Abstract

Software applications play an important role in our lives. The failure of the applications may harm people or equipment. Therefore, the correctness of the software is important. In fact, an application may consist of multiple processes, which are developed based on programming languages and operating systems (OSs). Under the mechanisms provided by these languages and environments, the processes can run simultaneously to increase the scalability. The applications are called concurrent systems. In fact, these systems are error-prone; for example, deadlock, livelock, or violations of constraints may occur in them. Because the processes of a concurrent system can be executed in different orders, their behaviors are difficult to verify.

As an exhaustive and automatic technique, model checking explores every execution of a system and automatically find possible errors. In comparison with other techniques, such as testing and simulation, model checking is more suitable to verify the concurrent systems. To model checking a system, we need to specify its behaviors (usually in a modeling language); then travel all the states of the system (called the state space) represented by its model using a search algorithm to check the corresponding property.

With the increasing of the complexness of a concurrent system, there is a need to schedule the execution of the processes. There are several scheduling strategies applied by real systems. For instance, in OSEK OS for automotive devices, an application can have multiple tasks executed under the priority and mixed preemption strategy. In model checking, the behaviors of a scheduler associate with the algorithm that explores the state space. However, verifying a concurrent system with considering all possible executions (interleaving behaviors) is an over-approximation approach and can produce spurious counterexamples because the errors may occur outside the executions indicated by the scheduler. Therefore, to accurately verify the systems, we need to take the scheduler into account in the verification.

Current methods in model checking to deal with sequential/concurrent systems are difficult to apply to verify with scheduling policies because these methods consider a different kind of behaviors and can cause spurious counterexamples. To deal with the scheduling policies, existing approaches try to limit the executions of the systems by encoding both of the processes and the scheduler into a model using a modeling language (e.g. Promela). In this case, the scheduling policy needs to be specified from scratch. This approach is hard to model interesting schedulers, error-prone, and time-consuming. This means that an approach to easily and flexibly describe the scheduling policies is needed.

In reality, the OSs use different policies to control the executions of the processes. For example, Linux OS can support several policies for its tasks based on their priorities (e.g. *round-robin* and *first-in-first-out*). However, the existing approaches cannot deal with the variation of the schedulers because the policy is fixed in the model of a system. That means to ensure the accuracy of the concurrent systems, a study on facilitating the variation of schedulers in model checking is necessary and important.

To overcome the problems above, in this research, we propose a method to analyze and verify concurrent systems executed under different scheduling policies using model checking techniques. Our method contains three main parts: 1) a language for modeling the processes, 2) a domain-specific language (DSL) to describe the scheduling policies, and 3) an algorithm to search all of the states of the system.

The originality of this research is proposing a DSL to specify the scheduling policies used in model checking techniques. In this approach, our language aims to provide a high-level support for specifying different policies easily. All the information necessary to analyze the system are automatically generated. From the specification of the scheduling policy in the DSL, a search algorithm is realized to explore the state space. Following this approach, we implemented a tool named SSpinJa, which is extended from SpinJa, a model checker implemented in Java. Our experiments indicate that the method is practical; it is easy to describe different scheduling policies and accurately verify the behaviors of the systems. In addition, in this research, we apply model-based testing techniques to generate the tests to check the correspondence between the policy in our DSL and the real scheduler in an OS; it helps us to increase the confidence of the policy in the DSL and accurately verify the systems.

The impact of this research is that we can easily apply model checking techniques to verify the concurrent systems with the different scheduling policies. The state space to be searched is now limited because the scheduler is taken into account in the verification. Therefore, we can verify systems more accurately. In addition, with our method, we can reuse the specifications of the processes and the scheduling policy. It helps to decrease the time necessary for designing and developing a concurrent system.

Key words: concurrent systems, model checking, scheduler, domain-specific language, model-based testing