Model-based Verification of Distributed Software

Wytse Oortwijn and Marieke Huisman

Formal Methods and Tools, Dept. of EEMCS, University of Twente
P.O.-box 217, 7500 AE Enschede, The Netherlands
{w.h.m.oortwijn, m.huisman}@utwente.nl

Keywords: deductive verification, model-based verification, distributed systems.

Distributed components that operate and interact concurrently are ubiquitous in industry software. Many safety-critical systems consist of concurrent components, yet providing any guarantees on their correctness is very challenging. Therefore, developing tools and techniques that allow to prove functional correctness of such software systems is essential.

In the field of program analysis, several approaches have been proposed that aim to increase the reliability of concurrent systems or to give guarantees about their correctness. For example, model checking considers an abstract model of a program and checks if it satisfies certain temporal properties. Another approach is static/deductive verification, in which the source code is annotated with a specification language, e.g. JML, that describes the program behaviour. Automated verifiers are then used to verify whether the program satisfies the described behaviour. This research aims to combine the strengths of both approaches.

Approach. Our approach is three-fold. First, an abstract model of the program is defined that captures the concurrent program behaviour. Modelling is done by specifying and composing *atomic actions* that match with the interactions between the different program components, e.g. message exchanges between components or accesses to shared memory locations. Secondly, we verify deductively whether the program behaves as specified by the abstract model. To do this, we extend well-known program logics (i.e. separation logic [4]) to connect the atomic actions in the abstract model with the concrete fragments in the source code that perform/handle the concurrent interactions. Thirdly, model checking is applied to analyse the abstract model. As result of the second step in our approach, the model checking results apply to the actual program and can thus be used in the program logic.

Our approach applies to concurrent software in general, but the research focusses on distributed software and targets both shared- and distributed-memory systems [3]. The results of this research can for example be applied to verify functional correctness of message-passing programs, e.g. MPI, and systems performing remote procedure calls, like web services.

Research directions. To abstractly model program behaviour we currently use a process algebra with support for data, similar to mCRL2 [2], as this formalisation benefits from extensive research on concurrent system analysis. We have formalised our approach and implemented tool support as part of the VerCors toolset [1], available at [5].

The current tool implementation targets concurrent programs. We now aim to cover distributed programs as well. Moreover, the current formalisation only preserves safety properties. We plan to investigate the preservation of progress and liveness properties and we plan to combine our approach with rely-guarantee reasoning.

References

1. S. Blom and M. Huisman. The VerCors tool for verification of concurrent programs. In *FM*, volume 8442 of *LNCS*, pages 127–131, 2014.
2. J.F. Groote and M.R. Mousavi. *Modeling and Analysis of Communicating Systems*. MIT Press, 2014.
3. W. Oortwijn, S. Blom, and M. Huisman. Future-based static analysis of message passing programs. In *PLACES*, pages 65–72, 2016.
4. J.C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Logic in Computer Science*, pages 55–74. IEEE Computer Society, 2002.
5. The VerCors tool online. http://www.utwente.nl/vercors/.