1. Output presence conditions
   1. Modify Execute() in TransformationController to retrieve and print out the output presence conditions to a csv file
   2. Modify the GM Transformation so that output root element gets the name “rootElem” and all SwCompToEcuMapping\_components in the apply model get their names from the Distributable object in the match model
2. Lift existential matching
   1. Modify TransformationFilter by adding lists to keep track of the ids of classes that are either existentially matched or backward links (both of which differ across existential match sites)
      1. Modify the constructor to keep track of these ids as they are added
      2. Add getter methods for the lists of ids
   2. Modify performMatch() in TransformationRule to modify the presence conditions based on the algorithm for existential matching
      1. Cluster the solutions together based on whether the object ids that differ between them are exclusively existentially matched objects or those resulting from backward links 🡪 if they are, then those solutions belong to the same existential match site cluster
      2. For each solution in a cluster, its output presence condition is modified so that the negated output presence conditions of all the solutions processed before it are included; this precludes any two solutions in the same cluster from co-existing
   3. Fix a bug in buildJoinPredicates1() and buildJoinPredicates2() to allow rules in which the match part has no universal component (only existential)
3. Simplify presence conditions
   1. Add a PresCondNode class in the *lifting* package that does all of the optimizations for a raw output presence conditions; PresCondNode does five simplifications:
      1. removal of double negation
      2. flattening from binary to N-ary
      3. removal of duplicate clauses
      4. de Morgan’s rule
      5. removing any extra nodes
   2. Modify performMatch() in TransformationRule so that the raw output presence conditions are simplified before they are saved (this reduced the number of clauses / clause-to-variable ratio by 78.6%)
4. Automated testing
   1. Add a genericTest() method to Tester that is used by a test case to run the transformation with a specified project directory and transformation file and then validate the results
      1. For output to be correct, there must be the same set of elements in the expected output and the actual output, though the order of elements does not matter
      2. For corresponding actual and expected output elements, the presence conditions must also be equivalent (i.e. *(and f1 (not f2))* & *(and f2 (not f1))* should both be UNSAT) – this is checked using the Microsoft Z3 solver
   2. Add a test suite called ComplexExampleTest to the *lifting* package that tests all transformations for the “complex example”
   3. Add a test suite called SimpleExampleTest to the *lifting* package that tests all transformations for the “simple example”
   4. Add a test suite called ContainedExampleTest to the *lifting* package that tests all transformations for the “contained example”
5. Plugin Wizard
   1. Created a plugin wizard that could be used to select the project directory and transformation file for a DSLTrans transformation (which worked perfectly as a standalone plugin)
   2. Attempts to integrate it as an extension of the current DSLTrans\_lifting project failed for unknown reasons; a two-hour debugging session with Cláudio Gomes over Skype was also unsuccessful
   3. Cláudio Gomes’ advice was to make the wizard an entirely separate project for which the DSLTrans\_lifting project is a dependency; this could then only be done after the DSLTrans\_lifting project is released and can be installed as a plugin
   4. As a compromise, a GUI that can be used to select the project directory and transformation file has been created in FileChooser
6. Lifting NAC’s
   1. Add five instance variables (and getter and setter methods where necessary) to MatchFilter to keep track of:
      1. the ids of negative match classes
      2. the ids of the objects matched to those classes
      3. the negative component of the Prolog query
      4. the negative component of each solution that is found
      5. a switch to treat the negative match classes and associations as positive
   2. modify the MatchFilter constructor to keep track of the ids of the negative match classes and associations as they are created
   3. modify process() in MatchFilter
      1. remove the negatively matched objects and associations in a solution, join the presence conditions of the removed objects, and keep track of which solution this negative clause belongs to
      2. remove the backward links in a solution as well, because they make solutions that would otherwise be identical differ
      3. when there are multiple sets of negatively matched objects / associations affecting the same match site, join these negative clauses together accordingly 🡪 one negative clause per solution
      4. after running regular query, build and execute it again by treating negative matchers as positive; this is to find solutions that only emerge due to lifting (they emerge because the objects that would have prevented the transformation from occurring are now a part of the solution itself)
   4. modify buildJoinPredicates1() and buildJoinPredicates2() so that the second time the query is run, the negative matchers are all treated as positive; generateDifferentEntitiesFactCall() is also done before getNegativeClauses() because the second method now calls the former
   5. modify generateDifferentEntitiesFactCall() in MatchFilter so that an object cannot be matched to multiple match classes, regardless of whether those classes are positive or negative (this was an error from before lifting NACs was introduced)
   6. modify getNegativeClauses() in MatchFilter so that all of the negative match classes must be matched at the same time (originally, in regular DSLTrans, a bug allowed a single object to satisfy multiple negative match classes); use the result from the modified generateDifferentEntitiesFactCall() to ensure that the objects matched to the different classes are distinct
   7. Modify performMatch() in TransformationRule to retrieve and join the negative clause of each solution presence condition with the rest of the expression
7. Lifting Indirect Links
   1. Add two hashtables to MatchFilter to keep track of the ids of the source and target match classes for both positive and negative indirect links
   2. Modify MatchFilter constructor so that when a new indirect link is added, the source and target match class ids are added to the appropriate hash table
      1. the target is the key and the source is the value because the in-between objects / associations are found by starting at the target and working your way up to the source
      2. this is the most efficient traversal, since an object can only be the target of one containment association, even though it can be the source of many containment associations
   3. modify performMatch in TransformationRule to support positive indirect links: for a given match site, every element in the match site that is a target of some positive indirect link has its presence conditions conjoined to the presence conditions of all the in-between elements in its corresponding positive indirect link
      1. this is done by iterating over all the associations in the match model and trying to find a containment association where the target object is also the matched target of the indirect link
      2. the source of that containment association becomes the new target, its presence condition is conjoined, and the process is repeated until the target is finally the same as the source object of the indirect link
   4. modify process in MatchFilter to support negative indirect links: for a given match site, every element in the match site that is a target of some negative indirect link has its presence conditions conjoined to the presence conditions of all the in-between elements in its corresponding negative indirect link
      1. this is done in the same manner as for positive indirect links in TransformationRule
      2. this is only done for solutions that arise as a result of treating negative matchers as positive (i.e. in cases where, if it were not for lifting, the transformation would not occur)
      3. this is separate from positive indirect links because negatively matched objects are removed from the solution before the solution is added to the list of results; they do not persist to the point where they can be analyzed in TransfromationRule