Report of Dagstuhl Seminar 15491

Approximate and Probabilistic Computing: Design, Coding, Verification

Edited by

Antonio Filieri[1], Marta Kwiatkowska[2], Sasa Misailovic[3], and Todd Mytkowicz[4]

1 **Imperial College London, GB,** a.filieri@imperial.ac.uk
2 **University of Oxford, GB,** marta.kwiatkowska@cs.ox.ac.uk
3 **University of Illinois at Urbana-Champagin, US,** misailo@illinois.edu
4 **Microsoft Corporation – Redmond, US,** toddm@microsoft.com

Abstract

Computing has entered the era of *approximation*, in which hardware and software generate and reason about estimates. Navigation applications turn maps and location estimates from hardware GPS sensors into driving directions; speech recognition turns an analog signal into a likely sentence; search turns queries into information; network protocols deliver unreliable messages; and recent advances promise that approximate hardware and software will trade result quality for energy efficiency. Millions of people already use software which computes with and reasons about approximate/probabilistic data daily. These complex systems require sophisticated algorithms to deliver *accurate* answers quickly, at scale, and with energy efficiency, and approximation is often the only way to meet these competing goals.

Despite their ubiquity, economic significance, and societal impact, building such applications is difficult and requires expertise across the system stack, in addition to statistics and application-specific domain knowledge. Non-expert developers need tools and expertise to help them design, code, and verify these complex systems.

The aim of this seminar was to bring together academic and industrial researchers from the areas of probabilistic model checking, quantitative software analysis, probabilistic programming, and approximate computing to share their recent progress, identify challenges in computing with estimates, and foster collaboration with the goal of helping non-expert developers design, code, and verify modern approximate and probabilistic systems.

Seminar November 29–4, 2015 – http://www.dagstuhl.de/15491

1998 ACM Subject Classification D.2.4 Software/Program Verification, D.3.3 Language Constructs and Features, F.3.2 Semantics of Programming Languages, F.1.2 Modes of Computation

Keywords and phrases approximation, model checking, performance, probability, program analysis, systems, verification

Digital Object Identifier 10.4230/DagRep.5.11.151

Edited in cooperation with Sara Achour

Approximate and Probabilistic Computing: Design, Coding, Verification, *Dagstuhl Reports*, Vol. 5, Issue 11, pp. 151–179
Editors: Antonio Filieri, Marta Kwiatkowska, Sasa Misailovic, and Todd Mytkowicz
Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Executive Summary

Antonio Filieri
Marta Kwiatkowska
Sasa Misailovic
Todd Mytkowicz

Uncertainty and approximation are becoming first class concepts in software design and development. Many application domains, including biology, multimedia processing, finance, engineering, and social sciences, need software to formalize and study intrinsically uncertain phenomena. Furthermore, the ubiquity of software, especially driven by the Internet and mobility – such as driving applications that estimate routes, speech processing applications that estimate most likely sentences, or fitness applications that estimate heart-rate – require software engineers to design their applications taking into account unpredictable and volatile operational conditions, and noisy data, despite the limited support provided by current unintuitive design and quality assurance methodologies. Finally, the hardware community is designing devices that trade result accuracy for computational efficiency and energy saving, providing only probabilistic guarantees on the correctness of the computed results.

Several research communities are independently investigating methodologies and techniques to model, analyze, and manage uncertainty in and through software systems. These areas include (1) probabilistic model checking, (2) quantitative software analysis, (3) probabilistic programming, and (4) approximate computing. However, despite the substantial overlap of interests, researchers from different communities rarely have the opportunity to meet at conferences typically tailored to single specific areas. Therefore, we organized this seminar as a forum for industrial and academic researchers from these areas to share their recent ideas, identify the main research challenges and future directions, and explore collaborative research opportunities on problems that span across the boundaries of the individual areas.

This report presents a review of each of the main areas covered by the seminar and summarizes the discussions and conclusions of the participants.

Acknowledgements. The organizers would like to express their gratitude to the participants and the Schloss Dagstuhl team for a productive and exciting seminar. We thank Prof. Martin Rinard for his support and contribution to the organization of the seminar. We thank Sara Achour for her help with preparing this report.

2 Table of Contents

Executive Summary
Antonio Filieri, Marta Kwiatkowska, Sasa Misailovic, and Todd Mytkowicz 152

Research Areas

Probabilistic Model Checking 155

Quantitative Program Analysis 156

Probabilistic Programming 157

Approximate Computing 158

Overview of Talks

Approximate computation with outlier detection in Topaz
Sara Achour 164

Numerical Program Analysis Tools: A Wish List
David Bindel 165

Optimizing Synthesis with Metasketches (for Automated Approximate Programming)
James Bornholt 165

Stochastic approximations for Stochastic Model Checking
Luca Bortolussi 166

Counterexample Explanation by Learning Small Strategies in Markov Decision Processes
Tomas Brázdil 166

Approximate Computing on Unreliable Silicon
Andreas Peter Burg 167

Approximate Overview of Approximate Computing
Luis Ceze 167

Programming with Numerical Uncertainties
Eva Darulova 167

The dual value of Probabilistic Abstract interpretation
Alessandra Di Pierro 168

Termination of Probabilistic Programs
Luis María Ferrer Fioriti 168

Quality-Energy Aware System Design
Andreas Gerstlauer 169

Probabilistic Programming Process Algebra
Jane Hillston 169

On Quantification of Accuracy Loss in Approximate Computing
Ulya R. Karpuzcu 170

Understanding and Analysing Probabilistic Programs
Joost-Pieter Katoen 170

Computing Reliably with Molecular Walkers
Marta Kwiatkowska 170

Approximate counting for SMT
Rupak Majumdar 171

Smoothed Model Checking: A Machine Learning Approach to Probabilistic Model Checking under Uncertainty
Dimitrios Milios 171

Accuracy-Aware Compiler Optimizations
Sasa Misailovic 172

Intuitors, Computers and Validators: Towards Effective Decision-Making Systems
Ravi Nair 172

Error Resilient Systems and Approximate Computing: Conjoined Twins Separated at Birth
Karthik Pattabiraman 173

ACCEPT: We Built an Open-Source Approximation Compiler Framework So You Don't Have To
Adrian Sampson 173

Approximate Storage
Karin Strauss 174

DNA Storage
Karin Strauss 174

Quantifying Program Differences
Willem Visser 175

On a Framework for Quantitative Program Synthesis
Herbert Wiklicky 175

Achievements of this Seminar 175

Participants 179

3 Research Areas

3.1 Probabilistic Model Checking

Probabilistic modelling is widely used in the design and analysis of computer systems, and has been rapidly gaining in importance in recent years. Traditionally, models such as Markov chains have been used to analyse system performance, where typically queuing theory is applied to obtain quantitative characteristics. Probability is also needed to quantify unreliable or unpredictable behaviour, for example in fault-tolerant systems and communication protocols, where properties such as component failure and packet loss can be described probabilistically. Probabilistic models with nondeterminism, e.g., Markov decision processes, are employed for modelling of distributed co-ordination protocols which use randomisation as a symmetry breaker, in wireless medium-access control, and probabilistic routing in security and anonymity protocols. More generally, Markovian models are useful to support decision making, for example in economics, operations research, planning and robotics, to optimise a certain goal function.

Probabilistic model checking [66, 29, 5, 27] is an automatic procedure for establishing if a desired property holds in a probabilistic system model. Conventional model checkers input a description of a model, representing a state-transition system, and a specification, typically a formula in some temporal logic, and return “yes” or “no”, indicating whether or not the model satisfies the specification. In the case of probabilistic model checking, the models are probabilistic (typically variants of Markov chains), in the sense that they encode the probability of making a transition between states instead of simply the existence of such a transition. A probability space induced on the system behaviours enables the calculation of likelihood of the occurrence of certain events during the execution of the system. This in turn allows one to make quantitative statements about the system [37], in addition to the qualitative statements made by conventional model checking. Probabilities are captured via probabilistic operators that extend conventional (timed or untimed) temporal logics, affording the expression of probabilistic specifications such as minimising the probability of a security attack, reliability of a nanotechnology design, and ensuring that expected energy usage of the protocol is below a specified bound.

Probabilistic model checking combines graph-theoretic analysis, drawn from conventional model checking, together with probabilistic analysis. The latter involves numerical computation, such as solving linear equations or linear programming problems, which for scalability reasons is typically implemented using iterative methods in symbolic data structures [3, 36]. This lends itself to approximate computation, where one can trade off accuracy for speed by terminating the computation early. The models are described in high-level modelling notations, or can be extracted from, e.g., C programs extended with random assignment [34]. An alternative approach, called approximate or statistical model checking [30, 68, 67], is based on simulating execution runs and applying statistical techniques such as hypothesis testing to estimate the probability or expectation of some event holding. However, no inference on data is currently combined with probabilistic model checking techniques, which focus on system dynamics. An important new direction is synthesis, which aims to construct a model that is guaranteed to satisfy a given probabilistic specification. Recently formulated and implemented simpler variants of this problem include parameter synthesis [14, 16], which finds optimal parameter values that satisfy the property and for model repair, and controller/strategy synthesis [17], with which one can generate correct-by-construction controllers from specifications.

Probabilistic model checking algorithms were proposed in the 1980s [66, 15], but it was not until early 2000s when the first industrially-relevant tools were released, notably

PRISM [38] and MRMC [33]. PRISM, in particular, is based on symbolic techniques that provide compact storage for probabilistic models and ensure efficiency of (approximate) computation of the probability. In [9], the performance of PRISM was recently improved by incorporating machine learning, with which one can obtain guarantees on accuracy while exploring only a portion of the state space. PRISM supports five probabilistic models, including probabilistic timed automata and stochastic games, for both verification and strategy synthesis. Applications of probabilistic model checking using PRISM have spanned multiple fields, from wireless protocols and source code analysis of Linux networking utilities, through debugging DNA computing designs, to smart energy grids and strategy synthesis for autonomous urban driving. The software technology underpinning probabilistic model checking has matured; it has been applied to analyse the reliability of NAND gates design, detecting a bug in an analytical model, and is being adopted, for example, in software engineering and resource management of cloud computing systems.

3.2 Quantitative Program Analysis

Probabilistic model checking developed a set of theories, algorithms, and tools aimed at verifying the properties of a variety of stochastic models. However, their applications to software engineering is mostly limited to early stages of development, where design models are translated in a more or less automatic way to corresponding stochastic models. These semantic views on the software to-be are valuable decision support systems for designers that can quantitatively evaluate the impact of their choices, especially with respect to nonfunctional requirements such as reliability or performance. However, design models are hard to keep consistent with implementation, where code artifacts are in general only partially compliant with their intended design. To mitigate this inconsistency the three main approaches are simulation [43], profiling [28], and keeping models "alive" at runtime via continuous monitoring [20]. The goal of these techniques is to perform additional measurements on the implemented artifacts in order to update the initial design assumptions as captured by design-stage models. However, these approaches can only provide coarse grained information on the implemented software that can hardly be linked to the code.

Furthermore, the widespread use of agile development processes makes the code the central, and often unique, formal model of the program. Several reverse engineering approaches attempted to automatically extract models from the code, however the extraction of meaningful models remains an open problem [10]. Black-box analysis approaches have also been proposed [64]; though useful for overall quality assessment, these approaches do not support the localization of errors or otherwise drive the improvement of the program.

Static program analysis techniques aim at checking a variety of properties of an application starting from its source code. These properties include, for example, correctness, robustness, liveness or reachability of specific statements. However, most of these techniques cannot take advantage of the characterization of uncertainty about a program inputs or about its execution flow, providing in turn less informative true-false answers. Probabilistic analysis has to be brought at the code level to support the entire development processes, from design to code and quality assurance.

Several researchers have proposed probabilistic variants of static analysis techniques, such as data flow analyses [56, 50]. In these approaches the distributions determining the probability of following each of the edges of an execution branch are supposed to be provided by the users or are coarsely estimated by monitoring a set of program executions as in [1].

Neither of these approaches is fully satisfactory since they characterize the probability of a given branch independently from the program state when the branch occurs, limiting the precision of the resulting quantitative analysis.

Probabilistic symbolic execution (PSE) is a recent technique that can be directly applied, in combination with an input probability distribution, to compute information about the probability of executing a program path, statement, or branch or, more generally, of reaching a program state [23, 21]. This technique is an example of white-box source code analysis that relies only on program semantics to quantify program behavior, taking also into account probabilistic information about its execution environment, including its deployment environment and the interaction with users and third-party components. Among the recent PSE-based techniques, [23, 21] perform an exhaustive analysis of Java programs whose branch conditions are limited to linear numeric constraints, providing precise results but suffering from scalability issues; [6] addresses the approximate analysis of non-linear constraints; [41] deals with nondeterminism and multithreaded programs; [22] provides incremental statistical analysis with quantified confidences on the results.

3.3 Probabilistic Programming

Quantitative program analysis is focused on general programs dealing with probabilistic phenomena (e.g., unpredictable interaction with users). On the other hand, probabilistic programming makes uncertainty a first-class concept and thus enables probabilistic inference.

Probabilistic programming languages augment existing programming languages with probabilistic primitives [26]. The major goal of these languages is the efficient implementation of probabilistic inference, which combines a model (written in the probabilistic programming language) with observed evidence to *infer* a distribution over variables in the program in light of that evidence. These languages abstract the details of inference, and so see frequent use by machine learning experts when building their models. Probabilistic programming has made significant strides in democratizing probabilistic inference; they let machine learning experts encode models and then ask complicated and computationally demanding queries via probabilistic inference, of those models. While, in general probabilistic inference is `NP-Hard`, probabilistic programming languages work hard to make (potentially approximate) inference efficient for many applications of practical interest.

Probabilistic programming is a well-studied field: some probabilistic programming languages such as Church [25] are theoretically universal, in that they can perform inference on any distribution they can represent. Venture [44] extends Church to allow the programmer to determine the inference algorithm to use on each part of the model. Other probabilistic programming languages restrict the distributions they allow, to make inference more tractable and efficient. Infer.NET [45, 7] uses various approximate and exact inference engines, each of which has different restrictions. For example, its Gibbs sampling [24] engine requires the distributions of related variables to be conjugate, a very strong restriction. These restrictions often require statistical expertise to evaluate, making such algorithms inappropriate for an abstraction aimed at non-experts.

Park *et al.*[54] propose a probabilistic programming language based upon sampling functions [54] which represents distributions as sampling functions, and uses operations from the probability monad [57] to build more complex distributions. Bornholt *et al.* [8] extends this idea to treat normal imperative programs, which compute with estimates, as sampling functions, thus lowering the expertise required to write a probabilistic program.

However, Bornholt *et al.*'s approach does not yet allow full probabilistic inference, like the aforementioned probabilistic programming languages.

3.4 Approximate Computing

Many modern applications are inherently approximate. For instance, multimedia processing, machine learning, and big-data analytics applications perform approximate operations on large data sets. Applications that run on today's mobile and wearable computing devices make decisions based on data from approximate hardware components (e.g., GPS, gyroscope, or accelerometer).

Up to now, developers of approximate applications had to manually reason about accuracy, energy consumption, and timely execution. Design and implementation of these applications have often been ad-hoc – hardware and software would be developed independently of each other, and integration required significant expertise at each layer of the system stack.

Approximate computing is an emerging research area that focuses on devising systematic approaches for automating development and compilation of approximate software that runs on today's commodity and approximate hardware, or tomorrow's more exotic approximate hardware. Its goal is to (1) empower a developer with the understanding of how approximate hardware and software affect the application's accuracy results, and (2) automate the management of application's accuracy, energy consumption, and performance. To achieve this goal, approximate computing brings together researchers from software systems – programming languages and software engineering – and hardware systems – circuit design and hardware architecture.

Researchers have recently proposed a number of approximate hardware designs and software optimization techniques that trade accuracy for performance and/or energy savings:

- *Approximate Hardware Architectures.* Researchers in academia have proposed a number of hardware designs with approximate accelerators or cores [39, 19, 51], ALUs [52, 18, 51], and memories [40, 61]. Typically, these designs specify the frequency of failure of their components (e.g., an addition instruction may produce a wrong result with a small probability), and/or the magnitude of error (e.g., an addition instruction may produce a small bounded noise). Researchers in industry have also proposed novel approximate hardware components, including Qualcomm's and IBM's neuromorphic accelerators [55, 32], Intel's approximate Minerva ALU design [35], and Lyric Semiconductor's (now a part of Analog Devices) belief propagation accelerator [42].
- *Approximation-Aware Compiler Optimizations.* These transformations automatically change the semantics of programs that execute on reliable (commodity) hardware to trade the accuracy of the program's result for the improved performance and/or energy consumption [58, 49, 13, 69, 47, 59]. For instance, loop perforation is a software-only technique that modifies the program to execute fewer loop iterations and therefore make the program run faster [49]. A compiler can also automate placement of operations that execute on approximate hardware [46].
- *Approximation-Aware Programming Languages and Libraries.* Programming languages such as Eon [65], EnerJ [60], and Rely [11] expose the hardware-level approximation to the developer through specific language constructs. Libraries, such as Uncertain<T> [8], provide abstractions that encapsulate approximate data within standard object-oriented programming languages. Runtime systems, such as those in Green [2], Dynamic Knobs [31], and Paraprox [59] dynamically adapt an approximate application to maintain desired result accuracy or responsiveness.

Key challenges to adopting these and other approximation techniques include characterizing their effects on the accuracy of program results and program performance. We discuss these challenges below.

- *Modeling Uncertainty:* Uncertainty can enter computation through inputs, hardware, or emerge in computation by using probabilistic language constructs. Researchers have often modeled this uncertainty probabilistically. For instance, hardware instructions produce correct results with a specified probability, a computation specifies probabilities of executing one of several approximate function versions, or the input noise has a specific probability distribution [60, 69, 48].
- *Accuracy Analysis:* Probabilistic static program analyses compute conservative bounds on the probability of large output deviations. These analyses reason about programs that operate on approximate hardware [46], programs transformed using accuracy-aware transformations [48, 69, 13], and programs that operate on uncertain inputs [63, 62]. Sampling and sensitivity testing based dynamic program analyses estimate the probability of large output deviations by running these programs on representative inputs [58, 12, 49, 4].
- *Searching for Optimal Tradeoffs:* Approximate hardware components and program transformations induce a tradeoff space between application's accuracy and performance. Optimization techniques therefore explore the tradeoff space looking for the approximate program configurations that maximize performance or energy savings subject to constraints on the accuracy of the results. Exploration can be performed using dynamic testing [58, 49, 2, 31, 59, 53], or statically reducing computation optimization to linear or integer mathematical programming [69, 46].

References

1 Glenn Ammons and James R. Larus. Improving data-flow analysis with path profiles. In *Proceedings of the ACM SIGPLAN 1998 Conference on Programming Language Design and Implementation*, PLDI'98, pages 72–84. ACM, 1998. doi:10.1145/277650.277665.

2 Woongki Baek and Trishul M. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. In *Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI'10, pages 198–209, New York, NY, USA, 2010. ACM. doi:10.1145/1806596.1806620.

3 Christel Baier, Edmund M. Clarke, Vasiliki Hartonas-Garmhausen, Marta Kwiatkowska, and Mark Ryan. Symbolic model checking for probabilistic processes. In Pierpaolo Degano, Roberto Gorrieri, and Alberto Marchetti-Spaccamela, editors, *Automata, Languages and Programming*, volume 1256 of *Lecture Notes in Computer Science*, pages 430–440. Springer, 1997. doi:10.1007/3-540-63165-8_199.

4 Tao Bao, Yunhui Zheng, and Xiangyu Zhang. White box sampling in uncertain data processing enabled by program analysis. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA'12, pages 897–914. ACM, 2012. doi:10.1145/2384616.2384681.

5 Andrea Bianco and Luca de Alfaro. Model checking of probabilistic and nondeterministic systems. In P.S. Thiagarajan, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513. Springer, 1995. doi:10.1007/3-540-60692-0_70.

6 Mateus Borges, Antonio Filieri, Marcelo d'Amorim, Corina S. Păsăreanu, and Willem Visser. Compositional solution space quantification for probabilistic software analysis. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI'14, pages 123–132. ACM, 2014. doi:10.1145/2594291.2594329.

7 Johannes Borgström, Andrew D. Gordon, Michael Greenberg, James Margetson, and Jurgen Van Gael. Measure transformer semantics for bayesian machine learning. In Gilles Barthe, editor, *Programming Languages and Systems*, volume 6602 of *Lecture Notes in Computer Science*, pages 77–96. Springer, 2011. doi:10.1007/978-3-642-19718-5_5.

8 James Bornholt, Todd Mytkowicz, and Kathryn S. McKinley. Uncertain<t>: A first-order type for uncertain data. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS'14, pages 51–66, New York, NY, USA, 2014. ACM. doi:10.1145/2541940.2541958.

9 T. Brázdil, K. Chatterjee, M. Chmelík, V. Forejt, J. Křetínský, M. Kwiatkowska, D. Parker, and M. Ujma. Verification of markov decision processes using learning algorithms. In *Proc. 12th International Symposium on Automated Technology for Verification and Analysis (ATVA'14)*, LNCS. Springer, 2014. To appear.

10 Gerardo Canfora, Massimiliano Di Penta, and Luigi Cerulo. Achievements and challenges in software reverse engineering. *Commun. ACM*, 54(4):142–151, April 2011. doi:10.1145/1924421.1924451.

11 Michael Carbin, Sasa Misailovic, and Martin C. Rinard. Verifying quantitative reliability for programs that execute on unreliable hardware. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA'13, pages 33–52, New York, NY, USA, 2013. ACM. doi:10.1145/2509136.2509546.

12 Michael Carbin and Martin C. Rinard. Automatically identifying critical input regions and code in applications. In *Proceedings of the 19th International Symposium on Software Testing and Analysis*, ISSTA'10, pages 37–48. ACM, 2010. doi:10.1145/1831708.1831713.

13 Swarat Chaudhuri, Sumit Gulwani, Roberto Lublinerman, and Sara Navidpour. Proving programs robust. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE'11, pages 102–112, New York, NY, USA, 2011. ACM. doi:10.1145/2025113.2025131.

14 Taolue Chen, E.M. Hahn, Tingting Han, M. Kwiatkowska, Hongyang Qu, and Lijun Zhang. Model repair for markov decision processes. In *Theoretical Aspects of Software Engineering (TASE), 2013 International Symposium on*, pages 85–92, July 2013. doi:10.1109/TASE.2013.20.

15 Costas Courcoubetis and Mihalis Yannakakis. Markov decision processes and regular events. In MichaelS. Paterson, editor, *Automata, Languages and Programming*, volume 443 of *Lecture Notes in Computer Science*, pages 336–349. Springer, 1990. doi:10.1007/BFb0032043.

16 M. Diciolla, C. H. P. Kim, M. Kwiatkowska, and A. Mereacre. Synthesising optimal timing delays for timed i/o automata. In *14th International Conference on Embedded Software (EMSOFT'14)*, 2014 - to appear.

17 Klaus Drager, Vojtěch Forejt, Marta Kwiatkowska, David Parker, and Mateusz Ujma. Permissive controller synthesis for probabilistic systems. In Erika Abraham and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 8413 of *Lecture Notes in Computer Science*, pages 531–546. Springer, 2014. doi:10.1007/978-3-642-54862-8_44.

18 Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Architecture support for disciplined approximate programming. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, pages 301–312. ACM, 2012. doi:10.1145/2150976.2151008.

19 Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural acceleration for general-purpose approximate programs. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, pages 449–460. IEEE Computer Society, 2012. doi:10.1109/MICRO.2012.48.

20 Antonio Filieri, Carlo Ghezzi, and Giordano Tamburrelli. A formal approach to adaptive software: continuous assurance of non-functional requirements. *Formal Aspects of Computing*, 24(2):163–186, 2012. `doi:10.1007/s00165-011-0207-2`.

21 Antonio Filieri, Corina S. Păsăreanu, and Willem Visser. Reliability analysis in symbolic pathfinder. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE'13, pages 622–631. IEEE Press, 2013. `doi:10.1109/ICSE.2013.6606608`.

22 Antonio Filieri, Corina S. Păsăreanu, Willem Visser, and Jaco Geldenhuys. Statistical symbolic execution with informed sampling. In *Proceedings of the ACM SIGSOFT 22nd International Symposium on the Foundations of Software Engineering*, FSE'14. ACM, 2014. URL: http://goo.gl/GXxFLi.

23 Jaco Geldenhuys, Matthew B. Dwyer, and Willem Visser. Probabilistic symbolic execution. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, ISSTA 2012, pages 166–176. ACM, 2012. `doi:10.1145/2338965.2336773`.

24 Stuart Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-6(6):721–741, Nov 1984. `doi:10.1109/TPAMI.1984.4767596`.

25 Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: A language for generative models. In *Uncertainty in Artificial Intelligence*, pages 220–229, 2008. URL: http://arxiv.org/pdf/1206.3255.

26 Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. Probabilistic programming. In *Proceedings of the on Future of Software Engineering*, FOSE 2014, pages 167–181, New York, NY, USA, 2014. ACM. `doi:10.1145/2593882.2593900`.

27 Christel gordon and Marta Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11(3):125–155, 1998. `doi:10.1007/s004460050046`.

28 K. Goseva-Popstojanova, M. Hamill, and R. Perugupalli. Large empirical case study of architecture-based software reliability. In *Software Reliability Engineering, 2005. ISSRE 2005. 16th IEEE International Symposium on*, pages 52–61, Nov 2005. `doi:10.1109/ISSRE.2005.25`.

29 Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994. `doi:10.1007/BF01211866`.

30 Thomas Herault, Richard Lassaigne, Frederic Magniette, and Sylvain Peyronnet. Approximate probabilistic model checking. In Bernhard Steffen and Giorgio Levi, editors, *Verification, Model Checking, and Abstract Interpretation*, volume 2937 of *Lecture Notes in Computer Science*, pages 73–84. Springer Berlin Heidelberg, 2004. `doi:10.1007/978-3-540-24622-0_8`.

31 Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. Dynamic knobs for responsive power-aware computing. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pages 199–212, New York, NY, USA, 2011. ACM. `doi:10.1145/1950365.1950390`.

32 New ibm synapse chip could open era of vast neural networks. http://www-03.ibm.com/press/us/en/pressrelease/44529.wss.

33 Joost-Pieter Katoen, Ivan S. Zapreev, Ernst Moritz Hahn, Holger Hermanns, and David N. Jansen. The ins and outs of the probabilistic model checker mrmc. *Perform. Eval.*, 68(2):90–104, February 2011. `doi:10.1016/j.peva.2010.04.001`.

34 M. Kattenbelt, M. Kwiatkowska, G. Norman, and D. Parker. Abstraction refinement for probabilistic software. In N. Jones and M. Muller-Olm, editors, *Proc. 10th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'09)*, volume 5403 of *LNCS*, pages 182–197. Springer, 2009.

35 H. Kaul, M. Anders, S. Mathew, S. Hsu, A. Agarwal, F. Sheikh, R. Krishnamurthy, and S. Borkar. A 1.45ghz 52-to-162gflops/w variable-precision floating-point fused multiply-add unit with certainty tracking in 32nm cmos. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pages 182–184, Feb 2012. doi:10.1109/ISSCC.2012.6176987.

36 M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. *International Journal on Software Tools for Technology Transfer (STTT)*, 6(2):128–142, 2004.

37 Marta Kwiatkowska. Quantitative verification: Models techniques and tools. In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC-FSE'07, pages 449–458. ACM, 2007. doi:10.1145/1287624.1287688.

38 Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification*, volume 6806 of *Lecture Notes in Computer Science*, pages 585–591. Springer, 2011. doi:10.1007/978-3-642-22110-1_47.

39 L. Leem, Hyungmin Cho, J. Bau, Q.A. Jacobson, and S. Mitra. Ersa: Error resilient system architecture for probabilistic applications. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 1560–1565, March 2010. doi:10.1109/DATE.2010.5457059.

40 Song Liu, Karthik Pattabiraman, Thomas Moscibroda, and Benjamin G. Zorn. Flikker: Saving dram refresh-power through critical data partitioning. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pages 213–224. ACM, 2011. doi:10.1145/1950365.1950391.

41 Kasper Luckow, Corina S. Păsăreanu, Matthew B. Dwyer, Antonio Filieri, and Willem Visser. Exact and approximate probabilistic symbolic execution for nondeterministic programs. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE'14, pages 575–586. ACM, 2014. doi:10.1145/2642937.2643011.

42 Lyriclabs: High probability of success. http://newsoffice.mit.edu/2013/ben-vigoda-lyric-0501.

43 Michael R. Lyu, editor. *Handbook of Software Reliability Engineering.* McGraw-Hill, Inc., Hightstown, NJ, USA, 1996.

44 Vikash K. Mansinghka, Daniel Selsam, and Yura N. Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *CoRR*, abs/1404.0099, 2014. URL: http://arxiv.org/abs/1404.0099.

45 T. Minka, J.M. Winn, J.P. Guiver, and D.A. Knowles. Infer.NET 2.5, 2012. Microsoft Research Cambridge. URL: http://research.microsoft.com/infernet.

46 Sasa Misailovic, Michael Carbin, Sara Achour, Zichao Qi, and Martin C. Rinard. Chisel: Reliability- and accuracy-aware optimization of approximate computational kernels. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA'14, pages 309–328. ACM, 2014. doi:10.1145/2660193.2660231.

47 Sasa Misailovic, Deokhwan Kim, and Martin Rinard. Parallelizing sequential programs with statistical accuracy tests. *ACM Trans. Embed. Comput. Syst.*, 12(2s):88:1–88:26, May 2013. doi:10.1145/2465787.2465790.

48 Sasa Misailovic, Daniel M. Roy, and MartinC. Rinard. Probabilistically accurate program transformations. In Eran Yahav, editor, *Static Analysis*, volume 6887 of *Lecture Notes in Computer Science*, pages 316–333. Springer, 2011. doi:10.1007/978-3-642-23702-7_24.

49 Sasa Misailovic, Stelios Sidiroglou, Henry Hoffmann, and Martin Rinard. Quality of service profiling. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering*, ICSE'10, pages 25–34, New York, NY, USA, 2010. ACM. `doi:10.1145/1806799.1806808`.

50 David Monniaux. An abstract monte-carlo method for the analysis of probabilistic programs. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL'01, pages 93–101. ACM, 2001. `doi:10.1145/360204.360211`.

51 Sriram Narayanan, John Sartori, Rakesh Kumar, and Douglas L. Jones. Scalable stochastic processors. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE'10, pages 335–338. European Design and Automation Association, 2010. `doi:10.1109/DATE.2010.5457181`.

52 K.V. Palem. Energy aware computing through probabilistic switching: a study of limits. *Computers, IEEE Transactions on*, 54(9):1123–1137, Sept 2005. `doi:10.1109/TC.2005.145`.

53 J. Park, X. Zhang, K. Ni, H. Esmaeilzadeh, and M. Naik. Expectation-oriented framework for automating approximate programming. Technical Report GT-CS-14-05, Georgia Institute of Technology, 2014. URL: https://smartech.gatech.edu/handle/1853/49755.

54 Sungwoo Park, Frank Pfenning, and Sebastian Thrun. A probabilistic language based on sampling functions. *ACM Trans. Program. Lang. Syst.*, 31(1):4:1–4:46, December 2008. `doi:10.1145/1452044.1452048`.

55 Introducing qualcomm zeroth processors: Brain-inspired computing. https://www.qualcomm.com/news/onq/2013/10/10/introducing-qualcomm-zeroth-processors-brain-inspired-computing.

56 G. Ramalingam. Data flow frequency analysis. In *Proceedings of the ACM SIGPLAN 1996 Conference on Programming Language Design and Implementation*, PLDI'96, pages 267–277. ACM, 1996. `doi:10.1145/231379.231433`.

57 Norman Ramsey and Avi Pfeffer. Stochastic lambda calculus and monads of probability distributions. In *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL'02, pages 154–165. ACM, 2002. `doi:10.1145/503272.503288`.

58 Martin Rinard. Probabilistic accuracy bounds for fault-tolerant computations that discard tasks. In *Proceedings of the 20th Annual International Conference on Supercomputing*, ICS'06, pages 324–334. ACM, 2006. `doi:10.1145/1183401.1183447`.

59 Mehrzad Samadi, Davoud Anoushe Jamshidi, Janghaeng Lee, and Scott Mahlke. Paraprox: Pattern-based approximation for data parallel applications. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS'14, pages 35–50, New York, NY, USA, 2014. ACM. `doi:10.1145/2541940.2541948`.

60 Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. Enerj: Approximate data types for safe and general low-power computation. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI'11, pages 164–174. ACM, 2011. `doi:10.1145/1993498.1993518`.

61 Adrian Sampson, Jacob Nelson, Karin Strauss, and Luis Ceze. Approximate storage in solid-state memories. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-46, pages 25–36, New York, NY, USA, 2013. ACM. `doi:10.1145/2540708.2540712`.

62 Adrian Sampson, Pavel Panchekha, Todd Mytkowicz, Kathryn S. McKinley, Dan Grossman, and Luis Ceze. Expressing and verifying probabilistic assertions. In *Proceedings of*

the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI'14, pages 112–122, New York, NY, USA, 2014. ACM. doi:10.1145/2594291.2594294.

63 Sriram Sankaranarayanan, Aleksandar Chakarov, and Sumit Gulwani. Static analysis for probabilistic programs: Inferring whole program properties from finitely many paths. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI'13, pages 447–458, New York, NY, USA, 2013. ACM. doi:10.1145/2491956.2462179.

64 Koushik Sen, Mahesh Viswanathan, and Gul Agha. Statistical model checking of black-box probabilistic systems. In Rajeev Alur and Doron A. Peled, editors, *Computer Aided Verification*, volume 3114 of *Lecture Notes in Computer Science*, pages 202–215. Springer, 2004. doi:10.1007/978-3-540-27813-9_16.

65 Jacob Sorber, Alexander Kostadinov, Matthew Garber, Matthew Brennan, Mark D. Corner, and Emery D. Berger. Eon: A language and runtime system for perpetual systems. In *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*, SenSys'07, pages 161–174. ACM, 2007. doi:10.1145/1322263.1322279.

66 M.Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Foundations of Computer Science, 1985., 26th Annual Symposium on*, pages 327–338, Oct 1985. doi:10.1109/SFCS.1985.12.

67 HakanL.S. Younes, EdmundM. Clarke, and Paolo Zuliani. Statistical verification of probabilistic properties with unbounded until. In Jim Davies, Leila Silva, and Adenilso Simao, editors, *Formal Methods: Foundations and Applications*, volume 6527 of *Lecture Notes in Computer Science*, pages 144–160. Springer, 2011. doi:10.1007/978-3-642-19829-8_10.

68 HåkanL.S. Younes, Marta Kwiatkowska, Gethin Norman, and David Parker. Numerical vs. statistical probabilistic model checking. *International Journal on Software Tools for Technology Transfer*, 8(3):216–228, 2006. doi:10.1007/s10009-005-0187-8.

69 Zeyuan Allen Zhu, Sasa Misailovic, Jonathan A. Kelner, and Martin Rinard. Randomized accuracy-aware program transformations for efficient approximate computations. In *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL'12, pages 441–454. ACM, 2012. doi:10.1145/2103656.2103710.

4 Overview of Talks

4.1 Approximate computation with outlier detection in Topaz

Sara Achour (MIT – Cambridge, US)

Joint work of Achour, Sara; Rinard, Martin
Main reference S. Achour, M. C. Rinard, "Approximate computation with outlier detection in Topaz," in Proc. of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'15), pp. 711–730, ACM, 2015.
URL http://dx.doi.org/10.1145/2814270.2814314

We present Topaz, a new task-based language for computations that execute on approximate computing platforms that may occasionally produce arbitrarily inaccurate results. Topaz maps tasks onto the approximate hardware and integrates the generated results into the main computation. To prevent unacceptably inaccurate task results from corrupting the main computation, Topaz deploys a novel outlier detection mechanism that recognizes and precisely re-executes outlier tasks. Outlier detection enables Topaz to work effectively with approximate hardware platforms that have complex fault characteristics, including platforms with bit pattern dependent faults (in which the presence of faults may depend on values stored in adjacent memory cells). Our experimental results show that, for our set of benchmark applications, outlier detection enables Topaz to deliver acceptably accurate results (less than 1% error) on our target approximate hardware platforms. Depending on the application and the hardware platform, the overall energy savings range from 5 to 13 percent. Without outlier detection, only one of the applications produces acceptably accurate results.

4.2 Numerical Program Analysis Tools: A Wish List

David Bindel (Cornell University, US)

In my scientific computing work, I am constantly faced with different sources of error: model error, stochastic error, discretization error, error due to approximation of some difficult term, error due to termination of iterations, and error due to roundoff effects. I deal with these errors by reasoning about forward and backward errors, stability and conditioning of iterations and of problems, the role of singularities, and structural properties of the computation must be retained for meaningful results. I dream of compilers with which I can share optimizations that I know are possible (and those that will break my code) and PL tools that understand enough to help me check my error analyses. I will share some of my own preliminary work in this direction, and will make an appeal to the audience to help produce the tools I wish I knew how to write.

4.3 Optimizing Synthesis with Metasketches (for Automated Approximate Programming)

James Bornholt (University of Washington – Seattle, US)

URL http://synapse.uwplse.org/

An ideal programming model for approximate computing would apply approximations automatically, translating an exact program and a quality specification into the most efficient program that meets that specification. Program synthesis is the task of automatically generating a program that meets a given specification, and sounds like a good fit for the approximate computing problem. But existing synthesis tools rarely consider the efficiency of solutions, because the required techniques require substantial domain-specific modifications to existing solvers. Optimal synthesis is the task of producing a solution that not only satisfies the specification but also minimizes a desired cost function.

We present metasketches, a general framework for specifying and solving optimal synthesis problems. Metasketches offer strategic control over the underlying synthesizer by specifying a fragmentation of the search space into an ordered set of classic sketches. We provide two cooperating search algorithms to effectively solve metasketches. A global optimizing search coordinates the activities of local searches, informing them of the costs of potentially-optimal solutions as they explore different regions of the candidate space in parallel. The local searches execute an incremental form of counterexample-guided inductive synthesis to incorporate information sent from the global search.

We present Synapse, an implementation of these algorithms, and show that it effectively solves optimal synthesis problems with a variety of different cost functions. In particular, we show that Synapse can find novel approximations to computational kernels that achieve speed-ups of between 1.6x and 35x without hardware support.

4.4 Stochastic approximations for Stochastic Model Checking

Luca Bortolussi (University of Trieste, IT)

Main reference L. Bortolussi, J. Hillston, "Model checking single agent behaviours by fluid approximation," Information and Computation, Vol. 242, pp. 183–226, 2015.
URL http://dx.doi.org/10.1016/j.ic.2015.03.002

We will briefly review a recent line of work trying to exploit different types of stochastic approximation (fluid approximation, linear noise approximation, moment closures) to model check a Markov population model against specific classes of properties. In the talk, we will focus mostly on individual properties, specified by CSL with rewards or by DTA.

4.5 Counterexample Explanation by Learning Small Strategies in Markov Decision Processes

Tomas Brázdil (Masaryk University – Brno, CZ)

Main reference T. Brázdil, K. Chatterjee, M. Chmelík, A. Fellner, J. Křetínský, "Counterexample Explanation by Learning Small Strategies in Markov Decision Processes," arXiv:1502.02834v1 [cs.LO], 2015.
URL http://arxiv.org/abs/1502.02834v1

While for deterministic systems, a counterexample to a property can simply be an error trace, counterexamples in probabilistic systems are necessarily more complex. For instance, a set of erroneous traces with a sufficient cumulative probability mass can be used. Since these are too large objects to understand and manipulate, compact representations such as subchains have been considered. In the case of probabilistic systems with non-determinism, the situation is even more complex. While a subchain for a given strategy (or scheduler, resolving non-determinism) is a straightforward choice, we take a different approach. Instead, we focus on the strategy – which can be a counterexample to violation of or a witness of satisfaction of a property – itself, and extract the most important decisions it makes, and present its succinct representation. The key tools we employ to achieve this are (1) introducing a concept of importance of a state w.r.t. the strategy, and (2) learning using decision trees. There are three main consequent advantages of our approach. Firstly, it exploits the quantitative information on states, stressing the more important decisions. Secondly, it leads to a greater variability and degree of freedom in representing the strategies. Thirdly, the representation uses a self-explanatory data structure. In summary, our approach produces more succinct and more explainable strategies, as opposed to e.g. binary decision diagrams. Finally, our experimental results show that we can extract several rules describing the strategy even for very large systems that do not fit in memory, and based on the rules explain the erroneous behaviour.

4.6 Approximate Computing on Unreliable Silicon

Andreas Peter Burg (EPFL – Lausanne, CH)

Joint work of Burg, Andreas; Karakonstantis, Georgios; Constantin, Jeremy; Teman, Adam

Approximate computing refers not only to approximating complex computations and algorithms with less complex ones, but can also be the basis for providing robustness against errors due to reliabilities of the underlying hardware. In this talk, we consider two types of hardware failure: timing errors and reliability issues in memories. We describe their impact and critically discuss their potential and issues in the context of approximate computing. We show that tolerating timing errors is particularly tricky, while errors in memories are more straightforward to model and exploit. For the latter, we also point out strategies for testing and quality assurance of unreliable hardware and we mention algorithm techniques to reduce the impact of errors on quality.

4.7 Approximate Overview of Approximate Computing

Luis Ceze (University of Washington – Seattle, US)

Motivation for approximate computing. Overview of approximate computing techniques from language to hardware.

4.8 Programming with Numerical Uncertainties

Eva Darulova (MPI-SWS – Saarbrücken, DE)

Numerical software, common in scientific computing or embedded systems, inevitably uses an approximation of the real arithmetic in which most algorithms are designed. Finite-precision arithmetic, such as fixed-point or floating-point, is a common and efficient choice, but introduces an uncertainty on the computed result that is often very hard to quantify. We need adequate tools to estimate the errors introduced in order to choose suitable approximations which satisfy the accuracy requirements. I will present a new programming model where the scientist writes his or her numerical program in a real-valued specification language with explicit error annotations. It is then the task of our verifying compiler to select a suitable floating-point or fixed-point data type which guarantees the needed accuracy. I will show how a combination of SMT theorem proving, interval and affine arithmetic and function derivatives yields an accurate, sound and automated error estimation which can handle nonlinearity, discontinuities and certain classes of loops.

4.9 The dual value of Probabilistic Abstract interpretation

Alessandra Di Pierro (University of Verona, IT)

Probabilistic Abstract Interpretation is a framework for program analysis that allows us to accommodate probabilistic properties and properties of probabilistic computations. We illustrate the dual value of this framework for both deducing and inferring probabilities. More specifically we show the use of PAI for both static analysis and statistical reasoning. The basic ingredient of the PAI framework that makes this possible is the notion of Moore-Penrose pseudo inverse and its least-square approximation property.

Suppose that we want to analyse a program to check whether it is secure up to a given level of accuracy. We can use probabilistic abstract interpretation as follows:

- Define mathematically what 'secure' means (e.g. as a probabilistic relation)
- Consider the semantics of the program restricted to this property (abstraction)
- Construct the Moore-Penrose generalised inverse of the abstraction in order to identify an ideal concrete system that satisfies the property up to the fixed accuracy.

Note that the concrete probabilities defining the concrete ideal system are just assumed and may have no relation with the real world.

Now suppose that we have some observations y at hand and we want to use them in order to define an ideal concrete system which is closer to the real one. To this purpose we can use probabilistic abstract interpretation as a linear statistical model in the way explained below:

- Consider the space V of all possible ideal concrete semantics (abstract domain)
- Define a mapping X from V to all possible observations (design matrix)
- Construct the MP generalised inverse of X in order to obtain the best estimate b of the concrete semantics that realises y.

Note that this is nothing else than the application of the Gauss-Markov theorem for linear regression in its simplest version.

4.10 Termination of Probabilistic Programs

Luis María Ferrer Fioriti (Universität des Saarlandes, DE)

Main reference L. M.Ferrer Fioriti, H. Hermanns, "Probabilistic Termination: Soundness, Completeness, and Compositionality," in Proc. of the 42nd Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL'15), pp. 489–501, ACM, 2015.
URL http://dx.doi.org/10.1145/2775051.2677001

The talk is an overview of the ranking supermartingale framework to prove almost sure termination of probabilistic programs.

4.11 Quality-Energy Aware System Design

Andreas Gerstlauer (University of Texas – Austin, US)

URL http://www.ece.utexas.edu/~gerstl/research.html#approximate

Approximate computing has emerged as a novel paradigm for achieving significant energy savings by trading off computational precision and accuracy in inherently error-tolerant applications. This introduces a new notion of quality as design parameter. Such approaches will only be successful, however, if quality can be guaranteed and design spaces can be efficiently explored. While ad-hoc solutions have been explored, systematic approaches are lacking. We have been investigating such quality-energy aware system design. At the hardware level, design strategies for synthesis of approximate arithmetic and logic circuits, including adders and multipliers demonstrate existence of a large design space of Pareto-optimal solutions. Such building blocks in turn form the basis for high-level synthesis of hardware and software into approximate datapaths of custom or programmable processors under a range of statistical quality constraints. Finally, at the system level, we envision a key question to be how to address the problem of quality-energy aware mapping and scheduling of application tasks onto general, quality-configurable system platforms.

4.12 Probabilistic Programming Process Algebra

Jane Hillston (University of Edinburgh, GB)

Joint work of Hillston, Jane; Georgoulas, Anastasis; Sanguinetti, Guido
Main reference A. Georgoulas, J. Hillston, D. Milios, G. Sanguinetti, "Probabilistic Programming Process Algebra," in Proc. of the 11th Int'l Conf. on Quantitative Evaluation of Systems (QEST'14), LNCS, Vol. 8657, pp. 249–264, Springer, 2014.
URL http://dx.doi.org/10.1007/978-3-319-10696-0_21

Formal modelling languages such as process algebras are effective tools in computational biological modelling. However, handling data and uncertainty in these representations in a statistically meaningful way is an open problem, limiting their usefulness in many real biological applications. In contrast, the machine learning community have recently proposed probabilistic programming as a way of expressing probabilistic models in a language which incorporates distributions and observations, and offers automated inference to update the likely distribution over values given the observations.

I will present work which seeks to combine these approaches allowing formal mechanistic models which encompass uncertainty, observations and inference.

4.13 On Quantification of Accuracy Loss in Approximate Computing

Ulya R. Karpuzcu (University of Minnesota – Minneapolis, US)

Emerging applications such as R(ecognition), M(ining), and S(ynthesis) suit themselves well to approximate computing due to their intrinsic noise tolerance. RMS applications process massive, yet noisy and redundant data by probabilistic, often iterative, algorithms. Usually the solution space has many more elements than one, rendering a range of application outputs valid, as opposed to a single golden value. A critical step in translating this intrinsic noise tolerance to energy efficiency is quantification of approximation-induced accuracy loss using application-specific metrics. This article covers pitfalls and fallacies in the development and deployment of accuracy metrics.

4.14 Understanding and Analysing Probabilistic Programs

Joost-Pieter Katoen (RWTH Aachen, DE)

We develop program analysis techniques, based on static program analysis, deductive verification, and model checking, to make probabilistic programming more reliable, i.e., less buggy. Starting from a profound understanding from the intricate semantics of probabilistic programs (including features such as observations, possibly diverging loops, continuous variables, non-determinism, as well as unbounded recursion), we study fundamental problems such as checking program equivalence, loop-invariant synthesis, almost-sure termination, and pre- and postcondition reasoning. The aim is to study the computational hardness of these problems as well as to develop (semi-) algorithms and accompanying tool-support. The ultimate goal is to provide lightweight automated means to the probabilistic programmer so as check elementary program properties.

4.15 Computing Reliably with Molecular Walkers

Marta Kwiatkowska (University of Oxford, GB)

Main reference F. Dannenberg, M. Kwiatkowska, C. Thachuk, A. J. Turberfiled, "DNA walker circuits: computational potential, design, and verification," Natural Computing, 14(2):195–211, 2015; pre-print available from project repository.
URL http://dx.doi.org/10.1007/s11047-014-9426-9
URL http://www.veriware.org/bibitem.php?key=DKTT15

DNA computing is emerging as a versatile technology that promises a vast range of applications, including biosensing, drug delivery and synthetic biology. DNA logic circuits can be achieved in solution using strand displacement reactions, or by decision-making molecular robots-so called 'walkers'-that traverse tracks placed on DNA 'origami' tiles. Similarly to conventional silicon technologies, ensuring fault-free DNA circuit designs is challenging, with the difficulty compounded by the inherent unreliability of the DNA technology and lack of scientific understanding. This lecture will give an overview of computational models that capture DNA walker computation and demonstrate the role of quantitative verification and synthesis in ensuring the reliability of such systems. Future research challenges will also be discussed.

4.16 Approximate counting for SMT

Rupak Majumdar (MPI-SWS – Kaiserslautern, DE)

♯SMT, or model counting for logical theories, is a well-known hard problem that generalizes such tasks as counting the number of satisfying assignments to a Boolean formula and computing the volume of a polytope. In the realm of satisfiability modulo theories (SMT) there is a growing need for model counting solvers, coming from several application domains (quantitative information flow, static analysis of probabilistic programs). We show a reduction from an approximate version of ♯SMT to SMT.

We focus on the theories of integer arithmetic and linear real arithmetic. We propose model counting algorithms that provide approximate solutions with formal bounds on the approximation error. They run in polynomial time and make a polynomial number of queries to the SMT solver for the underlying theory. We show an application of ♯SMT to the value problem for a model of loop-free probabilistic programs with nondeterminism.

4.17 Smoothed Model Checking: A Machine Learning Approach to Probabilistic Model Checking under Uncertainty

Dimitrios Milios (University of Edinburgh, GB)

Probabilistic model checking can provide valuable insights on the properties of stochastic systems. In many application fields however, it is not always possible to accurately identify some of the parameters of the model in question. It is therefore desirable to be able to perform model checking in presence of uncertainty. We show that the satisfaction probability of a temporal logic formula is a smooth function of the model parameters. This smoothness property enables us to construct an analytical approximation of the satisfaction function by using a well-established machine learning framework for approximating smooth functions. Extensive experiments on non-trivial case studies show that the approach is accurate and several orders of magnitude faster than naive parameter exploration with standard statistical model checking methods.

4.18 Accuracy-Aware Compiler Optimizations

Sasa Misailovic (MIT – Cambridge, US)

Joint work of Martin Rinard, Michael Carbin, Hank Hoffmann, Stelios Sidiroglou, Sara Achour, Zichao Qi, and Sasa Misailovic

Many modern applications (such as multimedia processing, machine learning, and big-data analytics) exhibit a natural tradeoff between the accuracy of the results they produce and the application's execution time or energy consumption. These applications allow us to investigate new, more aggressive optimization approaches.

I present a novel approximate optimization framework based on accuracy-aware program transformations. These transformations trade accuracy in return for improved performance, energy efficiency, and/or resilience. The optimization framework includes program analyses that characterize the accuracy of transformed programs and search techniques that navigate the tradeoff space induced by transformations to find approximate programs with profitable tradeoffs. I will present how we can use this accuracy-aware optimization framework to 1) automatically generate approximate programs with significantly improved performance and acceptable accuracy, and 2) automatically generate approximate functions that maximize energy savings when executed on approximate hardware platforms, while ensuring that the generated functions satisfy the developer's accuracy specifications.

4.19 Intuitors, Computers and Validators: Towards Effective Decision-Making Systems

Ravi Nair (IBM TJ Watson Research Center – Yorktown Heights, US)

Traditional computer systems are designed for applications such as transaction processing and physical simulations, largely using systematic algorithms with reliable computation and data movement. Machines are increasingly being asked to produce actionable results to large scale problems for which neither the data nor the available contextual information is 100% reliable. Approximate computing has been making significant headway towards better resource utilization for such new workloads, but the machines executing them still largely maintain the logical and deliberate nature of computer systems designed for traditional workloads. In several respects, today's computers are analogous to the slow, logical, and deliberate System 2 mode of human thought as described in the Nobel Laureate, Daniel Kahneman's book, "Thinking, Fast and Slow." We postulate that Kahneman's System 1 mode of thought, characterized by fast, intuitive, and energy-efficient decision making, suggests a new type of machine for new workloads, which we call an intuitor, which is different from a traditional computer. The incorporation of a validator which monitors the validity of the decision produced by an intuitor, allows the system to tolerate extreme forms of approximation, employing new types of devices and non-traditional architectures, in the design of intuitors. This talk will outline the symbiotic role of intuitors, computers, and validators in future decision-making systems.

4.20 Error Resilient Systems and Approximate Computing: Conjoined Twins Separated at Birth

Karthik Pattabiraman (University of British Columbia – Vancouver, CA)

Main reference A. Thomas, K. Pattabiraman, "Error detector placement for soft computation," in Proc. of the 43rd Annual IEEE/IFIP Int'l Conf. on Dependable Systems and Networks (DSN), pp. 1–12, IEEE, 2013.
URL http://dx.doi.org/10.1109/DSN.2013.6575353

The fields of approximate computing and error resilient systems have evolved independently, though they have a shared origin, namely how to ensure correctness in the presence of hardware faults ? In this talk, I will examine the similarities and differences between the two fields and how we can learn from each other. I will also present an example of a system that my students and I have worked on that attempts to bridge the gap between the two areas. I will conclude by presenting future challenges and opportunities in this area.

4.21 ACCEPT: We Built an Open-Source Approximation Compiler Framework So You Don't Have To

Adrian Sampson (University of Washington – Seattle, US)

Main reference A. Sampson, "Probabilistic Programming," Lecture notes, 2015.
URL http://adriansampson.net/notes/5d3mpq5r6ab0/
Main reference ACCEPT – An Approximate Compiler, Documentation.
URL http://accept.rocks/

Building and evaluating a new technique for approximate computing involves a lot of boring infrastructure work that can be far afield from the core of your work. You need a program annotation system to choose what to approximate, and you will want help writing annotations. You will want to tune each benchmark to take the best advantage of your new technique, and you will need to evaluate the final results on new inputs. If your technique works at a coarse grain, like a hardware accelerator does, you will need to search for large approximate regions to maximize the technique's effectiveness.

If every researcher continues to plod through these same steps independently, the community will waste a tragic amount of time in aggregate. As a fledgling research community, we need to collaborate on common infrastructure to build momentum in the field.

ACCEPT, the Approximate C Compiler for Energy and Performance Trade-offs, is an open-source framework that includes all the boring parts of building and evaluating an approximation technique. It has an annotation system, compiler feedback for the programmer, region inference, an auto-tuner, and Pareto frontier evaluation output. It comes with a suite of C and C++ benchmarks ready to run through the system. The source and documentation for ACCEPT are available now at http://accept.rocks/.

4.22 Approximate Storage

Karin Strauss (Microsoft Corporation – Redmond, US)

Joint work of Sampson, Adrian; Guo, Qing; Nelson, Jacob; Strauss, Karin; Ceze, Luis; Malvar, Henrique
Main reference A. Sampson, J. Nelsen, K. Strauss, L. Ceze, "Approximate Storage," in Proc. of the 46th Annual IEEE/ACM Int'l Symp. on Microarchitecture, pp. 25–36, ACM, 2013; pre-print available from company's repository.
URL http://dx.doi.org/10.1145/2540708.2540712
URL http://research.microsoft.com/apps/pubs/default.aspx?id=207379
Main reference Q. Guo, K. Strauss, L. Ceze, R. Malvar, "High-Density Image Storage Using Approximate Memory Cells," to appear in Proc. of the 21th ACM Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS'16); pre-print available from company's repository.
URL http://research.microsoft.com/apps/pubs/default.aspx?id=258662

In this talk, I will present the concept of approximate storage. Certain applications have inherent levels of noise and imprecision in them, yet memories still provide very high fidelity storage. However, scaling these memories to higher density is ever more challenging, and relaxing high fidelity requirements for tolerant applications may come to the rescue. I will show how to do this in a disciplined manner and report on the benefits of such approach. I will then describe our experience with storing images in approximate storage. If done naively, the quality degradation can be unacceptable. I will present an algorithm that takes importance of encoded bits on output quality into account during the encoding process to appropriately leverage approximate storage. It requires a small modification to an existing algorithm, yet it reduces quality degradation to practically imperceptible levels.

4.23 DNA Storage

Karin Strauss (Microsoft Corporation – Redmond, US)

Joint work of Bornholt, James; Lopez, Randolph; Carmean, Douglas M.; Ceze, Luis; Seelig, Georg; Strauss, Karin
Main reference J. Bornholt, R. Lopez, D. M. Carmean, L. Ceze, G. Seelig, K. Strauss, "A DNA-Based Archival Storage System," to appear in Proc. of the 21th ACM Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS'16); pre-print available from company's repository.
URL http://research.microsoft.com/apps/pubs/default.aspx?id=258661

In this talk, I will describe our project on using a DNA substrate to store digital data. DNA is dense, can be made very durable, and is easy to manipulate. I will explain how data can be stored in DNA, its advantages and challenges, and how to address some of these challenges. In specific, I will provide an overview of how to implement random access by leveraging existing protocols very common in life sciences research, and one way to encode digital data in DNA to improve its reliability while keeping overheads low.

4.24 Quantifying Program Differences

Willem Visser (Stellenbosch University – Matieland, ZA)

We will show to calculate the difference between two programs using Probabilistic Symbolic Execution. More specifically we will show that one can count the number of solutions to a path condition during symbolic execution and use this to calculate the percentage of inputs on which two programs give different outputs. A brief example will be given of how this work to analyse program mutations.

4.25 On a Framework for Quantitative Program Synthesis

Herbert Wiklicky (Imperial College London, GB)

Main reference H. Wiklicky, "Program Synthesis and Linear Operator Semantics," in Proc. of the 3rd Workshop on Synthesis (SYNTH'14), EPTCS, Vol. 157, pp. 17-33, 2014.
URL http://dx.doi.org/10.4204/EPTCS.157.6

Arguably most work on the problem of program synthesis is based on various models based in discrete structures, e.g. related to model checking, game theoretic models, combinatorial optimisation, etc. In this talk we aim in recasting program synthesis as a non-linear, continuous optimisation problem. This allows among other things for a smoother integration of non-functional constraints. Initial experiments demonstrate that, maybe surprisingly, it is possible to avoid algebraic reasoning for algebraic problems and replace it entirely by continuous optimisation constraints.

5 Achievements of this Seminar

Participants attending the seminar represented all four themes of the seminar. The program consisted of (1) tutorials, which introduced each of the main areas to all of the participants on the first day of the seminar, (2) 15-minute individual talks, which presented current research of the participants during the remaining days, (3) breakout sessions, during which the participants had an opportunity to discuss in more details specific points of interest, and (4) a panel, which discussed the main challenges and interactions between the areas.

Relations between the Areas. The participants identified probability and probabilistic reasoning as the underlying basis of all four areas. Figure 1 presents the main interactions between the areas[1]. For instance, some of the existing and anticipated interactions include:

- Probabilistic model checking, with its ability to establish whether a desired property of a probabilistic system holds, can be used to (1) verify the properties of approximate hardware and software systems against the formal specifications of their desired behavior, and (2) verify probabilistic assertions in probabilistic programs. In addition, probabilistic

[1] Figure 1 was compiled by Luis Ceze.

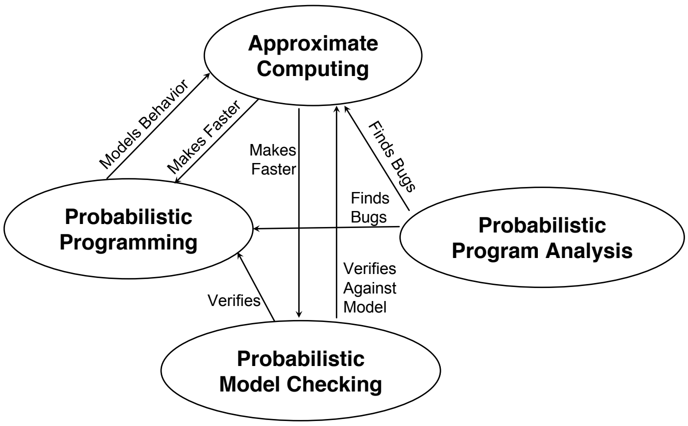

Figure 1 The main identified interactions among the areas.

model checking techniques based on dynamic programming have the flavor of any-time computation, and naturally lend themselves to approximate computation.
- Quantitative program analysis, such as probabilistic symbolic execution, can be used to (1) help find bugs and analyze properties of approximate software, which often implements randomized and/or probabilistic algorithms, and (2) improve testing of probabilistic inference engines and provide alternative strategies for computing results for some classes of probabilistic inference problems.
- Probabilistic programming, with its ability to represent complicated probabilistic models as computer programs and automate inference, can, in principle, represent a basis for specifying rich models of approximate software and hardware systems and enable Bayesian reasoning about the properties and self-adaptability of such systems.
- Approximate computing, with its ability to find efficient architecture and system level approximations for many emerging application domains, including probabilistic inference, has the potential to speed up various common inference tasks in probabilistic computing. But as it requires qualitative assurance of accuracy, it represents a potentially fruitful domain for applying systematic probabilistic reasoning studied in the remaining three areas, and thus creating novel expressive, precise, and scalable program/system analyses.

Open Research Questions. During the individual talks and the breakout sessions the participants identified and discussed many open research challenges and potentially fruitful directions, including the following:
- A key challenge for applying probabilistic reasoning to analyze approximate hardware is precise-enough modeling of the underlying phenomena that lead to approximate and/or unreliable results produced by a device. To that end, future research includes (1) selecting an appropriate levels of abstraction for a variety of hardware models and sources of result inaccuracy and (2) exposing inaccuracy and unreliability via appropriate specifications to the software level of the computing stack are open research problems. The approximate computing community, along with researchers from probabilistic programming, probabilistic model checking, and probabilistic verification, all need to develop a cogent specification of what it means to be approximate.
- Specifying and checking quality of approximate programs can be more systematized. Further work on benchmark suites for approximate computing – including specifications of representative inputs, quality metrics, and acceptable tolerance – can improve design

of future approximation and optimization techniques, and provide researchers from other areas with representative programs for testing their analyses. Furthermore, the development of domain-agnostic, standardized quality measures to push the interoperability of approximate computing applications.

- Understanding quality requirements of approximate subcomputations and code-level specifications, such as the frequency and/or magnitude of the errors of approximate subcomputations, can lead to new numerical analysis approaches that take advantage of system-level approximations, while providing theoretical guarantees for the behavior (for instance convergence) of the full algorithm.
- Some software is inherently resilient. For example, many numerical methods (e.g., iterative methods to learn a linear model) are naturally robust to noise. These algorithms offer a special playground for approximate hardware: if their robustness is sufficient to deal with the weak, non deterministic guarantees of an approximate hardware, the latter can be used for a faster and cheaper execution; otherwise the program can fall back to non-approximate hardware. Identifying for which algorithm this pattern can be fruitfully applied can drive a new generation of numerical libraries and pave the way to the definition of design guidelines for extending the approach to other classes of algorithms.
- *Thinking, Fast and Slow* is a best-selling book by Daniel Kahneman which posits humans use two high level modes of thought: "*system 1*", which is a fast and instinctive judgement and "*system 2*", which is computationally demanding and logical. This insight has been discussed in the context of approximate computing, where a cheap, fast to compute *system 1* approximate solution may be enhanced with a quantified confidence measure; the lack of a sufficient confidence on *system 1* results may trigger the use of a more deliberate, expensive, and proof-based *system 2*, which can provide more accurate results and reasons about whether the model uses by *system 1* is sufficient. This two-level pattern for building approximate systems seems promising for a variety of applications.
- Developing verification and abstraction techniques for probabilistic programs is a critical issue. The specification of probabilistic programs, as well as the meaning of correctness in this quantitative domain, have no generally accepted formalization. The semantics of simplified languages (e.g., constraining the input domain or the language operations) has been successfully abstracted into established stochastic models, such as Markov chains or Bayesian networks, inheriting the corpus of techniques developed in that area. However, the abstraction of more complex language constructs is still an open challenge. Furthermore, the generalization of recent results on probabilistic termination have to be investigated for complex probabilistic programming languages.
- Probabilistic programming and probabilistic program analysis share the development of a core of inference techniques. During the seminar, some inference problem arising from probabilistic programming have been efficiently solved using solution space quantification techniques from quantitative program analysis. However, the expressiveness of probabilistic programming goes beyond the current capabilities of quantitative program analysis, pushing for the study of new and more efficient solution space quantification techniques.
- Quantitative information about a program execution can inform program synthesis and repair approaches. Their usage at compiler level can be the basis of program optimization tailored to specific usage profiles. At the application level, quantitative information may guide the developer in representing the impact different code blocks have on the satisfaction of a program requirements, guiding debugging and prioritizing code refinements.

Case Studies. The seminar participants discussed various applications that can be used as inspiration for new research ideas that span multiple areas, in addition to classical application domains previously discussed in the literature. Two new emerging applications that span the spectrum include *self-driving cars* (investigated by several car manufacturers) and *mobile personal assistant programs* (such as Apple Siri, Google Now, and Microsoft Cortana). Both of these applications are characterized by uncertain data (e.g., coming from sensors) and environment (e.g., physical properties of the hardware), and their operation is routinely affected by human interaction.

However, the approaches for developing these applications have different objectives and different complementary expertise of the designers. Self-driving cars require strict certification, which in most cases includes formal verification of various timing and safety properties of the car components. Probabilistic verification, analysis, and control under uncertainty can, in principle, provide required guarantees that these properties hold. For this example, system-level approximations have the potential to help meet timing deadlines, but they need to be rigorously modeled and controlled.

In contrast, the tasks of personal assistant programs, which extract information and provide recommendations/opinions to the user, are considered best-effort computations. These applications typically perform natural language processing, probabilistic inference, and learning, for which guarantees of desirable program properties are welcome, they are typically not required for an end-to-end result quality. Personal assistant programs running on mobile devices therefore have more freedom to select the type and level of approximation, especially using new configurable approximate hardware components that give promise to significantly increase battery life.

Conclusion. The main objective of this seminar has been to discuss approaches to model and enable programs to seamlessly operate on uncertain data and computations. It has brought together academic and industrial researchers from the areas of probabilistic model checking, quantitative software analysis, probabilistic programming, and approximate computing. The discussion, enriched by the heterogeneity of the participants' perspectives, allowed the identification of several intersections among the interests of the four areas and a variety or research challenges that span across their boundaries. We anticipate that these together will contribute to the definition of the shared agenda among the four research communities.

Participants

- Sara Achour
MIT – Cambridge, US
- David Bindel
Cornell University, US
- Mateus Araújo Borges
Universität Stuttgart, DE
- James Bornholt
University of Washington – Seattle, US
- Luca Bortolussi
University of Trieste, IT
- Tomas Brázdil
Masaryk University – Brno, CZ
- Andreas Peter Burg
EPFL – Lausanne, CH
- Luis Ceze
University of Washington – Seattle, US
- Eva Darulova
MPI-SWS – Saarbrücken, DE
- Alessandra Di Pierro
University of Verona, IT
- Luis María Ferrer Fioriti
Universität des Saarlandes, DE
- Antonio Filieri
Imperial College London, GB
- Jaco Geldenhuys
University of Stellenbosch, ZA
- Andreas Gerstlauer
University of Texas – Austin, US
- Lars Grunske
HU Berlin, DE
- Jane Hillston
University of Edinburgh, GB
- Ulya R. Karpuzcu
University of Minnesota – Minneapolis, US
- Joost-Pieter Katoen
RWTH Aachen, DE
- Marta Kwiatkowska
University of Oxford, GB
- Rupak Majumdar
MPI-SWS – Kaiserslautern, DE
- Dimitrios Milios
University of Edinburgh, GB
- Sasa Misailovic
MIT – Cambridge, US
- Subhasish Mitra
Stanford University, US
- Todd Mytkowicz
Microsoft Corporation – Redmond, US
- Ravi Nair
IBM TJ Watson Res. Center – Yorktown Heights, US
- Karthik Pattabiraman
University of British Columbia – Vancouver, CA
- Adrian Sampson
University of Washington – Seattle, US
- Karin Strauss
Microsoft Corporation – Redmond, US
- Willem Visser
Stellenbosch University – Matieland, ZA
- Herbert Wiklicky
Imperial College London, GB