Electronic Communications of the EASST
Volume 23 (2009)

Proceedings of the Ninth International Workshop on Automated Verification of Critical Systems (AVOCS 2009)

Proofs-as-Programs in Computable Analysis (extended abstract)

Ulrich Berger

5 pages

Guest Editor: Markus Roggenbach
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Proofs-as-Programs in Computable Analysis (extended abstract)

Ulrich Berger

u.berger@swansea.ac.uk, http://cs.swan.ac.uk/~csulrich/
University of Wales Swansea, Swansea, SA2 8PP, Wales UK

Abstract: Since the work of Brouwer, Kolmogorov, Goedel, Kleene and many others we know that constructive proofs have computational meaning. In Computer Science this idea is known as the "proofs-as-programs paradigm" or "Curry-Howard correspondence". We present examples from computable analysis showing that this paradigm not only works in principle, but can be used to automatically synthesise practically relevant certified programs.

Keywords: Proof theory, program extraction, exact real number computation, coinduction

Besides the contributions to its intrinsic domains such as philosophy and the foundations of mathematics there are many applications of logic in areas outside logic. They mainly come from logical disciplines such as set theory, model theory, computability theory and modal logic and include, for example, in mathematics, proofs of the existence of certain algebraic structures, and, in computer science, methods for specifying and verifying computing systems and classifying the complexity of algorithmic problems. These applications are widely known and form a well-established part of theoretical and practical research, in particular in Computer Science.

Less known are extra-logical applications from *proof theory*, a branch of logic that has formal proofs as its main object of study. Classic results here are bounding informations about functions whose totality (i.e. termination) is proven in a certain formal system. For example, in the case of Peano Arithmetic the growth of the function is bounded by a transfinite extension of the Ackermann Function below the ordinal ε_0 (the limit of towers of ω exponentials) [Gen43, Wai72]. If one further restricts the induction scheme to purely existential formulas, then the function is even primitive recursive [Par72, Göd58]. Still, this is of little practical interest since primitive recursion goes far beyond feasible computability. The extraction of (from a computer science point of view) more relevant information, for example polynomial time or lower complexity, can be achieved by further restricting the proof system [Bus86, CU93, Lei95, BNS00, OW05, ABHS04]. Other kinds of bounding information relevant in Approximation Theory, Functional Analysis and similar areas have been obtained rather recently on the basis of Kohlenbach's monotone variant of Gödel's Functional Interpretation [Koh08].

A further large class of applications of proof theory in computer science can be summarised under the so called "proofs-as-programs paradigm" a.k.a. "Curry-Howard correspondence" which is the observation that –under certain conditions– formal proofs can be viewed and executed as programs. This correspondence is most direct for intuitionistic natural deduction (or Hilbert-style) systems that can immediately be viewed as dependently typed lambda-calculi (or combinatory logics) [HLS72, Tro73, vD80, GLT89] and hence as functional programming languages.

Extensions of this correspondence to systems with classical logic have led to interesting techniques such as programming with continuations and control operators [FF87] and extensions of the lambda-calculus such as the lambda-mu-calculus [Par92].

In this note we want to draw the reader's attention to a very direct (and some may say bold) application of the proofs-as-program paradigm namely the extraction of practically useful programs from formal proofs. Since proofs are often very long and consist to a large part of purely logical information, it is necessary to remove these purely logical parts and retain only the computationally relevant components. This is achieved by a proof-theoretic technique known as *realisability* that goes back to Kleene and Kreisel [Kle59, Kre59]. The realisability interpretation not only extracts an executable program, but also a specification of that program and a formal proof that the specification is fulfilled.

It would be an exaggeration to claim that program extraction is already being applied in practice. However, some substantial case studies have been carried out indicating that the method is feasible and has the potential to become a viable method for safe software engineering in the future. We mention three such case studies: A fast higher-order algorithm for normalising simply typed lambda-terms has been extracted from Tait's normalisation proof [Tai67, Ber93]. The algorithm is known as "normalisation-by-evaluation" and had been discovered earlier by Schwichtenberg [BS91]. This case study was carried out independently in the proof assistants Coq, Isabelle and Minlog [BBLS06]. The extracted program is unusual because it utilises higher types of any level to perform a purely syntactic task. The second example is concerned with Dickson's Lemma a result in infinitary combinatorics that is used e.g. in Gröbner Base theory. Dickson's Lemma states that for fixed n the set of n-tuples of natural numbers is well-quasiordered (famous generalisations of this are Higman's Lemma, Kruskal's Theorem and the Graph Minor Theorem). The extracted program computes for every infinite sequence of n-tuples a pair of indices $i < j$ such that the i-th tuple is pointwise $\leq$ the j-th [BSS01]. The interesting aspect of this example is that the program is extracted from the classical i.e. non-constructive proof by Nash-Williams [NW63] using a version of Friedman's A-translation [Fri78] that translates classical into constructive proofs. Finally, we mention the extraction of programs from the Intermediate Value Theorem and the Inverse Function Theorem for continuous real functions with a positive lower bound on the slope [Sch06]. These are probable the first non-trivial examples of program extraction in Computable Analysis.

The last of the three case studies above is concerned with problems in constructive analysis, and it uses a constructive representation of real numbers by Cauchy sequences. Our recent work in program extraction is situated in constructive analysis as well, but is based on a signed digit representation [BH08, Ber09a, BL09]. More precisely, we use inductive and coinductive definitions to characterise uniformly continuous functions in such a way that one can extract an implementation of such functions by non-wellfounded trees that act as transformers of signed digit streams. A suitable adaptation of realisability to this setting is described in [Ber09b]. We have extracted from constructive proofs programs that compute high iterations of the logistic map, integrals of continuous functions, the constant π, and functions defined power series.

The latter two examples make use of a more general setting where the set of signed digits $\mathrm{SD} := \{0,1,-1\}$, which is to be thought of as the set of representations of the contractions $\mathrm{av}_d(x) := (d+x)/2$ ($d \in \mathrm{SD}$), is replaced by an arbitrary set D of endomaps on a set X. Large parts of our theory can be developed for arbitrary such structures (X,D) which we call *digit*

spaces. This greater generality not only leads to clearer proofs but also to new algorithms, the last two case studies mentioned above being examples.

The programs in the latter case study have been extracted "by hand" from proofs in the theory of digit spaces which are "nearly formal". The implementation of the theory of digit spaces using a suitable proof assistant (for example, Minlog or Coq) is a matter of future work.

The method of program extraction from proofs can be viewed as a consistent and rather radical continuation of methods of program development, advocated by Dijkstra and others [Dij97, Gri81, DJ78] where programs are correct "by construction". Whether or not these methods will eventually be accepted and used in practice remains to be seen. The experimental results obtained so far are encouraging, however.

Bibliography

[ABHS04] K. Aehlig, U. Berger, M. Hofmann, H. Schwichtenberg. An arithmetic for non-size-increasing polynomial-time computation. *Theor. Comput. Sci.* 318:3–27, 2004.

[BBLS06] U. Berger, S. Berghofer, P. Letouzey, H. Schwichtenberg. Program extraction from normalization proofs. *Studia Logica* 82:25–49, 2006.

[Ber93] U. Berger. Total Sets and Objects in Domain Theory. *Ann. Pure Appl. Logic* 60:91–117, 1993.

[Ber09a] U. Berger. From coinductive proofs to exact real arithmetic. In Grädel and Kahle (eds.), *Computer Science Logic*. LNCS 5771, pp. 132–146. Springer, 2009.

[Ber09b] U. Berger. Realisability and Adequacy for (Co)induction. In Bauer et al. (eds.), *6th Int'l Conf. on Computability and Complexity in Analysis*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, Dagstuhl, Germany, 2009.
http://drops.dagstuhl.de/opus/volltexte/2009/2258

[BH08] U. Berger, T. Hou. Coinduction for Exact Real Number Computation. *Theory of Computing Systems* 43:394–409, 2008.
doi:DOI: 10.1007/s00224-007-9017-6

[BL09] U. Berger, S. Lloyd. A coinductive approach to verified exact real number computation. 2009. To appear: Proceedings of Automated Verification of Critical Systems (AVOCS), Gregynog, 23-25 September.

[BNS00] S. Bellantoni, K.-H. Niggl, H. Schwichtenberg. Higher Type Recursion, Ramification and Polynomial Time. *Ann. Pure Appl. Logic* 104:17–30, 2000.

[BS91] U. Berger, H. Schwichtenberg. An inverse of the evaluation functional for typed λ–calculus. In Vemuri (ed.), *Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science*. Pp. 203–211. IEEE Computer Society Press, Los Alamitos, 1991.

[BSS01] U. Berger, H. Schwichtenberg, M. Seisenberger. The Warshall Algorithm and Dickson's Lemma: Two Examples of Realistic Program Extraction. *J. Autom. Reasoning* 26(2):205–221, 2001.

[Bus86] S. Buss. *Bounded Arithmetic*. Studies in Proof Theory, Lecture Notes. Bibliopolis, Napoli, 1986.

[CU93] S. Cook, A. Urquhart. Functional Interpretations of Feasibly Constructive Arithmetic. *Ann. Pure Appl. Logic* 63:103–200, 1993.

[vD80] D. van Dalen. *Logic and Structure*. Springer–Verlag, Berlin, 1980.

[Dij97] E. W. Dijkstra. *A Discipline of Programming*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.

[DJ78] B. Dines, C. Jones. *The Vienna Development Method: The Meta-Language*. LNCS 61. Springer, Berlin, Heidelberg, New York, 1978.

[FF87] M. Felleisen, D. P. Friedman. Control Operators, the SECD–Machine, and the λ–Calculus. In Wirsing (ed.), *Formal Description of Programming Concepts — III*. Pp. 193–219. Elsevier (North–Holland), Amsterdam, 1987.

[Fri78] H. Friedman. Classically and intuitionistically provably recursive functions. In Scott and Müller (eds.), *Higher Set Theory, Lecture Notes in Mathematics*. Volume 669, pp. 21–28. Springer, 1978.

[Gen43] G. Gentzen. Beweisbarkeit und Unbeweisbarkeit von Anfangsfällen der transfiniten Induktion in der reinen Zahlentheorie. *Mathematische Annalen* 119:140–161, 1943.

[GLT89] J.-Y. Girard, Y. Lafont, P. Taylor. *Proofs and Types*. Cambridge University Press, 1989.

[Göd58] K. Gödel. Über eine bisher noch nicht benützte Erweiterung des finiten Standpunktes. *Dialectica* 12:280–287, 1958.

[Gri81] D. Gries. *The Science of Programming*. Springer, 1981.

[HLS72] J. Hindley, B. Lercher, J. Seldin. *Introduction to Combinatory Logic*. London Mathematical Society Lecture Notes Series 7. Cambridge University Press, 1972.

[Kle59] S. C. Kleene. Countable functionals. In Heyting (ed.), *Constructivity in Mathematics*. Pp. 81–100. North–Holland, 1959.

[Koh08] U. Kohlenbach. *Proof Interpretations and their Use in Mathematics*. Springer Monographs in Mathematics. Springer, 2008.

[Kre59] G. Kreisel. Interpretation of analysis by means of constructive functionals of finite types. *Constructivity in Mathematics*, pp. 101–128, 1959.

[Lei95] D. Leivant. Ramified Recurrence and Computational Complexity I: Word Recurrence and Poly–time. In Clote and Remmel (eds.), *Feasible Mathematics II*. Pp. 320–343. Birkhäuser, Boston, 1995.

[NW63] C. Nash-Williams. On well-quasi-ordering finite trees. *Proc. Cambridge Phil. Soc.* 59:833–835, 1963.

[OW05] G. Ostrin, S. Wainer. Elementary Arithmetic. *Ann. Pure Appl. Logic* 133:275–292, 2005.

[Par72] C. Parsons. On *n*-quantifier induction. *Jour. Symb. Logic* 37:466–482, 1972.

[Par92] M. Parigot. $\lambda\mu$–calculus: an algorithmic interpretation of classical natural deduction. In *Proceedings of Logic Programming and Automatic Reasoning, St. Petersburg*. LNCS 624, pp. 190–201. Springer, 1992.

[Sch06] H. Schwichtenberg. Inverting monotone functions in constructive analysis. In Beckmann et al. (eds.), *CiE 2006: Logical Approaches to Computational Barriers*. LNCS 3988, pp. 490–504. Springer, 2006.

[Tai67] W. Tait. Intensional Interpretations of Functionals of Finite Type I. *The Journal of Symbolic Logic* 32(2):198–212, 1967.

[Tro73] A. Troelstra. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*. Lecture Notes in Mathematics 344. Springer, 1973.

[Wai72] S. S. Wainer. Ordinal recursion, and a refinement of the extended Grzegorcyk hierarchy. *Jour. Symb. Logic* 37:281–292, 1972.