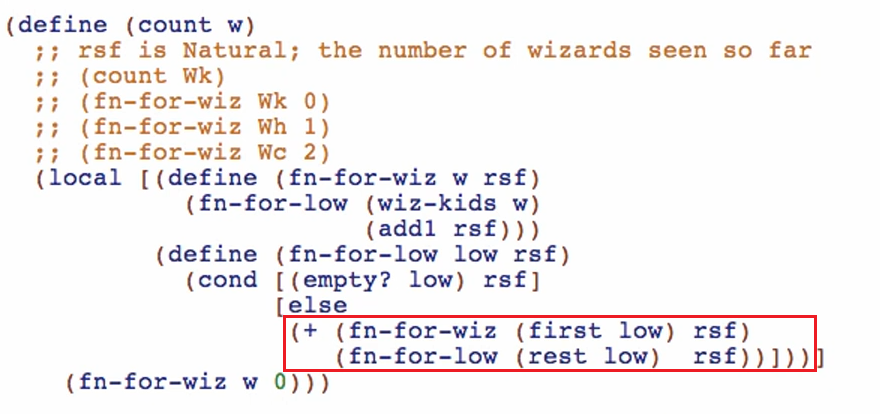
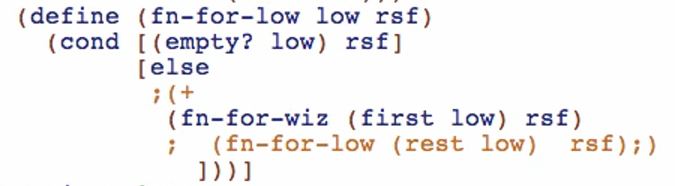
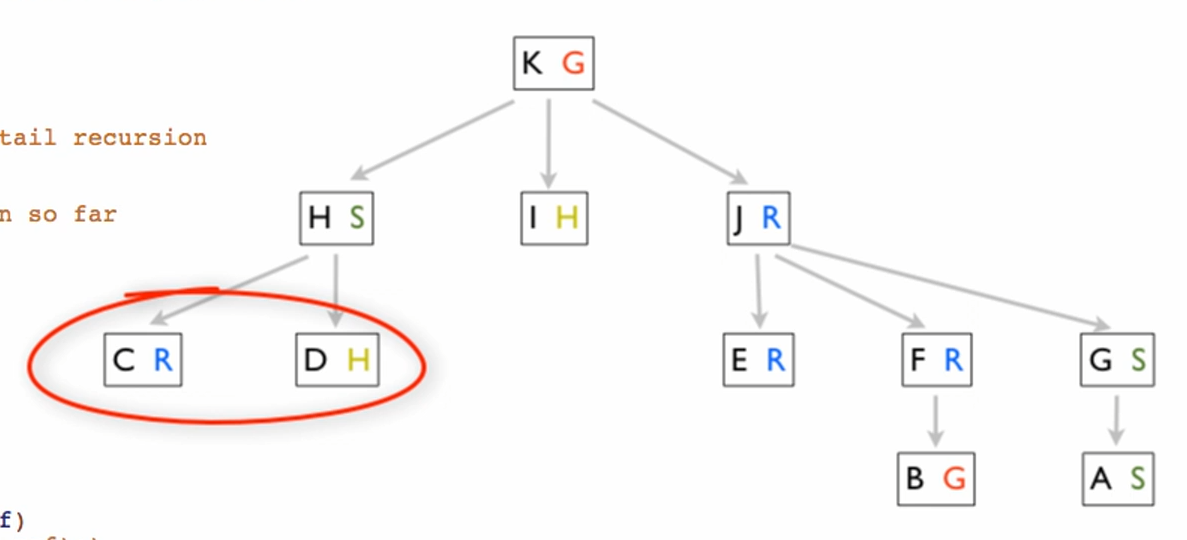
Fixing this first. This should be in tail position



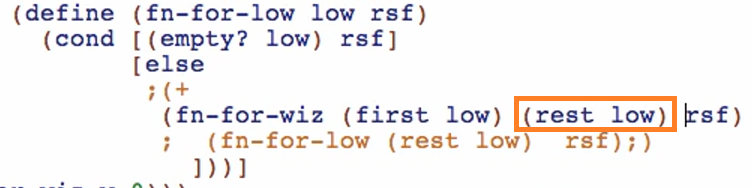
We must get rid of the primitive + and the natural recursion. We should only call the mutual recursion as our recursion in TAIL position



But doing this, we’ve LOST the NATURAL RECURSION but somehow, we want to call fn-for-wiz in tail position.

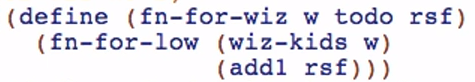


We need a new accumulator for this!



We need to preserve the rest of low for the fn-for-wiz to work on, since we are not using natural recursion anymore!

Going back to fn-for-wiz (adding the new accumulator)



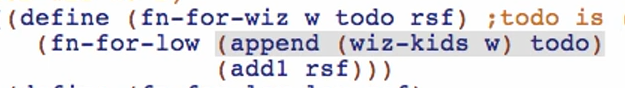


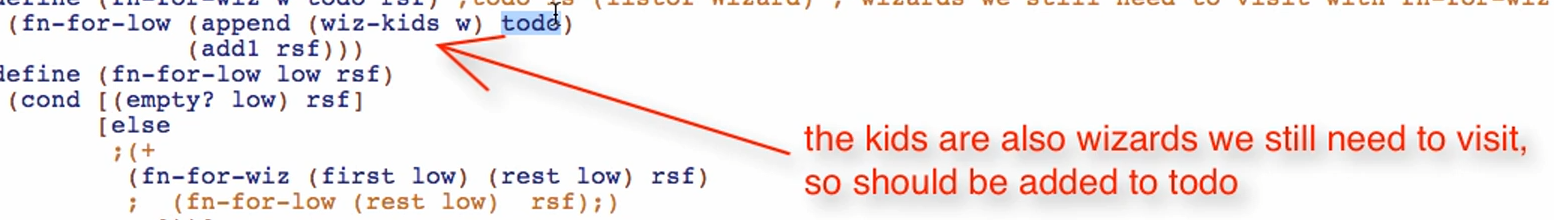
Notice the relationship:

todo: we haven’t visited yet

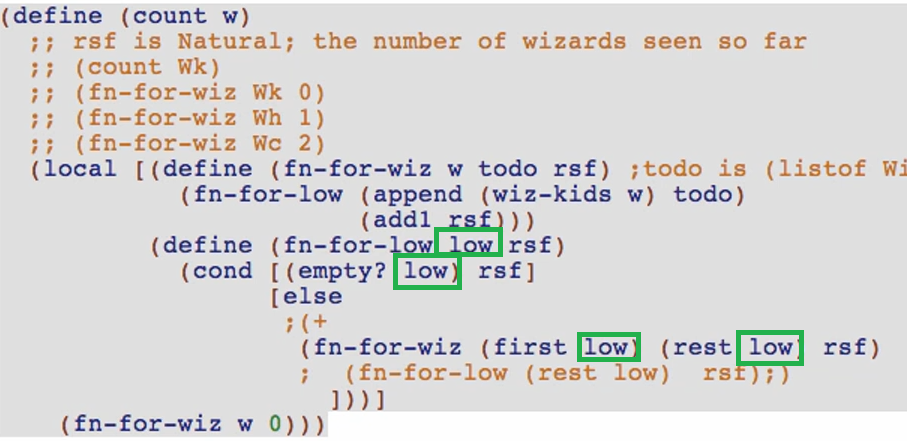
rsf: we have visited already

Add todo (rest of the listof Wizard)

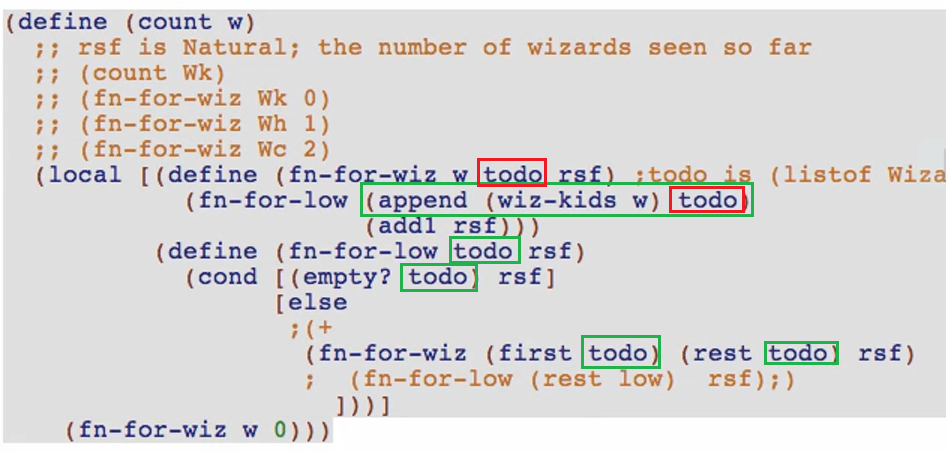




Rename low to todo in fn-for-low



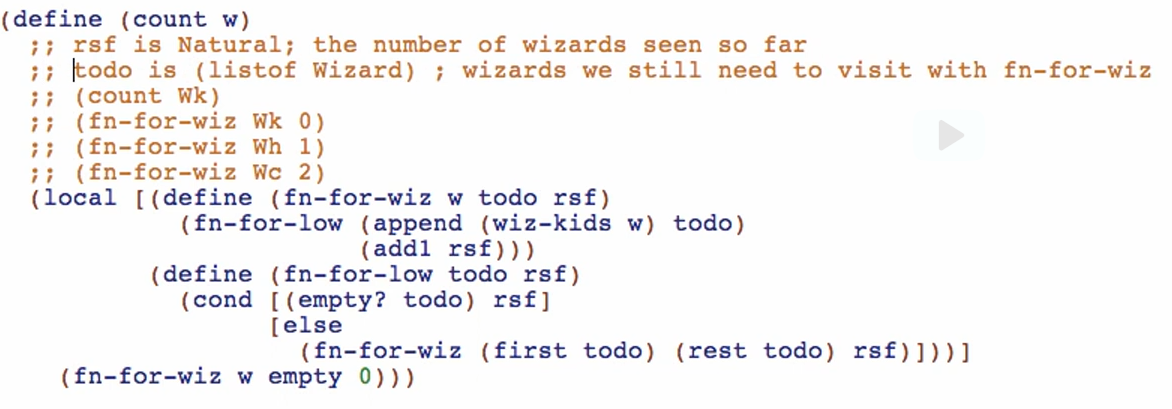
But take note that this is not the “todo” parameter in fn-for-wiz



Add also the extra argument in our initial value (no pending children yet)



Clean your function (remove unnecessary comments)

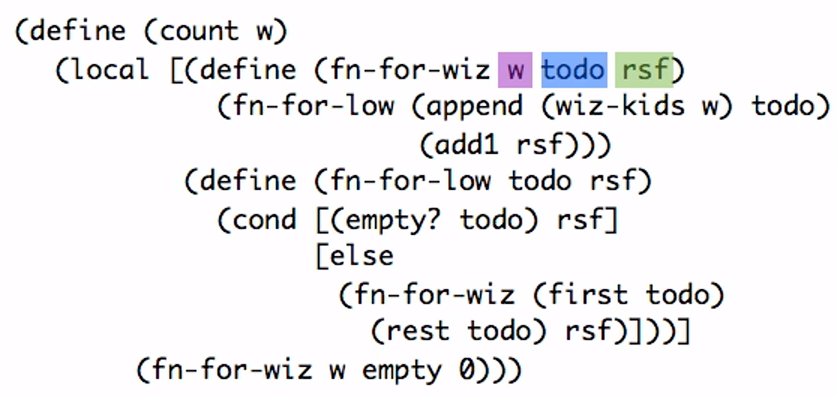


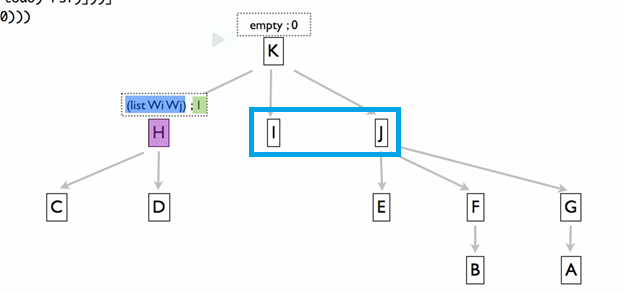
Run and debug!

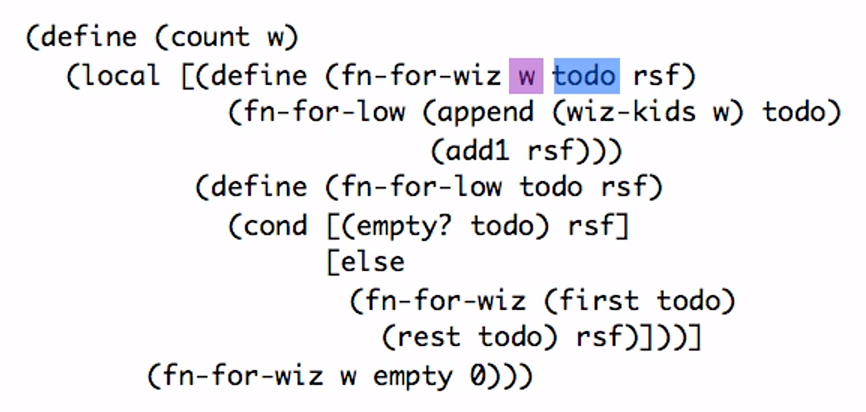
todo is a good example of a WORKLIST accumulator

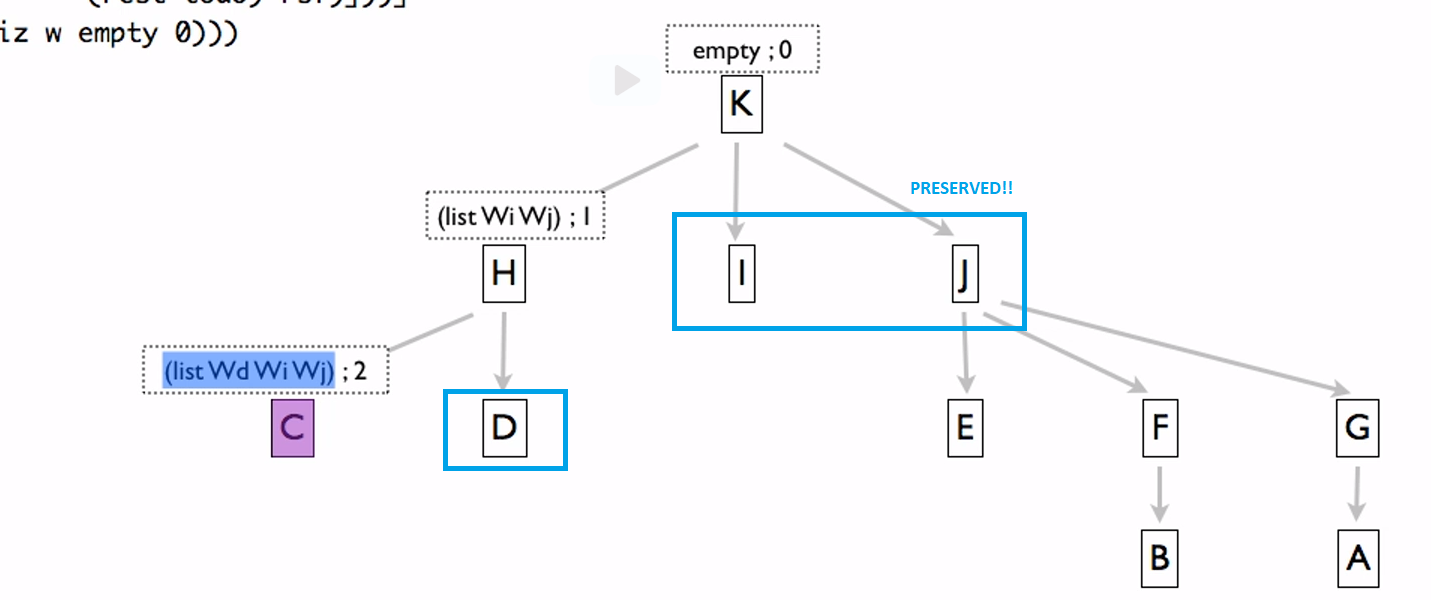
* It preserves what elements to be work on next

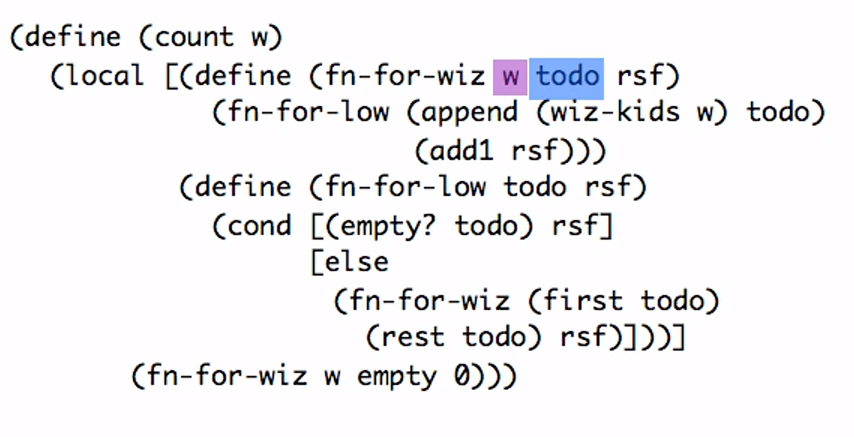
Step by step checking of values:

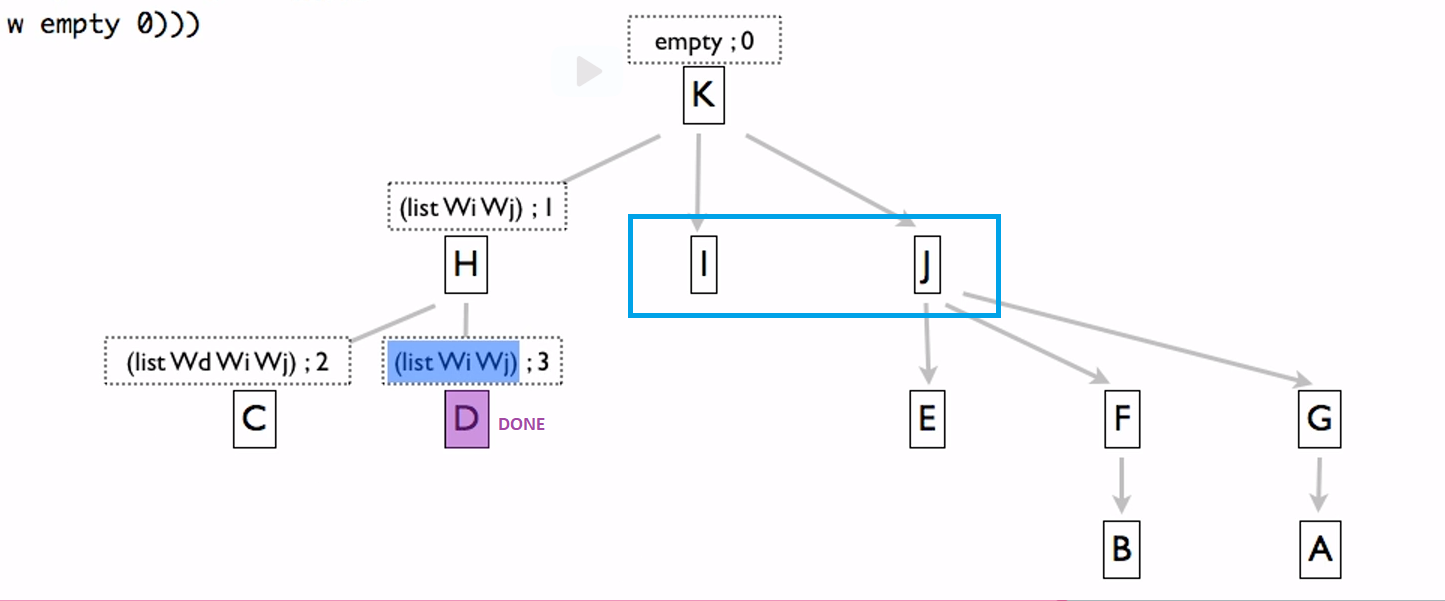




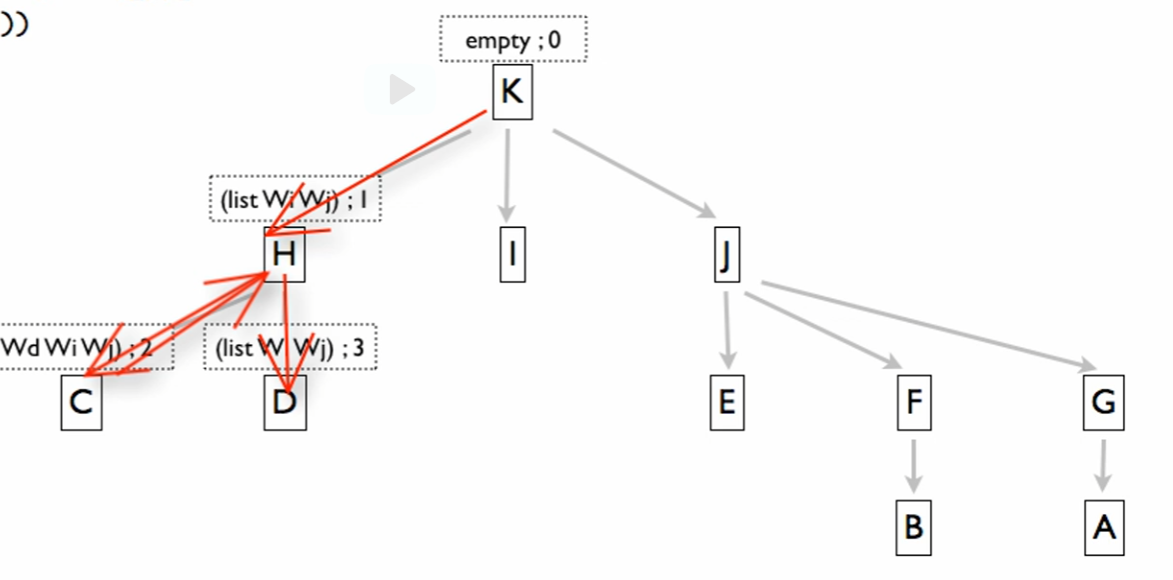






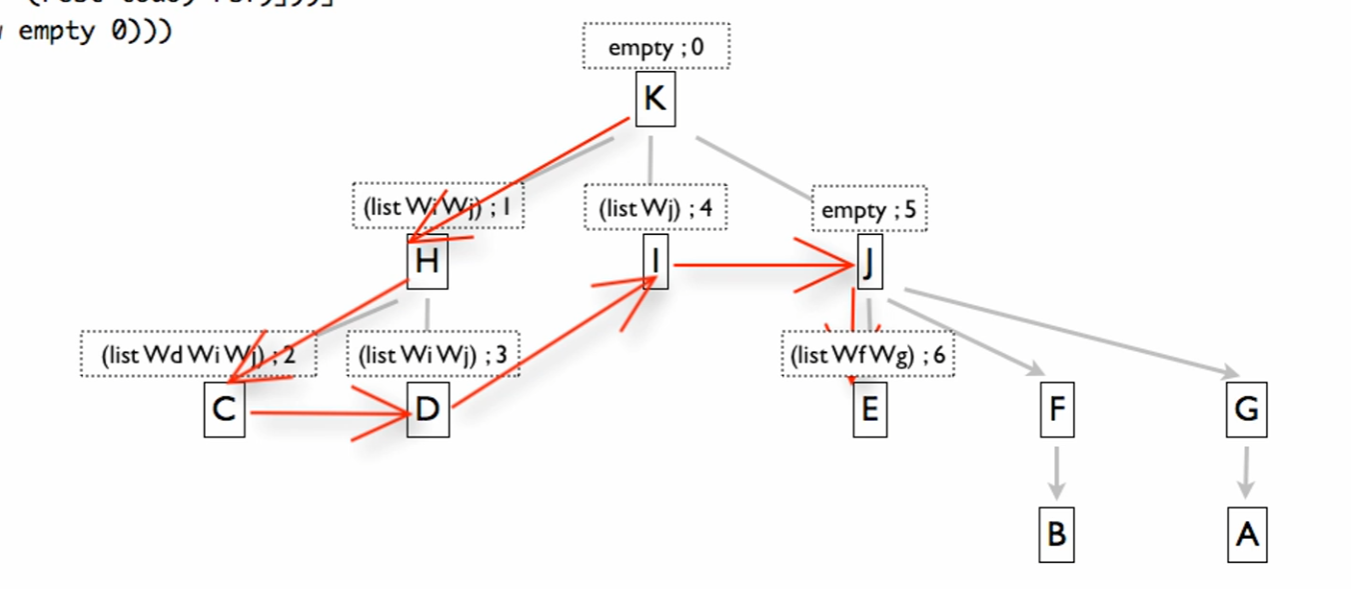


Not tail recursive:



Goes back to the previous element. This is the reason why we came up with count 30 earlier since there are elements that we are counting twice because of this backing up.

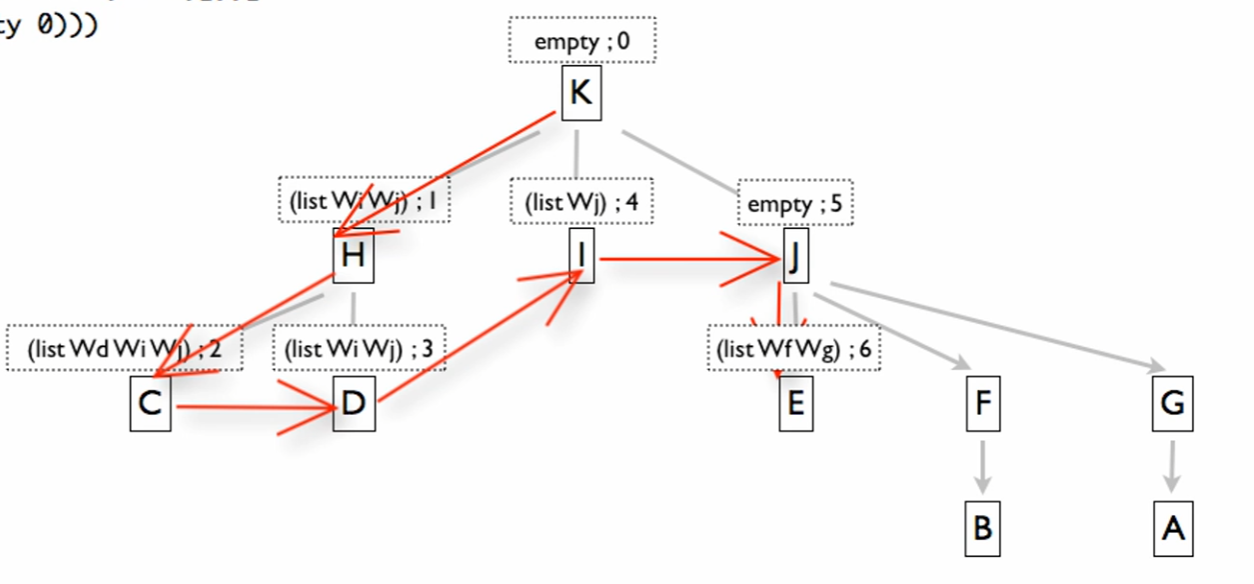
In a tail recursive traversal in an arbitrary arity tree, it does not backup to the previous element!



It backs up to our worklist accumulator (the list where we have not visited yet!)

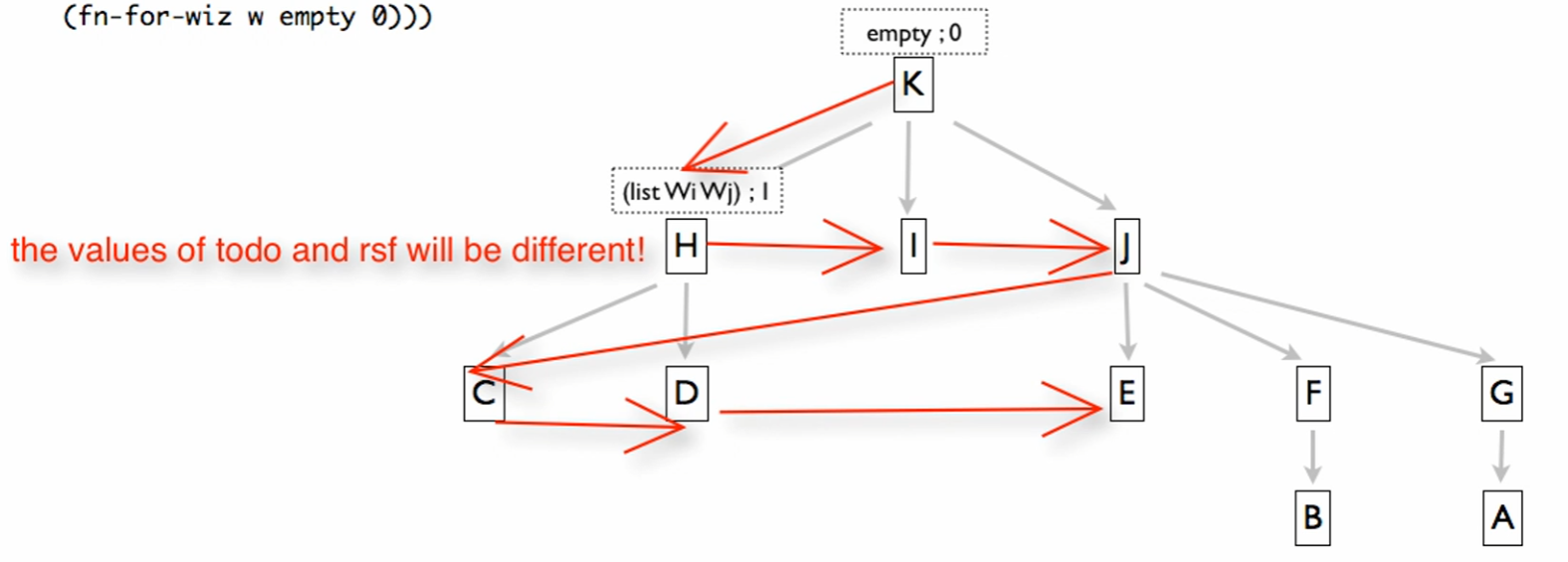
Order of appending affects the traversal of our tail recursive function





This order produce BREADTH-first traversal





But it has the same result 😊