Chapter 1. Introduction

1.1 Motivation

Along with the rapid changes of software development, concurrent applications have been widely used for many practical requirements. How to guarantee the correctness of concurrent programs has become a popular research topic in software engineering discipline. Most concurrency errors, such as deadlock, starvation, race condition etc, are hard to detect by traditional testing techniques due to the non-deterministic execution of concurrent programs.

Many considerable progresses have been proposed in recent years. The *model driven approach* [1][2] is served as the most practical way in verifying concurrent systems automatically. This kind of approaches abstracts the model of a system in the form of a finite state machine, which simply represents the behavior of a system by a kind of labeled state-transition system. Ideally, after modeling a whole system, verification tools can analyze the model with respect to the desired properties, such as the ones specified in temporal logic, by exhaustively exploring the reachable states of the entire model.

But, there are difficulties for real applications. The most critical problem is the excessive exhaustion of memory, which is known as *state explosion problem*.

Constructing a whole system model at once results in blowing up the available memory space easily and fails in further analysis. Therefore, many approaches for alleviating state explosion problem are presented.

In contrast to explicitly enumerating the state space of a model, *Ordered Binary Decision Diagram* (OBDD) [8] is a typical method to represent a system model with a kind of characterized graph data structure for saving more memory, but this approach has shown (in [9]) no obvious merit in software verification due to the wide variety of software designs and structures.

Compositional analysis is another promising approach to mitigate the state explosion problem [10][11][12][13][14][15]. In contrast to constructing a whole system model at once, compositional analysis first breaks apart a system into many primitive processes formed in a hierarchical structure, and then incrementally composes those closely associated processes into a subsystem from a bottom-up manner. This approach scales better due to the minimization by hiding internal behaviors among associated processes which are not concerned by processes outside the subsystem. In this thesis, we call the composed and minimized subsystem an *interface process*, which means that it contains only those actions interacted with other processes existed in the abstract model of the composed subsystem, but its behavior is equivalent to the original one with bisimulation relation.

However, compositional analysis is sensitive to the compositional hierarchy. Yeh and Young [15] have shown that the as-built structure of a practical concurrent system is often unsuitable for compositional analysis. To address the problem, Cheng et al. [19] have proposed an approach called *model refactoring* to enable compositional analysis for systems which are originally prohibited by their as-built architectures. The refactoring technique divides and transforms the original system model into new structured models which are more amenable to compositional analysis.

In addition to good hierarchies, many researches [12][14][17] have also found that processes in a concurrent system may require the coordination with other neighboring processes in the same system which form the *context* (or environment) of the process. The behavioral constraints imposed onto the process by its context are referred to as context constraints [12]. Inclusion of these constraints in composing the processes of a subsystem may well eliminate those behavioral traces forbidden within the global context [17]. Cheung et al. in [17] have proposed an automated approach for compositional analysis.

Moreover, in order to obtain an interface process with simple interface to represent a subsystem's behavior, there are several reduction and minimization techniques which can be applied to the composed subsystem to mitigate its state space efficiently. Reduction strategies [1][21][22][23] decrease the use of memory space by

eliminating those unconcerned states and transitions in the composed subsystems. By contrast, minimization techniques scale down the model of subsystem's behavior by equivalent relations [3][24]. In this thesis, we choose to implement the ***branching bisimulation minimization*** due to its nice properties [24][26] and the efficiency of its algorithm [27]. Below is our objective to the thesis.

1.2 Objective

In conventional model checking techniques, minimization was done after enumerating the whole state space of a system. This has shown in [29] that the cost of performing the minimization outweighs that of conventional model checking. If the state space of global composing is tractable, then it is wasting time to minimize the model before checking properties.

But this result described above is not true for compositional analysis. In compositional analysis, automatically generated interface process for a subsystem is attractive. Bisimulation minimization therefore remains as an important means for the approach. In this thesis, we call this approach an ***off-line minimization*** which means that the minimization is done after the state space of a subsystem is generated.

Although off-line minimization seems to work well after composing, it is possible to trigger the state explosion during composing a quite large subsystem in practice,

especially when memory used is closer to the memory limits of the hardware system. The verification tool may spend most time on thrashing hardly due to the page replacement of virtual memory.

The first objective of this thesis is to implement an *on-the-fly minimization* integrated with compositional analysis. Because it is possible to fail for off-line minimization due to composing a quite large subsystem, we hope to lower the risk of abruptly growing on state space immediately in the progress of composing. For instance, the state number of a composed subsystem without minimization may be 10,000, but that with minimization may be only 1000. Therefore, we advocate doing minimization on the fly when the partially enumerated state space is larger than a given threshold.

We propose an algorithm for integrating branching bisimulation minimization based on [27] into compositional analysis to achieve on-the-fly minimization. We design an interface to interact with parallel composition engine. It can be easily invoked without modifying the module of branching bisimulation minimization.

The advantage of the on-the-fly approach is to avoid state explosion and stop O.S. from entering thrashing state while composing quite large subsystems. . But there is cost to pay for it – the more time in minimization in composing, the more time has to spend for completing the composing than the time of off-line minimization. First

reason is the cost of procedure calls wrapped in the interface procedure which may lower the system performance. Secondly, each call to minimization procedure could not presume which group of states had been minimized in previous round and ignore them. The minimization procedure serves the given model as an undiminished one and minimizes it entirely each time. Thirdly, the time spending for each minimization may increase along with the growing state space. Often the sum of time for on-the-fly minimizations may exceed the time of one off-line minimization.

The second objective of the thesis is to present several ideas on determining the threshold for triggering the minimization in order to tackle the problem described above in a better way.

1.3 Thesis Organization

In chapter 2, we review the basic concept for compositional analysis and branching bisimulation equivalence. An implementation details for branching bisimulation minimization base on [27] is presented in chapter 3. In chapter 4 we propose an on-the-fly minimization algorithm fully integrated with general compositional analysis, and give some approaches on determining the threshold for triggering the minimization in chapter 5. Chapter 6 shows evaluations of some practical examples with our tool. Chapter 7 is the summary of the thesis.