

XML and Databases

Exam Preparation

Discuss Answers to last year's exam

Sebastian Maneth
NICTA and UNSW

CSE@UNSW -- Semester 1, 2008

(1) For each of the following, explain why it is not well-formed XML (is a WFC or the XML grammar violated?)

- a) `<author></author><title></title>`
- b) `<author><title></author></title>`
- c) `<info temp=' 25C' >content</info>`
- d) `<!DOCTYPE greeting [
 <!ELEMENT greeting (#PCDATA)>
 <!ENTITY e1 "&e2; e3">
 <!ENTITY e2 "&e3;">
 <!ENTITY e3 "&e2;">
>
<greeting> &e1; </greeting>`
- e) `<a at1="blah" at< 2="foo"> 1 < 5 `
- f) `<a b3="a" b2="b" b1="a" b2="5" />`
- g) `<a><c><c/></c><c/>ab&e; `

(1) For each of the following, explain why it is not well-formed XML (is a **WFC** or the **XML grammar** violated?)

- a) `<author></author><title></title>` → **XML grammar**
(cannot be derived by grammar!)
- b) `<author><title></author></title>`
- c) `<info temp=' 25C' >content</info>`
- d) `<!DOCTYPE greeting [
 <!ELEMENT greeting (#PCDATA)>
 <!ENTITY e1 "&e2; e3">
 <!ENTITY e2 "&e3;">
 <!ENTITY e3 "&e2;">
>
<greeting> &e1; </greeting>`
- e) `<a at1="blah" at< 2="foo"> 1 < 5 `
- f) `<a b3="a" b2="b" b1="a" b2="5" />`
- g) `<a><c><c/></c><c/>ab&e; `

(1) For each of the following, explain why it is not well-formed XML (is a WFC or the XML grammar violated?)

- a) `<author></author><title></title>` → XML grammar
- b) `<author><title></author></title>` → WFC
- c) `<info temp=' 25C' >content</info>`
- d) `<!DOCTYPE greeting [
 <!ELEMENT greeting (#PCDATA)>
 <!ENTITY e1 "&e2; e3">
 <!ENTITY e2 "&e3;">
 <!ENTITY e3 "&e2;">
>
<greeting> &e1; </greeting>`
- e) `<a at1="blah" at< 2="foo"> 1 < 5 `
- f) `<a b3="a" b2="b" b1="a" b2="5" />`
- g) `<a><c><c/></c><c/>ab&e; `

(1) For each of the following, explain why it is not well-formed XML (is a **WFC** or the **XML grammar** violated?)

- a) `<author></author><title></title>` → XML grammar
- b) `<author><title></author></title>` → WFC
- c) `<info temp=' 25C' >content</info>` → XML grammar (should be "25c";
-is actually OK..!)
- d) `<!DOCTYPE greeting [
 <!ELEMENT greeting (#PCDATA)>
 <!ENTITY e1 "&e2; e3">
 <!ENTITY e2 "&e3;">
 <!ENTITY e3 "&e2;">
>
<greeting> &e1; </greeting>`
- e) `<a at1="blah" at< 2="foo"> 1 < 5 `
- f) `<a b3="a" b2="b" b1="a" b2="5" />`
- g) `<a><c><c/></c><c/>ab&e; `

(1) For each of the following, explain why it is not well-formed XML (is a **WFC** or the **XML grammar** violated?)

- a) `<author></author><title></title>` → XML grammar
- b) `<author><title></author></title>` → WFC
- c) `<info temp=' 25C' >content</info>` → XML grammar (should be "25c";
-is actually OK..!)
- d) `<!DOCTYPE greeting [
 <!ELEMENT greeting (#PCDATA) >
 <!ENTITY e1 "&e2; e3">
 <!ENTITY e2 "&e3;">
 <!ENTITY e3 "&e2;">
>
<greeting> &e1; </greeting>` → WFC
- Well-formedness constraint: No Recursion**
A parsed entity *MUST NOT* contain a recursive reference to itself, either directly or indirectly.
- e) `<a at1="blah" at< 2="foo" > 1 < 5 `
- f) `<a b3="a" b2="b" b1="a" b2="5" />`
- g) `<a><c><c/></c><c/>ab&e; `

(1) For each of the following, explain why it is not well-formed XML (is a WFC or the XML grammar violated?)

- a) `<author></author><title></title>` → XML grammar
- b) `<author><title></author></title>` → WFC
- c) `<info temp=' 25C' >content</info>` → XML grammar (should be "25c";
-is actually OK..!)
- d) `<!DOCTYPE greeting [
 <!ELEMENT greeting (#PCDATA)>
 <!ENTITY e1 "&e2; e3">
 <!ENTITY e2 "&e3;">
 <!ENTITY e3 "&e2;">
>
<greeting> &e1; </greeting>` → WFC
- e) `<a at1="bl ah" at< 2="foo" > 1 < 5 ` → XML grammar
- f) `<a b3="a" b2="b" b1="a" b2="5" />`
- g) `<a><c><c/></c><c/>ab&e; `

[41]	Attribute	::=	<u>Name</u> <u>Eg</u> <u>AttValue</u>
[5]	Name	::=	(<u>Letter</u> '_' ':') (<u>NameChar</u>)*
[84]	Letter	::=	[a-zA-Z]

(1) For each of the following, explain why it is not well-formed XML (is a WFC or the XML grammar violated?)

- a) `<author></author><title></title>` → XML grammar
- b) `<author><title></author></title>` → WFC
- c) `<info temp=' 25C' >content</info>` → XML grammar (should be "25c";
-is actually OK..!)
- d) `<!DOCTYPE greeting [
 <!ELEMENT greeting (#PCDATA)>
 <!ENTITY e1 "&e2; e3">
 <!ENTITY e2 "&e3;">
 <!ENTITY e3 "&e2;">
>
<greeting> &e1; </greeting>` → WFC
- e) `<a at1="blah" at< 2="foo" > 1 < 5 ` → XML grammar
- f) `<a b3="a" b2="b" b1="a" b2="5" />` → WFC
- g) `<a><c><c/></c><c/>ab&e; `

Well-formedness constraint: Unique Att Spec
An attribute name *MUST NOT* appear more than once in the same start-tag or empty-element tag.

(1) For each of the following, explain why it is not well-formed XML (is a WFC or the XML grammar violated?)

- a) `<author></author><title></title>` → XML grammar
- b) `<author><title></author></title>` → WFC
- c) `<info temp=' 25C' >content</info>` → XML grammar (should be "25c";
-is actually OK..!)
- d) `<!DOCTYPE greeting [
 <!ELEMENT greeting (#PCDATA)>
 <!ENTITY e1 "&e2; e3">
 <!ENTITY e2 "&e3;">
 <!ENTITY e3 "&e2;">
>
<greeting> &e1; </greeting>` → WFC
- e) `<a at1="blah" at< 2="foo" > 1 < 5 ` → XML grammar
- f) `<a b3="a" b2="b" b1="a" b2="5" />` → WFC
- g) `<a><c><c/></c><c/>ab&e; ` → WFC

Well-formedness constraint: Entity Declared
... the Name given in the entity reference *MUST match* that
in an entity declaration that...

(2) Show sequences of Unicode characters for which

a) UTF-8 needs more space than UTF-16

b) UTF-16 needs more space than UTF-8

together with the corresponding UTF codes and their lengths.

c) Explain how to binary sort a sequence of UTF-8 characters.

Use pseudo code if appropriate.

(2) Show sequences of Unicode characters for which

a) UTF-8 needs more space than UTF-16

b) UTF-16 needs more space than UTF-8

together with the corresponding UTF codes and their lengths.

c) Explain how to binary sort a sequence of UTF-8 characters.

Use pseudo code if appropriate.

a) U+FFFF

UTF-8: 11101111 10111111 10111111 = 24bits

UTF-16: 11111111 11111111 = 16bits

(2) Show sequences of Unicode characters for which

a) UTF-8 needs more space than UTF-16

b) UTF-16 needs more space than UTF-8

together with the corresponding UTF codes and their lengths.

c) Explain how to binary sort a sequence of UTF-8 characters.

Use pseudo code if appropriate.

a) U+FFFF

UTF-8: 11101111 10111111 10111111 = 24bits

UTF-16: 11111111 11111111 = 16bits

b) U+00

UTF-8: 00000000 = 8bits

UTF-16: 00000000 00000000 = 16bits

- (2) Show sequences of Unicode characters for which
- UTF-8 needs more space than UTF-16
 - UTF-16 needs more space than UTF-8 together with the corresponding UTF codes and their lengths.
 - Explain how to binary sort a sequence of UTF-8 characters. Use pseudo code if appropriate.
-

a) U+FFFF

UTF-8: 11101111 10111111 10111111 = 24bits
UTF-16: 11111111 11111111 = 16bits

b) U+00

UTF-8: 00000000 = 8bits
UTF-16: 00000000 00000000 = 16bits

c)

To binary compare two characters, simply start from the highest bit!
In this way, for characters with different lengths in UTF-8, after ≤ 4 bits we will be done!

- In case UTF-8 lengths are same \rightarrow normal binary compare..

(3) Show an element node with mixed content, using the **XML Information Set**. Assume that for a node M, Type(M) is its type, i.e., is one of DOC, ELEM, ATTR, or CHAR.

Using the Infoset, show pseudo code that, given a node N,

a) returns all ancestors of the node

b) returns the previous sibling of the node.

(3) Show an element node with mixed content, using the [XML Information Set](#). Assume that for a node M, Type(M) is its type, i.e., is one of DOC, ELEM, ATTR, or CHAR.

Using the Infoset, show pseudo code that, given a node N,

a) returns all ancestors of the node

b) returns the previous sibling of the node.

```
local name(e1) = "elem"
children(e1) = [e2, c1]
local name(e2) = "elem"
children(e2) = []
parent(e2) = e1
code(c1) = U+00
parent(c1) = e1
attributes(e1) = []
attributes(e2) = []
```

(3) Show an element node with mixed content, using the **XML Information Set**. Assume that for a node M, Type(M) is its type, i.e., is one of DOC, ELEM, ATTR, or CHAR.

Using the Infoset, show pseudo code that, given a node N,

a) returns all ancestors of the node

b) returns the previous sibling of the node.

```
local name(e1) = "el em"
children(e1) = [e2, c1]
local name(e2) = "el em"
children(e2) = []
parent(e2) = e1
code(c1) = U+00
parent(c1) = e1
attributes(e1) = []
attributes(e2) = []
```

```
a) getAncestors(Node n): NodeSet
{
    NodeSet result=NULL;

    if(n.type!=DOC){
        for(; n=n->parent ; n.type!="DOC") Add(n, result);
        Add(n, result);
    }
    return result;
}
```

(3) Show an element node with mixed content, using the **XML Information Set**. Assume that for a node M, Type(M) is its type, i.e., is one of DOC, ELEM, ATTR, or CHAR.

Using the Infoset, show pseudo code that, given a node N,

a) returns all ancestors of the node

b) returns the previous sibling of the node.

```

local name(e1) = "elem"
children(e1) = [e2, c1]
local name(e2) = "elem"
children(e2) = []
parent(e2) = e1
code(c1) = U+00
parent(c1) = e1
attributes(e1) = []
attributes(e2) = []

```

```

a) getAncestors(Node n): NodeSet
{
    NodeSet result=NULL;

    if(n.type!=DOC){
        for(; n=n->parent ; n.type!="DOC") Add(n, result);
        Add(n, result);
    }
    return result;
}

```

```

b) getPrevSi b(Node n): Node
{
    NodeList l=NULL;
    if(n.type!="DOC")
    {
        Node parent=n->parentNode();
        l=n->children;
        if(l==NULL) return NULL;
        s=first(l);
        if(s==l) return NULL;
        while(s->next!=n)
            s=s->next();
    }
    return s;
}

```

(4) Using DOM, give pseudo code that determines the average depth of the XML tree. The average depth of `<a/>` is 1.

(4) Using DOM, give pseudo code that determines the average depth of the XML tree. The average depth of <a/> is 1.

```
int total=0;
int count=0;

call calcAverage(root, 1);

return total/count;

void calcAverage(Node n, int depth)
{
    NodeList children = n->childList();
    if(children->isEmpty())
    {
        total += depth;
        count++;
        return;
    }
    else for each Node c in children calcAverage(c, depth+1);
}
```

(5) Explain in detail, using an example, why hashing is useful for finding the minimal DAG of a tree.

Why are updates more expensive on a DAG than on a tree?

Give an example that clearly explains this.

(5) Explain in detail, using an example, why hashing is useful for finding the minimal DAG of a tree.

Why are updates more expensive on a DAG than on a tree?

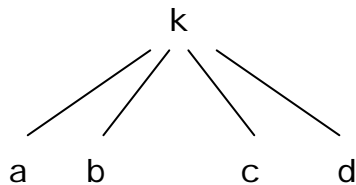
Give an example that clearly explains this.

When computing the minimal DAG, we need to determine whether a given subtree has occurred already. If we keep a table of pointers to the subtrees that have occurred already, then to check for a given subtree whether or not it occurs in the table takes in the worst case $(\# \text{ of trees in the table}) \times (\# \text{ nodes in subtree})$ which, in the worst case, is quadratic to the size of the input tree!

With hashing, we only need $(\# \text{ trees in the hash bucket}) \times (\# \text{ nodes in the subtree})$.

Thus, if a bucket has only a constant number of trees, on average, then the complexity goes from quadratic to linear!

Example tree:



Seen: =NULL
Seen: =seen + "a-tree"
Contains(Seen, "b-tree")? ← needs 1 comparison
Seen: =Seen + "b-tree"
Contains(Seen, "c-tree")? ← needs 2 comparisons
Seen: =Seen + "c-tree"
Contains(Seen, "d-tree")? ← needs 3 comparisons
Seen: =Seen + "d-tree"

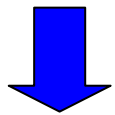
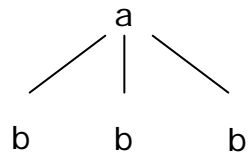
Assume
hash("a-tree")=1
hash("b-tree")=2
hash("c-tree")=3
hash("d-tree")=4

Then we need no
(tree) comparisons
whatsoever!

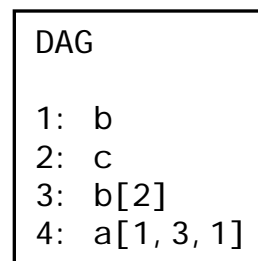
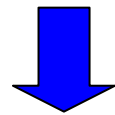
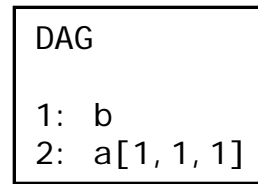
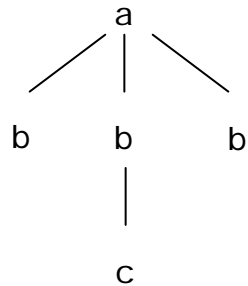
(5) Explain in detail, using an example, why **hashing** is useful for finding the minimal DAG of a tree.

Why are **updates** more expensive on a DAG than on a tree?

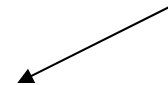
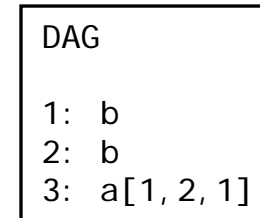
Give an example that clearly explains this.



Insert(2, "c-tree")



Duplicate node "2"



Inserting a single new child required adding two rows and changing one existing one.

(the shared 2nd child "b-subtree" of the a-node must be duplicated first, before the c-child can be added.)

(6) Give the PRE/POST table for the tree

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

b) Give pseudo code that computes the POST order of a tree in an iterative way, i.e., without any recursive calls(!). You can use `firstChild(n)`, `nextSibling(n)`, and `parent(n)` for a node `n`.

Using the PRE/POST-encoding, explain how to obtain

c) the ancestors of a node

d) the last child of a node

e) the maximal depth of the subtree at a node.

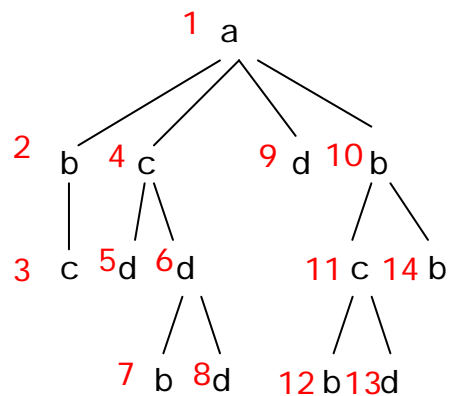
(6) Give the PRE/POST table for the tree

<a><c/><c><d/><d></d></c><d/><c><d/></c><d/>

b) Give pseudo code that computes the POST order of a tree in an iterative way, i.e., without any recursive calls(!). You can use firstChild(n), nextSibling(n), and parent(n) for a node n.

Using the PRE/POST-encoding, explain how to obtain

- c) the ancestors of a node
- d) the last child of a node
- e) the maximal depth of the subtree at a node.



PRE	POST	Label
1	14	a
2	2	b
3	1	c
4	7	c
5	3	d
6	6	d
7	4	b
8	5	b
9	8	d
10	13	b
11	11	c
12	12	b
13	10	d
14	12	d

(6) Give the PRE/POST table for the tree

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

b) Give pseudo code that computes the POST order of a tree in an iterative way, i.e., without any recursive calls(!). You can use `firstChild(n)`, `nextSibling(n)`, and `parent(n)` for a node `n`.

Using the PRE/POST-encoding, explain how to obtain

c) the ancestors of a node

d) the last child of a node

e) the maximal depth of the subtree at a node.

```
b) int i=1;
   Node n=root;
   repeat{
     while(firstChild(n)!=NULL) n=firstChild(n);
     post(i)=n;
     i++;
     while(nextSibling(n)==NIL){
       n=parent(n);
       if(n==NULL) break;
       post(i)=n;
       i++;
     }
     n=nextSibling(n);
   }
```

(6) Give the PRE/POST table for the tree

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

b) Give pseudo code that computes the POST order of a tree in an iterative way, i.e., without any recursive calls(!). You can use `firstChild(n)`, `nextSibling(n)`, and `parent(n)` for a node `n`.

Using the PRE/POST-encoding, explain how to obtain

- c) the **ancestors** of a node
- d) the last child of a node
- e) the maximal depth of the subtree at a node.

-
- c) Given (pre, post) of a node, its **ancestors** are all **nodes with pre-value < pre and post-value > post.**

(6) Give the PRE/POST table for the tree

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

b) Give pseudo code that computes the POST order of a tree in an iterative way, i.e., without any recursive calls(!). You can use `firstChild(n)`, `nextSibling(n)`, and `parent(n)` for a node `n`.

Using the PRE/POST-encoding, explain how to obtain

c) the ancestors of a node

d) the **last child** of a node

e) the maximal depth of the subtree at a node.

c) Given (pre, post) of a node, its ancestors are all nodes with pre-value < pre and post-value > post.

d) **If there is a node with pre-value > pre and with post-value=post-1, then that is the last child of (pre, post)**

(6) Give the PRE/POST table for the tree

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

b) Give pseudo code that computes the POST order of a tree in an iterative way, i.e., without any recursive calls(!). You can use `firstChild(n)`, `nextSibling(n)`, and `parent(n)` for a node `n`.

Using the PRE/POST-encoding, explain how to obtain

c) the ancestors of a node

d) the last child of a node

e) the **maximal depth** of the subtree at a node.

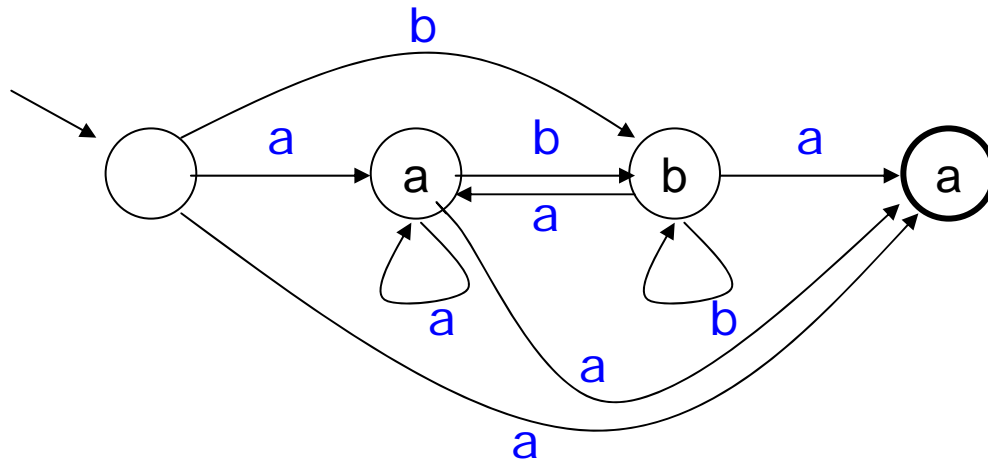
c) Given (pre, post) of a node, its ancestors are all nodes with pre-value < pre and post-value > post.

d) If there is a node with pre-value > pre and with post-value=post-1, then that is the last child of (pre, post)

```
e) int maxDepth(int pr){
    size(int p): int{
        int s=0;
        for(int pr2=p+1; post(pr2)<post(p); pr2++) s++
        return s;
    }
    int D=0; int u,L;
    L=pr+size(pr)-post(pr);
    for(int pr2=pr+1; post(pr2)<post(pr); pr2++){
        u=pr2+size(pr2)-post(pr2)-L;
        if(u>D) D=u;
    }
    return D;
}
```

(8) Show the **Glushkov automaton** for the regular expression $E=(a \mid b)^*a$.
Is this expression 1-unambiguous? Explain!
Give a deterministic automaton for the same expression.
Is $E_2=(b^*a(a \mid b))^*a$ equivalent to E ? Is it 1-unambiguous?
Show a 1-unambiguous expression that is equivalent to $a(a \mid b)^*$.

(8) Show the **Glushkov automaton** for the regular expression $E=(a \mid b)^*a$.
 Is this expression 1-unambiguous? Explain!
 Give a deterministic automaton for the same expression.
 Is $E2=(b^*a(a|b))^*a$ equivalent to E ? Is it 1-unambiguous?
 Show a 1-unambiguous expression that is equivalent to $a(a \mid b)^*$.



Deterministic??

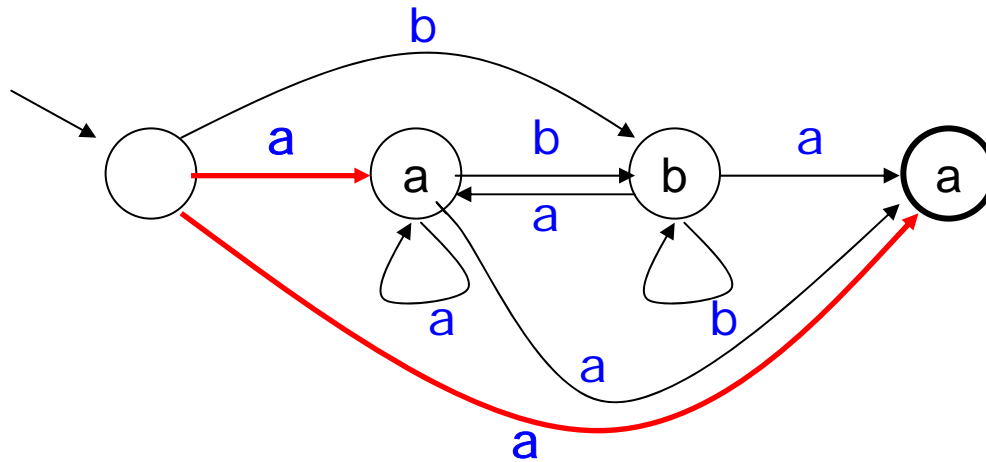
(8) Show the Glushkov automaton for the regular expression $E=(a \mid b)^*a$.

Is this expression 1-unambiguous? Explain!

Give a deterministic automaton for the same expression.

Is $E_2=(b^*a(a \mid b))^*a$ equivalent to E ? Is it 1-unambiguous?

Show a 1-unambiguous expression that is equivalent to $a(a \mid b)^*$.



Deterministic??

→ no!

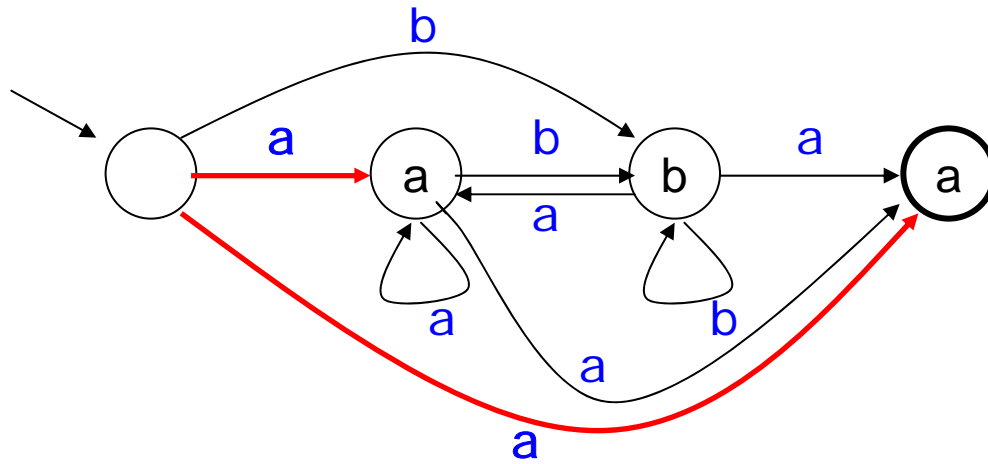
Thus, E is not 1-unambiguous.

(8) Show the Glushkov automaton for the regular expression $E=(a \mid b)^*a$.
 Is this expression 1-unambiguous? Explain!

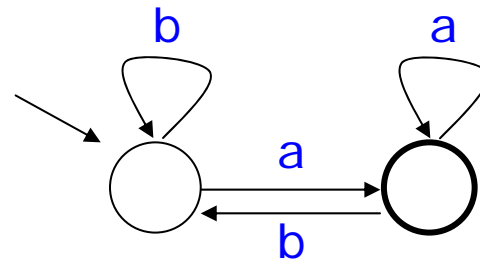
Give a deterministic automaton for the same expression.

Is $E_2=(b^*a(a \mid b))^*a$ equivalent to E ? Is it 1-unambiguous?

Show a 1-unambiguous expression that is equivalent to $a(a \mid b)^*$.



Deterministic automaton:

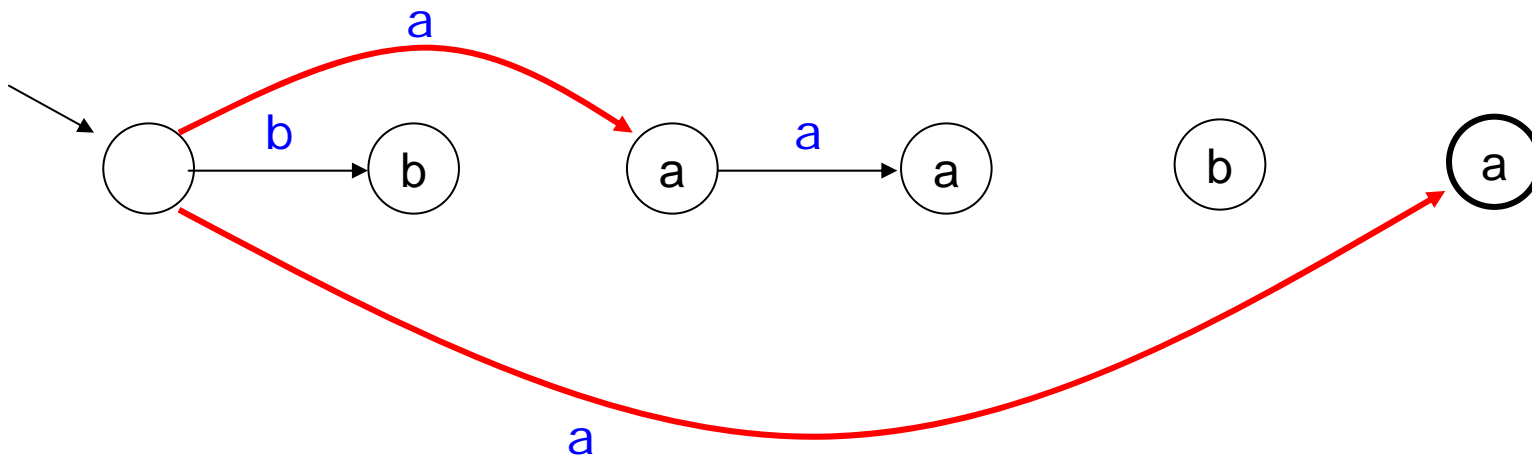


(8) Show the Glushkov automaton for the regular expression $E=(a \mid b)^*a$.
 Is this expression 1-unambiguous? Explain!
 Give a deterministic automaton for the same expression.
 Is $E2=(b^*a(a \mid b))^*a$ equivalent to E ? Is it 1-unambiguous?
 Show a 1-unambiguous expression that is equivalent to $a(a \mid b)^*$.

NOT equivalent to E !

The string "ba" is matched by E , but NOT by $E2$.

$E2$ is NOT 1-unambiguous:



(8) Show the Glushkov automaton for the regular expression $E=(a \mid b)^*a$.
Is this expression 1-unambiguous? Explain!
Give a deterministic automaton for the same expression.
Is $E_2=(b^*a(a \mid b))^*a$ equivalent to E ? Is it 1-unambiguous?
Show a 1-unambiguous expression that is equivalent to $a(a \mid b)^*$.

The expression $b^*a(b^*a)^*$
is

- equivalent to E
- 1-unambiguous.