On Synthesis of Concurrent Programs from Scenarios

Yutaka WATANABE

School of Information Science,
Japan Advanced Institute of Science and Technology

February 15, 1999

Keywords: scenario, concurrent program, hyper sequential programming, dependency relation, synchronizing actions.

In this research, we study synthesis of concurrent programs that consist of sequential programs running concurrently and communicating each other. In general, testing and debugging of concurrent programs are much more difficult than those of sequential programs, and so currently we have a few effective debugging tools. In addition, we need enormous time for checking all possible states of a concurrent program because of *the state space explosion problem*, i.e., the number of states of a concurrent program increases exponentially in the size of the program, and this makes analysis of concurrent programs very difficult. For these reasons, it is required that fewer errors should be included at early stages of making programs.

Nondeterministic behavior in concurrent executions are knows as one of characteristic properties of concurrent programs, and this may cause errors. Such errors are usually not easy to be found, comparing with debugging sequential programs.

Uchihira et al. proposed a new programming paradigm, called *hyper sequential programming*, to remove such kind of errors. In this method, a concurrent program is synthesized from a set of sequential executions of the concurrent programs called *scenarios*, where each scenario represents an expected correct behavior of the program and is given by the designer.

In hyper sequential programming, each execution sequence is treated as a sequence of symbols, by replacing each instruction of the program with a label. The meaning of each instruction is reflected on the dependency relation on the set of labels. There exists a kind of dependency between instructions such that the order of their execution affects the final results, e.g., *read* and *write* action for a shared variable have such dependency. We give this dependency as a reflexive and symmetric relation on the set of labels.

In this approach, first a scenario graph, which is a directed graph having each global state of the system as a node and each instruction as a directed edge is constructed from given scenarios. Next, synchronizing actions are inserted as directed edges in the scenario graph, and then the graph is projected onto each sequential program. When deletion of a synchronizing action does not change the equivalence to the scenarios with respect to the dependency, the synchronizing action is deleted. Finally, replacing each label with the corresponding instruction, we obtain a concurrent program that shows expected behavior.

This algorithm is not very efficient because deletion of synchronizing actions are done by trial and error. In addition, optimality of synthesized programs is not sufficiently discussed. In this research, we improve this method, and propose a new efficient algorithm for synthesizing concurrent programs acting correctly and containing no redundant synchronizing actions.

In the proposed algorithm, a set of scenarios is given as a regular expression over the set of labels. The algorithm consists of the following steps: decomposition of scenarios into blocks, insertion of synchronizing actions, restoring scenarios, projection onto each sequential programs, and code generation.

In the first step, decomposition of scenarios into blocks, a regular expression representing scenarios are decomposed into blocks with or without control structures.

Next, synchronizing actions are inserted into each block so as to keep the equivalence to the partial ordering defined by the scenario. Detail of this procedure is described as follows: first a directed acyclic graph representing the total ordering in the scenario is constructed, next its transitive closure is computed, directed arcs between nodes without dependency are removed, and finally its transitive reduction is computed. Remaining edges shows positions for inserting synchronizing actions. Through this procedure, inserting positions of synchronizing actions are uniquely determined. Uniqueness is proved by the result that the transitive reduction of a directed acyclic graph is unique.

After this step, the scenarios are restored by substituting each block symbol by corresponding block. The restored scenarios are projected onto each sequential program, and a concurrent program is obtained by replacing each label with the corresponding instruction,

For the scenarios without control structure, the concurrent program synthesized by the proposed algorithm acts correctly to the scenarios, and contains no redundant synchronizing actions. Moreover, it is optimal in the sense that it generates all sequences permitted in the partial ordering defined by the scenarios. For the scenarios with control structure, we can also prove the correctness of its behavior, but the optimality is not necessary guaranteed. Because we treat each loop and branches as blocks in the algorithm.