

# Merging of ordered trees

Lars Georg Techau Jørgensen [larsj@diku.dk](mailto:larsj@diku.dk)

13. november 2002

# Scope

- We will be looking at 3-way merging.
- Merging the differences between two trees  $T_1$  and  $T_2$  derived from the same tree  $T_B$  into a fourth tree  $T_M$ .
- XML can be seen as ordered trees.

## (3-way) tree merging

- Finding the operations used to construct  $T_1$  and  $T_2$ .
- Selecting which operations to propagate to  $T_M$ .
- Applying them.

# Design parameters

- Is a edit history available?
- What are the edit operations?
- How is data represented?
- Use semantic knowledge?

## Edit script

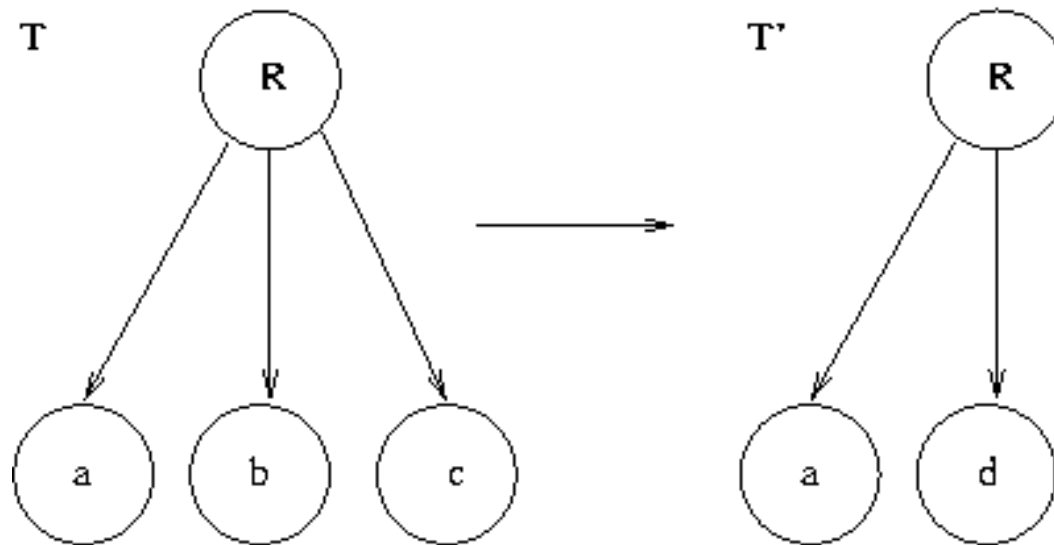
*Edit script*: A sequence of operations that can transform  $T$  into  $T'$ .

*Edit distance*: The minimum cost of all possible edit scripts transforming  $T$  into  $T'$ .

## Edit operations

- $delete(x)$   
Delete the node  $x$ , and insert the child of  $x$  between the left and the right neighbors of  $x$ .
- $insert(x, y, n, s, t)$   
Insert the node  $x$  as the  $n$ :th child of node  $y$ , making the children  $s$  through  $t$  of  $y$  children of  $x$ . This is the inverse of the  $delete$  operation.
- $update(x, y)$   
Update the content of  $x$  to  $y$

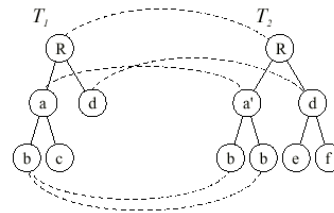
## Example



Edit script 1: `delete(b); delete(c); insert(d, R, 2, 2, inf)`

Edit script 2: `update(b,d); delete(c)`

# Matching



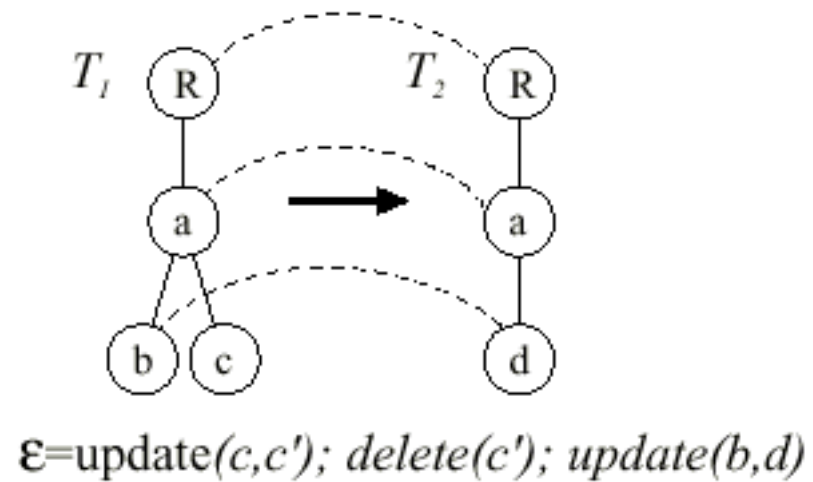
- **Matching** finds the correspondence between the nodes of two trees (lacking unique identities).
- Implicitly as a list of edit operations
- Explicitly as a list of pairs of nodes.
- Content (moved), structural (updated) and full matches.

## Matching (cont.)

- Node not matched in  $T_1 \rightarrow$  deleted
- Node not  $T_B \rightarrow$  inserted
- Node  $a \in T_B$  and  $a'$  in  $T_1 \rightarrow$  update

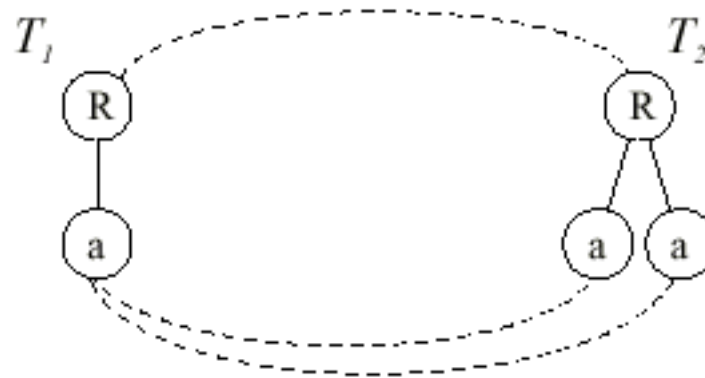
# Matching vs. Edit scripts

Loss of edit information:



## Matching vs. Edit scripts II

Not expressible using insert/delete/update:



## Other operations than insert/deletes?

- Enables us to create smaller diffs.
- More natural (?)
- More expressive.

# Merging

- Apply all non-conflicting changes from  $T_1$  and  $T_2$  and get  $T_M$

## The problem 3DM should solve

How can we, with minimal application support, synchronize related structural data in a weakly coupled environment?

## 3DM - Hypothesis

Provided the structure of the trees  $T_1$ ,  $T_2$  and  $T_B$  reflects the structure of the data they represent, it is in most cases possible to create a semantically valid structural 3-way merge without the merge tool having knowledge of the semantics of the trees.

## 3DM - Goals

- The operations of tree merge should be easy to understand. → No unpredictable behavior.
- Any change to a node in either branch, ..., should be present in the merged tree.
- Node operations should be considered relative rather than global. Reordering of children of a moved root node should not decouple the children from the root node. The should me moved relative to the root node.

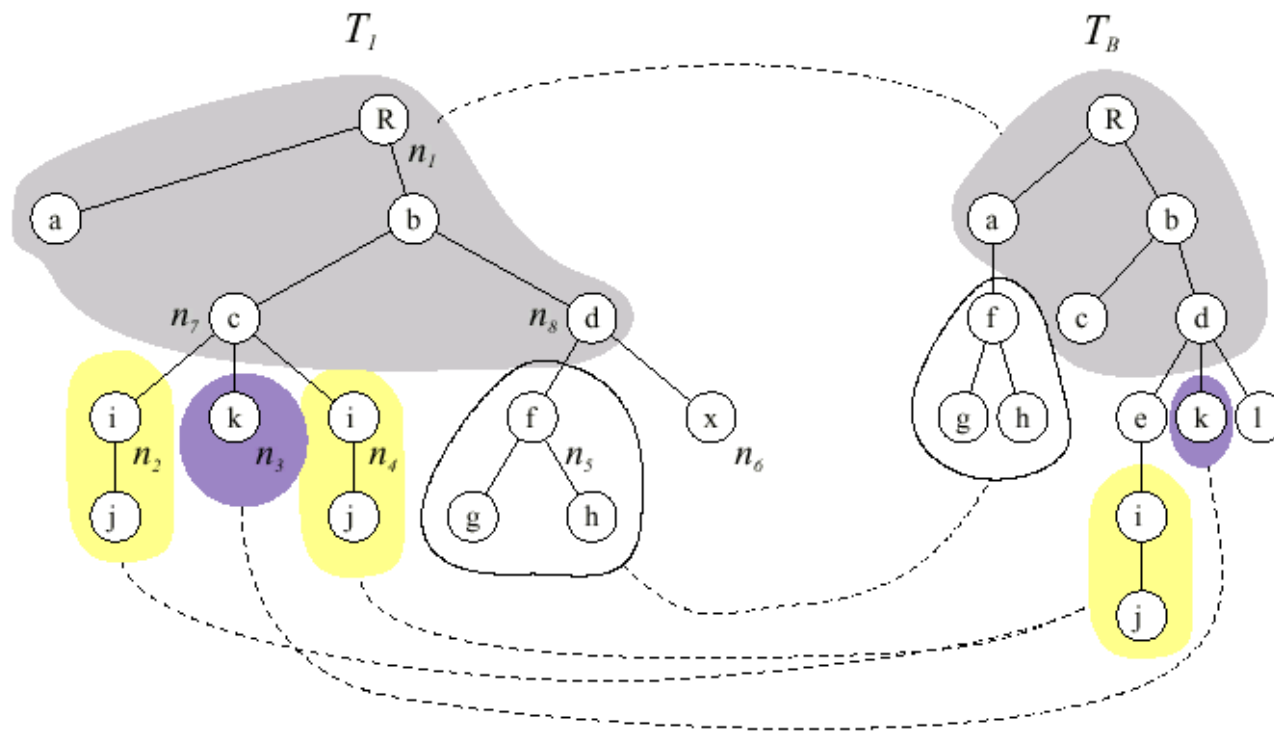
## 3DM - Goals (cont.)

- Wants to preserve the context in the merged tree. Moving a paragraph in between another pair of paragraphs that are modified in  $T'$  should not be allowed.
- Merging of extended child lists.
- Symmetric merge.  $3dm\ 1\ 2\ b$  equivalent to  $3dm\ 2\ 1\ b$
- Should detect conflicts

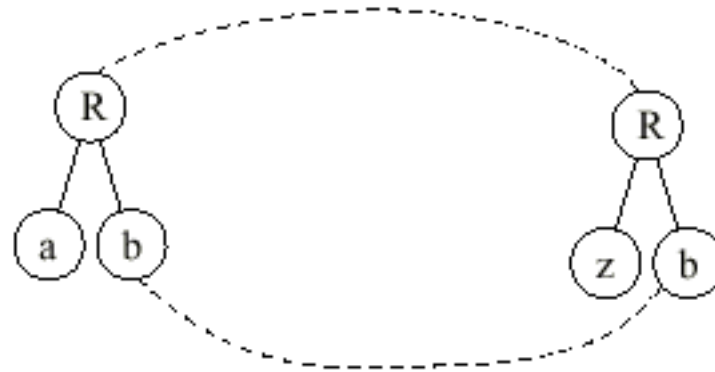
# Matching in 3DM

- Uses a heuristic to match nodes:
  1. Greedily find the largest subtree.
  2. Match unmatched nodes that are similar or simliar in context.
  3. Remove matches that cause small amounts of data to be copied.

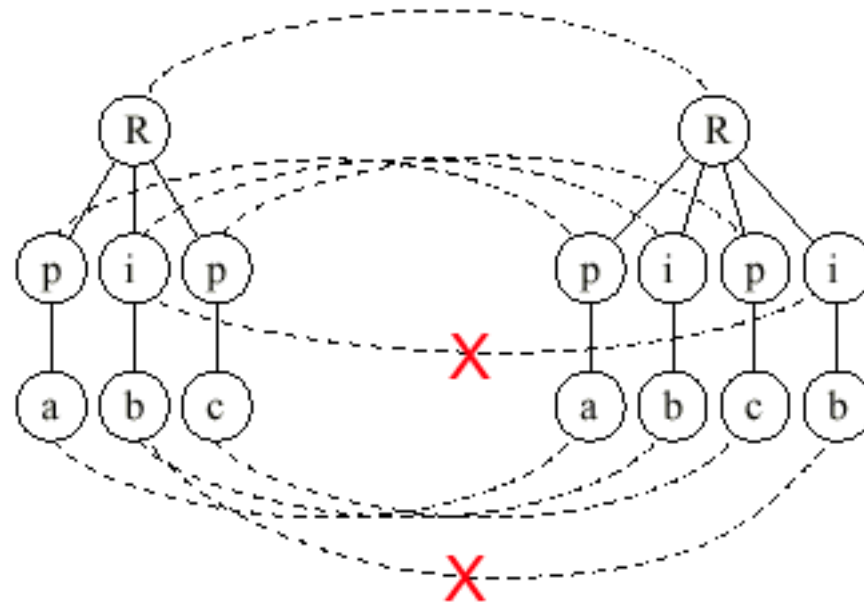
# Greedy find the largest subtree



**Match unmatched nodes that are similar or similar in context**



# Remove matches that cause small amounts of data to be copied



## Merging in 3DM

- Does not generate a edit script.
- Generates a merge list based on a cursor.
- A node is moved if it has another parent or predecessor.
- A node is copied if its moved an there is more then one copy.

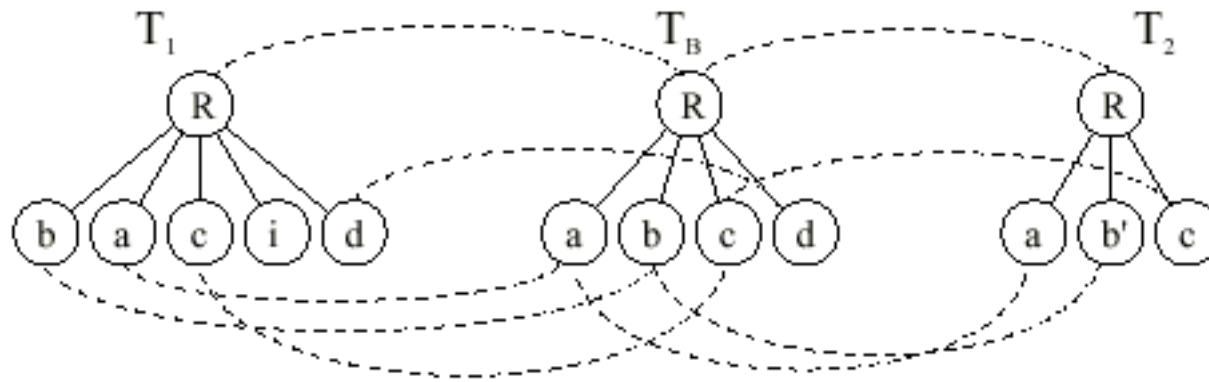
## Cursor movement

Match type of $(u_i, b_k)$	Match type of $(v_j, b_k)$	Assignments of $C_1$ and $C_2$
$u_i$ is not present	Any	$C_1 = \emptyset, C_2 = v_j$
Any	$v_j$ is not present	$C_1 = v_i, C_2 = \emptyset$
Content	Structural	$C_1 = u_i, C_2 = \emptyset$
Content	Content	$C_1 = u_i, C_2 = v_j$
Structural	Content	$C_1 = \emptyset, C_2 = v_j$
Structural	Structural	$C_1 = u_i, C_2 = v_j$

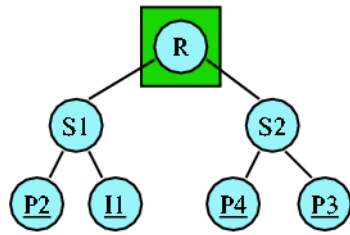
## Conflicts and propagation

Operation in $u_i$	Operation in $v_j$	Insert
-	-	$(a, v) \in u_i$
-	Delete	-
-	Change	$(a, v) \in v_j$
Delete	-	-
Delete	Delete	-
Delete	Change	Conflict
Change	-	$(a, v) \in u_i$
Change	Delete	Conflict
Change	Change	Conflict

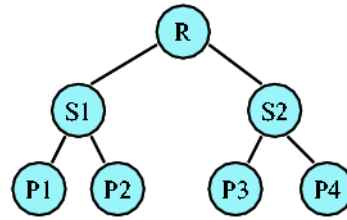
## Example (matching)



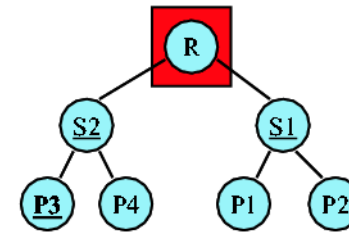
# Example



Branch A



Base

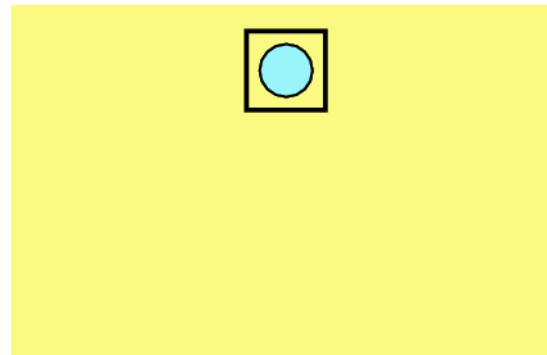


Branch B

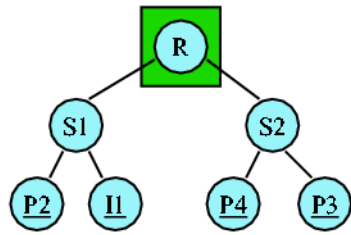
Child lists to merge



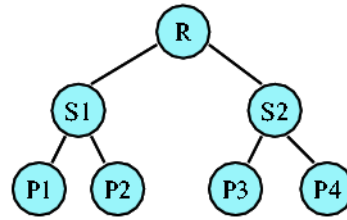
Merged List



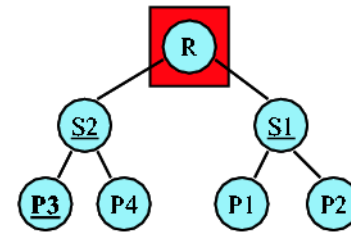
# Example



Branch A



Base

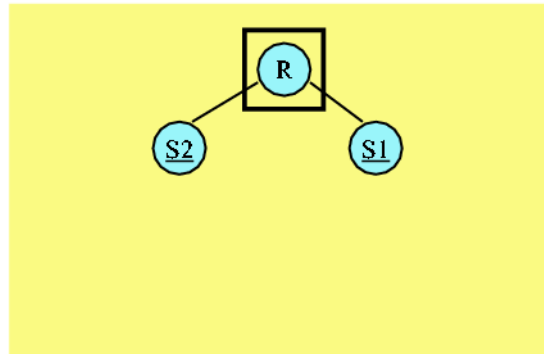


Branch B

Child lists to merge

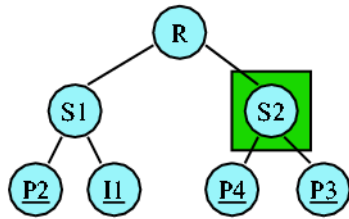


Merged List

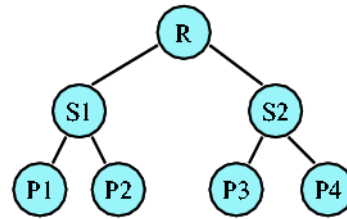


Recurse for S2

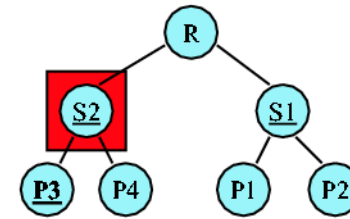
# Example



Branch A



Base

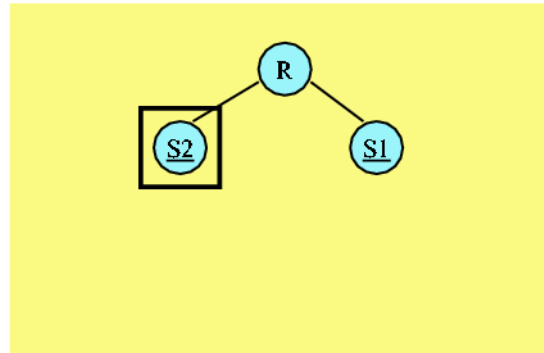


Branch B

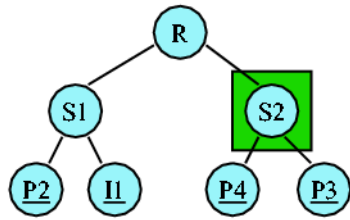
Child lists to merge



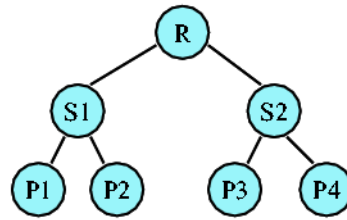
Merged List



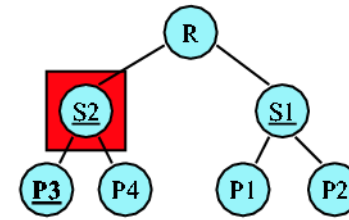
# Example



Branch A



Base

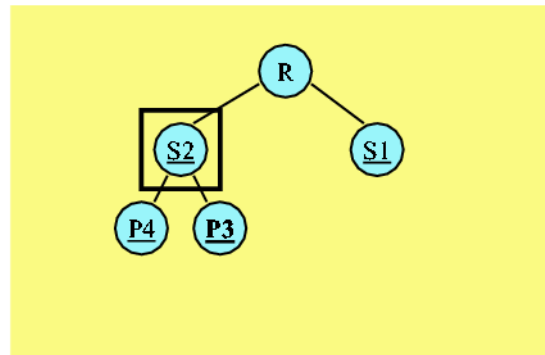


Branch B

Child lists to merge

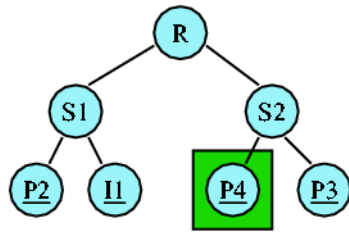


Merged List

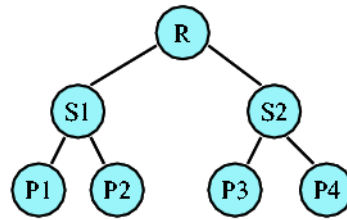


Recurse for P4

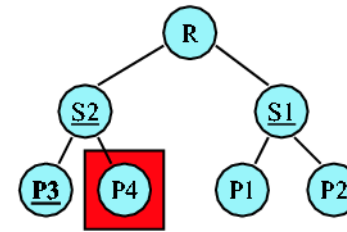
# Example



Branch A



Base

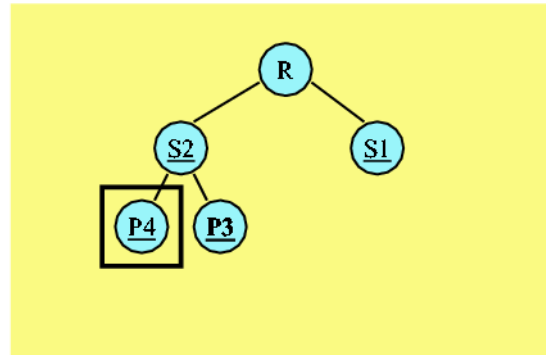


Branch B

Child lists to merge

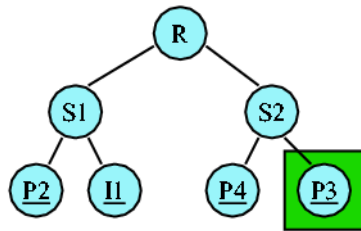


Merged List

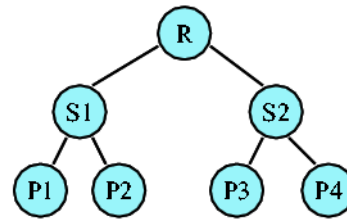


Recurse for P3

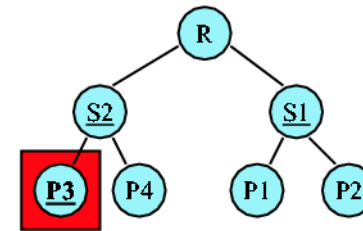
# Example



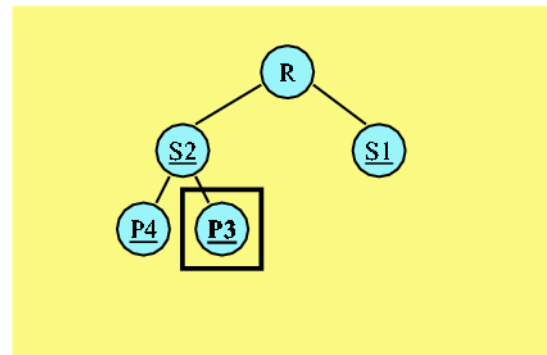
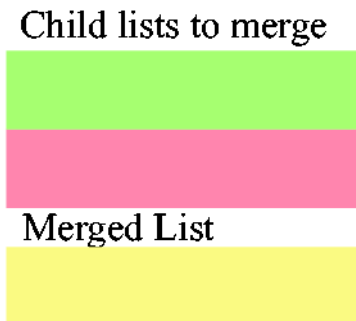
Branch A



Base

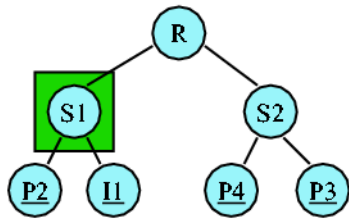


Branch B

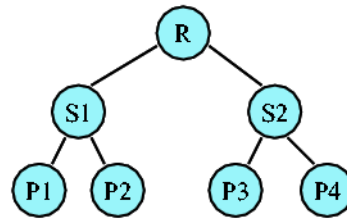


Recurse for S1

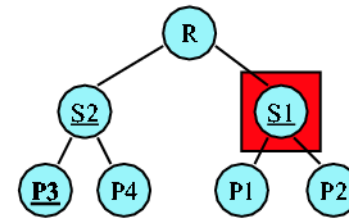
# Example



Branch A

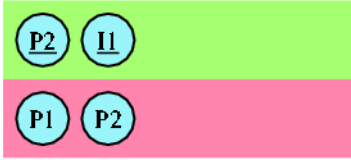


Base

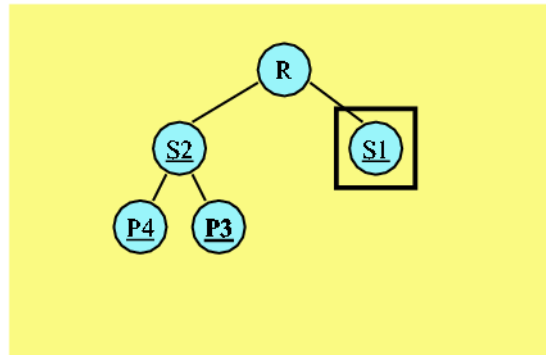


Branch B

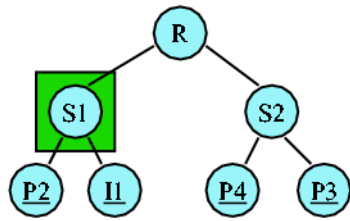
Child lists to merge



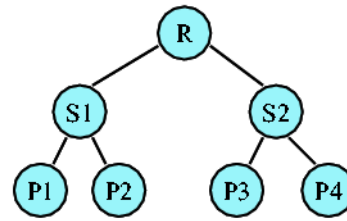
Merged List



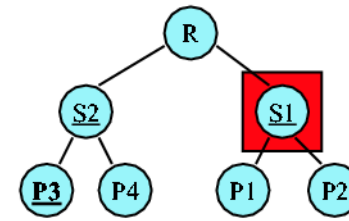
# Example



Branch A



Base

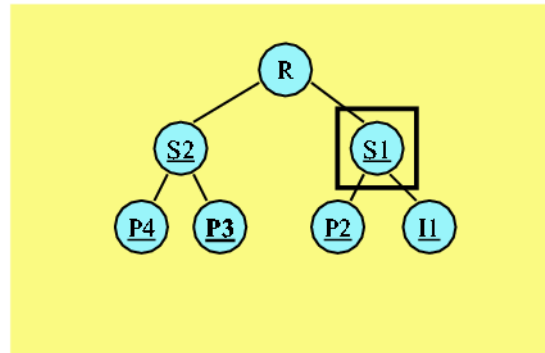


Branch B

Child lists to merge

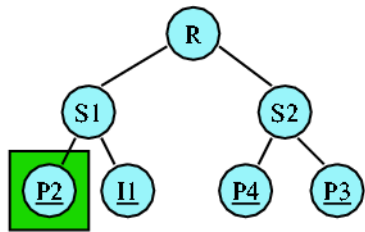


Merged List

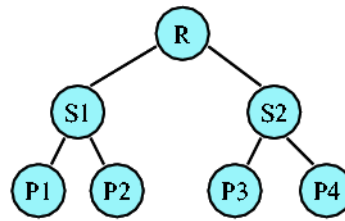


Recurse for P2

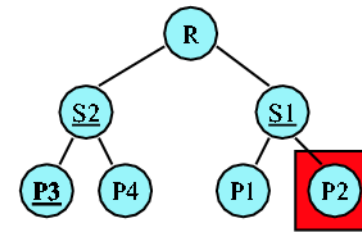
# Example



Branch A

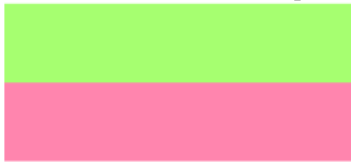


Base

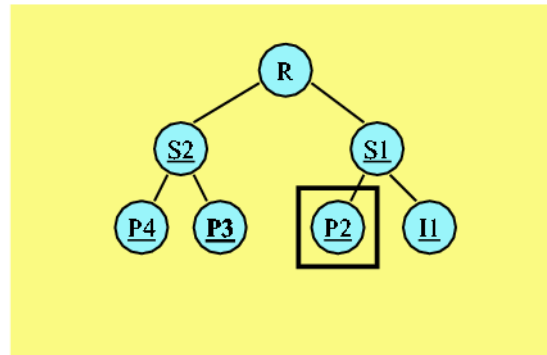


Branch B

Child lists to merge

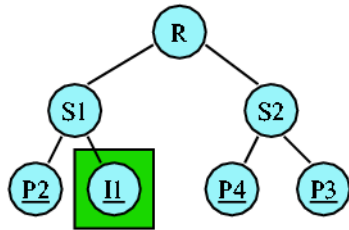


Merged List

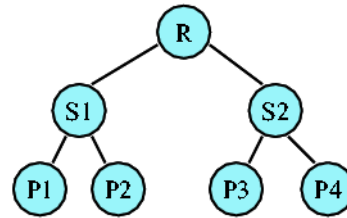


Recurse for I1

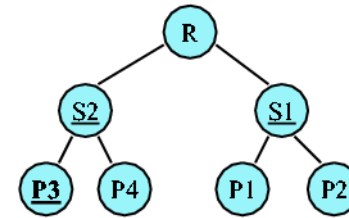
# Example



Branch A

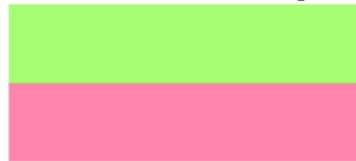


Base

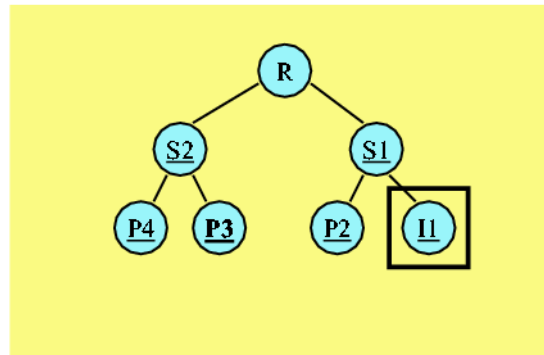


Branch B

Child lists to merge



Merged List



No cursor!

DONE!

# Summary

- Chapter 4, 5, 3, (6, 7, 8) (in that order).
- Tree merging has two phases: Matching and Merging
- Merge decisions can be made on behalf of a edit script or deduced from a matching
- 3DM implements tree 3-way merging without edit history.

# Questions

- The author claims that the use of a DTD would make his tool more powerfull. In what ways could the DTD assist the tool?
- Is there a cleaner solution then the 128 byte purging of matching to aviod matching (small) unrelated data?
- Would it have helped the author to work with DAGs insted of trees?