CSE 403: Project Pitch
INDEX CHECKER FOR CHANGING STRUCTURES

Jake Chiang (jchiang2)
Rodney Olson (rolsonjr)

Index checkers are a subset of checker frameworks that statically analyze programs in order to determine potential defects. In particular, index checkers identify unsafe code that may manipulate data collections in unintended ways, such as accessing out of bounds indices. Modern index checkers, such as Java's Checker Framework, are restricted to only operate on fixed length data structures like arrays, strings, and immutable lists. Knowing a data structure to be of fixed size allows certain guarantees to be known at compile time, thus allowing the analysis to take place.

Mutable data structures, however, are not handled by index checkers, as it is impossible to fully understand what indices may be valid for a structure that can have elements added or removed freely during execution. A naïve solution is to treat mutable types as immutable up until a modifying operation (such as adding or removing) is executed, at which point the index checker is invalidated from making any further checks. This approach is unlikely to produce comprehensive results however, as the simple presence of mutable structures means that addition or removal operations will likely be very prevalent.

We propose a modified approach that would allow for index checking on mutable data structures in limited situations by making guarantees based on conditional information gained from surrounding context. Using size guarantees provided by conditional statements (such as if, while, for, etc.), this information could be incorporated into the structure's type information used by the checker for use in the scope of the conditional. With the structure bounds known, conventional index checking can then be performed. Furthermore, additions or removals to the data structure in the same context can be accounted for and used to update the structure type information.

Schedule oriented challenges in implementing this extension fall primarily around incorporating it into the existing Java Checker Framework. Understanding how to formulate the problem within a reasonable scope and then develop it in the existing environment will likely contribute a large portion of development time. This can hopefully be mitigated by a thorough testing of the basic initial tool cases before expanding the scope of the project.