Report from Dagstuhl Seminar 16172

Machine Learning for Dynamic Software Analysis: Potentials and Limits

Edited by

Amel Bennaceur[1], Dimitra Giannakopoulou[2], Reiner Hähnle[3], and Karl Meinke[4]

1 **The Open University – Milton Keynes, GB,** amel.bennaceur@open.ac.uk
2 **NASA – Moffett Field, US,** dimitra.Giannakopoulou@nasa.gov
3 **TU Darmstadt, DE,** haehnle@cs.tu-darmstadt.de
4 **KTH Royal Institute of Technology – Stockholm, SE,** karlm@kth.se

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 16172 "Machine Learning for Dynamic Software Analysis: Potentials and Limits". Machine learning is a powerful paradigm for software analysis that provides novel approaches to automating the generation of models and other essential artefacts. This Dagstuhl Seminar brought together top researchers active in the fields of machine learning and software analysis to have a better understanding of the synergies between these fields and suggest new directions and collaborations for future research.

Seminar April 24–27, 2016 – http://www.dagstuhl.de/16172
1998 ACM Subject Classification D.2 Software Engineering, D.2.4 Software/Program Verification, I.2 Artificial Intelligence, I.2.6 Learning
Keywords and phrases Machine learning, Automata learning, Software analysis, Dynamic analysis, Testing, Model extraction, Systems integration
Digital Object Identifier 10.4230/DagRep.6.4.161

1 Executive Summary

Amel Bennaceur
Reiner Hähnle
Karl Meinke

Machine learning of software artefacts is an emerging area of interaction between the machine learning (ML) and software analysis (SA) communities. Increased productivity in software engineering hinges on the creation of new adaptive, scalable tools that can analyse large and continuously changing software systems. For example: agile software development using continuous integration and delivery can require new documentation models, static analyses, proofs and tests of millions of lines of code every 24 hours. These needs are being addressed by new SA techniques based on machine learning, such as learning-based software testing, invariant generation or code synthesis.

Machine learning is a powerful paradigm for SA that provides novel approaches to automating the generation of models and other essential artefacts. However, the ML and SA communities are traditionally separate, each with its own agenda. This Dagstuhl Seminar

Machine Learning for Dynamic Software Analysis: Potentials and Limits, *Dagstuhl Reports*, Vol. 6, Issue 4, pp. 161–173
Editors: Amel Bennaceur, Dimitra Giannakopoulou, Reiner Hähnle, and Karl Meinke

Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

brought together top researchers active in these two fields who can present the state of the art, and suggest new directions and collaborations for future research. We, the organisers, feel strongly that both communities have much to learn from each other, and the seminar focused strongly on fostering a spirit of collaboration.

The first day was dedicated to mutual education through a series of tutorials by leading researchers in both ML and SA to familiarise everyone with the terminology, research methodologies, and main approach of each community. The second day was dedicated to brainstorming and focused discussion in small groups, each of which supported by one of the organisers acting as a facilitator. At the end of the day a plenary session was held for each group to share a summary of their discussions. The participants also reflected and compared their findings. The morning of the third day was dedicated to the integration of the groups and further planning.

This report presents an overview of the talks given at the seminar and summaries of the discussions of the participants.

Acknowledgements. The organisers would like to express their gratitude to the participants and the Schloss Dagstuhl team for a productive and exciting seminar.

2 Table of Contents

3 Overview of Talks

3.1 Machine Learning for Emergent Middleware

Amel Bennaceur (The Open University – Milton Keynes, GB)

Highly dynamic and heterogeneous distributed systems are challenging today's middleware technologies. Existing middleware paradigms are unable to deliver on their most central promise, which is offering interoperability. In this talk, I argue for the need to dynamically synthesise distributed system infrastructures accord- ing to the current operating environment, thereby generating "Emergent Middleware" to mediate interactions among heterogeneous networked systems that interact in an ad hoc way. I will explain the overall architecture underlying Emergent Middleware, and in particular focuses on the key role of learning in supporting such a process, spanning statistical learning to infer the semantics of networked system functions and automata learning to extract the related behaviours of networked systems.

3.2 Learning Register Automata Models

Falk Howar (IPSSE – Goslar, DE)

Learning algorithms for register automata infer models with parameterized actions, symbolic guards, and memory. In this talk, we give a brief overview of how active automata learning has been extended over the past decade to infer such richer models. We focus on one line of work that is based on a generalized Myhill-Nerode theorem for data languages and the inference of symbolic decision trees from test cases. We present some key insights and open questions from this line of work and compare it to other approaches that tackle the same problem.

3.3 Static (Software) Analysis

Reiner Hähnle (TU Darmstadt, DE)

We give a brief introduction into the field of static analysis: definition, scope, techniques, challenges, trends. We also juxtapose static analysis of software with dynamic based on learning. We compare their relative strengths and weaknesses and try to work out where the opportunities for their possible combination lie.

3.4 Learning-based Testing: Recent Progress and Future Prospects

Karl Meinke (KTH Royal Institute of Technology – Stockholm, SE)

We present a survey of recent progress in the area of learning-based testing (LBT). The emphasis is primarily on fundamental concepts and theoretical principles, rather than applications and case studies. After surveying the basic principles and a concrete implementation of the approach, we describe recent directions in research such as: quantifying the hardness of learning problems, over-approximation methods for learning, and quantifying the power of model checker generated test cases. The common theme underlying these research directions is seen to be metrics for model convergence. Such metrics enable a precise, general and quantitative approach to both speed of learning and test coverage. Moreover, quantitative approaches to black-box test coverage serve to distinguish LBT from alternative approaches such as random and search-based testing. We conclude by outlining some prospects for future research.

3.5 Towards Automata Learning in Practice

Bernhard Steffen (TU Dortmund, DE)

In the last decade automata learning has attracted practical attention in software engineering e.g. as a means to lower the hurdle of so-called model-based testing, by overcoming the problem of the required a priori models, or as a way to mine runnable (legacy) software for behavioural specifications. In order to achieve true practicality, automata learning has 1) to increase scalability, 2) to move from its original theoretical roots, which focused on standard finite state machines to more expressive formalisms, and finally 3) to establish notions of quality assurance. Concerning 1) and 3), the TTT algorithm [1, 2] is promising, as it provides a scalable solution for dealing with extremely long counter examples, which are characteristic for the so-called life-long learning paradigm. Rather than making explicit quality statements, this paradigm establishes a continuous improvement cycle, and therefore an approach to quality assurance adequate, in particular, for agile software development. 2) has been addressed e.g. via extensions to Register Automata [3] and extended finite State Machines [4] (cf. the contributions of Falk Howar to the seminar). The Open-Sources Learnlib [5] aims at making all these algorithms available to the public.

References

1 Malte Isberner, Falk Howar, Bernhard Steffen: The TTT Algorithm: A Redundancy-Free Approach to Active Automata Learning. RV 2014:307–322
2 Malte Isberner Foundations of active Automata Learning: an algorithmic perspective (PhD thesis, Dortmund, 2015)
3 Malte Isberner, Falk Howar, Bernhard Steffen: Learning register automata: from languages to program structures. Machine Learning 96(1–2):65–98 (2014)
4 Sofia Cassel, Falk Howar, Bengt Jonsson, Bernhard Steffen: Active learning for extended finite state machines. Formal Asp. Comput. 28(2):233–263 (2016)
5 Malte Isberner, Falk Howar, Bernhard Steffen: The Open-Source LearnLib – A Framework for Active Automata Learning. CAV (1) 2015:487–495

3.6 Learning State Machines

Sicco Verwer (TU Delft, NL)

This tutorial is dedicated to learning complex state machines (including timing and parameters), using learning for more than just prediction (for instance model checking), and the power of search methods (using SAT-solvers and Mixed Integer Programming) in machine learning.

4 Working groups

4.1 Different Kinds of Models

Andreas Abel (Universität des Saarlandes – Saarbrücken, DE), Amel Bennaceur (The Open University – Milton Keynes, GB), Roland Groz (LIG – Grenoble, FR), Falk Howar (IPSSE – Goslar, DE), Frits Vaandrager (Radboud University Nijmegen, NL), Sicco Verwer (TU Delft, NL), and Neil Walkinshaw (University of Leicester, GB)

In this break-out session we discussed which models should we target for learning: what kind of models should we learn, and in particular what are the challenges for learning extended finite state models.

Since all participants in the group were dealing with state based models, we mostly discussed the issues related to this type of model, although we acknowledged that logic based models, in particular rule-based models, are just as worthy of interest for software engineering.

The type of mode may depend on the context in which the learning is done, the assumptions on what is available for learning (do we have negative as well as positive samples, do we have an oracle, can we reset or checkpoint a system ?) as well as the intended use of the models: is the model meant for testing, for static analysis, for reverse engineering or documenting etc.

Regarding the flavours of state models that can be learnt, we reckoned that many different kinds of models are already supported: Mealy machines and DFA are the most common, but we also have Moore machines, register automata, combination with rule based systems, hybrid automata. We could also consider LTS, IOTS, timed models. It is also interesting in some contexts to learn non-deterministic models of deterministic systems, and similarly to learn stochastic models. The main frontier is about what kind of extended state machine models could we learn, more powerful than register automata. Another so far not addressed issue is that of concurrency models, for which e.g. Petri Nets could be a target.

The major challenge for model learning in software analysis is about learning parametrised actions and associated data relations. We reckon that register automata are still too limited, we need richer models. At the same time, it is important to be able to identify relevant abstractions from data. Typically, being able to extract a parametrised model of actions with the relevant parameters from a traffic capture, without the pains of having to have a human expert doing it and writing the corresponding adaptor (aka mapper) is a key challenge. One direction can be to use statistical learning methods, such as PCA (Principal Component

Analysis). In this context, it is also important to consider methods that can lean in the presence of noisy data.

Other challenges have also been discussed: it is important to learn not just a system, but also a model of its environment. For a wide use of model learning, it is also important to be able to learn models that address non-functional characteristics of software such as performance or security. For a large number of applications, it is also important to learn understandable models, that can easily be interpreted. In this view, it may be better to learn simpler, more abstract models than more accurate ones that could be too complex.

Throughout the seminar, we considered a crucial issue: can learning end up with a consistent approximation: probably an over approximation, or maybe an under approximation? Most state based learning approaches produce models that are neither an over- nor an under-approximation. Rule based systems are more sensitive to the notion of monotonicity. But maybe the issue of having a consistent over-approximation is only useful in the context of verification, where we need a boolean answer. Another direction is to consider accuracy, as in the PAC concept (Probably Approximately Correct).

Finally, other issues with models have also been discussed. First, with most current tools, handling of timing issues, esp. events caused by a timeout, is done in ad hoc manner. Research is still need to adapt learning with a sound model for the passage of time. We also discussed the use of causality for identifying data relations. The idea is to use variations on some input parameters to check their influence on output parameters.

4.2 Combining White-Box and Glass-Box Analysis

Falk Howar (IPSSE – Goslar, DE), Andreas Abel (Universität des Saarlandes – Saarbrücken, DE), Pavol Bielik (ETH Zürich, CH), Radu Grosu (TU Wien, AT), Roland Groz (LIG – Grenoble, FR), Reiner Hähnle (TU Darmstadt, DE), Bengt Jonsson (Uppsala University, SE), Mohammad Reza Mousavi (Halmstad University, SE), Zvonimir Rakamaric (University of Utah – Salt Lake City, US), Alessandra Russo (Imperial College London, GB), Sicco Verwer (TU Delft, NL), and Andrzej Wasowski (IT University of Copenhagen, DK)

In this session, we explored ideas for combining black-box and analyses glass-box analyses in a meaningful way that would benefit either one analysis technique, or lead to a new and more powerful combined approach. The session identified three potential scenarios for the integration of approaches from both worlds, and discussed resulting practical challenges.

Learning for Environment Generation

The first idea was using learned models for verification: a model of a component's environment could be used to close the (open) component for verification. The main challenge in this scenario is providing quality information or correctness guarantees for models. Without such guarantees verification may not be sound, but it may still be able to find bugs.

Using Glass-Box Techniques in Learning

A second idea was using glass-box techniques in automata learning. There exist already some works that incorporate domain information generated by static analysis (e.g., for partial order reduction on the alphabet). Other glass-box techniques may be useful for as a basis for implementing equivalence queries or new kinds of queries that can speed up the learning process (e.g., that after some prefix a certain behaviour is never observable).

Integrating Learning for Glass-Box Analysis

The third idea was a tight integration of glass-box and black-box techniques: Static analysis could profit from dynamic analysis when it fails to produce good enough results. At the same time, the results produced by a learning algorithm could be improved by using static analysis to determine what to expose of a system during learning. The main challenge in this scenario is defining an interface for exchanging information between the two analysis techniques.

4.3 Machine learning for System Composition

Falk Howar (IPSSE – Goslar, DE), Amel Bennaceur (The Open University – Milton Keynes, GB), Bengt Jonsson (Uppsala University, SE), Alessandra Russo (Imperial College London, GB), Sicco Verwer (TU Delft, NL), and Andrzej Wasowski (IT University of Copenhagen, DK)

The topic of machine learning for system composition was discussed in this break-out session. Machine Learning approach, in this context, is primarily intended to to be automata learning. The system composition problem is therefore how to combine together automata that describe behaviours of components of a system. Two possible approaches were identified. On one hand, the union of the languages of the different components can be considered and the automata of the entire system can be computed using automata learning mechanisms. But clearly this may have scalability problems. A second approach would be to start from individual automata of each component and then learn mediator models that allow the composition of the individual automata. Learning automata can in this case be used for both learning individual automata as well as learning mediator's automata. On the other hand, the area of learning state machine has also seen work in the context of logic-based learning, where the objectives is to learn temporal specification that can be translated into labelled transition systems that cover given positive traces and do not cover negative traces. Interesting question is there whether there is a potential synergy between logic-based learning and learning automata that could lead to novel mechanisms and/or improvements of state machine learning processes. Main features of logic-based learning include knowledge about the problem in hand. For instance they can include general knowledge about state merge; they take in input labelled positive and negative traces; and they are capable of learning linear temporal logic descriptions that, when translated into labelled transition system behaviour models, are guaranteed to accept the positive traces and reject all the negative traces. Finally they learn within the search space of a defined language bias, which can declaratively constraints with domain-specific expert knowledge. So, one of the open questions is how the two approaches relate with each

other. Are there similarities between learning automata and logic-based learning that can be exploited to allow synergisms between the two types of ML approaches in the context of software analysis. It has been pointed out that earlier work exists on how to translate State Machine language into first-order (FO) logic knowledge representation language IDP. This FO formula gets grounded to a propositional satisfiability, which is solved using a SAT solver. The LBL approach is similar but aims to find an FO formula directly, which gets translated into a labeled transition system after learning takes place. So, a very promising direction for research is to combine these methods, adding the capability of including expert-knowledge as constraints to the state machine learning process. A synergy between logic-based learning and learning automata may also lead to the development of distributed learning algorithms that allow models of system components to be learned collaboratively in a manner that guarantee properties of the composition expressed as constraints in the expert knowledge of the learner of each component. For the learning of mediator models, it would be interesting to try to use logic-based learning to infer the mediator (glue code) between existing components. In this case, the descriptions of the components represent a model of the environment while the desirable properties of the interactions represents the goal model from which a declarative specification that gives the mediator automata could be learned. We could then compare/contrast the result with that obtained by using automata learning.

4.4 Combination of Static Analysis and Learning

Bernhard Steffen (TU Dortmund, DE), Pavol Bielik (ETH Zürich, CH), Radu Grosu (TU Wien, AT), Reiner Hähnle (TU Darmstadt, DE), Bengt Jonsson (Uppsala University, SE), Mohammad Reza Mousavi (Halmstad University, SE), Daniel Neider (Los Angeles, US), and Zvonimir Rakamaric (University of Utah – Salt Lake City, US)

In particular, because of the previous discussion in the Discussion Group on Black Box and White Box Methods, the discussion here focused on two topics:

1. How to combine concolic execution and learning. As a success [1] was mentioned which uses the (concrete) access sequences of a learned hypothesis as a means to provide concrete values to drive the concolic/symbolic execution by inserting fitting concrete values into the symbolic execution process. Essentially this means that one uses automata learning as a supportive oracle for providing adequate concrete values for concolic execution. On the other side, the concolic or symbolic execution may support the test-based search for counter examples (the typical practical realization of the so-called equivalence queries) by reducing the search space. A more general underlying question here is how to synchronize the two worlds, meaning how symbolic states that arise during symbolic execution can be related to the states of a learned hypothesis. This relation hinges on a common notational understanding in terms of an adequate abstraction level. Such a level may be revealed via the search for e.g. key variables (e.g. those that strongly influence the control flow) using 'classical' static analysis techniques (slicing etc.).
2. In contrast to static analysis and typical other white box techniques, learning does not provide any guarantees like over approximation or under approximation. We discussed the question of whether it might be possible use static analysis to guarantee over-approximation during the learning process. It seems that reducing the chain of hypothesis

to only contain over approximations almost inevitably breaks termination in general.
Some compare automata learning with polynomial interpolation, as it like automata learning:

- Provides precise answers if the function to be approximated is indeed a polynomial of degree n, the degree of the current interpolation (just replace degree by number of states).
- There is in general no way to guarantee that the polynomial is an over or upper approximation of the target function.

The way proposed during the discussion to replace all the parts that are not really known by 'chaos' (i.e. the process which can do whatever it likes), reminds me of setting all function value outside the supported set to infinity, which certainly guarantees over approximation, but, in a way, destroys the charm of the interpolation idea, which consists of leveraging finite information to obtain something infinite (unfortunately at the cost of the abovementioned guarantees). However, like in polynomial interpolation, one may think of metrics and ways of error approximation (here in terms of probabilities). It still has to be seen, how practical such approaches might be.

An alternative approach could be to not guarantee over approximation continuously, but only on demand, and this approach could well profit from static analysis, showing that all the abstractly feasible paths are indeed covered by the hypothesis.

References

1 CY Cho, D Babic, P Poosankam, KZ Chen, EXJ Wu, D Song *MACE: Model-inference-Assisted Concolic Exploration for Protocol and Vulnerability Discovery*. USENIX Security Symposium, pp. 139–154. 2011

4.5 Benchmark Building and Sharing

Frits Vaandrager (Radboud University Nijmegen, NL), Dalal Alrajeh (Imperial College London, GB), Amel Bennaceur (The Open University – Milton Keynes, GB), Roland Groz (LIG – Grenoble, FR), Karl Meinke (KTH Royal Institute of Technology – Stockholm, SE), Daniel Neider (Los Angeles, US), Bernhard Steffen (TU Dortmund, DE), and Neil Walkinshaw (University of Leicester, GB)

We had an interesting discussion on the use of benchmarks for automata learning and testing. Radboud University has set up a repository for Mealy machines and register automata. Frits Vaandrager encouraged everybody to submit benchmarks. Extension to other classes of models such as Moore machines, DFAs,.. are encouraged. All participants agreed that benchmarks are important and useful. Benchmarks measure the relative performance of different approaches/systems and they are important to check whether tools and methods advance and whether new methods are effective. Systematic use of benchmarks is a sign of maturity of a scientific field. Several participants offered to contribute benchmarks to the Radboud University repository. The discussion made it clear that (a) we have different types of benchmarks, and (b) many different criteria for evaluation of algorithms and tools.

Different types of benchmarks:

- Benchmarks for evaluating efficiency of algorithms and tools
- Challenges for pushing the state-of-the-art (e.g. RERS challenges)
- Benchmarks for illustrating usefulness of a method or tool.

Criteria for evaluation of algorithms and tools:

- Does the tool aim at fully learning the benchmark or just at giving suitable aggregate information of data that have been gathered. E.g. there are tools for invariant generation.
- Number of inputs events
- Number of test sequences
- Wall clock time needed for learning (reset or certain inputs may require lot of time)
- Quality of intermediate hypothesis; how long it takes before you get first reasonable model
- How interpretable are the results (e.g. by discovering hierarchy and parallel composition)
- How easy it is to parallelize learning

We agreed to elaborate this list; when people come with new tools/algorithms they should make it clear, using the benchmarks, at which points they are improving the state-of-the-art We also discussed formats for the benchmarks: For Mealy machines dot files appear to be ok For register automata, Fides Aarts and Falk Howar have proposed a format. Since in the area of EFSMs things have not stabilized yet standardization may be A mature field is characterized by the presence of two other types of evaluation: Substantial (industrial) case studies are important to check whether tools/methods are relevant for practical problems and do sufficiently scale. Experimental usability studies are needed to find out whether a new method/tool can be integrated into practical workflow with advantage.

4.6 Learning and Testing

Neil Walkinshaw (University of Leicester, GB), Andreas Abel (Universität des Saarlandes – Saarbrücken, DE), Dalal Alrajeh (Imperial College London, GB), Pavol Bielik (ETH Zürich, CH), Roland Groz (LIG – Grenoble, FR), Reiner Hähnle (TU Darmstadt, DE), Karl Meinke (KTH Royal Institute of Technology – Stockholm, SE), Mohammad Reza Mousavi (Halmstad University, SE), Daniel Neider (Los Angeles, US), Zvonimir Rakamaric (University of Utah – Salt Lake City, US), Bernhard Steffen (TU Dortmund, DE), and Frits Vaandrager (Radboud University Nijmegen, NL)

We had an active discussion on the combination of testing and inference. The discussion was structured according to SWOT (Strengths, Weaknesses, Opportunities and Threats), to provide an idea of the state-of-the-art, and where things could go from here. These are summarised individually in the following paragraphs.

Strengths: Automation is a key strength, addressing the key weakness with conventional Model Based Testing (the need to invest effort into producing a model). The automation is especially pronounced when the models that are inferred are associated with established testing algorithms (as is the case for Finite State Machines). Ultimately, the approach does require a degree of manual intervention (e.g. to validate test outputs), but the point was made that another strength of the approach is that this actually dovetails nicely with iterative, agile techniques. If the underlying development process is iterative, inference-based testing can be used to automatically generate test inputs, whilst the validation can occur in its normal setting. There was also the observation that current approaches that combine testing

and inference tend to extract value from established test sets – the existing tests provide the basis for the (initial) model inference.

Weaknesses: Current efforts to combine model inference and testing still suffer from numerous weaknesses, presenting a fertile basis for future research. They do not tend to scale well, often requiring an unrealistic number of test executions or other forms of user input. The currently inferred models tend to have limited expressivity (being predominantly FSMs). In settings where there are different potential Machine Learners, the task of choosing a given learner is often still a matter of intuition. Then there is also the fact that current efforts at empirical evaluation are often very limited; they are rarely compared against random testing, rarely involve genuine / seeded faults, often focus on small-scale systems, and are often based upon questionable metrics of accuracy and efficiency. There is also the broader question of what it means when a test-run has "finished" – there has been little discussion of how much assurance can be derived from this. Finally, there is the task of implementation; the challenge of abridging what are often complex Machine Learning systems with a fully fledged testing engine is challenging, and tools can accordingly rarely be easily deployed.

Opportunities: There is an extensive interest from industry; the problems addressed by the combination of testing and inference are timely. There are also several avenues by which additional knowledge can be embedded into the process (e.g. domain knowledge to prevent inference mistakes), which could easily address some of the aforementioned weaknesses. Product-lines offer an interesting, more controlled environment within which to develop test/inference systems. If the approaches are used as the basis for regression testing, this can remove a lot of the manual effort required for validating test outputs (because this can be checked against previous versions instead).

Threats: Happily, the room could only think of few threats. The top one was that, although industry is generally enthusiastic about the idea of automated testing and the opportunities that ML brings, there are often unrealistic expectations. This can lead to frustration, e.g. when it comes to the realisation that abstractions have to be generated.

Participants

- Andreas Abel
Universität des Saarlandes – Saarbrücken, DE
- Dalal Alrajeh
Imperial College London, GB
- Amel Bennaceur
The Open University - Milton Keynes, GB
- Pavol Bielik
ETH Zürich, CH
- Radu Grosu
TU Wien, AT
- Roland Groz
LIG – Grenoble, FR
- Reiner Hähnle
TU Darmstadt, DE
- Falk Howar
IPSSE – Goslar, DE
- Bengt Jonsson
Uppsala University, SE
- Karl Meinke
KTH Royal Institute of Technology – Stockholm, SE
- Mohammad Reza Mousavi
Halmstad University, SE
- Daniel Neider
Los Angeles, US
- Zvonimir Rakamaric
University of Utah – Salt Lake City, US
- Alessandra Russo
Imperial College London, GB
- Bernhard Steffen
TU Dortmund, DE
- Frits Vaandrager
Radboud Univ. Nijmegen, NL
- Sicco Verwer
TU Delft, NL
- Neil Walkinshaw
University of Leicester, GB
- Andrzej Wasowski
IT Univ. of Copenhagen, DK