SESSION VII

DATA GENERAL USERS' MEETING

PATRICIA SIPLE, *University of Rochester, President*

SESSION VIII

PROGRAMMING LANGUAGES AND METHODOLOGIES: A TUTORIAL SESSION

JOHN W. COTTON, *University of California, President*

LISP: An introduction and some great programs

LAURENT SIKLÓSSY

Department of Information Engineering, University of Illinois at Chicago Circle, Chicago, Illinois 60680

Some of the advantages of LISP and some of its unsavory reputation are discussed. Some LISP programs in a variety of areas (chemistry, medicine, question answering, natural language, mathematics, airline tariffs, etc.) are briefly described.

The task proposed to me, to introduce the reader to LISP in a few pages, is beyond my capabilities. So let me just try to dispel some misconceptions about LISP, give the flavor of LISP programming, and mention very briefly some great (or possibly not so great) programs written in LISP.

There is undoubtedly a negative prejudice floating "in the air" about LISP. Some of my undergraduate students came to a third semester course already quite brainwashed about LISP. Here are some of their comments: "LISP is a very difficult, if not impossible, language to learn." "LISP is no longer used; it is a dying language. It is quite old anyway." "LISP is used only as a playtoy language in universities." "LISP is unreadable." "LISP? Couldn't we just forget about it?" Let us review each of these points and determine the extent to which it may be correct.

LISP is Very Difficult

LISP is difficult to some and perhaps even to many. The fault lies not with LISP, but with the very poor mental habits that have been acquired by frequenting such languages as FORTRAN, COBOL, or assembly language.

These languages make a person think control (where am I, where am I going next?) and think in terms of

pigeonholes holding information. The LISP programmer thinks flow of information: What information am I receiving? How do I transform it? Where will it go to be processed next? There is no doubt that information processing is indeed processing information, and that one thinks information processing much more readily in LISP than in the languages mentioned above and many other languages, too.

LISP is Old

LISP is old indeed, but instead of just aging gracefully, it appears to have improved with age. Starting in 1975-1976, there has been considerable renewed interest in LISP shown by theoretical computer scientists. The semantics of LISP continue to be among the cleanest of any programming language, old or new.

There is also more to a language than the language itself. The functional description used by LISP has met with no improvements: The same notation, or some variants of it, is still used in most branches of mathematics. The concept of a function remains fundamental. Since it is the basis of LISP programming, one experiences a feeling of resting on very solid ground in LISP. The same cannot be said of many other languages, which were the creations of committees.

In the past few years, LISP has been integrated in user-friendly environments. It appears very well suited to timesharing. Sophisticated LISP editors, debuggers, and error recovery systems have turned LISP into a remarkable tool.

The author is a Visiting Professor at the University of Illinois at Chicago Circle. His permanent address is the Computer Sciences Department, University of Texas, Austin, Texas 78712.

LISP is Restricted to Universities

This statement is essentially true. It is certainly the case that students do not ingratiate themselves to future employers by quoting their knowledge of LISP.

LISP is Unreadable

To some extent this is true. The designers of LISP made some errors, which have not been corrected since. They allowed functions of an indeterminate number of arguments, which in turn necessitated the "Lots of Insane and Stupid Parentheses" (or similar acronym). They did not maintain consistency in evaluation: Most functions evaluate their arguments, but some (that should, for uniformity) do not, such as GO. And the language lacks a variety of repeating or looping statements that can be easily added, but really should be there to start with.

Actually, and quite unfortunately, programming languages are all unreadable to a large degree. A nicely formatted, or "pretty-printed", in LISP jargon, LISP program appears just as readable as a PASCAL or PL/1 program, and it is significantly more readable than a FORTRAN program.

Couldn't We Just Forget About LISP?

Not yet. The day will come when new languages will appear that provide good facilities for expressing thoughts and are also readable. We are looking forward to that day, but since that day does not appear to be tomorrow, for the time being let's talk LISP (Siklóssy, 1976).

THE APPEARANCE OF LISP

LISP uses parenthesized Polish prefix notation, the parentheses being required by the lack of restriction on some functions. So, to add 3 and 4, we write: (PLUS 3 4). This actually constitutes a complete LISP program. A complete program usually consists of a sequence of smaller programs, such as the one above. Usually, every program consists of a function, such as PLUS, applied to some arguments, such as 3 and 4. Some functions permit one to define new functions, which in turn can be applied to arguments.

For those who think that writing 3 + 4 is really easier, let them think again. First, since there are only a limited number of characters on a typewriter or terminal, one would soon run out of ways to characterize new functions. In addition, the notation used in 3 + 4, called infix notation, happens to be highly undesirable and inefficient. In fact, Polish prefix notation is not the best either. Postfix notation, in which the function name comes after the arguments as in 3 4 PLUS, has many advantages. Our use of infix, and our preference for it, comes from many years (at least 10) of drill and practice. There is no doubt that we should switch our entire educational system to

use postfix notation, but it would be at the formidable cost of having to retrain all our teachers.

A LISP programmer's session at a terminal is typically the following: He defines some new functions, runs them, modifies them, runs them again, incorporates them into more complex functions, runs these, and so on. Each new function need be only marginally more difficult than the preceding function, and in this way, quite complex functions can be built rapidly. The productivity of LISP programmers is very high, and in one case (discussed below), absolutely phenomenal.

SOME GREAT—AND PERHAPS NOT SO GREAT—PROGRAMS IN LISP

We shall now describe several programs written in LISP. Some can be considered great programs, some perhaps not. In a few years, they may well be sneered at. They represent contributions in a variety of fields. We have left out many outstanding programs, some of which are certainly "greater" than some, or all, of the ones mentioned here. I have looked for diversity above quality per se. The programs are given in no particular order, except that I took the liberty of including two programs to which I contributed at the end of this all too brief description, which does not do justice to these sophisticated programs.

DENDRAL in Chemistry

DENDRAL (Buchanan, Sutherland, & Feigenbaum, 1969) incorporates considerable knowledge of chemistry. The DENDRAL program (there have been several versions of DENDRAL, we sketch just one of these) accepts as input the results of experiments carried out to determine the chemical composition of a substance. From these results, the program hypothesizes a few chemical formulas. For each hypothesis, the program "pretends" to perform the same physical experiment used originally on the substance: The program knows enough chemistry to determine what would actually happen to a substance having the particular chemical composition. If the results of the pretended experiment agree with the actual physical results, the formula has been found correctly. Otherwise, a new formula is hypothesized.

Results of DENDRAL have been published in chemical journals; some knowledge of chemistry had to be modified.

MYCIN in Medicine

MYCIN (Davis, Buchanan, & Shortliffe, Note 1) is a diagnostician. Given a patient with certain symptoms, the program requests additional tests, and eventually makes a diagnosis and suggests therapy. A successor program (Pople, 1977) is considered in its area of expertise (several hundred diseases out of a total of a few thousand, many of which are extremely rare) to

outperform the average medical hospital-affiliated diagnostician (who in turn can be expected to leave far behind the average practitioner).

AM in Mathematics (Mathematical Discovery)

AM (Lenat, 1977) is given some information about elementary mathematics and, more importantly, about concepts: What makes a concept interesting? How can one generate new concepts? How can one test the plausibility of a new concept, and so on? For example, having discovered the concept of a prime number, which are numbers that are almost never divisible (they are divisible only by one and themselves), AM "thought" of the "dual" concept, in a certain sense: numbers that are maximally divisible, that is, that admit many divisors. AM then hypothesized (but could not prove, except by testing extensively, which is not a proof) a variety of interesting properties of these numbers. For a while, it was thought that these numbers had never been studied. Unfortunately (or fortunately), it was discovered that, in fact, they had been investigated.

The author of AM has kept careful tallies of his productivity in writing AM, a very sophisticated program. His productivity has been phenomenal: some 1,000 lines of LISP code per day for several months.

MATHLAB in Mathematics (Mathematical Tools)

MATHLAB (Bogen, 1975) embodies a variety of programs that can help physicists, chemists, and so on, perform integrations, solve differential equations, and so on. MATHLAB is used productively by professionals, and since mathematicians are no longer interested in such practical mathematics, MATHLAB is without peer.

MATHLAB is also a high-performance program: It performs its mathematical tasks efficiently and rapidly. It is one of several high-performance programs that disprove the oft-expressed belief that LISP programs are slow.

SOPHIE in Natural Language and Training

SOPHIE (Brown & Burton, 1975) consists of two components: a natural language component that permits dialogue with a user, and a component that understands the functioning of some domain of interest, such as a machine or an electric circuit. The user can find out what would happen to the machine if he made a particular move; he can be presented with a faulty machine that must be diagnosed, and so on. SOPHIE was built for training, since a user can gain experience about a variety of machines and their problems without incurring the cost of preparing the machines, modifying them, damaging them to make them faulty, and so on. To a sophist, SOPHIE's sophistication may be minimal, but the goal of SOPHIE's authors did not lie there.

MOON ROCKS in Natural Language

Question Answering

The MOON ROCKS (Woods & Kaplan, Note 2)

program also consists of a natural language component and a data base of facts about moon rocks. It has been used productively by geologists and other scientists who wanted to find out what was known about the rocks. Unlike SOPHIE, and because of the nature of the knowledge it has, the MOON ROCKS program cannot "run" a machine, so that a dynamic component is missing.

PARRY in Psychology

PARRY (Colby, Weber, & Hilf, 1971) is a program that incorporates knowledge about paranoia: It behaves as a paranoid entity. A natural language component permits PARRY to interact with human beings. PARRY has been made to converse with expert psychologists and psychiatrists, the majority of whom did not guess that they were communicating with a program and not a human being. In fact, when apprised of the identity of their interlocutor, several of the human professionals refused to believe that they had failed to recognize a program, and that therefore they really had not been speaking to a program but to a real human being.

LAWALY in Robot Problem Solving

LAWALY (Siklóssy & Dreussi, 1973) is given a description of a robot's capability and a description of a desired state that must be reached from the present state by using the robot's capabilities. A solution requires the robot to perform a sequence of actions; obtaining this sequence of actions results in the solution of the problem of reaching the desired state. LAWALY tries to sort the various features of the world according to the amount of freedom that remains after some feature is achieved and is frozen (i.e., cannot be changed subsequently). For example, leaving a door closed constricts the robot considerably, since she can no longer go through that door. On the other hand, leaving a door open would leave the robot with considerable leftover freedom to achieve other tasks. By attempting subtasks from the least to the most constraining, very long tasks (several hundred steps long) have been solved. These tasks are far beyond the capabilities of human beings who are not familiar with the problem environment. We can assume that we survive in our world because, in fact, we can order subtasks according to the same principles of remaining freedom. We probably could not survive in a world, such as chess, where every single move has widespread consequences everywhere, that is, where everything is intimately tied together.

Airline Tariff System in Sophisticated

Question Answering

The airline tariff system (Siklóssy, 1978) is an example of an "impertinent" question-answering system. Such a system not only answers the user's questions, but also provides unsolicited information that might help the user make better decisions. For example, the system might suggest that a user remain 1 day longer at his or her destination, since he or she would then be eligible

for a lower priced excursion fare. In our salesman-oriented society, where hiding or distorting information is standard practice and even considered desirable, the idea of impertinent question-answering systems, which not only answer questions accurately, but even suggest new questions that had not been considered, may well prove to be revolutionary.

The airline tariff system computes tariffs for complicated routes in seconds, thereby far outperforming human routing agents. The results obtained are also better than those produced by human agents: The program will often calculate cheaper routes, or make better suggestions (human agents seldom make suggestions, anyway) about alternate, improved routes.

CONCLUSIONS

We have sketched hastily some LISP programs in a variety of areas. Many worthwhile programs have not been mentioned. At this point, we must say that each of the described programs could have been written in a different programming language. LISP may have been chosen because of the lack of other good candidate languages on the computer available, or because the programmers only knew LISP. Neither of these suppositions appears likely, however. More likely, LISP was chosen because it held the best promise for the rapid development of a program, for quick experimentation with ideas, and for a final product that achieved a reasonable level of performance. Undoubtedly, any of the programs could be recoded in some other language (or even LISP) and result in a new program that is smaller in memory and runs faster. But, when the human cost of original programming and experimentation is included, very often LISP appears as the best initial choice.

In spite of our liking for LISP, we are looking forward to the ultimate language that will be easy to

read and write, in which it will be natural to express oneself, and which will come packaged in a helpful environment. LISP is not this ultimate language, but it is certainly nice to have around.

REFERENCE NOTES

1. Davis, R., Buchanan, B., & Shortliffe, E. *Production rules as a representation for a knowledge-based consultation program*. Report AIM-266, Stanford University, 1975.

2. Woods, W. A., & Kaplan, R. M. *The lunar sciences natural language information system*. Cambridge Mass: Bolt, Beranek & Newman, Inc., BBN Report 2265.

REFERENCES

- BOGEN, R. *MACSYMA reference manual*. Cambridge, Mass: MIT Press, 1975.
- BROWN, J. S., & BURTON, R. Multiple representation of knowledge for tutorial reasoning. In D. Bobrow & A. Collins (Eds.), *Representation and understanding*. New York: Academic Press, 1975.
- BUCHANAN, B., SUTHERLAND, G., & FEIGENBAUM, E. A. Heuristic DENDRAL: A program for generating explanatory hypotheses in organic chemistry. In B. Meltzer & D. Michie (Eds.), *Machine intelligence 4*. New York: American Elsevier, 1969.
- COLBY, K. M., WEBER, S., & HILF, F. D. Artificial paranoia. *Artificial Intelligence*, 1971, 2, 1-25.
- LENAT, D. B. Automated theory formation in mathematics. *Proceedings of the 5th International Joint Conference on Artificial Intelligence*. Cambridge, Mass: MIT Press, 1977.
- POPLE, H. The formation of composite hypotheses in diagnostic problem solving: An exercise in synthetic reasoning. *Proceedings of the 5th International Joint Conference on Artificial Intelligence*. Cambridge, Mass: MIT Press, 1977.
- SIKLÓSSY, L. *Let's talk LISP*. Englewood Cliffs, N.J: Prentice-Hall, 1976.
- SIKLÓSSY, L. Impertinent question answering systems: Justification and theory. *Proceedings of the ACM National Conference*, 1978.
- SIKLÓSSY, L., & DREUSSI, J. An efficient robot planner which generates its own procedures. *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*. Stanford, Calif: Stanford University Press, 1973.