Assigning Meaning to Form

Christos Kloukinas
Dep. of Computing, City University, London, EC1V 0HB, U.K.
Tel: +44.20.7040.8848 Email: `C.Kloukinas@soi.city.ac.uk`

Abstract

Complex critical systems need to be formally described and analysed (i.e., *engineered*). Many different formal methods have been developed for analysing almost all aspects of these systems and in certain cases even synthesising parts of these systems themselves through, e.g., controller synthesis. In many cases however, it is not at all obvious what the results of the tools developed for these methods really mean and practitioners can spend a considerable time trying to understand the results, such as a scheduling created by a compiler for a synchronous language or a timed-automata based controller for a real-time system. This situation decreases not only the use itself of these tools and the corresponding formal methods but also hinders the ability of the practitioners to *understand*, *optimise* and *validate* their systems.

1 Introduction

Convincing practitioners to start using formal methods tools for the design and analysis of their systems is not an easy task. Some of the reasons have to do with problems that the corresponding tools have in scaling up to be able to handle real world size problems and this is an area which has received considerable attention in the past years, leading to spectacular improvements. Another reason has to do with usability problems, since many of the tools demand a significant level of sophistication from the part of the user. This aspect has also received some attention, with some tools working in an automated, almost black-box fashion. Such examples are compilers for synchronous languages, which can produce ready to run code using a scheduling of the various tasks which has been computed by the compiler itself, or controller synthesisers which can produce a set of control constraints for a system so that it can meet its requirements. Some users seem to be still unhappy though. They look at the automatically constructed artifacts (compiled code, controller constraints, etc.) and try to understand them, usually failing miserably. The developers of the tools tend to find this situation a bit bizarre - after all, who would like to examine the machine code that a compiler outputs and expect to understand it? The sad truth however is that this situation is not caused by some users having strange, illogical expectations. Indeed, it is very often *imperative* for one to be able to *understand* the output of a tool for a number of reasons:

1. To ensure that the tool has not any *bugs* or at least has not introduced any bugs in the code.
2. To understand how the tool has translated some part of the code and see whether an *optimisation* could be made possible by expressing the input differently.
3. In general, to try to figure out *further optimisations* which could be applied to the automatically constructed artifact, e.g., further control constraints for properties which, for some reason, could not be described in the original model.
4. To be able to figure out what are the *assumptions* that the tool has used in order to derive its results and *validate* the artifact.
5. To be able to *describe* the artifact to some certification authority and thus have it *certified*.

2 Assigning meaning to form

In our case, we had this experience when we were trying to understand our synthesised scheduler for a small case study which we had created ourselves [2]. Even though there was only a single possibility for a deadlock, the scheduling constraints for avoiding it were not as simple as we might had hoped and it took us quite some time to fully understand why they were correct. This effectively meant that for a bigger system and more complex properties, such as timeliness properties, our synthesised scheduling constraints would be almost impossible to decipher. For these reasons we attempted to use *machine learning* techniques to better structure the scheduling constraints produced by our controller synthesiser and present them in a user-friendly form [1]. The initial results obtained were very positive, showing the core descriptions of the scheduling constraints in a manner which a practitioner (ourselves included...) could easily understand. It allowed to find out task *interdependencies*, easily identify possible *optimisations*, produced hints for further system *transformations*, such as *safely* reducing the non-determinism of the scheduler or distributing tasks to multiple processors, etc. These positive initial results have convinced us of the big potential of the marriage of machine learning/AI techniques with formal methods in solving the problem of understanding the artifacts produced by formal methods tools. As such, we believe that the community should start paying more attention to this problem and to how the different approaches (type systems, theorem provers, etc.) could take advantage of ML/AI techniques for rendering their results more easily understood by practitioners.

3 Conclusions

Increasing our level of confidence in the correct behaviour of a complex and highly critical system naturally requires the use of formal methods for its design and development. At the same time, it requires that the results of these formal methods can be easily understood by the practitioners, since no one can be confident of a proof/controller/etc. that they cannot really understand, no matter how much mathematical sophistication has been used in the design and development of the tools which have produced it. In order to increase our understanding of the artifacts of formal methods tools we believe that the community should seriously examine the use of machine learning (and possibly also HCI) techniques for restructuring the artifacts of the tools and presenting them in a way which is much more user friendly. By increasing our understanding of how the system works, we can then have an increased level of confidence in it, find further optimisations, be able to validate and certify it easier and in general reduce the overall cost of designing and developing these complex systems.

References

[1] C. Kloukinas. Data-mining synthesised schedulers for hard real-time systems. In *19th IEEE Conference on Automated Software Engineering (ASE-2004)*, pages 14–23, Linz, Austria, Sept. 2004. IEEE Press.

[2] C. Kloukinas and S. Yovine. Synthesis of safe, QoS extendible, application specific schedulers for heterogeneous real-time systems. In *ECRTS 2003*, pages 287–294, July 2003. DOI: `10.1109/EMRTS.2003.1212754` .