This merges data from the b tree into the a tree; that is, it transforms the a tree so that it accurately represents both the input used to construct a and the input used to construct b.

merge\_tree(a,b):

(This is only called on nodes that are either both roots of their respective trees,

or that represent the same utterance or action. It modifies a and its descendants.)

if neither a nor b have any children, return

if a and b each either have exactly one child, or are marked that their children are alternates then:

for each child of b:

for each child of a:

if the two children represent the same utterance or atomic action

**and** the multiset of all leaf nodes descended from them are the same then:

call merge\_tree() on those two children and break the inner loop

if no matching child of a was found:

add the child of b as an (alternate) child of a

else if a and b have the same number of children then:

if each child of a represents the same utterance or atomic action as the child of b at the

corresponding position in b’s child list:

call merge\_tree() on each matching pair

return

merge\_children(a, b, a.children, b.children)

merge\_children(a, b, alist, blist):

*(Note: this assumes that the multiset of leaf nodes descended from nodes in alist equals the multiset of*

*leaf nodes descended from nodes in blist)*

make a mapping from the nodes in alist to the nodes in blist

such that, if one node is mapped to another node, they represent the same utterance or atomic action

update\_children\_relationships(a, b, mapping)

for each node in alist that was mapped to a node in blist, call merge\_tree() on those two nodes

make lists of all nodes that were not mapped

while there are still unmapped nodes remaining:

let maxnode be the non-leaf unmapped node that has the most leaves descended from it

let other be the root of the tree maxnode is **not** in (either a or b)

using dynamic programming (as described below), find a set of unmapped nodes such that

the multiset of all leaf nodes descended from the nodes in the set is equal to the multiset of all leaves descended from maxnode, and all the nodes in the set descended from other

(call this mappedset)

if there is no such set, error

merge\_children(maxnode, other, maxnode.children, mappedset)

insert maxnode as a child of other immediately before the first node in mappedset

for each child of other:

if it is parallel to some node in mappedset,

then mark it parallel to maxnode

else if it comes before the first node in mappedset but there is

a corresponding node in the other tree that comes after maxnode (or vice versa):

then mark it parallel to maxnode

else if it comes before the first node in mappedset,

then mark it as coming before maxnode

else if it comes after the first node in mappedset,

then mark it as coming after maxnode

else if it comes before some nodes in mappedset and after others,

then mark it parallel to maxnode

remove all nodes in mappedset from other

The dynamic programming algorithm mentioned above:

Let M[i] be the smallest set obtainable by subtracting the multisets of leaves descended from some of the first i children of other, and let S[i] be the set of children of other whose leaves were subtracted from the leaves of maxnode to obtain M[i]

M[0] = multiset of all leaves descended from maxnode

S[0] = {}

For i from 1 to n, where n is the number of children other has:

M[i] = the smallest set out of

{M[i-1], (M[j] minus all nodes descended from the jth child of other) for j from 1 to i }

S[i] = S[i-1] if M[i] is M[i-1], otherwise S[j] + the jth child of other for whatever value of j was used to determine M[i]

If M[i] is empty, terminate; S[i] is the set whose leaves are equal to those of maxnode.