This is the central method of maketree.py. It takes in a list of TMRs and atomic actions

and attempts to place them into a hierarchical tree structure.

The function about\_part\_of(a,b) takes two TMRs and determines whether the second one has a THEME that HAS-OBJECT-AS-PART the THEME of the first one (that is, the first’s TMR is about an object that is part of the theme of the second’s TMR).

function construct\_tree:

make an empty TreeNode (that is, without a TMR or a name) to be the root

current\_parent = root (New nodes for non-atomic actions will be added  
 as children of whatever current\_parent is)

for each input (action or utterance) in the list of inputs:

if this input is an utterance, then

if there are no events in its tmr **then** continue

**else** if its tmr is in past tense: (that means its children came before it)

if the theme of the current parent’s tmr is part of the new tmr **then**

go up the tree from current\_parent until an ancestor is found for which

this is not true, and set that node as current\_parent

make a new TreeNode from the new tmr

set a pivot j such that, for each child *c* of current\_parent **before** the jth,

the theme of c’s tmr is not part of the theme of the new tmr

set the children of current\_parent before the jth

as disputed between current\_parent and the new

node (that is, they might be children of either one,

and it is not known which)

put the children of current\_parent after the jth as

children of the new node, and not current\_parent

set the new node as a child of current\_parent

else if the previous input was an atomic action

(in the case where the about\_part\_of relation is false)

("previous node" refers to the most-recently-added child of current\_parent)

if that action currently is a child of a nameless node:

(that is, a non-primitive node without a tmr assigned)

let this tmr be the nameless node’s tmr

else if this new tmr’s theme is a part of the theme of the previous node’s tmr:

current parent = previous node

insert a node with the new tmr between

current\_parent and its children

(that is, as a child of current\_parent and as the

parent of all of its children)

else:

mark all children of previous node

(which should all be atomic actions) as disputed

(i.e. they might belong to previous node,

but they might belong to this node)

else: (the part\_of relation is false and the previous input was not an atomic action):

this situation hasn't come up yet,

but more heuristics will be added here when it does

else: (the input's children come after it)

~~look up the tree to see if there is an ancestor of current\_parent such that   
 the new tmr’s theme is part of the ancestor’s tmr’s theme~~

~~if so, set that ancestor as current\_parent~~

while the current parent’s tmr’s theme is about part of the new tmr’s theme

or the current parent’s tmr has the same main event as the new tmr,

set the current parent’s parent to be the current parent

make a new TreeNode with this tmr and set it as a child of current\_parent

if the next input is not an atomic action, set this new node as current\_parent,

so that the node for the next utterance will be added under the new node

else: (the input is an atomic action rather than an utterance)

if the last child of current\_parent already has children of its own:

make a child of current\_parent with no tmr (to be filled in with a tmr later)

set this action (and all consecutive atomic actions until the next utterance)

as children of the last child of current\_parent

disambiguate(root)

serialize the current state of the tree to be sent to the front-end

return root