Titles and abstracts for *Logic, Complexity and Automation*

Invited speakers

Patrick Baillot. *On a type-based complexity analysis of subrecursive programs*

We present a method for the time complexity analysis of higher-order functional programs, based on a type system. The source language we consider is actually a variant of Godels System T, so subrecursive programs, enriched with references. We equip it with a linear dependent type and effect system, called dlT, that can estimate the complexity of programs, as a function of the size of their inputs. The type system is intentionally sound, that is to say it over-approximates the complexity of executing the programs on an abstract machine. Moreover, we provide a sound and complete type inference algorithm which critically exploits the subrecursive nature of dlT. If time permits we will illustrate the expressivity of this language by analysing the complexity of a cryptographic reduction.

Based on joint work with Gilles Barthe and Ugo Dal Lago.

Arnold Beckmann. *Provably total NP search problems of Bounded Arithmetic*

Bounded Arithmetic forms a collection of weak theories of Peano Arithmetic related to complexity classes of functions like the Polynomial Time Hierarchy. Many connections between Bounded Arithmetic and important questions about complexity classes have already been established. Recent research considers total NP search problems in the context of Bounded Arithmetic. Total NP search problems have been studied by Papadrimitriou et al as a means to characterise the complexity of natural search problems which cannot be analysed as decision problems.

In my talk I will briefly review the research programme of characterising the provably total NP search problems in Bounded Arithmetic, and explain why it is important for current research questions in the area. I will aim to describe a particular result about the provably total NP search problems of a Bounded Arithmetic theory related to PSPACE reasoning.

Ulrich Berger. *Correctness vs. Efficiency*

Programs extracted from formal proofs come with a certificate of their correctness. Extra information about their computational efficiency can be obtained by either carrying out an additional analysis by showing that the extracted program can be defined in a restricted term system that defines functions of restricted complexity only such as PV-omega [1] for higher type computation over the integers, or stream computation [2], or else working with a restricted proof calculus that extracts programs of restricted complexity only such as in [3].

We review recent examples of program extraction in the area of parsing, imperative programming and exact real number computation under the aspect of efficiency discussed above.

[1] S. Cook. Computability and Complexity of Higher Type Functions. In Logic from Computer Science, Proceedings of a Workshop held November 13-17, 1989, Y.N. Moschovakis, editor, Vol. 21, pages 51-72, Springer, 1992.

[2] D. Ramyaa, R. Leivant. Feasible functions over co-inductive data. WoLLIC 2010, Brasilia, Brazil, July 6-9, LNCS 6188, pages 191-203. Springer, 2010.

[3] K. Aehlig, U. Berger, M. Hofmann, and H. Schwichtenberg. An arithmetic for non-size-increasing polynomial-time computation. Theoretical Computer Science, 318(1-2):3-27, June 2004.

Silvia Steila. *Reverse mathematical bounds for the termination theorem*

In 2004 Podelski and Rybalchenko expressed the termination of transition based programs as a property of well-founded relations. The classical proof by Podelski and Rybalchenko requires Ramseys Theorem for pairs which is a purely classical result, therefore extracting bounds from the original proof is non-trivial task.

We investigate the termination analysis from the point of view of Reverse Mathematics. By studying the strength of Podelski and Rybalchenko Termination Theorem we can extract some information about termination bounds.

Joint work with Keita Yokoyama

Florian Zuleger. *Automated complexity analysis with the size-change abstraction*

The size-change abstraction (SCA) is a popular program abstraction for automated termination analysis, and has successfully been implemented for imperative, functional and logic programs. Recently, it has been shown that SCA is also an attractive domain for the automated analysis of the computational complexity of programs. In this talk, I will present theoretical results on automated complexity analysis with SCA. I will show that the computational time complexity of a given abstract program is decidable: the maximal length of any sequence of transitions is exactly of asymptotic order $\Theta(N^r)$, where r is an effectively computable rational number and where N is the maximal value of the variables in the program. This result is obtained by translating abstract programs into max-plus automata and analyzing their asymptotic behavior. I will further present a construction of asymptotically precise ranking functions for the special case of deterministic size-change systems. This construction resembles an iterated application of the powerset construction known from automata theory.

Project participants (internal)

Stéphane Gimenez. *A simple protocol for parallel computation*

We present a protocol for generic parallel computation which was directly inspired by the structure of interaction nets. This protocol relies on logic formulas for type-safety. Moreover resource annotations can be added to facilitate the automated distribution of the computation.

Alexander Maringele. *FLEA dismantlement*

FLEA is a tiny automated first order logic prover with equality near completion. A general implementation overview will be given and working features will be presented.

Georg Moser. *Complexity of acyclic term graph rewriting*

Term rewriting has been used as a formal model to reason about the complexity of logic, functional, and imperative programs. In contrast to term rewriting, term graph rewriting permits sharing of common sub-expressions, and consequently is able to capture more closely reasonable implementations of rule based languages. However, the automated complexity analysis of term graph rewriting has received little to no attention. With this work, we provide first steps towards overcoming this situation. We present adaptions of two prominent complexity techniques from term rewriting, viz, the interpretation method and dependency tuples. Our adaptions are non-trivial, in the sense that they can observe not only term but also graph structures, i.e., take sharing into account. In turn, the developed methods allow us to more precisely estimate the runtime complexity of programs where sharing of sub-expressions is essential.

Thomas Powell. *Complexity in higher types*

One of the main topics of my research to date has been the study of recursion in higher types. A natural question which arises from this is: What is the complexity of a higher type functional? This is a non-trivial problem: even the concept of computability does not have a universally accepted definition for higher types, rather there exist a selection of contrasting approaches. The same is true for complexity.

In this talk I will outline, on an informal level, some personal thoughts about higher order complexity. My starting point will be the idea that the complexity of a general higher order functional should be expressed by another higher order functional. Therefore we should have some notion of a sound transformation which maps functionals to their complexities. Ultimately I aim to discuss in quite abstract terms how such transformations and the corresponding soundness proofs can be constructed. Much of this is inspired by the recent research by Danner et al. on providing a denotational cost semantics for higher order functional languages.

Michael Schaper. *T. B. A.*

In this talk we discuss Term Based Abstractions for the automated complexity analysis of programs with heap.

Maria Schett. *Trees and graphs, sharing and termination*

When moving from the tree structure of terms to a graph structure of term graphs, one can share equal subterms/graphs. However, this influences the potential rewrite steps, and therefore the termination behaviour. We are interested in termination techniques directly for term graph rewriting, and present a lexicographic path order for term graphs.

Manuel Schneckenreither. *Amortized resource analysis meets term rewrite systems*

Amortized resource analysis yields highly accurate resource complexity bounds which are commonly much better than the ones by other resource analysis techniques, like the worst-case analysis. During the last decades techniques for amortized resource analysis were developed and transformed to be compatible with arbitrary data structures and (typed) Term Rewrite Systems (typed TRS). This presentation briefly explains the main concepts of the theory and introduces the latest developments of a prototype for the univariate polynomial analysis of TRS.

Project participants (external)

Guillaume Bonfante. *On the complexity of regular languages*

We explore the notion of genus of regular languages. After we show that it gives rise to a complexity notion of regular languages, we try to discover some invariants for that notion. In particular, we relate it to a notion in graph theory which is called di-emulators.

Flavien Breuvart *On higher-order probabilistic subrecursion*

We study the expressive power of subrecursive probabilistic higher-order calculi. More specifically, we show that endowing a very expressive deterministic calculus like Gödels T with various forms of probabilistic choice operators may result in calculi which are not equivalent as for the class of distributions they give rise to, although they all guarantee almost-sure termination. We end with some considerations about function representation: essentially, the expressive power of the considered calculi is the same as that of T.

Jürgen Giesl. *Lower bounds for runtime complexity of term rewriting*

We present a novel approach to deduce lower bounds for (worst-case) runtime complexity of term rewrite systems (TRSs) automatically. Inferring lower runtime bounds is useful to detect bugs and to complement existing methods that compute upper complexity bounds. A crucial idea of our approach is to search for

"decreasing loops". Decreasing loops generalize the notion of loops for TRSs, and allow us to detect families of rewrite sequences with linear, exponential, or infinite length. We implemented our approach in the tool AProVE and evaluated it by extensive experiments.

This is joint work with Florian Frohn, Jera Hensel, Cornelius Aschermann, and Thomas Ströder.

Nao Hirokawa. *Automated normalization analysis for term rewriting*

A term rewrite system is called normalizing if every term has a normal form. In this talk, which is based on joint work with Ryoko Watanabe, I will present our normalization prover Nort. Its key feature is a modularity-based decomposition, which enables us to combine several methods to decide normalization.

Johannes Waldmann. *Two concrete challenges in complexity analysis of actual Haskell code*

I will propose two concrete questions regarding resource consumption of Haskell libraries (for balanced search trees, for pretty-printing) that are widely used. In both cases, an actual answer would be good, and a method for giving it automatically would be even better - but I have no idea how to obtain either.