Modular Verification of Message Passing Programs

Master's Thesis Project Description

Gaurav Parthasarathy
Supervisor: Alexander J. Summers

November 20, 2017

1 Introduction

Concurrent programs have become ubiquitous in practice. One way to deal with the communication between processes running concurrently in a program is to use shared memory and synchronization constructs such as locks. If one is not careful this can lead to deadlocks and scalability issues. A different approach that attempts to circumvent these problems is a message passing approach, where processes running concurrently communicate via asynchronous messaging. The actor model [2] is a model that follows this asynchronous messaging approach, where processes are independent units called *actors*.

Many concurrent programs (as well as distributed systems) have been implemented with the actor model as the main inspiration, using, for example, the programming language Erlang [1] or the Akka toolkit [4] which provides actor abstractions for different programming languages. We call such programs *actor programs*.

Formal reasoning about actor programs is important to ensure the programs have the desired properties. *Actor services* [3] is a program logic for actor programs. It enables modular verification of functional and response properties. Modularity in this context means that properties about parts of an actor program can be verified without knowledge of the complete program, and these properties can then be composed to derive specifications of larger parts of the program. Such modular reasoning is essential for the

reuse of verified specifications of subprograms that appear in different contexts. It also makes reasoning easier and scalable by not having to consider the behaviours of the whole program at once.

The actor services logic has not been used extensively for the verification of real-world programs. Due to the logic's limitations there are various properties of message passing programs which cannot be verified. The main goal of this project is to extend the actor services logic such that it can be applied to a wider class of existing programs.

2 Known Limitations of Actor Services

We give a brief overview of some known limitations of the actor services logic.

Can only use a single message as trigger. Actor services can be used to verify response properties where a single message triggers a response. It is not possible to verify properties where multiple messages trigger a response.

Cannot deallocate actors. Actors must always be ready to receive valid messages. As a result there is no way to model the deallocation of actors.

No support for protocols. There is no way of specifying that at certain points some messages cannot be received by actors. This means that in some cases the logic is overly conservative with respect to behaviour that cannot occur since actor programs may follow protocols.

Actor code assumptions. The actor services logic assumes that the code in each actor only communicates with other actors via messages. Yet in practice asynchronous message passing is sometimes combined with communication via constructs that block, such as channels.

3 Core Tasks

The core tasks in this project are the following:

- Identify appropriate programming domains and design patterns for which an extended actor services logic may be suitable.
- Analyze the strengths and limitations of the actor services logic by applying it to actor programs which are taken from the identified programming domains or contain the identified design patterns.

- Lift the most essential limitations by extending the logic. In an optimal case the extensions should preserve most of the modularity properties of the original logic.
- Verify actor programs using the extended logic.

4 Extension Tasks

Possible extensions after completing the core tasks are the following:

- Apply the extended logic to a larger corpus of examples.
- Prove soundness of the introduced program logic extensions.
- Design an approach for the automation of the actor services logic and the proposed logic extensions.

References

[1] J. Armstrong. *Programming Erlang: Software for a Concurrent World*. Pragmatic Bookshelf, 2007.

[2] C. Hewitt, P. Bishop, and R. Steiger. A universal modular actor formalism for artificial intelligence. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, IJCAI'73, pages 235–245, San Francisco, CA, USA, 1973. Morgan Kaufmann Publishers Inc.

[3] A. J. Summers and P. Müller. Actor services: Modular verification of message passing programs. In P. Thiemann, editor, *European Symposium on Programming (ESOP)*, LNCS, pages 699–726. Springer Berlin Heidelberg, 2016.

[4] D. Wyatt. *Akka Concurrency*. Artima Incorporation, USA, 2013.