Report on BCTCS 2017

The 33nd British Colloquium for Theoretical Computer Science

26–28 April 2017, St Andrews University

Markus Pfeiffer

The British Colloquium for Theoretical Computer Science (BCTCS) is an annual forum in which researchers in Theoretical Computer Science can meet, present research findings, and discuss developments in the field. It also provides an environment for PhD students to gain experience in presenting their work in a wider context, and to benefit from contact with established researchers.

BCTCS 2017 was hosted by the School of Computer Science at the University of St Andrews, and held from 26th to 28th April, 2017. The event attracted over 30 participants, and featured an interesting and wide-ranging programme of six invited talks and 13 contributed talks. Abstracts for all of the talks from BCTCS 2017 are provided below.

We are grateful to the *Heilbronn Institute for Mathematical Research* who provided funding for 6 PhD students (including 3 female students) and funding for invited speakers' expenses. We are also thankful to the *London Mathematical Society* for their annual sponsorship of the *LMS keynote speaker in Discrete Mathematics* – Prof László Babai (University of Chicago).

Talks covered a range of topics in theoretical computer science, with participation from across the United Kingdom. The opening talk by Felix Fischer (University of Glasgow), presenting his research on *position auctions* in which self-interested parties bid for a commodity (usually advertisement space on the internet), and are not honest about their intentions.

The morning also featured a dedicated meet-and-greet session where participants were encouraged to meet and talk to people they did not know yet.

Jessica Enright (University of Stirling) opened the next session showing us about her endeavours to build a maze for the flash game mouse maze in which the mouse takes a maximal amount of moves to escape the maze, which turns out to be a very difficult combinatoric problem, and Kitty Meeks (University of Glasgow) showed off her work on computational complexity; knowing that a problem is decidable, but hard (for instance NP complete) begs the question how many (small) solutions there are.

In the afternoon session Perdita Stevens (University of Edinburgh) told us about the important connection between the practice of software engineering and the theory of bisimulations.

The evening was then dedicated to the LMS Keynote speaker in Discrete Mathematics, Prof László Babai (University of Chicago), who talked about the group theoretic aspects of his recent breakthrough result on the complexity of the Graph Isomorphism problem. His lecture was well publicised and attracted an audience of about 100 people from St Andrews, Edinburgh and Glasgow.

Thursday was opened by Edwin Brady (University of St Andrews) speaking about his work encoding state and state machines in the dependently typed programming language *Idris* – a language he developed himself to connect the theoretical advances in type theory with practical software engineering. His PhD student Matú Tejiák, who as part of his PhD project improved erasure of runtime-irrelevant proofs from Idris programs gave even more insight into this exciting topic.

The second session of the day consisted of Dylan McDermott, Georgina Lungu, and James Hoey, all PhD students, and the afternoon session featured a very entertaining talk by Conor McBride from the University of Strathclyde, who explained to us what Syntax really is. The afternoon was then dedicated to an additional talk by László Babai about combinatorial aspects of this Graph Isomorphism result.

The conference dinner took place on Thursday evening, and was a great success.

Friday was opened by Mehrnoosh Sadrzadeh who explained how Monoids, Vectors, and Tensors play a role in her research on modeling natural language, and the conference concluded with further talks given by PhD students.

As a satellite event, László Babai agreed to teach St Andrews undergraduates in Mathematics and Computer Science about permutation groups and the graph isomorphism problem. This event took place on Saturday after the BCTCS and was met with great enthusiasm by the students. Particularly the Computer Science students were very fascinated about how interesting Mathematics and Theoretical Computer Science can be.

It was noted by participants how enjoyable the event was, and that there was a good proportion of female speakers and participants.

BCTCS 2018 will be hosted by Royal Holloway University of London. Researchers and PhD students wishing to contribute talks concerning any aspect of Theoretical Computer Science are cordially invited to do so. Further details are available from the BCTCS website at `www.bctcs.ac.uk`.

Invited Talks at BCTCS 2017

László Babai (University of Chicago)
LMS Keynote Lecture in Discrete Maths
Graph Isomorphism

In the first of two lectures, we discuss the group theoretic aspects at the core of recent progress on the Graph Isomorphism problem: a lemma (the "Unaffected Stabilizers Lemma") and its algorithmic application (the "Local Certificates algorithm"). The latter produces a canonical structure, breaking the symmetry that has been the bottleneck for Luks' classic (1980) Divide-and-Conquer strategy.

In the second of the two lectures, we begin with further details of the group theory involved in the Graph Isomorphism algorithm, followed by a discussion of the combinatorial techniques. Classical (binary) and higher (k-ary) coherent configurations will be the central objects of the discussion.

Edwin Brady (University of St Andrews)
State Machines All The Way Down

A useful pattern in dependently typed programming is to define a state transition system, for example the states and operations in a network protocol, as an indexed monad. We index each operation by its input and output states, thus guaranteeing that operations satisfy pre- and post-conditions by typechecking. However, what if we want to write a program using several systems at once? What if we want to define a high-level state transition system, such as a network application protocol, in terms of lower level states, such as network sockets and mutable variables?

In this talk, I present an architecture for dependently typed applications based on a hierarchy of state transition systems implemented in a generic data type ST. This is based on a monad indexed by contexts of resources, allowing us to reason about multiple state transition systems in the type of a function. Using ST, we show: how to implement a state transition system as a dependent type with type level guarantees on its operations; how to account for operations which could fail; how to combine state transition systems into a larger system; and how to implement larger systems as a hierarchy of state transition systems. I illustrate the system with a high level network application protocol, implemented in terms of POSIX network sockets.

Felix Fischer (Glasgow University)
Truthful Outcomes from Non-Truthful Position Auctions

The area of mechanism design is concerned with the development of algorithms that produce good outcomes given inputs from self-interested individuals. One of the stars of mechanism design theory is the Vickrey-Clarke-Groves (VCG) mechanism, which makes it optimal for each individual to truthfully reveal its input. In the real world, however, the VCG mechanism is used with surprising rarity and the mechanisms we actually find are non-truthful. An example of this phenomenon are position auctions used by internet search engines like Google and Bing to allocate space for advertisements displayed next to genuine search results. Here only non-truthful mechanisms have ever been used, and what in theory looks like a beginner's mistake was a huge success in practice. An obvious advantage of the non-truthful mechanisms is that they use simpler payments, and it has been known for some time why this simplicity does not lead to chaos when participants decide strategically how to bid. We will see that the simplicity of payments also comes with a greater robustness to uncertainty on the part of the search engines regarding the relative values of positions. This talk is based on joint work with Paul Dütting (LSE) and David C. Parkes (Harvard).

Conor McBride (Strathclyde University)
Syntax: What's it like?

When we write programs or prove theorems about a programming language, the first thing we often do is write down a datatype for its syntax. But a syntax is not just any old datatype: there are usually notions of "scope" and "sort" which impose structure on terms and drive the implementations of operations such as substitution. In this talk, I adapt the technology of datatype-generic programming to the specific needs of types for syntax, showing how types and operations can be generated from a first class notion of grammar. Why keep a dog and bark yourself?

Mehrnooah Sadrzadeh (Queen Mary University of London)
Monoids, Vectors, and Tensors for Natural Language

Formalising different aspects of natural language has kept computational linguistics, logicians, and mathematicians busy for decades. On the syntactic side, we have: Chomsky and generative grammars from the 1950's; Ajdukiewicz and Bar-Hillel and functional grammars from the 1930's; and Lambek and residuated monoids from the 1950's (to name a few). On the semantic side, we have the seminal work of Montague on lambda calculus and higher order logics. Recently, distributional semantics based on Firth and Harris' insights has become a successful method of representing meanings of words. This method takes advantage of vectors and linear algebra to reason about the information encoded in large quantities of data available in the form of text and corpora of documents. I present work with my colleagues on extending distributional semantics from words to phrases and sentence. In order to realise this passage, we use a notion of grammatical compositionality formalised by Lambek and employed in other categorial grammars such as the CCG. Our model relies on the notion of tensors and operations from multilinear algebra. I present the theory behind the model and its applications to natural language tasks such as phrase and sentence similarity, paraphrasing, classification, and entailment. In many cases the predications of the tensor models better those of simple vector addition and multiplication.

Perdita Stevens (Edinburgh University)
Bisimulations, Bidirectionality and the Future of Software Engineering

A bidirectional transformation is a means of maintaining consistency between two or more data sources. In model driven development, the data sources are models:

abstract, usually graphical, representations of some aspects of a system. Model driven development has so far been successful in some contexts, but not others; a difficulty that remains is how to make it sufficiently agile, while retaining the advantages it presents where it works. The vision of future software engineering that I plan to put before you involves groups of people who are experts in something but not necessarily software developers, each working with their own model, and using bidirectional transformations to manage consistency between them. I explain (something about) why that would be a good thing, what needs to happen for the vision to become reality, and what bisimulations have to do with it.

Contributed Talks at BCTCS 2017

Obad Abdullah Alhumaidan (Newcastle University)
Modelling Multi-Valued Networks using Rewriting Logic

In this talk we give an overview of recent work on modelling Multi-Valued Networks (MVNs) using Rewriting Logic (RL). We define a formal translation of asynchronous and synchronous MVNs into RL and model the MVN behaviour using RL. Correctness proofs were done for these translations based on showing they are sound and complete, and we worked with a range of case studies (e.g. Tryptophan biosynthesis genetic network) to illustrate our translation. Maude's model checking capabilities for Linear Temporal Logic (LTL) was used to analyse an MVN model, and performance testing was carried out using a scalable test model.

Tom Bourne (University of St Andrews)
Subwords and Stars

In the field of formal language theory, the generalised star-height problem asks whether or not there exists an algorithm to determine the minimum nesting depth of stars required in order to represent a given regular language by a regular expression. In this setup, we consider complement as a basic operator. In particular, it is not yet known whether there exist languages of generalised star-height greater than one. We consider the generalised star-height of the languages in which a fixed word occurs as a contiguous subword an exact number of times and of the languages in which a fixed word occurs as a contiguous subword modulo a fixed number, and see that in each case it is at most one.

Jessica Enright (University of Stirling)
Building a better mouse maze

Mouse Maze is a Flash game about Squeaky, a mouse who has to navigate a subset of the grid graph using a simple deterministic rule, which naturally generalises to a game on arbitrary graphs with some interesting chaotic-looking dynamics. We present efforts to generate graphs which effectively trap Squeaky in the maze for long periods of time, and some theoretical results bounding how long he can be trapped. We show that Squeaky can always escape the graph in finite time, but Squeaky can be trapped forever if he cannot count properly.

Peter Fulla (University of Oxford)
The complexity of Boolean surjective VCSPs

Boolean-valued constraint satisfaction problems (VCSPs) are discrete optimization problems with the goal of finding an assignment of Boolean (0/1) labels to variables that minimizes the objective function given as a sum of constant-arity constraints. In the surjective setting, an assignment is additionally required to use each label at least once. For example, the minimum cut problem falls within this framework. We give a complexity classification of Boolean surjective VCSPs parameterized by the set of available constraints.

Chris Hickey (University of Warwick)
Annotated Streaming Protocols for Data Analysis

As the popularity of outsourced computations increases, concerns about accuracy and trust between the client and the cloud computing service become ever more relevant. Our work aims to provide faster and more practical methods to verify the analysis of large data sets, where the client's memory costs are independent of the size of the data set. We do this by using annotated data streaming methods, in which the cloud computing service provides a short proof alongside the results which can be used to confirm the correctness of the computation. We supply an optimally efficient protocol for verifying matrix multiplication, and use this to provide protocols for ordinary least squares (OLS) and principal component analysis (PCA).

James Hoey (University of Leicester)
Reversing Simple Imperative Programs

We propose an approach for reversing simple imperative programs. Inspired by the Reverse C Compiler of Perumalla, we produce both an augmented version and a corresponding inverted version of the original program. Augmentation involves saving necessary information in an auxiliary data store, and maintaining segregation between this reversal information and the program state, whilst never altering the data store in any other way than that of the original program. Inversion uses this information to revert the final program state to the state as it was before execution. At the same time, the final auxiliary store is reversed to its initial state, as the reversal information is removed as it is used. We prove that augmentation and inversion work as intended, and illustrate our approach with several examples. Potential applications include PDES and debugging.

Chris Jefferson (University of St Andrews)
Canonical Images of Sets in Permutation Groups

Many combinatorial and group theoretic problems are equivalent to finding, given a group G which acts on a set S and two members X and Y of S, if there is a member of G which maps X to Y. If we make S equal to the set of all graphs, and G the group which permutes the vertices of the graphs, then this is equivalent to the graph isomorphism problem. In this talk I consider a similar problem: considering S to contain sets instead of graphs, but allowing G to be any group. When solving this problem in practice, we often use a "Canonicalizing Function" which maps elements of S to a representative in their orbit under G; if we can map from X to Y, both X and Y will have the same canonical image. When we have many elements of S to consider, using a canonicalising function can greatly improve performance. I demonstrate a family of new algorithms for finding canonical images. These algorithms make use of the orbit structure of the group to efficiently reduce the amount of search which must be performed to find a canonical image.

Georgina Elena Lungu (Royal Holloway)
Coercive Subtyping in Signatures

It has been shown that adding subtyping to type theories with canonical objects, such as Martin-Löf's type theory and Luo's UTT, requires a different understanding than the one employed by programming languages in the form of subsumptive subtyping. I present a type system with coercive subtyping entries in signatures which can be used to embed a subsumptive subtyping system, giving a way to reason about it in type theories with canonical objects. At the same time the system can itself be embedded in the previously developed coercive subtyping system, hence bridging between the practical notion of subsumptive subtyping and the theoretical coercive subtyping.

Dylan McDermott (University of Cambridge)
Effects for Lazy Languages

Effect systems augment types with information about the behaviour of programs. They have been used for many purposes, such as optimizing programs, determining resource usage, and finding bugs. So far, however, work on effect systems has largely concentrated on call-by-value languages. We consider the problem of designing an effect system for a lazy language. This is more challenging because it depends on the ability to locate the first use of each variable. Coeffect systems, which track contextual requirements of programs, provide a method of doing this. We describe how to track variable usage in a type system for a call-by-need lambda calculus using coeffects. We then add effects to the resulting system, allowing work that has been done on effect systems for call-by-value languages to be applied to lazy languages. We also show that classical strictness analysis appears as a special case of our lazy effect system.

Kitty Meeks (University of Glasgow)
Approximately enumerating small witnesses

Although much of the theory of computational complexity focusses on decision problems ("Does this problem have at least one solution?"), in many applications we actually need to know how many such solutions there are, or indeed to find all solutions. Of course, the problems of counting and enumerating all solutions are at least as hard as the corresponding decision problem, and in many cases (up to standard complexity theoretic assumptions) these versions of the problem are provably harder than the decision version. The difficulty of counting solutions exactly has led to extensive research into the feasibility of approximate counting, but as yet there is no accepted definition of a corresponding notion of "approximate enumeration". In this talk I propose a definition of approximate enumeration, and show how recent work on the enumeration of small witnesses using a deterministic decision oracle (presented at IPEC 2016) can be adapted to enumerate small witnesses approximately when only a randomised decision procedure is available.

Tobias Rosenberger (Swansea University)
Formal semantics for UML State Machines

UML diagrams are widely used for specifying the desired structure and behaviour of software. However, it is not entirely clear what such a specification means. UML still has no formal semantics and the informal semantics are not precise enough to be formalised in a straightforward way.

One approach for solving this problem is based on the theory of institutions, a formalisation of the notion of a logic. The idea is to define an institution for each UML diagram type, thereby increasing modularity and reducing the complexity of each individual logic. Institutions come with a notion of semantics-preserving translations both between different signatures of one logic and between different logics. This allows for heterogeneous reasoning and tool support (theorem provers, model checkers, code generation, etc.).

We present a modal logic for classes of UML state machines which allows us to control the presence or absence of transitions. With this approach we fix a violation of the preservation of semantics under translation between signatures which appears in previous attempts at institutionalising UML state machines.

Sarah Sigley (University of Leeds)
The Relative Proof Complexity of Modal Resolution Systems

Proof complexity measures how efficiently theorems can be proved in a given proof system. We compare the strength of two proof systems via polynomial simulations; given two proof systems P and Q, we say P polynomially simulates Q if, given any Q proof, we can efficiently construct a corresponding P proof. Whilst proof complexity has traditionally focused on proof systems for propositional logic, recent work has been carried out regarding the proof complexity of non-classical logics, including modal logics. A motivation for studying the proof complexity of modal logics is that they are suited to a wide variety of applications throughout computer science.

Over the past 30 years, various resolution calculi have been proposed for modal logics, however no work has been carried out regarding the complexity of these calculi. I give an overview of two modal resolution calculi, proposed in 1989 and 2007 respectively. I then discuss the relative complexity of these calculi, outlining a proof that they both polynomially simulate one another and so are polynomially equivalent, despite being technically rather different. To conclude I discuss whether certain lower bound proving techinques for propositional resolution might also be applied to modal resolution systems.

Matúš Tejiščák (University of St Andrews)
Adding Erasure to Dependently Typed Programming

In dependently typed languages, we often express algorithms in ways amenable to reasoning; e.g., we program with views, and add indices to type families. Our programs compute with more data – views, proofs, indices – and thus may end up asymptotically slower (e.g. exponential vs. linear). Experience indicates that this problem is important for practical programming with dependent types.

In this talk, I show a dependently typed calculus that has erasure built into its type system, along with inductive type families and pattern matching. The native erasure support allows us to infer erasure annotations from a program, check consistency of these annotations, and erase everything marked as runtime-irrelevant before execution, thus recovering the intended run-time behaviour. On the side, we get features like dependent erasure and forms of erasure polymorphism.

Using erasure allows us to write dependently typed programs using the devices we normally use to obtain correctness guarantees, but without the burden of time and memory complexity pessimisation caused by runtime-irrelevant structures present in the executed program.