DAGSTUHL

REPORTS

Volume 6, Issue 3, March 2016

Dagstuhl Reports, Vol. 6, Issue 3 ISSN 2192-5283

ISSN 2192-5283

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at
http://www.dagstuhl.de/dagpub/2192-5283

Publication date
September, 2016

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

License
This work is licensed under a Creative Commons Attribution 3.0 DE license (CC BY 3.0 DE).
In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:
- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/DagRep.6.3.i

Aims and Scope
The periodical *Dagstuhl Reports* documents the program and the results of Dagstuhl Seminars and Dagstuhl Perspectives Workshops.
In principal, for each Dagstuhl Seminar or Dagstuhl Perspectives Workshop a report is published that contains the following:
- an executive summary of the seminar program and the fundamental results,
- an overview of the talks given during the seminar (summarized as talk abstracts), and
- summaries from working groups (if applicable).

This basic framework can be extended by suitable contributions that are related to the program of the seminar, e. g. summaries from panel discussions or open problem sessions.

Editorial Board
- Gilles Barthe
- Bernd Becker
- Stephan Diehl
- Hans Hagen
- Hannes Hartenstein
- Oliver Kohlbacher
- Stephan Merz
- Bernhard Mitschang
- Bernhard Nebel
- Bernt Schiele
- Nicole Schweikardt
- Raimund Seidel (*Editor-in-Chief*)
- Arjen P. de Vries
- Klaus Wehrle
- Reinhard Wilhelm

Editorial Office
Marc Herbstritt *(Managing Editor)*
Jutka Gasiorowski *(Editorial Assistance)*
Dagmar Glaser *(Editorial Assistance)*
Thomas Schillo *(Technical Assistance)*

Contact
Schloss Dagstuhl – Leibniz-Zentrum für Informatik
Dagstuhl Reports, Editorial Office
Oktavie-Allee, 66687 Wadern, Germany
reports@dagstuhl.de
http://www.dagstuhl.de/dagrep

Report from Dagstuhl Seminar 16101

Data Structures and Advanced Models of Computation on Big Data

Edited by

Alejandro Lopez-Ortiz[1], Ulrich Carsten Meyer[2], Markus E. Nebel[3], and Robert Sedgewick[4]

1 **University of Waterloo, CA,** alopez-o@uwaterloo.ca
2 **Goethe-Universität – Frankfurt a. M., DE,** umeyer@cs.uni-frankfurt.de
3 **TU Kaiserslautern, DE,** nebel@techfak.uni-bielefeld.de
4 **Princeton University, US,** rs@cs.princeton.edu

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 16101 "Data Structures and Advanced Models of Computation on Big Data". In today's computing environment vast amounts of data are processed, exchanged and analyzed. The manner in which information is stored profoundly influences the efficiency of these operations over the data. In spite of the maturity of the field many data structuring problems are still open, while new ones arise due to technological advances.

The seminar covered both recent advances in the "classical" data structuring topics as well as new models of computation adapted to modern architectures, scientific studies that reveal the need for such models, applications where large data sets play a central role, modern computing platforms for very large data, and new data structures for large data in modern architectures.

The extended abstracts included in this report contain both recent state of the art advances and lay the foundation for new directions within data structures research.

Seminar March 6–11, 2016 – http://www.dagstuhl.de/16101
1998 ACM Subject Classification E.1 Data Structures, F.1 Computation by Abstract Devices, F.2 Analysis of Algorithms and Problem Complexity, H.3 Information Storage and Retrieval
Keywords and phrases algorithms, big data, cloud services, data structures, external memory methods, information theory, large data sets, streaming, web-scale
Digital Object Identifier 10.4230/DagRep.6.3.1
Edited in cooperation with Sebastian Wild

1 Executive Summary

Alejandro Lopez-Ortiz (University of Waterloo, CA)
Ulrich Carsten Meyer (Goethe-Universität – Frankfurt a. M., DE)
Markus E. Nebel (TU Kaiserslautern, DE)
Robert Sedgewick (Princeton University, US)

License Creative Commons BY 3.0 Unported license
© Alejandro Lopez-Ortiz, Ulrich Carsten Meyer, Markus E. Nebel, and Robert Sedgewick

About the Seminar

Data structures provide ways of storing and manipulating data and information that are appropriate for the computational model at hand. Every such model relies on assumptions that we have to keep questioning. The aim of this seminar was to exchange ideas for new

Except where otherwise noted, content of this report is licensed under a Creative Commons BY 3.0 Unported license

algorithms and data structures, and to discuss our models of computations in light of recent technological advances. This Dagstuhl seminar was the 12th in a series of loosely related Dagstuhl seminars on data structures.

Topics

The presentations covered both advances in classic fields, as well as new models for recent trends in computing, in particular the appearance of big-data applications.

The talks by Brodal (§3.5), Penschuck (§3.19), Silvestri (§3.29), and Vahrenhold (§3.31) covered methods in the *external-memory model* that models the situation that data does no longer fit into internal memory. This limit can be pushed a bit further by using *succinct* data structures, which use only as much memory as absolutely necessary. Such methods were covered in the talks of Hagerup (§3.14), Raman (§3.25), and Gog (§3.12). If the task is to generate large random instances, Even (§3.9) showed that one can delay generation of large parts until they really become requested.

Big-data applications rely on *parallel* computation to speed up processing. Bingmann (§3.4) announced the creation of a new framework to simplify developing such applications. Brodnik (§3.6) presented a parallel string-searching algorithm. Since such methods are often used in a distributed setting, the cost of *communication* can become dominating. Sanders (§3.24) discussed several algorithms from this point of view.

Iacono (§3.15) and Mehlhorn (§3.18) reported on recent advances in the long-standing open problem of *dynamic optimality* of binary search trees (BSTs). The classic problem of finding *optimal* static *BSTs* was taken up by Munro (§3.21): it becomes significantly harder if the objective is to minimize the number of *binary* comparisons instead of the classic ternary comparisons.

Wild (§3.32) used the connection between BSTs and recursion trees of Quicksort to analyze Quicksort on inputs with equal keys, including multi-way partitioning Quicksort. The latter was discussed in detail by Aumüller (§3.3) who presented a novel analysis for comparison-optimal partitioning.

Neumann (§3.22) introduced a new randomized dictionary implementation based on jumplists. Kopelowitz (§3.17) showed a much simplified solution to the file-maintenance problem.

In the context of large sparse graphs, Andoni (§3.2), Fagerberg (§3.10), and Sun (§3.30) showed how to exploit special structure in the input for algorithmic applications. Pettie (§3.28) showed how to efficiently answer connectivity queries in graphs when vertices can be deleted.

The seminar also enjoyed contributions on new algorithms: two innovative applications of hashing were presented by Silvestri (§3.29) and Jacob (§3.16); Meyer auf der Heide (§3.20) applied the primal-dual approach for *online algorithms* to online leasing problems. Driemel (§3.7) reported on clustering methods for time series.

The theory-focused talks were complemented by broader perspectives from practice: Ajwani (§3.1) presented his vision for future communication tools that are supported by context-sensitive agents, and Sedgewick (§3.27) sketched his views on the future of higher education. Finally, Salinger (§3.26) summarized the approaches taken by SAP to include data-specific algorithms directly in their HANA database system.

New models of computation were also discussed. Owens (§3.23) explained how the architecture of graphic cards calls for different approaches to design data structures; Dütsch (§3.8) discussed the cost of virtual address translation in several algorithms. Finally, Farach-Colton (§3.11) and Graefe (§3.13) challenged the claim that data structures are independent of the application they are used in: they showed intriguing examples where the context a data structure was applied in entailed unforeseen additional requirements.

Final Thoughts

The organizers would like to thank the Dagstuhl team for their continuous support; the welcoming atmosphere made the seminar both highly productive and enjoyable. They also thank all participants for their contributions to this seminar.

2 Table of Contents

3 Overview of Talks

3.1 Towards fully-informed communication

Deepak Ajwani (Bell Labs – Dublin, IE)

License Creative Commons BY 3.0 Unported license
© Deepak Ajwani

I described a vision of a software cognitive layer that provides users with all the information they need at the time of communication. To realize such a vision, a communication platform should understand a user's communication, link it to other information available and mine the linked information in real-time. I presented a range of open problems related to graph algorithms and graph systems, to address these challenges.

3.2 Parallel Algorithms for Geometric Graph Problems

Alex Andoni, Grisha Yaroslavtsev, Krzysiek Onak, and Sasho Nikolov

License Creative Commons BY 3.0 Unported license
© Alex Andoni, Grisha Yaroslavtsev, Krzysiek Onak, and Sasho Nikolov

Main reference A. Andoni, A. Nikolov, K. Onak, G. Yaroslavtsev, "Parallel algorithms for geometric graph problems", in Proc. of the 46th Annual ACM Symp. on Theory of Computing (STOC'14), pp. 574–583, ACM, 2014.
URL http://dx.doi.org/10.1145/2591796.2591805

Motivated by modern parallel computing models (think MapReduce), we give a new algorithmic framework for geometric graph problems. Our framework applies to problems such as the Minimum Spanning Tree (MST) problem over a set of points in a low-dimensional space, which is useful for hierarchical agglomerative clustering. Our algorithm computes a $(1+\epsilon)$-approximate MST in a *constant* number of rounds of communication, while using total space and communication proportional to the size of the data only.

Our framework proves to have implications beyond the parallel models as well. For example, we consider the Earth-Mover Distance (EMD) problem, for which we obtain a new near-linear time algorithm as well as a first streaming algorithm (assuming we can pre-sort the points). Technically, our framework for EMD shows how to effectively break up a "big Linear Programming problem" into a number of "small Linear Programming problems," which can be later recombined using a dynamic programming approach.

3.3 News on Multi-Pivot Quicksort

Martin Aumüller (IT University of Copenhagen, DK)

License Creative Commons BY 3.0 Unported license
© Martin Aumüller

Joint work of Martin Aumüller; Martin Dietzfelbinger; Clemens Heuberger; Daniel Krenn; Helmut Prodinger
Main reference M. Aumüller, M. Dietzfelbinger, C. Heuberger, D. Krenn, H. Prodinger, "Counting Zeros in Random Walks on the Integers and Analysis of Optimal Dual-Pivot Quicksort", arXiv:1602.04031 [math.CO], 2016.
URL http://arxiv.org/abs/1602.04031

We discuss two results with respect to multi-pivot quicksort.

In the first part of this talk, we present an exact average case analysis of two variants of dual-pivot quicksort, one with a non-algorithmic comparison-optimal partitioning strategy, the other with a closely related algorithmic strategy. For both we calculate the expected number of comparisons exactly as well as asymptotically, in particular, we provide exact expressions for the linear, logarithmic, and constant terms. An essential step is the analysis of zeros of lattice paths in a certain probability model. Furthermore, we show that the closely related algorithmic strategy yields a comparison-optimal dual-pivot algorithm.

In the second part of this talk, I will talk about rearranging elements to produce a partition. A substantial part of the partitioning cost is caused by rearranging elements. A rigorous analysis of an algorithm for rearranging elements in the partitioning step is carried out, observing mainly how often array cells are accessed during partitioning. The algorithm behaves best if 3 or 5 pivots are used. Experiments show that this translates into good cache behavior and is closest to predicting observed running times of multi-pivot quicksort algorithms.

3.4 Thrill: Distributed Big Data Batch Processing in C++

Timo Bingmann (KIT – Karlsruher Institut für Technologie, DE)

License Ⓒ Creative Commons BY 3.0 Unported license
© Timo Bingmann

We present on-going work on a new distributed Big Data processing framework called Thrill. It is a C++ framework consisting of a set of basic scalable algorithmic primitives like mapping, reducing, sorting, merging, joining, and additional MPI-like collectives. This set of primitives goes beyond traditional Map/Reduce and can be combined into larger more complex algorithms, such as WordCount, PageRank, k-means clustering, and suffix sorting. These complex algorithms can then be run on very large inputs using a distributed computing cluster. Among the main design goals of Thrill is to lose very little performance when composing primitives such that small data types are well supported. Thrill thus raises the questions of a) how to design algorithms using the scalable primitives, b) whether additional primitives should be added, and c) if one can improve the existing ones using new ideas to reduce communication volume and latency.

3.5 External Memory Three-Sided Range Reporting and Top-k Queries with Sublogarithmic Updates

Gerth Stølting Brodal (Aarhus University, DK)

License Ⓒ Creative Commons BY 3.0 Unported license
© Gerth Stølting Brodal

Main reference G. Stølting Brodal, "External Memory Three-Sided Range Reporting and Top-k Queries with Sublogarithmic Updates", in Proc. of the 33rd Annual Symp. on Theoretical Aspects of Computer Science (STACS'16), LIPIcs, Vol. 47, pp. 23:1-23:14, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.
URL http://dx.doi.org/10.4230/LIPIcs.STACS.2016.23

An external memory data structure is presented for maintaining a dynamic set of N two-dimensional points under the insertion and deletion of points, and supporting unsorted 3-sided range reporting queries and top-k queries, where top-k queries report the k points with highest y-value within a given x-range. For any constant $0 < \varepsilon \leq \frac{1}{2}$, a data structure is constructed that supports updates in amortized $O(\frac{1}{\varepsilon B^{1-\varepsilon}} \log_B N)$ IOs and queries in amortized $O(\frac{1}{\varepsilon} \log_B N + K/B)$ IOs, where B is the external memory block size, and K is the size of the output to the query (for top-k queries K is the minimum of k and the number of points in the query interval). The data structure uses linear space. The update bound is a significant factor $B^{1-\varepsilon}$ improvement over the previous best update bounds for these two query problems, while staying within the same query and space bounds.

3.6 Parallel Queries

Andrej Brodnik (University of Primorska, SI)

License Ⓒ Creative Commons BY 3.0 Unported license
© Andrej Brodnik
Joint work of Andrej Brodnik; Tanja Štular; Matevž Jekovec
Main reference M. Jekovec, A. Brodnik, "Parallel Query in the Suffix Tree", arXiv:1509.06167 [cs.DS], 2015.
URL http://arxiv.org/abs/1509.06167

When considering parallel queries, the researchers in past were mostly concerned with parallel execution of several queries on the same data structure. In this contribution we ask ourselves how p processors can be employed to jointly perform a single search under CREW PRAM model.

The query we study is a search of pattern P in a text T, where $|P| = m$ and $|T| = n$. Our presentation starts with employing automaton based approach first and later evolve it into an index based. For the later we show, that one can perform search in time $O(m/p + \log p)$, deploying $O(m + m \log p)$ work and using $O(n^2)$ space. We are also able to change the solution to $O(m/p \log p)$ time, $O(m \log p)$ work and $O(n \log p)$ space.

The trivial lower is, of course, $\Omega(m/p)$ time, $\Omega(m)$ work and $\Omega(n)$ space. It remains an open question whether it is achievable. Mind though, that in our solution we pay a log factor for communication among the processors. Another interesting open question is how the scheme can be used for an approximate searching.

3.7 Clustering time series under the Frechet distance

Anne Driemel (TU Eindhoven, NL)

License Ⓒ Creative Commons BY 3.0 Unported license
© Anne Driemel
Joint work of Anne Driemel; Amer Krivosija; Christian Sohler
Main reference A. Driemel, A. Krivosija, C. Sohler, "Clustering time series under the Fréchet distance", in Proc. of the 27th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA'16), pp. 766–785, SIAM, 2016.
URL http://dx.doi.org/10.1137/1.9781611974331.ch55

The Frechet distance is a popular distance measure for curves. We study the problem of clustering time series under the Frechet distance. In particular, we give $(1+\varepsilon)$-approximation algorithms for variations of the following problem with parameters k and l. Given n univariate time series P, each of complexity at most m, we find k time series, not necessarily from P, which we call cluster centers and which each have complexity at most l, such that (a) the maximum distance of an element of P to its nearest cluster center or (b) the sum of these distances is minimized. Our algorithms have running time near-linear in the input size. To the best of our knowledge, our algorithms are the first clustering algorithms for the Frechet distance which achieve an approximation factor of $(1+\varepsilon)$ or better.

3.8 Algorithm Design Paradigms in the VAT-Model

Fabian Dütsch (Universität Münster, DE)

License Creative Commons BY 3.0 Unported license
© Fabian Dütsch

Recently, Jurkiewicz and Mehlhorn [ALENEX '13] observed that the cost of virtual address translation affects the practical runtime behavior of several fundamental algorithms on modern computers. In this talk, we extend their results to two dimensions and investigate the translation cost of some algorithm design paradigms. For this purpose, we analyze closest pair algorithms representing the divide and conquer, plane-sweep and randomized incremental construction paradigms in the VAT-model. Furthermore, we investigate the VAT-complexities of hashing and comparison-based searching. Finally, we verify the theoretical analyses by experimental results.

3.9 Sublinear Random Access Generators for Preferential Attachment Graphs

Guy Even (Tel Aviv University, IL)

License Creative Commons BY 3.0 Unported license
© Guy Even
Joint work of Reut Elvi; Guy Even; Moti Medina; Adi Rosen
Main reference G. Even, R. Levi, M. Medina, A. Rosen, "Sublinear Random Access Generators for Preferential Attachment Graphs", arXiv:1602.06159 [cs.DS], 2016.
URL http://arxiv.org/abs/1602.06159

We consider the problem of generating random graphs in evolving random graph models. In the standard approach, the whole graph is chosen randomly according to the distribution of the model before answering queries to the adjacency lists of the graph. Instead, we propose to answer queries by generating the graphs on-the-fly while respecting the probability space of the random graph model.

We focus on two random graph models: the Barabási-Albert Preferential Attachment model (BA-graphs) and the random recursive tree model. We present sublinear randomized generating algorithms for both models. Per query, the running time, the increase in space, and the number of random bits consumed are $\text{poly}\log(n)$ with probability $1 - 1/\text{poly}(n)$, where n denotes the number of vertices.

This result shows that, although the BA random graph model is defined sequentially, random access is possible without chronological evolution. In addition to a conceptual contribution, on-the-fly generation of random graphs can serve as a tool for simulating sublinear algorithms over large BA-graphs.

3.10 On Routing in Geometric Spanners

Rolf Fagerberg (University of Southern Denmark – Odense, DK)

License Creative Commons BY 3.0 Unported license
© Rolf Fagerberg
Joint work of Rolf Fagerberg; Jit Bose; André van Renssen; Sander Verdonschot
Main reference P. Bose, R. Fagerberg, A. van Renssen, S. Verdonschot, "Optimal Local Routing on Delaunay Triangulations Defined by Empty Equilateral Triangles", SIAM Journal on Computing, 44(6):1626–1649, 2015.
URL http://dx.doi.org/10.1137/140988103

A geometric spanner on a point set in the Euclidean plane is a straight-line graph on the point set in which any pair u, v of points has a path between them which is not more than a constant factor longer than the direct distance between u and v. The constant factor is called the spanning ratio. Such spanners have practical applications in e.g. add-hoc wireless networks, and can be seen as static data structures for approximate route finding in geometric graphs. Two classical constructions of geometric spanners are Yao$_k$-graphs and θ_k-graphs.

Recently, bounds on the spanning ratio of these graphs have evolved substantially. We give an overview of the current knowledge, and then focus on a result on the half-θ_6-graph which shows that while its spanning ratio is 2, actually following such a short path between u and v using local routing is only feasible in one direction, whereas in the other direction the factor becomes $5/\sqrt{3} = 2.886\ldots$. We do this by giving a lower bound and a routing algorithm matching this bound.

3.11 Migrating a data structure from one system to another

Martin Farach-Colton (Rutgers University – Piscataway, US)

License Creative Commons BY 3.0 Unported license
© Martin Farach-Colton
Joint work of Michael Bender; Martin Farach-Colton; Bradley Kuszmaul; Don Porter; Rob Johnson

Data structures are typically design independent of any particular use case. Sometimes, data structures get deployed in a system, during which they are optimized for the specific needs of the system. In this case, I discuss the experience of migrating a data structure from one system to another. Specifically, I discuss deploying an external-memory dictionary that has been optimized for a data base key-value store in a file system.

3.12 Practical Compact Indexes for Top-k Document Retrieval

Simon Gog (KIT – Karlsruher Institut für Technologie, DE), Gonzalo Navarro, and Roberto Konow

License Creative Commons BY 3.0 Unported license
© Simon Gog, Gonzalo Navarro, and Roberto Konow
Joint work of Simon Gog; Gonazlo Navarro; Roberto Konow

In this talk we present a fast and compact index for top-k document retrieval on general string collections. For a given string pattern P, the index returns the k documents where P occurs most often, i.e. we score by pattern frequency. We adapt a linear-space and optimal-time theoretical solution of Navarro and Nekrich [1], whose implementation poses various algorithm engineering challenges. While a naive implementation of the optimal solution is estimated to require around $80n$ bytes for a text collection of n symbols, we show how this can be improved to $2.5n - 3.0n$ bytes (including the text). The resulting index can still answer queries within microseconds and outperforms all previous work.

References

1 Gonzalo Navarro, Yakov Nekrich. *Top-k document retrieval in optimal time and linear space.* SODA 2012: 1066-1077

3.13 Lock-free data structures

Goetz Graefe (HP Labs – Madison, US)

License 🅭 Creative Commons BY 3.0 Unported license
© Goetz Graefe

I am trying to understand lock-free data structures, their rules and their limitations, and I appreciate the other attendees' help in grasping the fundamentals and the subtleties. This is part of a larger effort to understand optimistic and pessimistic concurrency control as well as transaction isolation levels as known in SQL databases.

3.14 Succinct Choice Dictionaries

Torben Hagerup (Universität Augsburg, DE) and Frank Kammer

License 🅭 Creative Commons BY 3.0 Unported license
© Torben Hagerup and Frank Kammer

The choice dictionary is introduced as a data structure that can be initialized with a parameter n in $\{1, 2, \ldots\}$ and subsequently maintains an initially empty subset S of $\{1, \ldots, n\}$ under insertion, deletion, membership queries and an operation choice that returns an arbitrary element of S. The choice dictionary appears to be fundamental in space-efficient computing. We show that there is a choice dictionary that can be initialized with n and an additional parameter t in $\{1, 2, \ldots\}$ and subsequently occupies $n + O(n(t/w)^t + \log n)$ bits of memory and executes each of the four operations insert, delete, contains (i.e., a membership query) and choice in $O(t)$ time on a word RAM with a word length of $w = \Omega(\log n)$ bits. In particular, with $w = \Theta(\log n)$, we can support insert, delete, contains and choice in constant time using $n + O(n/(\log n)^t)$ bits for arbitrary fixed t. We extend our results to maintaining several pairwise disjoint subsets of $\{1, \ldots, n\}$.

A static representation of a subset S of $\{1, \ldots, n\}$ that consists of $n + s$ bits $b_1, \ldots, b_{n+s}$ is called systematic if $b_l = 1 \iff l$ is in S for $l = 1, \ldots, n$ and is said to have redundancy s. We extend the former definition to dynamic data structures and prove that the minimum redundancy of a systematic choice dictionary with parameter n that executes every operation in $O(t)$ time on a w-bit word RAM is $\Theta(n/(tw))$. Allowing a redundancy of $\Theta(n \log(t \log n)/(t \log n) + n^{\varepsilon})$ for arbitrary fixed $\varepsilon > 0$, we can support additional $O(t)$-time operations p-rank and p-select that realize a bijection from S to $\{1, \ldots, |S|\}$ and its inverse. The bijection may be chosen arbitrarily by the data structure, but must remain fixed as long as S is not changed. In particular, an element of S can be drawn uniformly at random in constant time with a redundancy of $O(n \log\log n/\log n)$.

We study additional space-efficient data structures for subsets S of $\{1, \ldots, n\}$, including one that supports only insertion and an operation extract-choice that returns and deletes an arbitrary element of S. All our main data structures can be initialized in constant time and support efficient iteration over the set S, and we can allow changes to S while an iteration over S is in progress. We use these abilities crucially in designing the most space-efficient algorithms known for solving a number of graph and other combinatorial problems in linear time. In particular, given an undirected graph G with n vertices and m edges, we can output a spanning forest of G in $O(n + m)$ time with at most $(1 + \varepsilon)n$ bits for arbitrary fixed $\varepsilon > 0$, and if G is connected, we can output a shortest-path spanning tree of G rooted at a designated vertex in $O(n + m)$ time with $n \log_2 3 + O(n/(\log n)^t)$ bits for arbitrary fixed t in $\{1, 2, \ldots\}$.

3.15 Weighted dynamic finger in binary search trees

John Iacono (New York University, US)

License Creative Commons BY 3.0 Unported license
© John Iacono
Joint work of John Iacono; Stefan Langerman
Main reference J. Iacono, S. Langerman, "Weighted dynamic finger in binary search trees", in Proc. of the 27th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA'16), pp. 672–691, SIAM, 2016.
URL http://dx.doi.org/10.1137/1.9781611974331.ch49

It is shown that the online binary search tree data structure GreedyASS performs asymptotically as well on a sufficiently long sequence of searches as any static binary search tree where each search begins from the previous search (rather than the root). This bound is known to be equivalent to assigning each item i in the search tree a positive weight w_i and bounding the search cost of an item in the search sequence $s_1, \ldots, s_m$ by

$$O\left(1 + \log \frac{\sum_{\min(s_{i-1},s_i) \le x \le \max(s_{i-1},s_i)} w_x}{\min(w_{s_i}, w_{s_{i-1}})}\right)$$

amortized. This result is the strongest finger-type bound to be proven for binary search trees. By setting the weights to be equal, one observes that our bound implies the dynamic finger bound. Compared to the previous proof of the dynamic finger bound for Splay trees, our result is significantly shorter, stronger, simpler, and has reasonable constants.

3.16 Fast Output-Sensitive Matrix Multiplication

Riko Jacob (IT University of Copenhagen, DK)

License Creative Commons BY 3.0 Unported license
© Riko Jacob
Joint work of Riko Jacob; Mortten Stöckel
Main reference R. Jacob, M. Stöckel, "Fast Output-sensitive Matrix Multiplication", in Proc. of the 23rd Annual European Symp. on Algorithms (ESA'15), LNCS, Vol. 9294, pp. 766–778, Springer, 2015.
URL http://dx.doi.org/10.1007/978-3-662-48350-3_64

We consider the problem of multiplying two $U \times U$ matrices A and C of elements from a field $\mathbb{F}$. We present a new randomized algorithm that can use the known fast square matrix multiplication algorithms to perform fewer arithmetic operations than the current state of the art for output matrices that are sparse.

In particular, let ω be the best known constant such that two dense $U \times U$ matrices can be multiplied with $O(U^\omega)$ arithmetic operations. Further denote by N the number of nonzero entries in the input matrices while Z is the number of nonzero entries of matrix product AC. We present a new Monte Carlo algorithm that uses $\tilde{O}\left(U^2\left(\frac{Z}{U}\right)^{\omega-2}+N\right)$ arithmetic operations and outputs the nonzero entries of AC with high probability. For dense input, i.e., $N = U^2$, if Z is asymptotically larger than U, this improves over state of the art methods, and it is always at most $O(U^\omega)$. For general input density we improve upon state of the art when N is asymptotically larger than $U^{4-\omega}Z^{\omega-5/2}$.

The algorithm is based on dividing the input into "balanced" subproblems which are then compressed and computed. The new subroutine that computes a matrix product with balanced rows and columns in its output uses time $\tilde{O}\left(UZ^{(\omega-1)/2}+N\right)$ which is better than the current state of the art for balanced matrices when N is asymptotically larger than $UZ^{\omega/2-1}$, which always holds when $N = U^2$.

In the I/O model – where M is the memory size and B is the block size – our algorithm is the first nontrivial result that exploits cancellations and sparsity of the output. The I/O complexity of our algorithm is $\tilde{O}\left(U^2(Z/U)^{\omega-2}/(M^{\omega/2-1}B)+Z/B+N/B\right)$, which is asymptotically faster than the state of the art unless M is large.

3.17 File Maintenance: When in Doubt, Change the Layout!

Tsvi Kopelowitz (University of Michigan – Ann Arbor, US)

License 🅭 Creative Commons BY 3.0 Unported license
© Tsvi Kopelowitz
Joint work of Michael Bender; Jeremy Fineman; Seth Gilbert; Tsvi Kopelowitz; Pablo Montas

In this talk I will describe a new deamortized solution to the sequential-file-maintenance problem. The data structure uses several new tools, for solving this historically complicated problem. These tools include an unbalanced ternary-tree layout embedded in the sparse table, a level-based approach for triggering, and one-way rebalancing.

3.18 Self-Organizing Binary Search Trees: Recent Results

Kurt Mehlhorn

License 🅭 Creative Commons BY 3.0 Unported license
© Kurt Mehlhorn
Joint work of Parinya Chalermsook; Mayank Goswami; Lazlo Kosma; Kurt Mehlhorn; Thatchaphol Saranurak
Main reference P. Chalermsook, M. Goswami, L. Kozma, K. Mehlhorn, T. Saranurak, "Pattern-Avoiding Access in Binary Search Trees", in Proc. of the IEEE 56th Annual Symp. on Foundations of Computer Science (FOCS'15), pp. 410–423, IEEE CS, 2015.
URL http://dx.doi.org/10.1109/FOCS.2015.32

The dynamic optimality conjecture is perhaps the most fundamental open question about binary search trees (BST). It postulates the existence of an asymptotically optimal online BST, i.e. one that is constant factor competitive with any BST on any input access sequence. The two main candidates for dynamic optimality in the literature are splay trees [Sleator and Tarjan, 1985], and Greedy [Lucas, 1988; Munro, 2000; Demaine et al. 2009]. Despite BSTs being among the simplest data structures in computer science, and despite extensive effort over the past three decades, the conjecture remains elusive. Dynamic optimality is trivial for almost all sequences: the optimum access cost of most length-n sequences is $\Theta(n \log n)$, achievable by any balanced BST.

Thus, the obvious missing step towards the conjecture is an understanding of the “easy” access sequences. Preorder sequences (the access sequence arises from a preorder traversal of a tree) can easily be served in linear time by an off-line algorithms. No online BST is known to serve them in linear time.

We prove (FOCS 2015) two different relaxations of the traversal conjecture for Greedy: (i) Greedy with an arbitrary initial tree is almost linear for preorder sequences. (ii) Greedy with a fixed initial tree is in fact linear. These statements are corollaries of our more general results that express the complexity of access sequences in terms of a pattern avoidance.

Splay trees satisfy the so-called *access lemma*. Many of the nice properties of splay trees follow from it. *What makes self-adjusting binary search trees (BSTs) satisfy the access lemma?* In our ESA 2015 paper, we give sufficient conditions for the access lemma to hold and give strong hints of their necessity.

3.19 Generating Massive Scale-Free Networks under Resource Constraints

Ulrich Carsten Meyer (Goethe-Universität – Frankfurt a. M., DE) and Manuel Penschuck (Goethe-Universität – Frankfurt a. M., DE)

License Creative Commons BY 3.0 Unported license
© Ulrich Carsten Meyer and Manuel Penschuck
Main reference U. Meyer, M. Penschuck, “Generating Massive Scale-Free Networks under Resource Constraints”, in Proc. of the 18th Workshop on Algorithm Engineering and Experiments (ALENEX’16), pp. 39–52, SIAM, 2016.
URL http://dx.doi.org/10.1137/1.9781611974317.4

Random graphs as mathematical models of massive scale-free networks have recently become very popular. For experimental evaluation and in order to provide artificial data sets, huge instances of such networks actually need to be generated. We consider generation methods for random graph models based on linear preferential attachment under limited computational resources and investigate our techniques using the well-known Barabási-Albert (BA) graph model. We present the two I/O-efficient BA generators, MP-BA and TFP-BA, for the external-memory (EM) model and then extend MP-BA to massive parallelism based on but not limited to GPGPU.

3.20 Online Resource Leasing

Friedhelm Meyer auf der Heide (Universität Paderborn, DE)

License Creative Commons BY 3.0 Unported license
© Friedhelm Meyer auf der Heide
Main reference S. Abshoff , P. Kling, C. Markarian, F. Meyer auf der Heide, P. Pietrzyk, “Towards the price of leasing online”, Journal of Combinatorial Optimisation, pp. 1–20, 2015.
URL http://dx.doi.org/10.1007/s10878-015-9915-5

We consider online leasing problems in which demands arrive over time and need to be served by leased resources. Each resource can be leased for K different durations, each

incurring a different cost (longer leases cost less per time unit). This model is a natural generalization of Meyerson's Parking Permit Problem (FOCS 2005). In the talk, I review Meyerson's result and present it using the primal-dual approach. In addition, I present new online leasing variants of classical problems like facility location and set cover, and present primal-dual-based online algorithms for them together with their competitive analysis.

3.21 Optimal search trees with 2-way comparisons

Ian Munro (University of Waterloo, CA) and Mordecai Golin (HKUST – Kowloon, HK)

License Ⓒ Creative Commons BY 3.0 Unported license
© Ian Munro and Mordecai Golin
Joint work of Marek Chrobak; Mordecai Golin; J. Ian Munro; Neal E. Young

This talk is about finding a polynomial time algorithm that you probably thought was known almost a half century ago, but it wasn't. The polynomial time algorithm is still rather slow and requires a lot of space to solve, so we also look at extremely good and fast approximate solutions.

In 1971, Knuth gave an $O(n^2)$-time algorithm for the now classic problem of finding an optimal binary search tree. Knuth's algorithm works only for search trees based on 3-way comparisons, but most modern programming languages and computers support only 2-way comparisons ($<$, $=$ and $>$). Until this work, the problem of finding an optimal search tree using 2-way comparisons remained open – polynomial time algorithms were known only for restricted variants. We solve the general case, giving (i) an $O(n^4)$-time algorithm and (ii) a linear time algorithm that gives a tree with expected search cost within 2 comparisons of the optimal.

3.22 Randomized k-Jumplists

Elisabeth Neumann (TU Kaiserslautern, DE)

License Ⓒ Creative Commons BY 3.0 Unported license
© Elisabeth Neumann
Joint work of Elisabeth Neumann; Markus Nebel; Sebastian Wild

In this talk, I presented an extension of randomized jumplists, introduced by Brönnimann, Cazals and Durand. To improve search costs, the number of jump-pointers per node have been increased from one to k, ($k > 1$), to allow faster navigation through the list. Pointer targets are chosen at random such that the pointer structure is (strongly) nested and no two jump-pointers point to the same node. I presented algorithms for search, construction and insertion, extending algorithms for regular jumplists, all of which run in expected logarithmic time. Finally, I analysed the expected costs of searching and came to the conclusion that k-jumplists ($k > 1$) outperform regular jumplists if binary search is used to determine which pointer to follow during the search.

3.23 Dynamic Data Structures for the GPU

John D. Owens (University of California, Davis, US)

License Creative Commons BY 3.0 Unported license
© John D. Owens
Joint work of Martin Farach-Colton; John D. Owens

Today's GPU programming environments feature few general-purpose data structures. Only a handful of those can be *constructed* on the GPU, and to first order, none of them can be *updated* on the GPU. We aim to develop a family of GPU data structures that permit dynamic updates without rebuilding, and identify cross-cutting issues – e.g., modeling the memory hierarchy, leveraging task parallelism vs. cooperative parallelism, and choosing the right granularity of GPU parallelism for data-structure operations – that will affect their design.

3.24 Communication efficient algorithms

Peter Sanders (KIT – Karlsruher Institut für Technologie, DE)

License Creative Commons BY 3.0 Unported license
© Peter Sanders

I proposed to have a closer look at algorithms that have sublinear bottleneck communication volume. Examples were given for duplicate detection, distributed Bloom filters and various top-k problems. The talk is based on the two papers [1, 2].

References

1 Peter Sanders and Sebastian Schlag and Ingo Müller. *Communication Efficient Algorithms for Fundamental Big Data Problems.* IEEE Int. Conf. on Big Data. 2013.

2 Lorenz Hübschle-Schneider and Peter Sanders. *Communication Efficient Algorithms for Top-k Selection Problems.* IPDPS 2016.

3.25 Encoding Data Structures

Rajeev Raman (University of Leicester, GB)

License Creative Commons BY 3.0 Unported license
© Rajeev Raman
Main reference R. Raman, "Encoding Data Structures," in Proc. of the 9th Int'l Workshop on Algorithms and Computation (WALCOM'15), LNCS, Vol. 8973, pp. 1-7, Springer, 2015.
URL http://dx.doi.org/10.1007/978-3-319-15612-5_1

Note: Survey talk based on several papers.

Driven by the increasing need to analyze and search for complex patterns in very large data sets, the area of compressed and succinct data structures has grown rapidly in the last 10–15 years. Such data structures have very low memory requirements, allowing them to fit into the main memory of a computer, which in turn avoids expensive computation on hard disks.

This talk will focus on a topic that has become popular recently: encoding "the data structure" itself. Some data structuring problems involve supporting queries on data, but the queries that need to be supported do not allow the original data to be deduced from the queries. This presents opportunities to obtain space savings even when the data is incompressible, by pre-processing the data, extracting only the information needed to answer the queries, and then deleting the data. The minimum information needed to answer the queries is called the effective entropy of the problem: precisely determining the effective entropy can involve interesting combinatorics.

3.26 Towards a Web-scale Data Management Ecosystem Demonstrated by SAP HANA

Alejandro Salinger (SAP SE – Walldorf, DE)

License Creative Commons BY 3.0 Unported license
© Alejandro Salinger
Joint work of Franz Faerber; Jonathan Dees; Martin Weidner; Stefan Baeuerle; Wolfgang Lehner
Main reference F. Faerber, J. Dees, M. Weidner, S. Bäuerle, W. Lehner, "Towards a web-scale data management ecosystem demonstrated by SAP HANA", in Proc. of the IEEE 31st Int'l Conf. on Data Engineering (ICDE'15), pp. 1259–1267, 2015.
URL http://dx.doi.org/10.1109/ICDE.2015.7113374

The requirements for modern data management systems have changed in the last years mainly due to the growth in application space with different usage patterns, changes in underlying hardware, and growing data volumes. In this scenario, a solution must deal with a multidimensional problem space with multiple domain-specific data types, data consumption models, consistence notions, and query languages, among others. As no single engine can handle all the different dimensions, it is natural to tackle and optimize each dimension with specialized approaches. However, we argue for a deep integration of individual engines into a single coherent and consistent data management ecosystem that provides a common understanding of the overall business semantics.

We describe SAP HANA as an example of what such a holistic but also flexible data management ecosystem could look like. We describe the system's in-memory column store engine as well as its specialized engines that allow for data processing beyond relational data (e.g., time series, text search, graph), and we argue about the advantages of bringing data processing closer to the data itself.

We then describe HANA's Scale-Out Extension (SAP HANA Vora) with its low footprint, highly scalable processing engines as well as the system's integration with the Hadoop ecosystem. We give an example of the techniques to store and process large amounts of time series data such as compression based on the combination of several approximation methods with varying accuracy in the representation for different data warmness levels.

3.27 A 21st Century Model for Disseminating Knowledge

Robert Sedgewick (Princeton University, US)

License Creative Commons BY 3.0 Unported license
© Robert Sedgewick

In this talk, we describe a scalable model for teaching and learning based on a combination of studio-produced video lectures, a web repository of associated materials, and an authoritative classic textbook. The approach has already proven effective for teaching algorithms and data structures, the analysis of algorithms, and analytic combinatorics, and will be further tested in the coming year with a new computer science textbook and associated materials that can serve as a basis for a first course in computer science that can stand alongside traditional first courses in physics, chemistry, economics, and other disciplines.

3.28 Connectivity Oracles

Seth Pettie (University of Michigan, Ann Arbor, MI, US)

License Creative Commons BY 3.0 Unported license
© Seth Pettie
Joint work of Ran Duan; Seth Pettie
Main reference R. Duan and S. Pettie, "Connectivity oracles for failure prone graphs", in Proc. of the 42nd ACM Symp. on Theory of Computing (STOC'10), pp. 465–474, 2010.
URL http://dx.doi.org/10.1145/1806689.1806754

A d-failure connectivity oracle is a data structure for undirected graphs that can answer connectivity queries after any d vertices have been deleted. The best d-failure connectivity oracles that have fast query time either have exorbitant preprocessing or linear deletion time. In this talk I'll discuss a simplified variant of the Duan-Pettie (2010) d-failure connectivity oracle that has polynomial (in n) preprocessing, polynomial (in d) time for vertex deletion, and $O(d)$ time to answer a connectivity query. A new type of graph decomposition is used, which is inspired by the Fürer-Raghavachari algorithm for approximating the minimum-degree spanning tree.

3.29 I/O-Efficient Similarity Join

Francesco Silvestri (IT University of Copenhagen, DK)

License Creative Commons BY 3.0 Unported license
© Francesco Silvestri
Joint work of Rasmus Pagh; Ninh Pham; Francesco Silvestri; Morten Stöckel
Main reference R. Pagh, N. Pham, F. Silvestri, M. Stöckel, "I/O-Efficient Similarity Join", in Proc. of the 23rd Annual European Symp. on Algorithms (ESA'15), LNCS, Vol. 9294, pp. 941–952, Springer, 2015.
URL http://dx.doi.org/10.1007/978-3-662-48350-3_78

We present an I/O-efficient algorithm for computing similarity joins based on locality-sensitive hashing (LSH). In contrast to the filtering methods commonly suggested our method has provable sub-quadratic dependency on the data size. Further, in contrast to straightforward implementations of known LSH-based algorithms on external memory, our approach is able to take significant advantage of the available internal memory: Whereas the time complexity of classical algorithms includes a factor of M^ρ , where ρ is a parameter of the LSH used, the I/O complexity of our algorithm merely includes a factor $(N/M)^\rho$, where N is the data size and M is the size of internal memory. Our algorithm is randomized and outputs the correct result with high probability. It is a simple, recursive, cache-oblivious procedure, and we believe that it will be useful also in other computational settings such as parallel computation.

3.30 Fast construction of graph sparsification: graphs, ellipsoids, and balls-into-bins

He Sun (University of Bristol, GB)

License Creative Commons BY 3.0 Unported license
© He Sun
Joint work of Yin Tat Lee; He Sun
Main reference Y. Tat Lee, H. Sun, "Constructing Linear-Sized Spectral Sparsification in Almost-Linear Time", in Proc. of the IEEE 56th Annual Symp. on Foundations of Computer Science (FOCS'15), pp. 250–269, IEEE CS, 2015; pre-print available as arXiv:1508.03261 [cs.DS], 2015.
URL http://dx.doi.org/10.1109/FOCS.2015.24
URL http://arxiv.org/abs/1508.03261

Spectral sparsification is the procedure of approximating a graph by a sparse graph such that many properties between these two graphs are preserved. Over the past decade, spectral sparsification has become a standard tool in speeding up runtimes of algorithms for various combinatorial and learning problems.

In this talk I will present our recent work on constructing a linear-sized spectral sparsification in almost-linear time. In particular, I will discuss some interesting connections among graphs, ellipsoids, and balls-into-bins processes.

3.31 Revisiting the Construction of SSPDs in the Presence of Memory Hierarchies

Jan Vahrenhold (Universität Münster, DE)

License Creative Commons BY 3.0 Unported license
© Jan Vahrenhold
Joint work of Sylvie Temme; Jan Vahrenhold

We revisit the randomized internal-memory algorithm of Abam and Har-Peled [SoCG 2010] for constructing a semi-separated pair decomposition (SSPD) for N points in $\mathbb{R}^d$ in the context of the cache-oblivious model of computation. Their algorithm spends $O(n\varepsilon^{-d}\log_2 N)$ time (assuming that the floor function can be evaluated in constant time, $O(n\varepsilon^{-d}\log_2^2 N)$ time otherwise) in expectation and produces an SSPD of linear size in which each point participates in only a logarithmic number of pairs with high probability.

Using a modified analysis of their algorithm and several cache-oblivious techniques for tree construction, labeling, and traversal, we obtain a cache-oblivious algorithm that spends an expected number of $O(\text{sort}(n\varepsilon^{-d})\log_2 N)$ memory transfers.

3.32 Quicksort with Equal Keys

Sebastian Wild (TU Kaiserslautern, DE)

License Creative Commons BY 3.0 Unported license
© Sebastian Wild
Joint work of Martin Aumüller; Martin Dietzfelbinger; Conrado Martínez; Markus Nebel; Sebastian Wild

In this talk, I present the first analysis of generalized Quicksort on inputs with equal keys, confirming in part a conjecture of Sedgewick and Bentley.

I consider Quicksort variants which partition inputs into s segments, around $s-1$ pivots chosen as order statistics from a sample, generalizing on Quicksort variants used in practice.

The input model is random u-ary words, i. e., n elements chosen i. i. d. uniformly from $\{1, \dots, u\}$. Generalized Quicksort needs on average $\frac{\bar{a}}{\mathcal{H}} n \ln(u) + O(n)$ ternary comparisons to sort a random u-ary word of length n, provided that $u = \omega(1)$ and $u = O(n^{1/3-\varepsilon})$. The corresponding number for the classic model of random permutations is $\frac{\bar{a}}{\mathcal{H}} n \ln(n) + O(n)$, so the same relative speedup from sampling and multi-way partitioning is attained in both input models. Here, $\frac{\bar{a}}{\mathcal{H}}$ is a (known) constant that depends only on the used partitioning algorithm and the pivot-sampling scheme.

The analysis relies on the connection of Quicksort and search trees: I reduce the analysis of Quicksort to determining the path length in search trees built from inserting elements drawn i.i.d. from $\{1, \dots, u\}$.

4 Open problems

The seminar included an open problem session during which the following problems were discussed.

4.1 Open Problem 1

Deepak Ajwani (Bell Labs – Dublin, IE)

License 🅭 Creative Commons BY 3.0 Unported license
© Deepak Ajwani

The problem that I posed was the following: Given a directed graph G, find an acyclic subgraph D, such that TransitiveClosure(D) is as close to TransitiveClosure(G) as possible.

Since the transitive closure of D can only be a subset of transitive closure of G and it is a 0-1 matrix, the problem can also be formulated as "Given a directed graph G, find an acyclic subgraph D, such that TransitiveClosure(D) has as many ones as possible."

This is an interesting theoretical problem (particularly as the related problem of minimum feedback arc set is APX-hard). But my motivation for this problem came from a practical consideration, namely cleaning up crowd-sourced taxonomies. These taxonomies capture the specificity or generality of semantic concepts/categories and should logically not have directed cycles. Unfortunately, because they are created in a crowd-sourced way, they usually have thousands of cycles. So, the question is "can we remove the cycles while preserving the logical structure of the taxonomy as much as possible?"

The discussion in the session revolved around the correct formulation of this problem, particularly in going from "preserving the logical structure" to "maximising the transitive closure." In addition, I was asked if additional input related to number of different users creating an edge is available and I replied in negative. Also, I clarified that I am interested in good approximation solutions as well as heuristics that can deal with taxonomies that have hundreds of millions of edges.

4.2 Open Problem 2

Alejandro Lopez-Ortiz (University of Waterloo, CA)

License Ⓒ Creative Commons BY 3.0 Unported license
© Alejandro Lopez-Ortiz

My meta-problem is as follows:

Nowadays large data sets live in distributed NoSQL databases, often in main memory. A typical computation locks a view of the data (but not necessarily the data itself) that is at most a handful of operations old, until the transaction completes and the lock on the view is released.

This means that our current data structures and algorithms need to (1) be adapted to run in a distributed fashion (2) with as low as possible amounts of communication between nodes and (3) supporting limited, bounded persistence in the most efficient manner.

For some data structures and algorithms this can be achieved in a straightforward manner (e.g. BSTs, embarrassingly parallelizable algorithms), some others an efficient implementation requires new tools and last but not least, in some cases this might not be achievable. In this case a lower bound for the data structure/algorithm would be desirable, e.g. how expensive it is to recompute Dijkstra in a distributed, persistent setting with a small number of communication rounds.

4.3 Open Problem 3

Sebastian Wild (TU Kaiserslautern, DE)

License Ⓒ Creative Commons BY 3.0 Unported license
© Sebastian Wild

I posed the problem to compute or approximate the expected costs of the optimal alphabetic search tree for *random weights* on the leaves.

More precisely, assume that we draw $U_1, \ldots, U_{n-1}$ i. i. d. uniformly in $(0, 1)$ and denote by $D_1, \ldots, D_n$ the *spacings* between the sorted numbers, i. e., D_j is the difference of the jth smallest and the $(j-1)$st smallest of the U_i, where we add $U_0 = 0$ and $U_n = 1$. The resulting vector $(D_1, \ldots, D_n)$ is a stochastic vector, drawn uniformly from the closed $n-1$ dimensional simplex; it thus represents a uniformly chosen random probability distribution over the numbers $\{1, \ldots, n\}$. The vector $(D_1, \ldots, D_{n-1})$ is also said to have a *Dirichlet-distribution* with parameter $(1, \ldots, 1)$. We now construct the optimal binary search tree with leaf weights $D_1, \ldots, D_n$, and consider as cost of the tree $C = \sum_i D_i \cdot \mathrm{depth}(i\text{th leaf})$, i. e., the average leaf depth.

This problem is related to the analysis of comparison-optimal partitioning methods in Quicksort, which have been proposed by Aumüller and Dietzfelbinger [1]: The expected costs of the optimal alphabetic search tree are the leading-term coefficient of the expected number of comparisons of comparison-optimal partitioning.

The problem is also a natural information-theoretic question: How much can an alphabet with random letter access probabilities be compressed *on average* using an alphabetic prefix code (i. e., one that retains the order of symbols among code words; such codes are also known as *Hu-Tucker codes*). If we subtract from the average code word length C the (binary) entropy of $(D_1, \ldots, D_n)$, we obtain the *redundancy* of the code, $R = C - \mathcal{H}(D_1, \ldots, D_n) \geq 0$.

The problem is thus to determine the expected redundancy $\mathbb{E}[R]$, where the expectation is taken over the weights $D_1, \ldots, D_n$.

From the information-theoretic perspective, one could also ask for other coding schemes, such as *Huffman codes* or *Shannon codes*; results for Huffman codes yield an upper bound on Hu-Tucker codes. For Huffman codes, worst-case bounds on the redundancy are known for given symbol weights (see, e. g., [2] and the references therein). The bound depends only on the probability of the most probable symbol, but a precise computation of the expected value is still challenging.

References

1 M. Aumüller and M. Dietzfelbinger. "Optimal Partitioning for Dual-Pivot Quicksort", *ACM Transactions on Algorithms* 12:2 article 18, 2016. http://dx.doi.org/10.1145/2743020

2 Chunxuan Ye and R. W. Yeung. "A simple upper bound on the redundancy of Huffman codes", *IEEE Transactions on Information Theory* 48:7, 2132–2138, 2002. http://dx.doi.org/10.1109/TIT.2002.1013158

Participants

- Deepak Ajwani
Bell Labs – Dublin, IE
- Helmut Alt
FU Berlin, DE
- Alexandr Andoni
Columbia Univ. – New York, US
- Martin Aumüller
IT Univ. of Copenhagen, DK
- Timo Bingmann
KIT – Karlsruher Institut für Technologie, DE
- Gerth Stølting Brodal
Aarhus University, DK
- Andrej Brodnik
University of Primorska, SI
- Martin Dietzfelbinger
TU Ilmenau, DE
- Anne Driemel
TU Eindhoven, NL
- Fabian Dütsch
Universität Münster, DE
- Guy Even
Tel Aviv University, IL
- Rolf Fagerberg
University of Southern Denmark – Odense, DK
- Martin Farach-Colton
Rutgers Univ. – Piscataway, US
- Simon Gog
KIT – Karlsruher Institut für Technologie, DE
- Mordecai Golin
HKUST – Kowloon, HK
- Goetz Graefe
HP Labs – Madison, US
- Torben Hagerup
Universität Augsburg, DE
- Herman J. Haverkort
TU Eindhoven, NL
- John Iacono
New York University, US
- Riko Jacob
IT Univ. of Copenhagen, DK
- Tsvi Kopelowitz
University of Michigan – Ann Arbor, US
- Moshe Lewenstein
Bar-Ilan University – Ramat Gan, IL
- Alejandro Lopez-Ortiz
University of Waterloo, CA
- Jérémie Lumbroso
Princeton University, US
- Conrado Martinez
UPC – Barcelona, ES
- Kurt Mehlhorn
MPI für Informatik – Saarbrücken, DE
- Ulrich Carsten Meyer
Goethe-Universität – Frankfurt a. M., DE
- Friedhelm Meyer auf der Heide
Universität Paderborn, DE
- Ian Munro
University of Waterloo, CA
- Markus E. Nebel
TU Kaiserslautern, DE
- Elisabeth Neumann
TU Kaiserslautern, DE
- John D. Owens
Univ. of California, Davis, US
- Manuel Penschuck
Goethe-Universität – Frankfurt a. M., DE
- Seth Pettie
University of Michigan – Ann Arbor, US
- Rajeev Raman
University of Leicester, GB
- Alejandro Salinger
SAP SE – Walldorf, DE
- Peter Sanders
KIT – Karlsruher Institut für Technologie, DE
- Robert Sedgewick
Princeton University, US
- Francesco Silvestri
IT Univ. of Copenhagen, DK
- He Sun
University of Bristol, GB
- Jan Vahrenhold
Universität Münster, DE
- Sebastian Wild
TU Kaiserslautern, DE

Report from Dagstuhl Seminar 16111

Rethinking Experimental Methods in Computing

Edited by

Daniel Delling[1], Camil Demetrescu[2], David S. Johnson[3], and Jan Vitek[4]

1 **Apple Inc., Cupertino, US,** daniel.delling@live.com
2 **Sapienza University of Rome, IT,** demetres@dis.uniroma1.it
3 **Columbia University, New York, NY, US**
4 **Northeastern University, Boston, US,** j.vitek@neu.edu

Abstract

This report documents the talks and discussions at the Dagstuhl seminar 16111 "Rethinking Experimental Methods in Computing". The seminar brought together researchers from several computer science communities, including algorithm engineering, programming languages, information retrieval, high-performance computing, operations research, performance analysis, embedded systems, distributed systems, and software engineering. The main goals of the seminar were building a network of experimentalists and fostering a culture of sound quantitative experiments in computing. During the seminar, groups of participants have worked on distilling useful resources based on the collective experience gained in different communities and on planning actions to promote sound experimental methods and reproducibility efforts.

Seminar March 13–18, 2016 – http://www.dagstuhl.de/16111
1998 ACM Subject Classification D.2 Software Engineering, D.3 Programming Languages, E.1 Data Structures, F.2 Analysis of Algorithms and Problem Complexity, H.3 Information Storage and Retrieval, C.3 Special-Purpose and Application-based Systems, C.2 Computer-Communication Networks, C.4 Performance of Systems, B.8 Performance and Reliability
Keywords and phrases Algorithms, Benchmarks, Data sets, Experiments, Repeatability, Reproducibility, Software Artifacts, Statistics
Digital Object Identifier 10.4230/DagRep.6.3.24
Edited in cooperation with Emilio Coppa

1 Executive Summary

Emilio Coppa
Camil Demetrescu
Daniel Delling
Jan Vitek

License Creative Commons BY 3.0 Unported license
© Emilio Coppa, Camil Demetrescu, Daniel Delling, and Jan Vitek

This seminar is dedicated to the memory of our co-organiser and friend David Stifler Johnson, who played a major role in fostering a culture of experimental evaluation in computing and believed in the mission of this seminar. He will be deeply missed.

The pervasive application of computer programs in our modern society is raising fundamental questions about how software should be evaluated. Many communities in computer science and engineering rely on extensive experimental investigations to validate and gain insights on properties of algorithms, programs, or entire software suites spanning several layers of

Except where otherwise noted, content of this report is licensed
under a Creative Commons BY 3.0 Unported license

complex code. However, as a discipline in its infancy, computer science still lags behind other long-standing fields such as natural sciences, which have been relying on the scientific method for centuries.

There are several threats and pitfalls in conducting rigorous experimental studies that are specific to computing disciplines. For example, experiments are often hard to repeat because code has not been released, it relies on stacks of proprietary or legacy software, or the computer architecture on which the original experiments were conducted is outdated. Moreover, the influence of side-effects stemming from hardware architectural features are often much higher than anticipated by the people conducting the experiments. The rise of multi-core architectures and large-scale computing infrastructures, and the ever growing adoption of concurrent and parallel programming models have made reproducibility issues even more critical. Another major problem is that many experimental works are poorly performed, making it difficult to draw any informative conclusions, misdirecting research, and curtailing creativity.

Surprisingly, in spite of all the common issues, there has been little or no cooperation on experimental methodologies between different computer science communities, who know very little of each others efforts. The goal of this seminar was to build stronger links and collaborations between computer science sub-communities around the pivotal concept of experimental analysis of software. Also, the seminar allowed exchange between communities their different views on experiments. The main target communities of this seminar were algorithm engineering, programming languages, operations research, and software engineering, but also people from other communities were invited to share their experiences. Our overall goal was to come up with a common foundation on how to evaluate software in general, and how to reproduce results. Since computer science is a leap behind natural sciences when it comes to experiments, the ultimate goal of the seminar was to make a step forward towards reducing this gap. The format of the seminar alternated talks intended for a broad audience, discussion panels, and working sessions in groups.

Figure 1 David Stifler Johnson, 1945–2016.

The organisers would like to thank the Dagstuhl team and all the participants for making the seminar a success. A warm acknowledgement goes to Amer Diwan, Sebastian Fischmeister, Catherine McGeoch, Matthias Hauswirth, Peter Sweeney, and Dorothea Wagner for their constant support and enthusiasm.

2 Table of Contents

3 Overview of Talks

3.1 Soundness of Experiments in Parallel Computing

Umut A. Acar (Carnegie Mellon University – Pittsburgh, US)

License Creative Commons BY 3.0 Unported license
© Umut A. Acar

Recent advances in hardware such as the mainstream use of SMPs (multicore) computers, and large-scale data centers have brought parallelism back to the forefront of computing. Parallelism is now common in many different forms of hardware ranging from mobile phones to laptops and desktop computers. Unfortunately, writing performant parallel code can require low-level optimizations that make it extremely difficult to reproduce results and compare different approaches via standard empirical methodologies. In this talk, I will consider two specific problems–granularity control and locality–that can require such optimizations. As a solution, I propose the general idea of developing programming languages and systems that manage performance automatically for the programmer. This approach requires developing programming-language abstractions and algorithms for managing hardware resources in execution. As examples, I propose specific solutions to the two problems considered—granularity control and locality–and show some evidence that this approach can work well. (Joint work with Guy Blelloch, Arthur Chargueraud, Matthew Fluet, Stefan Muller, Ram Raghunathan, Mike Rainey)

3.2 How Did This Get Published? Pitfalls in Experimental Evaluation of Computing Systems

José Nelson Amaral (University of Alberta – Edmonton, CA)

License Creative Commons BY 3.0 Unported license
© José Nelson Amaral

This talk illustrates, through simple analogies, that summarizing either percentages or normalized quantities, such as speedups, using the arithmetic mean is always wrong when the quantities were normalized using different denominators. The talk then presents two alternative goals for summarization – latency or throughput – and presents suitable solutions for the summarization of several measurements in both cases.

3.3 What is the Value of the Data?

José Nelson Amaral (University of Alberta – Edmonton, CA)

License Creative Commons BY 3.0 Unported license
© José Nelson Amaral

In publications in Computing Science that report experimental results there is often lack of rigour and precision in the description of the experimental methodology and experimental setup, in the reporting and summarization of results and in the formulation of claims based on the experimental results. In this talk I put forward the notion that this state of affairs is, in a non-trivial portion, due to the low value of the experimental results. There are several reasons for the low value placed on published results. Experimental evaluations are difficult to reproduce because the required code and specifications are seldom available. The results of a given evaluation are very much dependent on variations in hardware operation and in configurations of the software stack. Benchmarks used for evaluation often do not reflect the programs Thus often, when an organization is interested in a technique described in the literature that organization puts very little weight on the experimental evaluation. Thus, there is a cycle where the low value placed in the experimental results gives little incentive for more rigorous experimental evaluation, which in turns keeps the valuation of the experimental results low.

3.4 Experimental Methodology in Parallel and Streaming Analytics

David A. Bader (Georgia Institute of Technology – Atlanta, US)

License Creative Commons BY 3.0 Unported license
© David A. Bader

Emerging real-world graph problems include: detecting community structure in large social networks; improving the resilience of the electric power grid; and detecting and preventing disease in human populations. Unlike traditional applications in computational science and engineering, solving these problems at scale often raises new challenges because of the sparsity and lack of locality in the data, the need for additional research on scalable algorithms and development of frameworks for solving these problems on high performance computers, and the need for improved models that also capture the noise and bias inherent in the torrential data streams. In this talk, the speaker will discuss experimental methodologies in massive data-intensive computing for applications in computational science and engineering.

3.5 The Importance of %: Why We Need to Think about Goals, Targets and Populations

Judith Bishop (Microsoft Research – Redmond, US)

License Creative Commons BY 3.0 Unported license
© Judith Bishop

Experiments frequently involve counting, and the absolute numbers are then measured over time (a graph that increases) or compared to other software that has similar characteristics (a bar chart). Sometimes, larger numbers are broken down into groups (a pie chart). Recently, there has been a push towards asking an additional question: what is the denominator? In other words, having measured a number, how does that relate to perfect? The result can be presented as a percentage. An example might be reporting the figure of 10,000 uses a month of a website. Immediately one can ask some more questions: geographically, is that for US students – well, a nice number – or for US consumers – not so good. If the denominator is known, then one can have more confidence regarding relative numbers. However, a chart showing usage of 10,000 and 15,000 by two companies, might still be missing by a million uses of a third company: the denominator reveals this hidden number. Nevertheless, it is extremely difficult to find denominators, even to estimate them. Emphasizing denominators is very important to industry where return on investment (ROI) is a factor in research. I shall give some examples and raise discussion on the issue. I'd be interested to know how the search for denominators fits into the other aims and expectations of the rest of the workshop.

3.6 Reproducibility in Computing: The Role of Professional Societies

Ronald F. Boisvert (NIST – Gaithersburg, US)

License Creative Commons BY 3.0 Unported license
© Ronald F. Boisvert

A scientific result is not fully established until it has been independently reproduced. Unfortunately, much published research is not independently verified. And, in the rare cases when a systematic effort has been made to do so, the results have not been encouraging. This threatens to undermine public confidence in the enterprise, and has led to calls for improvements to the process of reporting and reviewing scientific research in, for example, the biomedical sciences. The record in computing is not much better. Changing this state of affairs is not easy. Reproducing the work of others can be quite challenging, and does not garner the same credit as performing the original research. Professional societies have an important role in developing and promoting an open science ecosystem. As part of their role of arbiters and curators of the research literature, they can play a key role in changing the incentive structure to promote higher standards of reproducibility. In this talk I will describe some of the grassroots efforts being undertaken to improve the scientific process within the Association for Computing Machinery (ACM), the world's largest professional society in computing research. I will also describe steps within the ACM Publications Board being taken to support this.

3.7 Tools from Statistics, Machine Learning and Data Visualization for the Assessment of Heuristics for Optimization

Marco Chiarandini (University of Southern Denmark – Odense, DK)

License Creative Commons BY 3.0 Unported license
© Marco Chiarandini

The experimental analysis of algorithms within the engineering cycle may be conducted at different levels and with different goals. For example, we may be interested in describing behaviours for understanding and explaining algorithm execution or in modeling performance for prescribing the best algorithm configurations. Several tools from statistics and machine learning have been used to address these tasks. In the field of optimization heuristics, a machine learning approach seems, with good reasons, to be prevailing. New opportunities to make sense of data and improve presentation and reproducibility are offered by data-driven documents with interactive and dynamic graphics and by virutalized environments. In the talk, I will briefly go through these themes drawing examples from my experience as practitioner and reviewer.

3.8 Computing in the Cloud: Tools and Practices

Dmitry Duplyakin (University of Utah – Salt Lake City, US)

License 🅭 Creative Commons BY 3.0 Unported license
© Dmitry Duplyakin

In this talk, I will outline the current work on supporting experiments on CloudLab, an NSF-funded large-scale cloud testbeds. I will share our motivation for using a configuration management system for "orchestrating" software components and discuss scenarios in which such systems can benefit experimenters. I will also highlight some of the proposed work, both on the infrastructure side and on techniques for characterizing the available computational resources and better understanding of scientific applications.

3.9 Network Testbeds and Repeatable Research

Eric Eide (University of Utah – Salt Lake City, US)

License 🅭 Creative Commons BY 3.0 Unported license
© Eric Eide

The Flux Research Group at the University of Utah has developed and operated public network testbeds, such as Emulab and CloudLab, for more than fifteen years. Beyond providing realistic and highly configurable environments for a variety of computer-based experiments, one of the goals of these testbeds is to encourage repeatable research. In this talk, I will review the history of testbed development at Utah with focus on features that support repeatable research. In addition, I will discuss issues and opportunities for network testbeds to better support repeatable research in the future.

3.10 Experimentation and Replication in Embedded and Real-Time Systems

Sebastian Fischmeister (University of Waterloo, CA)

License 🅭 Creative Commons BY 3.0 Unported license
© Sebastian Fischmeister

Main reference A. Born de Oliveira, J.-C. Petkovich, T. Reidemeister, and S. Fischmeister, "DataMill: rigorous performance evaluation made easy", in Proc. of the 4th ACM/SPEC Intl. Conf. on Performance Engineering (ICPE '13), pp. 137–148, 2013.
URL http://dx.doi.org/10.1145/2479871.2479892

This talk will discuss the need for rigorous performance analysis and report results of a replication experiment conducted on the DataMill infrastructure. Based on the lessons learnt from building DataMill and running experiments, the talk then also discusses the challenges of performance analysis with specific focus on experimentation in embedded and heterogeneous computing systems.

3.11 The PRIMAD Model of Reproducibility: A Framework Model of Reproducibility (Result of Dagstuhl Seminar 16041)

Norbert Fuhr (Universität Duisburg-Essen, DE)

License Ⓒ Creative Commons BY 3.0 Unported license
© Norbert Fuhr

For describing different degrees of reproducibility, the participants of Dagstuhl Seminar 16041 started from a model referring to the components of an experiment: (R) the research goal, (M) the method proposed for achieving this goal, (I) the implementation of this method, (P) the platform on which the implementation is run, (D) the data (input + parameters) used in the experiment, and finally (A) the actor performing the experiment. When a researcher tries to reproduce an experiment, he should specify which components are changed, i.e. 'primed': R → R' repurpose for a new research goal, M → M': a new method, I → I': alternative implementation, P → P': different platform, D → D': other input/parameters. Finally, other important aspects of reproducibility are consistency of experimental results, and transparency, i.e. the ability to look into all necessary components to verify that the experiment does what it claims.

3.12 Lessons Learned from Shortest Path Algorithm Evaluation

Andrew V. Goldberg (Amazon.com, Inc. – Palo Alto, US)

License Ⓒ Creative Commons BY 3.0 Unported license
© Andrew V. Goldberg

This is a retrospective on experimental evaluation of shortest path algorithms. We discuss bridging the gap between theory in practice in the area that happened in 1990s. We discuss the general problem, with negative-length arcs and possible negative cycles, and the special cases: No negative cycles and no negative arcs.

3.13 The TIRA Experiment Platform

Matthias Hagen (Bauhaus-Universität Weimar, DE)

License Ⓒ Creative Commons BY 3.0 Unported license
© Matthias Hagen

The TIRA experimentation platform supports organisers of shared tasks in computer science to accept the submission of executable software and allows for reproducible experimental settings. Through virtualization techniques, participants in a shared task can directly work as they usually would on their own machines. TIRA also hosts the datasets used in a shared task with the option of keeping test data in a secure execution environment that protects them from leaking to the participants. Experimental results are displayed on a dedicated web page. Later reproducing of results or comparing to the state of the art for a given task becomes as easy as clicking a button.

3.14 Artifact Evaluation: Approach and Experience from OOPSLA's first AEC

Matthias Hauswirth (University of Lugano, CH)

License Creative Commons BY 3.0 Unported license
© Matthias Hauswirth

The programming languages community recently introduced so-called "artifact evaluation committees" to their conferences (PLDI, POPL, OOPSLA, ECOOP and others). Those AECs complement the traditional program committees by evaluating the artifacts underlying the papers. I will describe the idea behind AECs and will report on the experience of running the first two AECs at OOPSLA.

3.15 Incentives & Rewards

Matthias Hauswirth (University of Lugano, CH)

License Creative Commons BY 3.0 Unported license
© Matthias Hauswirth

We use the term "incentivize" quite often when thinking about solving certain problems or steering the community in certain ways. One such incentive is the AEC badge authors can place on their papers. While I designed and used that badge at OOPSLA, I claim that incentives don't usually work as intended. In this talk I will provide a brief glimpse into the evidence (in terms of arguments and experimental results) against using incentives in our situation, and in most situations we care about.

3.16 Rigorous Benchmarking in Reasonable Time

Tomas Kalibera (Northeastern University – Boston, US)

License Creative Commons BY 3.0 Unported license
© Tomas Kalibera

I speculate that the quality of experiments performed in systems/PL is often so low because some researchers believe they would have to run excessive amounts of experiments of benchmarks for excessive amount of time. We show how to adapt the experiment scale to a particular problem/system to keep the experimentation time reasonable, while not giving up on the rigor. We present the method on the example of DaCapo and SPEC CPU benchmarks. It is not that every time a new source of performance dependency (e.g. unix environment size, symbol naming at compile time, page allocation at process startup) is found, we have to multiply the size of all experiments we ever run.

3.17 Data Analysis for Performance Modeling

Catherine C. McGeoch (Amherst College, US)

License Ⓒ Creative Commons BY 3.0 Unported license
© Catherine C. McGeoch

This talk will discuss the problem of building an empirical performance model for algorithms: that is, developing a function that describes the relationship between time performance and the main factors that affect performance, including input, algorithm, and platform parameters. Contrasting advice as to choice of statistical and data analysis tools for this task may be found, between proponents of the Scientific Method (hypothesis testing, confirmatory statistics), and what has been called The New Experimentalism (exploratory data analysis). I will briefly survey these two schools of thought and how they apply to algorithmic performance modeling. To illustrate these points I will talk about some empirical surprises that arise when developing a performance model for D-Wave quantum annealing systems.

3.18 Chaos in Computer Performance

J. Eliot B. Moss (University of Massachusetts – Amherst, US)

License Ⓒ Creative Commons BY 3.0 Unported license
© J. Eliot B. Moss

This talk will first give a brief argument as to how any cache-like mechanism, and indeed most state machines, are subject to chaotic behavior. Whether and how chaos shows up depends on a program's interaction with the mechanism, however, and I will show some examples from cache behavior from SPEC CPU 2000 benchmarks to give a sense of this and close with suggestions of directions for future experimental methodology.

3.19 Assessing the Performance of Heuristics in Multiobjective Optimization: an Overview

Luís Paquete (University of Coimbra, PT)

License Ⓒ Creative Commons BY 3.0 Unported license
© Luís Paquete

Multiobjective optimization problems are usually hard to solve. Heuristics are often used to find an reasonably good approximation to the set of optimal solutions. The image of such approximation in the objective space consists of a set of points that contains a particular structure. Since most of these heuristic approaches are of stochastic nature, the sets of points produced may differ from run to run. This raises an interesting challenge: how to characterize and compare heuristics based on the sets of points produced in a collection of runs? In this talk, we give an overview of the main methods for assessing the performance of heuristics, from a solution quality perspective, for this class of optimization problems.

3.20 Algorithm Engineering: An Attempt at a Definition

Peter Sanders (KIT – Karlsruher Institut für Technologie, DE)

License 🅭 Creative Commons BY 3.0 Unported license
© Peter Sanders

This talk defines algorithm engineering as a general methodology for algorithmic research. The main process in this methodology is a cycle consisting of algorithm design, analysis, implementation and experimental evaluation that resembles Popper's scientific method. Important additional issues are realistic models, algorithm libraries, benchmarks with real-world problem instances, and a strong coupling to applications. Algorithm theory with its process of subsequent modeling, design, and analysis is not a competing approach to algorithmics but an important ingredient of algorithm engineering. At the end of the talk, we additionally discuss how algorithm engineering might help with interdisciplinary research in particular in grand challenge big data applications.

3.21 The Truth, the Whole Truth and Nothing but the Truth

Peter F. Sweeney (IBM TJ Watson Research Center – Yorktown Heights, US)

License 🅭 Creative Commons BY 3.0 Unported license
© Peter F. Sweeney

The EVALUATE 2011 workshop, co-located with PLDI, brought together members of the programming language community to discuss experimental evaluation. The outcome of this endeavor has resulted in "The Truth, the Whole Truth, and Nothing but the Truth: A Pragmatic Guide to Assessing Empirical Evaluations" that has been accepted for publication at TOPLAS. Specifically, an unsound claim can misdirect a field, encouraging the pursuit of unworthy ideas and the abandonment of promising ideas. An inadequate description of a claim can make it difficult to reason about the claim, for example to determine whether the claim is sound. Many practitioners will acknowledge the threat of unsound claims or inadequate descriptions of claims to their field. We believe that this situation is exacerbated and even encouraged by the lack of a systematic approach to exploring, exposing, and addressing the source of unsound claims and poor exposition. This paper proposes a framework that identifies three sins of reasoning that lead to unsound claims and two sins of exposition that lead to poorly described claims. Sins of exposition obfuscate the objective of determining whether or not a claim is sound, while sins of reasoning lead directly to unsound claims. Our framework provides practitioners with a principled way of critiquing the integrity of their own work and the work of others. We hope that this will help individuals conduct better science and encourage a cultural shift in our research community to identify and promulgate sound claims.

3.22 Experimenting with Humans vs. Experimenting with Machines

Walter F. Tichy (KIT – Karlsruher Institut für Technologie, DE)

License Ⓒ Creative Commons BY 3.0 Unported license
© Walter F. Tichy

Experiments in computing are done either with or without human participants. I will contrast the experimental protocols. Benchmarks can often substitute for human subjects. Analyzing recorded data does not normally qualify as a controlled experiment, as indpendent variables cannot be varied systematically; thus sich studies can show correlation (which can be used for prediction) but not causation. Exeriments with humans, as for instance in HCI or software engineering, are time-consuming, expensive, and high-risk. Often reviewers do not understand the difficulties of such experiemnts and reject them out of hand for minor imperfections.

3.23 I Think Nobody Wants to Do Bad Science!

Petr Tuma (Charles University – Prague, CZ)

License Ⓒ Creative Commons BY 3.0 Unported license
© Petr Tuma

This is a very short talk based on the initial seminar survey results, which aims to pose some questions not explicitly discussed in the survey responses. The questions revolve around our ability to cover all the technical intricacies of experiments, balancing the costs and benefits of performing robust experiments, and more generally finding ways of accepting the limits of experimental results.

3.24 Some remarks on data sharing and the replication of results

Dorothea Wagner (KIT – Karlsruher Institut für Technologie, DE)

License Ⓒ Creative Commons BY 3.0 Unported license
© Dorothea Wagner

During the last 5 to 10 years, national and international science and research funding organization started a discussion on principles of good scientific practice with regard to the replication of research results. According agreements clearly state that primary research data should be shared openly and promptly. In many scientific disciplines, we can observe intensive common efforts to support "open data in science". In this talk I want to raise the question if our community is doing enough in this respect. How can we support and intensify the collection of data (like benchmark data), and the reproducibility and comparability of experiments?

3.25 Experimenting with Innocent Humans

Roger Wattenhofer (ETH Zürich, CH)

License 🅭 Creative Commons BY 3.0 Unported license
© Roger Wattenhofer

A random sentence from a random paper: "We tested our software with 7 subjects". The 7 PhD students in your lab, probably... Anyway, a few years ago we started testing some of our software with thousands of innocent users. In short this talk will present a few lessons we learned. We will discuss the ethical parameters of such massive user studies.

4 Working groups

4.1 Educating the community

Umut A. Acar (Carnegie Mellon University – Pittsburgh, US), José Nelson Amaral (University of Alberta – Edmonton, CA), David A. Bader (Georgia Institute of Technology – Atlanta, US), Judith Bishop (Microsoft Research – Redmond, US), Ronald F. Boisvert (NIST – Gaithersburg, US), Marco Chiarandini (University of Southern Denmark – Odense, DK), Markus Chimani (Universität Osnabrück, DE), Daniel Delling (Apple Inc. – Cupertino, US), Camil Demetrescu (Sapienza University of Rome, IT), Amer Diwan (Google – San Francisco, US), Dmitry Duplyakin (University of Utah – Salt Lake City, US), Eric Eide (University of Utah – Salt Lake City, US), Erik Ernst (Google – Aarhus, DK), Sebastian Fischmeister (University of Waterloo, CA), Norbert Fuhr (Universität Duisburg-Essen, DE), Paolo G. Giarrusso (Universität Tübingen, DE), Andrew V. Goldberg (Amazon.com, Inc. – Palo Alto, US), Matthias Hagen (Bauhaus-Universität Weimar, DE), Matthias Hauswirth (University of Lugano, CH), Benjamin Hiller (Konrad-Zuse-Zentrum – Berlin, DE), Richard Jones (University of Kent – Canterbury, GB), Tomas Kalibera (Northeastern University – Boston, US), Marco Lübbecke (RWTH Aachen, DE), Catherine C. McGeoch (Amherst College, US), Kurt Mehlhorn (MPI für Informatik – Saarbrücken, DE), J. Eliot B. Moss (University of Massachusetts – Amherst, US), Ian Munro (University of Waterloo, CA), Petra Mutzel (TU Dortmund, DE), Luís Paquete (University of Coimbra, PT), Mauricio Resende (Amazon.com, Inc. – Seattle, US), Peter Sanders (KIT – Karlsruher Institut für Technologie, DE), Nodari Sitchinava (University of Hawaii at Manoa – Honolulu, US), Peter F. Sweeney (IBM TJ Watson Research Center – Yorktown Heights, US), Walter F. Tichy (KIT – Karlsruher Institut für Technologie, DE), Petr Tuma (Charles University – Prague, CZ), Dorothea Wagner (KIT – Karlsruher Institut für Technologie, DE), and Roger Wattenhofer (ETH Zürich, CH)

License 🅭 Creative Commons BY 3.0 Unported license
© Umut A. Acar, José Nelson Amaral, David A. Bader, Judith Bishop, Ronald F. Boisvert, Marco Chiarandini, Markus Chimani, Daniel Delling, Camil Demetrescu, Amer Diwan, Dmitry Duplyakin, Eric Eide, Erik Ernst, Sebastian Fischmeister, Norbert Fuhr, Paolo G. Giarrusso, Andrew V. Goldberg, Matthias Hagen, Matthias Hauswirth, Benjamin Hiller, Richard Jones, Tomas Kalibera, Marco Lübbecke, Catherine C. McGeoch, Kurt Mehlhorn, J. Eliot B. Moss, Ian Munro, Petra Mutzel, Luís Paquete, Mauricio Resende, Peter Sanders, Nodari Sitchinava, Peter F. Sweeney, Walter F. Tichy, Petr Tuma, Dorothea Wagner, and Roger Wattenhofer

One of the key challenges discussed during the seminar was how to educate the community to care and recognise the pitfalls and the benefits of conducting sound experiments. After a common discussion, we formed three subgroups, which worked on: (i) guidelines for authors and reviewers in writing and assessing experimental work, (ii) a Wikipedia page on experimental methods in computing, and (iii) resources on software experimental methods for students.

4.1.1 Great Papers: a reviewer's guide to evaluating experimental research in computer science

The main goal of this working group was distilling some key aspects that characterise good quantitative experimental work in computer science. In particular, the group identified six lines that authors and reviewers are encouraged to consider in writing and assessing experimental work in computing:

1. **Experimental context.** Great papers have: clearly specified goals, scope, and research questions matching the claims.
2. **Experimental design.** Great papers have: clear description of the methodology, which matches the stated goals of the experiments and encourages reproduction; well chosen baselines, competing approaches, and benchmarks; consistent comparison to previous experimental work; independent and control variables identified that are most important to the stated goals; attention paid to possible hidden variables.
3. **Conduct of the experiment.** Great papers have: a clearly stated procedure for collecting data; metrics and measurement procedures that match experimental goals; sufficient repetitions of trials in cases of non-deterministic or random outcomes.
4. **Analysis.** Great papers: use appropriate statistical procedures in terms of the data, its distribution, and the experimental goals; expose unusual distribution properties (including skew, bimodality, and outliers); acknowledge and attempt to explain anomalous or missing data.
5. **Presentation of results.** Great papers have: clear and insightful presentation that addresses the stated goals of the work; careful choice of aggregate and summary statistics; effective use of visualisation techniques; accompanying text that describes figures and tables.
6. **Interpretation of results.** Great papers have: claims that are clearly justified by the data and analysis; consideration of alternative causes for the observations; explicit separation between justified claims and generalisations beyond the scope of the experiment.

The group plans to extend the list with concrete examples and reach out to program chairs of conferences and journal editors to refine and tailor the list to specific sub-communities.

References

1 Preliminary Guidelines for Empirical Research in Software Engineering.
`http://evaluate.inf.usi.ch/node/30`
2 How to Write a Scientific Evaluation Paper.
`http://evaluate.inf.usi.ch/node/54`

4.1.2 Wikipedia page on Experimental methods in computing

A second goals of this working group was to start writing a Wikipedia page on *Experimental methods in computing.* An initial draft of this page can be found at
`https://en.wikipedia.org/wiki/Draft:Experimental_methods_in_computing`.

4.1.3 How to produce sound quantitative research: information for students

This working group has also produced a summary document providing useful resources for new graduate students and young researchers. An on-going version of the document is available at http://tinyurl.com/j6cbghz.

4.2 Evangelism

Mauricio Resende (Amazon.com, Inc. – Seattle, US), David A. Bader (Georgia Institute of Technology – Atlanta, US), Ronald F. Boisvert (NIST – Gaithersburg, US), Catherine C. McGeoch (Amherst College, US), J. Eliot B. Moss (University of Massachusetts – Amherst, US), and Dorothea Wagner (KIT – Karlsruher Institut für Technologie, DE)

License Creative Commons BY 3.0 Unported license
© Mauricio Resende, David A. Bader, Ronald F. Boisvert, Catherine C. McGeoch, J. Eliot B. Moss, and Dorothea Wagner

The main goal of this working group was to consider how best to publicize the work of Dagstuhl Seminar 16111 and how to facilitate uptake in the larger community. A list of the main ideas follows:

1. Contributions to *CRA Best Practices Memos*, *Informatics in Europe* or to a stand-alone collection. Announce presence with targeted emails. Topic ideas:
   - How to review an experimental paper in subfield *X*.
   - How to add artifact submission and evaluation to your conference or journal submission process.
   - Advice for tenure committees on reviewing experimental work in computer science.
   - How to integrate experimental projects in your classroom.
   - How to build and maintain a living benchmark repository.
   - How to run a DIMACS Challenge.
2. Ensure continuity of DIMACS Challenge series and evaluate if there should be challenges in other areas.
3. Talk to people who organize summer schools and people who might teach summer schools, about an experimental methodology course.
4. Contact conference and journal steering committees, to propose that they consider initiating procedures for artifact submission and evaluation.
5. Put together a short list of best papers in experimental methodology.
6. Talk to organisers about a follow-up of this meeting, at Dagstuhl or FCRC.
7. Write letters (position papers) to key players in CS research. Introduce experimental methodology/support for artifacts; explain why it is important; highlight key points; suggest that the recipient creates incentives and support mechanisms; say how to find out more information. The group suggested two white papers, addressing different topics for different (but overlapping) focus groups:
   a. Position paper on research methodology.
   b. Position paper on artifact evaluation and publishing.
8. Contact steering committees of appropriate meetings, to propose that they consider experimentalists when selecting plenary speakers.
9. Propose speakers to ACM speaker series or to Sigma Xi speaker series.

10. Start an online ask-the-experimentalist resource, e.g., in StackOverflow, ResearchGate, for questions about experimental methodology and statistics/data analysis. Identify people willing to monitor and answer the questions.
11. Find someone to write a regular column in a discipline-wide venue with topics in experimental methodology, or start a blog.
12. Teach a MOOC on experimental methods in computer science. It could be taught cooperatively (several professors) with standalone 'units' in different areas.
13. Suggest special issues on empirical methodology to appropriate journal Editors-in-Chief.

4.3 Replicability

Petr Tuma (Charles University – Prague, CZ), Umut A. Acar (Carnegie Mellon University – Pittsburgh, US), Judith Bishop (Microsoft Research – Redmond, US), Ronald F. Boisvert (NIST – Gaithersburg, US), Amer Diwan (Google – San Francisco, US), Dmitry Duplyakin (University of Utah – Salt Lake City, US), Eric Eide (University of Utah – Salt Lake City, US), Norbert Fuhr (Universität Duisburg-Essen, DE), Matthias Hagen (Bauhaus-Universität Weimar, DE), J. Eliot B. Moss (University of Massachusetts – Amherst, US), and Peter F. Sweeney (IBM TJ Watson Research Center – Yorktown Heights, US)

License 🅭 Creative Commons BY 3.0 Unported license
© Petr Tuma, Umut A. Acar, Judith Bishop, Ronald F. Boisvert, Amer Diwan, Dmitry Duplyakin, Eric Eide, Norbert Fuhr, Matthias Hagen, J. Eliot B. Moss, and Peter F. Sweeney

4.3.1 Guidelines

As a first activity, the group identified a set of guidelines to be given to a team doing a reproducibility study:

1. *What is a reproducibility study?*
   A reproducibility study attempts to confirm independently some (not necessarily all) important claim(s) made in a preceding paper; it may also attempt to provide more insight. Such a study should be carried out by a different team, using a different apparatus, in a different location. A different implementation of the same algorithm may be used, additional benchmarks or inputs, additional metrics and/or additional statistical analysis.
2. Motivation: *Why do a reproducibility study?*
   A reproducibility study serves to increase the confidence in reported scientific results by confirming (or refuting) the claims of the original study in changed settings. A reproducibility study may also contribute to the discipline methodologically.
3. *What issues should a reproducibility study address?*
   - Motivation: Why did you choose this work to reproduce? Argue why this is an interesting and important paper to reproduce.
   - What claim or claims (implicit or explicit) in the original paper are you reproducing?
   - What additional or broader claims (if any) are you making in this study?
   - What artifacts (software, hardware, data, etc.) did you use from the original paper? How did you audit their correctness?
   - What challenges did you face in doing this study and how did you surmount those challenges?
   - What interaction did you have with the original researchers? (Your evaluation should be arguably independent of the original study.)
   - What did you change in your experiment compared to the original evaluation?

- Do your claims support or refute the claims in the original paper? If they refute the claims, please explain the likely cause, if possible.
- What additional insights did you gain? (Remember, the goal is not to find fault but to increase insight!)
- What recommendations do you make (if any) in the light of your experience?

4. Experimental methodology. A reproducibility study should:
   - Deal with nondeterminism: multiple runs: look at distribution, statistics to aggregate; i.e. average stdev
   - Analyze uncertainty: systematic and random effects
   - Apply appropriate statistical methods on raw data
   - Investigate any failures to support the original claims. Are their hidden variables?
5. *How is a reproducibility study different from an artifact evaluation?*
   An artifact evaluation only tries to repeat the original results using apparatus and conditions as similar as possible to the original, within the time constraints of the review process for a scientific publication. Specifically, an artifact evaluation checks that the artifact is consistent with the original paper, complete, well documented and easy to reuse. A reproducibility study has broader goals, intentionally works with different apparatus (such as a different computer system, possibly different software, etc.), produces and checks results, and does not have similar time constraints.

4.3.2 A suggested format of a study

As a second contribution, the group discussed possible formats that may be suggested for a reproducibility study, which can be summarised as follows:

1. Synopsis of original idea: motivation for choosing this work.
2. Description of original evaluation.
3. Description of reproducibility evaluation:
   - What artifacts from original work have been used.
   - How experimental system has changed:
     a. New benchmarks.
     b. New metrics.
     c. New experimental setup.
     d. New analysis of raw data.
4. Results
   - Replicability success?
   - Reproducibility success on different dimensions of change.
   - Graphs.
5. Lessons learnt: positives and negatives; what was easy, what was hard.

4.3.3 Letter to journal editors

Following up on the activities of Section 4.2, the group drafted a letter for editors-in-chief of journals to propose the idea of devoting an issue to selected reproducibility results:

Dear Editor,

How do we know that an idea in a paper is effective? Many papers include evaluations to determine the effectiveness of an idea; however, evaluations are difficult to do. For example, if an evaluation fails to consider some important variable, it may produce invalid results. Even if an evaluation is valid it is likely severely limited in scope (e.g., applies only to a benchmark suite or to a particular programming language) and thus of limited use.

The traditional method for improving the confidence in an idea or to extend the generality of an idea is to do reproducibility studies: some group other than the original group tests the idea in a different context from the original evaluation (e.g., using different benchmarks or hardware). If the reproduction is successful (i.e., the results are compatible with the original results) we gain confidence in the effectiveness of the idea and also generalize the original results. If the reproduction is unsuccessful, we discover that the idea has limited applicability or even that the claim in the original paper is invalid. A great reproduction study provides deep insights into an important idea from prior work.

Unfortunately, Computer Science does not have a tradition of reproducibility studies. We would like to change that.

Our proposal is for several journals (each in a different area of Computer Science) to reserve and advertise one or more reserved slots for reproducibility papers once a year (we propose the first issue of each year, but can certainly adjust this based on the responses we get from the journals). This paper would be held to the same standards as any other paper in the journal however (i) submissions for this slot would have a deadline; (ii) the journal editors would prioritize the review of these submissions so that they are ready in time for the next slot; and (iii) the reviewers would be instructed to evaluate the paper as a reproduction paper. Of course, if there is no reproduction submission that meets the journal's standards, the journal would use the slot for a regular paper.

By having multiple top-tier journals publishing reproducibility papers, the journals will send a clear message that such papers are valued contributions to top publication venues. In doing so, the journals will promote the identification of effective ideas and thus improve upon the scientific rigor in Computer Science.

Our team (attendees of the Dagstuhl Rethinking Experimental Methods in Computing Seminar) will (i) help advertise these slots widely; and (ii) help identify authors of reproducibility studies (and of course authors can simply submit in response to our advertisement). If you need help finding reviewers for a reproduction study, we can help to identify reviewers. The final decision to accept the paper, of course, lies in the hands of the journal editors.

Participants

- Umut A. Acar
Carnegie Mellon University – Pittsburgh, US
- José Nelson Amaral
University of Alberta – Edmonton, CA
- David A. Bader
Georgia Institute of Technology – Atlanta, US
- Judith Bishop
Microsoft Res. – Redmond, US
- Ronald F. Boisvert
NIST – Gaithersburg, US
- Marco Chiarandini
University of Southern Denmark – Odense, DK
- Markus Chimani
Universität Osnabrück, DE
- Emilio Coppa
Sapienza University of Rome, IT
- Daniel Delling
Apple Inc. – Cupertino, US
- Camil Demetrescu
Sapienza University of Rome, IT
- Amer Diwan
Google – San Francisco, US
- Dmitry Duplyakin
University of Utah – Salt Lake City, US
- Eric Eide
University of Utah – Salt Lake City, US
- Erik Ernst
Google – Aarhus, DK
- Sebastian Fischmeister
University of Waterloo, CA
- Norbert Fuhr
Universität Duisburg-Essen, DE
- Paolo G. Giarrusso
Universität Tübingen, DE
- Andrew V. Goldberg
Amazon.com, Inc. – Palo Alto, US
- Matthias Hagen
Bauhaus-Universität Weimar, DE
- Matthias Hauswirth
University of Lugano, CH
- Benjamin Hiller
Konrad-Zuse-Zentrum – Berlin, DE
- Richard Jones
Univ. of Kent – Canterbury, GB
- Tomas Kalibera
Northeastern University – Boston, US
- Marco Lübbecke
RWTH Aachen, DE
- Catherine C. McGeoch
Amherst College, US
- Kurt Mehlhorn
MPI für Informatik – Saarbrücken, DE
- J. Eliot B. Moss
University of Massachusetts – Amherst, US
- Ian Munro
University of Waterloo, CA
- Petra Mutzel
TU Dortmund, DE
- Luís Paquete
University of Coimbra, PT
- Mauricio Resende
Amazon.com, Inc. – Seattle, US
- Peter Sanders
KIT – Karlsruher Institut für Technologie, DE
- Nodari Sitchinava
University of Hawaii at Manoa – Honolulu, US
- Peter F. Sweeney
IBM TJ Watson Research Center – Yorktown Heights, US
- Walter F. Tichy
KIT – Karlsruher Institut für Technologie, DE
- Petr Tuma
Charles University – Prague, CZ
- Dorothea Wagner
KIT – Karlsruher Institut für Technologie, DE
- Roger Wattenhofer
ETH Zürich, CH

Report from Dagstuhl Seminar 16112

From Theory to Practice of Algebraic Effects and Handlers

Edited by

Andrej Bauer[1], Martin Hofmann[2], Matija Pretnar[3], and Jeremy Yallop[4]

1 **University of Ljubljana, SI,** `andrej.bauer@fmf.uni-lj.si`
2 **LMU München, DE,** `hofmann@ifi.lmu.de`
3 **University of Ljubljana, SI,** `matija.pretnar@fmf.uni-lj.si`
4 **University of Cambridge, GB,** `jeremy.yallop@cl.cam.ac.uk`

Abstract

Dagstuhl Seminar 16112 was devoted to research in algebraic effects and handlers, a chapter in the principles of programming languages which addresses computational effects (such as I/O, state, exceptions, nondeterminism, and many others). The speakers and the working groups covered a range of topics, including comparisons between various control mechanisms (handlers vs. delimited control), implementation of an effect system for OCaml, compilation techniques for algebraic effects and handlers, and implementations of effects in Haskell.

Seminar March 13–18, 2016 – http://www.dagstuhl.de/16112
1998 ACM Subject Classification D.3 Programming Languages, D.3.3 Language Constructs and Features: Control structures, Polymorphism, F.3 Logics and Meanings of Programs, F.3.1 Specifying and Verifying and Reasoning about Programs, F.3.2 Semantics of Programming Languages, F.3.3 Studies of Program Constructs: Control primitives, Type structure
Keywords and phrases algebraic effects, computational effects, handlers, implementation techniques, programming languages
Digital Object Identifier 10.4230/DagRep.6.3.44
Edited in cooperation with Niels F. W. Voorneveld and Philipp G. Haselwarter

1 Executive Summary

Andrej Bauer
Martin Hofmann
Matija Pretnar
Jeremy Yallop

License Creative Commons BY 3.0 Unported license
© Andrej Bauer, Martin Hofmann, Matija Pretnar, Jeremy Yallop

Being no strangers to the Dagstuhl seminars we were delighted to get the opportunity to organize Seminar 16112. Our seminar was dedicated to algebraic effects and handlers, a research topic in programming languages which has received much attention in the past decade. There are strong theoretical and practical aspects of algebraic effects and handlers, so we invited people from both camps. It would have been easy to run the seminar as a series of disconnected talks that would take up most of people's schedules – we have all been to such seminars – and run the risk of disconnecting the camps as well. We decided to try a different format, and would like to share our experience in this executive summary.

On the first day we set out to identify topics of interest and organize working groups around them. This did not work, as everybody wanted to be in every group, or was at least

Except where otherwise noted, content of this report is licensed under a Creative Commons BY 3.0 Unported license

worried they would miss something important by choosing the wrong group. Nevertheless, we did identify topics and within them ideas began to form. At first they were very general ideas on the level of major research projects, but soon enough people started asking specific questions that could be addressed at the seminar. Around those questions small groups began to form. Out of initial confusion came self-organization.

We had talks each day in the morning, with the schedule planned two days ahead, except for the first day which started by a tutorial on algebraic effects and handlers. We left the afternoons completely free for people to work in self-organized groups, which they did. The organizers subtly made sure that everybody had a group to talk to. In the evening, just before dinner, we had a "show & tell" session in which groups reported on their progress. These sessions were the most interesting part of the day, with everyone participating: some showing what they had done so far, and others offering new ideas. Some of the sessions were accompanied by improvised short lectures.

Work continued after dinner and late at night. One of the organizers was shocked to find, on his way to bed, that the walls of a small seminar room were completely filled with type theoretic formulas, from the floor to the ceiling. He was greatly relieved to hear that the type theory was not there to stay permanently as the Dagstuhl caretakers painted the walls with a special "whiteboard" paint. They should sell the paint by the bucket as a Dagstuhl souvenir.

We are extremely happy with the outcome of the seminar and the way we organized it. An open format that gives everyone ample time outside the seminar room was significantly boosted by the unique Dagstuhl environment free of worldly distractions. We encourage future organizers to boldly try new ways of organizing meetings. There will be confusion at first, but as long as the participants are encouraged and allowed to group themselves, they will do so. If a lesson is to be taken from our seminar, it is perhaps this: let people do what they want, but also make sure they report frequently on what they are doing, preferably when they are a bit hungry.

2 Table of Contents

3 Overview of Talks

3.1 Handlers considered harmful?

Andrzej Filinski (University of Copenhagen, DK)

License Ⓒ Creative Commons BY 3.0 Unported license
© Andrzej Filinski

At a seminar about handlers for algebraic effects – often presented as an operationally oriented alternative to more explicitly monad-based approaches for specifying and implementing computational effects – it is important not to forget about some of the considerable theoretical and practical strengths of monads. This talk outlines some areas in which effect handlers – as currently conceived in languages like Eff – may show comparative weaknesses, and is meant to inspire reflection and discussion on how those can best be addressed.

3.2 Andromeda: Type theory with Equality Reflection

Philipp G. Haselwarter (University of Ljubljana, SI)

License Ⓒ Creative Commons BY 3.0 Unported license
© Philipp G. Haselwarter
Joint work of Andrej Bauer; Gaëtan Gilbert; Philipp G. Haselwarter; Matija Pretnar; Christopher A. Stone
Main reference http://andromedans.github.io/andromeda/
URL https://github.com/Andromedans/andromeda/releases/tag/dagstuhl-2016

We present Andromeda, a proof assistant for dependent type theory with equality reflection in the style of [1]. The design of Andromeda follows the tradition of Edinburgh LCF, in the sense that

- there is an abstract datatype of type-theoretic judgements whose values can only be constructed by a small, trusted nucleus
- the user interacts with the nucleus by writing programs in a high-level Andromeda meta-language (AML)

The type theory of Andromeda has dependent products and equality types. The rules for products are standard and include function extensionality. This flavour of dependent type theory is very expressive, as it allows one to postulate new *judgemental equalities* through the equality reflection rule. However, this comes at the expense of rendering type-checking undecidable. As there is no complete type-checking algorithm that we could implement in the nucleus, we rely on user code written in AML to prove complex equality judgements.

We demonstrate how we use effects and handlers as a mechanism for the nucleus to communicate with the user-code by asking questions about equalities. We then showcase how equality reflection can be used to introduce inductive types with their judgemental computation rules and to control opaqueness of definitions. Finally, we present how a meta-language with effects can be used to implement a memoization tactic.

References

1 Per Martin-Löf. *Intuitionistic Type Theory.* Bibliopolis, 1984.

3.3 No value restriction is needed for algebraic effects and handlers

Ohad Kammar (University of Cambridge, GB), Sean Moss, and Matija Pretnar (University of Ljubljana, SI)

License Creative Commons BY 3.0 Unported license
© Ohad Kammar, Sean Moss, and Matija Pretnar

We present a straightforward sound Hindley-Milner polymorphic type system for algebraic effects and handlers which allows type variable generalisation of arbitrary computations, and not just values. This result is surprising. On the one hand, the soundness of Hindley-Milner polymorphism is known to fail when not restricted in the presence of computational effects such as reference cells and continuations. On the other hand, many programming examples can be recast to use effect handlers instead of these effects. We place this result in the wider context in two ways. First, we discuss the expressive difference between reference cells and programming with algebraic effects and handlers. Second, we present a parametric set-theoretic denotational semantics that highlights the smooth interaction of algebraic effects and polymorphism.

3.4 Parameterized Extensible Effects and Session Types

Oleg Kiselyov (Tohoku University – Sendai, JP)

License Creative Commons BY 3.0 Unported license
© Oleg Kiselyov

Parameterized monad goes beyond monads in letting us represent type-state. An effect executed by a computation may change the set of effects it may be allowed to do afterwards. We describe how to easily 'add' and 'subtract' such type-state effects. Parameterized monad is often used to implement session types. We point out that extensible type-state effects are themselves a form of session types.

3.5 Adequacy for Infinitary Algebraic Effects

Gordon Plotkin (University of Edinburgh, GB)

License Creative Commons BY 3.0 Unported license
© Gordon Plotkin

Moggi famously proposed a monadic account of computational effects which includes the computational λ-calculus, a core call-by-value functional programming language. One naturally then seeks a correspondingly general treatment of operational semantics. In the algebraic theory of effects, a refinement of Moggi's theory, the effects are obtained by appropriate operations, and the monad is generated from an equational theory over these operations.

In a previous paper with John Power, a general adequacy theorem was given for the case of monads generated by finitary operations. This covers examples such as probabilistic nondeterminism and exceptions. The idea is to evaluate terms symbolically in the absolutely free algebra with the same signature as the equational theory. Without recursion, the evaluated terms are finite; with recursion, they may be infinitely deep.

In general, however, one needs infinitary operations, for example for interactive I/O. We review the previous work and show it can be extended to include such operations by allowing infinitely wide terms. We can also define a general contextual equivalence for any monad, however an extensional characterisation is elusive. The work should be extended to cover handlers.

In most cases the natural adequacy theorem for a given effect is directly obtained from the symbolical one. An exception is state, as the symbolic operational semantics has no state component. It remains an interesting question to give a general operational semantics with a notion of state.

3.6 A tutorial on algebraic effects and handlers

Matija Pretnar (University of Ljubljana, SI)

License Creative Commons BY 3.0 Unported license
© Matija Pretnar
Main reference M. Pretnar, "An Introduction to Algebraic Effects and Handlers. Invited tutorial paper", Proc. of the 31st Conference on the Mathematical Foundations of Programming Semantics (MFPS XXXI), Electr. Notes Theor. Comput. Sci., Vol. 319, pp. 19–35, 2015.
URL http://dx.doi.org/10.1016/j.entcs.2015.12.003

The seminar started with a tutorial, which had a two-fold purpose of establishing a common terminology and of introducing algebraic effects and handlers to anyone not yet familiar with them. Roughly half of the audience was familiar with algebraic effects, but everyone was well versed in functional programming and computational effects.

In the tutorial, we first looked at the basic idea of algebraic effects: every computation returns a value or performs an effect by calling an operation. Therefore, the effectful behaviour can be captured in an algebraic theory comprising a set of basic operations and equations between them. We have shown how this leads to an interpretation of computations with trees that have called operations as branching points and returned values as leaves. This furthermore results in an algebraic denotational semantics, where computations are interpreted with free models of the aforementioned algebraic theory.

Next, we have looked at how one may generalize exception handlers to handlers of other algebraic effects, and the subtleties involved in the generalisation. Using many simple examples of input & output handlers, we explored the flexibility that handlers offer in managing the control flow of programs. As a more involved example, we took a look at how one may implement many variants of backtracking with a handler for the non-deterministic choice operation. We have also revisited the algebraic semantics and seen how handlers correspond exactly to the homomorphisms, induced by the universal property of the free model.

Finally, we sketched how one may adapt a standard type system for a call-by-value language into a type & effect system, which captures the set of potentially called operations in addition to the type of returned values.

3.7 Compiling Eff to OCaml

Matija Pretnar (University of Ljubljana, SI), Amr Hany Shehata Saleh (KU Leuven, BE), and Tom Schrijvers (KU Leuven, BE)

License Creative Commons BY 3.0 Unported license
© Matija Pretnar, Amr Hany Shehata Saleh, and Tom Schrijvers

We introduce a compilation technique for Eff, a functional language with handlers of algebraic effects. Our compiler converts an Eff program into an OCaml program that produces an element of the free monad. In order to reduce the performance overhead of the generated code, we introduce a number of optimizations.

The most crucial technique, when feasible, is to translate pure computations into direct OCaml code. For example, an Eff computation `1 + 3`, is first translated into

```
(fun x -> Return (fun y -> Return (x + y))) 1 >>= fun f ->
f 3
```

where `Return` and `>>=` are the unit and binding operation of the free monad, and `+` is native addition in OCaml. However, monadic binds are costly, so our desire is to optimize the generated code to just `Return (1 + 3)`, which we do through a series of rewriting rules.

According to our benchmarks, the optimized generated code performs at about half the speed of hand-written OCaml code. We plan to use the information provided by an effect system to further optimize the output code.

3.8 Effect Handlers in Scope

Tom Schrijvers (KU Leuven, BE)

License Creative Commons BY 3.0 Unported license
© Tom Schrijvers

Joint work of Nicolas Wu; Tom Schrijvers; Ralf Hinze
Main reference N. Wu, T. Schrijvers, R. Hinze, "Effect handlers in scope", in Proc. of the 2014 ACM SIGPLAN Symp. on Haskell (Haskell'14), pp. 1–12, ACM, 2014.
URL http://dx.doi.org/10.1145/2775050.2633358

Algebraic effect handlers are a powerful means for describing effectful computations. They provide a lightweight and orthogonal technique to define and compose the syntax and semantics of different effects. The semantics is captured by handlers, which are functions that transform syntax trees. Unfortunately, the approach does not support syntax for scoping constructs, which arise in a number of scenarios. While handlers can be used to provide a limited form of scope, we demonstrate that this approach constrains the possible interactions of effects and rules out some desired semantics. This paper presents two different ways to capture scoped constructs in syntax, and shows how to achieve different semantics by reordering handlers. The first approach expresses scopes using the existing algebraic handlers framework, but has some limitations. The problem is fully solved in the second approach where we introduce higher-order syntax.

3.9 Compositional reasoning for algebraic effects

Alex Simpson (University of Ljubljana, SI)

License 🅭 Creative Commons BY 3.0 Unported license
© Alex Simpson

We obtain compositional proof systems for program verification by combining a set of generic rules, common to all language instantiations, with composition principles that must be supplied on an effect-specific basis. The proposed framework considers effects as generated by a signature of algebraic operations. The effect-specific composition principles then replace the customary equations (which are derivable from them).

3.10 Substitution, jumps and algebraic effects

Sam Staton (University of Oxford, GB)

License 🅭 Creative Commons BY 3.0 Unported license
© Sam Staton
Joint work of Marcelo Fiore; Sam Staton
Main reference M. Fiore and S. Staton, "Substitution, jumps, and algebraic effects", in Proc. of the Joint Meeting of the 23rd EACSL Annual Conf. on Computer Science Logic and the 29th Annual ACM/IEEE Symp. on Logic in Computer Science (CSL-LICS'14), 2014; pre-print available from author's webpage.
URL http://dx.doi.org/10.1145/2603088.2603163
URL http://www.cs.ox.ac.uk/people/samuel.staton/papers/lics2014-substitution.pdf

I spoke about the relationship between jumps and the theory of substitution. To give an algebra for the theory of substitution is to give a first-order algebraic theory. I discussed how this explains the implementation of algebraic effects using control effects.

3.11 LiquidHaskell: Refinement Types for Haskell

Niki Vazou (University of California – San Diego, US)

License 🅭 Creative Commons BY 3.0 Unported license
© Niki Vazou
Joint work of Alexander Bakst; Eric Seidel; Ranjit Jhala; Niki Vazou
Main reference N. Vazou, A. Bakst, R. Jhala, "Bounded refinement types", in Proc. of the 20th ACM SIGPLAN Int'l Conf. on Functional Programming (ICFP'15), pp. 48–61, ACM, 2015; pre-print available from author's webpage.
URL http://dx.doi.org/10.1145/2784731.2784745
URL http://goto.ucsd.edu/~nvazou/icfp15/main.pdf

We saw LiquidHaskell, a decidable and highly automated verifier that uses refinement types for Haskell source code. I presented some examples (including safety of division and list sorting) that can be found in the online demo [1].

Also we saw how refinement types can be extended with bounds [2] leading to more expressive specifications that can be used to specify and verify effectual computations.

References

1 Online demo: http://goto.ucsd.edu/~nvazou/compose16/_site/01-index.html
2 Niki Vazou, Alexander Bakst, Ranjit Jhala. *Bounded refinement types.* ICFP 2015. http://goto.ucsd.edu/~nvazou/icfp15/main.pdf

4 Working groups

4.1 Towards an effect system for OCaml

Matija Pretnar (University of Ljubljana, SI), Stephen Dolan (University of Cambridge, GB), KC Sivaramakrishnan (University of Cambridge, GB), and Leo White (Jane Street – London, GB)

License Creative Commons BY 3.0 Unported license
© Matija Pretnar, Stephen Dolan, KC Sivaramakrishnan, and Leo White

With the introduction of algebraic effects to OCaml[1], extending OCaml's type system into a type & effect system is a natural next step. In such a system, programs receive a type $A!\mathcal{E}$, where A is the type of returned values, and $\mathcal{E}$ is the effect annotation, whose exact form is yet to be determined. Even though there is already an existing polymorphic effect system for handlers with an inference algorithm [3], it is not obvious how to include it in OCaml due to backwards compatibility.

There are a number of properties that a feasible effect system should satisfy:

Soundness If a program e receives a type $A!\mathcal{E}$, every potential effect `E` should be captured in $\mathcal{E}$.

Usefulness An effect system that annotates each program with every possible effect there is, is obviously sound, but not very useful. Thus, an effect information should not mention an effect that is guaranteed not to happen.

Backwards compatibility We want each program that was typable before introducing effect annotations, to remain typable. Furthermore, the effect system should play along nicely with OCaml's module system, thus whole-program analysis is out of the question.

To see what the above properties imply, take a program

```
if X then perform E1 else perform E2
```

The effect information of `perform E1` must mention `E1` for the sake of soundness, but omit `E2` for the sake of usefulness. Conversely, the effect information of `perform E2` should mention `E2` but not `E1`. But the whole program must remain typable due to backwards compatibility, and its type should mention both `E1` and `E2` due to soundness. From this, it follows that the effect system needs to provide a way of enlarging effect information. There are two established ways of providing this flexibility: subtyping [4] or row polymorphism [1]. Both are difficult to apply directly to OCaml, due to already-existing language features:

Monomorphic types The ML type system makes a distinction between monomorphic and polymorphic types, and in certain contexts only monomorphic types are permitted. Many existing programs are typeable only because, say, `int` $\rightarrow$ `int` is monomorphic, and would break if it became a polymorphic type.

Signature matching Comparing a module implementation against its interface requires not only inferring polymorphic types, but checking whether a given polymorphic type is more polymorphic than another.

Invariant contexts While OCaml supports (explicit) subtyping, not all type parameters are either co- or contra-variant. For instance, the type parameters to `ref`, the indices of GADTs, and unannotated abstract types are neither co- nor contra-variant.

Subtyping makes type inference difficult by breaking unification, so the usual approach is to infer *constrained types* of the form $A|\mathcal{C}$, where $\mathcal{C}$ is the set of constraints between

[1] https://github.com/ocamllabs/ocaml-effects

type (and later also effect) parameters in A [2]. However, there are a number of practical problems. First, it is hard to determine when a constrained type $A|\mathcal{C}$ is an instance of $A'|\mathcal{C}'$, causing problems for compatibility with the module system. Next, constraint generation in the inference algorithm needs to be directed in order to keep track of covariance and contravariance. This causes problems with the current inference algorithm of OCaml, which mostly works with equations and is undirected. Finally, constraints are cumbersome to write and difficult to read, decreasing chances of adoption in the programming community.

A possible solution for subtyping is to encode constraints in types, potentially dropping some of them, which results in types that satisfy a weak form of principality: the inferred type is unique and captures most of possible typings of the given program, but not all of them.

For row polymorphism, typability of existing programs poses a problem. These programs, which may cause any effect provided by OCaml (input/output, references, . . .), should receive an annotation, say `IO`, that distinguishes them from pure programs. Furthermore, existing monomorphic types should remain monomorphic. For example, a function `old_fun` that used to have a type `unit` → `unit` should get a type `unit` → (`unit!IO`). However, one then cannot type the program `if X then old_fun () else perform E`, as the type of the left branch does not contain a row variable and cannot be expanded to mention `E`.

A possible solution for this issue is to give monomorphic types to existing monomorphic programs, but allow a limited form of subeffecting, which weakens the effect annotation during application. Then, for example, `old_fun` would have a type `unit` → (`unit!IO`), but its application `old_fun ()` would get the type `unit`!$[$`IO`$|\rho]$.

References

1 Daan Leijen. Koka: Programming with row polymorphic effect types. In *MSFP*, volume 153 of *EPTCS*, pages 100–126, 2014.

2 François Pottier. Type inference in the presence of subtyping: from theory to practice. Technical Report RR-3483, INRIA, 1998.

3 Matija Pretnar. Inferring algebraic effects. *Logical Methods in Computer Science*, 10(3), 2014.

4 Keith Wansbrough and Simon L. Peyton Jones. Once upon a polymorphic type. In *POPL*, pages 15–28. ACM, 1999.

5 Open problems

5.1 Are all functions continuous and how to prove it?

Andrej Bauer (University of Ljubljana, SI)

License 🅭 Creative Commons BY 3.0 Unported license
© Andrej Bauer

5.1.1 Mathematical background

Brouwer's statement *"all functions are continuous"* can be formulated without reference to topology as follows. A functional $f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ is *continuous* at $a : \mathbb{N} \to \mathbb{N}$ when there exists $m : \mathbb{N}$ such that, for all $b : \mathbb{N} \to \mathbb{N}$, if $\forall k < m, ak = bk$ then $fa = fb$. This says that the value of fa depends only on the initial segment $a\,0$, $a\,1$, ..., $a\,(m-1)$.

The statement *"all functionals are continuous everywhere"* is valid in various models of intuitionistic mathematics, such as Kleene's number realizability and Kleene's function realizability. We can ask whether the statement is realized in any given functional programming language. Such a realizer is called a *modulus of continuity* and is a functional $\mu : ((\mathbb{N} \to \mathbb{N}) \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ such that, for all $f : (\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ and $a, b : \mathbb{N} \to \mathbb{N}$, if $\forall k < \mu f a \,.\, a\,k = b\,k$ then $fa = fb$. Essentially, $\mu f a$ computes how much of a is needed to compute fa.

5.1.2 Implementing the modulus of continuity

It is impossible to implement μ in PCF and Haskell. Briefly, every hereditarily total functional definable in PCF is extensional (one can use Ulrich Berger's theory of totality [2] to establish this), while a result by Troelstra [3, §9.6.10–9.6.11] shows that an extensional modulus of continuity violates choice principles that are realized by PCF.

Therefore, we necessarily need additional computational features that let μ inspect the workings of f. Here are a few attempts, where we pretend that the type of integers is the type of natural numbers (we ignore negative values).

ML with references

Consider ML with references (and no other features). Then a possible μ is

```
let mu_ref f a =
  let k = ref 0 in
  let a’ n = (k := max !k n; a n) in
    f a’ ; !k
```

However:
1. Can f use its own local references? If it can use them in an unrestricted way then it can break `mu_ref`. How do we reasonably restrict the use of local references by f?
2. More generally, how do we formulate the exact preconditions on f and a?
3. What is the theorem that needs to be proved, and how is it proved?

ML with exceptions

With exceptions (and no other features) we can do it as follows:

```
exception Abort

let mu_exc f a =
  let rec search k =
    try
      let a’ n = (if n < k then a n else raise Abort) in
        f a’ ; k
    with Abort -> search (k+1)
  in
    search 0
```

However:
1. What if f catches `Abort`? May it do so? What is the exact precondition on f?
2. Would local exceptions help? If so, can f use its own local exceptions?

Other setups

1. In Haskell we could do everything inside a fixed monad. This is still not entirely easy, even if we figure out what it means for f to be "pure".
2. Moving to a total language is probably helpful. However, keep in mind that μ does not exist in pure λ-calculus, so straight Agda or some such system is out of the question.
3. Other effects can be used to implement a candidate μ, but it seems like they should be *local* (local references, local exceptions, delimited control) or else f has access to them.

5.1.3 Open problem

At first sight it seems that the above implementations of μ work, but as soon as we try to formulate exactly what it is that we want to prove, it becomes clear that not everything is clear, so the first problem is:

> *Explain what it means to realize "all functions are continuous" in a realizability model based on a programming language with computational effects.*

One has to find a good notion of a realizer that uses effects in a "benign way". For instance, asking for purity in the sense of [1] seems too restrictive. Once it is clear what problem we are trying to solve, we may attempt to prove that the modulus is really there:

> *Identify computational effects which allow realization of the modulus of continuity, and prove* rigorously *that the realizer works.*

Attacking the problem ought to improve our ability to argue about higher-type computation in the presence of computational effects.

References

1 Andrej Bauer, Martin Hofmann and Aleksandr Karbyshev. On Monadic Parametricity of Second-Order Functionals. *Foundations of Software Science and Computation Structures – 16th International Conference, FOSSACS 2013*, 225–240, 2013.

2 Ulrich Berger. Computability and Totality in Domains. *Mathematical Structures in Computer Science* 12(3), 281–294, 2002.

3 Anne Troelstra and Dirk van Dalen. Constructivism in mathematics, volume 2. Elsevier, 1988.

5.2 Capturing algebraic equations in an effect system

Matija Pretnar (University of Ljubljana, SI)

License 🅭 Creative Commons BY 3.0 Unported license
© Matija Pretnar

Equational theories

The main premise of algebraic effects is that effects can be described with an equational theory consisting of a set of operations and equations between them [7]. For example, non-determinism can be described by an operation `choose` and three equations stating its idempotency, commutativity and associativity. Computations returning values from X are then interpreted as elements of the *free model* of such a theory.

Issues with interpreting handlers

Handlers of algebraic effects, which assign a handling term for each operation, can be interpreted as homomorphisms from the free model to some other (not necessary free) model of the same theory [8]. However, there are computationally interesting handlers that do not respect all of the expected equations. One is a handler that collects all possible results of a non-deterministic computation in a list. This respects the associativity, but not the idempotency or commutativity of `choose`. Similarly, a state handler that logs all memory updates handles a computation that sequentially writes two values differently than one that writes only the second one, even though these two computations are often considered equivalent [6].

Since handlers that do not respect the equations cannot receive an algebraic interpretation [8], some recent work [1, 4] assumes no non-trivial equations to hold, giving up most of the existing results on combining algebraic theories [3] and optimizations [5].

Extending types with equations

A possible way of resolving this issue is to capture the subset of valid equations in types. An algebraic approach already has a natural effect system, in which computations receive a type $A!\mathcal{O}$, where A is the type of returned values, and $\mathcal{O}$ is the set of operations that may get called [1, 4]. For example, a non-deterministic computation returning integers would be given the type $\texttt{int}!\{\texttt{choose}\}$, while a pure computation would have the type $\texttt{int}!\emptyset$.

This description can be extended to one of the form $A!\mathcal{O}\&\mathcal{E}$, where $\mathcal{E}$ is a now the subset of equations we assume to hold between operations $\mathcal{O}$. This type may be interpreted as the free model of the theory with the same operations, but with equations only from $\mathcal{E}$. For example, `if choose () then 1 else 2` and `if choose () then 2 else 1` can be considered as equivalent computations of type $\texttt{int}!\{\texttt{choose}\}\&\{\texttt{comm}\}$, but not of type $\texttt{int}!\{\texttt{choose}\}\&\{\texttt{assoc}\}$. This generalizes both the traditional approach to algebraic effects, if one considers $\mathcal{E}$ to be the set of all equations in the theory, or the approach with no equations, if $\mathcal{E} = \emptyset$.

Similar interpretation applies to handlers of type $A_1!\mathcal{O}_1\&\mathcal{E}_1 \Rightarrow A_2!\mathcal{O}_2\&\mathcal{E}_2$, where $\mathcal{E}_1$ is now the set of equations the handler must respect. For example, the handler

```
let choose_left = handler
  | choose () k -> k true
```

which makes `choose` constantly yield `true` in the handled computation, can be given the type $A!\{\texttt{choose}\}\&\{\texttt{assoc}, \texttt{idem}\} \Rightarrow A!\emptyset\&\emptyset$. Next, the handler

```
let choose_all = handler
  | choose () k -> (k true) @ (k false)
  | val x -> [x]
```

which returns the list of all possible results of the handled computation, can be given the type $A!\{\texttt{choose}\}\&\{\texttt{assoc}\} \Rightarrow A\,\text{list}!\emptyset$. Finally, the handler

```
let choose_sum = handler
  | choose () k -> (k true) + (k false)
```

which returns the sum of all possible results of the handled computation, can be given the type $\texttt{int}!\{\texttt{choose}\}\&\{\texttt{assoc}, \texttt{comm}, \texttt{idem}\} \Rightarrow \texttt{int}!\emptyset$.

The equations expected for the domain of the handler can also depend on the ones holding for the codomain. For example, one expects the handler

```
let choose_opposite = handler
  | choose () k -> if choose () then (k false) else (k true)
```

to have the type $A!\{\texttt{choose}\}\&\mathcal{E} \Rightarrow A!\{\texttt{choose}\}\&\mathcal{E}$
for any set of equations $\mathcal{E} \subseteq \{\texttt{assoc}, \texttt{comm}, \texttt{idem}\}$.

Open questions

Exact typing rules. When a computation may receive an enriched type remains to be determined. One may expect rules such as

$$\frac{\Gamma \vdash c : A!\mathcal{O}\&\mathcal{E} \qquad \mathcal{E} \subseteq \mathcal{E}'}{\Gamma \vdash c : A!\mathcal{O}\&\mathcal{E}'}$$

as we may always consider additional equivalences between programs to hold. The most involved rule seems to be one for assigning a type $A_1!\mathcal{O}_1\&\mathcal{E}_1 \Rightarrow A_2!\mathcal{O}_2\&\mathcal{E}_2$ to a handler. Here, we must check that the given handler respects all the equations $\mathcal{E}_1$, probably in a similar way as checking whether a handler is correct [8]. Since the equations describe the properties of effects on the level of algebraic theories, we can expect the resulting type system to be simpler than one involving dependent types or refinement types, however one must bear in mind that determining whether a handler respects a given set of equations is undecidable [8].

Applications

Handlers provide a very powerful control mechanism, which can dynamically change the context in which programs are run. One potential application of the described approach is to at least partially convey information about this behaviour through equations. The equations could also be used for enforcing behaviour. Even though determining their validity is undecidable, one could take a tool such as QuickCheck [2], which verifies properties of pure values by generating random tests, and extend it to testing impure computations.

Another prospective application is modular reasoning about handlers. For example, one can show that the usual monadic state handler satisfies certain properties [1], but the exact proof works only for the particular handler and needs to be redone for a different implementation. With equations in types, one could split the reasoning into two parts: (1) showing that a handler respects certain equations and has a given type, and (2) showing that any handler with that type satisfies a given property.

References

1 Andrej Bauer and Matija Pretnar. An effect system for algebraic effects and handlers. *Logical Methods in Computer Science*, 10(4), 2014.
2 Koen Claessen and John Hughes. Quickcheck: a lightweight tool for random testing of Haskell programs. In *ICFP*, pages 268–279. ACM, 2000.
3 Martin Hyland, Gordon D. Plotkin, and John Power. Combining effects: Sum and tensor. *Theor. Comput. Sci.*, 357(1-3):70–99, 2006.
4 Ohad Kammar, Sam Lindley, and Nicolas Oury. Handlers in action. In *ICFP*, pages 145–158. ACM, 2013.
5 Ohad Kammar and Gordon D. Plotkin. Algebraic foundations for effect-dependent optimisations. In *POPL*, pages 349–360. ACM, 2012.
6 Gordon D. Plotkin and John Power. Notions of computation determine monads. In *FoSSaCS*, volume 2303 of *LNCS*, pages 342–356. Springer, 2002.
7 Gordon D. Plotkin and John Power. Algebraic operations and generic effects. *Applied Categorical Structures*, 11(1):69–94, 2003.
8 Gordon D. Plotkin and Matija Pretnar. Handling algebraic effects. *Logical Methods in Computer Science*, 9(4), 2013.

Participants

- Sandra Alves
University of Porto, PT
- Kenichi Asai
Ochanomizu Univ. – Tokyo, JP
- Robert Atkey
University of Strathclyde – Glasgow, GB
- Clément Aubert
Appalachian State University – Boone, US
- Andrej Bauer
University of Ljubljana, SI
- Edwin Brady
University of St. Andrews, GB
- Xavier Clerc
Apimka – Paris, FR
- Stephen Dolan
University of Cambridge, GB
- Andrzej Filinski
University of Copenhagen, DK
- Philipp Haselwarter
University of Ljubljana, SI
- Martin Hofmann
LMU München, DE
- Patricia Johann
Appalachian State University – Boone, US
- Yukiyoshi Kameyama
University of Tsukuba, JP
- Ohad Kammar
University of Cambridge, GB
- Oleg Kiselyov
Tohoku University – Sendai, JP
- Daan Leijen
Microsoft Res. – Redmond, US
- Sam Lindley
University of Edinburgh, GB
- Conor McBride
University of Strathclyde – Glasgow, GB
- Gordon Plotkin
University of Edinburgh, GB
- Matija Pretnar
University of Ljubljana, SI
- Amr Hany Shehata Saleh
KU Leuven, BE
- Gabriel Scherer
Northeastern University – Boston, US
- Tom Schrijvers
KU Leuven, BE
- Alex Simpson
University of Ljubljana, SI
- KC Sivaramakrishnan
University of Cambridge, GB
- Sam Staton
University of Oxford, GB
- Niki Vazou
University of California – San Diego, US
- Niels Voorneveld
University of Ljubljana, SI
- Leo White
Jane Street – London, GB
- Jeremy Yallop
University of Cambridge, GB

Report from Dagstuhl Seminar 16131

Language Based Verification Tools for Functional Programs

Edited by

Marco Gaboardi[1], Suresh Jagannathan[2], Ranjit Jhala[3], and Stephanie Weirich[4]

1 **SUNY – Buffalo, US,** `gaboardi@buffalo.edu`
2 **Purdue University – West Lafayette, US,** `suresh@cs.purdue.edu`
3 **University of California – San Diego, US,** `jhala@cs.ucsd.edu`
4 **University of Pennsylvania – Philadelphia, US,** `sweirich@cis.upenn.edu`

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 16131 “Language Based Verification Tools for Functional Programs”. This seminar is motivated by two converging trends in computing – the increasing reliance on software has led to an increased interest in seeking formal, reliable means of ensuring that programs possess crucial correctness properties, and the dramatic increase in adoption of higher-order functional languages due to the web, multicore and “big data” revolutions.

While the research community has studied the problem of language based verification for imperative and first-order programs for decades – yielding important ideas like Floyd-Hoare Logics, Abstract Interpretation, Model Checking, and Separation Logic and so on – it is only relatively recently, that proposals have emerged for language based verification tools for functional and higher-order programs. These techniques include advanced type systems, contract systems, model checking and program analyses specially tailored to exploit the structure of functional languages. These proposals are from groups based in diverse research communities, attacking the problem from different angles, yielding techniques with complementary strengths.

This seminar brought diverse set of researchers together so that we could: compare the strengths and limitations of different approaches, discuss ways to unify the complementary advantages of different techniques, both conceptually and in tools, share challenging open problems and application areas where verification may be most effective, identify novel ways of using verification techniques for other software engineering tasks such as code search or synthesis, and improve the pedagogy and hence adoption of such techniques.

Seminar March 28 to April 1, 2016 – http://www.dagstuhl.de/16131
1998 ACM Subject Classification D.2.4 [Software Engineering]: Software/Program Verification F.3.1 [Logics and Meanings of Programs]: Specifying and Verifying and Reasoning about Programs
Keywords and phrases Functional Programming, Type Systems, Contracts, Dependent Types, Model Checking, Program Analysis
Digital Object Identifier 10.4230/DagRep.6.3.59

Except where otherwise noted, content of this report is licensed under a Creative Commons BY 3.0 Unported license

1 Summary

Marco Gaboardi
Suresh Jagannathan
Ranjit Jhala
Stephanie Weirich

License Creative Commons BY 3.0 Unported license
© Marco Gaboardi, Suresh Jagannathan, Ranjit Jhala, and Stephanie Weirich

This report summarizes the program and the outcomes of Dagstuhl Seminar 16131 "Language Based Verification Tools for Functional Programs", organized by:

- Marco Gaboardi, School of Computing, University of Dundee, UK
- Suresh Jagannathan, Purdue University, USA
- Ranjit Jhala, University of California, San Diego, USA
- Stephanie Weirich, University of Pennsylvania, USA.

The web, multi-core and "big-data" revolutions have been largely built on higher-order programming constructs pioneered in the Functional Programming community. Despite the increasing importance of such programs, there are relatively few tools that are focussed on ensuring that functional programs possess crucial correctness properties. While language based verification for imperative and first-order programs has been studied for decades yielding important ideas like Floyd-Hoare Logics, Abstract Interpretation, and Model Checking. It is only relatively recently, that researchers have proposed language based verification tools e.g. advanced type systems, contract systems, model checking and higher-order program analyses for functional and higher-order programs.

We organised this seminar to bring together the different schools of researchers interested in software reliability, namely, the designers and implementers of functional programming languages, and experts in software verification, in order create a larger community of researchers focused on this important goal, to let us compare the strengths and limitations of different approaches, to find ways to unite both intellectually, and via tools the complementary advantages of different techniques, and to devise challenging open problems and application areas where verification may be most effective. To this end, the seminar comprised a program of 30 talks from the leading experts on the above topics, and breakout sessions on:

1. Integrating formal methods tools in the curriculum
2. Hands on Tool Tutorials
3. User Interaction
4. Types and Effects

2 Table of Contents

3 Overview of Talks

3.1 Coinduction using copatterns and sized types

Andreas Martin Abel (Chalmers UT – Göteborg, SE)

License Creative Commons BY 3.0 Unported license
© Andreas Martin Abel
Joint work of Andreas Martin Abel, Brigitte Pientka, Anton Setzer, and David Thibodeau
Main reference A. Abel, B. Pientka, "Wellfounded recursion with copatterns: a unified approach to termination and productivity", in Proc. of the 18th ACM SIGPLAN Int'l Conf. on Functional Programming (ICFP'13), pp. 185–196, ACM, 2013.
URL http://dx.doi.org/10.1145/2500365.2500591

I present the new coinduction mechanism of Agda based on copatterns and sized types (Abel et al., POPL 2013, Abel/Pientka, ICFP 2013). As a running example, I use a coinductive definition of formal languages. I show how to use infinite tries to represent languages (sets of strings) and demonstrate that Agda allows an elegant representation of the usual language constructions like union, intersection, concatenation, and Kleene star. I also show how to reason about equality of languages using bisimulation and coinductive proofs.

3.2 Verified Compilers for a Multi-Language World

Amal Ahmed (Northeastern University – Boston, US)

License Creative Commons BY 3.0 Unported license
© Amal Ahmed
Joint work of Amal Ahmed; James Perconti
Main reference A. Ahmed, "Verified Compilers for a Multi-Language World", in Proc. of the 1st Summit on Advances in Programming Languages (SNAPL 2015), LIPIcs, Vol. 32, pp. 15–31, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.
URL http://dx.doi.org/10.4230/LIPIcs.SNAPL.2015.15

Verified compilers are typically proved correct under severe restrictions on what the compiler's output may be linked with, from no linking at all to linking only with code compiled from the same source language. Such assumptions contradict the reality of how we use these compilers since most software systems today are comprised of components written in different languages compiled by different compilers to a common target, as well as low-level libraries that may be handwritten in the target language.

The key challenge in verifying compilers for today's world of multi-language software is how to formally state a compiler correctness theorem that is compositional along two dimensions. First, the theorem must guarantee correct compilation of components while allowing compiled code to be composed (linked) with target-language components of arbitrary provenance, including those compiled from other languages (horizontal compositionality). Second, the theorem must support verification of multi-pass compilers by composing correctness proofs for individual passes (vertical compositionality).

In this talk, I'll describe a new methodology for building compositional verified compilers for today's world of multi-language software and discuss the challenges that lie ahead. Our project has two central themes, both of which stem from a view of compiler correctness as a language interoperability problem. First, to specify correctness of component compilation, we require that if a source component S compiles to target component T, then T linked with some arbitrary target code T' should behave the same as S interoperating with T'. The latter demands a formal semantics of interoperability between the source and target languages. Second, to enable safe interoperability between components compiled from languages as different as ML, Rust, C, and Coq's Gallina, we plan to design a gradually type-safe target language based on LLVM that supports safe interoperability between more precisely typed, less precisely typed, and type-unsafe components.

3.3 DeepSpec, CertiCoq and Verified Functional Algorithms

Andrew Appel (Princeton University – US)

License Ⓒ Creative Commons BY 3.0 Unported license
© Andrew Appel

I will speak briefly on three different topics: The new NSF-funded project at Princeton/Penn/Yale/MIT "The Science of Deep Specification," the CertiCoq project (Appel/Morrisett/-Pollack/Sozeau and students) to formalize the extraction and compilation process, and my new interactive-in-Coq textbook in the Software Foundations series, "Verified Functional Algorithms."

3.4 Type Systems as Proof Strategies

Iavor Diatchki (Galois Systems – Portland, US)

License Ⓒ Creative Commons BY 3.0 Unported license
© Iavor S. Diatchki

I'd like to share some thoughts on the design of type systems for existing dynamically typed languages such as Lua, JavaScript, Python, etc. The core idea is to blur the line between formal-verification and type-checking, and try to present a type system as a library of strategies that are able to discharge certain proof obligations. This is part of my ongoing research, and the ideas are not yet fully realized, but – if possible – I'd like to share them to get feedback and advice by fellow researchers.

3.5 Dependently Typed Programming in GHC 8

Richard A. Eisenberg (University of Pennsylvania – Philadelphia, US)

License Ⓒ Creative Commons BY 3.0 Unported license
© Richard A. Eisenberg

This talk demonstrates some of the new features I have added in the newest release of the Glasgow Haskell Compiler (GHC 8). GHC 8 supports the (Type : Type) axiom and visible type application, allowing easier and more expressive dependently typed programs than were possible previously. The main example in this talk is a program that performs type-safe database access while inferring the desired database schema from the program text. By inferring the schema, this program contains less boilerplate code and is more flexible than other type-safe database access approaches would allow.

3.6 Deductive Verification with Why3

Jean-Christophe Filliâtre (CNRS & University Paris Sud, FR)

License Creative Commons BY 3.0 Unported license
© Jean-Christophe Filliâtre
URL http://why3.lri.fr/

This short talk is a brief overview of Why3, a tool for deductive program verification.

Why3 provides an imperative programming language (with polymorphism, algebraic data types, pattern matching, exceptions, references, arrays, etc.), a mathematical language that is an extension of first-order logic, and a technology to discharge verification conditions using several, off-the-shelf interactive and automated theorem provers (Coq, Alt-Ergo, Z3, CVC3, etc.).

More details at http://why3.lri.fr/.

3.7 Relational cost analysis

Deepak Garg (MPI-SWS – Saarbrücken, DE)

License Creative Commons BY 3.0 Unported license
© Deepak Garg

Many applications require analysis of the relative use of resources (time, space) by two different programs or the same program with two different inputs. We call this the problem of relational cost analysis. Examples include compiler optimizations, incremental computation, static detection of side-channel leaks in security-critical programs and the stability analysis of algorithm implementations. This talk presents the beginnings of a type theory for relational cost analysis. It explains how a combination of lightweight dependent types, ideas from information flow analysis and a bit of co-monadic analysis can be combined to perform such analysis on non-trivial examples.

3.8 The Interactive Software Verification System KeY

Reiner Hähnle (TU Darmstadt, DE)

License Creative Commons BY 3.0 Unported license
© Reiner Hähnle
URL http://www.key-project.org

A brief overview of how KeY works, what can be done with it, the degree of automation, the user interfaces

3.9 Dependent Types and Multi-Monadic Effects in F*

Cătălin Hriţcu (INRIA – Paris, FR)

License Creative Commons BY 3.0 Unported license
© Cătălin Hriţcu
Joint work of Karthikeyan Bhargavan; Antoine Delignat-Lavaud; Simon Forest; Cédric Fournet; Cătălin Hriţcu; Chantal Keller; Markulf Kohlweiss; Aseem Rastogi; Pierre-Yves Strub; Nikhil Swamy; Santiago Zanella-Béguelin; Jean-Karim Zinzindohoue
Main reference N. Swamy, C. Hriţcu, C. Keller, A. Rastogi, A. Delignat-Lavaud, S. Forest, K. Bhargavan, C. Fournet, P.-Y. Strub, M. Kohlweiss, J. K. Zinzindohoue, S. Zanella Béguelin, "Dependent types and multi-monadic effects in F", in Proc. of the 43rd Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL'16), pp. 256–270, ACM, 2016; pre-print and supplementary material available from author's webpage.
URL http://dx.doi.org/10.1145/2837614.2837655
URL https://www.fstar-lang.org/papers/mumon/

We present a new, completely redesigned, version of F*, a language that works both as a proof assistant as well as a general-purpose, verification-oriented, effectful programming language.

In support of these complementary roles, F* is a dependently typed, higher-order, call-by-value language with *primitive* effects including state, exceptions, divergence and IO. Although primitive, programmers choose the granularity at which to specify effects by equipping each effect with a *monadic*, predicate transformer semantics. F* uses this to efficiently compute weakest preconditions and discharges the resulting proof obligations using a combination of SMT solving and manual proofs. Isolated from the effects, the core of F* is a language of pure functions used to write specifications and proof terms – its consistency is maintained by a semantic termination check based on a well-founded order.

We evaluate our design on more than 55,000 lines of F* we have authored in the last year, focusing on three main case studies. Showcasing its use as a general-purpose programming language, F* is programmed (but not verified) in F*, and bootstraps in both OCaml and F#. Our experience confirms F*'s pay-as-you-go cost model: writing idiomatic ML-like code with no finer specifications imposes no user burden. As a verification-oriented language, our most significant evaluation of F* is in verifying several key modules in an implementation of the TLS-1.2 protocol standard. For the modules we considered, we are able to prove more properties, with fewer annotations using F* than in a prior verified implementation of TLS-1.2. Finally, as a proof assistant, we discuss our use of F* in mechanizing the metatheory of a range of lambda calculi, starting from the simply typed lambda calculus to System F$_\omega$ and even a sizeable fragment of F* itself – these proofs make essential use of F*'s flexible combination of SMT automation and constructive proofs, enabling a tactic-free style of programming and proving at a relatively large scale.

Talk slides available at:
http://materials.dagstuhl.de/files/16/16131/16131.CatalinHritcu.Slides.html

3.10 Relational Reasoning about Higher-Order Shape Properties

Gowtham Kaki (Purdue University – West Lafayette, US) and Suresh Jagannathan (Purdue University – West Lafayette, US)

License Creative Commons BY 3.0 Unported license
© Gowtham Kaki and Suresh Jagannathan
Main reference G. Kaki, S. Jagannathan, "A Relational Framework for Higher-order Shape Analysis", in Proc. of the 19th ACM SIGPLAN Intl. Conf. on Functional Programming (ICFP'14), pp. 311–324, ACM, 2014.
URL http://dx.doi.org/10.1145/2628136.2628159

In this talk, I present CATALYST, a relational reasoning framework integrated within a dependent type system that is capable of automatically verifying complex invariants over the shapes of algebraic datatypes. A novel contribution of CATALYST is the concept of parametric relations – relations parameterized over other relations that enable intuitive specifications for higher-order polymorphic functions, while retaining the compositionality and decidability of type checking. I describe an encoding of fully instantiated parametric relations in a decidable subset of first-order logic, and show how it is sufficient to type check higher-order functions against expressive specifications. I describe an implementation of CATALYST in SML, present multiple examples, and end the talk with a brief note on inferring rich dependent types in CATALYST.

3.11 Program Verification Based on Higher-Order Model Checking

Naoki Kobayashi (University of Tokyo – Tokyo, Japan)

License Creative Commons BY 3.0 Unported license
© Naoki Kobayashi

Higher-order model checking has recently been applied to fully automated verification of higher-order functional programs. In the talk, I plan to provide a tutorial to explain what is higher-order model checking and how it can be applied to program verification. I will also summarize recent progress on the higher-order model checking approach to program verification.

3.12 Certified Automated Theorem Proving for Type Inference

Ekaterina Komendantskaya (Heriot-Watt University – UK)

License Creative Commons BY 3.0 Unported license
© Ekaterina Komendantskaya
Joint work of Peng Fu; Ekaterina Komendantskaya; Tom Schrijvers
URL http://staff.computing.dundee.ac.uk/katya/CoALP/

Two facts are universally acknowledged: critical software must be subject to formal verification and modern verification tools need to scale and become more user-friendly in order to make more impact in industry.

There are two major styles of verification: (*) algorithmic: verification problems are specified in an automated prover, e.g. (constraint) logic programming or SMT solver, and properties of interest are verified by the prover automatically. Such provers can be fast, but their trustworthiness is hard to establish without producing and checking proofs. This is due to complexity of modern-day solvers, e.g. SMT solvers have codebases 100K in C++. These tools exhibit bugs and are not trustworthy enough for critical systems.

An alternative is (**) a typeful approach to verification: instead of verifying programs in an external prover, a programmer may record all properties of interest as types of functions in his programs. Thanks to Curry-Howard isomorphism, type inhabitants also play the role of executable functions and proof witnesses, thus completing the certification cycle.

At their strongest, types can cover most of the properties of interest to the verification community, e.g. recent dialects Liquid Haskell and F* allow pre- and post-condition formulation at type level. But, when properties expressed at type level become rich, type inference engines have to assume the role of automated provers, e.g. Liquid Haskell and F* connect directly to SMT solvers. Thus, once again, we delegate trust without having proper certification of automated proofs.

This talk was about our recent work [Fu, Komendantskaya, Schrijvers, in FLOPS 2016] that resolves the above dichotomy "scale versus trust" by offering a new, typeful, approach to automated proving for type inference. Recently, we designed a new method of using logic programming in Haskell type class inference: Horn clauses can be represented as types, and proofs by resolution – as proof terms inhabiting the types. Thus, the problem of automated inference in Horn Clause logic is re-cast as the problem of type inhabitation in a suitable type system. In this way, outputs of the resulting Curry-Howard Horn Clause prover are directly compatible with type system of Haskell's compiler. Overall, this method allows to achieve both high standards of automated proof certification and compatibility of the automated prover with the target compiler.

The question is: can this method apply to other existing algorithmic and typeful approaches?

3.13 Ramsey-based Methods: From Size-Change Termination to Satisfiability in Temporal Logics

Martin Lange (Universität Kassel, DE)

License Creative Commons BY 3.0 Unported license
© Martin Lange
Joint work of Oliver Friedmann; Felix Klaedtke; Martin Lange
Main reference O. Friedmann, F. Klaedtke, M. Lange, "Ramsey-Based Inclusion Checking for Visibly Pushdown Automata", ACM Transactions on Computational Logic (TOCL), Vol. 16(4), Article No. 34, 2015.
URL http://dx.doi.org/10.1145/2774221

Ramsey's Theorem about the existence of infinite monochromatic subgraphs in the finitely coloured graph on the natural numbers was a crucial ingredient in Büchi's original proof of the complementation closure of the class of omega-regular languages. For most of the time since then, this has just been seen as a mathematical tool for obtaining a proof.

In the early 2000s, Lee, Jones and Ben-Amram introduced the size-change termination principle – a method for proving termination of abstract functional programs that can only reduce, copy and permute their arguments. They showed that the problem could be solved by a reduction to the inclusion problem for Büchi automata, yet the existing algorithms based on explicit complementation were not competitive enough in practice. They devised a method whose correctness directly relies Ramsey's Theorem which essentially showed that Büchi's original complementation proof has more to offer than just mathematical truth, but that it can also lead to elegant and practical algorithms for the analysis of omega-automata. Since then, Ramsey-based methods have proved to be very efficient for such tasks.

Lately, Friedmann, Klaedtke and myself have shown that Ramsey-based methods can be used effectively and efficiently beyond the class of omega-regular languages, namely for visibly pushdown omega-languages recognised by corresponding parity automata.

It still remains to be seen whether a similar development is possible for tree automata, namely whether there are successful program analysis techniques which could similarly lead to better algorithms for tree automata inclusion.

3.14 Automated verification of functional programs

Rustan Leino (Microsoft Research – USA)

License 🅭 Creative Commons BY 3.0 Unported license
© Rustan Leino

Dafny started as an imperative language with specifications, where the specifications had mathematical elements also found in functional languages. These functional features grew beyond uses in specifications and now include datatypes, co-datatypes, recursive functions, and predicates defined as least/greatest fixpoints. This means Dafny can be used as a functional language. Dafny also includes an automated verifier, and the language has proof-authoring support for when automation doesn't hold up. In difference to some other functional languages with verification support, the logic of Dafny is not based around dependent types but rather Hoare logic. In this talk, I'll demo a tour through Dafny's features and point out some limitations.

3.15 A Static Type Analysis for Lua

Jan Midtgaard (Technical University of Denmark – Lyngby, DK)

License 🅭 Creative Commons BY 3.0 Unported license
© Jan Midtgaard

Higher-order, dynamically-typed programming languages are flexible but come at the price of less tool support. To address this challenge we develop a static type analysis for the Lua programming language. Lua represents an interesting, yet minimal mix of imperative, functional, and object-oriented language features which makes this a challenging task. We present a prototype implementation of the developed analysis.

3.16 Subtle points

Conor McBride (University of Strathclyde – UK)

License 🅭 Creative Commons BY 3.0 Unported license
© Conor McBride

I'll sketch the approach to computation in Homotopy Type Theory that we're currently exploring at Strathclyde. In the business of constructing paths between isomorphic types, we introduce a notion of segmentation: nontrivially segmented paths have intermediate points and can thus be constructed in a piecewise continuous manner. Segmentations are naturally ordered by their subtlety, as any segment can be split in two. Crucially, segmented paths deliver the means to transport values between any two of their points.

3.17 Higher-order horn clauses and higher-order model checking

Luke Ong (University of Oxford – Oxford, UK)

License Creative Commons BY 3.0 Unported license
© Luke Ong
Joint work of Luke Ong; Steven Ramsay

Higher-order model checking (HOMC) is the model checking of trees generated by recursion schemes. In the (standard) intersection type approach to HOMC, one would construct a certain type environment, which constitutes a symbolic representation of the invariant. In this ongoing work, we consider the problem of finding higher-order inductive invariants in a purely logical setting, namely, the satisfiability problem for (constrained) higher-order horn clauses. Formulated as higher-order constraint solving, the problem has a much broader appeal than recursion scheme model checking, yet we argue that much of the technology already developed by the HOMC community can be made highly effective at solving it. In particular, we describe an adaptation of Kobayashi's Hybrid Algorithm to the problem and highlight its similarities to McMillan's Lazy Annotation algorithm (for solving first-order constrained horn clauses).

3.18 From analysis-directed semantics to specifications-in-types

Dominic Orchard (University of Oxford – Oxford, UK)

License Creative Commons BY 3.0 Unported license
© Dominic Orchard

Various recent works on effects and resource usage (coeffects) have provided semantic models that are indexed in some way by analysis information, e.g., by effect systems, Hoare logic triples, resource bounds. Such models typically provide inductive families of denotations, following the shape of an inductively defined program analysis. For example, "graded monads" are indexed by a monoidal effect system. This is a powerful new paradigm as it provides a way to refine semantics by analysis information, and exposes analysis information in semantics. In this talk, I give an overview of the general approach, which I call analysis-directed semantics. I then show how such models can be used directly in programming, where the indices of these semantic structures can be used to embed functional specifications at the type-level. I'll give some examples in Haskell including effects, computational complexity, communication safety for concurrency, and well-bracketed file handlers.

3.19 Programming Coinductive Proofs Using Observations

Brigitte Pientka (McGill Universitiy – Montreal, Canada)

License Creative Commons BY 3.0 Unported license
© Brigitte Pientka
Joint work of Andreas Abel; Andrew Cave; Brigitte Pientka; Anton Setzer; David Thibodeau

Coinduction is a key proof technique to establish properties about systems that continue to run and produce results (i.e. network or communications protocols, I/O interaction, data streams, or processes) . Yet, mechanizing coinductive proofs about formal systems and representing, generating and manipulating such proofs remains challenging. In this talk, we develop the idea of programming coinductive proofs dual to the idea of programming inductive proofs. Unlike properties about finite data which can be defined by constructing a derivation, properties about infinite data can be described by the possible observations we can make. Dual to pattern matching, a tool for analyzing finite data, we develop the concept of copattern matching, which allows us to describe properties about infinite data. This leads to a symmetric proof language where pattern matching on finite and infinite data can be mixed.

3.20 Language-based Verification of Untyped Expressions

Ruzica Piskac (Yale University, US)

License Creative Commons BY 3.0 Unported license
© Ruzica Piskac

Software failures resulting from configuration errors have become commonplace as modern software systems grow increasingly large and more complex. The lack of language constructs in configuration files, such as types and grammars, has directed the focus of a configuration file verification towards building post-failure error diagnosis tools. In addition, the existing tools are generally language specific, requiring the user to define at least a grammar for the language models and explicit rules to check.

In this talk, we outline a framework which analyzes datasets of correct configuration files and derives rules for building a language model from the given dataset. The resulting language model can be used to verify new configuration files and detect errors in them. Our proposed framework is highly modular, does not rely on the system source code, and can be applied to any new configuration file type with minimal user input.

3.21 Program Synthesis from Refinement Types

Nadia Polikarpova (MIT – Cambridge, US)

License Creative Commons BY 3.0 Unported license
© Nadia Polikarpova
Joint work of Ivan Kuraj; Nadia Polikarpova; Armando Solar-Lezama

The key to scalable program synthesis is modular verification: the better a specification for a program can be broken up into independent specifications for its components, the fewer combinations of components the synthesizer has to consider, leading to a combinatorial reduction in the size of the search space. This talk will present Synquid: a synthesizer that takes advantage of the modularity offered by type-based verification techniques to efficiently generate recursive functional programs that provably satisfy a given specification in the form of a refinement type.

We have evaluated Synquid on a large set of synthesis problems and found that it exceeds the state of the art in terms of both scalability and usability. Synquid was able to synthesize more complex programs than those reported in prior work (for example, various sorting algorithms, operations on balanced trees). It was also able to handle most of the benchmarks tackled by existing tools, often starting from a significantly more concise and intuitive user input. Moreover, due to automatic refinement discovery through a variant of liquid type inference, our approach fundamentally extends the class of programs for which a provably correct implementation can be synthesized without requiring the user to provide full inductive invariants.

3.22 Demand Driven Analysis For Functional Programs

Scott Smith (Johns Hopkins University – Baltimore, US)

License Creative Commons BY 3.0 Unported license
© Scott Smith

Joint work of Zachary Palmer, Scott Smith
Main reference Z. Palmer, S. Smith, "Higher-Order Demand-Driven Program Analysis", in Proc. of the 30th Europ. Conf. on Object-Oriented Programming (ECOOP'16), LIPIcs, Vol. 56, pp. 19:1–19:25, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.
URL http://dx.doi.org/10.4230/LIPIcs.ECOOP.2016.19

We explore a novel approach to higher-order program analysis that brings ideas of on-demand lookup from first-order CFL-reachability program analyses to functional programs. The analysis needs to produce only a control-flow graph; it can derive all other information including values of variables directly from the graph. Several challenges had to be overcome, including how to build the control-flow graph on-the-fly and how to deal with nonlocal variables in functions. The resulting analysis is flow- and context-sensitive with a provable polynomial-time bound.

3.23 Equations: A toolbox for function definitions in Coq

Matthiew Sozeau (Université Paris Diderot – Paris, France)

License Creative Commons BY 3.0 Unported license
© Matthieu Sozeau

We present a compiler for definitions made by pattern matching on inductive families in the Coq system. It allows to write structured, recursive dependently-typed functions, automatically find their realization in the core type theory and generate proofs to ease reasoning on them. The high-level interface allows to write dependently-typed functions on inductive families in a style close to Agda or Epigram, while their low-level implementation is accepted by the vanilla core type theory of Coq. This setup uses the smallest trusted code base possible and additional tools are provided to maintain a high-level view of definitions.

3.24 Temporal Verification of Higher-Order Functional Programs

Tachio Terauchi (JAIST – Japan)

License 🅭 Creative Commons BY 3.0 Unported license
© Tachio Terauchi

We present an automated approach to verifying arbitrary omega-regular properties of higher-order functional programs. Previous automated methods proposed for this class of programs could only handle safety properties or termination, and our approach is the first to be able to verify arbitrary omega-regular liveness properties. Our approach is automata-theoretic, and extends our recent work on binary-reachability-based approach to automated termination verification of higher-order functional programs to fair termination published in ESOP 2014. In that work, we have shown that checking disjunctive well-foundedness of (the transitive closure of) the "calling relation" is sound and complete for termination. The extension to fair termination is tricky, however, because the straightforward extension that checks disjunctive well-foundedness of the fair calling relation turns out to be unsound, as we shall show in the paper. Roughly, our solution is to check fairness on the transition relation instead of the calling relation, and propagate the information to determine when it is necessary and sufficient to check for disjunctive well-foundedness on the calling relation. We prove that our approach is sound and complete. We have implemented a prototype of our approach, and confirmed that it is able to automatically verify liveness properties of some non-trivial higher-order programs.

3.25 Occurrence typing modulo theories

Sam Tobin-Hochstadt (Indiana University – Bloomington, US)

License 🅭 Creative Commons BY 3.0 Unported license
© Sam Tobin-Hochstadt

Occurrence typing has been successful in enabling Typed Racket to handle a wide variety of existing Racket idioms. In this talk, I present a new extension, adding dependent refinement types parameterized over arbitrary solvers to Typed Racket.

Dependent refinement types allow Typed Racket programmers to express rich type relationships, ranging from data structure invariants such as red-black tree balance to preconditions such as vector bounds. Refinements allow programmers to embed the propositions that occurrence typing in Typed Racket already reasons about into their types. Further, extending occurrence typing to refinements allows us to make the underlying formalism simpler and more powerful.

3.26 Refinement Caml: A Refinement Type Checking and Inference Tool for OCaml

Hiroshi Unno (Tsukuba University – Japan)

License Creative Commons BY 3.0 Unported license
© Hiroshi Unno

We will demonstrate Refinement Caml (RCaml), a fully-automated path-sensitive verification tool for the OCaml functional language based on refinement type checking and inference. RCaml supports advanced language features such as algebraic data structures and higher-order recursive functions. RCaml can solve various program analysis problems formulated as refinement type inference problems, including static assertion checking, termination and non-termination analysis, precondition inference, relational verification, and symbolic game solving. RCaml first reduces these problems into constraint solving problems, where the constraints are expressed by Horn clauses with predicate variables that are placeholders for preconditions, postconditions, safe inductive invariants, and well-founded recursion relations of the original program. RCaml then solves the generated constraints by using invariant and ranking function synthesis techniques.

3.27 Contract Verification and Refutation

David Van Horn (University of Maryland – College Park, USA)

License Creative Commons BY 3.0 Unported license
© David Van Horn

In this talk, I'll present a new approach to automated reasoning about higher- order programs by endowing symbolic execution with a notion of higher-order, symbolic values. Our approach is sound and relatively complete with respect to a first-order solver for base type values. Therefore, it can form the basis of automated verification and bug-finding tools for higher-order programs. To validate our approach, we use it to develop and evaluate a system for verifying and refuting behavioral software contracts of components in a functional language, which we call soft contract verification. In doing so, we discover a mutually beneficial relation between behavioral contracts and higher-order symbolic execution.

3.28 Refinement Types for Haskell

Niki Vazou (University of California – San Diego, US)

License 🅭 Creative Commons BY 3.0 Unported license
© Niki Vazou
Joint work of Alexander Bakst; Ranjit Jhala; Eric L. Seidel; Niki Vazou
Main reference N. Vazou, E. L. Seidel, R. Jhala, D. Vytiniotis, S. Peyton-Jones, "Refinement Types For Haskell", Manuscript.
URL http://goto.ucsd.edu/~nvazou/refinement_types_for_haskell.pdf
Main reference N. Vazou, A. Bakst, R. Jhala, "Bounded refinement types", in Proc. of the 20th ACM SIGPLAN Int'l Conf. on Functional Programming (ICFP'15), pp. 48–61, ACM, 2015; pre-print available from author's webpage.
URL http://dx.doi.org/10.1145/2784731.2784745
URL https://ranjitjhala.github.io/static/bounded_refinement_types.pdf

Haskell has many delightful features, perhaps the most beloved of which is its type system which allows developers to specify and verify a variety of program properties at compile time. However, many properties, typically those that depend on relationships between program values are impossible, or at the very least, cumbersome to encode within Haskell's type system.

Liquid types enable the specification and verification of value-dependent properties by extending Haskell's type system with logical predicates drawn from efficiently decidable logics.

In this talk, we will start with a high level description of Liquid Types. Next, we will present an overview of LiquidHaskell, a liquid type checker for Haskell. In particular, we will describe the kinds of properties that can be checked, ranging from generic requirements like totality (will 'head' crash?) and termination (will 'mergeSort' loop forever?), to application specific concerns like memory safety (will my code SEGFAULT?) and data structure correctness invariants (is this tree BALANCED?).

3.29 Lazy staged binary decision diagrams

Wouter Swierstra (Utrecht University, NL)

License 🅭 Creative Commons BY 3.0 Unported license
© Wouter Swierstra
Joint work of Joao Paulo Pizani Flor; Rob Spoel; Wouter Swierstra

This talk reviews the proof-by-reflection method of proof automation. By exploiting the computational nature of type theory, we can implement a verified decision procedure for a specific domain, rather than write manual proof terms. We would like to use this to implement a decision procedure on boolean expressions. The usual techniques to do so, binary decision diagrams, rely on manipulating pointers in memory – which does not work well with most proof assistants based on type theory. Instead, we propose to investigate how we might be able to create the desired data structures in memory using metaprogramming.

3.30 Verification by Optimization: Two Approaches to Manifest Contracts

Michael Greenberg (Pomona College – Claremont, US)

License 🅭 Creative Commons BY 3.0 Unported license
© Michael Greenberg

Contract systems can be used for program verification: if you can optimize away the contract checks, you have proved the program correct!

There are two approaches in the literature: subtyping and static analysis. The subtyping approach (typical in the manifest setting, and due to Flanagan) removes upcasts, casts from a subtype to a super type. The static analysis approach (typical in the latent setting, and due most immediately to Van Horn, though many others have written on similar topics before) builds an abstract model of which values each variable can hold – if the set of values all pass a given contract, that contract can be eliminated.

Is one "better" than the other? Can we use both in the same setting? Do they optimize away different kinds of contracts?

In this brief talk, I lay out the differences between the two and propose a "non-disjointness" judgment for determining when to reject a program because of a bad cast.

4 Breakout Sessions

In addition to the formal talks, we had breakout sessions on the following topics:

- **Integrating formal methods tools in the curriculum.** In this session, we discussed a variety of topics ranging from current best exemplars of classes and textbooks for formal methods, what makes for a good class or text, and regular classes (e.g. operating systems or compilers) that could be extended with Formal Methods module.
- **Hands on Tool Tutorials.** In this session, different participants gave short demonstrations of their tools and then encouraged others to use them to carry out various tasks or work through tutorials. These tools included: Agda, Dafny, F^*, Haskell, LiquidHaskell, Racket, RCaml, Synquid and Why3.
- **User Interaction** In this session, the participants identified several key challenges that must be addressed to improve the user experience for formal tools, as well as new modes of using the tools, e.g. not just for verification but to synthesize programs.
- **Types and Effects** In this session, the participants discussed recent advances in how to track effects, and different applications of effect systems.

Participants

- Andreas Martin Abel
Chalmers UT – Göteborg, SE
- Amal Ahmed
Northeastern University – Boston, US
- Andrew W. Appel
Princeton University, US
- Lennart Augustsson
Standard Chartered Bank – London, GB
- Edwin Brady
University of St. Andrews, GB
- Iavor Diatchki
Galois – Portland, US
- Richard A. Eisenberg
University of Pennsylvania – Philadelphia, US
- Jean-Christophe Filliâtre
CNRS & University Paris Sud, FR
- Cormac Flanagan
University of California – Santa Cruz, US
- Marco Gaboardi
SUNY – Buffalo, US
- Deepak Garg
MPI-SWS – Saarbrücken, DE
- Michael Greenberg
Pomona College – Claremont, US
- Reiner Hähnle
TU Darmstadt, DE
- Cătălin Hriţcu
INRIA – Paris, FR
- Suresh Jagannathan
Purdue University – West Lafayette, US
- Ranjit Jhala
University of California – San Diego, US
- Gowtham Kaki
Purdue University – West Lafayette, US
- Gabriele Keller
UNSW – Sydney, AU
- Naoki Kobayashi
University of Tokyo, JP
- Ekaterina Komendantskaya
University of Dundee, GB
- Martin Lange
Universität Kassel, DE
- K. Rustan M. Leino
Microsoft Corporation – Redmond, US
- Conor McBride
University of Strathclyde – Glasgow, GB
- Jan Midtgaard
Technical University of Denmark – Lyngby, DK
- Chih-Hao Luke Ong
University of Oxford, GB
- Dominic Orchard
University of Cambridge, GB
- Brigitte Pientka
McGill Univ. – Montreal, CA
- Ruzica Piskac
Yale University, US
- Nadia Polikarpova
MIT – Cambridge, US
- Scott Smith
Johns Hopkins University – Baltimore, US
- Matthieu Sozeau
University Paris-Diderot, FR
- Wouter Swierstra
Utrecht University, NL
- Tachio Terauchi
JAIST – Ishikawa, JP
- Sam Tobin-Hochstadt
Indiana University – Bloomington, US
- Hiroshi Unno
University of Tsukuba, JP
- David Van Horn
University of Maryland – College Park, US
- Niki Vazou
University of California – San Diego, US
- Stephanie Weirich
University of Pennsylvania – Philadelphia, US
- Nobuko Yoshida
Imperial College London, GB