A note on LISP universal S-functions

C. R. Jordan

Department of Computer Science, Lanchester Polytechnic, Eastlands, Rugby, Warwickshire

This note describes some inconsistencies discovered in the LISP Universal S-Function defined in a paper by McCarthy and gives methods of correcting these inconsistencies.

(Received May 1972)

Introduction

This note describes some inconsistencies discovered in the LISP Universal S-function *apply* defined in a paper by McCarthy (1960). The Universal function that is defined in this paper by McCarthy is not the same as the one defined in the LISP 1.5 Programmers Manual (McCarthy *et al.*, 1963). Both these Universal functions refer to functions *assoc* and *pair* and these have different definitions in the two papers. This note uses the definitions of the 1960 paper throughout.

An attempt was made to code the function *apply*, given on page 189 of the CACM containing the McCarthy (1960) paper, for an Elliott 803 computer. This work was undertaken as a student project (Baxter, 1971). It was decided that the Universal function given in McCarthy's paper, rather than the one in the LISP 1.5 Manual would be coded as it was felt that this would provide fewer problems—in particular, there would be no need to implement dot notation, although, of course, this would put a restriction on the S-expressions that the system could deal with.

It was found that the constructed program did not evaluate recursive functions correctly, except in the trivial case when the answer could be obtained before recurring.

The Universal function *apply*, referred to above, calls another function, *eval*, defined on the same page of the McCarthy paper. The three lines of the definition of *eval* to be discussed are those corresponding to the expression, e, undergoing evaluation being atomic
i.e.

$$\text{atom}\,[e] \rightarrow \text{assoc}\,[e; a] \qquad (1)$$

the head of the expression, e, being a named function, so that the head is an atom, but not QUOTE, ATOM, EQ, COND, CAR, CDR or CONS
i.e.

$$T \rightarrow \text{eval}\,[\text{cons}\,[\text{assoc}\,[\text{car}\,[e]; a];\ \text{evlis}\,[\text{cdr}\,[e]; a]]; a] \qquad (2)$$

and lastly, the head of the expression, e, being a LAMBDA expression
i.e.

$$\text{eq}\,[\text{caar}\,[e]; \text{LAMBDA}] \rightarrow \text{eval}\,[\text{caddar}\,[e]; \text{append}\,[\text{pair}\,[\text{cadar}\,[e];\ \text{evlis}\,[\text{cdr}\,[e]; a]]; a]] \qquad (3)$$

(In the definition as given in McCarthy's paper, a bracket is missing in both the LAMBDA and LABEL lines.)

The notation used in this note is as follows: the version of *eval* incorporating the three lines above is called version(1, 2, 3); if line (2) is replaced by line (4), defined later, the version so obtained is referred to as version(1, 4, 3). Only these three lines differ from one version to another.

The fault and its correction

It is now shown, with the help of an example, that line (2) leads to an erroneous double evaluation of the arguments of recursive named functions. *Eval* reaches line (2) only when evaluating a recursive call of a named function. For example, if

$$\emptyset = (\text{LABEL FF}(\text{LAMBDA}(X)(\text{COND}((\text{ATOM } X)X)\ ((\text{QUOTE } T)(\text{FF}(\text{CAR } X))))))$$

then apply $[\emptyset; ((A))]$ eventually leads, after entering *evcon*, to eval $[(\text{FF}(\text{CAR } X)); \alpha]$ where $\alpha = ((X(A))(\text{FF } \emptyset))$. *Eval* is then re-entered from line (2), to give

$$\begin{aligned}\text{apply}[\emptyset; ((A))] &= \text{eval}\,[(\emptyset\ A); \alpha]\\ &= \text{eval}\,[((\text{LAMBDA}(X)\omega)A); \beta]\end{aligned}$$

where ω starts (COND, and $\beta = ((\text{FF } \emptyset)(X(A))(\text{FF } \emptyset))$;

$$= \text{eval}\,[\omega; \text{append}\,[\text{pair}\,[(X);\ \text{evlis}\,[(A); \beta]]; \beta]]\,.$$

Clearly evlis $[(A); \beta]$ is an error that has arisen due to the attempted double evaluation of the argument (CAR X) of the function FF.

In general, with a recursive named function, *eval* is re-entered from line (2) at some stage, and then again from line (3). Prior to each of these entries *evlis* is applied to the list of arguments, i.e. cdr $[e]$. This erroneous double evaluation must be avoided by changing line (2) as the evaluation is required for non-recursive functions at line (3). It can be avoided by either:

(*a*) quoting the results of the evaluation by *evlis* at line (2), so that this line is replaced by

$$T \rightarrow \text{eval}\,[\text{cons}\,[\text{assoc}\,[\text{car}\,[e]; a];\ \text{appq}\,[\text{evlis}\,[\text{cdr}\,[e]; a]]]; a] \qquad (4)$$

or

(*b*) leaving the evaluations until the LAMBDA line, i.e. line (3), is reached, in which case line (2) is replaced by

$$T \rightarrow \text{eval}\,[\text{cons}\,[\text{assoc}\,[\text{car}\,[e]; a]; \text{cdr}\,[e]]; a] \qquad (5)$$

Alternative versions

On pages 189 and 190 of the paper (McCarthy 1960), there are some numbered notes concerning the definition of this Universal function. If we construct the function *eval* from these notes we find differences from version(1, 2, 3) at all three of these lines. Note 3 implies that line (1) should read

$$\text{atom}\,[e] \rightarrow \text{eval}\,[\text{assoc}\,[e; a]] \qquad (6)$$

note 4 implies that line (2) should read as line (5); and note 6 implies that line (3) should read

$$\text{eq}\,[\text{caar}\,[e]; \text{LAMBDA}] \rightarrow \text{eval}\,[\text{caddar}\,[e];\ \text{append}\,[\text{pair}\,[\text{cadar}\,[e]; \text{cdr}\,[e]]; a]] \qquad (7)$$

The version(6, 5, 7) supposedly evaluates retrieved pairings at line (6) that have been placed on the association list, unevaluated, at line (7). In the paper, McCarthy then follows with a simulation of the evaluation of the function FF. This simulation, however, uses neither version(1, 2, 3) nor version(6, 5, 7). Instead, it appears to use version(6, 4, 7). This last version simulates the evaluation of FF correctly, whereas version (6, 5, 7) goes into a loop. It is shown here, however, that line (7) is at fault and that when this is corrected it may be combined with either lines (6) and (4) or with lines (6) and (5).

When version(6, 5, 7) of *eval* evaluates the form (FF(CAR X)), line (5) retrieves the definition of FF and line (7) places a new pairing for X on the association list a. Since neither line evaluates the argument of FF before it is bound to X, the pairing is (X(CAR X)). This binding is clearly circular and leads to looping. With this particular example, version(6, 4, 7) avoids the looping since line (7) is reached only after the definition of FF has been retrieved, the argument evaluated, and the result quoted. This somewhat lucky sequence of events is not always the case as the following example shows.

Let
$\emptyset$ = (LABEL FN(LAMBDA(Y)(COND((ATOM Y)X)
((QUOTE T)((LAMBDA (X)X)(FN(CAR Y))))))) .

The value of this function applied to any S-expression is the value bound to the free variable X. For example,

(LAMBDA(X Z)($\emptyset$ Z))

applied to a list of two arguments, gives bindings to X and Z, applies $\emptyset$ to the value of Z and eventually returns the value bound to X. Version(6, 4, 7), when evaluating the action part of the second condition action pair, produces the pairing (X(FN(CAR Y))) before evaluating X. Evaluation of X then involves evaluating (FN(CAR Y)), which involves applying $\emptyset$ to the reduced argument. Eventually the exit clause is taken, which again attempts to evaluate X and so on. In this example the pairing that leads to error is not explicitly circular, but the calculations implicit in it are.

The way to prevent new bindings interfering with evaluations that should be carried out in the context of earlier bindings, is to perform these evaluations first. The interpreter cannot rely on line (4) to do this before line (7) is reached, since in some cases (nested LAMBDA expressions) line (7) is arrived at independently of line (4). Thus, line (7) should be replaced by

eq [caar [e]; LAMBDA] →
eval [caddar [e]; append [pair [cadar [e];
appq [evlis [cdr [e]; a]]]; a]] (8)

and can now be combined with either line (4) or line (5) to give the same effect.

Summary

The three versions of the LISP Universal function, namely versions(1, 2, 3), (6, 5, 7), (6, 4, 7), that can be obtained from McCarthy's paper all have errors in them. Correct Universal functions are versions(1, 5, 3) and (6, 5, 8). In both of these correct-versions, it is permissable to substitute line (4) for line (5).

Acknowledgement

These notes are the results of discussions with Mr. A. A. Stravrinides of the Department of Cybernetics, Brunel University.

References

BAXTER, E. J. (1971). A project for the degree of B.Sc. in Mathematics, Lanchester Polytechnic.
MCCARTHY, J. (1960). Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part 1. *CACM*, Vol. 3, 1960.
MCCARTHY, J. *et al.* (1963). *Lisp* 1.5 *Programmers Manual*, MIT Press, 1963.

Book review

Computer Hardware Theory, by W. J. Poppelbaum, 1972; 730 pages. (*Collier-Macmillan*, £7·65)

Professor Poppelbaum is a very learned man, a physicist and mathematician of the highest capability and yet he is also a practical man with an engineer's interest in the application of science. Perhaps such men should not write books which claim to be, quoting from the Preface: 'as far as possible autonomous, in the sense of not needing prior acquaintance with other Engineering or Physic treatises'. When they are modest men, like Poppelbaum, they tend to imagine their less gifted brethren are cleverer than they are. This book of seven hundred pages is a veritable encyclopaedia of the science underlying the whole range of computer hardware; to quote, once more, from the preface: 'in spite of this wealth of material to be presented, every effort has been made to guarantee continuity of thought and adequate proofs'. Indeed every effort has been made but it makes for indigestible reading for ordinary men.

These unkind generalities aside, the book has many virtues. It is comprehensive indeed and does start each section from first principles even though it progresses in each topic to advanced study too quickly. The first three chapters are on basic Physics and Dynamics, ranging from the rules of simple vector addition through to grad, div and curl in Chapter 1, in 24 pages, including several simple illustrative experiments. Chapter 2, of 49 pages, is in the author's words 'a slight tour de force' on Electricity and Magnetism, rounded off with Maxwell's equations. The third chapter covers Waves, Particles and Quantum Theory and is wide as well as deep enough for most peoples' needs.

There are chapters on fundamental theory of semiconductors and semiconductor devices which lead into computer logic circuits and a brief treatment of register and logical operations. Chapter 7 deals with Analog, Hybrid and Stochastic Circuits, bringing in DDAs, digital filters and considerations of digital simulation of analog systems.

The chapter on Circuit Analysis is, once more, a tour de force. The author leaves little out, dealing with graphical methods for treating non-linear devices, negative resistance, gain and bandwidth in flip-flops, phase-plane theory, the Laplace Transform, charge storage theory of transistors, switching time calculations, pole-zero techniques; it is all here.

There are chapters that deal with semiconductor processing technology including computer graphics design aids, memory stores ranging from hierarchical considerations through to details of 'Bubbles', Cryotrons and Surface Wave Devices. 'Electron Beam and Matrix Devices' deals with readers and printers (mechanical), computer graphics devices and television.

Finally there are chapters on Light Theory, and Light Beam displays, backed up by treatments of Tensor Calculus and Fourier Transforms, and a chapter on Statistics, Tolerance and Noise.

It is not normally good reviewing to quote a detailed list of contents of a book but in this case it seems proper, for this one covers so much. Each chapter is adequately backed by reference lists, about half of which are to text books.

The book aims to be a teaching text and has exercises at the end of each chapter. It is more truly a handbook or reference book of more use to teachers and workers in the computer field who will find it invaluable as a starting point for further reading. It is hard to imagine any topic in the field in the next year or so to which Professor Poppelbaum's book will not provide a lead-in.

B. S. WALKER (Reading)