Model Learning and Testing

(and some observations on formal methods research in the Netherlands)

Frits Vaandrager
Institute for Computing and Information Sciences
Radboud University
Nijmegen, the Netherlands
F.Vaandrager@cs.ru.nl

1 Formal Methods in The Netherlands: Where Are We?

Formal methods is often defined as the applied mathematics of computer system engineering. The Netherlands has a strong tradition in this area. This started with scientific giants such as Van Wijngaarden and Dijkstra, continued with e.g., De Bakker, De Roever, Rozenberg, Rem, Barendregt, Bergstra and Klop, and led to the strong formal methods groups that we see at Dutch universities today. In addition, some highly visible Dutch formal methods researchers are active abroad, e.g., Katoen, Holzmann, Van Glabbeek and Bloem.

Still, my impression is that Dutch formal methods research is not as influential and authoritative as it used to be. A number of factors may have contributed to this:

1. The field has matured and changed considerably over the years. Whereas, for instance, Bergstra & Klop could create major impact with papers on complete axiomatizations of process algebras, nowadays a nice theoretical idea is not enough. You also must show (or at least make it plausible) that an idea can be implemented and be effectively used to advance the state-of-the-art of computer system engineering. This is a different game, where rather than one or two brilliant theoreticians, you need a whole team/network of researchers, with different people focusing on theory, tools and applications. Despite notable exceptions, the Dutch formal methods community as a whole has not adapted fast enough to this new reality. Too often, I see formal methods papers in which the introduction refers to the importance of correct software, but the proposed methods have not been applied to real software yet, and there is not even a plausible scenario of how the results could be applied to real systems. Too often, also, I see colleagues write papers on questions and research approaches that are almost identical to the ones they explored in their thesis many years ago.
2. The funding situation has not been very helpful with on the one hand the personal grants that support individuals (rather than teams/networks), and on the other hands projects that require direct support from industry (and typically have a focus on short term applications). The funding does not have the scale and time horizon needed to solve the challenges that our field is facing.
3. Maybe we no longer succeed to attract the most brilliant and ambitious students. Somehow, cyber security, artificial intelligence and data science appear to have a more attractive proposition.

2 Formal Methods in The Netherlands: Where Do We Want To Go?

So what should we do to address the above problems? A first thing is to bring the community together and to discuss the problems. I am most grateful to Marieke and Eelco for their initiative to organize this workshop. I suggest a couple of actions:

1. **Research agenda.** We need to identify a couple of major research challenges where we believe Dutch FM researchers can really make a difference on industrial practice within say seven years, and outline how we want to do it. Only by working together we can create the desired impact. Different teams may work on each of these challenges, combining expertise ranging from pure theory all the way to practical application. (Below I will discuss the research challenge we work on.)
2. **Industrial support crucial.** In his contribution for this meeting, Joost-Pieter Katoen argues for "More Programs, Less Models", and observes an international trend "from model-based to code-based analysis". In my view, there are many good reasons why the Dutch high-tech industry is still pursuing a model-based approach (high level of abstraction, possibility to simulate cyber-physical systems and explore design alternatives before they are built ("digital twins"), automatic code generation, etc,..). The presence of a number of big companies that invest in model-based development offers excellent opportunities for Dutch formal methods research, as long as we are willing to face the complexity of industrial design, and try to help people in industry with the problems they face (e.g., dealing with complexity of models and with legacy software for which no models are available. Having said that, I do think (based on my own experience and feedback from students who did internships) that within industry there is much ignorance about modern software

analysis techniques, and also within the formal method community we could use more expertise on e.g., static analysis and software verification.

3. **Funding strategy/lobby.** Once a national formal methods agenda is there, we should just start working on it, irrespective of whether the agenda is supported by funding agencies. Simultaneously, we should try to get funding from a variety of sources asap. For this it would be helpful if the agenda gets some formal status and is officially recognized by VERSEN, IPN, as part of the new sector plan, and/or even as part of the Dutch national science agenda. We should definitely lobby for a Jacquard like program funded by NWO. In the discussion about funding it is *urgent* to arrive at a clear division of work / areas of expertise between different formal methods groups in the Netherlands, e.g., via a list of ten subfields where each university claims a leading role in at most two or three topics.
4. **Attracting talent.** Clearly, having a convincing research agenda with challenging problems and a vision on how to tackle these problems will attract talent. My suggestion would be to organize another meeting at some point to exchange ideas on how we can attract more talented students to our area.

3 Model Learning and Testing

Active automata learning (or model learning) aims to construct black-box state machine models of software and hardware systems by providing inputs and observing outputs. State machines are crucial for understanding the behavior of many software systems, such as network protocols and embedded control software, as they allow us to reason about communication errors and component compatibility. Model learning is emerging as a highly effective bug-finding technique [1]. It has been successfully used in several different application domains, including

- generating conformance test suites of software components, a.k.a. *learning-based testing*,
- finding mistakes in implementations of security-critical protocols,
- learning interfaces of classes in software libraries,
- checking that a legacy component and a refactored implementation have the same behavior.

There is certainly a large potential for application of model learning to many different aspects of software development, maintenance and refactoring, especially when it comes to handling legacy software. To realize this potential, two major challenges must be addressed: (1) currently, techniques do not scale well, and (2) they are not yet satisfactorily developed for richer classes of models.

One way to address these challenges is to augment model learning with white-box information extraction methods (e.g., symbolic execution, tain analysis, static analysis), which are able to obtain information about the system-under-learning at lower cost than black-box techniques. When dealing with computer-based systems, there is a spectrum of how much information we have about the code. For third party components that run on separate hardware, we may not have access to the code at all. Frequently we will have access to the executable, but not anymore to the original code. Or we may have access to the code, but not to adequate tools for analyzing it (this often happens with legacy components). If we can construct a good model of a component using black-box learning techniques, we do not need to worry about the code. However, in cases where black-box techniques do not work and/or the number of queries becomes too high, it makes sense to exploit information from the code during model learning.

Active automata learning is closely related to model-based testing, and both activities can be viewed as two sides of the same coin. Whereas automata learning aims at constructing hypothesis models from observations, model-based testing checks whether a system under test conforms to a given model. Model-based test tools play a crucial role within active automata learning, as a way to determine whether a learned model is correct or not. For this reason, the activities in Nijmegen on model learning and model-based testing are closely aligned, and inspire/challenge each other.

Within our group, Nils Jansen recently started as an assistant professor working at the intersection between formal verification, machine learning, and control theory, along the lines described in Sections 4 and 5 of Katoen's contribution.

Research objective. Our objective is to reach — within say seven years – the point where active automata learning and model based testing have become standard tools in the toolbox of the software engineer, equally mature as model checking is right now.

Collaboration. Our group e.g., collaborates closely with the groups of Bernhard Steffen and Falk Howar at the TU Dortmund on learning tools, with the group of Andreas Zeller from Saarland University on the use of taint analysis in learning, with TNO ESI on the model-based testing tool TorXakis, with the Digital Security Group in Nijmegen on case studies related to security, and with ASML and Philips Healthcare on case studies related to refactoring of legacy software. All these collaborations are vital for reaching our research objectives. We would be most interested to collaborate with other groups in the Netherlands, e.g., on the use of white-box analysis techniques in automata learning.

References

[1] F.W. Vaandrager. 2017. Model Learning. *CACM* 60, 2 (Feb. 2017), 86–95. https://doi.org/10.1145/2967606