Dagstuhl Seminar No. 01411

Proof Theory in Computer Science

organized by

Reinhard Kahle (Tübingen)
Peter Schröder-Heister (Tübingen)
Robert F. Stärk (Zürich)

1 Motivation

Proof theory has long been established as a basic discipline of mathematical logic. In recent years it has become increasingly relevant to computer science. The deductive apparatus provided by proof theory has proved to be useful both for metatheoretical purposes and for practical applications in various fields of computer science.

The aim of this conference is to assess which role proof theory is already playing in computer science, and which role it might play in further developments. Is proof theory going to be the most preferential approach to the logical foundations of computer science? Does is provide viable alternatives in areas where model-theoretic approaches are predominant?

A central focus of the conference may be captured by the slogan *logics for programs*, i.e. the proof theoretic approach in dealing with design, development and application of programming languages.

Major divisions of PTCS are the following (but this list is not intended to be exclusive):

- The *proofs as programs* paradigm in general
- Typed and untyped systems related to functional programming
- Proof-theoretic approaches to logic programming
- Proof-theoretic ways of dealing with computational complexity
- Proof-theoretic semantics of languages for specification and programming
- Foundational issues

Proof theory is not a uniform subject at all. The list of invited participants includes researchers from different research paradigms. In particular, we are inviting both proof theorists in the more traditional "theoretical" or "mathematical" sense, and computer scientists using proof theoretic tools in the area of deduction.

2 Final Programm

Monday

9:00-9:45	Roger Hindley The Birth of Combinators and Lambd
9:45	*Coffee break*
10:15-11:00	Thierry Coquand A finitary subsystem of polymorphic λ-calculus
11:15-12:00	Roy Dyckhoff Cut elimination and explicit substitutions
12:15	*Lunch*
14:00-14:45	Jaco van de Pol Equational Binary Decision Diagrams
14:45-15:30	Andreas Weiermann Analytic combinatorics and proof theory
15:30	*Coffee break*
16:00-16:45	Arnold Beckmann Well-founded principles in weak arithmetics
17:00-17:45	Lev Gordeev Proof Theory and Post-Turin analysis
18:00	*Dinner*

Tuesday

9:00-9:45	Helmut Schwichtenberg Feasible computation with types
9:45	*Coffee break*
10:15-11:00	Jan Johannsen Fragments of Gödel's system $\mathcal{T}$ characterizing fast parallel computation
11:15-12:00	Lev Beklemishev On the induction for decidable predicates and related systems

12:15 *Lunch*

14:00-14:45 Geoffrey Ostrin
Exponential complexity within an elementary arithmetic

14:45-15:30 Isabel Oitavem
Implicit characterizations

15:30 *Coffee break*

16:00-16:45 Jeremy Avigad
Weak theories of nonstandard arithmetic and analysis

17:00-17:45 Sergei Artemov
Reflective λ-calculus

18:00 *Dinner*

Wednesday

9:00-9:45 Oliver Kullmann
Searching refutations and searching models in propositional logic — possible interactions

9:45 *Coffee break*

10:15-11:00 Peter Schmitt
Iterate logic

11:15-12:00 Stanislas Nanchen
A logic for abstract state machines

12:15 *Lunch*

14:00 **Excursion**

18:00 *Dinner*

Thursday

9:00-9:45 Alexander Leitsch
Cut-elmination by resolution

9:45 *Coffee break*

10:15-11:00 Wlodek Drabent
Proving correctness and completeness of logic programs — a declarative approach

11:15-12:00 Birgit Elbl
Modeling meta-logical features in a calculus with frozen variables

12:15 *Lunch*

14:00-14:45 Christian Fermüller
Proof theortic effects of the schema of identity

14:45-15:30 Anton Setzer
Coalgebras in dependent type theory

15:30 *Coffee break*

16:00-16:45 Kosta Došen
Conjunction and disjunction in categories

17:00-17:45 Ralph Matthes
A refinement of interpolation for natural deduction

18:00 *Dinner*

Friday

9:00-9:45 Robert Stärk
The problem of bytecode verification in current implementations of the JVM

9:45 *Coffee break*

10:15-11:00 Hans Leiss
Normal forms of CFGs in fixed-point-semirings

11:15-12:00 Ulrich Berger
Programs from proofs

12:15 *Lunch*

3 Abstracts

The Birth of Combinators and Lambda

by Roger Hindley, University Wales Swansea (joint work with Felice Cardone)

Despite a very natural tendency for combinatory logic to be seen as a modification of λ-calculus, combinators were actually invented in 1920, 8 years before λ. Further, they were invented at least twice : by Schönfinkel in Göttingen in 1920, by von Neuman in Göttingen (perhaps independently ?) in 1925, and by Curry in Harvard in 1927. They gave rise to two results : eliminability of bound variables (Schönfinkel and Curry), and finite axiomatizability of set theory (von Neumann). Then in about 1928, the λ-calculus was invented by Church as part of a general system intended to be an axiomatic basis for all of mathematics. This system was inconsistent, but its core was the pure λ-calculus, which gave him his proof of undecidability of the Entscheidungsproblem. This talk is based on material from an article by Cardone and Hindley on the history of lambda and combinators, for a book on the history of mathematical logic, to be published by North-Holland Elsevier.

A finitary subsystem of polymorphic λ-calculus

by Thierry Coquand, University of Göteborg (joint work with Thorsten Altenkirch)

Polymorphic λ-calculus, or system F, has a simple interpretation where types are interpreted as sets of untyped λ-terms. Such an interpretation, however, has to use strong proof theoretic principle. We present a natural subsystem of system F (no nested products), which has a finitary interpretation. This is essentially the previous interpretation above but over a suitable Kripke models. As an application, one can prove that the numerical functions representable in this subsystem are the ones provably total in PA. This is closely related to Buchholz's Omega-rule.NSString * FieldFieldTypeNil;

Cut elimination and explicit substitutions.

by Roy Dyckhoff, University of St Andrews

Cut elimination (in Gentzen's sequent calculus) and beta-reduction (in lambda calculus or, equivalently, natural deduction) are well-known to be

similar processes but without an exact correspondence. Since about 1990, the lambda calculus has been enriched by the study of explicit substitution calculi, allowing beta-reductions to be interleaved with substitution reductions. Herbelin proposed in 1994 a calculus intermediate between sequent calculus and natural deduction, suitable like the former for proof search but without the permutabilities of Gentzen's calculus; he proposed a complete cut elimination system and showed it to be confluent and strongly normalising. Work described in the talk (joint work with C. Urban) shows how to extend this system with extra rules that allow the simulation of beta-reduction.

Equational Binary Decision Diagrams

by Jaco van de Pol, CWI, Amsterdam

We allow equations in binary decision diagrams (BDD). The resulting objects are called EQ-BDDs. A straightforward notion of reduced ordered EQ-BDDs (EQ-OBDD) is defined, and it is proved that each EQ-BDD is logically equivalent to an EQ-OBDD. Moreover, on EQ-OBDDs satisfiability and tautology checking can be done in constant time.

Several procedures to eliminate equality from BDDs have been reported in the literature. Typical for our approach is that we keep equalities, and as a consequence do not employ the finite domain property. Furthermore, our setting does not strictly require Ackermann's elimination of function symbols. This makes our setting much more amenable to combinations with other techniques in the realm of automatic theorem proving, such as term rewriting.

We introduce an algorithm, which for any propositional formula with equations finds an EQ-OBDD that is equivalent to it. The algorithm has been implemented, and applied to benchmarks known from literature.

We will also discuss an extension of EQ-BDDs with defined functions. For this extension the theory is underdeveloped, but experiments show that such a system can be very useful.

Analytic combinatorics and proof theory

by Andreas Weiermann, University of Münster

Via analytic combinatorics and methods used in the average case analysis of algorithms we obtain independence results for first order Peano arithmetic and related theories. We obtain a classification of H. Friedman's miniaturization of Kruskal's tree theorem in terms of Otter's tree con-

stant 2.95576... and we get a classification of H. FRIEDMAN'S miniaturization of the transfinite induction up to ε_0. Similarly we classify hydra games and GOODSTEIN sequences and we refine the PARIS HARRINGTON theorem. We give a characterization of ϵ_0 in terms of zero one laws using methods from finite model theory. We indicate possible generalizations and we present some open problems.

Well-foundeness principles in weak arithmetics

Arnold Beckmann, TU Wien, Austria

Well-foundeness principles are essential in the characterization of proof and computational strength of theories of arithmetics like Peano arithmetic and its fragment. We describe how to make use of them in weak arithmetics, especially in bounded arithmetics defined by Buss.

For fragments T of bounded arithmetic we define their dynamic ordinal DO(T) given by those number-theoretic functions (represented by terms in the language of bounded arithmetic) such that T proofs the schema of order-induction up to $f(x)$ for all x for all $\Pi^b_1(X)$-sets. We compute dynamic ordinals of the theories $T^1_2(X)$, $S^1_2(X)$, $sR^2_2(X)$, ..., $s\Sigma^b_m$-L^mInd. This way we directly obtain separation results of relativised bounded arithmetic theories.

In the second part, we connect dynamic ordinals with definable multifunctions of the theories. The multifunctions are described by generalization of ptime witness oracle Turing machines due to Pollett, which have only restricted access to the witnessing oracle. The connection it that the Σ^b_2-definable multifunctions of T are exactly these multifunctions computable by a ptime witness oracle Turing machine with at most $\log(DO(T))$-many queries to a Σ^b_1-oracle, for all T where we can reasonably compute $DO(T)$. Also, conservativity results are obtained characterizing dynamic ordinals. Hence, we obtain a nice relation of dynamic ordinals to provable recursive functions for fragments of bounded arithmetic, quite similar to the relation of proof-theoretic ordinals to provable recursive functions of stronger subsystem of arithmetic.

Finally, we sketch how dynamic ordinals may look for other theories of bounded arithmetic, and how they may be defined.

Proof Theory and Post-Turin analysis

by Lev Gordeev, Tübingen University

We revise familiar descriptions of Turing machines M by viewing them parametric in the total number of states admitted in the vocabulary of M. Call them $p(M)$. It turns out that the computation corresponds to M with $p(M) > 2$ is a suitable iterated functional composition of the computations corresponding to the appropriate 4 machines M_0, M_1, M_2, M_3 such that $p(M_i) = p(M) - 1$ for each $i = 0, 1, 2, 3$. If M is a Ptime machine then the length of this iteration is bounded by $\#(Input)^C$ for $C = \text{const}$. We wish to specify topological structure of the computation of M by recursion on $P(M)$ via this M_0 - M_3 expansion. The initial cases $p(M) = 2, 3$ are easy, but the rest is much more involved. The desired feature is that the Boolean DNF Tautology evaluation is not continuous in any topology such obtained for $p(M) = 2, 3, 4, \ldots < \infty$. This yields as corollary P $\neq$ NP. The main idea is to describe basic environments as solutions of polynomial systems of polynomial order-relations.

Feasible programs from proofs

by Helmut Schwichtenberg, Univ. München

It is shown how to restrict recursion on notation in all finite types so as to characterize the polynomial-time computable functions. The restrictions are obtained by using a ramified type structure, and by adding linear concepts to the lambda calculus.

In the talk we simplify previous work along these lines (jointly with Stephen Bellantoni and Karl-Heinz Niggl, APAL 2000) by (1) restricting the use of the $\Box$-operator to premises of implications, and (2) – following a proposal of Bellantoni – using parse dags as our computational model.

Fragments of Gödel's System T Characterizing Fast Parallel Computation

by Jan Johannsen, Ludwig-Maximilians-Universität München (joint work with Klaus Aehlig, Helmut Schwichtenberg and Sebastian Terwijn)

A typed lambda calculus with recursion in all finite types, i.e., a fragment of Gödel's T, is defined such that the first-order terms exactly characterize the parallel complexity class NC. This is achieved by use of the appropriate

forms of recursion (concatenation recursion and logarithmic recursion), a ramified type structure and imposing of a linearity constraint.

On the induction for decidable predicates and related systems

by Lev Beklemishev, Steklov Mathematical Institute, Utrecht University

We give an overview of the results on Δ_1-induction in formal arithmetic, focusing on the still open problem of separating Δ_1-induction from Σ_1-collection.

Our approach is based on a reduction of these systems to fragments of arithmetic axiomatized by rules and studying the effects of these rules on classes of provably total computable functions. Connections of these problems with query complexity will be presented.

Exponential complexity within an elementary arithmetic

by Geoffrey Ostrin, University of Bern, Switzerland (joint work with Stan Wainer)

Our motivation comes from the variable separation applied to the function schemes of primitive recursion given by Bellantoni and Cook. Essentially this separates the recursion variables from the substitution variables. In proof theoretic terms this translates to a separation of the induction variables and the quantification variables. Our question was what would we obtain when this restriction is applied to a formal system such as Peano Arithmetic. It turns out that the provably recursive functions are now related to the Slow Growing Hierarchy, whereas the classical single-sorted results relate to the Fast Growing Hierarchy. Thus our two-sorted theory, PA(i), has provably terminating functions being the elementary functions, ϵ^3, thus alternatively we could call our theory an Elementary Arithmetic.

Further, by analysing the inductive fragments we obtain the ϵ^2 functions provably terminating in the Σ_1 fragment whereas for the level 2 (essentially Π_2) fragment this relates to these functions that are register mantric computable in time bounded by an exponential, $2^{\mathrm{Poly}(n)}$. It is hoped that a complete characterization can be soon obtained for all fragments.

Implicit characterizations

by Isabel Oitavem, Univ. Nova de Lisboa (contains joint work with S. Bellantoni)

Several machine independent approaches to relevant classes of computational complexity have been developed. All of them lead to characterizations of classes of computational complexity where the machine formulation and resources are implicit – implicit characterizations. We work in a free algebra context. That allows us to obtain implicit characterizations of classes as different as Ptime, Lspace and NC just by changing the starting free algebra. By doing so, we give a uniform approach to classes which result from processes as different as deterministic and parallel computations with time, space or time and space constraints.

The last part of this talk is devoted to Pspace. We describe and discuss implicit characterizations of Pspace.

Weak theories of nonstandard arithmetic and analysis

by Jeremy Avigad, Carnegie Melton University

In this talk, I will discuss weak higher-type theories of nonstandard arithmetic. Such theories provide a natural setting for formalizing nonstandard arguments in analysis and combinatorics, while an explicit forcing translation allows one to intepret them in their standard counterparts. This enables us to formalize interesting portions of mathematics in conservative extensions of primitive recursive arithmetic, elementary recursive arithmetic, and possibly even polynomial-time computable arithmetic.

Reflective lambda-calculus

by Sergei Artemov, The Graduate Center CUNY New York

We introduce a general purpose typed lambda-calculus which contains intuitionistic logic, typed lambda-calculus, is capable of internalizing its own derivations as lambda-terms and yet enjoys strong normalization with respect to a natural reduction system. The Curry-Howard isomorphism converting intuitionistic proofs into lambda-terms is a simple instance of the internalization property of the reflective lambda-calculus. The standard semantics of reflective lambda-terms is given by a proof system with proof checking capacities. This system is a theoretical prototype of reflective extensions of

a broad class of type-based systems in programming languages, provers, AI and knowledge representation, etc.

Searching refutations and searching models in propositional logic – possible interactions

by Oliver Kullmann, University of Wales Swansea

There is a fundamental difference between NP ("searching a model") and co-NP ("searching a refutation"). However, for SAT (e.g.), we are interested for deciding whether a propositional formula is satisfiable or unsatisfiable, and then the difference between NP and co-NP vanishes. I "conlude" that on one hand the tools for handling satisfiability and unsatisfiability are to be investigated separately, while on the other side the handling of satisfiability and unsatisfiability has to be integrated (the ultimate goal would be a generalised duality theory). As tool for handling satisfiability I propose the theory of autarkies, while for handling unsatisfiability one important tool use forced assignments. A first example for (very weak) duality it the theorem that for any clause set F and any clause $C \in F$ either these is a autarky satisfiability C, or these is a resolution refutation of F using C, but no both. A second example for interaction SAT - USAT is given by the maximal deficing of a clause-set, which yields an upper bound on resolution complexity of unsatisfiable clause-sets as well as an upper bound on the complexity of satisfying assignments of satisfiable clause-sets.

Iterate logic

by Peter Schmitt, Universität Karlsruhe

The topic of this talk evolved within the investigation into a formal semantics of the Unified Modeling Language (UML), in particular its text-based part, the Object Constraint Language (OCL). I will begin by a short review of what needs to been known from this area for the purpose of this talk. OCL looks at first very much like a notational variant of first-order logic. Closer inspection, however, reveals some interesting, non-first-order constructs. Among those figures most prominently the "iterate" construct. I will introduce "Iterate Logic", a new logic on finite structures extending classical first-order logic intended to isolate the principles of the iterate construct and study its logic properties "in vitro". I will present first results comparing "Iterate Logic" with other logics on finite structures.

A Logic for Abstract State Machines

by Stanislas Nanchen, ETH Zürich (joint work with R. Stärk)

We will discuss a logic for sequential, non distributed Abstract State Machines. Unlike other logics for ASMs which are based on dynamic logic, our logic is based on atomic propositions for the function updates of transition rules. We do not assume that the transition rules of ASMs are in normal form, for example, that they concern distinct cases. Instead we allow structuring concepts of ASM rules including sequential composition and possibly recursive submachine calls. We have shown that several axioms that have been proposed for reasoning about ASMs are derivable in our system and proved that the logic is complete for hierarchical (non-recursive) ASMs.

Cut-Elimination by Resolution

by Alexander Leitsch, TU Wien (joint work with Matthias Baaz)

A new cut-elimination method for Gentzen's LK is defined. First cut-elimination is generalized to the problem of redundancy-elimination. Then the elimination of redundancy in LK-proofs is performed by a resolution method in the following way: A set of clauses C is assigned to an LK-proof p and it is shown that C is always unsatisfiable. A resolution refutation of C then serves as a skeleton of an LK-proof p' with atomic cuts; p' can be constructed from the resolution proof and from p by a projection method. In the last step the atomic cuts are eliminated and a cut-free proof is obtained. The complexity of the method is discussed and it is illustrated how a nonelementary speed-up over Tait's method of cut-elimination can be achieved.

Proving Correctness and Completeness of Logic Programs - a Declarative Approach

by Wlodek Drabent, Polish Academy of Sciences and Linköpings universitet (joint work with M. Milkowska)

We advocate a declarative approach to proving properties of logic programs. Total correctness can be separated into correctness, completeness and clean termination; the latter includes non-floundering. Only clean termination depends on the operational semantics, in particular on the selection rule. We show how to deal with correctness and completeness in a declarative way, treating programs only from the logical point of view. Specifications used

in this approach are interpretations (or theories). We point out that specifications for correctness may differ from those for completeness, as usually there are answers which are neither considered erroneous nor required to be computed.

We present proof methods for correctness and completeness for definite programs and generalize them to normal programs. The considered semantics of normal programs is the standard one, given by the program completion in 3-valued logic.

The method of proving correctness of definite programs is not new and can be traced back to the work of Clark in 1979. However a more complicated approach using operational semantics was proposed by some authors. We show that it is not stronger than the declarative one, as far as properties of program answers are concerned.

Modeling Meta-Logical Features in a Calculus with Frozen Variables (a contribution to proof-theoretic semantics for predicational languages)

by Birgit Elbl, Univ. München

We consider predicational languages, i.e. logic programming in the sense that the central notion is the predicate expression applied to constructor terms, the computation mechanism is based on communicating partial information by instantiating variables in these terms, and the basic constructions are related to logic programming connectives and quantifiers. This does not include, however, that these ingredients have the classical meaning or that they are 'pure' in the sense of 'pure Prolog'. In particular, variables do not just represent all ground instances but turn into objects in their own right. In order to deal with these languages the binding mechanism 'freezing' is introduced. A calculus with frozen variables is used to model a language with term inspection predicates and the control conditional.

Proof theortic effects of the schema of identity

by Christian Fermüller, TU Wien

Every instance of the schema of identity is derivable from identity axioms. However, from a proof theoretic perspective the schema of identity is quite powerful. We show that from a single instance of it a family of formulas can be proofed uniformly, where no fixed number of instances of identity axioms is sufficient for this purpose.

Coalgebras in Dependent Type Theory

by Anton Setzer, University of Wales Swansea, UK

We study, how to define interactive programs in dependent type theory. We argue, that a suitable representation is in the form of non-well-founded trees with nodes labelled by commands $c : C$ which are to be executed in the real world, and branching degrees of a node with label c being the set of reponses R(c) the real world gives when command c has been executed. We then arrive at the problem, namely how to represent non-well-founded trees in dependent type theory. Standard dependent type theory has only inductive data types, so an extension is needed (an alternative would be some coding mechanism).

We introduce the notion of strictly positive functors, and consider the question of rules for the largest fixed point F_0^∞ of F. A first version formulates coiteration. Since it is difficult to define successor operations in it we develop a second version, corecursion. We then extend F_0^∞ to a functor F^∞.

Now we develop constructions for introducing atomic elements into F_0^∞ and a repeat loop. There are as well while-loops, compostion of programs and a redirect construct.

In the last part we investigate T. Coquand's principle of guarded induction. A suitable introduction-rule for F_0^∞ in this setting would be to apply a constructor μ to a function $f : (A \to F_0^\infty) \to A \to F(F_0^\infty)$, yielding a function $\mu\, f : A \to F_0^\infty$, the last fixed point of f. Because μ is a constructor, the recursion will be only evaluated one step when the elimination function elim is applied to $\mu\, f\, a$. However we will see that we don't get a normalizing version, unless we have a syntactic restriction on f, the principle of guardedness suggested by T. Coquand. We then see that repeat loops allow to define all elements which can be introduced by μ applied to guarded f, but that repeat loops are more powerful. However we can reduce repeat to intro, the introduction rule suggested above and all terms introduced by intro can be expressed as μ applied to guarded f. In follows that both principles allow to define the same elements of F_0^∞. We therefore conclude that in a formal systems the rules proposed above for corecursion are the appropriate ones, whereas μ applied to functions f is more suitable for implementations, although the latter has a syntactic condition, and is therefore not purely type theoretic. This situation similar to what we have with algebras: in the introduction rules we introduce eliminating functions by the (primitive) recursion operator, whereas in implementations full recursion plus a syntactic termination check is more suitable.

Conjunction and Disjunction in Categories

by Kosta Došen, Mathematical Institute, Belgrade

Abstract: For deciding equality of deductions involving conjunction and disjunction - this equality being induced by normalization, or cut elimination - one may use a simple graphical model of bicartesian categories in the category of relations on finite ordinals. This model vindicates up to a point the idea that two deductions should be considered equal if and only if they yield the same deduction after generalizing.

A refinement of interpolation for natural deduction

by Ralph Matthes, LMU München

Interpolation is *the* tool for reducing provable monotonicity to syntactic positivity. With the ultimate goal of finding a predicative simulation of the behavior of monotone inductive types by that of positive inductive types, one cannot restrict the attention to inhabitation (= provability via the Curry-Howard-isomorphism). One also has to be aware of proof terms. This addition of coherence information already appears in Čubrić (Archive for Mathematical Logic, 1994). We reprove the result for a variant ΛJ of λ-calculus which allows to define the interpolants by a single recursion on the build-up of normal terms of ΛJ: It has a generalized application $r^{\rho\to\sigma}(s^\rho, z^\sigma.t^\tau) : \tau$ which may be seen as the closure of the left implication rule of Gentzen under substitution. Normalization of ΛJ requires permutative conversions. The interpolation result à la Čubrić even requires some extended permutations with unknown reduction properties. Nevertheless, one gets the result on λ-calculus as a corollary.

Towards the intended application, the positivization of monotone type dependencies is presented. Some remarks on uniform interpolation and its consequences for positivization indicate future work.

The problem of bytecode verification in current implementations of the JVM

by Robert Stärk, ETH Zürich

The Java Programming language is a strongly typed general-purpose language. Java programms are compiled to bytecode instructions which are executed by the Java Virtual Machine (JVM). A correct Java compiler produces bytecode instructions which can be trustfully executed on a JVM. A

JVM, however, cannot distinguish between bytecode generated by a correct compiler and bytecode produced by a different source to exploit the JVM. This is the reason why a JVM has to verify bytecode programms before executing them. For the security of the JVM it is important that the verifier is sound, i.e. that bytecode programms which are accepted by the verifier are type-safe at run-time and cannot corrupt the memory. A bytecode verifier should also be complete. It should accept at least all bytecode programms generated by a correct Java compiler from legal Java programms. During an attempt to prove that our bytecode verifier (R. Stärk, J. Schmid, E. Börger : "Java and the JVM – Definition, Verification, Validation", Springer-Verlag, 2001) is complete we found exemples of legal Java programms which are rejected by any commercial bytecode verifier we tried. This discovery was a surprise for most experts.

Normal forms of CFGs in fixed-point-semirings

by Hans Leiß, Universität München

We consider regular μ-terms

$$t := x|0|1|(t_1 + t_2)|(t_1 \cdot t_2)|\mu x.t$$

and context-free grammars as systems $\vec{x} = \vec{t}(\vec{x}, \vec{y})$ resp. simultaneous fixed points $\mu\vec{x}.\vec{t}(\vec{x}, \vec{y})$ of such systems ($\vec{y}$ as alphabet). We show that classical normalizations of grammars like elimination of ϵ-rules, elimination of chain-rules, and Greibach normal form follow from semirings axioms, least-(pre)-fixed point properties for μ, fixed-point induction rule, and two equations $\mu x(t \cdot x + s) = \mu x(tx + 1) \cdot s$ and $\mu x(x \cdot t + s) = s \cdot \mu x(tx + 1)$. It follows that the CF6-normal forms hold not only int continuous semirings, but in all semirings having "enough" fixed points and satisfying the two equations. Idempotency of $+$ is needed for ϵ-elimination only.

Programs from proofs

by Ulrich Berger

- From a formalization of Gentzen's proof of transfinite induction up to the ordinal ε_0 we extract a program transforming any algorithm using transfinite recursion on ordinals $< \varepsilon_0$ into an equivalent, but more efficient one using higher type (Gödel) primitive recursion instead.

- From a classical proof of

$$\forall f^{\mathcal{N}\to\mathcal{N}}\exists n\ f(f(n)+1)\neq n$$

we extract a program computing a witness n from any given f.

Exercises: 1. Find the classical proof, that is, assume $\forall n\ f(f(n)+1)=n$ and derive a contradiction (easy).

2. Find (manually) finitely many terms t_i (containing f free) such that $\forall f\colon\mathcal{N}\to\mathcal{N}\,\exists i\ f(f(t_i)+1)\neq t_i$ (difficult).

In MINLOG we can extract these terms from the classical proof automatically.

- We show that the classical axiom of countable choice

$$\forall n\,\exists x^{\rho}A(n,x)\to\exists f^{\mathcal{N}\to\rho}\forall n\,A(n,f(n))$$

(A, ρ arbitrary) can be given a modified realizability interpretation (à la Kreisel/Troelstra). The idea for this is derived from a result by Berardi, Bezem and Coquand, but our construction is simpler and, apparently, more efficient.