

Axiomatic Analysis of Distributed Systems

Levente Bajczi 

Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary
levente.bajczi@edu.bme.hu

Vince Molnár 

Department of Measurement and Information Systems
Budapest University of Technology and Economics
Budapest, Hungary
molnar.vince@vik.bme.hu

Abstract—Safety critical embedded software require comprehensive proofs of safety. A mathematically precise way of ensuring freedom from bugs is formal verification, but that is often too complex to be applied to entire systems. However, embedded components seldom exist on their own, but rather as part of hardware-software systems. Communication protocols used among such components have often been the cause of insuperable complexity. This paper proposes a novel solution to this verification problem by extending concepts of weak memory modeling to communication protocols.

Index Terms—embedded software, formal verification, communication protocols, weak memory, CAT

I. INTRODUCTION

Embedded software govern many aspects of our lives where bugs could lead to serious monetary or physical harm. In such environments ensuring *safety*, i.e., freedom from unacceptable risk of harm is paramount to the development of any system. While conventionally testing has been used to achieve a level of certainty in the correctness of systems, it is not exhaustive by design and therefore obscure bugs might remain hidden. In contrast, *formal verification* is a mathematically precise way of reasoning about the safety of system, but its complexity remains to be prohibitive for most real-world systems.

One technique to overcome this barrier is to prove safety for individual components of the system, and thus inductively prove that the whole system is safe as well. Most notably, communication among components are usually abstracted away as nondeterministic inputs to the software, which is a safe but coarse over-approximation [1]. Thus, this method can very easily lead to *false alarms*, as communication partners seldom behave completely non-deterministically.

The main reason why communication is not usually included in the input to the formal verification algorithm is its inherent complexity. Some protocols will behave erratically with duplicated, reordered and lost messages; while others define strict sequences over sent and received messages. Enumerating all executions given just a few participants often results in an explosion of execution traces.

Note however, that handling the semantics of non-sequential communication patterns is not a unique prob-

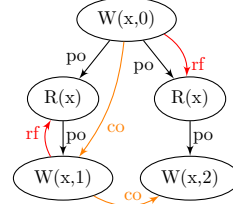


Fig. 1. Candidate execution

$\langle r \rangle ::= \langle b \rangle$	$\mid \langle r \rangle \cup \langle r \rangle$	$\mid$
$\langle r \rangle \cap \langle r \rangle$	$\mid \langle r \rangle \setminus \langle r \rangle$	$\mid$
$\langle r \rangle ; \langle r \rangle$	$\mid \langle r \rangle^{-1}$	$\mid$
$\langle r \rangle^+$	$\mid \langle r \rangle^*$	$\mid$
$\langle b \rangle ::= \langle id \rangle$	$\mid \langle s \rangle \times \langle s \rangle$	
$\langle s \rangle ::= \langle id \rangle$	$\mid \langle s \rangle \cup \langle s \rangle$	$\mid$
$\langle s \rangle \cap \langle s \rangle$	$\mid \langle s \rangle \setminus \langle s \rangle$	

Fig. 2. CAT rules

lem. Modern multi-core processor architectures utilize re-ordering and caching of shared memory operations as a means to speed up execution by mitigating direct memory access latency [2]. In such cases, analysing dataflow incurs the same level of complexity as communication protocols do. The solution in the multi-core case seems to be to use *axiomatic* models instead of *behavioral* ones [3], which means that only the effects of the memory consistency model need to be handled, and not its implementation in the hardware.

The *de-facto* axiomatic memory modeling language is CAT [4]. Most tools use CAT [3] or a subset of CAT [5], while others use similar concepts easily traceable to elements of CAT [6]. In this paper, I show how to model communication protocols in CAT and exploit existing verification approaches.

II. THE CAT LANGUAGE

To model weak memory models, CAT uses the concept of *candidate execution graphs*. An execution graph is a collection of vertex-labelled directed graphs $G(V, E, l_V)$, where l_V labels vertices in V with one or more *event sets*, and each G conforms to a binary *relation* defined over events [4].

Consider the example in Figure 1. It corresponds to an execution of a program with 5 memory events, and it shows 3 relations over these events: *po* for *program order* (i.e., the natural order of events stemming from the program source); *rf* for *read-from* (showing dataflow between *write* and *read* events); and *co* for *coherence order* (i.e., the actual order of write operations as seen by the shared memory). *Event sets* in this example are *reads* (R) and *writes* (W). Furthermore, events are shown to include the

corresponding variable (x in this example), but that is *not* part of the event set.

Such executions are considered *candidate executions* to a certain memory model, if we want to decide whether it would be *observable* on the target architecture. CAT defines derivation rules of relations and event sets as seen in Figure 2 (r : relation, b : basic relation, s : event set, id : name). Furthermore, CAT contains assertions on relations constraining them to be *(non-)empty*, *(ir)reflexive* or *(a)cyclic*.

III. COMMUNICATION AS MEMORY OPERATIONS

Conventionally, CAT was defined for shared memory operations (i.e., *Read* and *Write* events). While not encoded in the language itself, analysis tools can rely on assumptions such as a lack of duplication, always succeeding reads, etc. These do not always hold when the communication medium is *not* shared memory, as for example, it is trivial to not have any data to read (while shared memory will always have something at every address, even if only some uninitialized value). This means that conventional analysis tools [3], [5], [6] have to adapt their algorithms not to imply such constraints.

Without the constraints of shared memory imposed on communication protocols, CAT is capable of formulating guarantees of communication protocols easily. To show how such rules can be expressed, as well as to facilitate their easy prototyping, I determined a few key patterns that govern how commonly used protocols behave, and I created a *standard library*-like collection of memory modeling rules. These are not designed to be exhaustive (as modifications to these can easily be expressed in CAT), but rather to serve as a baseline for more complex patterns to build upon. These are the following:

Reordering Does the reception of messages follow some total order by all participants?

Duplication Can multiple receptions by the same participant occur from a single sent message?

Reliability Is the reception of a message guaranteed?

Broadcasting Can multiple participants receive a single sent message?

Sending synchronicity Can a participant issue new events before a sent message is received?

Blocking reception Can a participant receive data from nowhere, or will all receive instructions block?

IV. APPLICATION OF THE APPROACH

Consider the simple program in Figure 3. Given a communication protocol such as TCP, we need to create a corresponding memory model by taking its *guarantees* into account – it is important to note that the behavior is *not* explicitly modelled. As TCP guarantees no reordering, no duplication and no lost message (given a healthy connection), while being synchronous and blocking; the

send(x , 1)		$i = \text{receive}(x)$
$i = \text{receive}(y)$		send(y , i)
send(x , 2)		$j = \text{receive}(x)$

Fig. 3. Litmus test

```
TCP
include "no-reordering.cat"
call sc(rf, co, id, po)

include "global-no-dup.cat"
call nodup(rf, id)

include "no-loss.cat"
call noSendLost(W, IW, T, rf)

include "synchronous.cat"
call sync(rf, co, id, po)

include "blocking.cat"
call blocking(R, T, rf)
```

Fig. 4. TCP model

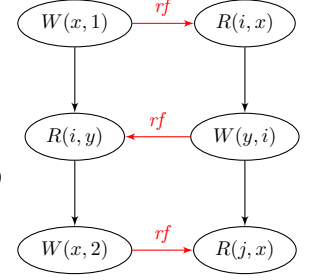


Fig. 5. Solution to Figure 3 over 4

representative memory model can be seen in Figure 4. Taking this model into account, the only candidate execution consistent with the memory model is the one in Figure 5.

V. CONCLUSION

As demonstrated by its application to TCP, the presented method is more than capable of verifying programs using message-based communication protocols. I implemented a proof-of-concept algorithm for the proposed approach in THETA [7], and preliminary results indicate that it is feasible in a practical setting. More work is necessary however to show how more complex protocols (such as some QoS-settings of the OMG-DDS standard) can be applied to CAT, or how to model heterogeneous systems observing weak memory *and* network communication.

REFERENCES

- [1] A. Tešanović, S. Nadjm-Tehrani, and J. Hansson, *Modular Verification of Reconfigurable Components*. Berlin, Heidelberg: Springer-Verlag, 2005, p. 59–81.
- [2] K. Gharachorloo, *Memory Consistency Models for Shared-Memory Multiprocessors*. Western Research Laboratory, 1995.
- [3] J. Alglave, L. Maranget, and M. Tautschnig, “Herding cats: Modelling, simulation, testing, and data mining for weak memory,” *ACM Trans. Program. Lang. Syst.*, vol. 36, no. 2, pp. 7:1–7:74, 2014.
- [4] J. Alglave, P. Cousot, and L. Maranget, “Syntax and semantics of the weak consistency model specification language cat,” *CoRR*, vol. abs/1608.07531, 2016.
- [5] H. P. de León, F. Furbach, K. Heljanko *et al.*, “BMC with Memory Models as Modules,” in *2018 Formal Methods in Computer Aided Design, FMCAD 2018*, N. Bjørner and A. Gurfinkel, Eds. IEEE, 2018, pp. 1–9.
- [6] M. Kokologiannakis, A. Raad, and V. Vafeiadis, “Model checking for weakly consistent libraries,” in *Conference on Programming Language Design and Implementation, PLDI 2019*, K. S. McKinley and K. Fisher, Eds. ACM, 2019, pp. 96–110.
- [7] T. Tóth, A. Hajdu, A. Vörös *et al.*, “Theta: a Framework for Abstraction Refinement-Based Model Checking,” in *17th Conference on Formal Methods in Computer-Aided Design*, D. Stewart and G. Weissenbacher, Eds., 2017, pp. 176–179.