Data structures play a central role in software development. While developing proper data structures is not easy, reasoning about them is even harder. The difficulty comes from their typical characteristic: the unboundedness of the data structures and/or the loops manipulating them. This makes two following fundamental issues more severe: interdependence between structures and data values, and complicated interactions between different data structures.

In this thesis, we consider the problem of reasoning about unbounded data structures such as strings, linked lists, trees. Specifically, we propose systematic techniques for the satisfiability problem of string constraints, and the entailment proving problem for heap-allocated data structures. These two problems are of great interest: for example, while the former is important for security analysis of web applications, the latter is important for automated verification of imperative programs.

The first technique is to implement lazy reasoning methodology. Its introduction is to mitigate the problem of combinatorial explosion in searching for a solution of the input constraints. Specifically, we incrementally reduce recursive predicates, which are used to represent string operations, via splitting (and/or unfolding) process, until their subparts are bounded with constant strings/characters to be consumed. We have applied this technique in building an efficient string solver. While modern string solvers exist, they suffer in one way or another: (1) the constraint language may not be expressive enough (even though the solver is fast); or (2) the solver may not be fast enough to accommodate realistically large programs. Thanks to lazy reasoning, we now have a fast symbolic string solver to support an expressive language, thus opening doors for future development of comprehensive frameworks for vulnerability detection in web applications.

Since lazy reasoning does not address non-termination in solving string constraints, we next propose a novel method, namely, progressive reasoning. The key feature of the new
algorithm is a pruning method on the subproblems, in a way that is directed. More specifically, our algorithm detects non-progressive scenarios with respect to a criterion of minimizing the "lexicographical length" of the returned solution, if a solution in fact exists. Informally, in the search process based on reduction rules, we can soundly prune a subproblem when the answer we seek can be found more efficiently elsewhere. If a subproblem is deemed non-progressive, it means if the original input formula is satisfiable, then another satisfiable solution of shorter "length" will be found. If, on the other hand, the input formula is unsatisfiable, then any pruning is obviously sound. A technical challenge we will overcome is that at the point of pruning, the satisfiability of the input formula is unknown. Experimental evaluations show the promising results of our new string solver in dealing with non-termination in string solving.

Finally, we propose a general method that includes inductive reasoning for entailment proving. It aims to address non-termination in proving heap-allocated data structure properties. The challenge is how to use induction correctly and avoid erroneous proof arising from a form of circular reasoning. Our method is able to use dynamically generated formulas as induction hypotheses, and to enforce an anti-circular condition so that any application of an induction step is guaranteed to be correct. The state-of-the-art methods are often unable to prove relationship between different data structures (e.g. to prove that a sorted list is a list). As a result, they would not be able to automatically verify a large class of programs. Inductive reasoning helps us to close such remaining gap in existing systems. More importantly, it also gets us back the power of compositional reasoning in dealing with user-defined recursive predicates that are used to represent data structures properties.