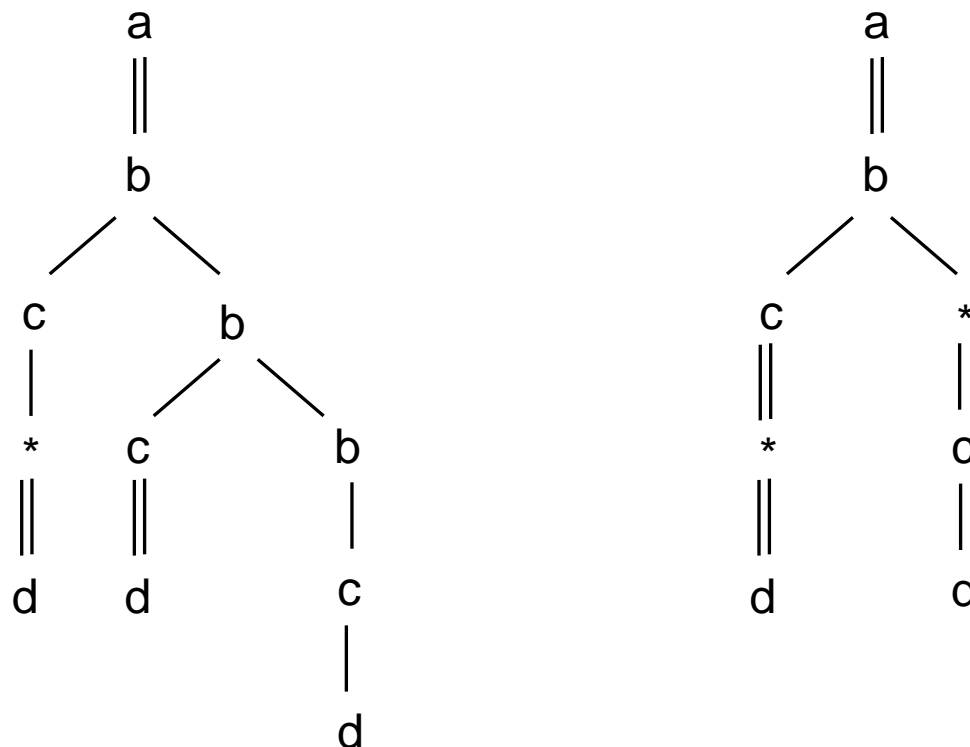
(11) Show an example of XPath queries  $q_1, q_2$  such that they are not equivalent, but  $q_1$  is included in  $q_2$ .

Show that  $q_1$  is included in  $q_2$  using one of the methods discussed.

Use the homomorphism technique to test whether

$p = a[.//b[c/*//d]/b[c//d]/b[c/d]]$  is included in

$q = a[.//b[c//*//d]/*[c/d]]$



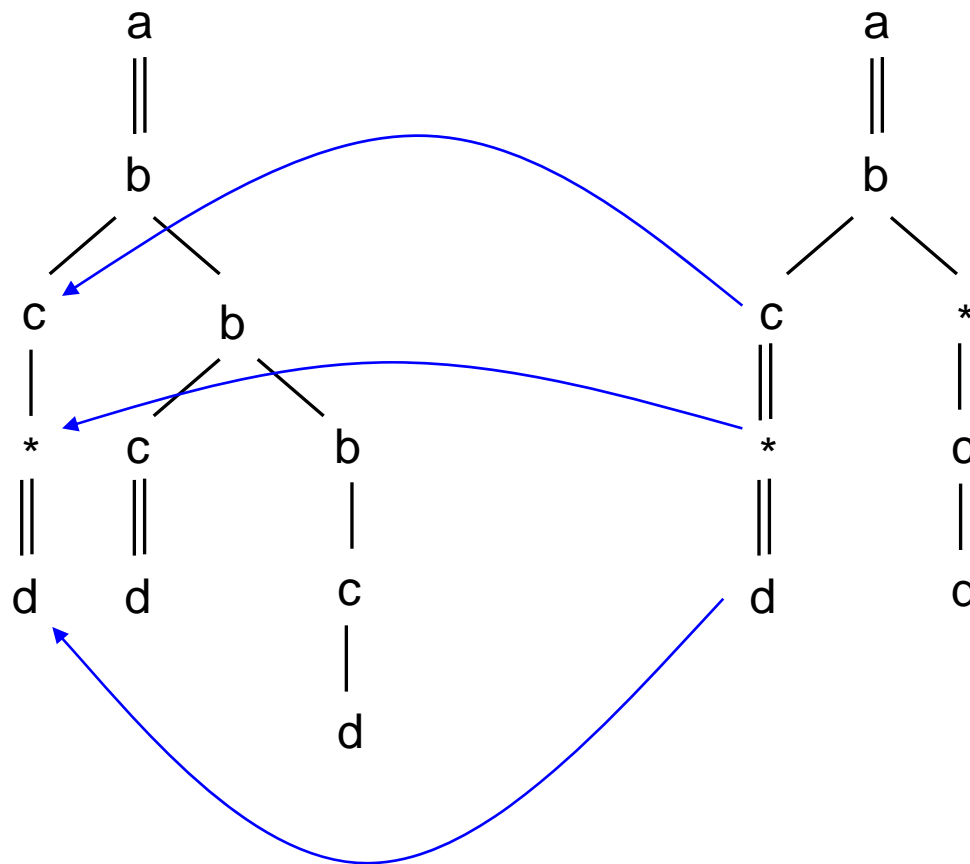
(11) Show an example of XPath queries  $q_1, q_2$  such that they are not equivalent, but  $q_1$  is included in  $q_2$ .

Show that  $q_1$  is included in  $q_2$  using one of the methods discussed.

Use the homomorphism technique to test whether

$p = a[.//b[c/*//d]/b[c//d]/b[c/d]]$  is included in

$q = a[.//b[c//*//d]/* [c/d]]$



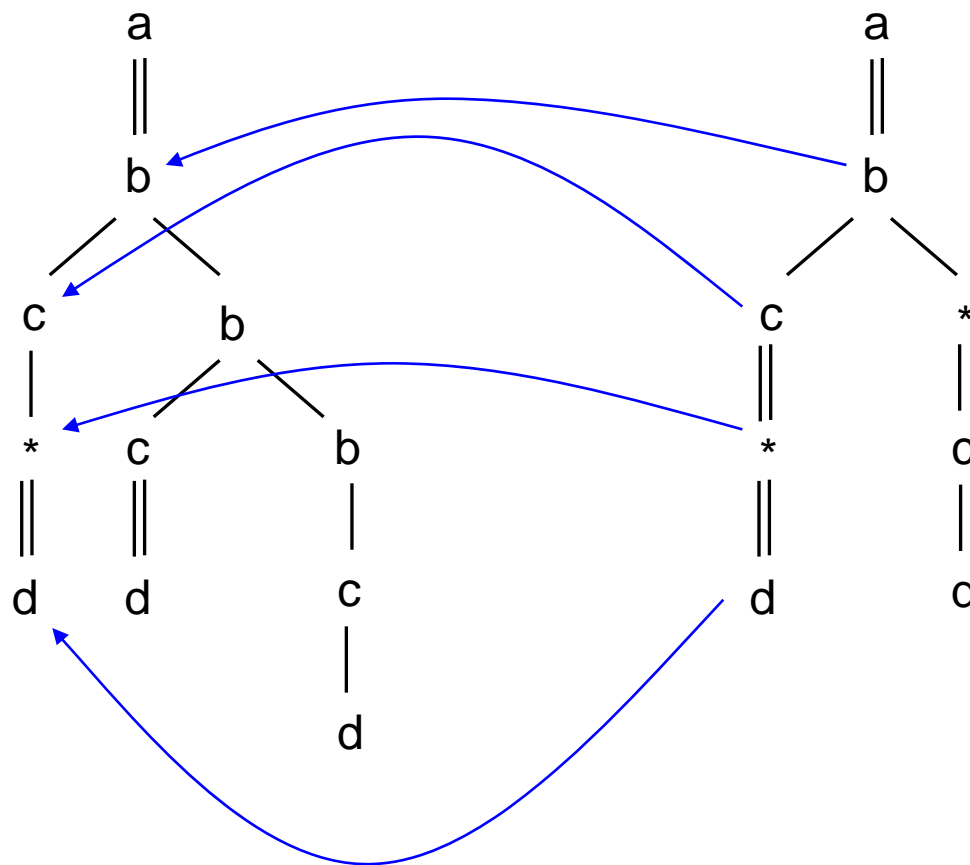
(11) Show an example of XPath queries  $q_1, q_2$  such that they are not equivalent, but  $q_1$  is included in  $q_2$ .

Show that  $q_1$  is included in  $q_2$  using one of the methods discussed.

Use the homomorphism technique to test whether

$p = a[.//b[c/*//d]/b[c//d]/b[c/d]]$  is included in

$q = a[.//b[c//*//d]/*[c/d]]$



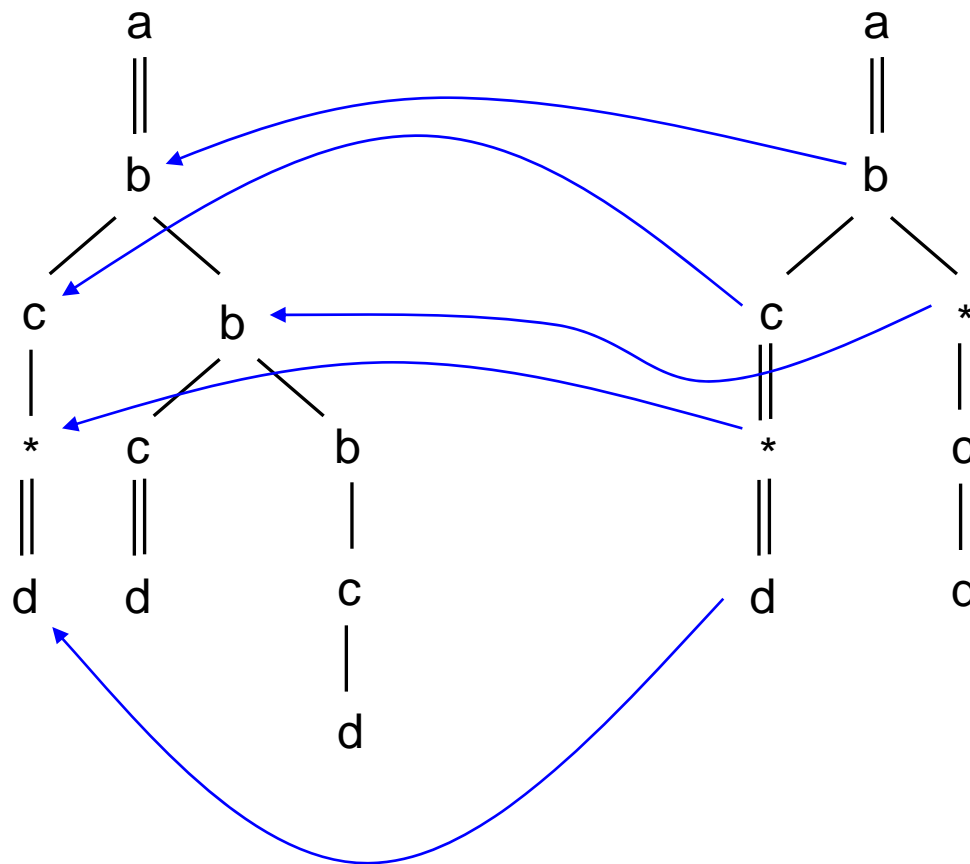
(11) Show an example of XPath queries  $q_1, q_2$  such that they are not equivalent, but  $q_1$  is included in  $q_2$ .

Show that  $q_1$  is included in  $q_2$  using one of the methods discussed.

Use the homomorphism technique to test whether

$p = a[.//b[c/*//d]/b[c//d]/b[c/d]]$  is included in

$q = a[.//b[c//*//d]/*[c/d]]$



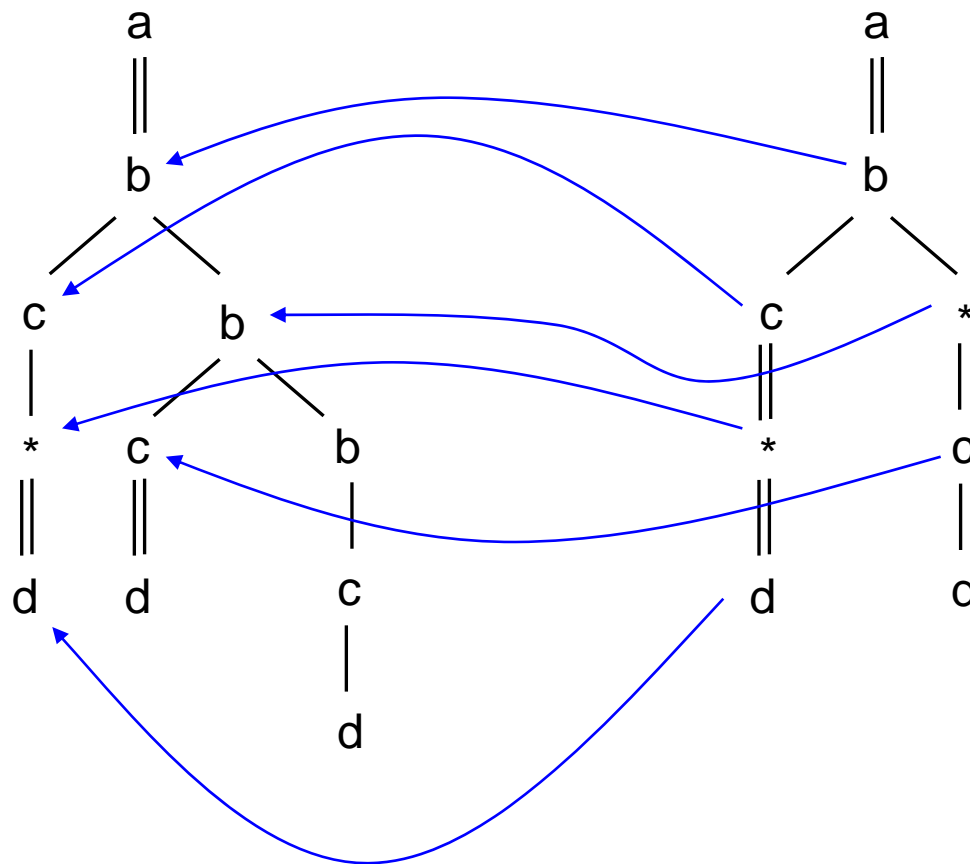
(11) Show an example of XPath queries  $q_1, q_2$  such that they are not equivalent, but  $q_1$  is included in  $q_2$ .

Show that  $q_1$  is included in  $q_2$  using one of the methods discussed.

Use the homomorphism technique to test whether

$p = a[.//b[c/*//d]/b[c//d]/b[c/d]]$  is included in

$q = a[.//b[c//*//d]/*[c/d]]$



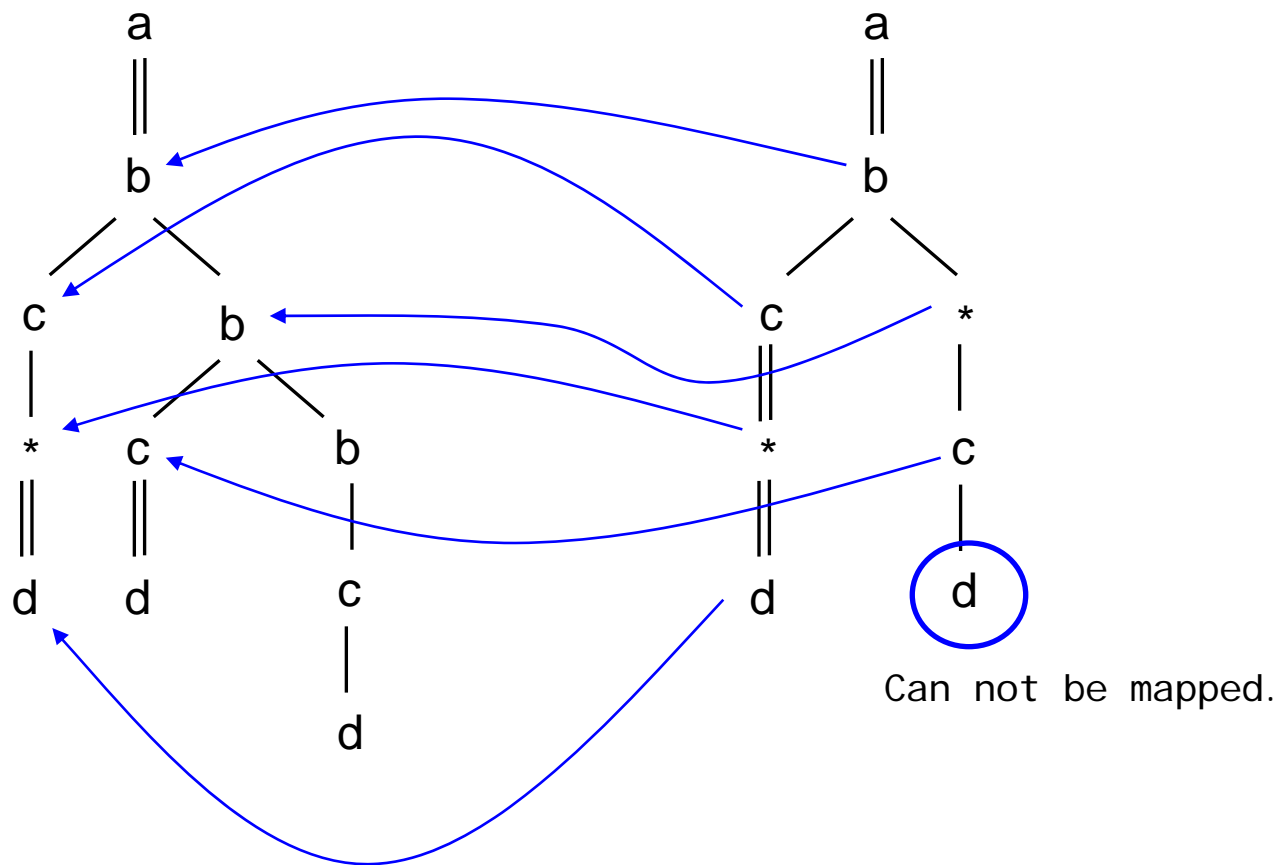
(11) Show an example of XPath queries  $q_1, q_2$  such that they are not equivalent, but  $q_1$  is included in  $q_2$ .

Show that  $q_1$  is included in  $q_2$  using one of the methods discussed.

Use the homomorphism technique to test whether

$p = a[.//b[c/*//d]/b[c//d]/b[c/d]]$  is included in

$q = a[.//b[c//*//d]/*[c/d]]$



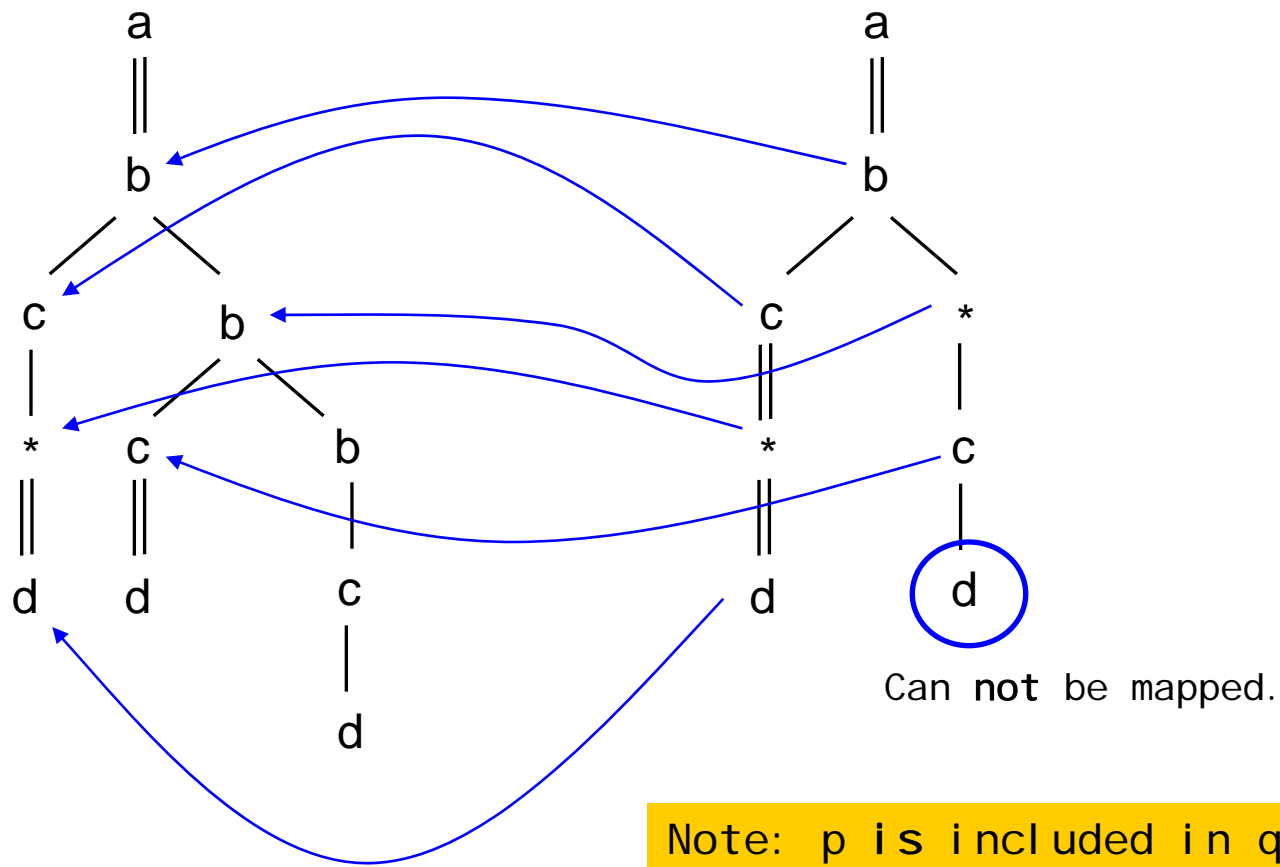
(11) Show an example of XPath queries q1, q2 such that they are not equivalent, but q1 is included in q2.

Show that q1 is included in q2 using one of the methods discussed.

Use the homomorphism technique to test whether

$p = a[.//b[c/*//d]/b[c//d]/b[c/d]]$  is included in

$q = a[.//b[c//*//d]/* [c/d]]$



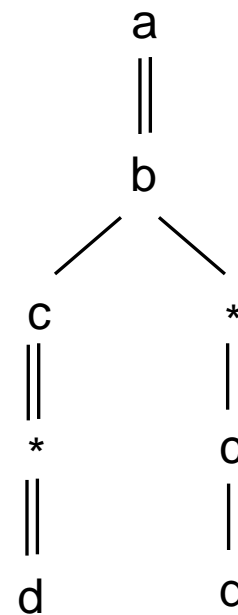
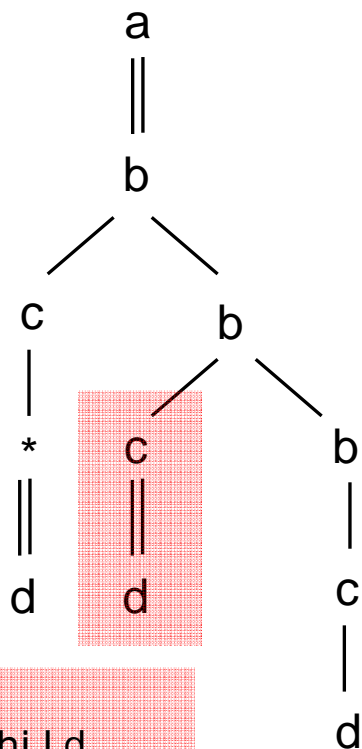
(11) Show an example of XPath queries q1, q2 such that they are not equivalent, but q1 is included in q2.

Show that q1 is included in q2 using one of the methods discussed.

Use the homomorphism technique to test whether

$p = a[.//b[c/*//d]/b[c//d]/b[c/d]]$  is included in

$q = a[.//b[c//*//d]/*[c/d]]$



Why?

Case 1:  
c has d-child

Case 2:  
c has no d-child

Note: p is included in q!!

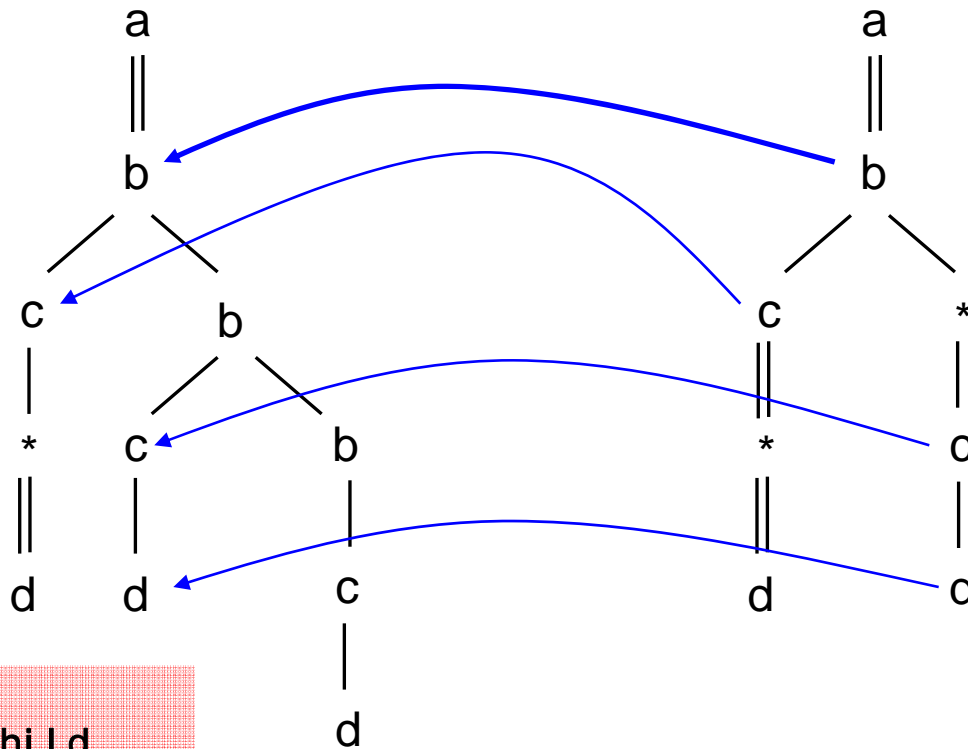
(11) Show an example of XPath queries q1, q2 such that they are not equivalent, but q1 is included in q2.

Show that q1 is included in q2 using one of the methods discussed.

Use the homomorphism technique to test whether

$p = a[.//b[c/*//d]/b[c//d]/b[c/d]]$  is included in

$q = a[.//b[c//*//d]/*[c/d]]$



→ OK!!  
p is included in q, for Case 1!

Case 1:  
c has d-child

Case 2:  
c has no d-child

Why?

Note: p is included in q!!

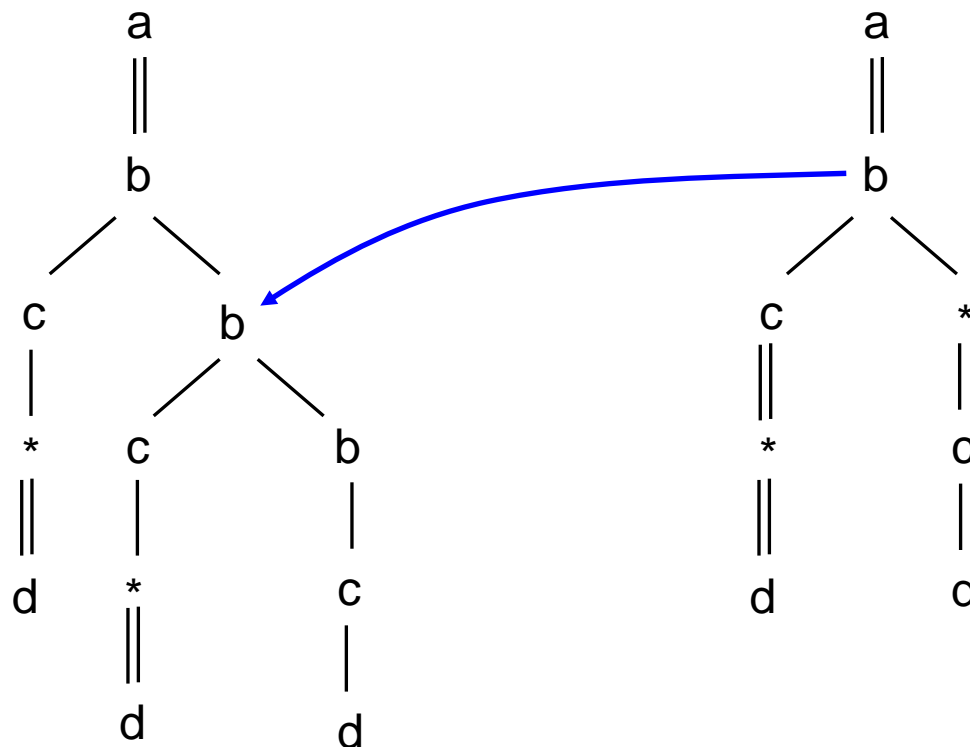
(11) Show an example of XPath queries q1, q2 such that they are not equivalent, but q1 is included in q2.

Show that q1 is included in q2 using one of the methods discussed.

Use the homomorphism technique to test whether

$p = a[.//b[c/*//d]/b[c//d]/b[c/d]]$  is included in

$q = a[.//b[c//*//d]/* [c/d]]$



Also OK!!  
p included in q, for Case 2!

Why?

Case 2:  
c has no d-child

Note: p is included in q!!

(12) Construct a DTD such that 10a) is included in 10e),  
and another DTD such that 10e) is included in 10a).

[Very easy!!]

10a) //c//d

10e) //c//d/preceding: : \*//d

---

(12) Construct a DTD such that 10a) is included in 10e),  
and another DTD such that 10e) is included in 10a).

[Very easy!!]

10a) //c//d

10e) //c//d/preceding: : \*//d

---

For the first part:

\*Any DTD\* so that c-nodes do NOT have d-descendants! ☺

For the second part:

Any DTD, so that all d-nodes are c-descendants.

(14) Given a PRE/POST/SIZE table, show SQL queries for the XPath queries

- a) /\*
  - b) /a/b/\*
  - c) //a/\*//b
  - d) //a/following-sibling: : b
-

(14) Given a PRE/POST/SIZE table, show SQL queries for the XPath queries

- a) /\*
  - b) /a/b/\*
  - c) //a/\*//b
  - d) //a/following-sibling::b
- 

```
SELECT DISTINCT r1.pre FROM doc_tbl r1
WHERE r1.pre=1
ORDERED BY r1.pr
```

(14) Given a PRE/POST/SIZE table, show SQL queries for the XPath queries

- a) /\*
  - b) /a/b/\*
  - c) //a/\*//b
  - d) //a/following-sibling: : b
- 

```
SELECT DISTINCT r4.pre FROM doc_tbl r1, r2, r3, r4
  WHERE r1.pre=0
    AND r2.pre>r1.pre
    AND r2.post<r1.post
    AND (r2.pre-r2.post+r2.size)=(r1.pre-r1.post+r1.size)+1
    AND r2.tag="a"
    AND r3.pre>r2.pre
    AND r3.post<r2.post
    AND (r3.pre-r3.post+r3.size)=(r2.pre-r2.post+r2.size)+1
    AND r3.tag="b"
    AND r4.pre>r3.pre
    AND r4.post<r3.post
    AND (r4.pre-r4.post+r4.size)=(r3.pre-r3.post+r3.size)+1
ORDERED BY r4.pre
```

Recall:  $\text{Level}(n) = \text{pre}(n) - \text{post}(n) + \text{size}(n)$

(14) Given a PRE/POST/SIZE table, show SQL queries for the XPath queries

- a) /\*
  - b) /a/b/\*
  - c) //a/\*//b
  - d) //a/following-sibling: : b
- 

```
SELECT DISTINCT r4.pre FROM doc_tbl r1, r2, r3, r4
  WHERE r1.pre=0
        AND r2.pre>r1.pre
        AND r2.post<r1.post
        AND r2.tag="a"
        AND r3.pre>r2.pre
        AND r3.post<r2.post
        AND (r3.pre-r3.post+r3.size)=(r2.pre-r2.post+r2.size)+1
        AND r4.pre>r3.pre
        AND r4.post<r3.post
        AND r4.tag="b"
ORDERED BY r4.pre
```

Recall:  $Level(n) = pre(n) - post(n) + size(n)$

(14) Given a PRE/POST/SIZE table, show SQL queries for the XPath queries

- a) /\*
- b) /a/b/\*
- c) //a/\*//b
- d) //a/following-sibling::b

```
SELECT DISTINCT r4.pre FROM doc_tbl r1, r2, r3, r4
  WHERE r1.pre=0
    AND r2.pre>r1.pre
    AND r2.post<r1.post
    AND r2.tag="a"
    AND r3.pre<r2.pre
    AND r3.post>r2.post
    AND (r3.pre-r3.post+r3.size)=(r2.pre-r2.post+r2.size)-1
    AND r4.pre>r2.pre
    AND r4.post<r3.post
    AND (r4.pre-r4.post+r4.size)=(r2.pre-r2.post+r2.size)
    AND r4.tag="b"
ORDERED BY r4.pre
```

parent of a-node

→after a

→before parent

→same level as a

Recall:  $Level(n) = pre(n) - post(n) + size(n)$