Domain-theoretic methods for program synthesis

Case for Support

Ulrich Berger
Department of Computer Science
University of Wales Swansea
Singleton Park
Swansea SA2 8PP

Part 1 Previous research track record

I was appointed lecturer in the Department of Computer Science at Swansea in October 1999; previously I was at Ludwigs-Maximilians-University, Munich.

Research Summary

My field is Logic and its applications to Computer Science. In my research I focus on *domain theory* and *proof theory* and on the use of domains in the design of proof systems.

Domain theory has been introduced in the seventies independently by D. Scott as a foundation for the denotational semantics of functional programming languages, and by Y. Ershov as a model for continuous and computable higher type functionals. My main contribution to this field is a mathematical analysis of the notion of *totality* in domains. My results imply, for example, rather broad generalisations of the Ceitin/Moschovakis theorem saying that every total computable operation on the real numbers is continuous. They also lay the foundation to a constructive total semantics of Intuitionistic Type Theory. The main parts of my work on domains are contained in my PhD-thesis (Munich, supervised by H. Schwichtenberg, published in (3)) and my Habilitationsschrift (Munich, published in (9,10)). In my research on domains I intensively collaborated with D. Normann (Oslo) and V. Stoltenberg-Hansen (Uppsala).

Proof theory is a discipline of Mathematical Logic concerned with the analysis of formal proofs. A recent outcome of this analysis is the development of computer systems for automated or interactive theorem proving that can for instance be used for computer aided program verification. An example of such a system is the interactive theorem prover MINLOG developed by the logic group at the University of Munich (7). As a former member of this group I was mainly involved in the theoretical background steering the implementation of the system. The system also exploits the so-called *proofs-as-programs paradigm* as a logical approach to correct software development: from a formal proof that a certain specification has a solution one fully automatically extracts a program that provably meets the specification. We carried out a number of extended case studies extracting programs from proofs in areas such as arithmetic (6), graph theory (7), infinitary combinatorics (7), and lambda calculus (1,2). Special emphasis has been put on an efficient implementation of higher order rewrite rules (8), and on the optimised extraction of programs from non-constructive proofs (4,5,6,7). The logic group in Munich is in close contact and scientific exchange with research groups in Nijmegen (H. Barendregt) and Gothenburg (T. Coquand) where similar systems are being developed.

Using domains in the design of a proof system: The MINLOG system (7) is based on a domain-theoretic model of data types and higher type functionals. This has a strong impact on the development of the system. In particular the above mentioned results on totality in domains are used to set up a sound formalisation of partiality and totality. Also the implementation of the built in higher order rewrite system is directly guided by and proved sound with respect to a domain model (8) using results on dependant domains (10).

Selected publications

(1) U. Berger, H. Schwichtenberg. An inverse of the evaluation functional for typed λ–calculus. In: R. Vemuri, editor, *Proceedings of the Sixth Annual IEEE Symposium on Logic in Computer Science (LICS)*, 203-211, IEEE Computer Society Press, 1991.

(2) U. Berger. Program extraction from normalization proofs. In: M. Bezem and J.F. Groote, editors, *Typed Lambda Calculi and Applications (TLCA'93)*, LNCS **664**, 91-106, 1993.

(3) U. Berger. Total sets and objects in domain theory. *Annals of Pure and Applied Logic* **60**, 91-117, 1993.

(4) U. Berger, H. Schwichtenberg. Program extraction from classical proofs. *Logic and Computational Complexity (LCC'94)*, LNCS **960**, 77-97, 1995.

(5) U. Berger. Programs from classical proofs. In: R. Fritsch, M. Behara, and R.G. Lintz, editors, *Symposia Gaussiana. Proceedings of the 2nd Gauss Symposium*, Munich, Germany, 1993, 187-200, de Gruyter, 1995.

(6) U. Berger, H. Schwichtenberg. The greatest common divisor: a case study for program extraction from classical proofs. In: S. Berardi, M. Coppo, editors: *Types for Proofs and Programs, International Workshop TYPES'95*, Torino, Italy, 1995, LNCS **1158**, 36-46, 1996.

(7) H. Benl, U. Berger, H. Schwichtenberg, M. Seisenberger, W. Zuber. Proof theory at work: Program development in the Minlog system. In: W. Bibel, P.H. Schmitt editors, *Automated Deduction – A Basis for Applications*. 41-71, Kluwer, 1998.

(8) U. Berger, M. Eberl, H. Schwichtenberg. Normalization by evaluation. In: B. Möller, J. V. Tucker, editors, *Prospects for hardware foundations (NADA)*, LNCS **1546**, 117-137, 1998.

(9) U. Berger. Density Theorems for the Domains-with-Totality Semantics of Dependent Types. *Applied Categorical Structures* **7**, 3-30, 1999.

(10) U. Berger. Continuous Functionals of Dependent and Transfinite Types. In: B. Cooper, J. Truss, editors, *Models and Computability, invited papers from Logic Colloquium '97, Leeds, UK, 1997.* London Mathematical Society Lecture Notes Series **259**, 1-22, 1999.

Projects and research visits

Twinning Project. EU Science Plan. Collaboration of the Universities of Leeds, Munich, and Oslo. Subject: Proof Theory and Computation. Several research visits at the University of Oslo (Prof. D. Normann). Research in domain theory. 1991-1994

Sonderforschungsbereich Deduktion. Deutsche Forschungsgemeinschaft (DFG). Subject: Computer Aided Theorem Proving and Program Verification. 1994–1998.

NADA - New Hardware Design Methods. Esprit Working Group 8533. Subject: Research on mathematically sound methods for the description and design of hardware systems including modelling and foundations. 1994–1998.

Research visit at the University of Swansea, Department of Computer Science (Prof. J. Tucker). Research in domain theory. Deutscher Akademischer Austauschdienst. 1994.

Habilitationsprojekt: Continuous Functionals of Dependent and Transfinite Types. Swedish Science Foundation and DFG. Research visit at Uppsala University (Prof. V. Stoltenberg-Hansen). 1995.

Graduiertenkolleg Logik in der Informatik. DFG. Research in Program synthesis. Since April 1997.

Research visit at Monash University Melbourne, Department of Computer Science (Prof. J. Crossley). Research in program synthesis. 1998.

Conferences

I presented my work at international conferences, among them *Logic in Computer Science (LICS'91, Amsterdam)*, *Typed Lambda Calculi and Applications (TLCA'93, Utrecht)*, *2nd Gauss Symposium (Munich, 1993)*, *Logic and Computational Complexity (LCC '94, Indianapolis)*, and *Types for Proofs and Programs (TYPES'95, Torino)*. I was an invited speaker at *Logic Colloquium '97 (Leeds)*, the *Heyting Symposium, Amsterdam, 1998*, the *Workshop on Realisability Semantics (FLOC'99, Trento)*, the *International Congress on Logic, Methodology and Philosophy of Science (Krakau, 1999)*, and the *International Symposium 'Logic and Applications' (Novosibirsk, 2000).* I was also invited to the conferences *Mathematical Logic* (Oberwolfach, Germany, 1993 and 1995), and to the seminar *Topology in Computer Science: Constructivity; Asymmetry and Partiality; Digitalization (Dagstuhl, 2000).*

I was co-organiser of the workshops *Domains III* (Munich, 1997), and *Reuniting the Antipodes: Constructive and Nonstandard Views of the Continuum* (Venice, 1999, funded by the Volkswagenstiftung).

Computer Science at Swansea

The Department of Computer Science at the University of Wales Swansea is well-known internationally for its excellent research in Theoretical Computer Science. The topics of the proposed project are closely connected with work of J.V. Tucker (head of department) on logical and algebraic methods for modelling and specification and on the computability theory for topological data types, and J. Blanck (STINT (Sweden) visiting fellow) on domain representability of topological spaces. The department maintains strong links with the groups around K. Weihrauch (Hagen, Germany) and D. Lester (Manchester) doing research in computable analysis, and is regularly visited by J. Bergstra (Amsterdam, process algebra and program verification), V. Stoltenberg-Hansen (Uppsala, computability on topological algebras) and J. Zucker (McMaster, Canada, computability on topological algebras). It will also host the Fourth Workshop on *Computability and Complexity in Analysis* September 2000 [1]. This strong research profile in modelling and computable analysis makes Swansea an ideal place for this project. I will also benefit very much from the expertise of the other members of the department, in particular from N.A. Harman (term rewriting and verification of micro-processors), P.W. Grant (functional programming), and M.F. Webster (numerical analysis). The department has been rated *4A* in the most recent research assessment exercise and received the rating *excellent* for its teaching.

[1] `http://www.informatik.fernuni-hagen.de/import/cca/cca2000/`

Part 2 Description of the proposed research and its context

2.1 Background and outline of the project

It is a central aim of Computer Science to provide techniques for developing software that is guaranteed to be correct and easy to maintain. The general objective of this project is to provide foundational contributions to this aim using domain- and proof-theoretic methods.

Domain theory and proof theory are branches of Mathematical Logic with important applications in Computer Science. In *domain theory* a great variety of data types and programming concepts can be modelled and analysed. This includes nondeterminism, computations on continuous and higher type data, and the operational semantics and expressive power of functional languages [51, 48, 53, 28, 1]. On the other hand *proof theory* provides formal methods for program development. For example, the so-called 'proofs-as-programs paradigm' [2, 50, 59], according to which a constructive proof of a specification corresponds to a satisfying program, has led to a method of program synthesis from proofs which has been implemented in a number of interactive proof systems [21, 39, 42, 30, 47, 3].

It is the objective of this project to pursue current research on the above mentioned applications of domain theory and proof theory in Computer Science, and to strengthen the existing links between these fields.

In domain theory various open problems in connection with the computational and topological structure of partial and total higher type functionals will be investigated. In particular, we will study and compare different models of computation (via approximation, numbering, functional programming, restricted forms of recursion) [24, 44, 46, 26, 5], and investigate to which extent methods of studying computability on structures like metric, topological, or Banach spaces [41, 37, 55, 49, 56, 17, 54, 18, 23] can be lifted to higher types, if possible retaining equivalence results as obtained in [57].

On the proof-theoretic side it is planned to further develop and improve the present proof system [3] and its program extraction mechanism. The program extraction method used in [3] is based on a variant of Kleene's realisability [32]. This method, which is also used in [30] and suggested in [61], has a great potential of being extended and being made more flexible which apparently has not yet been fully exploited. In the project this potential, which has been observed and used already in [6] and [7], shall be thoroughly investigated and implemented.

Domains are getting involved in program synthesis by extracting domain-theoretic algorithms (i.e. computable mappings between domains) from proofs, instead of concrete programs (like e.g. in [30] and [3]). This will give us the possibility to freely 'invent' domain-theoretic realisers of critical axioms (e.g. nonconstructive choice principles) without being committed to a specific formal language. In this way, domain-theoretic insight in computational phenomena may be directly used for improvements and extensions of the formal proof calculus. In particular results on higher type functionals will be important [43, 5, 8, 9, 10], since such functionals almost inevitably occur in programs extracted from proofs.

The general idea of our domain-theoretic method of program synthesis is to separate logical from computational issues: The formal logical calculus is supposed to capture a mathematical subject and its proof methods at a high level of abstraction without being specific about computational details, whereas the computational model can be developed and studied informally using classical mathematical methods. Nevertheless, the logical and the computational world are firmly linked by the realisability interpretation. This means that classical mathematical concepts and proofs can be used more or less as they are, and need not be constructivised in order to get hold on their computational content. Note that this is in contrast to common approaches to constructive logic such as Martin-Löf Type Theory [40], which tend to integrate or even identify proofs and programs. We also would like to point out that the interaction of domain theory and proof theory via realisability suggested here is quite different from the use of realisability models as a general approach to domain theory (see e.g. [38]). While we are studying specific topics (models of the reals etc., totality, higher types) in a relatively small part of domain theory (effective consistently complete algebraic (or continuous) dcpos) the latter provides a framework for the global theory.

Case studies exploring possible areas of application will play an important role in the project. Examples of program extraction in term rewriting and infinitary combinatorics will be studied, continuing work in [6] and [14]. As a new and very promising enterprise we want to do 'proof mining' (D.S. Scott) in computable and constructive analysis [49, 16], building on our enhanced realisability interpretation and on domain-theoretic representations of the real numbers and related structures [55, 26, 17]. We expect that our strategy of separating logical from computational issues will lead to more concise and transparent formalisations of analysis. Presently, constructive formalisations of analytical proofs often suffer from complicated representations of the reals and other basic analytical concepts [20, 31, 29] reflecting a particular model of computation. The methods proposed in this project will make a formalisation of these models unnecessary, and thus will considerably facilitate program synthesis from analytical proofs.

2.2 Programme and methodology

According to the considerations above the specific objectives of the project are

(i) *to investigate domain-theoretic models of computation, in particular for higher types and reals numbers*,

(ii) *to develop a method for program synthesis by means of an enhanced realisability interpretation*,

(iii) *to show its practicability by case studies, mainly in computable analysis and infinitary combinatorics*,

(iv) *to provide an implementation.*

Below we give some more details on these objectives and their realisation.

(i) *Domain-theoretic models of computation*

We will mainly study domain models of partial and total higher type functionals which are particularly important in connection with program extraction. In view of our planned applications to computable analysis higher type functionals over a domain model of the reals will be of particular interest. Note that the problem of relating computable analysis with higher type computations is explicitly posed in [49] (Open Problem 4). We will focus on the following problems:

Equivalence of different computability concepts. Three major ways of defining computability in higher types have been considered in the literature: 1. computability via effective enumeration of finite approximations, originating in [33] and modelled in domain theory [51, 25, 28], 2. computability via numberings [24], and 3. definability by some generation scheme, or some higher order programming language [48]. For *partial* continuous functionals equivalence of these concepts has been proven in all interesting cases: The equivalence of 1. and 2. (over arbitrary effective domains as base type), proven in [25], is a generalisation of the well-known Myhill-Shepherdson theorem, and the equivalence of 1. and 3. has been proven in [48] w.r.t. parallel PCF over the base type of partial integers, and in [26] w.r.t. parallel real PCF over the base type of partial reals. For *total* continuous functionals over the *integers* the equivalence of 1. and 2. has been proven in [25] (generalisation of the Kreisel-Lacombe-Shoenfield theorem), and recently the equivalence of 1. and 3. w.r.t. (sequential!) PCF has been shown in [46]. The relations between 1., 2. and 3. for *total* continuous functionals over the *reals*, however, are unknown. Their accurate determination will be one of the major challenges of this project.

Comparison of restricted forms of recursion. Computability defined via a higher order programming language, say PCF [48] (concept 3. above), can be modified by e.g. restricting recursion to μ-recursion (= minimisation plus primitive recursion, which corresponds to 'while' loops [62]) or restricting the type level of recursion. At type level one (number theoretic functions) all these concepts are equivalent, but in higher types the situation is more diverse. In [15] many results in this direction have been obtained, which however have to be handled with care, since the interpretations of computations in Kleene's functionals [33] and in domain models are different (see e.g. the computability of the fan-functional [45]). In [10] the relation between μ-recursion and full recursion is clarified, but the effect of further restricting the type level of recursion is unclear. Answers to these semantical questions will have immediate consequences to the operational semantics of PCF, via the adequacy result in [48]. These issues are also to be related to the results in [62] on 'while' programs on partial algebras.

Higher type computation vs. computability on abstract structures. There are many approaches to computability on abstract structures, in particular to structures relevant to analysis [41, 37, 55, 49, 56, 17, 54, 19, 18, 23]. In [57] the equivalence of a number of these approaches is shown. The question is: Can these approaches be lifted to higher types, and if so, are they still equivalent?

Topological approximations to totality. For the proof of the generalised Kreisel-Lacombe-Shoenfield theorem mentioned above two properties of the total continuous functionals over the integers were crucial: density and co-density [5, 9]. Co-density can be viewed as a topological approximation to totality. In [9] a density/co-density theorem is proven that can be applied to continuous functionals over the integers and over the reals. However, this notion of co-density seems to be to weak to prove a Kreisel-Lacombe-Shoenfield theorem for the continuous functionals over the reals. The *right* topological approximation to totality still has to be found.

(ii) *Program synthesis via realisability*

In order to explain some specific features of our proposed realisability interpretation we recall the basic ideas [32, 36]: To each formula A a type $\tau(A)$ and to each proof d of A a program $\mathsf{ep}(d)$ 'realising' A is associated. The latter intuitively means that $\mathsf{ep}(p)$ is a witness for the constructive validity of A. This so-called *modified realisability* [36] can be made more flexible in several respects:

– By interpreting types as domains and realising formulas by computable domain elements (and not by integers or syntactic programs) it is possible to 'invent' realisers of critical axioms freely using domain-theoretic methods, without being committed to a prescribed programming language. The problem of relating domains with concrete

models of computation can be studied separately, for example, by proving an adequacy result like in [48] relating the domain-theoretic semantics with an operational semantics of PCF.

– By suitably parametrising and modifying the definition of realisability it will be possible to encapsulate complicated statements and their realisers, which can lead to drastic simplifications of proofs and programs. For example, given an atomic formula $P(\vec{t})$, the type of realisers, $\tau(P(\vec{t}))$, as well as the statement 'r realises $P(\vec{t})$' may be chosen arbitrarily. As for quantifiers, the statement 'r realises $\exists x A$' can be defined simple to mean '$\exists x$ (r realises A)' (similarly for $\forall$), i.e. quantifiers are 'ignored', i.p. the x is not part of the realiser r. Note that this is closely related to realisability for second-order arithmetic [60]. The standard treatment of the quantifiers can be recovered as a special case by reading e.g. '$\exists x^{\mathbf{N}} A$' as '$\exists x(N(x) \wedge A)$' and interpreting 'r realises $N(x)$' as $r = x$. However, the usefulness of these modifications lies in the interpretation of e.g. 'r realises $R(x)$' ('R' for the reals) which typically will be 'r is a representation of the (abstract) real number x'. Of course, for the extracted program it matters which representation is chosen, but this choice need not be reflected in the formal system.

Positive effects expected from the suggested modifications are that (a) the formal system can be kept simple without losing grip on computational aspects, (b) extracted programs will contain less junk, as demonstrated in [6] where a short and efficient normalisation program for the simply typed lambda-calculus is extracted, (c) classical logic can be treated more smoothly [7] avoiding expensive proof translations [27, 12, 13, 14], and (d) crucial axioms of analysis like the classical quantifier free axiom of choice will be realisable (this is inspired by [4]). In this connection it will be interesting to compare our work with the approach in [35] to program development in analysis via Gödel's Dialectica Interpretation, since this interpretation is also able to constructively realise the above mentioned axiom.

(iii) *Case studies*

These will mainly be concerned with computable analysis and infinitary combinatorics, without excluding other areas, of course. The following topics seem to be particularly promising:

– The Fundamental Theorem of Algebra. Initial work on a formalisation of a constructive proof of this theorem [34] have been undertaken (in collaboration with a research group at the University of Nijmegen), but also well-known non-constructive proofs should be accessible for program extraction.

– Proofs of theorems in infinitary combinatorics based on non-constructive reasoning, e.g. Dickson's Lemma, have been exploited for program extraction in [14]. The next challenge would be to extend this to stronger theorems that make essential use of non-constructive choice principles, like e.g. Kruskal's Theorem.

– Exact algorithms for numerical integration. There are strong indications that domain- and proof-theoretic methods can be very useful (see e.g. [52], where a domain theoretic algorithm from my thesis is used for developing a functional program for integration).

– Algorithms extracted from proofs in the theory of rings and fields (see [58] for comprehensive exposition). We expect that this fields contains many gems waiting to be discovered. Important work in this direction has been done in [22].

(iv) *Implementation*

The implementation of the theoretical results of the project will build on previous work with the MINLOG system [3]. The implementation language will probably be SCHEME (facilitating compatibility with MINLOG), but also HASKELL with its lazy evaluation mechanism could be advantageous.

It is planned to involve a research student who will have to participate in all stages of the project. This will require an initial training period of about half a year during which the student will be introduced into the research area. At a later stage of the project, when parts of the theory and the proof system are consolidated, case studies will be ideal subjects for third year projects.

2.3 Relevance to beneficiaries

The objective of this project is to provide methods for improving the quality of software being developed in the future. Therefore the long term beneficiaries of this project are sectors of the software industry and their users. It is to be anticipated that the results of this project will have direct impact on research in domain theory, proof theory and computable analysis undertaken e.g. at the Universities of Edinburgh (Longley, Plotkin, Simpson), Gothenburg (Coquand), Hagen (Weihrauch, Brattka, Schröder), London (Imperial College, Edalat), Munich (Schwichtenberg), Nijmegen (Barendregt), Oslo (Normann), Pittsburgh (Carnegie Mellon, Scott), St Andrews (Escardo, moving to Birmingham), Swansea (Blanck, Tucker), and Uppsala (Stoltenberg-Hansen, Hertling).

2.4 Dissemination and Exploitation

The aspired results of this project are to be presented first at international conferences like the *Logic Colloquium* of the ASL, the IEEE Symposium *Logic in Computer Science (LICS)*, and *Typed Lambda Calculi and Applications (TLCA)*. It is also anticipated to publish the results through refereed journals. The implementation of the proof system shall be presented at international workshops, for example *Computability and Complexity in Analysis (CCA)*, and be made available on the Internet.

2.5 Justification of resources

Funds are requested for a research student, for whom the proposed project will provide an ideal opportunity to become an expert in the field of formal methods in computer science. After having learnt the basics of mathematical logic and constructive mathematics, and having gained some familiarity with functional programming, the student will be able to make extremely valuable contributions to the field in a short time.

As pointed out already the project will be closely linked to the MINLOG project at the University of Munich, but also to research groups in the UK (Edinburgh, Imperial, Leeds, Manchester, St. Andrews), Europe, non UK (Gothenburg, Nijmegen, Oslo, Uppsala), and USA (Pittsburgh). The funds requested for travel would enable these vital contacts to be maintained, and support the presentation of the results of this project at the previously mentioned conferences.

The equipment requested are two desktop computers and one laptop computer needed to implement and present the proof system and to provide a good working environment for the research student. The funds requested for technicians shall provide support for the equipment and installation, maintenance of software. Consumables include document production, software, and communication.

References

[1] S. Abramsky, A. Jung. Domain theory. In: S. Abramsky, D.M. Gabbay, T.S.E Maibaum, editors, *Handbook of Logic in Computer Science, Vol. 3 Semantic Structures*, 1-168, Oxford University Press, 1994.

[2] J. Bates, R. Constable, Proofs as Programs. *ACM Transactions on Programming Languages and Systems* **7**(1), 113-136, 1985.

[3] H. Benl, U. Berger, H. Schwichtenberg, M. Seisenberger, W. Zuber. Proof theory at work: Program development in the Minlog system. In: W. Bibel, P.H. Schmitt editors, *Automated Deduction – A Basis for Applications. II: Systems and Implementation Techniques*, 41-71, Kluwer, 1998.

[4] S. Berardi, M. Bezem, T. Coquand. On the computational content of the axiom of choice. *Typed Lambda Calculi and Applications (TLCA'95)*, LNCS **902**, 602-622, 1995.

[5] U. Berger. Total sets and objects in domain theory. *Annals of Pure and Applied Logic* **60**, 91-117, 1993.

[6] U. Berger. Program extraction from normalization proofs. In: M. Bezem and J.F. Groote, editors, *Typed Lambda Calculi and Applications (TLCA'93)*, LNCS **664**, 91-106, 1993.

[7] U. Berger. Programs from classical proofs. In R. Fritsch, M. Behara, and R.G. Lintz, editors, *Symposia Gaussiana. Proceedings of the 2nd Gauss Symposium. Munich, 1993*, 187-200, de Gruyter, 1995.

[8] U. Berger. Continuous Functionals of Dependent and Transfinite Types. In: B. Cooper, J. Truss, editors, *Models and Computability, invited papers from Logic Colloquium '97, Leeds, UK, 1997.* London Mathematical Society Lecture Notes Series **259**, 1–22, 1999.

[9] U. Berger. Computability and Totality in Domains. To appear in *Mathematical Structures in Computer Science.*

[10] U. Berger. Minimisation vs. Recursion on the Partial Continuous Functionals. Submitted for publication. `http://www-compsci.swan.ac.uk/ csulrich/recent-papers.html`

[11] U. Berger, M. Eberl, H. Schwichtenberg. Normalization by evaluation. *Prospects for hardware foundations (NADA)*, LNCS **1546**, 117-137, 1998.

[12] U. Berger, H. Schwichtenberg. Program development by proof transformation. In H. Schwichtenberg, editor, *Proof and Computation, Series F: Computer and Systems Sciences* **139**, 299-340, NATO ASI ISS Marktoberdorf, Springer, 1993.

[13] U. Berger, H. Schwichtenberg. Program Extraction from Classical Proofs. *Logic and Computational Complexity (LCC '94)*, LNCS **960**, 77-97, 1995.

[14] U. Berger, H. Schwichtenberg, M. Seisenberger. The Warshall algorithm and Dickson's lemma: Two examples of realistic program extraction. To appear in *Journal of Automated reasoning.*

[15] J. Bergstra. Computability and continuity in finite types. PhD thesis, Utrecht, 1976.

[16] E. Bishop, D. Bridges. Constructive Analysis. *Grundlehren der Mathematischen Wissenschaften* **279**, Springer, 1985.

[17] J. Blanck. Domain representability of metric spaces. *Annals of Pure and Applied Logic* **83**, 225-247, 1997.

[18] J. Blanck, V. Stoltenberg-Hansen, J.V. Tucker. Streams, Stream Transformers and Domain Representations. *Prospects for hardware foundations (NADA)*, LNCS **1546**, 27-68, 1998.

[19] V. Brattka, K. Weihrauch. Computability on subsets of Euclidean space I: Closed and compact subsets. *Theoretical Computer Science* **219**, 65-93, 1999.

[20] J. Chirimar, D.J. Howe. Implementing Constructive Real Analysis. In: J.P. Myers, Jr., M.J. O'Donnels, editors, *Constructivity in Computer Science*, San Antonio, Texas, 1991, LNCS **613**, 165–178, 1992.

[21] R. Constable et al. Implementing Mathematics with the Nuprl Proof Development System. Prentice–Hall, 1986.

[22] T. Coquand, H. Persson. Gröbner Bases in Type Theory. In: T. Altenkirch, W. Naraschewski, B. Reus, editors, *Types for Proofs and Programs (Types'98, Kloster Irsee, Germany)*, selected papers, LNCS **1675**, 33-46, 1999.

[23] A. Edalat, P Sünderhauf. Computable Banach spaces via domain theory. *Theoretical Computer Science* **291**, 169-184, 1999.

[24] Y. L. Ershov. Hereditarily effective operations. *Algebra i Logika* **15**(6), 642-654, 1976.

[25] Y. L. Ershov. Model C of partial continuous functionals. In: R. Gandy, M. Hyland, editors, *Logic Colloquium 1976*, North Holland, Amsterdam, 455-467, 1977.

[26] M. Escardó. Real PCF extended with $\exists$ is universal. *Proceedings of the Third Imperial College Workshop*, 13-24, 1996.

[27] H. Friedman. Classically and intuitionistically provably recursive functions. In: D.S. Scott and G. H. Müller, editors, *Higher Set Theory*, LNM **669**, 21–28, 1978.

[28] E. Griffor, I. Lindström, V. Stoltenberg–Hansen. Mathematical theory of domains. Cambridge University Press, 1993.

[29] J. Harrison. Theorem Proving with the Real Numbers. Springer, 1999.

[30] S. Hayashi, H. Nakano. PX: A Computational Logic. The MIT Press, 1988.

[31] C. Jones. Completing the Rationals and Metric Spaces in LEGO. In: G. Huet, G. Plotkin, editors, *Logical Environments*, Cambridge University Press, 1993.

[32] S. C. Kleene. On the interpretation of intuitionistic number theory. *Journal of Symbolic Logic* **10**, 109-124, 1945.

[33] S. C. Kleene. Countable functionals. In: A. Heyting, editor, *Constructivity in Mathematics*, 81-100, North–Holland, 1959.

[34] M. Kneser. Ergänzungen zu einer Arbeit von Hellmuth Kneser über den Fundamentalsatz der Algebra. *Mathematische Zeitschrift* **177**, 285-287, 1981.

[35] U. Kohlenbach. Effective bounds from ineffective proofs in analysis: An application of functional interpretation and majorization. *Journal of Symbolic Logic* **57**(4), 1238–1273, 1992.

[36] G. Kreisel. Interpretation of analysis by means of constructive functionals of finite types. In: A. Heyting, editor, *Constructivity in Mathematics*, 101-128, North–Holland, Amsterdam, 1959.

[37] C. Kreitz, K. Weihrauch. Theory of representations. *Theoretical Computer Science* **38**, 35–53, 1985.

[38] J. Longley, A. Simpson. A Uniform Approach to Domain Theory in Realizability Models. *Mathematical Structures in Computer Science* **11**, 1996.

[39] Z. Luo, R. Pollack, P. Taylor. How to Use Lego. Manual, University of Edinburgh, Department of Computer Science, 1989.

[40] P. Martin–Löf. Intuitionistic Type Theory. Bibliopolis, Naples, 1984.

[41] Y. N. Moschovakis. Recursive metric spaces. *Fundamenta Mathematicae* **55**, 215-238, 1964.

[42] B. Nordström, K. Peterson, J. Smith. Programming in Martin-Löf's Type Theory. Oxford University Press, 1990.

[43] D. Normann. Recursion on the countable functionals. LNM **811**, 1980.

[44] D. Normann. The continuous functionals of finite types over the reals. Preprint Series, Inst. Math. Univ. Oslo **19**, 1998.

[45] D. Normann. The continuous functionals. In: S. Abramsky, D.M. Gabbay, T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, Vol. 4 Semantic Modelling p. 251-257. Oxford University Press, 1995.

[46] D. Normann. Computability over the partial continuous functionals. To appear in *Journal of Symbolic Logic.*

[47] C. Paulin–Mohring. Inductive definitions in the system Coq; rules and properties. In: M. Bezem and J.F. Groote, editors, *Typed Lambda Calculi and Applications (TLCA'93)*, LNCS **664**, 328–345, 1993.

[48] G. Plotkin. LCF considered as a programming language. *Theoretical Computer Science* **5**, 223-255, 1977.

[49] M.B. Pour-El, I. Richards. Computability in Analysis and Physics. *Perspectives in Mathematical Logic*, Springer, 1989.

[50] H. Schwichtenberg, A. S. Troelstra Basic Proof Theory. Cambridge University Press, 1996.

[51] D. Scott. Data types as lattices. *SIAM Journal of Computation* **5**, 522-587, 1976.

[52] A. Simpson. Lazy Functional Algorithms for Exact Real Functionals. In *Mathematical Foundations of Computer Science*, LNCS **1450**, 456-464, 1998.

[53] M.B. Smyth. Powerdomains. *Journal of Computer and Systems Science* **16**, 23–36, 1978.

[54] D. Spreen. On effective topological spaces. *Journal of Symbolic Logic* **63**, 185–221, 1998.

[55] V. Stoltenberg-Hansen, J. V. Tucker. Complete local rings as domains. *Journal of Symbolic Logic* **53**, 603–624, 1988.

[56] V. Stoltenberg-Hansen, J. V. Tucker. Effective Algebras. In: S. Abramsky, D.M. Gabbay, T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science, Vol. 4 Semantic Modelling*, 357-526, Oxford University Press, 1995.

[57] V. Stoltenberg-Hansen, J.V. Tucker. Concrete models of computation for topological algebras. *Theoretical Computer Science* **219**, 347–378, 1999.

[58] V. Stoltenberg-Hansen, J.V. Tucker. Computable rings and fields. In: E. Griffor, editor, *Handbook of Recursion Theory*, North-Holland, 1999.

[59] W. Sieg, S.S. Wainer. Program Transformation and Proof Transformation. In: E. Börger, Y. Gurevich, K. Meinke, editors, *Computer Science Logic (CSL'93)*, Swansea, UK, LNCS **832**, 305–317, 1993.

[60] A.S. Troelstra. *Metamathematical Investigations of Intuitionistic Arithmetic and Analysis.* LNM **344**, 1973.

[61] J.V. Tucker, J.I. Zucker. Examples of semicomputable sets of real and complex numbers. In: J.P. Myers, Jr., M.J. O'Donnels, editors, *Constructivity in Computer Science*, San Antonio, Texas, 1991, LNCS **613**, 179-198, 1992.

[62] J.V. Tucker, J.I. Zucker. Computations by 'While' programs on topological partial algebras. *Theoretical Computer Science* **219**, 379–426, 1999.