Report from Dagstuhl Seminar 17291

Resource Bound Analysis

Edited by

Marco Gaboardi[1], Jan Hoffmann[2], Reinhard Wilhelm[3], and Florian Zuleger[4]

1 **University at Buffalo, US,** gaboardi@buffalo.edu
2 **Carnegie Mellon University – Pittsburgh, US,** jhoffmann@cmu.edu
3 **Universität des Saarlandes, Saarland Informatics Campus, DE,** wilhelm@cs.uni-saarland.de
4 **TU Wien, AT,** zuleger@forsyte.at

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 17291 "Resource Bound Analysis". Resource-bound analysis is studied in formal methods and programming languages at different levels of abstraction. The goal of the Dagstuhl seminar was to bring together leading researchers with different backgrounds in resource-bound analysis to address challenging open problems and to facilitate communication across research areas.

Seminar July 16–21, 2017 – http://www.dagstuhl.de/17291
1998 ACM Subject Classification C.3 [Special-Purpose and Application-Based Systems]: Real-Time and Embedded Systems, F.3.2 [Semantics of programming languages]: Program Analysis
Keywords and phrases quantitative analysis, resource-bound analysis, WCET
Digital Object Identifier 10.4230/DagRep.7.7.72

1 Executive Summary

Marco Gaboardi
Jan Hoffmann
Reinhard Wilhelm
Florian Zuleger

This seminar is dedicated to our friend and colleague Martin Hofmann (1965-2018). Martin's vision and ideas have shaped our community and the way resource analysis is performed and thought about. We are grateful for the time we spent with him and we will sorely miss his ingenuity, kindness, and enthusiasm.

There are great research opportunities in combining the three aforementioned approaches to resource bound analysis. The goal of the Dagstuhl seminar was to bring together leading researchers with different backgrounds in these three areas to address challenging open problems and to facilitate communication across research areas.

To this end, the program included seven tutorials on state-of-the-art techniques in the different communities, and short talks on concrete topics with potential for cross–fertilization. This included combining WCET analysis with higher-level bound analysis techniques, hardware-specific refinement of high-level cost models, and interaction of resource analysis with compilation. Additionally, the seminar included two tools sessions: the first

was a presentation of the aiT tool of AbsInt by Simon Wegener; the second was a session with presentations of different tools from different participants. Finally, the seminar included a discussion on open problems in the different areas as well as open problems for cross-fertilization.

The tutorials, the talks solicited from the participants, and the tool and discussion sessions allowed us to identify topics which are of common interest to the three different communities. Some of these topics are

- invariant and flow analysis,
- constraint solving and
- formalisms and logics for resource bounds.

Supporting information about program invariants and the possible control flow are often required by a resource analysis, e.g., the maximal value of a loop counter, or the infeasibility of a program path. The actual resource analysis is often reduced to solving a constraint system, e.g., using techniques from linear programming or recurrence equations. Verification logics for resource bounds as well as programming language formalisms are of common interest as they allow to specify or to guarantee that a program satisfies a required worst case resource bound.

We believe that the further study of these topics promises to increase the connections and to leverage the synergies between the different communities.

2 Table of Contents

3 Background

Marco Gaboardi (University at Buffalo, US), Jan Hoffmann (Carnegie Mellon University – Pittsburgh, US), Reinhard Wilhelm (Universität des Saarlandes, DE), and Florian Zuleger (TU Wien, AT)

Resource-bound analysis is studied in formal methods and programming languages at different levels of abstraction. This ranges from concrete clock-cycle bounds on specific hardware (WCET analysis), to high-level symbolic bound analysis (recurrence relations, type systems, abstract interpretation, term rewriting), to logical characterizations of asymptotic complexity (linear logic, type systems, semantics). These are active areas of research and there has been significant progress in all of them over the past decade. However, these problems are often studied by different communities with little overlap. Methods close to the implementation level base their approaches on concrete hardware models and derive precise execution-time bounds, vulgo WCETs, in number of clock-cycles, and bounds on the extension of stack and heap-space. WCET analysis is a success story of formal methods and is meanwhile applied in industry. WCET methods usually work on the machine code level, often lack compositionality and are restricted to programs without complex control flow. Techniques used in WCET analysis include Abstract Interpretation, ILP solving, and Model Checking.

The field of *automatic symbolic resource-bound analysis* studies methods for automatically deriving bounds that are functions of the inputs of a program. The bounds are usually non-asymptotic and many techniques can automatically handle complex data structures and control flow. However, the cost models that are used are often simplistic and ignore the characteristics of modern high-performance architectures. The used techniques include abstract interpretation, recurrence relations, type systems, LP solving, and SMT solving.

Finally, there is an active community that studies *theoretical foundations of resource analysis*. This includes derivation of asymptotic bounds, mechanization and verification of bounds, completeness results, relational resource analysis, formal cost semantics, and soundness proofs. The studied techniques are often very powerful and general. However, the used cost models are fairly abstract and the used bound-analysis techniques are hard to automate. The used techniques include linear logic, affine and dependent type systems, proof assistants, and operational semantics.

4 Overview of Tutorials

4.1 Tutorial: WCET Analysis: A Primer

Jan Reineke (Universität des Saarlandes, DE)

Worst-case execution time (WCET) analysis is concerned with computing an upper bound on the execution time of a program on a particular hardware platform.

After discussing the three main factors that influence the execution time of a program, 1. the program's inputs, 2. the state of the HW platform, and 3. interference from the environment, I discuss the structure of modern WCET analysis tools for single-core

architectures. Such tools essentially separately solve subproblems concerning (a) the possible paths through the program, and (b) the possible "microarchitectural" paths. The solutions to these subproblems can then be combined in an integer linear programming formulation to obtain an upper bound on the WCET.

4.2 Tutorial: Control Flow Analysis for WCET Analysis

Björn Lisper (Mälardalen University – Västerås, SE)

An important part of WCET analysis is the Control Flow Analysis (CFA), which in this context amounts to finding constraints on the program flow such as loop iteration bounds, and infeasible path constraints. We review the problem in the IPET setting, where the program flow constraints are expressed as linear inequalities on execution counters for basic blocks. We discuss the relation to CFA for functional programs. We then show some existing methods for CFA for WCET, and we discuss their respective pros and cons.

4.3 Tutorial: Automatic Amortized Analysis

Martin Hofmann (LMU München, DE)

Amortized complexity was introduced by Tarjan to facilitate the runtime analysis of data structures that sometimes perform costly operations which pay off later. Typical examples include self-organising binary search trees and also many functional data structures. Initially motivated by ideas from implicit computational complexity (ICC), amortized complexity has been harnessed for the automatic analysis of runtime and other resources. The crucial idea is that with an appropriate choice of a potential function the amortized complexity of basic operations becomes constant in many cases which avoids the need for tracking sizes and shapes of data structures during the analysis. Furthermore, the compositionality of amortized complexity makes it suitable for integration with type systems. This research has led to several systems for the automatic analysis of functional and object-oriented programs. There have also been applications to resource usage certification, to the analysis of low-level systems code and to lazy functional programming and to term rewriting. The talk gives a gentle introduction to this topic beginning from its sources in implicit computational complexity and closes with recent developments and some open questions.

4.4 Tutorial: Cost Analysis with Recurrence Relations (using CiaoPP) and its Applications

Manuel Hermenegildo (IMDEA Software – Madrid, ES)

We present in a tutorial fashion, how to perform cost analyses for a wide class of resources by setting up and solving recurrence relations on program data sizes and procedure costs. We follow the method we have proposed and developed for analysis of programs in Horn clause form and its implementation in the CiaoPP system, but the discussion is applicable in general to recurrence relation-based models.

We start by discussing a number of interesting applications, from our original motivation of task granularity control in automatic program parallelization to others such as resource verification, security, or static profiling, demonstrating them through examples run on the CiaoPP system. Based on the characteristics of the problem and the demands of these applications, we motivate the objective of obtaining upper and lower bounds, the requirement that these bounds be functions of program input sizes, and the need to infer both data size and procedure cost functions.

We continue by discussing how a good intermediate representation (in our case, Horn clauses) allows supporting multiple languages in a uniform way, showing several transformations from different languages into this intermediate form. We also present the concept of user-definable resources, via assertions, and demonstrate it through a number of simple examples run on the CiaoPP system. We also show how this concept is instrumental in supporting multiple languages.

We then present the method for performing cost analyses by setting up and solving recurrence relations on the program data sizes and procedures. We illustrate the method though a detailed worked example, deriving step by step the cost relations and the closed forms for both the sizes and the procedure cost. We also discuss a number of issues related to solving recurrence equations. We show through an example how to set up non-deterministic recurrence equations for inferring balanced costs (e.g., for obtaining quadratic bounds for quick-sort). To conclude we present the notions of accumulated cost and static profiling and also illustrate them with an example.

4.5 Tutorial: From Implicit Complexity to Resource Bound Analysis

Ugo Dal Lago (University of Bologna, IT)

Implicit computational complexity aims at giving precise, but machine-free, characterisations of complexity classes, like polynomial time or polynomial space computable functions. Many of the proposed systems can be turned into verification methodologies providing resource bounds on the input program. The intrinsically poor expressive power of these methodologies has been sometime overcome by making the underlying logic less implicit but more informative. We focus our attention on systems derived from Girard's linear logic.

4.6 Tutorial: Complexity Analysis of Term Rewrite System

Martin Avanzini (Universität Innsbruck, AT)

Over the last decade, the rewriting community has introduced various novel techniques for the runtime complexity analysis of term rewrite systems. In particular, this research has also led to various tools in this context.

In this talk, I give a general overview on how the rewriting community tackles the problem.

4.7 Tutorial: RTDroid: Toward dynamic real-time systems

Lukasz Ziarek (University at Buffalo, US)

Time predictability is a requirement for computer systems that have deadlines to meet, but it is frustratingly difficult to achieve in the complex, layered, execution environments that are common place today. This talk will consider how to bring a degree of time predictability to Android applications.

Potential solutions include fundamental changes to the Android framework and the introduction of a new programing model, which focuses on the interplay between real-time activities and the rest of the system. This talk will detail the changes in the Android APIs which are required for developers to express the timeliness requirements of code and how well those requirements can be met on stock hardware in the presence of multiple, potentially interacting applications.

The talk will also cover some experimental data validating feasibility over several applications including UAV fight control, implantable medical devices, as well as a wind turbine monitoring device. Lastly, I will discuss future directions, including adding adaptivity to the system to achieve a dynamically defined real-time system.

5 Tool demo session report

During the seminar some time was dedicated to the presentation of tools for resource bound analysis. Participants of 7 tools presented during a tool demo session on Thursday afternoon. Each presenter had 15 minutes to present their tool. The list of presented tools and presenters is the following.

- GenE (Peter Wegemann)
- OTAWA (Hugues Cassé)
- SWEET (Björn Lisper)
- Absynth (Chan Ngo)
- RAML (Ankush Das)
- LazyAA (Pedro Vasconcelos)
- AProVE (Jürgen Giesl)
- CiaoPP (Manuel Hermenegildo)

6 Overview of other Talks

6.1 Parametric Timing Analysis

Sebastian Altmeyer (University of Amsterdam, NL)

Parametric timing analysis provides computes symbolic WCET estimates instead of numeric bounds. In this talk, I will present which analyses are needed to extend the standard numeric WCET analysis and how these analyses work. The symbolic path analysis is of particular interest, and I will provide two variants of it.

6.2 Amortised Resource Analysis with Separation Logic

Robert Atkey (University of Strathclyde – Glasgow, GB)

Type-based amortised resource analysis following Hofmann and Jost–where resources are associated with individual elements of data structures and doled out to the programmer under a linear typing discipline–have been successful in providing concrete resource bounds for functional programs, with good support for inference. In this work we translate the idea of amortised resource analysis to imperative pointer-manipulating languages by embedding a logic of resources, based on the affine intuitionistic Logic of Bunched Implications, within Separation Logic. The Separation Logic component allows us to assert the presence and shape of mutable data structures on the heap, while the resource component allows us to state the consumable resources associated with each member of the structure.

We present the logic on a small imperative language, based on Java bytecode, with procedures and mutable heap. We have formalised the logic and its soundness property within the Coq proof assistant and extracted a certified verification condition generator. We also describe an proof search procedure that allows generated verification conditions to be discharged while using linear programming to infer consumable resource annotations.

We demonstrate the logic on some examples, including proving the termination of in-place list reversal on lists with cyclic tails.

6.3 Resource Analysis of Session Typed Programs

Ankush Das (Carnegie Mellon University – Pittsburgh, US)

In this work, we aim to measure the work done by session typed programs. For that, we define a potential based type system where the types provide an upper bound on the work performed by the processes. I will introduce and motivate session types, define a cost semantics to measure work, and design the type system. Finally, I will describe the soundness of our system, and conclude with an example and remarks on future directions.

6.4 A proof system for relational cost analysis

Deepak Garg (MPI-SWS – Saarbrücken, DE)

Formal frameworks for cost analysis of programs have been widely studied in the unary setting and, to a limited extent, in the relational setting. However, many of these frameworks focus only on the cost aspect, largely side-lining functional properties and value-sensitivity that are often required for cost analysis, thus leaving many interesting programs out of their purview. This talk shows how a simple, expressive proof system for costs (unary and relational) can be built using basic ingredients: a cost monad and higher-order refinements. The result is highly expressive. Besides several new examples, it can be used as a meta framework for embedding existing formal systems for cost analyses.

6.5 Compositional timing control in HLS

Dan R. Ghica (University of Birmingham, GB)

We examine how type systems are used to control space and time constraints in high-level synthesis. Syntactic Control of Interference, an affine restriction of Idealise Algol allows execution in constant space whereas a bounded-linear logic-like type system controls timings. However, this type system gives impractical timing constraints because of excessive required precision. We examine an alternative, simpler, notion of timing in type which is non-deductive, i.e. established directly from examining the semantic model, yet compositional.

6.6 Complexity for Java with AProVE

Jürgen Giesl (RWTH Aachen, DE)

While AProVE is one of the most powerful tools for termination analysis of Java since many years, we now extend our approach in order to analyze the complexity of Java programs as well. Based on a symbolic execution of the program, we develop a novel transformation of (possibly heap-manipulating) Java programs to integer transition systems (ITSs). This allows us to use existing complexity analyzers for ITSs to infer runtime bounds for Java programs. We demonstrate the power of our implementation on an established standard benchmark set.

6.7 Enabling Compositionality for (Multi-Core) Execution Time Analysis

Sebastian Hahn (Universität des Saarlandes, DE)

The property of compositionality links the low-level (WCET) analysis, which computes characteristics of single tasks run in isolation, with schedulability analysis, which accounts for interference on shared resources caused by other tasks. In this talk, we explain the meaning of this compositionality property, demonstrate that it does not come for free even on simple microarchitectures, and show ways how to enable compositionality.

6.8 Energy Consumption Analysis and Verification

Manuel Hermenegildo (IMDEA Software – Madrid, ES)

We present our overall approach to the inference and verification of upper- and lower-bounds on the energy consumption of programs, as well as some results from our tools. We translate low-level program representations into a block-based intermediate form, expressed as Horn clauses, and compute abstract minimal models of such Horn clauses on abstract domains that include resource functions on data intervals and sized shapes for structured data. The computed abstract models include, for each procedure, and for each possible abstract call state and path to it, functions that return bounds on the corresponding energy consumed for any given data size, as well as the contribution of each procedure to the overall consumption (static profiling). These analysis results are compared with energy specifications for program verification or (performance) error detection. We will present results for the energy analysis of embedded programs on the XS1-L architecture, making use of ISA- and LLVM-level models of the cost of instructions or sequences of instructions, and compare them to the actual energy consumption measured on the hardware.

6.9 Generating Invariants and Resource Bounds with Recurrence Analysis

Zachary Kincaid (Princeton University, USA)

Compositional recurrence analysis (CRA) is an invariant generation technique that approximates the transitive closure of loops by extracting recurrence relations from the loop body and computing their closed forms. In this talk I will give an overview of CRA and show how it can be used to solve resource bound problems. I will focus particularly on symbolic methods for computing recurrence equations and inequations from a logical representation of a loop body's behavior.

6.10 Type systems for Energy Management

Yu David Liu (SUNY Binghamton, USA)

This talk attempts to bridge two largely disjoint areas of research – type system design and energy management – through typed programming language design for energy-efficient and energy-aware software development. The first part of the talk will focus on Energy Types, a static type system to enable phase-based and mode-based energy management. The second part of the talk will discuss the challenges in promoting proactive and adaptive energy management at the same time, and describe a new programming language called Ent with mixed static type checking and dynamic type checking for addressing these challenges. An open-source compiler built on the ideas of Energy Types and Ent has been implemented, and ported to x86 machines, Android phones, and Raspberry Pi.

6.11 Weighted Automata Theory for Resource Analysis of Rewrite Systems

Georg Moser (Universität Innsbruck, AT)

Traditionally derivational and runtime complexity analysis for term rewrite systems (TRSs for short) is performed as restriction of termination methods for TRSs. One such technique is matrix interpretations, which is surprisingly versatile, but maybe costly. In the talk I presented the use of weighted automata theory (aka joint spectral radius theory) to bound the growth rate of matrix interpretations, which in turn yields sharp bounds on the complexity of TRSs.

6.12 Solving Recurrences Using Operational Calculus

Thomas W. Reps (University of Wisconsin – Madison, US)

Joint work of Thomas W. Reps, John Cyphert, Jason Breck, Zak Kincaid

This talk describes a tool/technique that is used to solve recurrence equations in the ICRA tool for finding program invariants in non-linear arithmetic. It is based on a variant of Mikusinski's operational calculus. The recurrences are transformed into equations in a field of sequences; the to-be-solved-for sequence is isolated by algebraic manipulations; and the result is transformed back to ordinary algebra. The technique is also applied to solve multivariate recurrences of the kind that arise in a loop that transforms multiple program variables on each iteration.

6.13 ECA: Energy Consumption Analysis of software controlled systems

Marko van Eekelen (University of Nijmegen, NL)

Energy consumption analysis of software controlled systems can play a major role in minimising the overall energy consumption of such systems both during the development phase and later in the field. ECA proposes such an energy analysis, analysing both software and hardware together, to derive the energy consumption of the system when executing. The energy analysis has the property of being adjustable both to the required precision and concerning the hardware used. In principle, this creates the opportunity to analyse which is the best software implementation for given hardware, or the other way around: choose the best hardware for a given algorithm.

The precise analysis is introduced for a high level language, that covers the essentials for control systems. Hardware is modelled as a finite state machine, in which the transitions are function calls that are made explicit in the source code. In these model state changes correspond to energy consumption level. For that reason timing is added to the finite state machines. All the transitions and states in the hardware models are annotated with energy consumptions, to account both for time-dependent and for incidental energy consumptions. A prototype of the ECA system has been developed. It is applied to a small case study.

7 Panel discussion and Open problems

During the seminar we had a panel discussion and open problems session on resource bound analysis. Here is our summary of this session with contributions by Jan Reineke and Tom Reps.

Efficiently computed low and precise time bounds

The most fundamental open problem in the Timing-Analysis area connects the performance of architectures with the efficiency of timing-analysis methods and the precision of their results: how to design architectures that allow the efficient determination of precise and low execution-time bounds.

Multi-core architectures

While the timing-analysis problem for single-core architectures is solved, there is currently no practically usable timing-analysis method for multi-core processors.

Non-standard architectures

A completely open problem is how to determine safe execution-time bounds for GPUs, which are heavily used in computer-vision systems in cars, as well as in other types of accelerators. Similar challenges are given from heterogeneuos systems including different kinds of architectures, e.g. mobile phones, wearable devices, data centers, etc.

Other language paradigms

How can one determine bounds on low-level execution-time and energy consumption for other types of languages, e.g. logic and functional languages and even higher order functional languages?

Amortized analysis for WCET

Is it useful to do amortized analyses in the determination of execution-time bounds, in particular for distributed systems, i.e. including communication?

Side channels

Modern execution platforms feature a multitude of shared resources, ranging from caches, branch predictors, and DRAM banks to shared functional units in case of hyperthreading. The execution of a program may leave traces on these resources, which side-channel attacks use to infer secrets, such as passwords or private keys. Techniques from resource-bound analysis may be used to quantify the amount of information that a program leaks through such side channels.

Memory management

Most modern languages have support for dynamic memory management. Dynamic memory management, however, is at odds with time predictability. While there have been efforts to build dynamic memory allocators and garbage collectors with bounded response times, almost all approaches ignore the memory hierarchy and in particular caches. This makes them unsuitable for hard real-time applications, where it is crucial to precisely account for the cache behavior of an application. It is an open problem to build a fully timing-predictable garbage collector taking into account microarchitectural effects.

Legacy software

Legacy software has for the most part been written to be executed on single-core platforms. Transitioning to multi-core platforms frequently uncovers race conditions between different software components. How can such legacy software be retrofitted to enable its predictable use on multi cores?

Low-level vs high-level analyses

Techniques for WCET traditionally are closer to the architecture, while techniques for high-level languages use abstract cost models. An open research question is how to integrate these two models. Specifically, how can we link the low-level cost to the high-level cost? Can we design models combining both low-level components and high-level analysis?

Types as interfaces and specifications

Several techniques for resource analysis for higher-level languages are based on types. Types can be seen as interfaces abstracting the behaviors of the different components but also as specifications for the components. Can we use types as interfaces and as specifications also for resource bounds? Can we design type-based techniques to combine precise resource bounds of different components whose resource analysis has been performed in isolation?

Can we use these techniques also to account for the cost of communications between the different components?

Resource analysis for effects

Pure functional languages are ameanable to simple resource bound analysis. Impure aspects of computations, like memory, I/O, communications, etc. are often encapsulated as general effects. These effects can change the control flow of the program, and add impredictability to the program beahavior. Providing precise resource bound analysis in presence of effects is a challenge.

Scalable benchmarks

Benchmarks are important to evaluate and compare different resource bound analyses. Current benchmark suites include several important examples that are however often designed for the small scale of one machine. Can we also scale them to modern needs of multi-core architectures, heterogeneous computations, cloud computing, etc ?

Optimizing Compilers

Can we create compilers/optimizers that can explain their results (including whether a given spec. was satisfied)?

Performance analysis and Performance Debugging

Many performance problems are only found after a software has been released. Profilers are the most common technique to understand performance problems. An open problem is how techniques from resource bound analysis can help: Can static techniques be used as an alternative to dynamic profiling techniques? Can resource bound analysis be used to find performance bugs early on during the development process?

Participants

- Sebastian Altmeyer
University of Amsterdam, NL
- Robert Atkey
University of Strathclyde – Glasgow, GB
- Martin Avanzini
Universität Innsbruck, AT
- Gilles Barthe
IMDEA Software – Madrid, ES
- Marc Brockschmidt
Microsoft Research UK – Cambridge, GB
- Hugues Cassé
University of Toulouse, FR
- Stephen Chong
Harvard University – Cambridge, US
- Ezgi Cicek
MPI-SWS – Saarbrücken, DE
- Ugo Dal Lago
University of Bologna, IT
- Norman Danner
Wesleyan Univ. – Middletown, US
- Ankush Das
Carnegie Mellon University – Pittsburgh, US
- Marco Gaboardi
University at Buffalo, US
- Deepak Garg
MPI-SWS – Saarbrücken, DE
- Samir Genaim
Complutense University of Madrid, ES
- Dan R. Ghica
University of Birmingham, GB
- Jürgen Giesl
RWTH Aachen, DE
- Reiner Hähnle
TU Darmstadt, DE
- Sebastian Hahn
Universität des Saarlandes, DE
- Kevin Hammond
University of St. Andrews, GB
- Manuel Hermenegildo
IMDEA Software – Madrid, ES
- Jan Hoffmann
Carnegie Mellon University – Pittsburgh, US
- Martin Hofmann
LMU München, DE
- Steffen Jost
LMU München, DE
- Zachary Kincaid
Princeton University, US
- Jens Knoop
TU Wien, AT
- Björn Lisper
Mälardalen University – Västerås, SE
- Yu David Liu
Binghamton University, US
- Alexey Loginov
GrammaTech Inc. – Ithaca, US
- Hans-Wolfgang Loidl
Heriot-Watt University – Edinburgh, GB
- Antonio Flores Montoya
TU Darmstadt, DE
- Georg Moser
Universität Innsbruck, AT
- Van Chan Ngo
Carnegie Mellon University – Pittsburgh, US
- Jan Reineke
Universität des Saarlandes, DE
- Thomas W. Reps
University of Wisconsin – Madison, US
- Christine Rochange
University of Toulouse, FR
- Claudio Sacerdoti Coen
University of Bologna, IT
- Marko van Eekelen
University of Nijmegen, NL
- Pedro B. Vasconcelos
University of Porto, PT
- Marcus Völp
University of Luxembourg, LU
- Peter Wägemann
Universität Erlangen-Nürnberg, DE
- Reinhard Wilhelm
Universität des Saarlandes, DE
- Lukasz Ziarek
University at Buffalo, US
- Florian Zuleger
TU Wien, AT