TagAl:

A tag algebra for document markup

a talk given at

Balisage:
The markup conference
Montréal, 4 August 2011

by

Lars Johnsen and
Claus Huitfeldt,
University of Bergen

Structure of this talk

1 Background etc.

2 Some reflections on nesting and matching

matching of tags

matching of parentheses

3 Lattices

for documents without overlap

for documents with overlap

4 Some implications

document structures and lattice properties

~~5 Conclusion~~

Background

- SGML/XML and the overlap problem
- Responses:
  - Alternate linear forms
  - Mapping to other document models
  - Stand-off markup
  - Transformation algorithms
- Problem: Identity criteria for documents and operations on documents

About algebras

- "Algebra": The abstract study of ***numbers*** and ***operations*** on numbers
- Numbers and operations on numbers are "the same", no matter which ***notation*** we use, and no matter which ***algorithms*** we employ in order to perform those operations

About algebras

- By analogy, a tag algebra would provide identity criteria for elements of marked up documents, and for operations on such elements, across varieties of notations and algorithms.
- We hope this work to provide a basis for such an algebra.
- At least we think it may help in
  - clearifying some basic notions
  - suggesting alternate ways of thinking

XML and O-XML

- For purposes of this discussion:
  - Focus only on element structure
    - No attributes, entities, comments, processing instructions, CDATA sections, declarations, ...
  - XML

    <a> ... ...
  - O-XML

    <a> ... ...

Observation

- XML

  <a> ... ...

  <a> ... ... </> ... </>

- O-XML

  <a> ... ...

- The element structure of XML documents can be identified without knowledge of GIs.
- GIs on end tags are redundant in XML, but essential in O-XML.

Nesting and overlap

<a> ... ...

Nesting and overlap

<a> ... ...

<a> ... ...

Nesting and overlap

<a> ... ...

<a> ... ...

< > ... < > ... </ > ... </ >

Nesting and overlap

<a> ... ...

<a> ... ...

< > ... < > ... </ > ... </ >

(... (...) ...)

Nesting and overlap

<a> ... ...

<a> ... ...

< > ... < > ... </ > ... </ >

(... (...) ...)

- Matching of start and end tags

Nesting and overlap

<a> ... ...

<a> ... ...

< > ... < > ... </ > ... </ >

(... (...) ...)

- Matching of start and end tags
- Matching of simple parentheses.

Matching parentheses

Why does a: ((( )))

Matching parentheses

Why does a: ((( )))

read b: ((( )))

Matching parentheses

Why does a: ((( )))

read b: ((( )))

and not c: ((( )))

Matching parentheses

Why does a: ((( )))

read b: ((( )))

and not c: ((( ))) ?

Matching parentheses

Why does a: ((( )))

read b: ((( )))

and not c: ((( ))) ?

Two basic assumptions:

1) A one-to-one correspondence between “(“ and “)”.

Matching parentheses

Why does a: **((( )))**

read b: **((( )))**

and not c: **((( )))** **?**

Two basic assumptions:

1) A one-to-one correspondence between “(“ and “)”.

2) For every “(“ there is a succeeding “)”.

Matching parentheses

Why does a: **((( )))**

read b: **((( )))**

and not c: **((( )))** **?**

Two basic assumptions:

1) A one-to-one correspondence between “(“ and “)”.

2) For every “(“ there is a succeeding “)”.

Both a, b, and c comply with these assumptions.

Matching parentheses

Why does a: **((( )))**

read b: **((( )))**

and not c: **((( )))** **?**

Two basic assumptions:

1) A one-to-one correspondence between “(“ and “)”.

2) For every “(“ there is a succeeding “)”.

Both a, b, and c comply with these assumptions.

(Well-formedness vs structure.)

Structure and derivation

Rewrite grammar:

Base step:

P -> ()

Subordination step:

P -> (P)

Coordination step:

P -> PP

Structure and derivation

Rewrite grammar:

Base step:

P -> ()

Subordination step:

P -> (P)

Coordination step:

P -> PP

Derivation:

0 P

1 S $(_1$ P $)_1$

2 C $(_1$ P P $)_1$

3 B $(_1$ $(_3$ $)_3$ P $)_1$

4 B $(_1$ $(_3$ $)_3$ $(_4$ $)_4$ $)_1$

But context-free grammars cannot describe overlap (at least that is what we are told).

Are there other ways of deriving structure, which can also be used for modelling overlap?

Lattices?

Let's try

- First: documents without overlap.
- Then: documents with overlap.

Document lattices

A document is a linearly ordered set of:

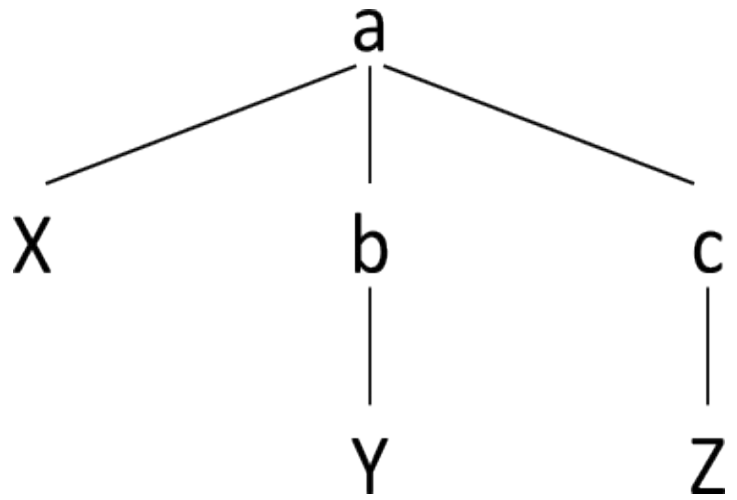
- Start tags
- End tags
- Simples (PCDATA+)

A subset* of the Cartesian product of all pairs of start and end tags, plus simples, constitutes lattice *nodes*.

Lattice nodes are ordered linearly as well as by containment.

A document without overlap:

<a> X Y <c> Z </c>



<a> X Y <c> Z </c>

1 <a>
2 X
3
4 Y
5
6 <c>
7 Z
8 </c>
9

<a> X Y <c> Z </c>

1	<a>
2	X
3	
4	Y
5	
6	<c>
7	Z
8	</c>
9	

<a>		1	5
<a>	</c>	1	8
<a>		1	9
		3	5
	</c>	3	8
		3	9
<c>	</c>	6	8
<c>		6	9
X		2	2
Y		4	4
Z		7	7

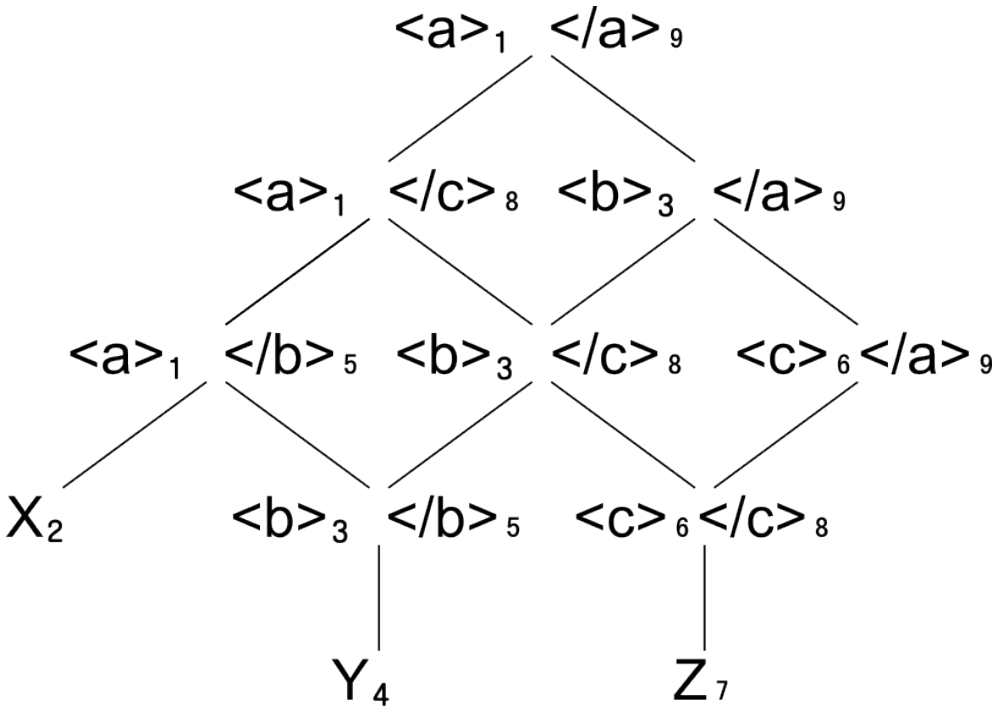
<a> X Y <c> Z </c>

1	<a>
2	X
3	
4	Y
5	
6	<c>
7	Z
8	</c>
9	

<a>		1	5	
<a>	</c>	1	8	
<a>		1	9	*
		3	5	*
	</c>	3	8	
		3	9	
<c>	</c>	6	8	*
<c>		6	9	
X		2	2	
Y		4	4	
Z		7	7	

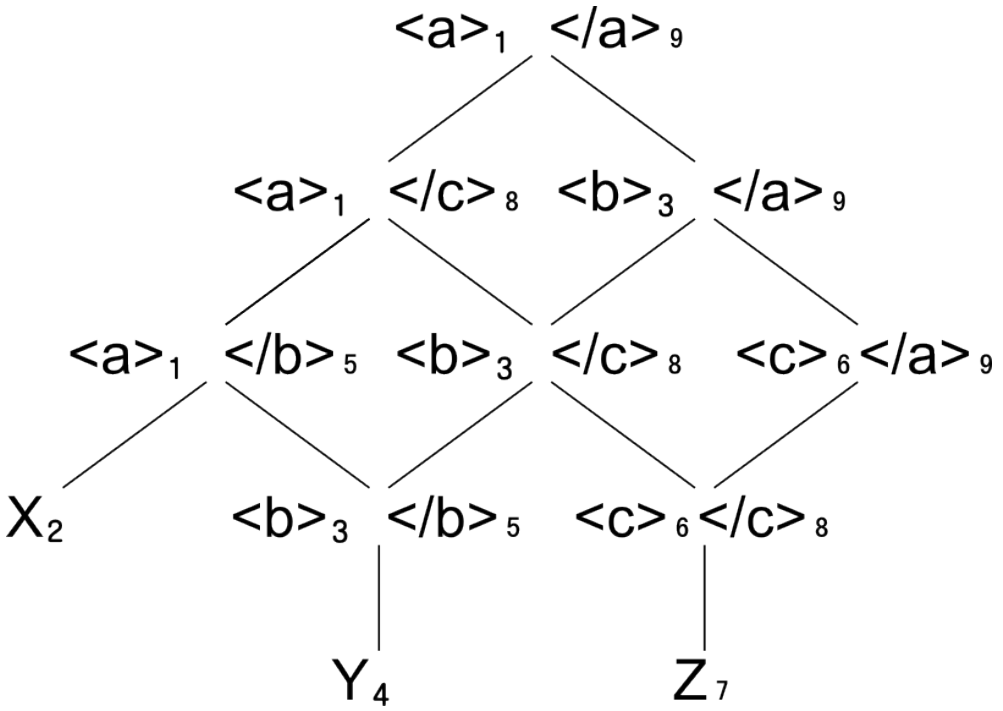
<a> X Y <c> Z </c>

Lattice for D:



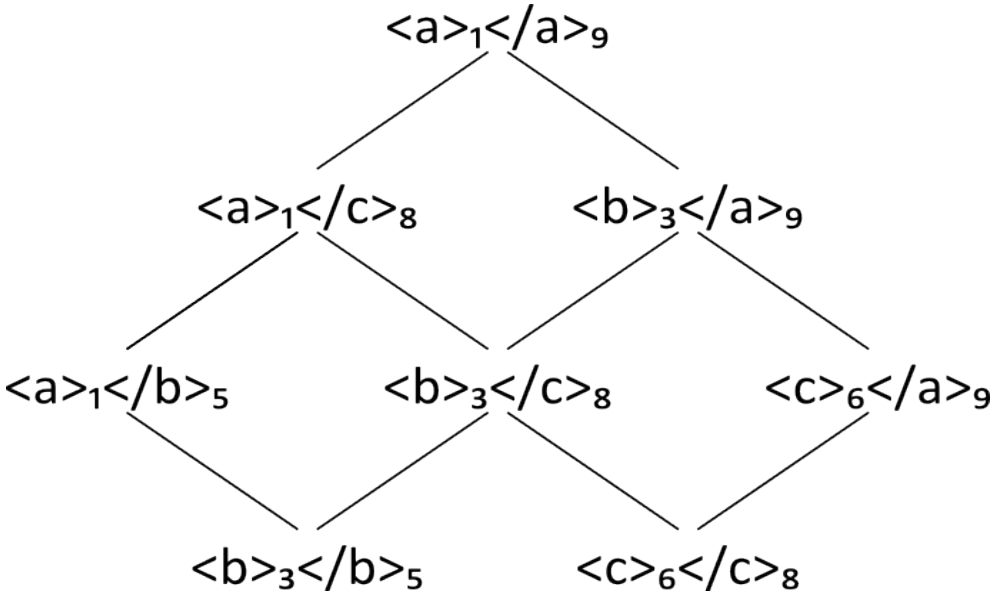
<a> X Y <c> Z </c>

Building a document model from the lattice



<a> X Y <c> Z </c>

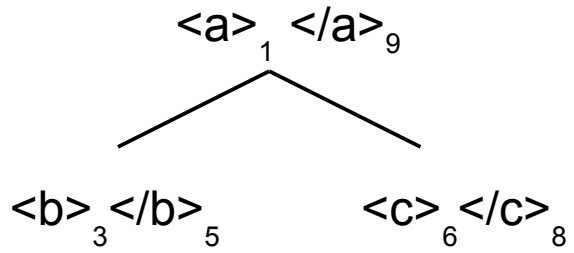
Building the model, step 1: Move minimal nodes



X_2 Y_4 Z_7

<a> X Y <c> Z </c>

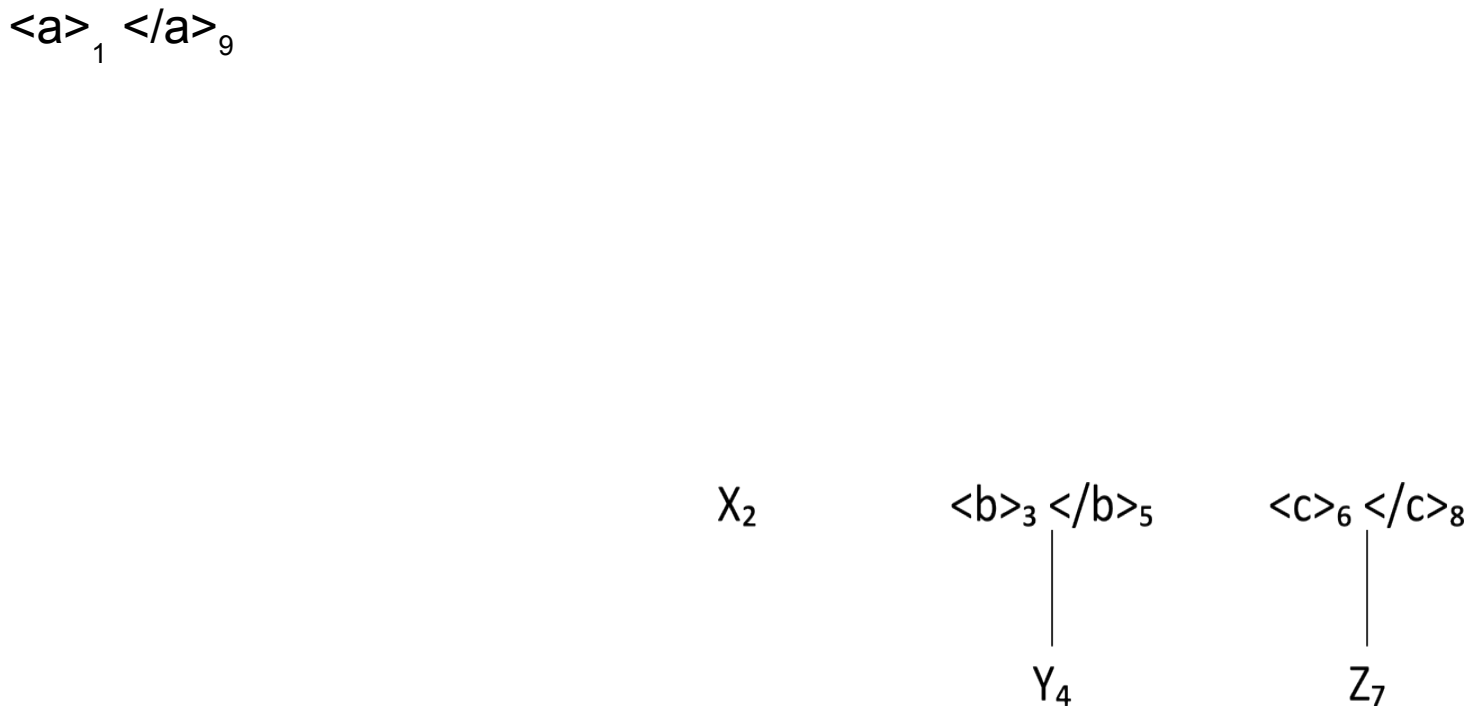
Step 2(a): Delete relatives of minimal nodes.



X$_2$ **Y**$_4$ **Z**$_7$

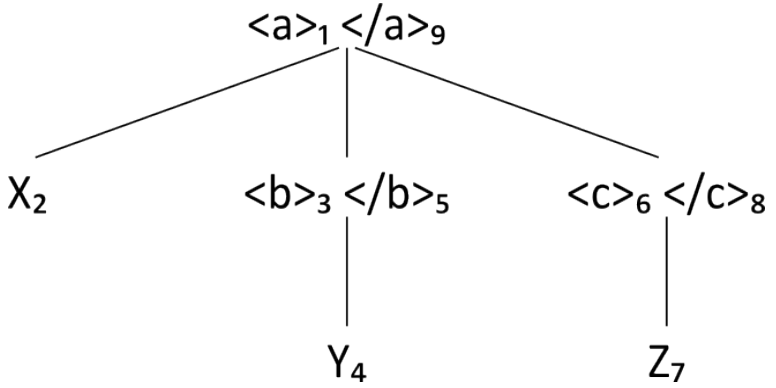
<a> X Y <c> Z </c>

Step 2(b): Move minimal nodes.

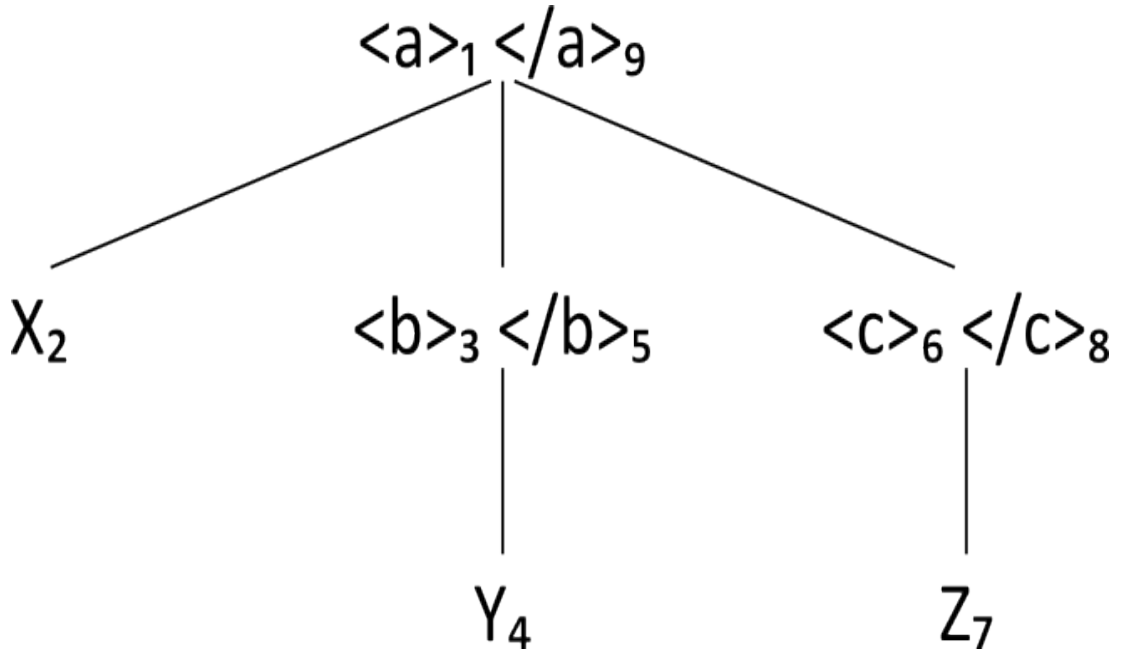


<a> X Y <c> Z </c>

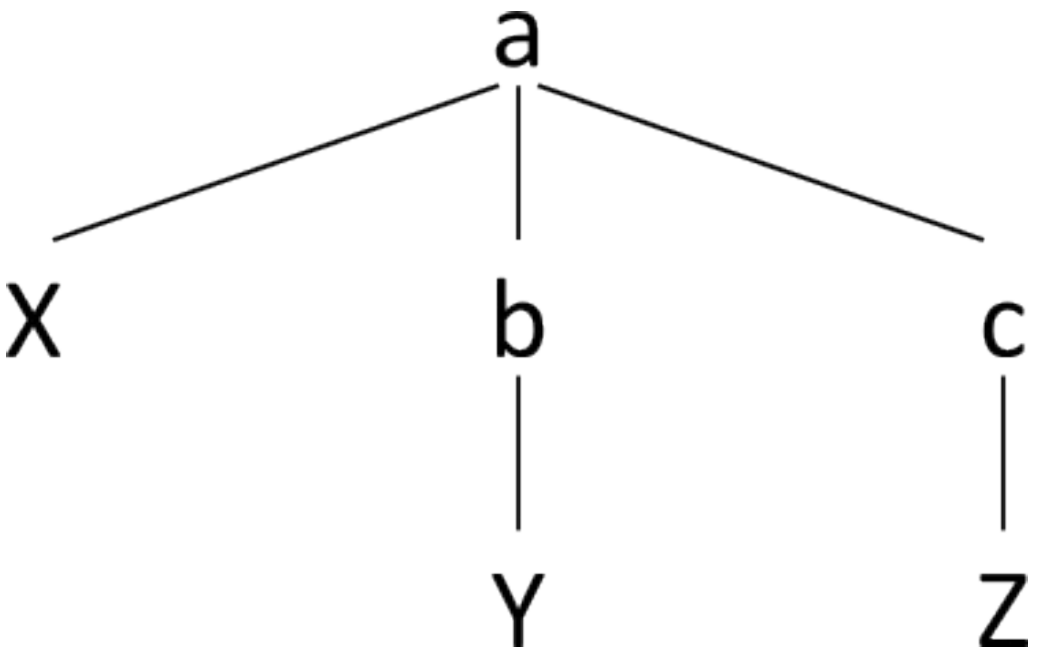
Step 3: Move remaining minimal node.



<a> X Y <c> Z </c>

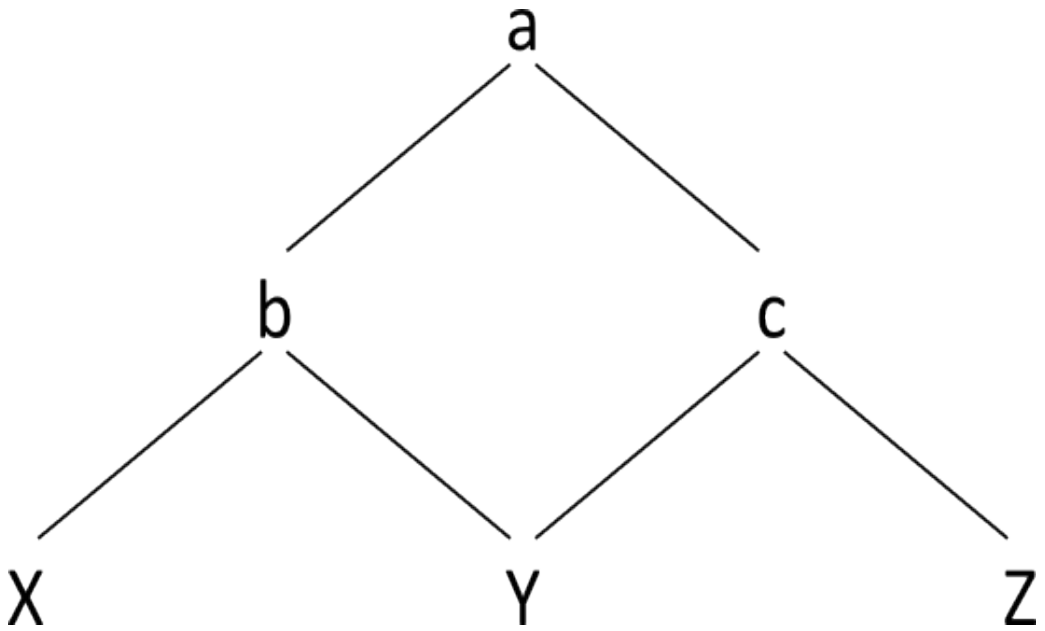


<a> X Y <c> Z </c>

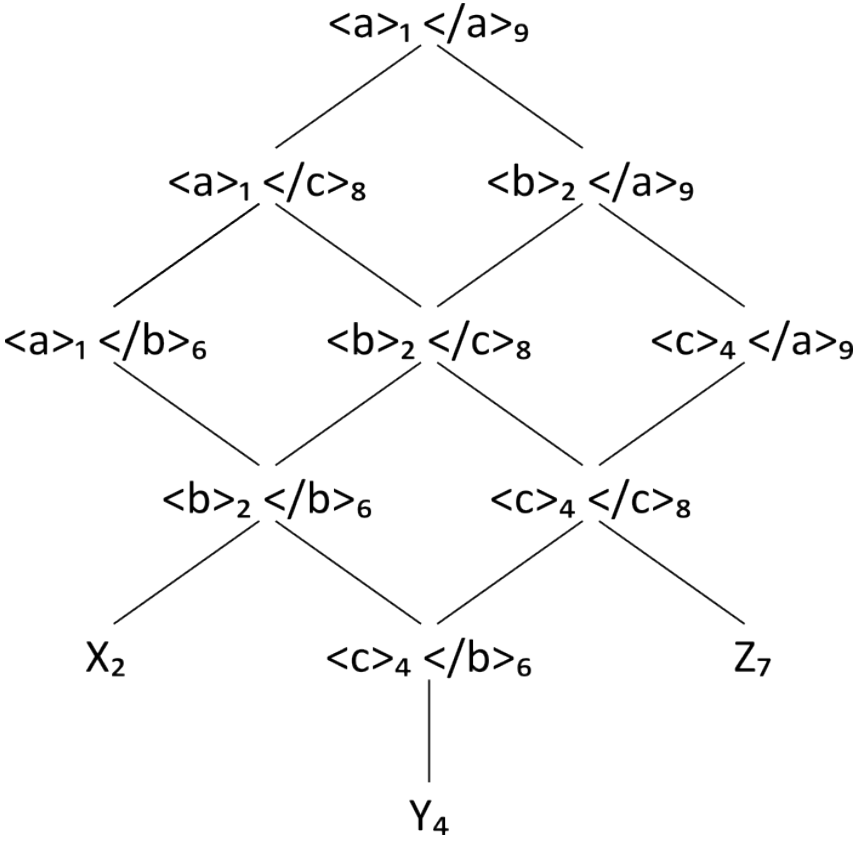


A document with overlap:

<a> X <c> Y Z </c>

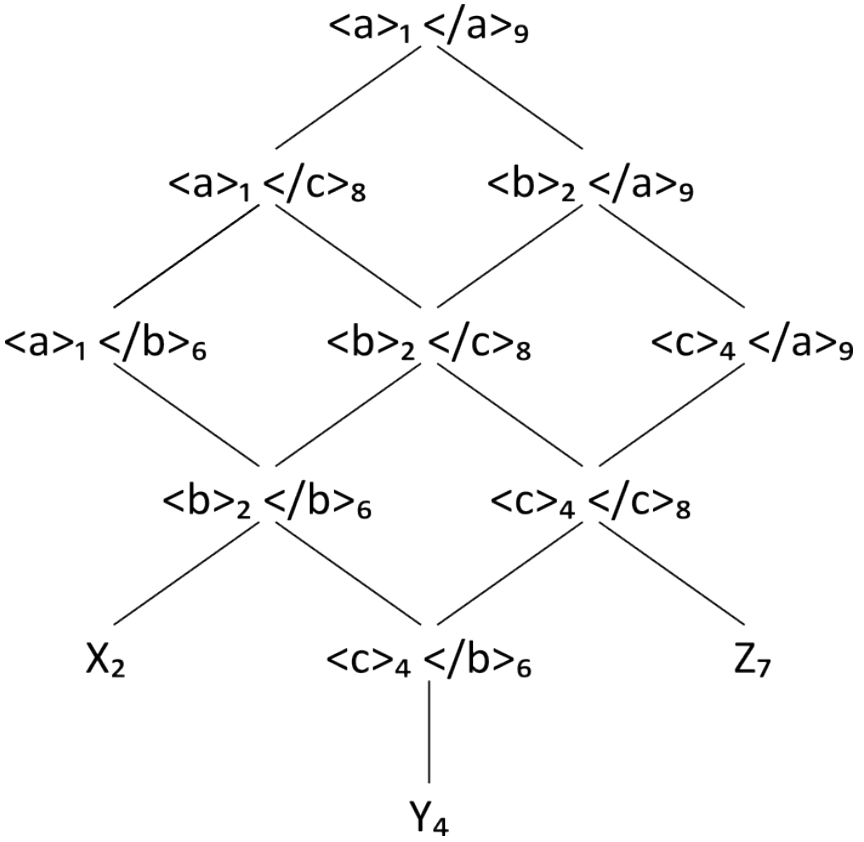


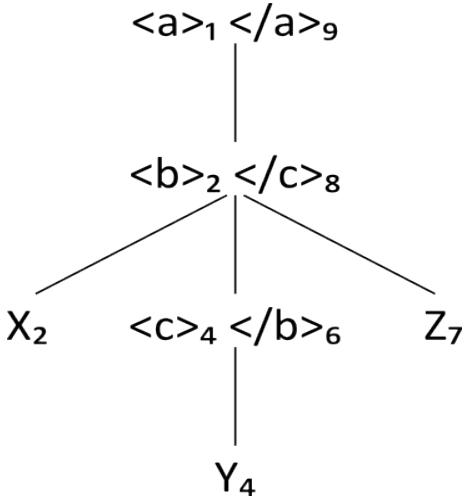
<a> X <c> Y Z </c>



Let us apply the same method to this lattice...

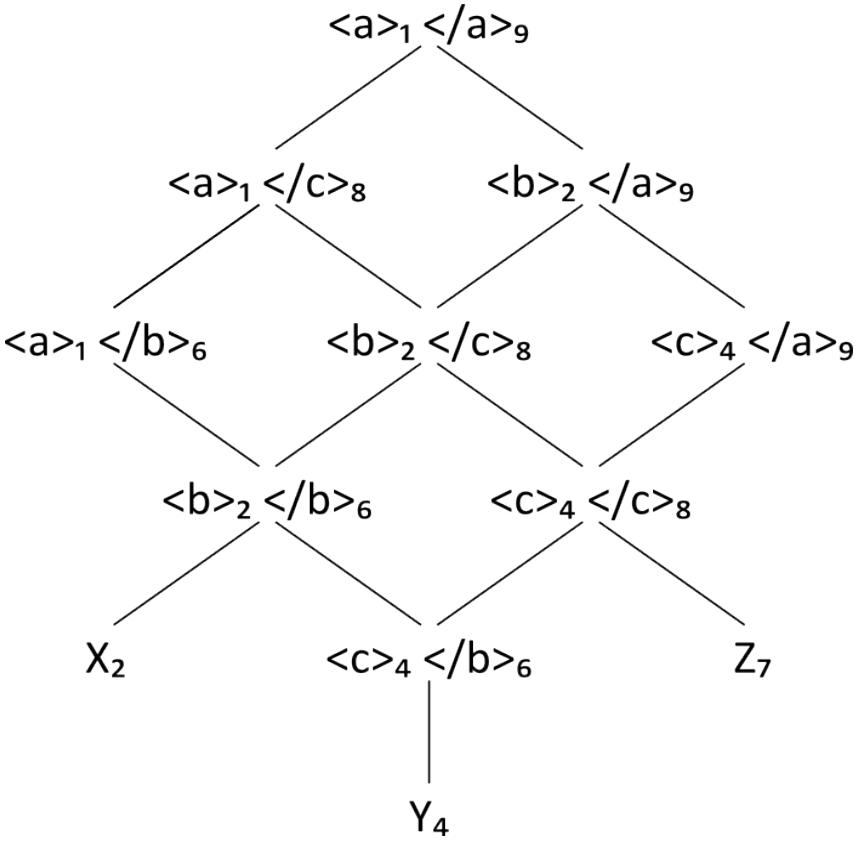
<a> X <c> Y Z </c>

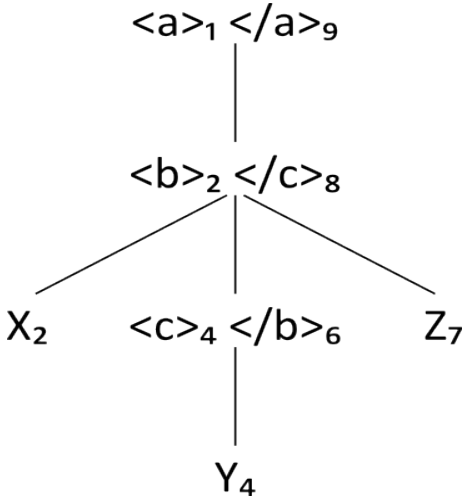




Something went wrong...

<a> X <c> Y Z </c>

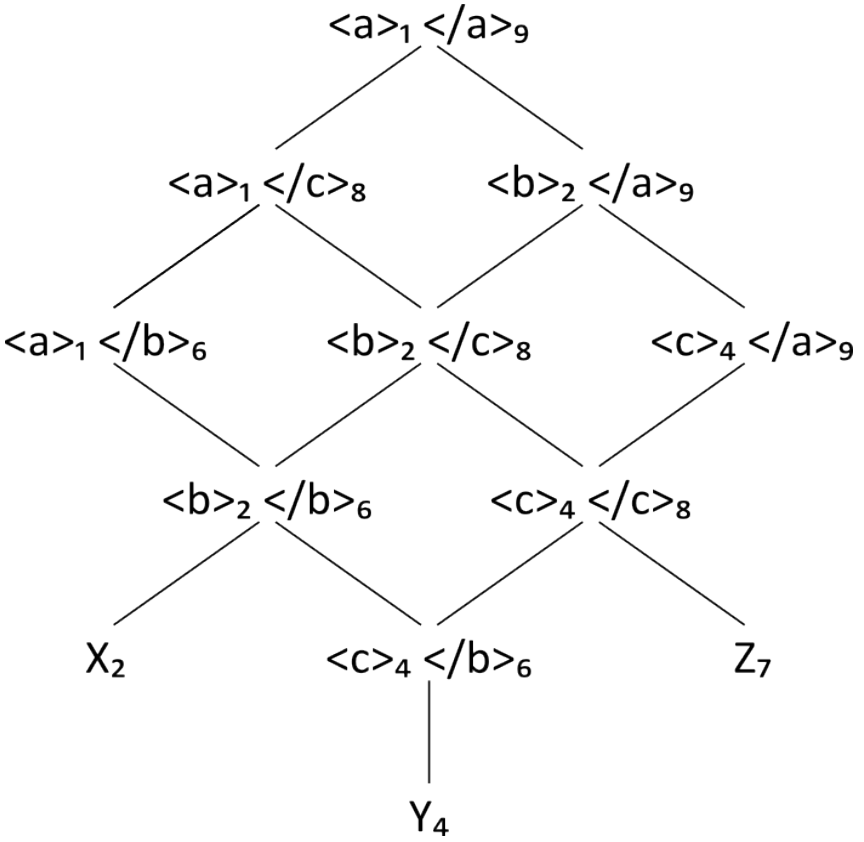


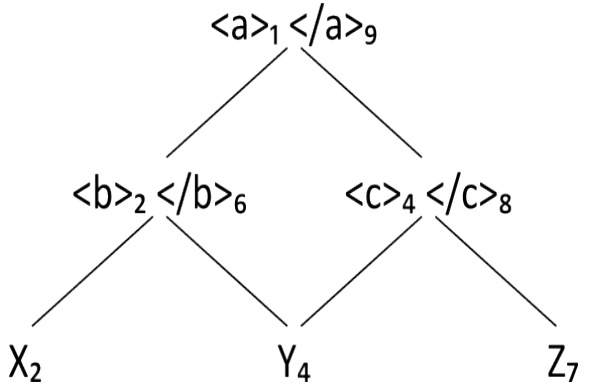


Something went wrong...

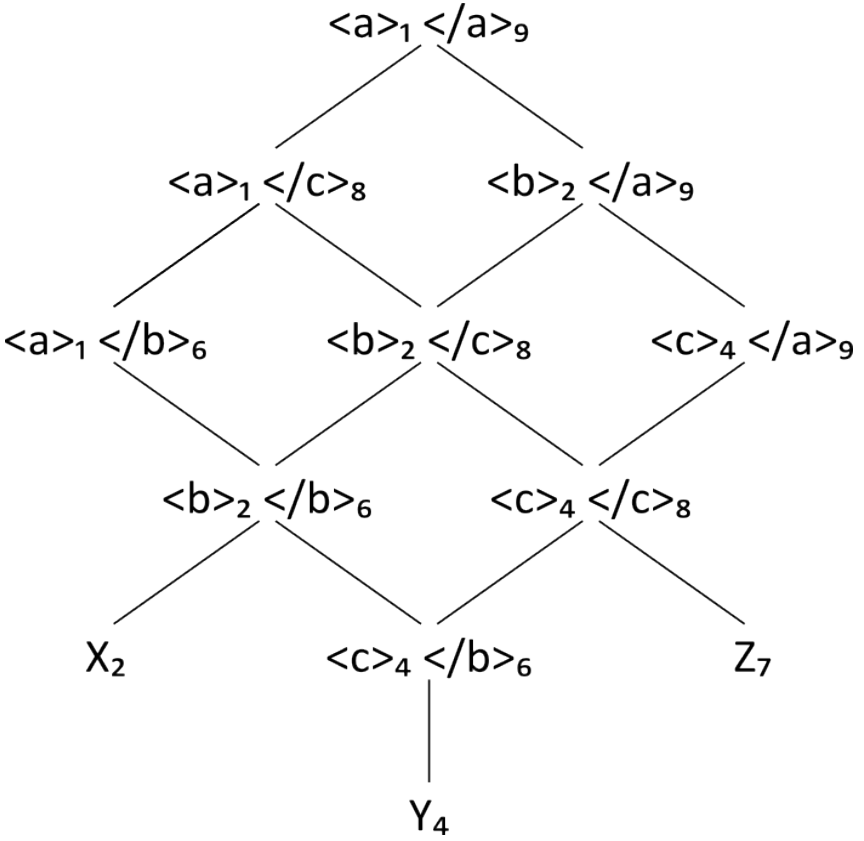
Let us first delete nodes with different start and end tag GIs...

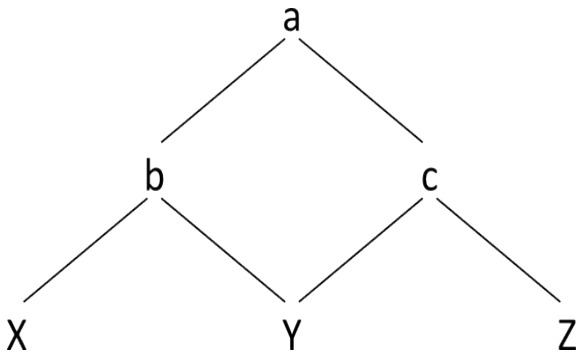
<a> X <c> Y Z </c>


<a>1 9
<a>1 </c>8
2 9
<a>1 6
2 </c>8
<c>4 9
2 6
<c>4 </c>8
X2
<c>4 6
Z7
Y4


<a>1 9
2 6
<c>4 </c>8
X2
Y4
Z7

<a> X <c> Y Z </c>





Eureka !

But, wait...

Objection:

If we had deleted nodes with non-matching start and end tag GIs to begin with, building the lattice for XML would have been so much easier.

Answer:

Yes, but:

- Normally, there will still be relatives to get rid of.
- The method demonstrates the point that element structure can be identified without looking at GIs

We have learned that:

- The element structure of XML documents can be identified without knowledge of GIs.
- GIs on end tags are redundant in XML, but essential in documents with overlap.

We have learned that:

- The element structure of XML documents can be identified without knowledge of GIs.
- GIs on end tags are redundant in XML, but essential in documents with overlap.

Yes, we knew that already!

We have learned that:

- The element structure of XML documents can be identified without knowledge of GIs.
- GIs on end tags are redundant in XML, but essential in documents with overlap.

Yes, we knew that already! But we have also learned that:

- XML and O-XML share some basic well-formedness constraints.
- Lattices may be used for identifying the element structure of both XML and O-XML
- Lattices can also be helpful in analyzing some further interesting properties of marked up documents...

Some further implications

- Well-formedness
- Relation to linear forms and document models
- “Spurious” overlap
- Algebraic characterization

Well-formedness hypothesis

For every tag t in D there is one and only one node x in M(D) or O(D) such that x.start=t or x.end=t.

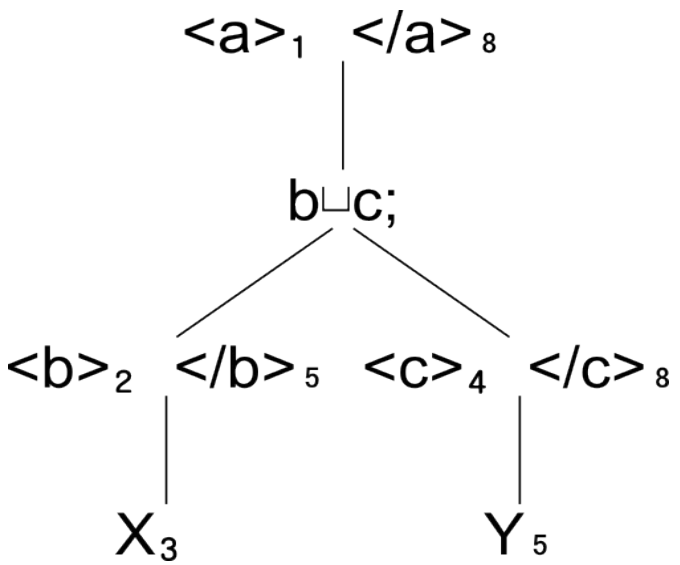
Linear forms and document models

- Document model (XML tree, various forms of GODDAGs) can be generated from the lattice.
- Roundtripping between lattice and linear form is possible.
  - Example: “Spurious” overlap

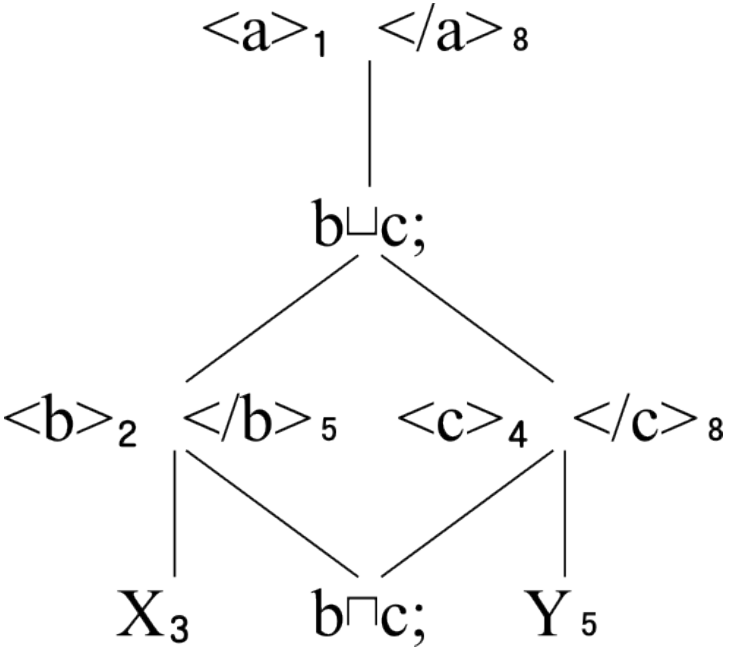
“Spurious” overlap

Closures of models

<a>X<c>Y</c><a>



<a>X<c>Y</c><a>



Algebraic characterization

- Observation: Tag typing
  - start-end, GI on start, GI on start and end, tag indexing
- Meet
- Join
- Quasi- and semi-elements
- Closure relations
- XML models constitute a subset of O-XML models

Algebraic characterization

- Observation: Tag typing
  - start-end, GI on start, GI on start and end, tag indexing
- Meet
- Join
- Quasi- and semi-elements
- Closure relations
- XML models constitute a subset of O-XML models
- Resolute stocls squinder polluparatizations of squid...

Thank you