IGR Report on EPSRC Grant GR/L89532: Notions of computability for general datatypes

Mike Fourman Gordon Plotkin John Longley

March 6, 2001

Background/Context

In broad terms, we see this research project as residing at the intersection of two established international research disciplines: theoretical computer science and mathematical logic.

From the point of view of theoretical computer science, our research forms part of a general international effort to develop mathematical and conceptual tools suitable for understanding the semantics and logic of a variety of programming languages. The long-term goals of this research effort are, typically, to support mathematical reasoning about the correctness of real-world software; to enable a clean and principled approach to the design of future programming languages; and to suggest innovations in compiler technology. The programming language semantics community is represented within Britain by groups e.g. at Birmingham, Cambridge, Oxford, Queen Mary College; and elsewhere by researchers in Paris, Marseille, Munich, Darmstadt, Genoa, Pittsburgh, Kyoto etc.

It is fair to admit that there remains an enormous gap between what these theoretical approaches are at present able to deliver and what is demanded by the real-world practice of software engineering. There is therefore room for a wide range of approaches from the theoretical side, which in practice tend to be mutually enriching: for instance, there are tensions between the demands of closeness to real-world programming languages and those of mathematical simplicity and clarity (the latter being a practical as well as an aesthetic concern in that it allows reasoning to proceed smoothly at higher levels of abstraction without fussing over idiosyncratic details of the programming language). We would place our work near the latter end of this spectrum, our intention being to uncover deep mathematical structure that is inherent in the nature of certain kinds of computation, independent of details of language design—and whose relevance will therefore persist as particular languages come and go.

The distinctive contribution of our project has been to focus on various natural *notions of computability* that programming languages may embody. It is well known that all reasonable programming languages give the same computable functions on natural numbers, but there are interesting senses in which this is not true for all other commonly occurring datatypes. Our philosophy is that an understanding of these notions of computability gives insight into those semantic models that most closely match the languages in question, which in turn supports the design of program logics with a clear operational meaning. Particular collaborators in the field whose approach is close to ours, and with whom we have enjoyed fruitful interaction, include Abramsky and his group at Edinburgh (now at Oxford); Ong and Nickau at Oxford; Streicher and his students in Darmstadt; and Scott and his collaborators at Carnegie Mellon, Pittsburgh.

We also see our project as making a contribution to mathematical logic, in particular to the understanding of computability at higher types. This is not so much a coherent and focussed area of active research within the mathematical logic community at present, as a widely diffused area of general interest to which researchers from many fields have contributed at one time or another. Our contribution here has mainly been to bring

together much of this widely scattered knowledge, and to begin to unify it in a coherent conceptual framework. We hope that our work will make this knowledge more readily accessible to computer scientists, and will help to foster a deeper level of interaction between the mathematics and computer science communities.

Our efforts have attracted the interest of the mathematical logic community, as witnessed by the invitation to Longley to give a series of tutorial lectures at the Logic Colloquium 2000 in Paris. Our personal contacts within the logic community have included Hyland in Cambridge, Wainer in Leeds, Normann in Oslo, and Feferman at Stanford.

Key Advances and Supporting Methodology

1. General framework for notions of computability

At the time of writing the proposal for this project, we had in mind a simple framework for *extensional* notions of computability (that is, notions of computable *function* for general datatypes). A brief presentation of this framework, including a proof that there is no "Church's thesis" for partial computable functionals at higher type, was given at the end of [Lon98a].

Shortly thereafter, in May 1999, we discovered a much more general framework which embraces both extensional and non-extensional notions of computability. This framework is based around a typed generalization of standard realizability and of material in Longley's Ph.D. thesis [Lon95]. This new framework remains in the spirit of our inquiry into notions of *computability* (what can be computed) as distinguished from particular models of *computation* (how it is computed); but it massively broadens the scope of our investigation from notions of computable function to much more general notions of "computable operation". In terms of theoretical computer science, this allows us to take account of the full expressive power of real programming languages, where non-functional features are frequently considered indispensable. In terms of mathematical logic, it greatly extends the scope of what we are able to bring together within a coherent framework. Though not technically difficult or complex, we regard this new framework as the most important breakthrough to have been made during the project.

Our paper at the Trento Realizability Workshop [Lon99a] represented a halfway stage towards the discovery of this new perspective: here the emphasis was on achieving a semantic understanding of fine-grained differences in computational power between languages whose class of computable *functions* was the same. The main conceptual breakthrough was presented more explicitly in an unpublished note [Lon99b] which has been circulated amongst our colleagues via the Internet. We have since then been occupied with reaping the benefits of this insight—a task which we expect to engage us for some time yet.

In the proposal we promised a survey paper bringing together current knowledge on extensional notions of computability. In the light of the new breakthrough, we decided to extend the scope of this survey to non-extensional notions. As a result, the form of the survey has now expanded to a series of three substantial papers. An overview of the material for the whole series was presented in three lectures at the Logic Colloquium 2000. Part I, a historical survey of the main ideas of higher-type computability [Lon01a], has now been submitted for publication in the Logic Colloquium proceedings. Parts II and III are each about half-completed, and working drafts of them are available online [Lon01b, Lon01c]. We have some intention of turning this series of papers into a monograph at a later date.

2. Algebraic operations and effects

A different approach to non-functional computation is via Moggi's notion of the computational lambda calculus, with its monadic semantics; here "effects" provide non-functional features. A previous paper [PP01a] by Plotkin and Power had shown that one could relate

operational and denotational semantics, for so-called algebraic effects, such as (probabilistic) nondeterminism and printing (but not states, or exceptions).

The algebraic operations involved (e.g. probabilistic choice) satisfied a naturality condition. In further work this was shown to be insufficiently strong, and a parametrised version was needed (incorporated in the final version of [PP01a]). The paper [PP01b] by Plotkin and Power explores this parametrised version, giving a number of equivalent characterisations, including a "generic element" one of a kind familiar from Lawvere's work on algebraic theories.

3. Sequentially realizable functionals

Part of the original proposal concerned the detailed study of one particular notion of computability: that embodied by the *sequentially realizable (SR)* functionals (there called the intensionally-sequential functionals). Here we have stuck very closely to what we proposed. A detailed theoretical study of the SR functionals appeared in our long paper [Lon98a]. This consolidates what we knew at the time of the proposal and includes some subsequently discovered material, such as a simple presheaf characterization of the SR functionals. We feel that the theoretical understanding of this type structure is now essentially complete, in the sense that any further questions about it could now be answered more or less routinely.

On the practical side, we have considered some possible programming applications of SR functionals, e.g. to general search algorithms and exact real number computation. Early on in the project we produced a Standard ML source file (available on the web [Lon98b]) containing some programming examples to explain the idea of SR-based programming. Though short and simple, this seems to have been an effective way of disseminating our ideas and has enjoyed a significant circulation in the functional programming community. We later consolidated the ideas into a paper presented at ICFP 1999 [Lon99c].

However, in the course of this work (and in our subsequent experiments in real number computation, see 5 below), we did not discover much in this area that we did not already know at the time of the proposal, and we now feel that these potential programming applications of SR functionals, though intriguing, are of less practical significance than we had originally hoped. The main practical importance of the theory of SR functionals, we now believe, lies in their implications for the design of program logics, to which we turn next.

4. Logics for programming languages

In September 1999 we realized that our state of theoretical knowledge was now sufficient to realize a long-standing ambition of Fourman's: to give a proof system in a "naive" spirit for most of Standard ML, which is complete relative to first-order arithmetic. This also realizes the intentions of the Extended ML project of Sannella *et al* in the 1980s and 90s.

Our approach to Standard ML is to provide logics for three sublanguages of increasing size and difficulty: a functional fragment,[1] a fragment including most everyday uses of exceptions, and a fragment which also includes some simple "first order" uses of references and even continuations. This approach builds on our theoretical work inasmuch as these sublanguages correspond to three mathematically natural notions of computability of which we have a good semantic understanding, and we allow these notions to guide the precise formulation of what is in each sublanguage.[2] Moreover, the understanding of the

[1] In the sense of the SR functionals (see 3). Here these turn out to have several practical advantages over using the usual and more restricted notion of functional program, e.g. the logic is simpler to axiomatize and has better decidability properties; and it allows us to apply a simple style of reasoning to programs with functional behaviour that make internal use of non-functional features (such as memoization operations).

[2] The relevant theory for the third sublanguage has been worked out informally but not consolidated yet. A programming language characterization can be given using the notion of *restartable exception* due to Ian Stark; a more semantic characterization can be given via a refinement of the van Oosten-Longley

relationships between these three notions, as arising from our general framework, tells us how the logics for these three fragments are to be tied together.

Longley and Aspinall spent three man-months prototyping our ideas in the theorem prover Isabelle. This experience confirmed the feasibility of our approach, and formed the basis of the proposal for the new EPSRC-funded project GR/N64571: "A proof system for correct program development" [FFL00], which commenced in January 2001. The fact that enough theory is now in place to attempt this is probably the biggest practical benefit to have flowed from the previous project so far.

5. Exact real number computation

During the first half of the project we investigated questions of computability for operations related to real numbers, with respect to the notions of "computability" embodied by various programming languages. Here we enjoyed very fruitful interaction with Martín Escardó (in Edinburgh until the end of 1999). Some of our results are reported in [Lon99a].

Later we spent some time conducting programming experiments in Standard ML to try to provide efficient realizations of some of these operations (e.g. exact integration). We discovered that the use of Ian Stark's ingenious "restartable exception" primitives (implemented using continuations) offered major gains in both expressivity and efficiency. Longley and Aspinall invested about two man-months of effort in the implementation of a system that would demonstrate these ideas. Unfortunately, however, this did not take us beyond the substantial amount of groundwork required to build an efficient exact calculator, to the point where our system does anything really interesting. Nevertheless, we are still very hopeful that ideas will lead to some kind of genuinely feasible exact integration, and we hope to devote some more time to this in the future. One particular challenge which seems to us to be within striking distance is to improve on the current best known upper bound on the area of the Mandelbrot set (surprisingly, the area is known only to within about 3%).

6. Computability for abstract types

Data abstraction is a key ingredient in modern programming methodologies, and a significant part of the original proposal concerned questions of computability for *abstract* datatypes. We have devoted less time to this than we originally planned, having decided instead to pursue the unexpected advances in non-extensional notions of computability (see 1).

We have, however, obtained one theorem which will significantly help us in axiomatizing program logics for abstract types (as in 4). a logical characterization of those abstract datatypes which are observationally equivalent to *retracts* of simple types (we provisionally call such abstract datatypes *translucent*). As noted in the proposal, notions of computability for simple types extend trivially to retracts of simple types, but what had escaped our notice was that a very large class of abstract datatypes—including all the usual implementations of stacks, queues, sets, multisets and lookup tables, and the type of theorems in an LCF-style theorem prover—could be *systematically* interpreted in this way. From the point of view of program logics, this means we can reason both about elements of abstract types up to observational equivalence, and about implementations of abstract type signatures up to observational indistinguishability. This material has not yet been written up—this will be an early priority for our future work.

We have also discovered a few counterexamples which suggest that outside the realm of translucent datatypes, questions of computability become very hard indeed. However, we are not aware of any abstract types used in functional programming practice which are not translucent.

---

combinatory algebra.

Project Plan Review

As already mentioned, the discovery of a more general framework for notions of computability than the one we originally had in mind (see 1 above) greatly expanded the scope of the first half of the proposed project; and almost all the departures from our original plan stem from this development. On the one hand, we decided to broaden the scope of the promised "extended survey paper" to embrace this extra generality, and this has resulted in the more ambitious series of three papers in preparation. On the other hand, having opted to spend more time studying non-extensional computability notions, we were able to spend less time on later parts of the proposal such as the study of abstract types.

We proposed and advertised a student project at MSc level to develop parts of our exact real number computation system, but unfortunately no students volunteered.

Research Impact and Benefits to Society

Several of our results have already been taken up and used by other researchers. Streicher and his students in Darmstadt have adopted our general framework for notions of computability and have obtained some interesting further results (e.g. in [LS00, Roh01]). Rosolini *et al* have considered our general framework from a more categorical point of view. Royer has carried our work on the SR functionals into the area of higher type complexity: his work has confirmed a conjecture of ours, and has furnished evidence that the models we consider lend themselves naturally to a complexity-theoretic analysis [Roy00]. Sannella *et al* have considered the use of prelogical relations in the context of data refinement, and drawing on our expertise in higher-type computability we were able to supply an example showing that the extra generality offered by prelogical relations was essential [HLST00]. On a more informal level, we have enjoyed much mutually beneficial interaction with Abramsky, whose work on linear and process realizability has close affinities with ours.

The long term benefit to society of our work will probably manifest itself mostly in its impact on technology for correct software development.

Explanation of Expenditure

The expenditure on the project has been much as originally planned, except that Aspinall was employed for 2 months to work with Longley on Isabelle prototyping of program logics and on implementation of an exact real number package; and Power was employed for 4 months to work with Plotkin on the study of non-functional computation features via algebraic operations.

Further Research or Dissemination Activities

Whilst we feel we have been very active in disseminating our results through conference presentations and other lectures and seminars, both nationally and internationally, we have not yet completed the task of turning all these results into published papers. The most significant outstanding task is to complete the series of three papers [Lon01a, Lon01b, Lon01c] which will embody the central insights gained during the project and offer a more unified perspective on the whole subject area. Work on the whole series is more than half completed, and we expect to finish all three papers within the next six months at the outside. These papers, together with the already completed paper [Lon98a], will represent the principal concrete deliverables of the project.

As regards further research, our new EPSRC-funded project [FFL00] will build on some of our results and apply them in a more practically oriented research area: that of building proof systems for program verification (see 4 above). We believe this project will

represent a significant step towards one of the major long-term goals of the theoretical computer science community. The consolidation of our work on abstract types (see 6) will also fit well into this project.

References

[FFL00] M.P. Fourman, J.D. Fleuriot, and J.R. Longley. A proof system for correct program development. Case for support for EPSRC grant GR/N64571., 2000.

[HLST00] F. Honsell, J. Longley, D. Sannella, and A. Tarlecki. Constructive data refinement. In *Proc. FOSSACS, Berlin*, March 2000.

[Lon95] J.R. Longley. *Realizability Toposes and Language Semantics.* PhD thesis, University of Edinburgh, 1995. Available as ECS-LFCS-95-332.

[Lon98a] J.R. Longley. The sequentially realizable functionals. Technical Report ECS-LFCS-98-402, Department of Computer Science, University of Edinburgh, 1998. To appear in *Annals of Pure and Applied Logic.*

[Lon98b] J.R. Longley. When is a functional program not a functional program?: a walkthrough introduction to the sequentially realizable functionals. Standard ML source file, available from `http://www.dcs.ed.ac.uk/home/jrl/`, 1998.

[Lon99a] J.R. Longley. Matching typed and untyped realizability. In L. Birkedal, J. van Oosten, G. Rosolini, and D.S. Scott, editors, *Proc. Workshop on Realizability, Trento*, 1999. Published as Electronic Notes in Theoretical Computer Science 23 No. 1, Elsevier. Available via `http://www.elsevier.nl/locate/entcs/volume23.html`.

[Lon99b] J.R. Longley. Unifying typed and untyped realizability. Electronic note, available at `http://www.dcs.ed.ac.uk/home/jrl/unifying.txt`., 1999.

[Lon99c] J.R. Longley. When is a functional program not a functional program? In *Proc. 4th International Conference on Functional Programming, Paris*, pages 1–7. ACM Press, 1999.

[Lon01a] J.R. Longley. Notions of computability at higher types I. Submitted to Proceedings of Logic Colloquium 2000, 2001.

[Lon01b] J.R. Longley. Notions of computability at higher types II. In preparation, 2001.

[Lon01c] J.R. Longley. Notions of computability at higher types III. In preparation, 2001.

[LS00] P. Lietz and T. Streicher. Impredicativity entails untypedness. Submitted for publication, 2000.

[PP01a] G.D. Plotkin and A.J. Power. Operational semantics for algebraic effects. In *Proc. FOSSACS*, 2001. To appear.

[PP01b] G.D. Plotkin and A.J. Power. Semantics for algebraic operations. To appear in MFPS 2001, 2001.

[Roh01] A. Rohr. *A universal realizability model for sequential computation.* PhD thesis, University of Darmstadt, 2001. To appear.

[Roy00] J.S. Royer. On the computational complexity of Longley's H functional. Presented at Second International Workshop on Implicit Computational Complexity, UC/Santa Barbara, 2000.