McCarthy's LISP and Basis for Theory of Computation

Hong-Yi Dai
University of Edinburgh, UK

It is well–known that LISP is an algebraic list–processing language designed for artificial intelligence, dedicated to symbolic computation, powered by function orientation, enhanced by program manipulation, and underpinned by garbage collection. Ever since it was born, LISP has been celebrated by practitioners as a genuine high–level, functional– and meta–programming language for potentially hard problems involving symbol manipulation. Largely overlooked is that LISP can also serve as a formalism for theory of computation. Indeed, it embraces and integrates ideas from all three major characterizations of computation, namely lambda calculus, general recursive function, and Turing machine. Unfortunately, both LISP itself and the formal basis distilled from it were not well received for theory of computation. 50 years later, given prominent and promising developments of the programming–language approach to computability and complexity, it is time to give McCarthy's pioneering work a reappraisal. We proceed as follows.

First, we review the development of LISP. We start with McCarthy's motivation for designing a programming language for artificial intelligence, and the inspirations behind the initial set of features: algebraic notation and list processing. Of other language features outlined in the 1st AI memo [7], we pay particular attention to two, namely conditional expression and recursive function definition (making use of the former), both later becoming the key components of McCarthy's "basis for a mathematical theory of computation" [12]. We note two revisions to the language documented in two subsequent AI memos – Memo 3 [8] introducing garbage collection and refining list operations, Memo 4 [9] adopting Church's –notation (to facilitate higher–order functions) and naming the language LISP (short for LISt Processor) – which simplified it to such an extent that McCarthy agilely realized its capability to describe computable functions in a much neater way "than the Turing machines or general recursive definitions used in [computable][1] function theory" [15, p. 219]. We highlight McCarthy's cast of LISP "both as a programming language and as a formalism for doing [computable] function theory" [15, p. 219] in the 8th AI memo [10] (draft of the landmark paper [11]) where he showed two similarities between LISP and Turing machine: one *shallow* by simulating Turing machine in LISP, the other *deep* by giving a universal function in LISP for LISP.

Second, we revisit McCarthy's computation–theoretical basis [12] distilled from LISP by parameterizing the computation domain. We briefly mention some elementary results in traditional theory of computation McCarthy reproduced in the formalism obtained by in–

[1] Following Soare's recommendation [17], we substitute the word *computable* for *recursive* in the term *recursive function theory*.

stantiating the computation domain with the set of natural numbers. We also cover some formal properties of the basis that, as McCarthy demonstrated, enable one to prove the equivalence of computations expressed in formalisms of various instantiations. We then turn to the unfortunate fact that, despite McCarthy's efforts, both LISP and its formal basis were not well received for theory of computation, not by his students of theoretical background nor by other researchers of theoretical expertise, a representative of the former group being Park [19], while one of the latter being Davis [1968]. We identify two reasons for the ill-reception: (1) LISP, focusing on the practical issue of briefly and straightforwardly expressing particular computable functions, "didn't address the questions that interested [computable-function theorists]" [15, p. 219]; (2) despite some initial results, those theorists remained "unconvinced of the *theoretical*[2] utility of any" of the formalisms [3]. However, we find inheritance of McCarthy's methodology in work decades later [5, 18, 4] that approached computability and complexity via programming languages and that developed sufficiently far to confront Park, Davis and others. Moreover, we see the heritage further into current research and development of implicit computational complexity [6, 2].

Looking back, we notice that McCarthy's discovery of LISP as a formalism for theory of computation might be somewhat accidental. However, we also sense a kind of necessity, owing to his position that "the question of whether there are limitations in principle of what problems man can make machines solve for him as compared to his own ability to solve problems, really is a technical question in [computable] function theory" [13, p. 28]. This position, from nowadays point of view a misconception may it be, however, helped settle the design of LISP. Looking forward, we see that the programming-language approach to theory of computation McCarthy pioneered is shining again. Moreover, McCarthy's hope for a relationship between computation and mathematical logic "as fruitful in [this] century as that between analysis and physics" [14, p. 69] is coming true.

[2] Davis' emphasis.

Acknowledgment

We appreciate Herbert Stoyan's donation of his Lisp archive to the Computer History Museum [1]. We especially thank Paul McJones of the museum's Software Preservation Group, for his dedicated efforts to preserve as many historical materials on Lisp [16], and also for his kind support of this work and constructive comments on this report.

References

[1] Computer History Museum (ed.), "The Herbert Stoyan Collection on LISP Programming". Lot Number X5687.2010, 2010.

[2] U. Dal Lago, "A Short Introduction to Implicit Computational Complexity", in N. Bezhanishvili, G. Valentin (eds.), *Lectures on Logic and Computation*, Lecture Notes in Computer Science 7388, Springer, 2012, pp. 89-109.

[3] M. Davis, "Review: John McCarthy, a Basis for a Mathematical Theory of Computation, Preliminary Report; J: McCarthy, P. Braffort, D. Hirschberg, a Basis for a Mathematical Theory of Computation".

Journal of Symbolic Logic 33(1). 1968.

[4] N.D. Jones, *Computability and Complexity: from a Programming Perspective*, The MIT Press, 1997.

[5] A.J. Kfoury, R.N. Moll, M.A. Arbib, *A Programming Approach to Computability*, Springer, 1982.

[6] S. Martini, *Implicit Computational Complexity*, http://homes.di.unimi.it/~sel/scuola2009/martini-aila-2.pdf, 2009

[7] J. McCarthy, *An Algebraic Language for the Manipulation of Symbolic Expressions*. AI Memo 1. Cambridge, MA, USA: Artificial Intelligence Laboratory; Massachusetts Institute of Technology, 1958.

[8] J. McCarthy, *Symbol Manipulating Language – Memo 3 – Revisions of the Language*. AI Memo 3. Cambridge, MA, USA: Artificial Intelligence Laboratory; Massachusetts Institute of Technology, 1958.

[9] J. McCarthy, *Symbol Manipulating Language – Memo 4 – Revisions of the Language*. AI Memo 4. Cambridge, MA, USA: Artificial Intelligence Laboratory; Massachusetts Institute of Technology, 1958.

[10] J. McCarthy, *Recursive Functions of Symbolic Expressions and Their Computation by Machine*. AI Memo 8. Cambridge, MA, USA: Artificial Intelligence Laboratory; Massachusetts Institute of Technology, 1959.

[11] J. McCarthy"Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I". *Communications of ACM* 3(4), 1960, pp. 184-95.

[12] J. McCarthy, "A Basis for a Mathematical Theory of Computation, Preliminary Report". In *Papers Presented at the May 9-11, 1961, Western Joint IRE-AIEE-ACM Computer Conference*, ACM, 1961, pp. 225-38.

[13] J. McCarthy, "Towards a Mathematical Science of Computation". In *IFIP Congress*, 1962, pp. 21-28.

[14] J. McCarthy, "A Basis for a Mathematical Theory of Computation". In *Computer Programming and Formal Systems*, P. Braffort, D. Hirschberg (eds.), North-Holland, 1963, pp. 33-70.

[15] J. McCarthy, "History of LISP". *SIGPLAN Notices* 13(8), 1978, pp. 217-23.

[16] P. McJones, "History of LISP". Software Preservation Group, Computer History Museum, 2005, http://www.softwarepreservation.org/projects/LISP/

[17] R.I. Soare, "Computability and Recursion". *Bulletin of Symbolic Logic* 2(3), 1996, pp. 284-321.

[18] R. Sommerhalder, C. van Westrhenen, *The Theory of Computability: Programs, Machines, Effectiveness and Feasibility*, Addison-Wesley, 1988.

[19] H. Stoyan, "LISP History". *Lisp Bulletin* 3, 1979, pp. 42-53.