Chapter 5 Discussion and Conclusion

5.1 Discussion

From the best of our knowledge, Java model extractor Pathfinder has not supported abstract data types from Java standard library. In other words, it assumes the behavior of a Java thread does not involve ADTs. Bandera can process code which uses *vector*. However, *vector* is just another safe-to-use array. Supporting ADTs requires a great amount of efforts and works. The reasons why they have not supported ADTs from standard library are simply an issue of cost [10]. So, they certainly did not address the problems we describe in this paper. Besides, they focus on global analysis, whereas compositional analysis and refactoring are our major concerns.

Supporting abstract data types in Promela can be done in another way. For example, we can use CPP's macros to implement ADTs. This approach is easier to maintain and implement. There is no need to modify Promela's grammar. However, it is more difficult to cope with object-oriented syntax nowadays. For example, many ADT objects may have methods all named *add()*. Solving naming conflicts in macro

programming is more difficult.

In COCOV, we use control flow graph for helping it to generate the CCS graph. Ideally, there is no need to construct CFG, to obtain CCS graph. In our experience, it is less convenient to traverse AST to obtain CCS state graph, so in this thesis, we improve COCOV and generate CFG as the intermediate representation for generating CCS state graph

5.2 Conclusion

We have described the implementation and improvement of *COCOV*, a tool for compositional concurrency verification with model refactoring including how to add abstract data types into *rc-Promela* and generate CCS state graph with abstract syntax tree (AST) and control flow graph (CFG).

In this thesis, we describe a special phenomenon of software verification – analysis (particularly compositional analysis) is sensitive to the modeling choices when array is used to implement some abstract data types. We give two examples to show that an implementation choice which is the definite choice in the view of programming may be a poor choice for analysis. On the other hand, a poor implementation choice for programming can be the best choice for analysis. We show that such sensitivity can be lessened if abstract data types are supported and the uses of array are suppressed. Two abstract data types SET and QUEUE are described.

Models rewritten with abstract data types have obvious advantages. First, the code speaks for itself so that our rc-Promela translator and refactoring engine have sufficient information to mechanically select the best implementation in a transparent way. Second, using abstract data types forces process behaviors to converge on a best one for analysis, therefore, precision of modeling is naturally ensured.

Although we have implemented some frequently ADTs, an essential question is whether all of the abstract data type can be well refactory. We still need to evaluate this question in the future.