FINAL REPORT

Elementary Strong Functional Programming

EPSRC Grant Ref: GR/L03279

Investigator: Professor D. A. Turner

Computing Laboratory, University of Kent, Canterbury, CT2 7NF, UK

January 2000

Introduction

The project, funded by the UK Engineering and Physical Sciences Research Council from 14.10.96 to 13.10.99, was undertaken to investigate the practical viability of a discipline of strong functional programming proposed in [18]. The research associate was Dr Alastair Telford.

In strong functional programming all expressions are guaranteed to have normal form. This has a number of theoretical and practical advantages which are discussed in [18]. In order to retain the possibility of programming with infinite structures, which is an essential feature of pure functional languages such as Haskell, a key part of the methodology of [18] is to maintain a separation in the type system between *data*, which is known to be finite, and *codata* which is permitted to be infinite. This may be contrasted with the usual situation in lazy functional programming in which infinite lists for example, have the same compile-time type as ordinary finite lists.

The separation between data and codata leads to a distinction between *recursion*, which to be “safe” must be `well-founded`, and *corecursion*, which must be `productive`. Neither of these properies is decidable in the general case: a proof is required, which may require arbitrary ingenuity. In the discipline under investigation proofs are *not* required, we settle instead for only those forms of recursion and corecursion which can be recognised by some decision procedure, and seek to discover how practical it is live with such a constraint. Such a language would not be Turing-complete, but might be convenient for a range of purposes, including teaching.

1 Progress of the Research

Following [18] we were initially committed to allowing only structural recursion and corecursion. A function defined over an inductive data type is said to be structurally recursive if it calls itself on an immediate subcomponent of its argument, i.e. for lists the *tail*, for naturals the *predecessor* etc. There is an analogous but dual definition of structural corecursion.

Our intended source language allows higher order, polymorphic, free-form recursion equations with pattern matching, as in Miranda or Haskell, and we began looking at algorithms to recognise the structurally {co-}recursive subset of this. To recognise pattern matching which corresponds to nested structural recursion, such as Ackermann's function, and mutual structural recursion, is in general non-trivial. Abel, with whom we corresponded, shows how to do this using call-graphs in his *foetus* system [1].

The RA appointed, Alastair Telford, had done his doctoral research in the area of abstract interpretation, which is a powerful technique for inferring static properties of functional programs.

Some early experiments convinced us that many of the function definitions that interested us, such as *quicksort*, *gcd*, $unification$, etc. could be expressed as structurally recursive only by rewriting them in ways that in general seemed non-obvious and rather unnatural.

Given some initially promising results our research therefore concentrated on devising abstract interpretations that could recognise wider classes of recursive and corecursive definitions as "safe" in their original form, without the necessity for them to be rewritten to make them structurally recursive.

The recursive and corecursive case require different, albeit related, theories. In both cases we believe we have made significant advances on previously known results. A PhD student partly funded by the project did excellent related work.

2 Results of the research

Our results, reported in [10, 11, 12, 14, 15, 16, 17], fall into three main areas.

2.1 Ensuring the productivity of infinite structures

The mathematical test for productivity of a corecursive definition of an infinite structure is undecidable [3]. A decidable class of productive definitions (corresponding to what is called structural corecursion in [18]) can be tested by Coquand and Gimenez's notion of *guardedness* [5, 7]. In [14, 15] we describe a technique of abstract interpretation that extends this notion to a considerably wider class of definitions. To take a simple example the definition $evens = 2 : map\,(+2)\,evens$ is accepted as productive by our method, where it fails that of [5, 7]. Our method for recognising productiveness also appears to be strictly more powerful than that of Hughes, Pareto, et al. [9]. For example we can accept the lazy functional algorithms for *Hamming numbers* and *fiblist*.

2.2 Ensuring termination of recursive functions

We have also developed an abstract interpretation that provides a decision procedure for recognising a class of recursively defined functions which are total. The analysis, which we describe in [16, 17], uses the same domain of abstract values employed to analyse the dual corecursive case. The class of functions recognised includes the nested and higher order structural recursions accepted by Abel's foetus system [1] but also many non-structural recursions such as those of *gcd* (by Euclid's algorithm), *quicksort*, *mergesort*.

A decidable test for a broader class of terminating recursions than primitive recursion is described in Arkoudas and McAllester [2]. However, their system is first order and monomorphic, while our method is designed to work in a system that is higher order and polymorphic. The method we describe in [17] also includes an automatic subtyping mechanism that enables us to handle functions like *head* which are total only if considered over (in this case) non-empty lists, and also avoids the necessity, present in [2], for functions such as *gcd* to be rewritten in a special "Walther-recursive" style.

We believe these results, together with those for corecursion (see previous section) significantly advance the practical viability of an elementary strong functional programming discipline, by allowing a wider class of standard function definitions to be admitted than previously.

2.3 Exact continuous arithmetic

Alex Kaganovsky, a PhD student associated with the group, did outstanding work on the investigation of an important class of corecursive algorithms over infinite lists of integers. The problem area he studied is the efficient implementation of *unbounded precision arithmetic* on real and complex numbers represented as streams of signed digits. The problem of performing mathematically exact arithmetic on real numbers represented in a computer by streams of signed digits was first studied by Wiedmer [20], and in 1982 Carl Pixley at Burroughs carried out a practical implementation of Wiedmer's algorithms in a lazy functional language [13]. Boehm and Cartwright [4] later claimed superior performance for an alternative implementation of the reals as functions from rationals to rationals.

Kaganovsky carried out a detailed complexity analyis of both approaches and was able to overcome the alleged inefficiency of the Wiedmer/Pixley representation by some ingenious adjustments to the algorithms, which he implemented in Miranda. He also adapted the method to complex numbers represented in a *imaginary base* by a single stream of signed digits. Interestingly his algorithms for computing various functions on real and complex numbers are in most cases identical in the two cases except for the choice of normalisation function, which differs for the imaginary base.

Kaganovsky's results, which include algorithms for many analytic functions and contain many innovations, including a significant advance in efficiency terms on previous work, are described in two papers and his thesis [10, 11, 12].

3 Future Directions

We see several directions for further research following on from this work.

3.1 Efficiency study

A second aim of the project as stated in the original case for support was to test the hypothesis that a discipline of elementary strong functional programming lends itself to improved opportunities for efficient execution.

Due to the change of focus described earlier we made only preliminary investigations in this area, concentrating instead on developing improved algorithms for automatic detection of termination (and dually, of productivity). Demonstrating efficiency gains from this in a toy compiler would not be convincing. A serious study requires modifying a production quality functional language compiler - say GHC (Glasgow Haskell Compiler) - to incorporate (i) the data/codata distinction and (ii) our abstract interpretation algorithms for enforcing termination/productivity. One would then have to try to demonstrate in practise that the additional information thus available could be used to permit further optimisations over and above what the compiler can already do.

Clearly this would be a major project in its own right.

3.2 Relating to other recent work

A second set of issues for future investigation is the relationship between our work in [14, 15, 16, 17] and some other recent developments. We see two interesting avenues here:

i) One topic for future investigation arises from Cousot's observation that type checking can be implemented by abstract interpretation [6]. This raises the possibility that our methods of termination detection could be combined with Cousot's for type inference to obtain a single integrated abstract interpretation which infers types and enforces termination in a single compiler phase.

ii) It can be argued that for many purposes what is required of a program is not merely a guarantee that it terminates, but something stronger: an assurance that it terminates within feasible bounds of time and space utilisation - and for most purposes this means bounds that grow only polynomially with input size.

The idea of *polynomial functional programming*, that is a programming discipline in which polynomial time complexity is guaranteed has already been studied by Martin Hofman: in [8] he shows how this can be enforced with a linear type system. Further, Wadler [19] has shown that there are abstract interpretations of a functional program that can be used to provide asymptotic time complexity analysis. So there is some reason to think that our abstract interpretations for termination detection developed in [17] could be adapted to enforce the stronger condition of polynomial time termination.

4 Contacts

Further information may be obtained from the investigator at the University of Kent (see title page for address). There is a project web page at http://www.cs.ukc.ac.uk/people/staff/dat/esfp from which most of our papers and reports can be downloaded.

References

[1] Andreas Abel "Eine semantische Analyse struktureller Rekursion", Diploma Dissertation, 50 pages, Ludwigs-Maximillians-University, Munich, February 1999.

[2] Kostas Arkoudas & David McAllester "Walther Recursion", Proceedings CADE 13, LNCS 1104, pp 643-657, Springer, 1996.

[3] J W de Bakker & J N Kok "Towards a uniform topological treatment of streams and functions over streams", Report CS-R8422 Centre for Maths and Comp Sci, Amsterdam, 1984, also in Proceedings ICALP 1985.

[4] H-J. Boehm, R. S. Cartwright "Exact Real Arithmetic: Formulating Real Numbers as Functions" in Research Topics in Functional Programming, pp 43-64, (ed) D. A. Turner, Addison Wesley, 1990.

[5] Thierry Coquand "Infinite Objects in Type Theory", Proceedings TYPES93, pages 62-78, 1993.

[6] P. Cousot "Types as abstract interpretations", 24th ACM Symposium on Principles of Programming Languages, pages 316-331, ACM Press, 1997.

[7] E. Gimenez "Codifying Guarded Definitions with Recursive Schemes", Proceedings TYPES94, pages 39-59, 1994.

[8] Martin Hofmann "Linear Types and Non-size-increasing Polynomial Time Computation", Proceedings 14th Annual Symposium on Logic in Computer Science, Trento, Italy, ed G. Longo, IEEE Computer Society Press, 1999.

[9] R.J.M. Hughes, L. Pareto, A. Sabry "Proving the Correctness of Reactive Systems using Sized Types", 23rd ACM Symposium on Principles of Programming Languages, St Petersburg, Florida, ACM Press, 1996.

[10] A.Y.Kaganovsky "Computing With Exact Real Numbers in a Radix-r System", Electronic Notes in Theoretical Computer Science, Volume 13, 27 Pages, Elsevier 1998. *Revised version available as Technical Report TR 19-99, 30 pages, Computing Laboratory, University of Kent, October 1999.*

[11] A.Y.Kaganovsky "Exact Complex Arithmetic in an Imaginary Radix System", Technical Report TR 9-99, 30 pages, Computing Laboratory, University of Kent, July 1999.

[12] A.Y.Kaganovsky "Exact Computing in Positional Weighted Systems", PhD Thesis, 212 pages, September 1999. *Under submission to University of Kent.*

[13] C. P. Pixley "Demand Driven Arithmetic", Burroughs Corporation Austin Research Center, Internal Report ARC 82-18, Nov 1982.

[14] A.J.Telford, D.A.Turner "Ensuring Streams Flow" Johnson, ed, Algebraic Methodology and Software Technology - AMAST '97. LNCS 1349, pages 509-523, Springer, 1997.

[15] A.J.Telford, D.A.Turner "Ensuring the Productivity of Infinite Structures" Technical Report TR 14-97, 37 pages, Computing Laboratory, University of Kent, March 1998. *Under submission to Journal of Functional Programming.*

[16] A.J.Telford, D.A.Turner "Ensuring Termination in ESFP", presented at 15th British Colloquium in Theoretical Computer Science, Keele, April 1999, 14 pages. *To appear in Journal of Universal Computer Science.*

[17] A.J.Telford, D.A.Turner "A Hierarchy of Elementary Languages with Strong Normalisation Properties", Technical Report TR 2-00, 66 pages, University of Kent Computing Laboratory, January 2000. *A revised version is in preparation for submission to a journal.*

[18] D.A.Turner "Elementary Strong Functional Programming". R.Plasmeijer, P.Hartel, eds, First International Symposium on Functional Programming Languages in Education. LNCS 1022, pages 1-13, Springer, 1995.

[19] Philip Wadler "Strictness analysis aids time analysis", Principles of Programming Languages, San Diego, California, ACM, 1988.

[20] E. Wiedmer "Computing with Infinite Objects", Theoretical Computer Science, vol 10, pp 133-155 (1980).