Polynomial interpretation of stream programs

Hugo Férée[1], Emmanuel Hainry[2,5], Mathieu Hoyrup[3,5], Romain Péchoux[4,5]

[1]*ENS Lyon, 46 allée d'Italie, 69364 Lyon Cedex 07, France*
[2]*Université Henri Poincaré, Nancy-Université, France*
[3]*INRIA Nancy – Grand Est, Villers-lès-Nancy, France*
[4]*Université Nancy 2, Nancy-Université, France*
[5]*LORIA, B.P. 239, 54506 Vandœuvre-lès-Nancy Cedex, France*
hugo.feree@ens-lyon.fr,{hainry,hoyrup,pechoux}@loria.fr

Stream languages including lazy functional languages like Haskell allows the programmer to represent functionals, functions over functions. From this perspective, they can be understood as a way to simulate type 2 functions. There are many works in the literature that study computability and (polynomial time) complexity of such functions [5, 14]. The implicit computational complexity (ICC) community has proposed characterizations of such complexity classes using function algebra and types [8, 9, 15]. These results are reminiscent of former characterizations of type 1 polynomial time functions [2, 4, 12] that led to other ICC works using polynomial interpretations.

Polynomial interpretations [11, 13] are a well-known tool used in the termination analysis of first order functional programs for several decades. Variants, like sup-interpretations and quasi-interpretations [3], which allow the programmer to perform program complexity analysis have emerged in the last ten years. One of their drawbacks is that such tools are restricted to first order programs on inductive data types. The paper [7] was a first attempt to adapt such a tool to co-inductive data types and, more precisely, to stream programs. We provide a second order variation of this interpretation methodology that fits to stream computation by replacing usual polynomials with type 2 polynomials (basically polynomials with function variables) to interprete programs.

It allows us to characterize exactly the set of functions computable in polynomial time by Unary Oracle Turing Machine (UOTM), that is functions computable by machines using oracles where the oracle has only unary input. It can also be used in order to characterize the set of functions computable in polynomial time by Oracle Turing Machine (OTM), that is shown to be equivalent to the BFF algebra in [9].

The first characterization has two advantages. First, it gives a new and intuitive notion of stream computational complexity in terms of Turing Machine. Second, it shows that this natural class can be easily captured using an adaptation of the interpretation tool. Using this tool we can analyze functions of this class in an easier way (based on the premise that it is practically easier to write a first order functional program on streams than the corresponding Turing Machine). The drawback is that the tool suffers from the same problem as type 1 polynomial interpretation: the difficulty to automatically synthesize the interpretation of a given program (see [1]).

The latter characterization gives a natural view of a well-know complexity class BFF, just by changing the interpretation domain, that is allowing exponential arguments for type 2 variables. It illustrates that the first characterization on UOTM is natural and flexible because it can be easily adapted to other complexity classes. Finally, it can be interpreted as a negative result showing that the BFF class, whose purpose is to study functions from $\mathbb{N} \to \mathbb{N}$, is surprisingly not well-suited to describe stream polynomial complexity.

We also go one step further showing that these tools can be adapted to characterize the complexity of functions computing over reals defined in Recursive Analysis [10], since real numbers can be seen as streams of rational numbers and the polynomial time complexity defined using UOTMs matches the common definition of polynomial time complexity for real functions.

Details of this work can be found in [6].

References

[1] Roberto M. Amadio. Synthesis of max-plus quasi-interpretations. *Fundamenta Informaticae*, 65(1):29–60, 2005.

[2] Stephen Bellantoni and Stephen A. Cook. A new recursion-theoretic characterization of the polytime functions. *Computational complexity*, 2(2):97–110, 1992.

[3] Guillaume Bonfante, Jean-Yves Marion, and Jean-Yves Moyen. Quasi-interpretations. *Theor. Comput. Sci.. to appear.*

[4] Alan Cobham. The Intrinsic Computational Difficulty of Functions. In *Logic, methodology and philosophy of science III: proceedings of the Third International Congress for Logic, Methodology and Philosophy of Science, Amsterdam 1967*, page 24. North-Holland Pub. Co., 1965.

[5] Robert L. Constable. Type two computational complexity. In *Proc. 5th annual ACM STOC*, pages 108–121, 1973.

[6] Hugo Férée, Emmanuel Hainry, Mathieu Hoyrup, and Romain Péchoux. Interpretation of stream programs: characterizing type 2 polynomial time complexity. In O. Cheong, K-Y. Chwa, and K. Park, editors, *21st*

International Symposium on Algorithms and Computation (ISAAC '10), number 6506 in LNCS, pages 291–303. Springer, 2010.

[7] M. Gaboardi and R. Péchoux. Upper Bounds on Stream I/O Using Semantic Interpretations. In *Computer Science Logic*, pages 271–286. Springer, 2009.

[8] Robert J. Irwin, James S. Royer, and Bruce M. Kapron. On characterizations of the basic feasible functionals (Part I). *J. Funct. Program.*, 11(1):117–153, 2001.

[9] Bruce M. Kapron and Stephen A. Cook. A new characterization of type-2 feasibility. *SIAM Journal on Computing*, 25(1):117–132, 1996.

[10] Ker-I Ko. *Complexity theory of real functions*. Birkhauser Boston Inc. Cambridge, MA, USA, 1991.

[11] D.S. Lankford. On proving term rewriting systems are noetherien. tech. rep., 1979.

[12] Daniel Leivant and Jean-Yves Marion. Lambda calculus characterizations of poly-time. *Typed Lambda Calculi and Applications*, pages 274–288, 1993.

[13] Z. Manna and S. Ness. On the termination of Markov algorithms. In *Third hawaii international conference on system science*, pages 789–792, 1970.

[14] Kurt Mehlhorn. Polynomial and abstract subrecursive classes. In *Proceedings of the sixth annual ACM symposium on Theory of computing*, pages 96–109. ACM New York, NY, USA, 1974.

[15] Anil Seth. Turing machine characterizations of feasible functionals of all finite types. *Feasible Mathematics II*, pages 407–428, 1995.

[16] Klaus Weihrauch. *Computable analysis: an introduction*. Springer Verlag, 2000.