The impact of the lambda calculus

Speaker: Henk Barendregt

Abstract:

The lambda calculus was conceived by Alonzo Church as a foundation for mathematics based on functions. The first formulation of the theory turned out to be inconsistent. Restricting to the computational part of mathematics the system became consistent. Church, helped by his student Kleene, made plausible that any computable function could be represented in lambda calculus. Using this, the first non-computable, but fully specified function could be defined. Alan Turing showed that lambda calculus calculations can be performed by a Turing machine and vice versa. Haskell Curry provided types for combinators, which was generalized by Church to a typing system for for lambda terms. Finally Dick de Bruijn used lambda terms to represent proofs.

All this has led to the technology of functional programing and automated proof-verification. Functional programs are powerful, as one can use arbitrarily complex notions having a clear meaning with economic notation and interface. Machine verified mathematics brings the precision of the subject a quantum leap ahead. Examples of both will be given.

Turing and Non-Turing Computers: A Tale of Two Complexities

Speaker: Ed Blakey

Abstract:

The scope of *Computing 2011* includes two topics-*different models of computing* and *complexity*-that we combine in the present talk. Specifically, we demonstrate that unconventional computers (that is, systemsincluding quantum, analogue, DNA/chemical and optical computersthat do not adhere to models polynomially equivalent to the Turing machine) demand computational complexity analysis that differs from that of Turing machines, random-access machines and equivalent. Indeed, simply following the Turing-style pattern of complexity analysis can lead one to conclude that, for example, certain factorization systems enjoy *polynomial* complexity; this is the case not only with computers such as Shors theoretically sound but (to date) practically unimplementable quantum system, but also with an analogue computer (which we describe in the talk) that in fact suffers from *exponential* complexity (which fact becomes evident once a suitable, model-independent notion of complexity is adopted). We present in this talk just such a model-independent framework for computational complexity theory.

Can We Pass the Turing Test Using Church's Lambda Calculus?

Speaker: Pierre Bourreau

Abstract:

"Can machines think?" In 1950, Alan Turing addressed this question under the definition of a test, well-known as the Turing test: can we imagine machines that can speak the same language as human, and would it be possible to make a difference between a communication with such a machine and a person? This problem relates to the Church-Turing thesis, asking whether human cognition can be simulated using computable functions.

One way of reformulating this question is: "Are human languages computable?". Such a problem can be divided in two: is it possible to compute the syntax of languages, and is it possible to compute the meanings of the sentences? In the 1970's, Richard Montague claimed that human languages should be computable in a way similar to programming languages. One of his main ideas was to use the lambda- calculus and Church simple type theory to describe meanings. As a result Montague's claim amounts to answer positively to the question of computability of human languages.

I will describe a formalism, proposed by de Groote and Muskens, and called lambda-grammars. This formalism revisits Montague's proposal by extending the use of the lambda-calculus to the syntactic derivations of languages. Lambda-grammars can be seen as an extensions of string grammars or tree grammars, and enjoy many nice computational properties. Indeed, the membership problem is decidable in the context- free version of these grammars. This shows not only that meanings of sentences are computable, but also that text generation from meaning representation is computable, giving a partial answer to Turing's problem. Furthermore, for some restricted classes of lambda-terms these problems can be efficiently solved. All these results are based on strong typing properties. The overall technique can be described under the slogan: "Parsing as Type cheking". I will give an overview on these techniques and show how they extend the usual techniques for parsing.

Termination Analysis–Turing Machines, Register Machines, and Real-Life Programming Languages

Speaker: Stephan Falke

Abstract:

Turing showed in his 1936 paper that the halting problem for Turing machines is undecidable. This results extends to other computational models like register machines or real-life programming languages. It is nonetheless possible to de-

velop methods that often succeed in proving that a given program terminates for all possible inputs. This talk discusses our recent work on developing a method for the termination analysis of real-life programming languages.

A Formalization and Proof of the "Extended Church-Turing Thesis"

Speaker: Evgenia Falkovich

Abstract:

One of the important parameters in the description of a modern algorithm is its complexity. This analysis strongly depends on the underlying computational model and domain representation. But, despite that, we are used to consider complexity classes like P, NP, or Exp to be well defined. This invariance relies on the famous "Extended Church-Turing Thesis", which asserts that every "effective" algorithm can be simulated by a Turing machine with only a polynomial overhead in the number of steps. True, this thesis is satisfied by all the usual "effective" models, and makes one optimistic that the hypothesis holds generally. Yet, the intriguing possibility of the existence of more powerful, but still effectively realizable, computational models than those we have today, remains open. To prove the thesis from first principles, one should not rely on any specific computational model, but rather capture the generic notion of effective algorithm. We suggest an axiomatic approach, which allows us to investigate the notion of effective algorithm on the most basic, yet most intuitive, level. The proof that every effective algorithm can be efficiently simulated by a Turing machine is accomplished by emulating an effective algorithm via an abstract state machine, and simulating such an abstract state machine by a random access machine with only logarithmic overhead. This is a joint work with Nachum Dershowitz.

Newmans typability algorithm

Speaker: Herman Geuvers

Abstract:

In 1943, the mathematician Newman (now best known from "Newman's Lemma" in rewriting theory) published a typability algorithm that also applies to simple type theory: it decides for an untyped lambda term, whether it can be given a type using simple types. The algorithm was left unnoticed until it was recently rediscovered by Hindley in 2008. (In 1944 Church reviewed Newmans paper in JSL, but he clearly didn't think much of it.) From our present day perspective, a remarkable thing is that Newman's algorithm decides typability without computing a type.

We give a modern presentation of the algorithm, which includes also a concise graphical description. We prove the correctness of the algorithm and we show that it implicitly computes the (Hindley-Milner) principal type. We also show how the typing algorithm can be extended to other type constructors. In fact, Newman's algorithm is in structure close to Wand's (1987) version of the typing algorithm: the typing problem is separated in two phases: (1) create a set of equations between type expressons, (2) solve these equations using unification. We show that Newmans algorithm actually includes a unification algorithm.

Problems with the Halting Problem

Speaker: Eric C.R. Hehner

Abstract:

Either we leave the definition of the halting function incomplete, not saying its result when applied to its own program, or we suffer inconsistency. If we choose incompleteness, we cannot require a halting program to apply to programs that invoke the halting program, and we cannot conclude that it is incomputable. If we choose inconsistency, then it makes no sense to propose a halting program. Either way, the incomputability argument is lost.

Pure Pointer Programs

Speaker: Martin Hofmann

Abstract:

We study the computational power of programs that mainpulate a constant number of pointers that are given as an abstract datatype, i.e. can only be followed and tested for equality. The arithmetic representation of such pointers is not accessible. As a partial substitute, we provide a mechanism for iterating over all pointers into a data structure (eg a graph) similar to the familiar iterator pattern from object-oriented programming. This construct does not allow one to recover pointer arithmetic (as does for example a total order on pointers in the presence of deterministic transitive closure).

In this talk, we survey recent and ongoing work on this computational model. In particular, we show that

- deterministic transitive closure (DTC) logic on locally-ordered graphs can be evaluated in this model
- undirected reachability cannot be solved in this model (as a corollary, DTC-logic cannot define undirected reachability)
- in the presence of recursion, one can decide reachability and evaluate formulas of transitive closure logic

- even in the presence of recursion the computational power is still weaker than LOGSPACE.

Ultimately, this work is aimed at a better understanding of the relationship between LOGSPACE (constant number of pointers) and PTIME (unbounded number of pointers). Even if a rigorous separation of these two classes is currently out of reach one could show that a hypothetical LOGSPACE algorithm for a PTIME complete problem cannot fall into certain patterns.

This is joint work with Uli Schöpp and partially with Ramyaa Ramyaa.

Descriptional Complexity–On Trade-Offs Between Descriptional Systems

Speaker: Markus Holzer

Abstract:

A short survey of the main aspects and results regarding the relative succinctness of different representations of languages, such as, e.g., finite automata, regular expressions, and descriptional systems from a more abstract perspective is given. Basic properties of these descriptional systems and their size measures are addressed. The trade-offs between different representations are either bounded by some recursive function, or reveal the phenomenon that the gain in economy of description can be arbitrary. In the latter case there is no recursive function serving as upper bound when changing from one descriptional system to an equivalent other one.

Equality in the Dependently Typed Lambda Calculus: An Introduction to Homotopy Type Theory

Speaker: Nicolai Kraus

Abstract:

While languages such as Haskell and ML are based on the simply typed lambda calculus, others, such as Agda and Epigram, go one step further by using dependent types. Especially with regard to the Curry-Howard correspondence, this might be desirable as dependent pair types and dependent function types are the equivalent of existential and universal quantification. Hence dependently typed languages are used in proof assistants such as Coq or Agda. Several questions about equality arise here. Clearly, equality in general cannot be decidable in a reasonable rich system. We therefore need equality proofs, which are just terms of the corresponding equality type. A good question here is to ask for

the number of such proofs for an equality statement. Conventionally, it was assumed that there is at most one ("Uniqueness of Identity Proofs"). Recently, Fields Medal winner Vladimir Voevodsky suggested to give up this principle, thereby revealing a connection between logic and topology (more precisely type theory and homotopy theory). This may lead to a new foundation of formal reasoning where equivalent structures can be identified.

Reactive Turing Machines

Speaker: Bas Luttik

Abstract:

Until the 1970s, a computer was a machine that computed a function: input was given at the beginning, a deterministic algorithm was carried out, and output was produced at the end. Today, a computing system is reactive and concurrent: it is not supposed to terminate, heavily depends on interaction with its environment, and often consists of many sub-components running concurrently. The Turing machine model accurately formalizes which functions can be computed by a computer, and thus also provided for an accurate conceptual model of computing, when computing was still primarily about the computation of functions. We argue that, now that reactivity and concurrency have become essential ingredients of computing, the Turing machine model as a model of computing needs an upgrade.

We propose reactive Turing machines, an extension that facilitates reactivity and concurrency. Formally, the extension consists of a facility to declare every transition to be either observable, by labeling it with an action symbol, or unobservable, by labeling it with a tau. Typically, a transition labelled with an action symbol models an interaction of the reactive Turing machine with its environment (or some other reactive Turing machine), while a transition labelled with tau refers to an internal computation step. A conventional Turing machine can be regarded as a special kind of reactive Turing machine in which all transitions are declared unobservable.

In my talk I will present reactive Turing machines, illustrate how they incorporate reactivity and concurrency, and discuss their expressiveness and the existence of universal reactive Turing machines.

Why IT Security Cannot Have a Universal Model of Computation

Speaker: Jörn Müller-Quade

Abstract:

The talk discusses the use of quantum mechanics, noise and the limitation of the speed of light to implement cryptographic functionalities. These schemes imply that one cannot hope for one single universal model of computation covering all aspects of cryptography/security. However, even the second best solution, i.e., several universal models of computation depending on the techniques used in the protocol, cannot be achieved. One can find such models of computation for honest participants of the protocol, but in security one important direction of research looks at the ways in which an attacker can attack by leaving the model. Classical turing machines are enough to describe honest parties in classical protocols, however, an attacker could use quantum technology or exploit details of the physical implementation of the protocol machines. The latter attacks are called side channel attacks and limiting the attacker to a specific model of computation could result in provably secure cryptographic protocols which are insecure when used in the real physical world.

Still, due to the separation of side channels and ordinary attacks universal models of computation are widely used in cryptography and we should celebrate this today.

Label Placement in Dynamic Maps

Speaker: Martin Nöllenburg

Abstract:

The problem of optimally placing textual labels in geographic maps has been studied in computational geometry and cartography for more than 20 years. Still, the focus was mostly on static maps. Only recently, with today's technological advancements, new labeling problems arise in dynamic maps that enable the user to interactively navigate the map. We present an algorithmic approach for optimally labeling point features under zooming and panning operations. We use the boundary labeling model, in which labels are placed on the boundary of the map. Corresponding points and labels are connected by rectilinear one-bend curves called leaders.

Planarity of Partially Embedded Graphs

Speaker: Ignaz Rutter

Abstract:

We study the following problem: Given a planar graph G and a planar drawing of a subgraph of G, can such a drawing be extended to a planar drawing of the entire graph G? This problem fits the paradigm of extending a partial solution to a complete one, which has been studied before in many different settings. Unlike many cases, in which the presence of a partial solution in the input makes hard an otherwise easy problem, we show that the planarity question remains polynomial-time solvable.

Further, following the famous work of Kuratowski, who showed that there essentially exist only two minimally non-planar graphs, namely $K_{3,3}$ and K_5, we introduce a containment relation for partially embedded graphs and show that there only exist finitely many minimal non-planar partially embedded graphs.

Combining the above two results yields an efficient certifying algorithm that for a given partially embedded graph either gives a planar embedding for the whole graph or decides that such an embedding does not exist. In the latter case the algorithm extracts as a proof of non-embeddability a minimal forbidden substructure from the input graph.

Introduction to Quantum Computing

Speaker: Giannicola Scarpa

Abstract:

In the last decades a new model of computation based on quantum mechanics has gained attention in the computer science community. We give an introduction to this model and, with the help of some examples, explain why it seems to invalidate the complexity-theoretical Church-Turing thesis. We also provide an overview of the main results and open problems in the field.

A Modern Perspective on Turing's Unorganized Machines

Speaker: Christof Teuscher

Abstract:

Turing proposed unorganized machines and evolutionary search in a 1948 National Physical Laboratory (NPL) report entitled "Intelligent Machinery". These machines have very similar properties to what is today known as random Boolean networks, which have been used as models for genetic regulatory networks. In

addition, the concept of (self-) assembling simple compute nodes that are interconnected in unstructured ways has gained significance with the advent of nano- and molecular electronics over the last decade. In this talk I will first present Turing's various original unorganized machines, extensions to them, and then relate the work to contemporary random Boolean networks, nano- and molecular electronics, and computing theory. I will illustrate that many of Turing's original ideas are more than ever current, influential, and deeply fascinating.

Computability in 1936: Pioneering Papers and Perspectives

Speaker: Wolfgang Thomas

Abstract:

In the first (and main) part of this talk, we review the emergence of a comprehensive concept of algorithm that preceeded (and culminated in) the pioneering work of Church, Turing, Kleene and Post: This remarkable "confluence of ideas in 1936" (R. Gandy) is based on integrated understanding of "computation", starting with Leibniz, where arithmetic and symbolic derivations of logic are merged. In the final part of the talk we discuss selected more recent developments that arose in computer science (but are also rooted in work of Church and Turing), in particular ideas of infinite and reactive computations.