Notions of Computability for General Datatypes

Case for Support to accompany EPS(RP)

M.P. Fourman G.D. Plotkin J.R. Longley

2 Proposed research and its context

A. Background

A1. Introduction One of the fundamental questions at the heart of computer science is: "What does it mean for an operation or function to be *computable* in principle?" For the case of functions on the natural numbers, the work of Church, Turing and Kleene in the 1930s yielded several mathematical characterizations of a good class of computable functions, the *partial recursive* functions. It is widely agreed that this coincides with the class of functions that can in principle be computed by a purely mechanical procedure.

The case of functions on the natural numbers is particularly significant, since any other kind of data that can be manipulated by a computer can ultimately be represented by natural numbers. However, this does not mean that questions of computability for all other datatypes are thereby automatically settled. For many kinds of data that occur in computing practice—functions, streams, continuations, abstract data values, objects, processes—the natural number representation will typically contain more information than will be accessible to "computations", at least in the senses likely to be of interest. This is because computations gain information about such data values only by *interacting* with them in ways prescribed by the programming language or environment. Thus, it is far from immediately clear what should be meant by a "computable function" on such datatypes; indeed, for some datatypes one can even find rival notions of computability, all perhaps equally natural and fundamental, arising from different representations of the data values and different protocols for interacting with them. (See Section B.I.1 below for some examples.)

In the proposed research, we will undertake a systematic investigation of notions of computability for a wide variety of datatypes, including higher-order function types, recursively defined types, abstract types, and object classes as in object-oriented programming styles. In each case, we will seek to identify the important classes of "computable operations", and to give a variety of different (syntactic and semantic) characterizations of these classes. We will also investigate the feasibility of designing programming languages based on some of these notions of computability, and will test the scope and usefulness of these notions by focusing on some specific application areas (e.g. search algorithms; exact real-number computation).

On the theoretical side, what this will provide is an understanding of fundamental concepts of computability for a range of datatypes, analogous to that provided for the natural numbers by basic recursion theory. On the more practical side, our work will provide semantic foundations for logics suitable for software development and verification, and will also inform and inspire the design and implementation of mathematically-based programming languages. Furthermore, we expect our work to have an impact on the particular application areas we consider, such as real-number computation.

A2. Relationship to other current research Much recent research in the semantics of computation exhibits a tension between two complementary ways of looking at computational systems: the *extensional* and *intensional* perspectives. Broadly, the extensional perspective focuses on *what* is computed—for example, on the mathematical function that relates outputs to inputs—while the intensional perspective focuses on *how* a computation occurs, i.e. on the process itself. In terms of our proposed research, one can loosely say that models of *computation* are intensional in nature, whereas notions of *computability* typically have a more extensional flavour.

The study of intensional models such as those based on *games* [AJM96, AM96, HO96] has been a very fruitful area of recent research. Many workers in the field (e.g. Abramsky and his group at Edinburgh; Hyland, Ong and their groups at Cambridge and Oxford) are currently engaged in a program of extending these models in various directions, and applying them to a wide range of datatypes, processes and computational paradigms. The proposed research will complement this by pushing forward a corresponding extensional theory of computability for a similar range of computational scenarios. Our aim is to see how far one can travel into this new territory whilst maintaining the logical simplicity and clarity afforded by an extensional framework. The two approaches are of course highly interdependent, and we expect our work to benefit a great deal from interaction with the researchers mentioned above. The distinctiveness of our proposed research, however, is that for us it is the notions of computability, rather than the models of computation, which will be the primary objects of study. Thus, for example, we will be interested in knowing which intensional settings give rise to the *same* extensional notion of computability (see [Lon95, Chapter 7] for some results of this nature); we will also be interested in properties of the extensional notions that are not dependent on any particular intensional model (see e.g. [Loa96]).

Along a different axis, the proposed research has affinities with work in the algebraic specification tradition by Tucker *et al* (e.g. [TZ88, TZ90, TZ94, BT95]), who have considered questions of computability for a range of datatypes such as streams and abstract types. However, our approach differs from theirs in its emphasis on the ideas and methods of denotational semantics; indeed, we believe that our programme will furnish a broader, more conceptual framework within which much of their work may be placed.

Our interest in general notions of computability has arisen from our study of *realizability models* [Lon95, LS97]. These have already given us a good handle on computability at higher types over the natural numbers, and we expect that the ideas of realizability will continue to play a major role in our research.

B. Programme and methodology

The proposed research will investigate classes of computable functions for a range of datatypes, including finite types over the natural numbers, recursive types, abstract types and object classes. It will also consider the more general question of what constitutes an "effective datatype".

There are two general criteria that we shall employ throughout the programme for deciding what should be considered a fundamental "notion of computability", or class of computable entities. The first is that such a class ought to arise from some intuitively compelling model of computation (e.g. register machines; dialogue games). The second (more objective) criterion is that a class should admit a variety of independent mathematical characterizations (as is the case for the partial recursive functions, for example). Our work on realizability models has already given us examples of such contrasting characterizations of the same extensional class (see [Lon95, Chapter 7]). The kinds of contrast we have in mind include the following:

- *Syntactic* characterizations (as the set of entities definable in some formal language) versus *semantic* characterizations (as the set of entities present as elements of some mathematical structure).
- Characterizations in terms of intensional objects that happen to behave extensionally (picked out e.g. by an *extensional collapse* construction) versus characterizations in terms of "inherently extensional" objects of some kind.
- Characterizations that involve different intensional models of a significantly distinct flavour.

Our strategy will be to proceed from more familiar to less familiar kinds of datatype. The rest of this section describes, in order, the specific topics on which we will focus.

I. Computability at finite types

The higher-order function types (or *finite* types) over the natural numbers have received a great deal of attention in recursion theory and computer science. From our point of view they are a good place to start our investigations of computability, not only because because there is already an established tradition of research in this area, but also because many other interesting and useful datatypes can be obtained from finite types, e.g. as *retracts* (see II below). Indeed, many results relevant to our concerns are already known, although at present they are scattered rather widely across the literature in recursion theory, domain theory, type theory and programming language semantics.

Our proposed research in this area falls broadly into two parts:

I.1. General theory of effective type structures Much previous work has been devoted to studying *particular* definitions of higher-type computability. We will take a more abstract view and develop a *general* theory of such notions. A mathematical framework for such a theory is provided by a notion of "effective finite type structure", together with a notion of "simulation" of one such type structure within another (see [Lon97]). Simple though these definitions are, we have recently shown that they give surprisingly powerful conceptual tools for unifying and clarifying much of the existing material. In addition, this abstract approach will shed light on particular concrete notions of computability, via a consideration of their place within the space of all possible such notions. For example, we are currently aware of three good notions of "computable partial functional" of finite type:

- The *sequential* computable functionals definable in Plotkin's language PCF [Plo77]; these may be characterized semantically via the *games* of Abramsky, Hyland *et al* [AJM96, HO96].
- The *parallel* computable functionals definable in PCF with 'parallel-or' and 'exists' operations [Plo77]; these coincide with the effective elements of Scott's classical domain model, and also with the partial functionals present in the familiar category of PERs [Lon95].
- The strongly stable or *intensionally-sequential* computable functionals (see I.2 below).

Whilst the first of these notions is weaker than the other two, Longley has recently shown that the second and third have no common upper bound. This fact is already a striking result, which one can interpret as saying that there can be no ultimate hierarchy of "all" possible computable functionals—a kind of "Church's anti-thesis for higher types".

One of the early goals of the project will be apply these abstract ideas to produce an extended *survey paper* on notions of computability at higher types, bringing together a diverse body of material within a uniform framework, and making explicit the connections between different strands of existing research. In particular, there is a significant corpus of work in the recursion-theoretic tradition from the '60s and '70s (e.g. [Kle63, GH77]) which is sometimes regarded as difficult and obscure—we will place this work in a modern setting and relate it to more recent developments in computer science. Our treatment will cover both *total* and *partial* notions of computable functional. In general it is the partial notions that arise more naturally from programming languages; however, the total notions—and their relation to the partial ones—are also of interest to computer science, as is shown by work of Berger [Ber93] and Plotkin [Plo97].

Besides the value of its conceptual contribution, we expect our abstract framework to inspire insightful new technical results. For example, we might inquire whether there are any other interesting hierarchies of computable partial functionals to be discovered. Such hierarchies could conceivably form a basis for new kinds of programming language, for instance. Ultimately we might hope to obtain a classification theorem, to the effect that the only "good" hierarchies were those in some concretely described class.

I.2. The intensionally-sequential computable functionals We will also devote some time specifically to the third (and much the least well-known) of the above classes of computable functionals. This class is especially interesting, as it embodies an intuitively "sequential" notion

of computability that is stronger than PCF. Its discovery is in essence due to Ehrhard [Ehr96], who showed that the functionals intensionally computable by *sequential algorithms* coincide with the *strongly stable* functions in a certain domain-theoretic model. Since then, the implicit notion of computability has been brought more clearly into focus via a realizability model discovered independently by van Oosten [Oos97] and Longley. Most recently, Longley has given a syntactic characterization of this class of functionals as those definable in PCF extended with a certain "universal" functional H.

Rather remarkably, although H cannot be implemented in the purely functional fragment of a language such as Standard ML [MTH90], it *can* be implemented using non-functional features of ML such as exceptions or references. However, the *behaviour* of H is purely functional, in the sense that it acts extensionally on its inputs. This suggests the possibility of a programming language that incorporates the power of H into its functional fragment. There are good practical reasons why this might be useful. For example, there are natural examples of programs in PCF+H which we believe would be (variously) impossible, inefficient or just inelegant to implement in pure PCF. For such programs, one would gain the transparency and ease of reasoning offered by pure functional programming, as well as increased scope for compiler optimization.

Besides consolidating our theoretical analysis of the intensionally-sequential functionals, we will investigate the practical applicability of this notion of computability. We will do this mainly by via prototype implementations (e.g. in Standard ML) of simple programming languages based on this notion. Firstly, this will allow us to investigate issues of computational feasibility and efficiency.[1] Secondly, these implementations will allow us (and others) to experiment with functional programs or styles of programming, and discover the kinds of task to which such an extended functional language is well-suited. We have in mind two particular application areas:

- *Search algorithms.* It seems that intensionally-sequential computation can be used to advantage in constructing general-purpose search algorithms, since control information is made available which can be exploited to prune the search tree.
- *Exact real-number computation.* Representations of real numbers via streams of approximations allow computations to be demand-driven and free from error accumulation. It seems that certain natural operations (e.g. Riemann integration for discontinuous functions) are not computable in PCF but become computable in the intensionally-sequential setting. Here we expect our ideas to have an impact on current research in real-number computation by Edalat, Escardó *et al* [EE96]. However, the connections between their setting and ours have yet to be explored in detail.

II. Other concrete datatypes

As mentioned above, many interesting datatypes—such as lazy lists and trees—arise as *retracts* of finite types. This means that notions of computability for the finite types can often be cheaply extended to corresponding notions for these other datatypes—provided, of course, that both halves of the retraction are themselves considered "computable" in the sense under consideration. (The point of this last caveat is that the class of computable retracts of finite types will vary from one notion of computability to another.)

In the case of intensionally-sequential computation, we anticipate that we will be able to obtain a very large class of types (including, for instance, arbitrary concrete data structures and recursive types) as computable retracts of finite types. (This is because of the existence of a *universal* finite type, of which all other finite types are retracts.) This should allow us to extend our various characterizations of this notion of computability easily to these other types, and indeed to expand our investigation of its practical programming possibilities in this way.

In some cases we wish to consider concrete types that do not arise as retracts in this way. Here the situation is much less clear: do the basic notions of computability remain the same, or do some of them split into several distinct notions in the presence of these extra types?

[1] An *inefficient* prototype implementation of the functional H has already been constructed!

Another challenging problem (recently considered by Plotkin) is to find good notions of *totality* for recursive types. Besides deepening our understanding of computability for these types, such notions would lead to useful logical principles for reasoning about termination of programs.

III. Abstraction and information hiding

III.1. Abstract types A more ambitious and speculative part of our programme will be the investigation of issues of computability for *abstract types*. In many modern programming languages, facilities for data abstraction are very important for the modular design of large programs. The basic idea is that we can only interact with the data values through some prescribed interface. Indeed, the finite types are abstract types in a certain sense, since (typically) the only way to interact with a function is via application.

Taking an extensional or "behavioural" view of datatypes, one is led to consider questions such as the following:

- Which functions to and from the abstract type are computable?
- When are two elements of the abstract type observationally equivalent?
- When are two implementations of the same abstract type signature observationally indistinguishable?

Even for abstract types with first-order signatures, these questions present a significant challenge. There is already a substantial body of work in this area from the algebraic specification community (see e.g. [TZ88, BT95, GM82, BHW95, HS96]). However, as far as we are aware, there has so far been very little work on truly *denotational* models for abstract types that capture notions of computability and behavioural equivalence (as achieved for simply-typed languages by the models in [Plo77, AJM96], for example).

We will seek natural and robust notions of computability for reasonable classes of abstract types, and investigate both syntactic and semantic characterizations of these notions. We expect that existing formalisms for abstract types (as in e.g. [MP82]), along with now well-known ideas of coinduction and bisimilarity (see e.g. [Pit94]), will be useful in constructing suitable syntactic models. Moreover, we are hopeful that some kind of realizability will allow us to obtain good semantic characterizations. A first line of attack will be to follow up the idea that implementations of abstract types can be regarded as values with System F types [MP82], and as such have interpretations in standard realizability models. To produce a semantic counterpart to abstraction we may need to go beyond these standard models; for example, by constructing permutation models. On the other hand, it may be that to obtain good models we will need to go beyond the usual notions of realizability. For example, in the standard setting the intensional models of computation are combinatory algebras, in which *application* is the fundamental operation, whereas it might be advantageous to consider intensional settings (such as Milner's π-calculus [Mil92]?) in which some more general kind of interaction was primitive.

In conjunction with our study of particular models, we will spend some time trying to give *general* descriptions of datatype abstraction in terms of appropriate categorical structure. In particular, given a category C to be thought of as a category of "concrete types and computable functions", we hope to make precise some notion of "the category of abstract types over C" which inherits its notion of computability from C.

The practical benefits of a semantic representation of abstraction will be to underpin principled language design, and to provide a basis for tools and techniques for program development and analysis.

III.2. Object classes A related problem is to find appropriate notions of computability for object-oriented languages such as Java [GJS96]. Object classes closely resemble abstract types in that internal representations are hidden and interaction is possible only via exported operations; they differ, of course, in that objects may have mutable state and so may be destructively modified in the course of computations. We are therefore led to think less in terms of mathematical functions

that return an output given an input, and more in terms of operations that *convert* some initial configuration of objects into a final configuration. But this need not be at odds with an extensional treatment of computability, if we consider such operations to have *linear* function types [Gir87] rather than the usual intuitionistic ones.

We hope that our study of abstract types, coupled with ideas from linear logic, will point the way towards an extensional treatment of computability for object classes. In the present programme we will undertake some preliminary exploration of these ideas; detailed investigation of this area may be a topic for future research.

IV. General notions of effective datatype

All the topics above are concerned with extending our understanding of computability to embrace *particular* classes of datatype. Alternatively one could ask a more fundamental question: what is a good *general* notion of an "effectively implementable datatype"? A good answer to this question might provide a kind of "Church's thesis" at the level of types, and would be likely to have repercussions in many areas of computer science.

We suspect that there are several reasonable answers to the above question, depending on just how general one wishes to be. We hope to make a start in identifying some of these notions and understanding the relationships between them. More particularly, from a certain point of view it seems reasonable to suppose that the notion of effective data algebra is no more general than the class of algebras that can be implemented as abstract types (cf. [TZ90]). In any case, it would be good to provide evidence for the status of some such good class of datatypes by giving several independent characterizations and showing that they coincide.

C. Relevance to beneficiaries

The proposed research is largely foundational in nature, and as such its main contribution will be to provide an understanding of certain fundamental concepts which will enable further progress in a number of areas of more direct practical concern. There are two main areas on which we expect our work to have a particular impact:

- *Logics for program verification and development.* A simple and usable program logic should have a clear operational interpretation of a kind that would be readily understood by programmers. In [Lon95, LP97] we have argued that in order to provide mathematical foundations for such logics, one should consider semantic models that closely reflect the notions of computability and behavioural equivalence embodied by the programming language. Our research will provide an in-depth understanding of many such models and their relationships to particular kinds of programming language. In this way it will provide mathematical underpinnings for current work on formalisms and tools for correct software development, such as the ongoing work at Edinburgh on Extended ML [KST97] and on proof assistance for programming (Lego, LAMBDA), and for related work elsewhere (e.g. Glasgow, Cornell, SRI, INRIA).
- *Language design and implementation.* A thorough mathematical understanding of fundamental and natural notions of computability will enable the design of programming languages that embody these notions in a clean and principled way. This will be most evident in parts I.2 and II of the project, which will specifically investigate the practical feasibility and benefits of a programming language based on the notion of intensional-sequentiality. The theoretical understanding gained from other parts of the project (e.g. part III on data abstraction) can also be expected to have an influence on future language designs.

In addition, we anticipate that part I.2 will have an impact on current work on computation with infinitary datatypes, e.g. exact real-number computation (Edalat, Escardó, di Gianantonio). This is itself a lively research area with promising applications to statistical physics, neural nets and dynamical systems. Finally, our work will provide foundational concepts for research on

complexity theory for higher types (Cook, Kapron, Hofmann), and perhaps even for more general datatypes.

D. Dissemination and exploitation

The main dissemination vehicles for the foundational results of this project will be conference and seminar presentations, WWW publication of preprints and reports, and journal and conference publications.

We will also develop prototype implementations and example application programs (primarily as student projects) to be disseminated among the international programming community via the Internet. This collection will form the basis for a course, given at MSc/PhD level, in the application of semantics to programming practice.

Our links with companies such as Persimmon, Harlequin and Abstract Hardware Limited will provide a rich source of problems/solutions.

E. Justification of resources

Manpower Longley will work full-time on the project as a postdoctoral research associate for the whole three years. Funding for Longley is requested on the AR1A to start at spinal point 9 with annual increments. We also seek funding for 15% of a computing officer (AD3.3) to provide infrastructure support for language prototyping and applications. and 5% of a secretary (CN3.3) to support publication and dissemination of our results. Profs. Plotkin and Fourman will each contribute an average of three hours a week for the parts of the project in which they are involved (see Diagrammatic Workplan), and also conduct termly project review meetings.

Travel We plan to attend an average of two international conferences a year, e.g. LICS, MFPS, POPL, ICALP, MFCS, CSL, TACS, CTCS, CAV, TPHOL, LAFACS. We request travel and subsistence support for five one-week visits to European centres (e.g. Aarhus, Sofia-Antipolis, Pisa), and one ten day visit to a number of centres within the US (e.g. CMU, SRI/Stanford, Pennsylvania, Montreal, Cornell). These visits will be made in conjunction with overseas conference travel. We also request support for two 2-3 day visits per year within the UK (e.g. Oxford, Cambridge, QMC).

Equipment A workstation, maintained over the project period, is requested to support language prototyping and the evaluation, in applications, of the novel language features we propose. We also request an appropriate contribution to consumables, shared networking, and fileserver provision.

References

[AJM96] S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. Submitted for publication, 1996.

[AM96] S. Abramsky and G. McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions (Extended abstract). *Electronic Notes in Theoretical Computer Science*, 3, 1996.

[Ber93] U. Berger. Total sets and objects in domain theory. *Ann. Pure Appl. Logic*, 60, 1993.

[BHW95] M. Bidoit, R. Hennicker, and M. Wirsing. Behavioural and abstractor specifications. *Science of Computer Programming*, 25, 1995.

[BT95] J.A. Bergstra and J.V. Tucker. Equational specifications, complete term rewriting systems, and computable and semicomputable algebras. *JACM*, 42(6), 1995.

[EE96] A. Edalat and M.H. Escardó. Integration in Real PCF (extended abstract). In *Proc. 11th Annual IEEE Symposium on Logic in Computer Science*, 1996.

[Ehr96] T. Ehrhard. Projecting sequential algorithms on strongly stable functions. *Ann. Pure Appl. Logic*, 77, 1996.

[GH77] R.O. Gandy and J.M.E. Hyland. Computable and recursively countable functions of higher type. In *Logic Colloquium '76*. North-Holland, 1977.

[Gir87] J.-Y. Girard. Linear logic. *Theoretical Comp. Sci.*, 50, 1987.

[GJS96] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. Addison-Wesley, 1996.

[GM82] J. Goguen and J. Meseguer. Universal realization, persistent interconnection and implementation of abstract modules. In *Proc. 9th ICALP*. Springer LNCS 140, 1982.

[HO96] J.M.E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II and III. Submitted for publication, 1996.

[HS96] M. Hofmann and D. Sannella. On behavioural abstraction and behavioural satisfaction in higher-order logic. *Theoretical Comp. Sci.*, 167, 1996.

[Kle63] S.C. Kleene. Recursive functionals and quantifiers of finite types II. *Trans. Amer. Math. Soc*, 108, 1963.

[KST97] S. Kahrs, D. Sannella, and A. Tarlecki. The definition of Extended ML: a gentle introduction. *Theor. Comp. Sci.*, 173, 1997.

[Loa96] R. Loader. Finitary PCF is undecidable. To appear, 1996.

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

[Lon97] J.R. Longley. Notions of computability at higher type. Unpublished notes from a talk at the *Domains III* workshop, Munich, 1997.

[LP97] J.R. Longley and G.D. Plotkin. Logical full abstraction and PCF. In J. Ginzburg, editor, *Tbilisi Symposium on Language, Logic and Computation.* SiLLI/CSLI, 1997. To appear.

[LS97] J.R. Longley and A.K. Simpson. A uniform approach to domain theory in realizability models. In *Darmstadt Workshop on Logic, Domains and Programming Languages*, 1997. To appear in *Mathematical Structures in Computer Science.*

[Mil92] R. Milner. Functions as processes. *Math. Struct. Comp. Sci.*, 2(2), 1992.

[MP82] J. Mitchell and G.D. Plotkin. Abstract types have existential type. *ACM Trans. Prog. Languages and Systems*, 10, 1982.

[MTH90] R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML.* MIT Press, 1990.

[Oos97] J. van Oosten. A combinatory algebra for sequential functionals of finite type. Technical Report 996, University of Utrecht, 1997.

[Pit94] A.M. Pitts. A coinduction principle for recursively defined domains. *Theoretical Comp. Sci.*, 124, 1994.

[Plo77] G.D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5, 1977.

[Plo97] G.D. Plotkin. Full abstraction, totality and PCF. In preparation, 1997.

[TZ88] J.V. Tucker and J.I. Zucker. *Program correctness over abstract data types, with error state semantics.* North-Holland, 1988.

[TZ90] J.V. Tucker and J.I. Zucker. Toward a general theory of computation and specification over abstract data types. In S.G. Akl, F. Fiala, and W.W. Koczkodaj, editors, *Int. Conf. Comp. Inf.* Springer LNCS 468, 1990.

[TZ94] J.V. Tucker and J.I. Zucker. Computable functions on stream algebras. In H. Schwichtenberg, editor, *Proc. Marktoberdorf Summer School on Proof and Computation.* Springer, 1994.