The “input.text” contains 6 Logged Java classes, where the first 4 logged Java classes are the same and the only difference is in the location of where logging calls are located, but the last 2 ones are different.

The “output.text” is the output file created by our tool after applying the clustering algorithm described in the *clustering algorithm* Section. The Figure 1 represents the hierarchy of clusters. The “output.text” lists constructed clusters, their AUAST node, their children, and the similarity between their children if there is any (each cluster can have 2 children at maximum) (clusters with no children contains AUAST leaves and clusters with 2 children contains anti-unified AUAST).

When the clustering is performed it first merges the first fourth Java classes (since they are the most similar ones) and then it merged the clusters 5 and 6. As depicted in Figure 2, logging calls are located inside a catch block in the AUAST 1 and 2, while logging calls are located in an if statement in AUAST 3 and at the beginning of a method in AUAST 4. First, Logging calls in AUAST 1 and 2 are anti-unified with each other (most similar ones) (AUAST 7). Second, logging calls in AUAST 3 and 4 are anti-unified with each other by inserting a NIL structure (AUAST 8). However, in the later step of clustering for creating an anti-unifiers from the AUAST 7 and 8 two logging calls cannot be anti-unified with each other since the structures containing them does not correspond to each other! I suggest to discard the anti-unifier when anti-unification of AUASTs does not allow the anti-unification of logging calls since our purpose is to anti-unify logging calls and while the structures containing them do not corresponded it is not possible (due to our constraints), no matter how similar is the code of logged Java classes. I think this way we might not even need to take heuristics (I am not sure about this yet!). Following my suggested solution we will come up with the anti-unified structures in Figure 3, instead.

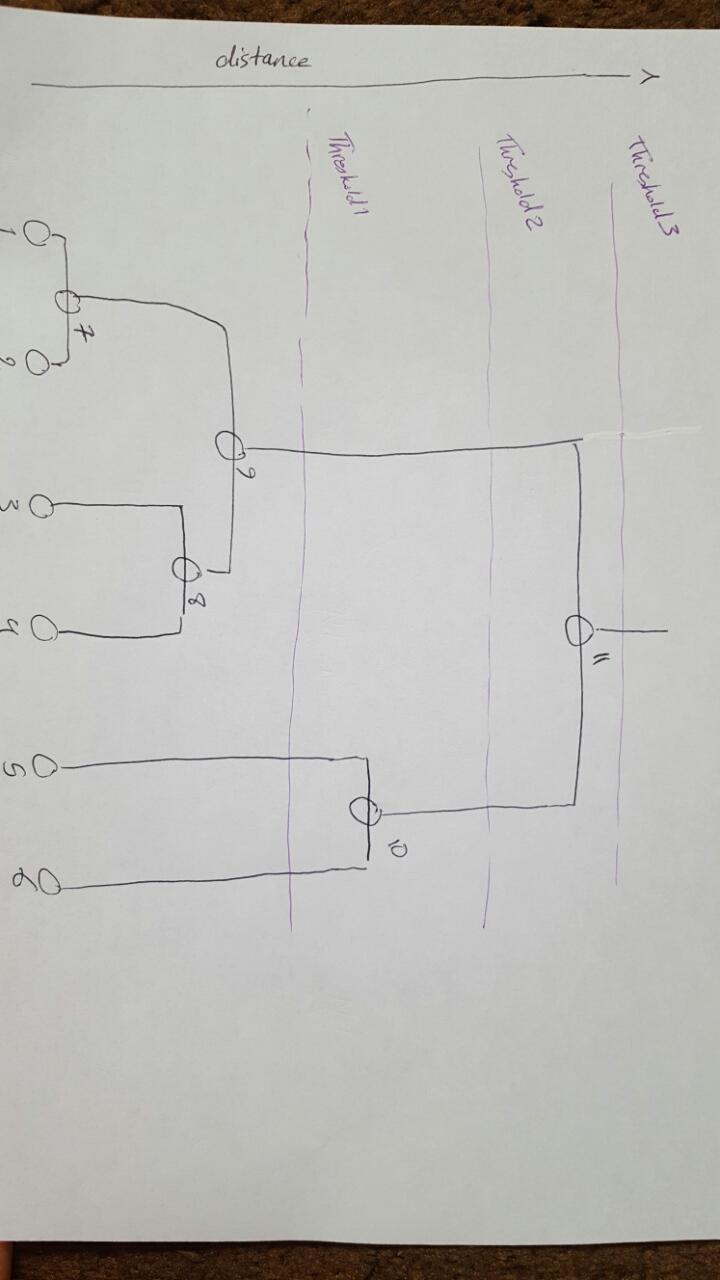


Figure 1. The hierarchy of clusters after applying the clustering algorithm on “input.text”. The threshold value indicates the number of clusters we will come up with.

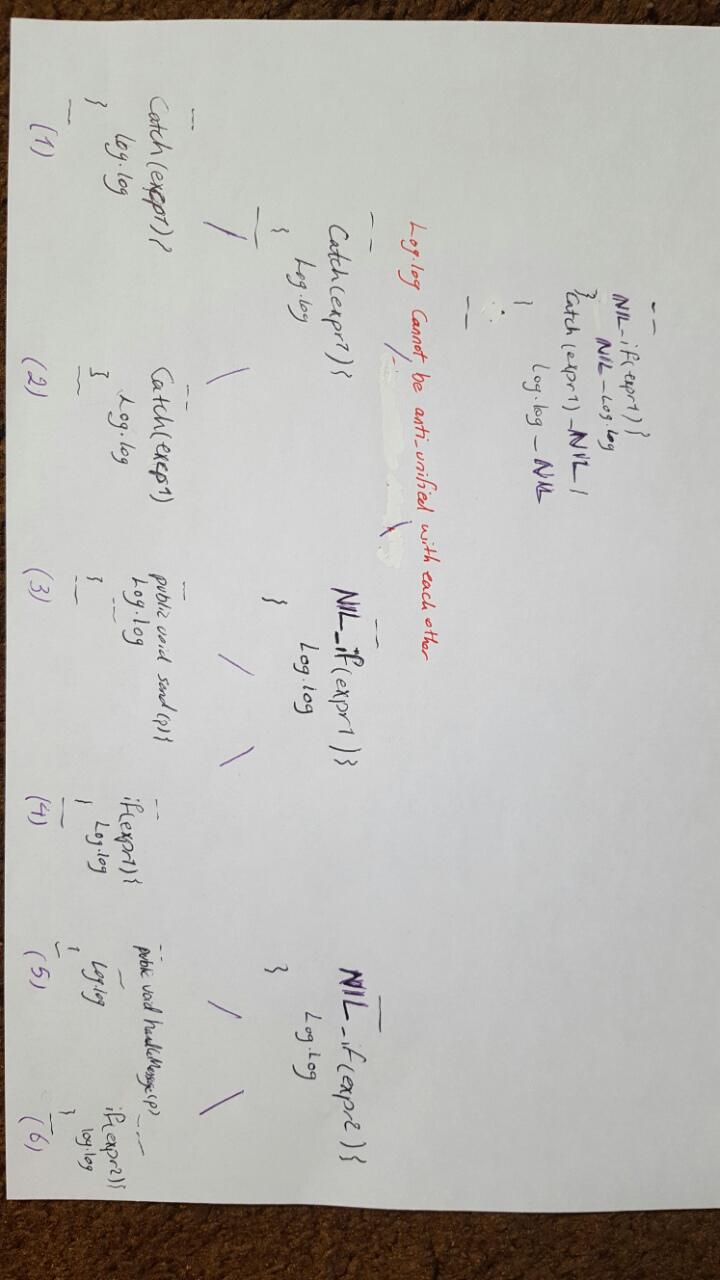


Figure 2. Applying anti-unification

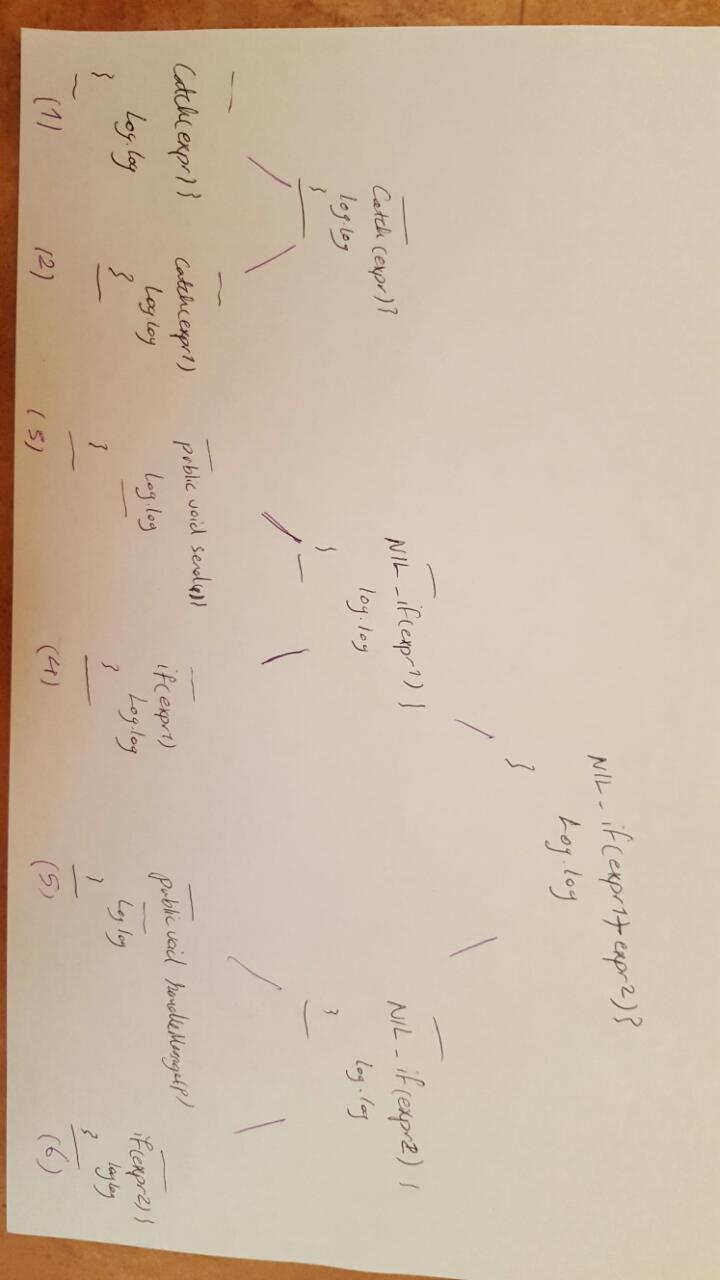


Figure 3. Applying anti-unification (My suggested solution)

## *Clustering Algorithm*

We have developed a modified version of a hierarchical agglomerative clustering algorithm (illustrated in Figure 4) as described below:

1. Start with singleton clusters, where each cluster contains one AUAST
2. Compute the similarity between clusters in a pairwise manner
3. Find the closest clusters (a pair of clusters with maximum similarity)
4. Merge the closest cluster pair and replace them with a new cluster containing anti-unifier of AUASTs of the two clusters
5. Compute the similarity between the new cluster and all remaining clusters

* Repeat Steps 3,4, and 5 until the similarity between closest clusters becomes below a predetermined threshold value
* The similarity between a pair of clusters is defined as the similarity between their AUASTs
* Determine the similarity threshold value through informal experimentation

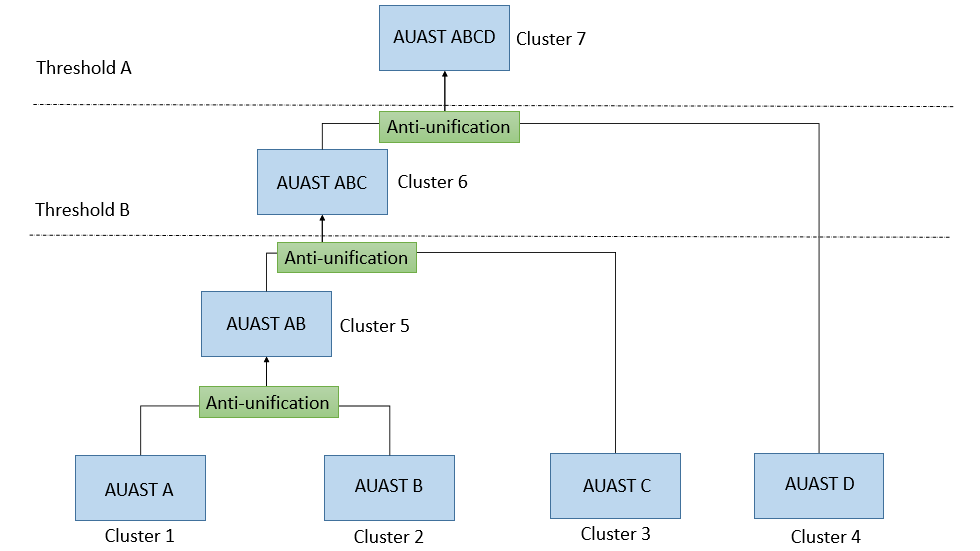


Figure 4. Anti-unification of 4 AUAST nodes using an agglomerative hierarchical clustering algorithm. The threshold value indicates the number of clusters we will come up with