Repair of XML documents

(w.r.t. given DTD)

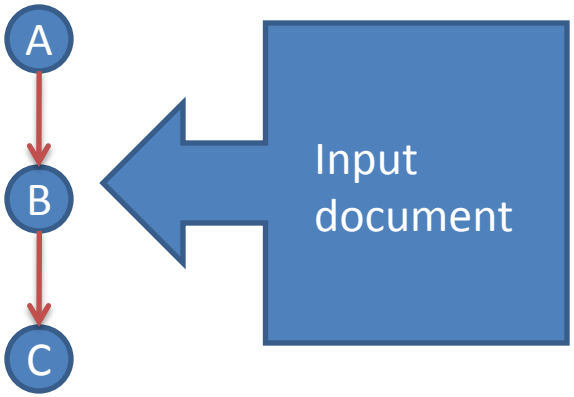
Outline of the problem

- What does it mean to „repair”?
  - Convert given xml such that it will be valid w.r.t to given DTD?
  - Should the conversion have some other features?
    - Should it use minimum possible number of operations?
  - What operations do we allow? How important this decision is?
  - How we can make potential user „happy” with our repair?

Why this problem is important?

- Integrating XML databases
  - Usually source DTD's known a priori
- Putting into existing XML database XML documents found in the Web
  - DTD's not known beforehand or even at all
  - XML's can be generated dynamically during crawling (to give structure to the data, to make it more usable – easier to search through, combine, aggregate)

Sample repairs

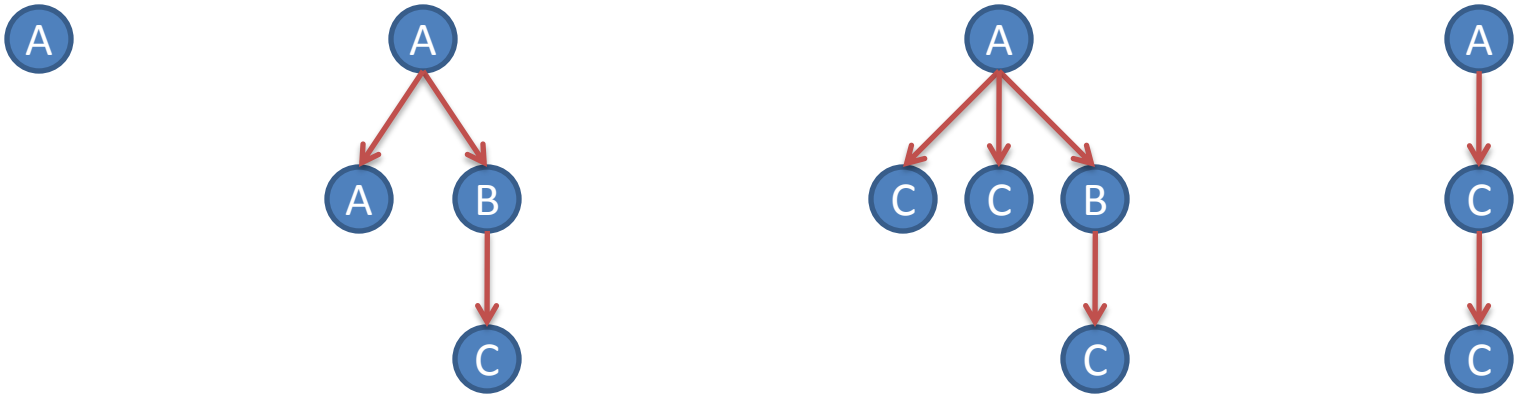


<!ELEMENT A ((((A | CC), B)*) | C)>

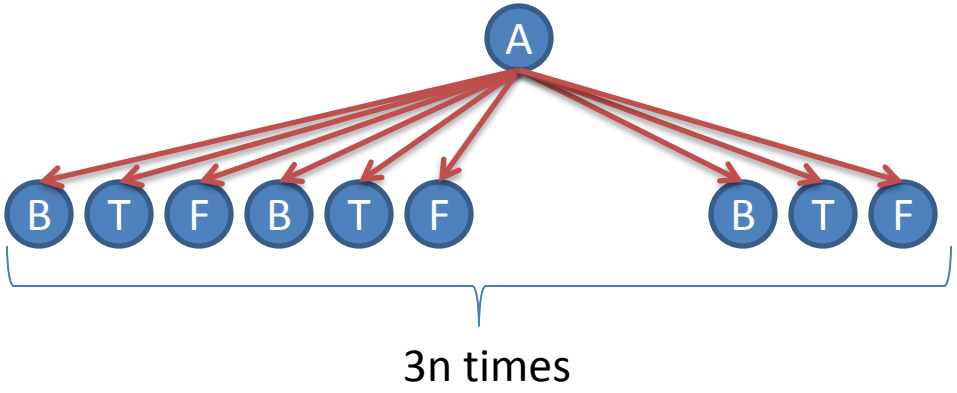
<!ELEMENT B (C)>

<!ELEMENT C (C?)>

Possible repairs



Sample repairs

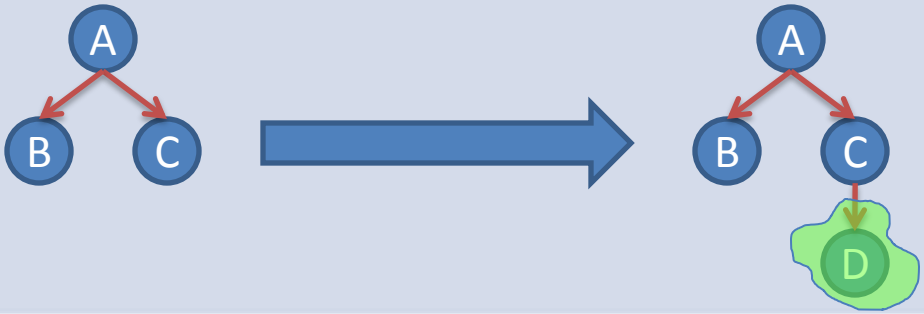



```
<!ELEMENT A ((B, (T | F))*)>
<!ELEMENT B (#PCDATA)>
<!ELEMENT T (#PCDATA)>
<!ELEMENT F (#PCDATA)>
```

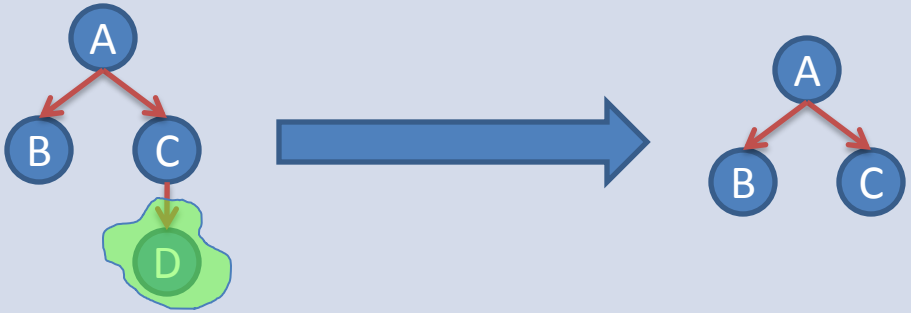
Even when considering only shorterst repair paths there is 2^n of repairs

Operations - basics

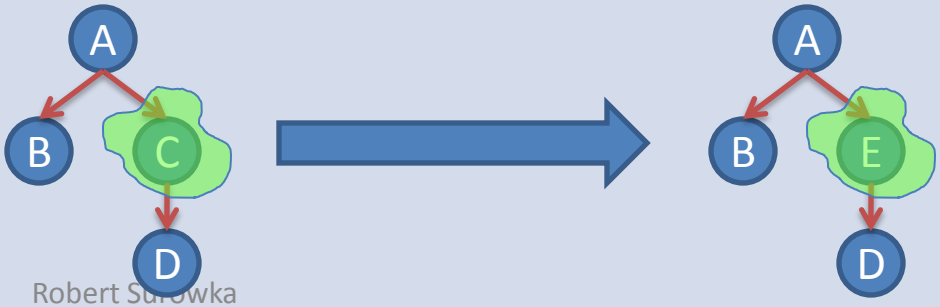
Add a leaf



Delete a leaf

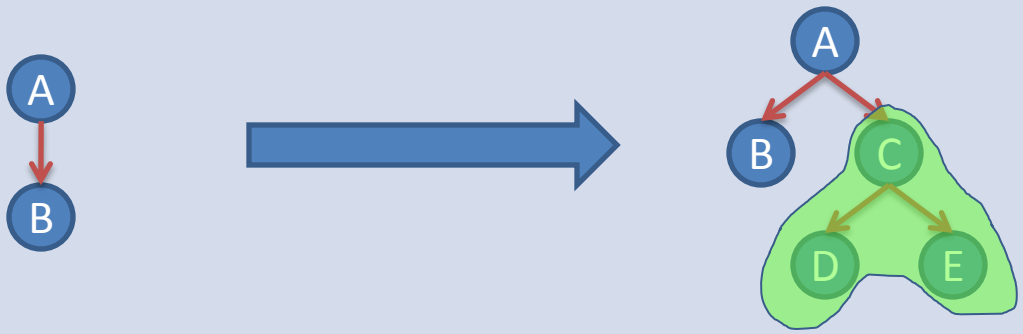


Rename a node

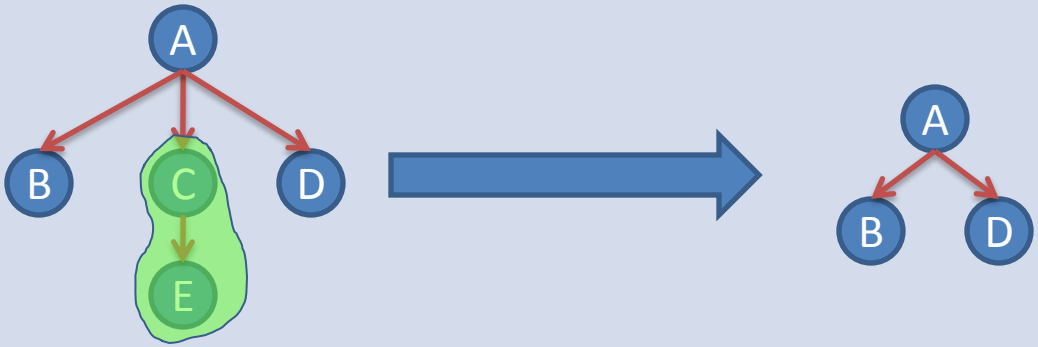


Operations - subtrees

Add a minimal subtree

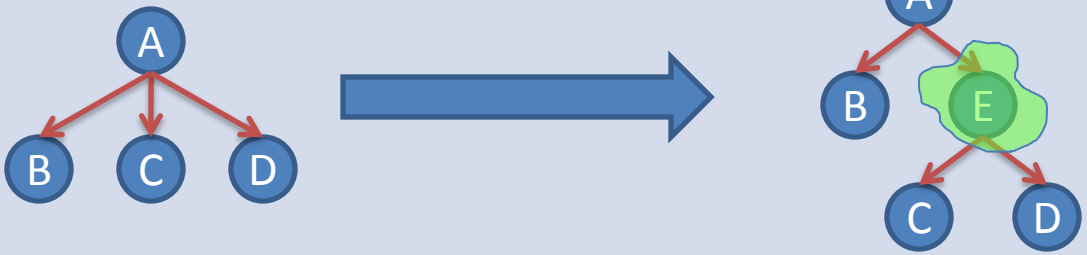


Delete a subtree

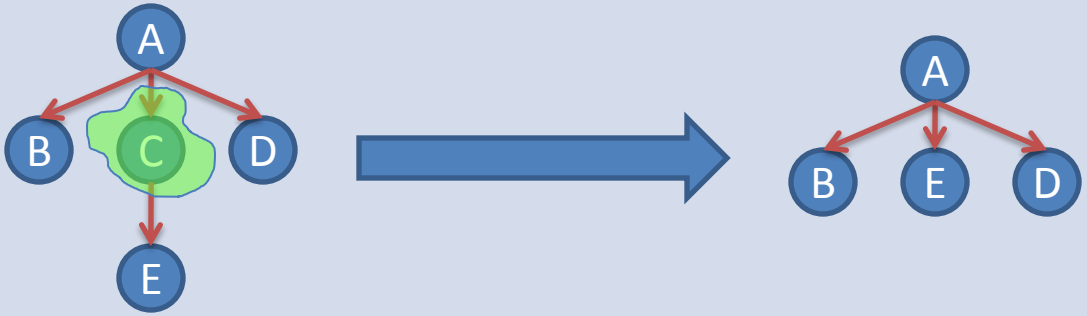


Operations - nodes

Add a node

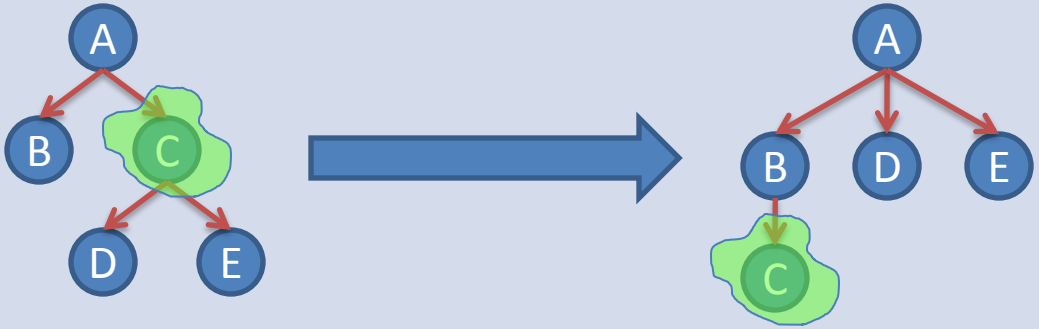


Delete a node

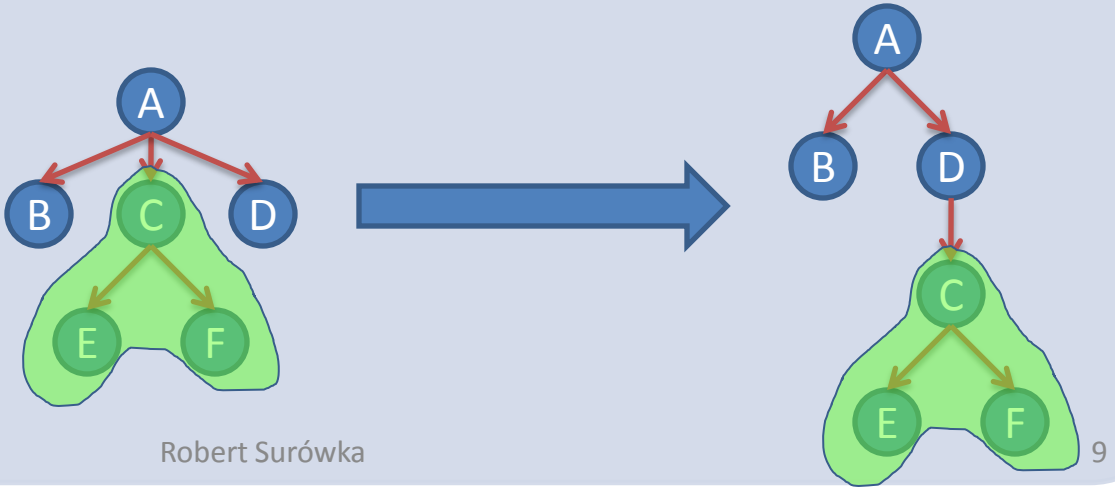


Operations - shifts

Shift a node



Shift a subtree



Sample existing algorithms

Sławomir Staworko

PhD dissertation

May 2007

Operations used:
- Add a subtree
- Delete a subtree
- Rename a node

Approximate Complexity:

$O(|t|\cdot(|S|^2|\Sigma|\cdot|R|+|S|\cdot|R|\cdot\lg(|S|\cdot|R|)))$

Nobutaka Suzuki

IPSJ Digital Courier Vol. 2

December 2006

Operations used:
- Add a node
- Delete a node
- Rename a node

Approximate Complexity:

$O(|\Sigma|^2w^4|t|^2r^2)$

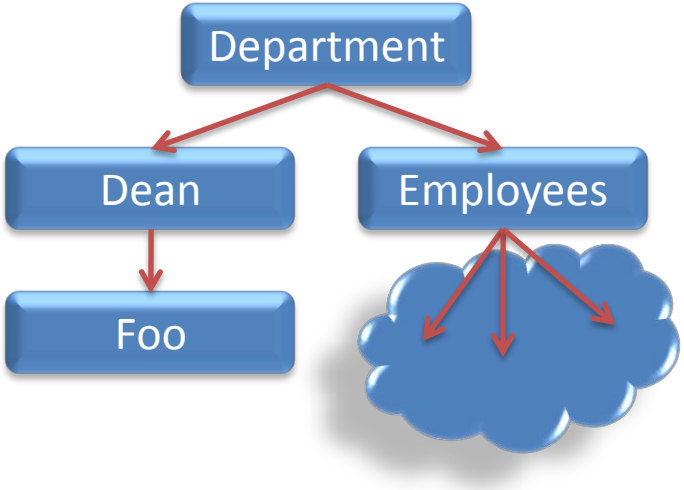
t – set of nodes of the given tree, S – parameter bounded by size of the DTD, R - maximum number of siblings in given tree, Σ - set of labels in the DTD, w – maximum degree of a node in the given tree, r – maximum length of a regular expression in the DTD

Conclusions so far…

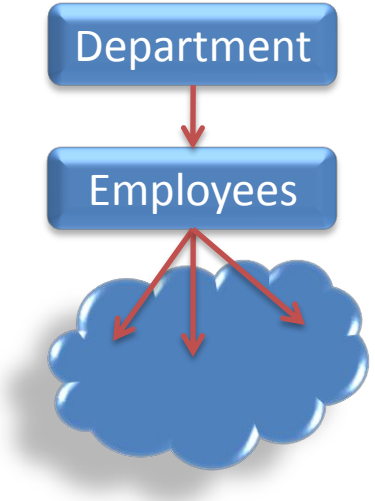
1. We have algorithms that find some shortest repair path of given XML document.
2. Operations that algorithm allows have a crucial impact on the repair.
   - If 2 algorithms have sets of supported operations such that neither of them includes the other then neither of the algorithms always finds a shorter repair than the other.
3. Complexity of already known algorithms is sufficiently fast for most uses.
4. But would the user be “happy” with what we already have?

Understanding an XML document

```
<!ELEMENT Department (Dean?, Employees?)>
<!ELEMENT Dean (Name?)>
<!ELEMENT Employees (Faculty?, Staff?, Name*)>
<!ELEMENT Staff (Name*)>
<!ELEMENT Faculty (Name*)>
<!ELEMENT Name (#PCDATA)>
```



There is a dean, whose name is “Foo”, in the department



This department doesn’t have a dean (e.g. he might have just resigned and new hasn’t been elected yet)

An unwelcomed repair – data corruption

<!ELEMENT Department (Dean?, Employees?)>
<!ELEMENT Dean (Name?)>
<!ELEMENT Employees (Faculty?, Staff?, Name*)>
<!ELEMENT Staff (Name*)>
<!ELEMENT Faculty (Name*)>
<!ELEMENT Name (#PCDATA)>

Operations: 1. Add a node 2. Delete a node 3. Rename a node

Moreover it seems that no matter what operations algorithm would support, always a data corruption may happen

Preventing “incorrect” repairs

There is a need to disallow repair paths that would “corrupt” data in an XML file.

Idea

1. User imposes constraints on possible repairs (e.g. a constraint “Dean node cannot be deleted and any subtree rooted at Dean node cannot be modified” would prevent the erroneous repair from previous example).
2. Repair attempt is undertaken, and either valid (w.r.t. given constraints) repair is done or information than no such repair exists is returned.

Realization

1. An appropriate constraint language needs to be defined.
2. A repair algorithm must be developed that would be able to work with those constraints.

Alternative solution

A lot of work is done in solving a problem stated like this:

"Given source DTD D_1, update script S changing it to DTD D_2 and XML document X in D_1 transform X so that it'll be in D_2"

So one could just define some source DTD's and edit scripts transforming them to DTD in the database. Then "correct" transformation of given XML's will be far more probable

E.g. Nobutaka Suzuki: On Inferring K Optimum Transformations of XML Document from Update Script to DTD. COMAD 2008: 210-221

Choosing language

Creating a new language

- The language will be well tailored to needs (and therefore it may be more compact and convenient)
- More work would be needed to create that language as well as tools for it

Use an existing language:

- Many of users will be already familiar with the language
- If for our needs some small tweaks to the language would be needed it may confuse the users
- The language can evolve in direction we won't like

Constraint language proposal

Expression	Meaning
A	Nodes A have to be preserved
A(B)	If node A has a child B, then in output both of them have to be preserved in that configuration
A((B))	If node A has a descendant B, then in output both of them have to be preserved in that configuration (but B may be e.g. promoted from grandchild to child)
A(B,C)	If node A has children B,C in that order, and there are no other siblings between B and C then in output the three of them have to be preserved in that configuration
A(B,*,C)	If node A has children B,C in that order then in output the three of them have to be preserved in that configuration
A(+,B,[1-2,5<],C)	If node A has children B,C in that order, and B has at least one left sibling and there are 1, 2 or more than 5 other siblings between B and C then in output that configuration has to be preserved (but, for example in input B may have exactly one left sibling D, but in output it may have 2 left siblings F,G).

Constraint language proposition

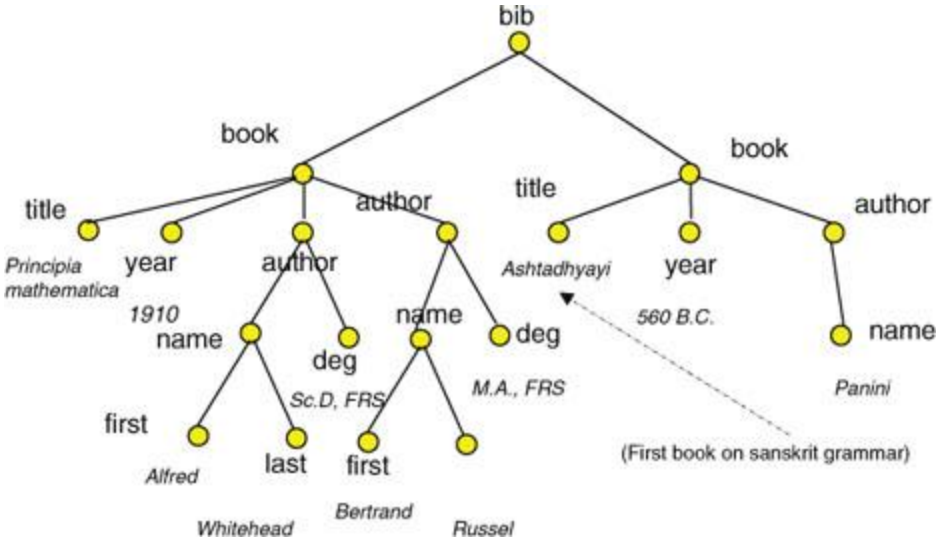
Special character or construction	Meaning	Example
~	Not exist	A(B,*,~C)
$	Doesn't matter in output	$A(B)
-	Has to be deleted	A(-B,C)
l	Level	A[l:<4,6](B)
s	Sibling index	A(B[s:<2,last])
ns	Number of siblings	A(B[s:3; ns:odd])
nc	Number of children	A[nc:1-10,~5]
nd	Number of descendants	A[nd:<=3]
nl	Number of leaves among descendants	A[nl:even,3]
t	Target	A(B[s:2; tc:<3])
id	Identifier	A[id:1]((B[l:>2*l(1)]))

Possible languages to use – Twig Query

- Twig query (also knows as tree pattern query)
  - It is a pair Q=(T,F) where T is node-labeled and edge-labeled tree with a distinguished x ∈ T and F is a boolean combination of constraints of nodes.
  - Node labels are variables like $x or $y
  - Edge labels are one of “pc” (parent-child) or “ad” (ancestor-descendent)
  - Constraints have form *$x.tag* = TagName or *$x.data* relOp *val*, where *$x.data* denotes the data content of node *$x*, and *relOp* is one of *=, <, >, ≤, ≥, ≠*.
  - In overall a Twig query (over an XML) is similar in concept to a selection condition in relational algebra

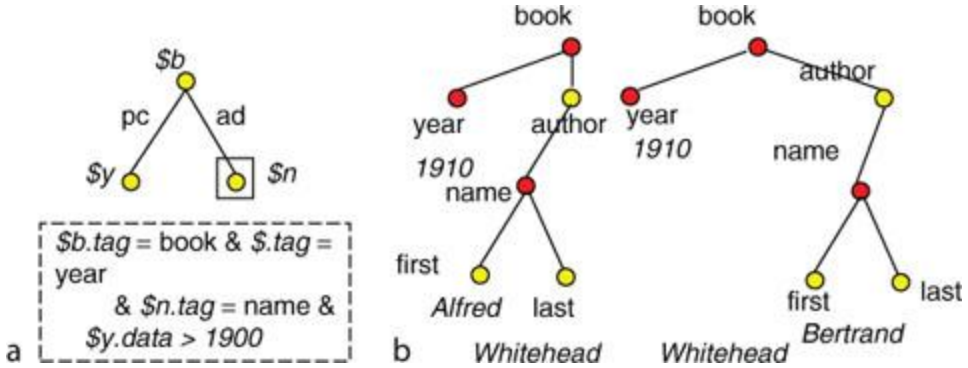
Source: Laks V. S. Lakshmanan: XML Tree Pattern, XML Twig Query. Encyklopedia of Database Systems 2009 : 3637-3640

Possible languages to use – Twig Query



Twig queries could in our project can be used to specify subtrees that cannot be changed by repairing algorithm

Twig query finds set of subtrees of a tree that conform to condition of the query



Source: Laks V. S. Lakshmanan: XML Tree Pattern, XML Twig Query. Encyklopedia of Database Systems 2009 : 3637-3640

Possible languages to use - XPath

- Xpath
  - There are many versions of Xpath:
    - Regular Xpath
    - First-Order Xpath
    - Aggregate Xpath
    - Aggregate XPath with position arithmetic
  - (Good source article about Xpath: http://portal.acm.org/citation.cfm?id=1456653)

Possible languages to use - Regular XPath

Regular Xpath query grammar:

$$\alpha ::\equiv \text{self} \mid \Downarrow \mid \Uparrow \mid \Rightarrow \mid \Leftarrow$$
$$f ::\equiv \text{lab}() = a \mid Q \mid \text{true} \mid \text{false} \mid \text{not } f \mid f \text{ and } f \mid f \text{ or } f$$
$$Q ::\equiv \alpha \mid [f] \mid Q/Q \mid Q \cup Q \mid Q^*$$

$[\![Q]\!]_t$ is the binary reachability relation on the nodes of tree t defined by the query Q

$$Ans(Q,t) = \{n \in N_t \mid (root_t, n) \in [\![Q]\!]_t\}$$

Possible languages to use - Regular XPath

Preservation constraints:

$Preserve_1(Q)$

$$(t, t') \models Preserve_1(Q) \text{ iff } Ans(Q, t) \subseteq N_{t'}$$

$Preserve_2(Q, Q')$

$$(t, t') \models Preserve_2(Q, Q') \text{ iff } Ans(Q, t) \subseteq Ans(Q', t')$$

$Preserve_3(Q, q')$

$$(t, t') \models Preserve_3(Q, Q') \text{ iff } [\![Q]\!]_t \subseteq [\![Q']\!]_{t'}$$

Possible languages to use - Regular XPath

Purity constraints:

$Pure_1(Q)$

$$(t, t') \models Pure_1(Q) \text{ iff } Ans(Q, t) \supseteq N_{t'}$$

$Pure_2(Q, Q')$

$$(t, t') \models Pure_2(Q, Q') \text{ iff } Ans(Q, t) \supseteq Ans(Q', t')$$

$Pure_3(Q, q')$

$$(t, t') \models Pure_3(Q, Q') \text{ iff } [\![Q]\!]_t \supseteq [\![Q']\!]_{t'}$$

Possible languages to use - Regular XPath

Problem of constraint satisfability:

$$\mathsf{SAT} = \{(t, \mathcal{I}, D) \mid Rep(t, \mathcal{I}, D) \neq \varnothing\}$$

is Exptime-hard. It means that in order to find practically usable algorithm it needs to be

- Probabilistic
- Approximate
- Or both

Future work

- Eventually defining the constraints language
- Getting an idea which constraints types are most usable for potential users (some survey?)
- Proposing a polynomial-time algorithm to solve the problem (or at least one being able for any problem instance to find an approximate solution with some probability)

Questions?