

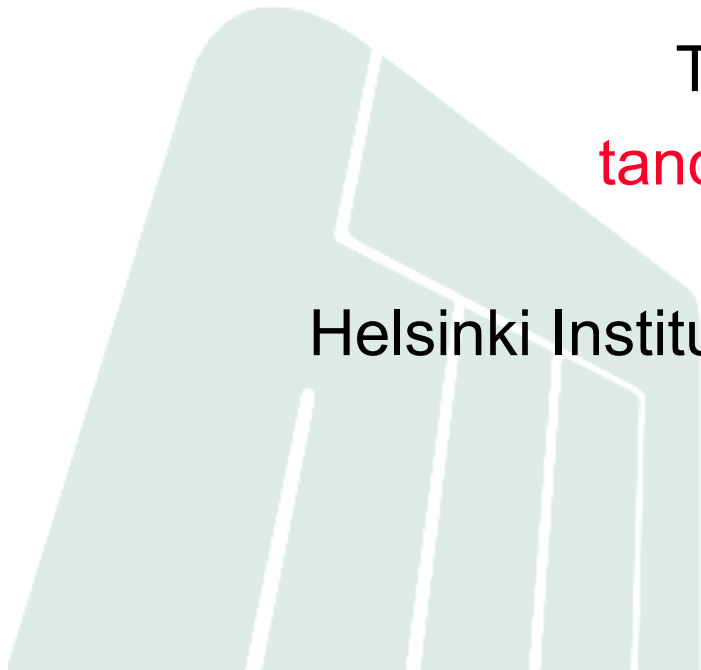
# A Three-way Merge for XML Documents

Tancred Lindholm

[tancred.lindholm@hiit.fi](mailto:tancred.lindholm@hiit.fi)

Helsinki Institute for Information Technology

<http://www.hiit.fi>



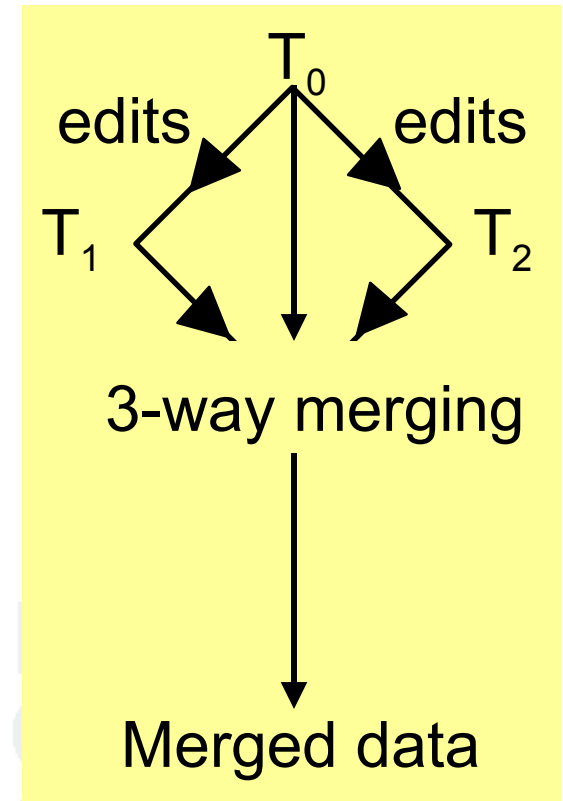
HELSINKI  
INSTITUTE FOR  
INFORMATION  
TECHNOLOGY

# Introduction

- **Research problem**
  - several copies of an XML document are independently modified
  - we want to integrate the edits into a single copy
- Line-based tools for merging ASCII text exist, but in general these are of limited applicability to XML
- We utilize the tree structure of the XML document
- We attempt to define a practically useful merge for **document-oriented XML**, based on the assumption that such XML has enough common traits
- No single “right merge”

# Three-way Merging

- Assume  $T_0$  is the *base* data set. Two copies of  $T_0$  are modified, yielding  $T_1$  and  $T_2$ .
- $T_1$  and  $T_2$  can be reconciled by a three-way merge of  $T_0$ ,  $T_1$  and  $T_2$
- The **three-way merge**  $T_m$  is the result of applying the changes between  $T_0$  and  $T_1$  as well as between  $T_0$  and  $T_2$  to  $T_0$ .
- Changes are **detected** by comparing  $T_0$  and  $T_1$  as well as  $T_0$  and  $T_2$  →  
no edit history needed



# Example 1: Structured Document

$T_0$ : <doc>  
 <sect title="Jokse" />  
 <sect title="Student joke">  
 <p>Q: How many students does it take to change a light bulb?</p>  
 <p>A: None. Light bulb changing isn't part of the course.  
 <footnote text="Except for projector bulbs" /></p>  
 </sect>  
</doc>

$T_2$ : <doc>  
 <sect title="Jokse" >  
 **a** <p>Here are several good jokes</p>  
 <sect title="Joke 1: Student joke" >**c**  
 <p>Q: How many students does it take to change a light bulb?</p>  
 <p>A: None. Light bulb changing isn't part of the course.  
 <footnote text="Except for projector bulbs" /></p>  
 </sect>  
 </sect>  
</doc>

$T_1$ : <doc>  
 <sect title="Jokes" />**d**  
 <sect title="Student joke">  
 <p>Q: How many students does it take to change a light bulb?</p>  
 <p>A: None. Light bulb changing isn't part of the course.</p>  
 **e** <p>A2: "Will this be on the test?"</p>**f**  
 </sect>  
</doc>

**a** **f** Insert: answer, <p>

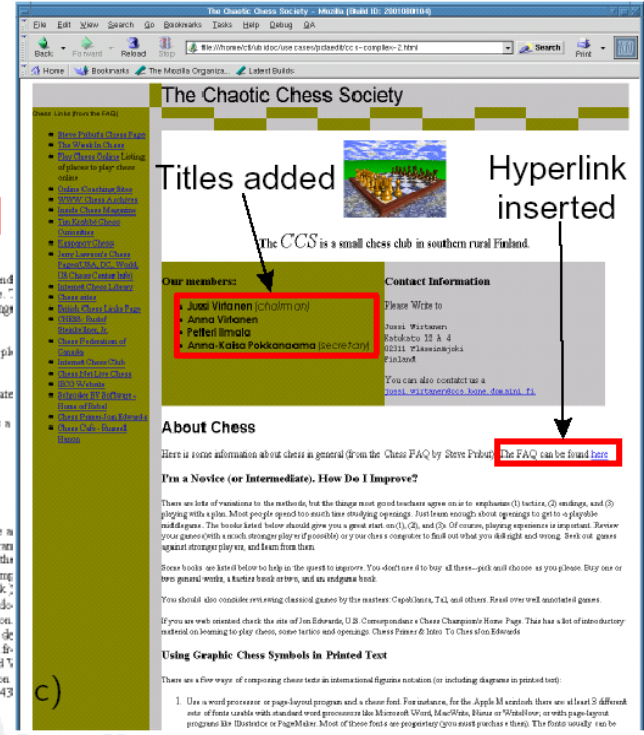
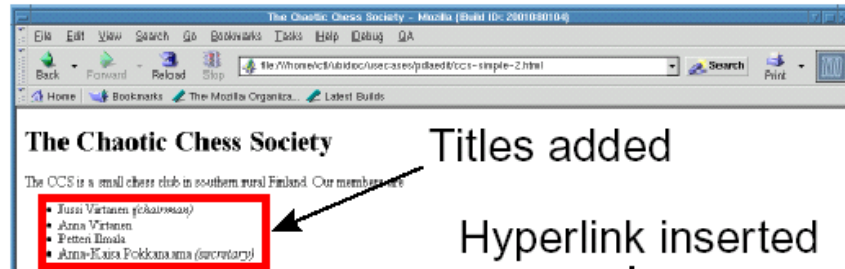
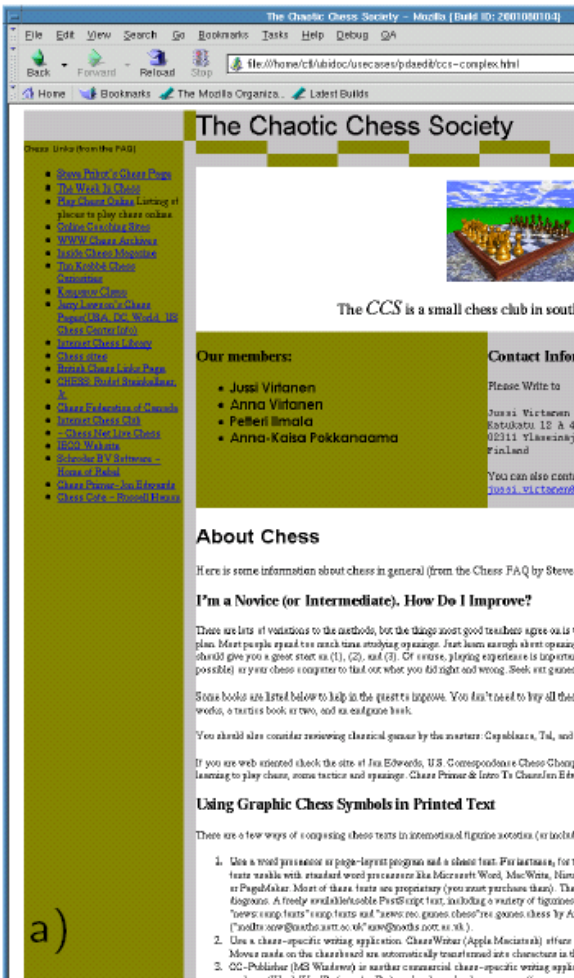
**b** **d** Updates: typo, title

**e** Delete: footnote

**c** Move: section to subsection

# Example 2: Web Page Variants

- Changes merged from simple to complex variant



# Merge Rules from Use Cases

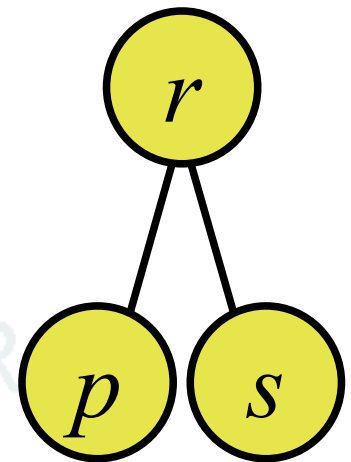
- Many ways to merge XML may be contrived
  - We wanted a way that is **useful in practice**
  - To find merging patterns we devised a number of small and large use cases:
    - 37 limited “study cases” on merging situations
    - 5 more extensive and realistic use cases
3. Concurrent editing of an OpenOffice document
  4. Merging changes between documents sharing XML fragments
  5. Keeping an inlined SVG drawing up-to-date
  6. Maintaining a shared shopping list
  7. Merging between web page variants (Example 2)

# Rules we Found in the Cases

- We need to know which nodes are the “same” across trees (rather obvious)
- Content **updates** in  $T_1$  or  $T_2$  should appear in  $T_m$
- An **inserted** node in  $T_1$  or  $T_2$  should appear in  $T_m$
- A node **deleted** from  $T_1$  or  $T_2$  should not appear in  $T_m$
- **Moved** nodes imply certain node contexts in  $T_m$ 
  - a *node context* is a child list fragment of a node
  - If  $n$  is moved from the child list  $anb$  of  $p_1$  to  $cnd$  of  $p_2$ , the contexts  $\dots ab\dots$  of  $p_1$  and  $\dots cnd\dots$  of  $p_2$  should appear in  $T_m$
- Some additional contexts (e.g. around deletes) to guard against merging changes that are “too close”

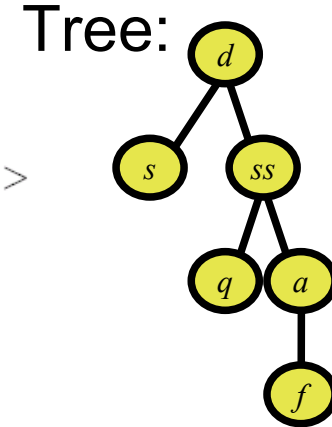
# XML Tree Model Used for Merging

- XML documents modelled as labelled, ordered trees
- Each tree node has a content corresponding to the XML fragment the node models:  $c(n, \langle \text{xml-content} \rangle)$
- For the purpose of merging we express the document tree as a set of *parent-child-successor* (PCS) relations
  - $\text{pcs}(r, p, s) := r$  is the parent of  $p$  and  $s$ , and  $p$  is next to  $s$ . Special symbols:
    - [ :=start-of-list ] :=end-of-list,
    - . :=artificial root
  - PCS relations model **node contexts**

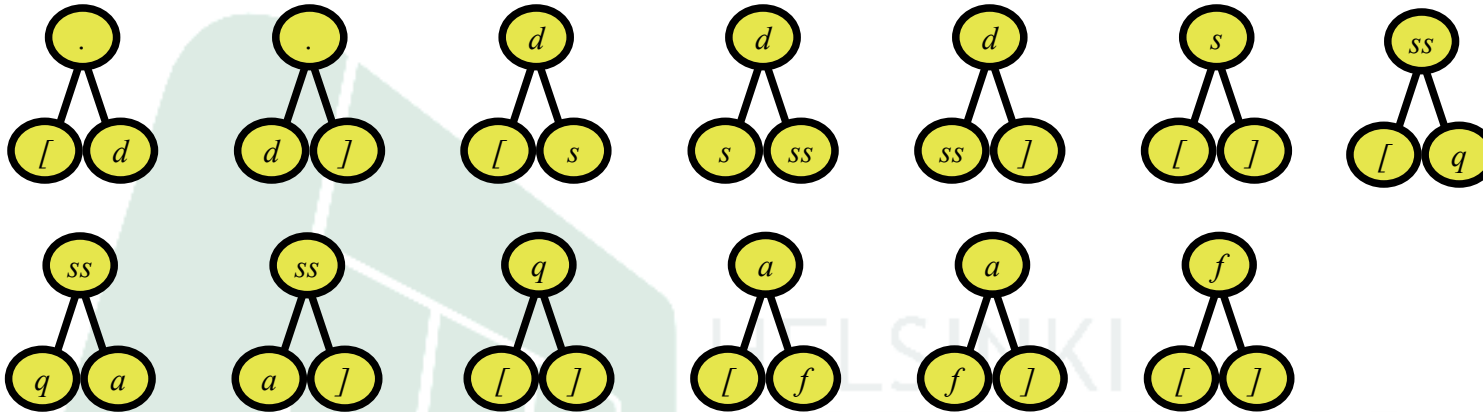


# Example 1 Tree and Relations

$T_0$ : `<doc d`  
`<sect title="Jokse" s`  
`<sect title="Student joke" ss`  
`<p>Q: How many students does it take to change a light bulb q</p>`  
`<p>A: None. Light bulb changing isn't part of the course a</p>`  
`<footnote text="Except for projector bulbs" f /></p>`  
`</sect>`  
`</doc>`



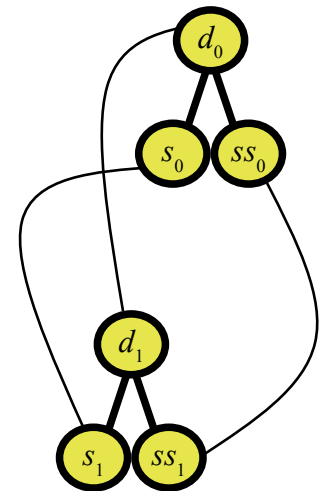
PCS relations:



Content relations:  $\{c(d, \langle doc \rangle), c(s, \langle sect \ title="Jokse" \rangle), c(ss, \langle sect \ title="Student \ joke" \rangle), \dots\}$

# Matching Tree Nodes

- To relate the “same” node in the different trees we build a matching of the nodes
- We assume that the matching relation is an equivalence relation, thereby partitioning the nodes into equivalence classes
- Each class represents the “same” node in the input trees
- Maximally one node from each tree in each class (we disallow *copies* and *glues*).
- Notation used here: same label, different subscript are matched nodes.  
The class  $\{n_0, n_1, n_2\}$  is denoted  $n$ .



# Changes

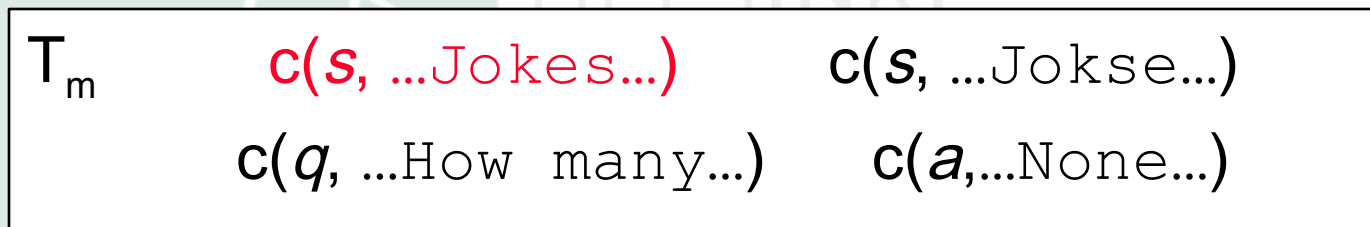
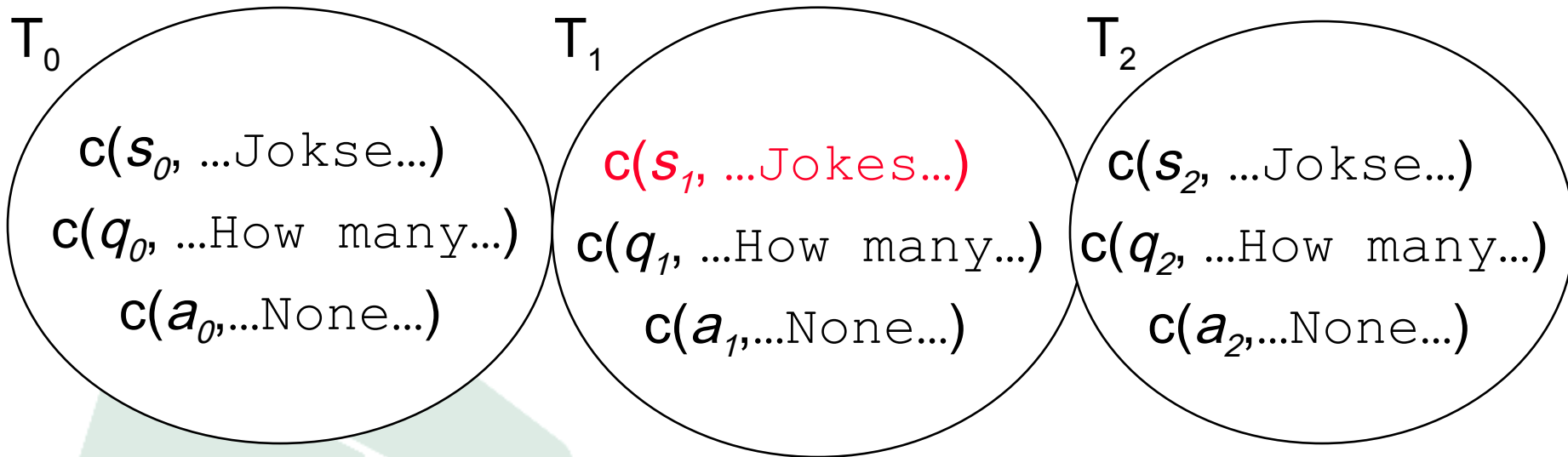
- By the definition of three-way merging, we need to merge *changes* from the modified trees
- We define two kinds of changes
  - **Content change:** change the content of a node (from an unspecified initial state)
  - **Structural change:** arrange nodes into a PCS relationship (from an unspecified initial state)
- We re-use the pcs() and c() relations for this purpose
- Thus, the set of content and PCS relations expressing a tree T is also a set of changes that yields T when applied to any initial tree.

# Change Consistency and Edits

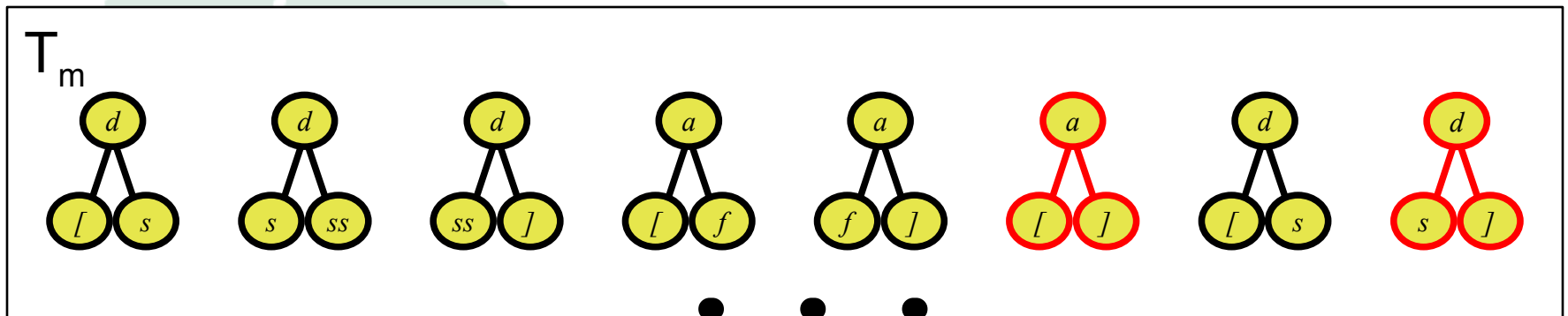
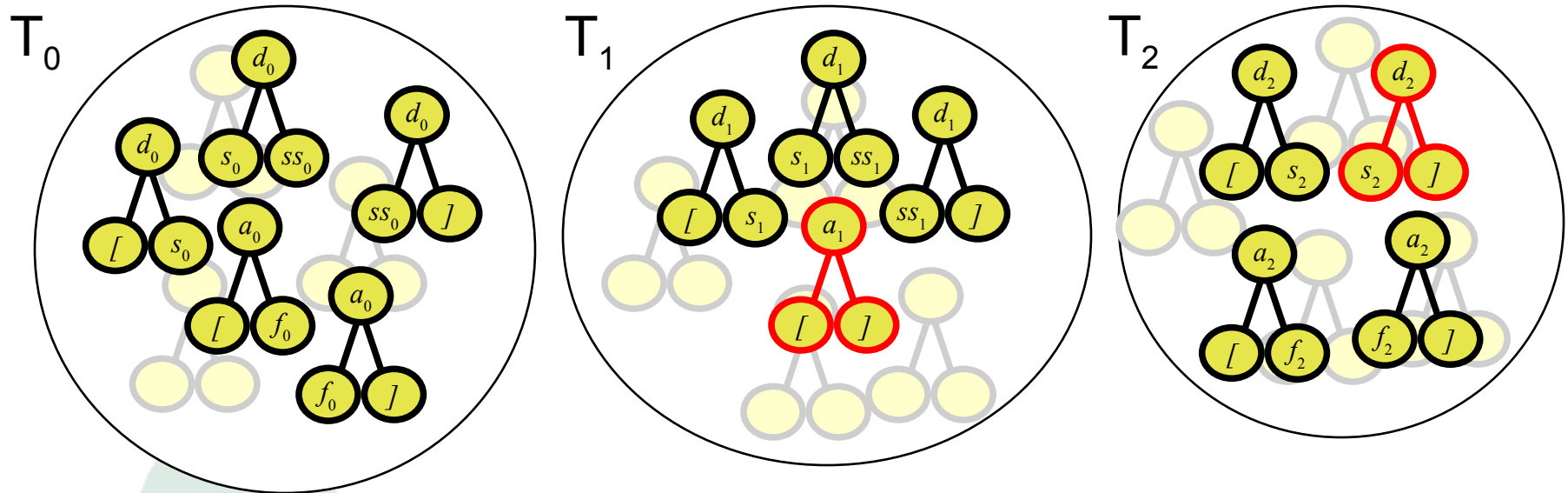
- When combining the changes from the input trees into a unified change set we need to ensure that the set is *consistent*, i.e. that each node has
  - A unique content
  - A unique parent
  - A unique predecessor
  - A unique successor
- In cases of inconsistencies we need to arbitrate between discardable and non-discardable changes
- Changes not in  $T_0$  need to be preserved. We call such changes **edits**

# Merge Step 1: Raw Merge (content)

- First we combine the change sets, equating nodes by their equivalence class, into a "raw" merge (edits in red)



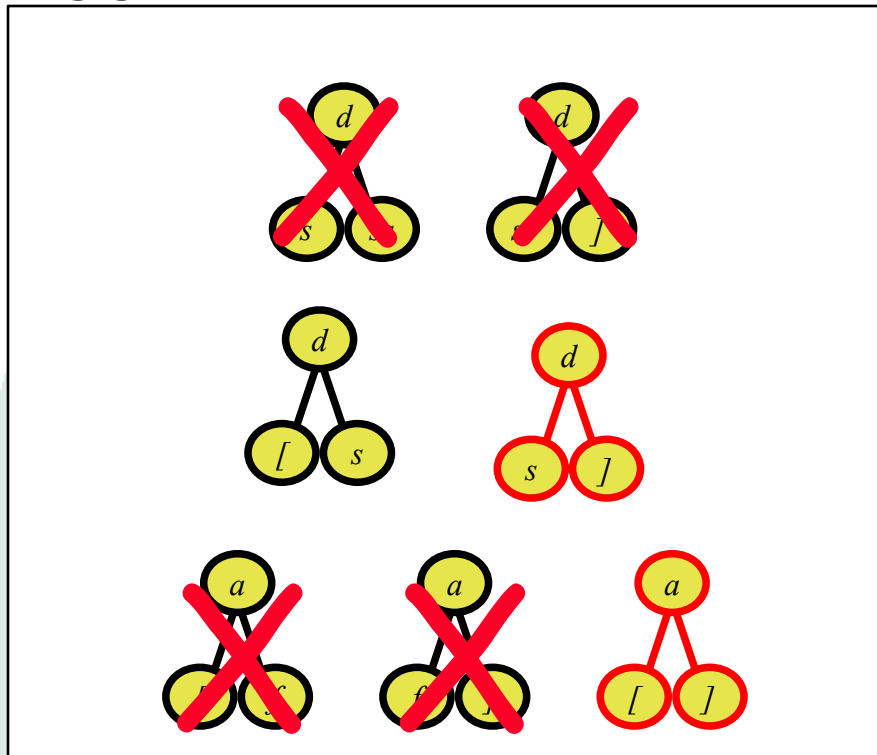
# Merge Step 1: Raw Merge (PCS)



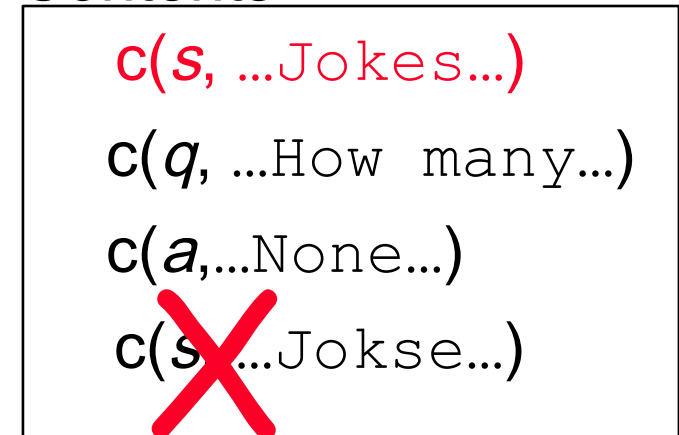
# Merge Step 2: Remove Inconsistencies

- Resolve inconsistencies by removing changes that are not edits

PCS

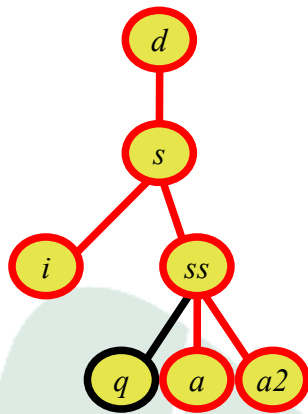


Contents



# The Merged Tree

- The merged tree



## Contents

```
c(d,"")
c(s,<sect title="Jokes" ...>
c(i,<p>Here...>
c(ss,<sect title="Joke 1...>
c(q,<p>Q: How...>
c(a,<p>A: None...>
c(a2,<p>A2: "Will...>
```

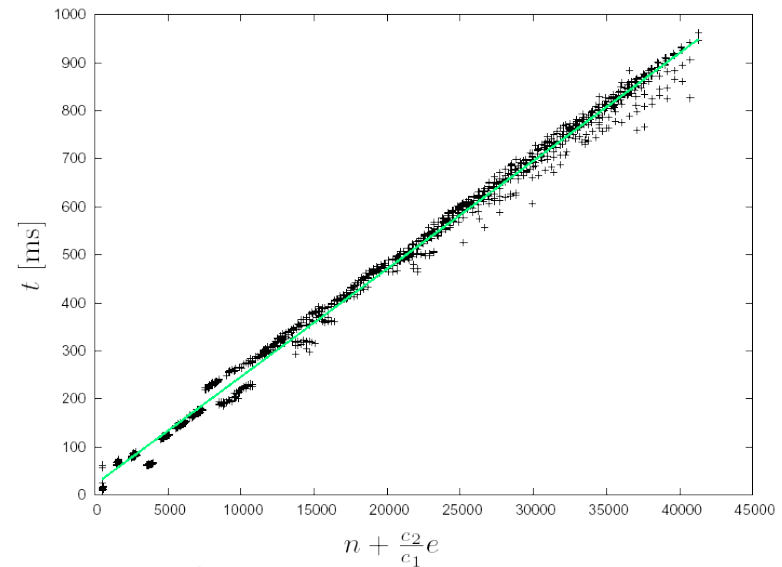
```
Tm: <doc>
  <sect title="Jokes">1
    <p>Here are several good jokes</p>
    <sect title="Joke 1: Student joke">2
      <p>Q: How many students does it take to change a light bulb?</p>
      <p>A: None. Light bulb changing isn't part of the course.</p>
      <p>A2: "Will this be on the test?"</p>
    </sect>
  </sect>
</doc>
```

# Conflicts

- The "conflict vocabulary" is an important part of a merge
- We try to identify intuitively meaningful conflicts
  - Do not find conflicts when there are none
  - Do not automatically resolve ambiguous situations
- Our merge identifies:
  - **Update/update**: node content is updated in both  $T_1$  and  $T_2$
  - **Position/position**: a node has been differently positioned in  $T_1$  and  $T_2$
- In addition we may detect **delete/edit**: edits in a part of the tree that is deleted

# Evaluation of the Merge

- 35 of 37 study cases successful
- Elaborate cases identified some issues
  - Metadata that changes inconsistently
  - Similar structure needed in areas of change
  - Changes that are "too close" to another
- Empirical performance appears linear on average
- We found overall performance to be quite satisfactory
- Implementation at <http://tdm.berlios.de>



# Properties of the Merge

- **Symmetry**

The merge result is independent of assignment of changed trees to  $T_1$  and  $T_2$

- **Preservation of Edits**

All edits are preserved if we consider delete/edit to be a conflict

- **Parallel edit model**

Detected edits are inherently parallel

- **Extensibility**

Has been extended for copies, should generalize to *n*-way merge

HELSINKI  
INSTITUTE FOR  
INFORMATION  
TECHNOLOGY

# Conclusions

- A generic merge for document-oriented XML
- Supports merging of structural changes
  - Merges “reorganization in the large” with “changes to the details”
- We hope it shall prove useful in practice
- Application to our own use cases was successful overall
- The use cases and merge rules should prove useful for the design of other merges