To answer the question, let’s look at an overview of our problem and how we can utilize

higher-order anti-unification modulo theories to help solve the problem of small-scale reuse. For example,

To construct a structural generalization form ASTs of Java classes that use logging calls as described above, we developed a prototype tool called~\ref{tool} , which applies Jigsaw framework to find candidate structural correspondence between two ASTs, and uses higher-order anti-unification modulo theories to construct a generalized structure.

It takes the source code of two logged Java classes as input and performs a sequence of actions on them, outlined by the algorithm \func{generalize}: (1) input into the algorithm are the ASTs of two input Java classes, created via Eclipse JDT framwork; (2) generating augmented form of each AST, the CAST, using Jigsaw framework, in which each node holds a list of candidate correspondence between two source codes (Line 1)(see Section 4.2); (3) Construct AUAST nodes from CASTs by extracting structural properties and adding constraint logging properties (Line 2) (see Section 4.3); (4) (Line 3) (see Section 4.4); and (5) transformation of the selected anti-unifier

(Lines 4 and 6 ) to propose an integration solution for the developer’s system (see Section 4.5).

Developing a measure of similarity to indicate structural correspondence between input classes with special attention to logging calls(Line~?)(see Section~\ref{});

Developing a measure of similarity to indicate structural correspondence between input classes with special attention to logging calls(Line~?)(see Section~\ref{});