Computer Recreations

by

PAUL BRATLEY AND JEAN MILLO

Département d'Informatique, University of Montreal, Montreal, Canada

Self-Reproducing Programs*†

It has been known for some years that certain systems are capable of creating independent copies of themselves, given a suitable supply of raw material in their environment. For instance, von Neumann[1] introduced the notion of a cellular space and envisaged an automaton embedded in the space which is capable of building a copy of itself and then setting the copy to work independently. Further work on these lines has been reported by Codd,[2] Myhill[3] and others. Besides these abstract automata, real mechanical systems which are capable of self-reproduction have been built, for instance by Penrose.[4]

Since there is an obvious relationship between the programs which can be written in a given programming language on the one hand, and the algorithms which can be implemented by a particular class of abstract automata on the other, the question naturally arises: In what programming languages can self-reproducing programs be written? We first make some attempt to define what we mean by 'self-reproducing' in this context and then exhibit some programs which have the desired property. In particular, we show that self-reproducing programs can be written in SNOBOL, LISP, FORTRAN and ALGOL 60.

No formal definition of self-reproducibility will be attempted, but we lay down the following ground rules.

(1) A self-reproducing program must produce a copy of itself which is in some sense exact. For instance, in the examples which follow, the SNOBOL program actually punches a deck of cards identical to the source program deck; the LISP program prints a list with the same structure as itself; and the FORTRAN program writes on to tape card images identical to the source deck. No brackets or string quotes are lost; if the original program necessarily contains continuation lines then so does the copy and so forth.

(2) No data files may be read or written by the program (with the exception, perhaps, of a file on which the copy is produced).

(3) The program must not depend essentially on a knowledge of the internal representation of any particular character, numerical value or other data object, nor must it make any special assumptions about the compiler which is to be used.

Certain trivial cases can be dismissed with little ado. Firstly, it seems extremely likely that self-reproducing programs can be written in the assembly language of any normal digital

* This work was supported in part by a grant from the National Research Council of Canada.

† $\aleph_0$ is delighted to print this article in place of his own regular contribution.

Received 21 June 1972

computer. (We except, for instance, curious devices fit only for missile guidance systems.) A little care is needed to avoid violating rule (3) above, but that is all.

Secondly, it is easy to invent programming languages in which self-reproduction is possible. For instance, let L be the language with the following definition:

(a) A program in L is any string of characters followed by an end-of-file mark.

(b) The effect of any program in L at execution time is for each character in the string to generate a copy of itself on the output medium.

Then clearly any program in L is a self-reproducing program: self-reproduction is not merely possible, but indeed inevitable. (L, trivial though it is, provokes one or two amusing questions. For instance, most computer systems possess an L-interpreter, in the shape of a utility 'copy' routine. Would it ever be worth while providing an L-compiler for strings of text which will be reproduced frequently?)

Our original question is therefore only interesting if we restrict our attention to widely known and well standardized programming languages.

Four examples of self-reproducing programs are given as Figures 1–4. The first three programs have actually been run on the University of Montreal's CDC 6400, using standard compilers available on the system. The fourth program, in ALGOL 60, has not been tested on a machine, since on the one hand the representation of ALGOL used by the 6400's compiler differs considerably from the reference language and, on the other, the I/O procedures offered by our system are not those used in the example.

The simplest example, Figure 1, is a SNOBOL4 program which writes its own text on to the card punch. The program obeys the rules for writing SNOBOL programs presented in Reference 5. Once it has been pointed out that the variable A contains the text of line 4, while B contains the text of line 5, no further explanation should be required.

```
X = '"'; Y = "'"
A = "PUNCH = 'X = 'Y X Y'; Y = 'X Y X; PUNCH = 'A = 'X A X"
B = "PUNCH = 'B = 'X B X; PUNCH = A; PUNCH = B;END"
PUNCH = 'X = 'Y X Y'; Y = 'X Y X; PUNCH = 'A = 'X A X
PUNCH = 'B = 'X B X; PUNCH = A; PUNCH = B;END
```

Figure 1. Self-reproducing SNOBOL program

Next, Figure 2 is a self-reproducing LISP[6] program. It consists essentially of the definition and call of a function C which, among other things, uses GET to print out its own definition.

```
DEFINE ((
(C (PROG (A)
(PRINT (QUOTE DEFINE))
(PRINT (LIST(LIST(LIST(QUOTE C) (GET(QUOTE C) (QUOTE EXPR))))))
(PRINT (QUOTE C))
(PRINT (LIST A))
(PRINT (QUOTE STOP))
(PRINT (QUOTE FIN))))))
C (NIL)
STOP
FIN
```

Figure 2. Self-reproducing LISP program

Figure 3 is a FORTRAN program which writes its own text as card images on unit 99. Except for the first line, which is necessary if the program is to run on the CDC 6400, the program is in ANSI FORTRAN as specified in Reference 7. It was convenient to make use of the knowledge that one machine word would hold ten Hollerith characters, but the program does not depend crucially upon this fact: the spirit of our rule (3) is therefore observed. The structure of the program is obvious.

```
 PROGRAM A (TAPE99)
 DIMENSION M(64)
 DATA (M(I), I = 1, 64)/
110H         WRIT,10HE (99, 1)M  ,10H               ,10H          ,
110H             ,10H            ,10H               ,10H          ,
110H     1 FORM,10HAT(6X,18HP    ,10HROGRAM A (,10HTAPE99),/,,
110H6X,15HDIME,10HNSION M(64,10H),/,6X,             ,10H          ,
110H        119HD,10HATA (M(I),  ,10HI = 1, 64)/,/  ,10H,15(5X,1H1,
110H,4(3H10H,A   ,10H10,1H,),/)  ,10H,5X,1H1,       ,10H          ,
110H         13(3H,10H10H,A10,1H ,10H,),3H10H,A     ,10H10,1H/)   ,
110H             ,10H            ,10H               ,10H          ,
110H         WRIT,10HE (99, 2)M  ,10H               ,10H          ,
110H             ,10H            ,10H               ,10H          ,
110H     2 FORM,10HAT(8A10)      ,10H               ,10H          ,
110H             ,10H            ,10H               ,10H          ,
110H         STOP,10H            ,10H               ,10H          ,
110H             ,10H            ,10H               ,10H          ,
110H          END,10H            ,10H               ,10H          ,
110H             ,10H            ,10H               ,10H          /
 WRITE (99, 1)M
1 FORMAT(6X,18HPROGRAM A (TAPE99),/,6X,15HDIMENSION M(64),/,6X,
119HDATA (M(I), I = 1, 64)/,/,15(5X,1H1,4(3H10H,A10,1H,),/),5X,1H1,
13(3H10H,A10,1H,),3H10H,A10,1H/)
 WRITE (99, 2)M
2 FORMAT(8A10)
 STOP
 END
```

Figure 3. Self-reproducing FORTRAN program

Finally, Figure 4 is a program in ALGOL 60[8] which writes its own text on to channel 1. The input–output procedures used are those suggested in Reference 9. These procedures do not allow one to specify such details as line-feed and carriage-return, but on the other

```
begin  procedure p(a); string a;
       begin outstring (1, a);
       outsymbol (1, "", 1); outstring (1, a);
       outsymbol (1, "", 2); outstring (1, ') end')
       end;
p ('begin  procedure p(a); string a;
           begin outstring (1, a);
           outsymbol (1, "", 1); outstring (1, a);
           outsymbol (1, "", 2); outstring (1, ') end')
           end;
p(') end
```

Figure 4. Self-reproducing ALGOL program

hand they make no restrictions on the length of lines. The reader is therefore asked to imagine that the self-reproducing program is typed simply as one continuous sequence of ALGOL basic symbols, and that the output will be produced in the same form. It is interesting to note that we were unable to write a self-reproducing ALGOL program using only the input–output procedures suggested in Reference 10, which are essentially the ones offered by the CDC system. The stumbling block is the apparent impossibility of printing or punching a single closing string quote. An opening string quote can be printed by some such instruction as

```
output 1 (chan, 'S', '‘')
```

but this trick will not work in the other case. In the example we present, the procedure outsymbol is used to get round this difficulty.

The example in ALGOL probably brings out most clearly the structure of the three programs which reproduce their text on an external medium. If one ignores for a moment all considerations of line-length, spacing and so forth, then a 'general' self-reproducing program has the following structure:

```
string = 'print ('string = ')  print (open quotes)
          print (string)  print (close quotes)  print (string)'
print ('string = ')
print (open quotes)
print (string)
print (close quotes)
print (string)
```

Once this has been noticed, then the details of any particular representation or language can often be filled in easily.

ACKNOWLEDGEMENTS

The authors would like to acknowledge the help given them by Hamish Dewar, of Edinburgh University, whose self-reproducing program in IMP encouraged attempts to find such programs in other languages, and by their colleague Olivier Lecarme, who wrote a self-reproducing program for the CDC 6400 ALGOL compiler, and also a self-reproducing program in PL/1. Indeed, if this article has been limited to four examples, it was not because others were not forthcoming. On the contrary, there is something about this particular problem which excites keen programmers, so that research in self-reproducing program design is now under way for practically every compiler on our system.

REFERENCES

1. J. von Neumann, *Theory of Self-reproducing Automata* (Ed. A. W. Burks), University of Illinois Press, Urbana, 1966.
2. E. F. Codd, *Cellular Automata*, Academic Press, New York, 1968.
3. J. Myhill, 'The abstract theory of self-reproduction', in *Views on General Systems Theory* (Ed. M. D. Mesarovic), Wiley, New York, 1964, Chap. 7.
4. L. S. Penrose, 'Self-reproducing machines', *Sci. Am.* **200**, 105–114 (1959).
5. R. E. Griswold, J. F. Poage and I. P. Polonsky, *The SNOBOL 4 Programming Language*, Prentice-Hall, Englewood Cliffs, N.J., 1968.
6. J. McCarthy (Chairman), *LISP 1.5 Programmer's Manual*, M.I.T. Press, Cambridge, Massachusetts, 1968.
7. 'FORTRAN vs basic FORTRAN', *Comm. ACM*, **7**, 590–625 (1964).
8. P. Naur (Chairman), 'Revised report on the algorithmic language ALGOL 60', *Comm. ACM*, **6**, 1–17 (1963).
9. 'Report on input–output procedures for ALGOL 60', IFIP/WG 2.1, *Comm. ACM*, **7**, 628–629 (1964).
10. D. E. Knuth (Chairman), 'A proposal for input–output conventions in ALGOL 60', *Comm. ACM*, **7**, 273–283 (1964).