

Domain Theoretic Methods for Program Synthesis
Individual Grant Review Report
EPSRC Grant GR/R16020/01

Ulrich Berger

July 2004

1 Background

This project was concerned with domain-theoretic methods for program synthesis from formal proofs. Since domain theory and program synthesis appear to have little in common, I think it is appropriate to begin with a short explanation of how the project connected these two areas. Domain theory, invented by Dana Scott in the 1960s, is widely used for modeling advanced computational phenomena, such as partiality, recursion, higher types, recursive types and polymorphism, as they occur in many modern high-level programming languages. By providing a clear, syntax independent, denotational semantics, domain theory has proven an invaluable tool for understanding these fundamental programming principles and reasoning about them in an informal and flexible, but mathematically concise way. The usefulness of domain theory is particularly striking when – as in this project – methods of program synthesis from proofs are applied to formal systems with strong logical and mathematical principles. In the project program synthesis was applied, for example, to systems of inductive definitions and to formalizations of analysis based on a higher type language with choice principles. This required the introduction of complicated higher order data types and new powerful computational principles, which were analyzed according to their termination and correctness using domain-theoretic methods. The key point was that for such an analysis often extensionality, or even continuity, are needed. These properties are usually very hard to establish on a syntactical level, but are available ‘for free’ in a domain-theoretic setting.

My main collaborator in this project was Monika Seisenberger who worked as a Research Assistant and completed her PhD in Mathematics (supervisor Hel-

mut Schwichtenberg, Munich) within the project. I also worked with Paulo Oliva during his three months research visit in Swansea in winter 2001/2 which was part of his PhD project (supervisor Ulrich Kohlenbach, then Aarhus, now Darmstadt). Important contributions are also due to Helmut Schwichtenberg, the leader of the Minlog project, whom Seisenberger and I visited several times in Munich to work on the project.

The overall aim of this project was to study the computational content of mathematical proofs towards a practically useful method of program synthesis from proofs. The general idea is as follows: Given a formal proof of a specification of the form

$$\forall x \exists y A(x, y)$$

one extracts a program p from the proof satisfying the specification, i.e.

$$\forall x A(x, p(x)).$$

Roughly speaking, the program is obtained by interpreting the logical rules and axioms of the proof as program constructs (logical control, iteration, procedure call e.t.c.) and deleting computationally irrelevant parts. This method, whose underlying idea is known as the ‘Curry-Howard correspondence’, or the ‘formulas-as-types/proof-as-programs paradigm’, and which is rendered technically as ‘realizability’, works for most systems based on intuitionistic logic, that is, logic that does *not* use the classical law of excluded middle (LEM).

There exist a number of implementations of intuitionistic formal systems (e.g. NuPrL, Coq, Agda, PX, Minlog), some of which are equipped with modules for program extraction. However, case studies in program extraction from proofs with these systems have either led to monstrous results [12], or

were restricted to small examples of demonstrational nature [5, 7], or were carried out on a semi formal level [4, 6]. The rather limited success of program extraction so far appears to have the following main reasons:

- Most conventional mathematics is based on classical logic, i.e., does use LEM, which is not directly accessible for program extraction. Therefore, classical proofs need to be translated into intuitionistic proofs. This can be done by various methods which, however, often yield practically unusable results (explosion in size) and apply to a restricted class of systems only.
- Existing methods of program extraction require fully formalized proofs as input, yielding output programs that tend to be very large and complex (in terms of type- and control-structure) and therefore are hardly applicable in practice.

In our project we were able to make significant contributions towards these issues. Besides optimizations of existing techniques, our results comprise new constructive axiomatizations of analysis with new computational interpretations, the correctness of which was established by domain-theoretic proof techniques.

2 Results

I now describe in some detail the main results of the project. Papers that are a direct outcome of the project are labeled ‘P’ if they are published or accepted for publication, and labeled ‘D’ if they are drafts. Copies are available at our project web page:

www-compsci.swan.ac.uk/~csulrich/epsrc-project.html

2.1 Intuitionistic versus classical mathematics

We studied various methods of transforming classical proofs into intuitionistic proofs and applied them to problems in arithmetic, analysis and infinitary combinatorics.

Classical arithmetic in finite types

An important outcome was a refinement of Gödel’s negative- and Friedman’s *A*-translation for classical arithmetic in finite types. This was published in

[P1] U. Berger, W. Buchholz, and H. Schwichtenberg. Refined program extraction from classical proofs. *Annals of Pure and Applied Logic*, 114:3–25, 2002.

The crucial advances made in this paper are twofold: (1) the negative-translation needs to be applied only partially, (2) the translation applies to an enrichment of arithmetic by free predicate symbols (without decidability assumptions) and axioms of a certain syntactical form (including Horn-formulas). (1) leads to simplified programs (see 2.2), (2) increases the applicability of the system since many notions and facts can be introduced axiomatically which means that proofs need only be partially formalized. This refined translation has been extended by Seisenberger to theories of constructive inductive definitions in her PhD thesis

[P2] M. Seisenberger. *On the Constructive Content of Proofs*. PhD thesis, University of Munich, 2003.

A further refinement of the proof translation described above is developed in

[P3] U. Berger. Uniform Heyting Arithmetic. *Annals of Pure and Applied Logic*, to appear.

In this paper a variant of the *A*-translation for arithmetic in the style of a Kripke semantics is developed, following an idea of Coquand and Hofmann [9]. The Kripke semantics avoids an increase of logical complexity of Π_2^0 -induction during the translation, and in this way unites the refined translation with Parsons’ result that classical Π_2^0 -induction is Π_2^0 -conservative of over its intuitionistic counterpart. The paper [P3] further improves [9] by admitting free predicate variables for which no decision procedure is required, a relaxation which, as case studies in [P3] show, is crucial for applications.

Infinitary combinatorics

Classical reasoning in combination with strong mathematical principles is particularly important in areas such as analysis, infinitary combinatorics and algebra. Examples are Higman’s lemma and Kruskal’s theorem, results in infinitary combinatorics with important applications to term rewriting theory. We studied a classical proof of Higman’s lemma due to Nash-Williams which uses classical logic and the axiom of dependent choice. In her thesis [P2] Seisenberger gave a new constructive proof of Higman’s

lemma in a theory of constructive inductive definitions. The crucial idea, due to Fridlender and Coquand [8], is to redefine the notion of a well-quasiorder, which speaks about *infinite* sequences, by a constructive inductive definition of a predicate of *finite* sequences. The novelty of Seisenberger’s work is that her proof deals with full Higman’s lemma (not restricted to finite alphabets) and closely follows the combinatorial idea of Nash-Williams’ proof. Seisenberger’s thesis also contains a rather comprehensive analysis and comparison of different constructive proofs of Higman’s Lemma. Seisenberger also gave a constructive proof of Kruskal’s theorem in the theory of inductive definitions without using ordinal notations. Currently we are trying to extend this work to a constructivization of Hilbert’s basis theorem.

Classical analysis

A more general proof-theoretic approach, which applies to full classical analysis, has been taken in

- [P4] U. Berger. A computational interpretation of open induction. In F. Titsworth, editor, *Proceedings of the Ninetenth Annual IEEE Symposium on Logic in Computer Science*, pages 326–334. IEEE Computer Society, 2004.

In this paper the axiom of dependent choice is replaced by the classically equivalent axiom of *open induction*, and it is shown that classical arithmetic plus open induction (which, by the above equivalence, is equivalent to full classical analysis) admits Gödel’s negative- and Friedman’s *A*-translation, i.e., a constructivization of proofs of arbitrary Π_2^0 -sentences. For the reader’s convenience and since open induction is less-known, I briefly discuss this axiom now. Let α, β range over infinite sequence of objects of some arbitrary type ρ , and let $\beta <_{\text{lex}} \alpha$ mean that, w.r.t. some wellordering on ρ , β is lexicographically smaller than α . For example, if ρ is the type of natural numbers with the usual ordering, then $10000\dots >_{\text{lex}} 01000\dots >_{\text{lex}} 00100$ and so on, so $<_{\text{lex}}$ is not wellfounded! Nevertheless the induction scheme

$$\forall \alpha (\forall \beta <_{\text{lex}} \alpha U(\beta) \rightarrow U(\alpha)) \rightarrow \forall \alpha U(\alpha)$$

holds for predicates $U(\alpha)$ which are open, i.e., depend on a finite initial segment of α only. Classically, one can prove the correctness of open induction

by Nash-Williams’ so-called ‘minimal-bad-sequence-argument’. But, as I showed in [P4], open induction can also be justified constructively by means of relativized bar induction.

2.2 Computational interpretation of formal systems

Technically, the computational interpretation of formal systems based on intuitionistic logic was carried out by variants of Kreisel’s modified realizability interpretation (an alternative would be Gödel’s Dialectica Interpretation, for which Kohlenbach [11] found interesting practical applications; however, realizability better preserves the structure of proofs and seems to be more accessible for optimizations). We concentrated on optimizations of a realizability interpretation of Heyting Arithmetic in finite types, HA^ω , extended by inductive definitions, open induction, and other ‘unnatural’ axioms arising from the negative/*A*-translation of choice principles. We also studied a realizability interpretation of restricted arithmetic with non-size-increasing polynomial time realizers.

Arithmetic and inductive definitions

In her thesis Seisenberger optimized the realizability interpretation of HA^ω by ‘tagging’ certain occurrences of logical rules as ‘computationally irrelevant’ and ignoring them in the program extraction process, following an idea in [4]. She extended this technique to inductive definitions and achieved in this way a considerable simplification of the types of realizers. Her thesis contains detailed proofs of the soundness of the refined system. A similar refinement is applied in the paper [P3], which also contains a number of small case studies highlighting the effect of these proof-theoretic optimizations of realizability.

Arithmetic for non-size-increasing polytime

In the paper

- [P5] K. Aehlig, U. Berger, M. Hofmann, and H. Schwichtenberg. An arithmetic for non-size-increasing polynomial-time computation. *Theoretical Computer Science*, 318:3–27, 2004.

we developed an arithmetic in higher types which has as realizers polynomial time non-size-increasing functionals. The system bares some similarity with systems by Bellantoni, Niggl and Schwichtenberg [1],

however, the crucial difference in [P5] is that, due to a restriction of resources (which has the effect that realizers are non-size-increasing) the induction scheme can be liberalized allowing for the representation of, e.g., sorting algorithms that cannot be dealt with in [1].

Choice principles and open induction

In the paper

[P6] U. Berger and P. Oliva. Modified bar recursion and classical dependent choice. In *Logic Colloquium 2001*. Springer, to appear.

we introduced a variant of Spector’s bar recursion and showed that it can be used for a realizability interpretation of the negative/ A -translated axioms of countable and dependent choice. In [P4] I used a new recursion scheme, which I called *open recursion*, for a realizability interpretation of open induction and showed that one can derive from this the interpretation of countable choice of Berardi, Bezem and Coquand [3].

2.3 Domain-theoretic analysis of computation in higher types

Throughout our research domain-theoretic semantics served as a guide in the design of formal systems, but was also used to prove core meta-mathematical results.

Correctness via totality

The realizers introduced in [P4] and [P6] are recursively defined functionals of rather complicated higher types. I proved the correctness and termination of these realizers by domain-theoretic methods. In order to get an idea how domain theory comes into play here, let us have a quick look at open recursion (the realizer of open induction) which is a functional Φ characterized by the recursion equation

$$\Phi\alpha =_{\text{nat}} f\alpha(\lambda n, y, \gamma. \text{if } y < \alpha n \text{ then } \Phi(\bar{\alpha}n * y @ \gamma) \text{ else } 0^\rho).$$

In this equation f is a parameter and $\beta := \bar{\alpha}n * y @ \gamma$ is the infinite sequence that coincides with α at arguments less than n , equals y at n , and coincides with γ at arguments larger than n , so β is an ‘arbitrary’ sequence $<_{\text{lex}} \alpha$. Essentially three facts about Φ need to be established: (1) Φ ‘exists’, that is, a constant satisfying this equation can be consistently added to the system of arithmetic, (2) Φ realizes

open induction, (3) Φ can be ‘computed’, that is, adding the equation above as a rewrite rule leads to a weakly normalizing system. For example, problems (1) and (3) can be reduced to proving that Φ is *total* in the domain-theoretic model PCF of partial continuous functionals, using Ershov’s characterization of the Kleene-Kreisel functionals [10] and Plotkin’s adequacy result [13]. The totality of Φ amounts to proving the formula

$$U(\alpha) := \alpha \text{ total} \Rightarrow \Phi(\alpha) \text{ total}$$

which can be proven by open induction because, by the continuity of f , the predicate $U(\alpha)$ is open.

Relative computability

The domain-theoretic analysis of higher type recursive functionals is continued in

[D1] U. Berger and P. Oliva. Modified bar recursion.

We compared different versions of bar recursion interpreted in the domain-theoretic models of partial and total continuous functionals according to their relative computational strength. The results are:

- The definition of the fan functional within PCF can be derived from Kohlenbach’s and our variant of bar recursion.
- Modified bar recursion of the lowest type is primitive recursively equivalent to the functional Γ .
- The type structure \mathcal{M} of strongly majorizable functionals is a model of modified bar recursion.
- Spector’s bar recursion is primitive recursively definable in modified bar recursion.
- Modified bar recursion is not S1-S9 computable over the total continuous functionals.

Strong normalization

The application of domain-theory to termination problems has been even more strengthened in the draft

[D2] U. Berger. Strong normalization proofs based on continuous semantics.

where I was able to show that even *strong* normalization follows from totality with respect to a strict domain-theoretic semantics. This result entails strong normalization for all variants of bar recursion.

Complexity in higher types

In the final phase of the project I began to work on a domain-theoretic analysis of computational complexity in higher types based on generalized logical relations:

[D3] U. Berger. Logical relations and feasibility in higher types.

Among others it contains an unexpected result on relative Banach-Mazur computability in higher types within the domain-theoretic model of partial continuous functionals: Banach-Mazur computability relative to any class of type one functions containing all polytime functions implies domain-theoretic computability.

2.4 Implementation

As part of her thesis and during her work as a RA in the project Seisenberger implemented large parts of the results described above in the Minlog system [2]. For example, she implemented the constructive proofs of Higman's lemma based on inductive definitions. She also implemented a refined A -translation and the constructive interpretation of the negative/ A -translated axiom of dependent choice via modified bar recursion as developed in [P6]. Seisenberger applied this to a complete implementation of Nash-Williams' classical proof of Higman's Lemma and extracted a program from the proof.

A summary of most of our results is contained in

[P7] U. Berger and M. Seisenberger. Applications of inductive definitions and choice principles to program synthesis. In L. Crosilla, P Schuster, editors, *Proceedings of the Workshop: From Sets and Types to Topology and Analysis – Towards Practicable Foundations for Constructive Mathematics*, Venice International University, 12-16 May 2003, Oxford University Press, to appear.

3 Project Review

In the proposal for this project I formulated the following objectives:

1. Investigation of domain-theoretic models of computation, in particular for higher types and real numbers.

2. Development of a method of program synthesis by means of an enhanced realizability interpretation.
3. Case studies, mainly in the areas of computable analysis and infinitary combinatorics.
4. Implementation of an interactive proof system equipped with a module for program synthesis.

Nearly all of these objectives have been met. Concerning the investigation of systems of analysis we have concentrated more on axiomatizations than on the study of concrete real number representations. Several results, such as, for example, the computational interpretation of open induction and the domain-theoretic strong normalization proof were unexpected and clearly go beyond the original targets.

3.1 Dissemination of results

The published output of this project consists in three journal papers [P1,P3,P5], three proceedings of major conferences (LICS and Logic Colloquium of the ASL) and one workshop [P4,P6,P7] and Seisenberger's PhD thesis [P2]. Three of these papers [P3,P6,P7] are still in press. Furthermore, there are three draft papers [D1,D2,D3] of which [D1] is ready to be submitted.

We presented our results at eighteen conferences and workshops, among them Logic Colloquium of the ASL 2001, LICS 2004, two Dagstuhl meetings (2001 and 2002) and one Oberwolfach meeting (2002). We also gave several seminar talks at Universities in the UK and elsewhere. More details are on our project page.

3.2 Research impact

I expect the results of this project to have profound impact on the development of computer aided program synthesis, constructive mathematics, and the theory of higher type algorithms. In particular the computational analysis of inductive definitions and classical choice principles including their optimized implementation can be considered as major advances in these areas. The new axiomatization of classical analysis by open induction will facilitate considerably the formalization and computational analysis of results with a 'minimal-bad-sequence-style' proof (for example, Kruskal's theorem and Hilbert's basis theorem). I also believe that my strong normalization results as well as my logical/domain-theoretic approach

to higher type complexity will considerably influence research in this area.

3.3 Explanation of Expenditure

In relation to the scientific output of this project the expenditure has been very modest. The budget for traveling has not been fully spent, mainly, because I was invited for many visits and conferences and expenses were covered by the host organization. Some savings could also be made by cost-effective use of transportation.

The only major deviation from the original plan was the fact that, instead of having a PhD student in the project, I decided to employ Seisenberger as a Research Assistant. The reason for this was that Seisenberger's PhD contributed fundamentally to my project and her experience with the Minlog system allowed us to tackle the core problems of the project straightaway. I think the extraordinary success of the project, which is to a great extent, due to Seisenberger's contributions, fully justifies this decision.

4 Further Work

The final phase of this project was dominated by work on two research topics which had not been anticipated in the proposal of this project:

- (1) domain-theoretic strong normalization proofs,
- (2) a logical/domain-theoretic approach to higher type complexity.

For both topics I am in the process of preparing research proposals. I will present my results on topic (1) at a Dagstuhl meeting (22-27 August 2004) and will continue work on this topic during a visit to Japan in April 2005 (invited by Y. Akama, Tohoku University). Further plans on topic (2) will be developed during a visit of J. Royer (Syracuse University) to Swansea in October. Royer is an expert on complexity in higher types,

Continuing my work on program extraction from proofs, I will work with S. Berardi (University of Turin) on a machine learning interpretation of classical logic during a visit of Berardi to Swansea this September.

References

- [1] S. Bellantoni, K.-H. Niggl, and H. Schwichtenberg. Higher type recursion, ramification and polynomial time. *Annals of Pure and Applied Logic*, 104:17–30, 2000.
- [2] H. Benl, U. Berger, H. Schwichtenberg, M. Seisenberger, and W. Zuber. Proof theory at work: Program development in the Minlog system. In W. Bibel and P. Schmitt, editors, *Automated Deduction – A Basis for Applications*, volume II: Systems and Implementation Techniques of *Applied Logic Series*, pages 41–71. Kluwer Academic Publishers, Dordrecht, 1998.
- [3] S. Berardi, M. Bezem, and T. Coquand. On the computational content of the axiom of choice. *Journal of Symbolic Logic*, 63(2):600–622, 1998.
- [4] U. Berger. Program extraction from normalization proofs. In M. Bezem and J. Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *LNCS*, pages 91–106. Springer Verlag, Berlin, Heidelberg, New York, 1993.
- [5] U. Berger. Programs from classical proofs. In M. Behara, R. Fritsch, and R. Lintz, editors, *Symposia Gaussiana. Proceedings of the 2nd Gauss Symposium. Conference A: Mathematics and Theoretical Physics. Munich, Germany, August 2-7, 1993*, pages 187–200, Berlin, New York, 1995. Walter de Gruyter.
- [6] U. Berger. Program extraction from gentzen's proof of transfinite induction up to ϵ_0 . In R. Kahle, P. Schroeder-Heister, and R. Stärk, editors, *Proof Theory in Computer Science*, volume 2138 of *LNCS*, pages 68–77. Springer-Verlag, 2001.
- [7] U. Berger and H. Schwichtenberg. Program extraction from classical proofs. In D. Leivant, editor, *Logic and Computational Complexity, International Workshop LCC'94, Indianapolis, 1994*, volume 960 of *LNCS*, pages 77–97. Springer-Verlag, 1995.
- [8] T. Coquand and D. Fridlender. A proof of Higman's lemma by structural induction, 1994.
- [9] T. Coquand and M. Hofmann. A new method for establishing conservativity of classical systems over their intuitionistic version. *Math. Struct. in Comp. Science*, 9:323–333, 1999.
- [10] Y. Ershov. Model C of partial continuous functionals. In R. Gandy and M. Hyland, editors, *Logic Colloquium 1976*, pages 455–467. North Holland, Amsterdam, 1977.
- [11] U. Kohlenbach. Effective bounds from ineffective proofs in analysis: an application of functional interpretation and majorization. *Journal of Symbolic Logic*, 57:1239–1273, 1992.
- [12] C. Murthy. *Extracting Constructive Content from Classical Proofs*. PhD thesis, Cornell University, 1990.
- [13] G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5:223–255, 1977.