Architecture-aware Partial-order Reduction to Accelerate Model Checking of Networked Programs

Cyrille Artho, Yoshinori Tanabe, Etsuya Shibayama
National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, Japan

Watcharin Leungwattanakit, Masami Hagiya
University of Tokyo, Tokyo, Japan

Programs are often structured into a main thread that delegates incoming requests, and worker threads. A similar structure also exists in applications where several processes have been merged ("centralized") into a single application. Such a transformation wraps processes as threads. and is used to model check networked programs. A direct implementation of wrapping allows for interleavings between initialization and execution of client threads. We present a partial-order reduction which, when applies to such programs, eliminates exploration of such interleavings. —

Most software model checkers [4] cannot handle multiple processes. To model check multiple processes in a single-process model checker, *centralization* has been proposed [3]. Centralization wraps several processes in a single process. Using a TCP/IP model library, networked applications can then be model checked [1]. However, the large number of thread interleavings limits scalability. Therefore, it is useful to optimize state space search as far as possible.

After centralization of an application, wrapper code runs as the main thread. The wrapper first starts the server process as a separate thread, and waits for its initialization to complete. After that, *initialization* and *execution* of each client is performed. This creates possible interleavings: After the first client is ready, it may already execute, even though the main (wrapper) thread is still initializing other clients. The model checker may analyze such interleavings, even though initialization of clients (in the main thread) does not interfere with execution of other clients. In simple programs, the model checker recognizes the redundancy in these interleavings. For more complex cases, the built-in partial order reduction fails. This observation led to a custom partial-order reduction. It takes this architectural property into account by only allowing schedules where the main (wrapper) thread finishes before client threads execute.

Using JPF version 3 [4] on small centralized programs [1], the gains achieved were not significant, because few client threads are used. However, in a more recent case study based on a different approach to analyzing networked software [2], a more complex client was analyzed. In that case, our manual optimization resulted in a significant speed-up. More work remains to be done whether centralized applications can be accelerated as well in some cases.

In the talk, reachability-based partial-order reduction in JPF is introduced first. It works on top of garbage collection. Second, custom partial-order reductions will be explained. They can be implemented either through program instrumentation or by extending the default search algorithm.

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, B. Zweimüller, A. Biere, E. Shibayama, and S. Honiden. Efficient model checking of applications with input/output. *Post-proceedings of Eurocast 2007*, 2007. To be published.

[3] S. Stoller and Y. Liu. Transformations for model checking distributed Java programs. In *Proc. SPIN 2001*, volume 2057 of *LNCS*. Springer, 2001.

[4] W. Visser, K. Havelund, G. Brat, S. Park, and F. Lerda. Model checking programs. *Automated Software Engineering Journal*, 10(2):203–232, 2003.