Efficient Model Checking of Applications with I/O

Cyrille Artho[1], Boris Zweimüller[2], Armin Biere[3], and Shinichi Honiden[1]

[1] National Institute of Informatics, Tokyo, Japan
[2] Computer Systems Institute, ETH Zürich, Switzerland
[3] Johannes Kepler University, Linz, Austria

Abstract. Most non-trivial applications use input/output (I/O), such as network communication. When model checking such an application, a simple state space exploration scheme is not applicable, as the process being model checked would replay I/O operations when revisiting a given state. Thus a software model checker requires a faithful model, or it has to encapsulate such operations in a cache layer that is capable of hiding redundant operations from external processes.

1 Introduction

Model checking explores the entire behavior of a system under test (SUT) by investigating each reachable system state [3] for different thread schedules. Recently, model checking has been applied directly to software. However, conventional software model checking techniques are not applicable to networked programs. The problem is that state space exploration involves backtracking. After backtracking, the model checker will execute certain parts of the program (and thus certain I/O operations) again. However, external processes, which are not under the control of the model checking engine, cannot be kept in synchronization with backtracking, causing direct communication between the SUT and external processes to fail.

2 Model checking distributed programs

State space exploration of a multi-threaded program analyzes all possible interleavings between threads. Alternative schedules are explored by storing the current program state and executing copies of said program state under different schedules. When model checking a SUT that is part of a distributed system using multiple processes, external processes are not backtracked during model checking. Thus, two problems arise:

1. The SUT will resend data after backtracking. This will interfere with the correct functionality of an external process.
2. After backtracking, the SUT will expect external input again. However, an external process does not resend previously transmitted data.

One possible solution to this problem is to lift the power of a model checker from process level to operating system (OS) level. This way, any I/O operation is under control of the model checker [4]. However, this approach suffers from scalability problems, as the combination of multiple processes yields a very large state space.

Similar scalability problems arise if one transforms several processes into a single process by a technique called *centralization* [5]. With a TCP/IP model, networked applications can be model checked, but the approach does not scale to large systems [1].

Our approach differs in that it only executes a single process inside the model checker, and runs all the other applications externally. Inter-process communication is supported by intercepting any network traffic in a special cache layer. This cache layer represents the state of communication between the SUT and external processes at different points in time. After backtracking to an earlier program state, data previously received by the SUT is replayed by the cache when requested again. Data previously sent by the SUT is not sent again over the network; instead, it is compared to the data contained in the cache. The underlying assumption is that communication between processes has to be independent of the thread schedule. Therefore, the order in which I/O operations occur must be consistent for all possible thread interleavings. If this were not the case, behavior of the communication resource would be undefined. Whenever communication proceeds beyond previously cached information, the new data is both physically transmitted over the network and also added to the cache. The only exception to this is closing a connection. The cache simulates the effect of closing communication but allows connections remain physically open for subsequent backtracking.

Initial experiments using the JNuke model checker [2] have shown the scalability and viability of our approach. It covers systems where the response sent to a client does not depend on the input of other clients. This includes web servers, time servers, and other services where clients cannot interact, but precludes systems such as chat servers. Our initial implementation was not flexible enough to handle certain interleavings between several clients; ongoing work aims at creating a cache model that can handle such communication patterns.

3 Conclusions and Future Work

Simple backtracking is not applicable to model checking distributed programs because external applications are not under control of the model checker. In order to solve this problem, I/O operations are intercepted by a special backtracking-aware cache layer. Implementations of services where clients do not interact can then be model checked. Future work includes a necessary relaxation regarding the order of I/O operations in order to make the approach applicable to a wider range of programs.

References

1. C. Artho and P. Garoche. Accurate centralization for applying model checking on networked applications. In *Proc. ASE 2006*, Tokyo, Japan, 2006.
2. C. Artho, V. Schuppan, A. Biere, P. Eugster, M. Baur, and B. Zweimüller. JNuke: Efficient Dynamic Analysis for Java. In *Proc. CAV '04*, Boston, USA, 2004. Springer.
3. E. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT Press, 1999.
4. Y. Nakagawa, R. Potter, M. Yamamoto, M. Hagiya, and K. Kato. Model checking of multi-process applications using SBUML and GDB. In *Proc. Workshop on Dependable Software: Tools and Methods*, pages 215–220, Yokohama, Japan, 2005.
5. S. Stoller and Y. Liu. Transformations for model checking distributed Java programs. In *Proc. SPIN 2001*, volume 2057 of *LNCS*, pages 192–199. Springer, 2001.