Algo for detecting cycles resulting from ADD move feature

Compared with the other ADD operations (e.g. ADD add feature, remove feature, etc.), ADD move feature requires extensive verification. Following is a description of an algorithm intended to ensure that adding a `move feature` operation results in a sound plan. For simplicity, we abstract away from groups and features and view the combination of the two as nodes.

Let n be the node to be moved and c_1 the target node, i.e. n's new parent node. Furthermore, let t_1 be the time point at which this operation is inserted, and t_e the time point where n is moved next or removed, or ∞. Assume we have a function `ancestors(node, time)` which takes a node and a time point and returns a list of `node`'s ancestors at time point `time`.

First, check whether $n \in \texttt{ancestors}(c_1, t_1)$. If this is the case, report that the move causes a cycle and terminate.

Next, find a list of critical nodes. Let $A_n = \texttt{ancestors}(n, t_1) = [a_1, a_2, \dots, SN, \dots, r]$ and $A_{c_1} = \texttt{ancestors}(c_1, t_1) = [c_2, c_3, \dots, c_n, SN, \dots, r]$ with SN the first common ancestor of n and c_1. The list of critical nodes is then $C = [c_1, c_2, \dots, c_n]$, which is essentially the list of n's new ancestors after the move.

Repeat this step until the algorithm terminates:

Look for the first move of one of the critical nodes. If no such moves occur until t_e, the operation causes no paradoxes, and the algorithm terminates successfully.
Suppose there is a `move` operation scheduled for t_k, with $t_1 < t_k < t_e$, where c_i is moved to k. There are two possibilities:

1. k is in n's subtree, which is equivalent to $n \in \texttt{ancestors}(k, t_k)$. Report that the move caused a cycle and terminate.
2. k is not in n's subtree, so this move is safe. Let $A_k = \texttt{ancestors}(k, t_k) = [k_1, k_2, \dots, k_n, SN', \dots, r]$, with SN' the first common element of A_k and A_n. Update the list of critical nodes to $[c_1, \dots, c_i, k_1, \dots, k_n]$.

Soundness and completeness for the algorithm

In this context, soundness means that a `move` operation which is accepted causes no paradox. Completeness mans that every move which causes no paradox is accepted.

Soundness

We argue that if no cycles are detected by this algorithm, no cycle occurs.

A *critical node* is a node whose planned move may cause a cycle in the feature model at a future point. From the assumption that the initial plan is sound, critical nodes can be limited to nodes that were allowed to move to n's subtree, but become the ancestors of n after the `move` operation is executed.

Reasoning behind critical nodes: We could have looked at all of n's ancestors instead of choosing just a few. The reason we don't is that no moves already in the plan could have moved an ancestor of n into n's subtree, since this would have caused a cycle. Therefore we only look at the nodes that were previously allowed to do such moves, but are not anymore. Futhermore, we only look at the interval $[t_1, t_e]$, since removing or moving n reverses the effects of this move.