Verifying Web Services over Unbounded, Reliable Channels

Alexander Heußner

Université Libre de Bruxelles

Abstract: Proving the maturity of the `verify` tool by improving a case study on web service business protocols that was not able to handle unbounded fifo channels.

Keywords: safety verification, distributed systems, web services, fifo channels, tool

Introduction The automatic verification of distributed algorithms and communication protocols is one of the most crucial tasks in software/hardware development and maintenance. It is also one of the hardest, e.g., as one cannot directly infer the global behaviour of a distributed system from its local components due to asynchronous communication. This renders already simple analysis, verification, and synthesis questions hard problems in theory. However, in practice, this leads to a growing demand for versatile tools that also apply semi-algorithmic solutions, approximations, abstractions, and heuristics.

We focus on the reachability/safety verification of *communicating finite-state machines* (CM), an infinite-state formalism that consists of a set of local, finite state machines that communicate via global, asynchronous, reliable and unbounded fifo channels. The latter are demanded in practice by, e.g., distributed applications based on TCP, the Sockets API, Erlang's message channels, or MPI, but also MSCs and UML state-charts. Note that CM do not demand the channels to be a priori point-to-point. The safety verification question demands, given a CM and a set of "bad" states, whether no execution of this CM reaches the bad states. It is folklore that CM are Turing powerful, hence this question is undecidable for CM.

Currently, there is no specific tool—at least to our knowledge—that directly focusses on the safety verification of CM. State-of-the-art approaches, like SPIN or CADP, rely on a priori bounding the size of the channels. The UPPAAL tool restricts channels to asynchronous rendezvous, TReX focusses on lossy channels, however both permit more powerful local process models. These under- & over-approximations for CM's reliable, unbounded channels, however, easily introduce false positives or false negatives with respect to safety verification of protocols. As CM are infinite-state systems, tools for this more general setting, like ARMC, would also be applicable. These, however, do not easily allow to input CM, and rely on abstractions for the infinite data domain that are not specifically adapted to fifo channels' intricacies.

We aim at filling this gap by presenting a *Model Checker for Systems of Communicating Machines* (McScM) that combines different algorithms for the safety verification problem of CM under the same roof and provides a ready-to-use front-end with the tool `verify`. The implemented algorithms are an abstract interpretation based approach (absint), abstract regular model checking (armc), counter-example guided abstraction refinement (cegar), and tree-refinement based lazy abstraction refinement (lart). McScM was introduced in more detail in [HGS12].

Web Services Case Study In the following we briefly show the merit of `verify` by comparing to a case study for verifying web services business activity protocols [RSV11], originally

	absint	armc	cegar	lart
BAwCC	1 s / 7.75 MiB	—	—	—
BAwCC (enh.)	0.37 s / 6.85 MiB	2.66 s / 29 MiB	8.5 s / 20.41 MiB	—
BAwPC	1.49 s / 11.69 MiB	3.05 s / 46.56 MiB	206 s / 126.97 MiB	—
BAwPC (enh.)	0.5 s / 6.84 MiB	0.16 s / 7.81 MiB	0.36 s / 6.84 MiB	0.95 s / 7.81 MiB

Figure 1: `verify`'s different algorithms applied to the four web service business protocols of [RSV11] (results: running time / memory for proving safety, "—" if time $>$ 10 min, on off-the-shelf computer with Intel Core i5 1.7GHz)

implemented with the help of UPPAAL. As protocols from the web service business activity protocol stack are crucial for protocols implemented on top of it, automatic verification with respect to safety requirements is a crucial task.

The approach in [RSV11] was able to verify the Business Agreement with Participant Completion (BAwPC) and Business Agreement with Coordinator Completion (BAwCC) protocols under the assumption that channels are bounded a priori, by modelling channels explicitly in UPPAAL. This, however, proved to be the weak point: as explicitly representing k-bounded channels by a sequence of k processes leads to a combinatorial explosion of the global system's state space. Hence, the original study was not able to derive verification results for more than two channels of size larger than three with reasonable resources. Figure 1 shows our results on the same protocols with `verify` (translating their XML-based protocol format to our input format beforehand), which is able to prove that the protocols are safe when asserting *unbounded* reliable channels with reasonable resources. Note that our various implemented algorithms adapt differently for proving safety or finding counterexamples, hence, the "—" for the abstract-check-refine approaches which are better suited for thoroughly searching for counterexamples.

Thus `verify` disproves the conjecture in [RSV11] that "there is no hope to establish the correctness of the protocol with unbounded FIFO communication in a fully automatic way" [RSV11, p10] in a positive way. The case study in [RSV11] additionally verified boundedness properties and compared different channel policies, like lossy channels. When fixing a bound k for the channels, k-boundedness can be reduced to safety and lossy channels are also verifiable in `verify`.

Conclusion / Outlook The `verify` tool with its different implemented algorithms proves to be the "swiss army knife" of safety verification for CM (as there is no silver bullet algorithm [HGS12], the choice of the right "blade" is up to the user). Further, McScM and thus `verify` are easily extensible to other algorithms due to their modular architecture. McScM also includes a synthesis tool that allows one to automatically fix unsafe protocols.

Currently, we plan to attack larger case studies, e.g., based on the industry derived examples in the CADP framework. `verify` is also used by external tools, e.g., for verifying distributed process models derived from log files, and will be included in the tool from [RSV11] as an additional back end. McScM and the `verify` tool can be downloaded with a series of example protocols, ranging from distributed leader election to a BRP-like protocol, at:

http://altarica.labri.fr/forge/projects/mcscm/wiki/.

References

[HGS12] A. Heußner, T. L. Gall, G. Sutre. McScM: A General Framework for the Verification of Communicating Machines (Tool). In *Proc. of TACAS'12*. LNCS 7214, pp. 478–484. Springer, 2012.

[RSV11] A. P. Ravn, J. Srba, S. Vighio. Modelling and Verification of Web Services Business Activity Protocol. In *Proc. of TACAS'11*. LNCS 6605, pp. 357–371. Springer, 2011.