Programming With(out) the GOTO

B.M. Leavenworth, IBM

A brief history of the goto controversy (retention or deletion of the goto statement) is presented. After considering some of the theoretical and practical aspects of the problem, a summary of arguments both for and against the goto is given.

KEY WORDS AND PHRASES: goto statement, computability theory, goto-less programming, combinatory logic, lambda calculus, Post systems, Markov algorithms, Turing machines, structured programming, control structures.
CR CATEGORIES: 1.3, 4.2, 5.2

INTRODUCTION

There are a number of issues connected with the retention or deletion of the goto statement in programs or programming languages, and we attempt to set the stage for a discussion of these issues by giving a brief history of the goto controversy. The possibility of eliminating the goto has both theoretical and practical aspects. It is of interest to discover that the goto does not appear in most formal systems of computability theory, but does appear in programming language extensions of these systems. Since programming style is an important component of the controversy, we give one example of the influence of a high level language on programming style, and its relation to the goto statement. Finally, a summary of arguments both for and against the goto is given.

HISTORY

The proposition that there might be something wrong with the goto statement, one of the pillars of practical programming since the invention of FORTRAN, has slowly penetrated the consciousness of programmers since Dijkstra's famous letter in the Communications of the ACM (D4). Actually, Professor Dijkstra considered programming languages without benefit of either assignment or goto in a paper presented at the 1965 IFIP Congress (D2). He concluded that a language without the former was elegant but inadequate. As to the latter, he enunciated the criterion that the quality of a programmer was inversely proportional to the density of goto statements in his program. While admitting the possibility of a conflict between convenience and efficiency, he made the following points, which are paraphrased below.

1. Since transfer of control is subsumed by more powerful notions, to wit:
   sequential execution
   procedure call and return
   conditional expression (statement)
   repetition clause (for statement in ALGOL, DO group in PL/I)
   is not the programmer led astray by giving him control over it?

2. The solution of the halting problem (the determination of whether a given program terminates) is made difficult by the unrestricted use of the goto statement. After elimination of the goto, there are only two ways in which a program may fail to stop: either by infinite recursion, or by the repetition clause.

Val Schorre reported in 1966 (S1) on the development of two procedural languages, LISPX and MOL-32, without the goto. Further, that he had been writing programs since 1960 in outline form using the principle of nested flow. These outlines, which served the purpose of flow charts, showed the flow of control graphically by indentation, and were used as original documentation of the program, which was coded in assembly language from the outline. This may be the first recorded instance of "goto-less" procedural programming (as distinct from functional programming in LISP), albeit not in a high level language.

Professor Van Wijngaarden (V1) showed that the goto statement could, in principle, be eliminated from ALGOL 60 programs by a preprocessing algorithm which replaced the set of given programming constructs by a smaller set of equivalent concepts. The purpose of this demonstration was the explication of syntax and semantics, rather than "goto-less" programming.

Peter Landin also argued in 1966 (L3) for a style of programming which eliminates not only the goto, but the notion of explicit sequencing and assignment as well. Landin introduced a language called ISWIM, and used a purely functional subset of this language to program in this style. It should be noted, however, that ISWIM contains imperative features such as "program points", roughly analogous to labels, and assignment so that the programmer has an out. We shall see that this theme constantly recurs in what follows.

COMPUTABILITY THEORY WITH[OUT] THE GOTO

Although the formal systems of computability theory (see below) have for the most part theoretical rather than practical significance, they demonstrate that the goto is not needed as a primitive in order to compute all computable functions. It is interesting to discover, however, that the goto has been included in pragmatic extensions to these systems.

The goto does not appear in the following formal systems:

Formal System	Programming Extension
combinatory logic of Curry & Feys (C11), and the lambda calculus of Church (C4)	a jumping operator 'J' was defined by Landin (L3) in ISWIM, which is an extension of the lambda calculus
Post systems (P5) and Markov algorithms (M1)	labelled Markov algorithms with branching were defined by Caracciolo et al (C1), and the language COMIT (Y1) and SNOBOL (F1) can be considered to be extensions of Markov algorithms with goto commands
Kleene general recursive functions (K1)	LISP (M2), which was also strongly influenced by the lambda calculus, has the PROG feature which allows assignment and goto (see below)

We see in each case that the goto is missing in the pristine form of the system, but has been added in programming extensions (whether for the sake of tradition or "convenience" is a matter for debate). The PROG feature (B5) was added to LISP in order to incorporate the goto, among other things, although a wide range of applications have been written in pure (no assignment or goto) LISP.

The goto appears in Turing machines (T1) (since instructions or states have explicit successors), and related automata such as Minsky's program machines (M6). It also appears in program schemata (L7), which can be characterized as flow charts with assignment statements in the boxes. And finally, it appears in the order codes of the general purpose computer.

THE INFLUENCE OF NOTATION

It may be truly said that the goto statement in its form as a machine primitive has profoundly influenced the long line of procedural high level language descendants. We wish to explore this point and its relation to what shall be called the FORTRAN II IF Syndrome.

The FORTRAN II IF statement --- IF (expr) n1, n2, n3 --- is a prime example of the power of language to influence program organization, and probably corrupted a generation of programmers. This statement effectively generates multiple gotos (which reflect the unconditional transfers in machine code), as can be seen by the equivalent PL/I statements:

```
IF expr < 0 THEN GO TO n1;
IF expr = 0 THEN GO TO n2;
IF expr > 0 THEN GO TO n3;
```

The sad fact is that many programmers, even after being liberated by compound and conditional statements in ALGOL and PL/I so that they could write

```
IF expr THEN DO; ...; END;
```

continued to write

```
IF expr THEN GO TO LAB;
```

from force of habit. Thus we see the influence that machine primitives have exerted through the present evolution of high level programming languages!

SUMMARY OF ISSUES

Since the theoretical possibility of eliminating the goto has been demonstrated, it will not be discussed further. We will therefore attempt to summarize the practical arguments both for and against the goto. The arguments for eliminating the goto (at the same time, replacing it by other control structures) are essentially the following:

1. Goto-less programs are easier to understand, debug and modify. This is the structured or top-down programming argument (D2) (D4) (D6) (M5) (W7) (W8) (W9).

2. If the goto statement is not replaced by more sophisticated control structures, the programmer is likely to misuse it (the goto) in order to synthesize those structures (D4) (W9).

3. It is easier to prove assertions about "goto-less" programs (L3) (P3) (S3).

The technical means of replacing gotos by other control structures are as follows:

1. by recursive procedures. This is a theoretical, rather than a practical, device (V1) (K2).

2. by the while construction. This can always be done without changing the program topology, by the introduction of auxiliary variables (A1).

3. by node splitting. This requires redundant code or procedure calls (K2) (W7).

The arguments against eliminating the goto can be summarized as follows:

1. the goto is needed for abnormal exits from a block or procedure. The "repeat-exit" mechanism of Knuth and Floyd (K2) only allows a one-level exit, whereas Wulf's leave construction (W7) requires the reintroduction of labels for multi-level exits. As Landin (L3) has admitted, "the most recalcitrant uses of explicit sequencing appear to be associated with success/failure situations and the action needed on failure."

2. the goto is often more efficient. For, consider the overhead introduced by node splitting and the while construction (setting of flags). Also, Knuth and Floyd (K2) have pointed out that procedure calls can sometimes be replaced by goto statements.

3. the goto is useful for synthesis purposes (W2) (H2). Two examples: the RETURN statement can be synthesized by goto, and the case statement of Wirth and Hoare (W3) can be synthesized in a language, say PL/I, which lacks it.

ACKNOWLEDGEMENT

Thanks are due to Jean Sammet who suggested the writing of this paper, and who provided valuable comments for improving several early versions.

REFERENCES & BIBLIOGRAPHY

A1. Ashcroft, Edward and Manna, Zohar. "The translation of 'goto' programs to 'while' programs". Proc. IFIP Congress 71, Ljubljana, Aug. 1971.

B1. de Bakker, J. W. "Semantics of programming languages". Advances in Information Systems Science 2 (Ed. Tou, J.T.) Plenum Press, New York, 1969.

B2. Barron, D.W. Recursive Techniques in Programming. American Elsevier, New York, 1968.

B3. Barron, D.W. and Strachey, C. "Programming". Advances in Programming and Non-Numerical Computation. (Ed. Fox, L.), Pergamon Press, New York, 1966.

B4. Berry, D. M. "Introduction to Oregano". Proc. Symposium on Data Structures in Programming Languages, SIGPLAN Notices 6,2 (Feb. 1971).

B5. Black, Fischer. "Styles of programming in LISP" The Programming Language LISP: Its Operation and Applications (Ed. Berkeley and Bobrow), Information International, Cambridge, Mass. 1964.

B6. Bohm, Corrado and Jacopini, Giuseppe. "Flow diagrams, Turing machines and languages with only two formation rules". CACM 9 (May 1966).

B7. Burge, W.H. "The evaluation, classification and interpretation of expressions". Proc. ACM 19th National Conf. 1964.

B8. Burge, W.H. "Notes on a model for programming systems: Part I". Report RC 2188 (Aug. 1968). IBM Research Division, Yorktown Heights, N.Y.

B9. Burstall, R.M. "Writing search algorithms in functional form" Machine Intelligence 3 (Ed. Michie, D.) Edinburgh Univ. Press, Edinburgh, 1968.

B10. Burstall, R.M. "Proving properties of programs by structural induction", Computer Journal 12,1 (Feb. 1969).

B11. Burstall, R.M. and Popplestone, R.J. "POP-2 reference manual" Machine Intelligence 2 (Ed. Dale & Michie), American Elsevier, New York 1968.

B12. Burstall, R.M. and Landin, P. J. "Programs and their proofs: an algebraic approach", Machine Intelligence 4 (Eds. Meltzer & Michie) Edinburgh Univ. Press, Edinburgh, 1969.

C1. Caracciolo di Forino, A., Spanedda, L. and Wolkenstein, N. "PANON-1B: A programming language for symbol manipulation", Calcolo, Vol. 3, 1966.

C2. Caracciolo di Forino, A. "Generalized Markov algorithms and automata", Automata Theory (Ed. Caianiello, E. R.), Academic Press, New York, 1966.

C3. Christenson, Carlos, "Examples of symbol manipulation in the AMBIT programming language". Proc. ACM 20th National Conf., Cleveland, Ohio, Aug. 1965.

C4. Church, A., "The calculi of lambda-conversion", Annals of Math. Studies No. 6, Princeton Univ. Press, Princeton, New Jersey (1951).

C5. Cohen, K. and Wegstein, J. H., "AXLE, an axiomatic language for string transformations", CACM 8, (1965), 657-661.

C6. Cooper, D. C. "On the equivalence of certain computations". Computer Journal 9 (1966), 45-52.

C7. Cooper, D. C. "Reduction of programs to a standard form by graph transformation", Theory of Graphs, International Symposium, Rome 1966 (Ed. Rosenstiehl, P.), Gordon and Breach, New York, 1967.

C8. Cooper, D. C. "Bohm and Jacopini's reduction of flow charts". Letter to the Editor, CACM 10 (Aug. 1967).

C9. Cooper, D. C. "Some transformations and standard forms of graphs, with applications to computer programs", Machine Intelligence 2 (Ed. Dale & Michie), American Elsevier, New York, 1968.

C10. Coulouris, G. F. "Principles for implementing useful subsets of advanced programming languages", Machine Intelligence 1 (Ed. Collins & Michie), Oliver & Boyd, Edinburgh, 1967.

C11. Curry, H. and Feys, R. Combinatory Logic, Vol. 1, North-Holland, Amsterdam, 1958.

D1. Dijkstra, E. W., "An attempt to modify the constituent concepts of serial program execution", Proc. ICC Symposium on Symbolic Languages in Data Processing, Gordon & Breach, New York, 1962.

D2. Dijkstra, E. W. "Programming considered as a human activity", Proceedings IFIP Congress 65,65, edited by W. A. Kalenich, Spartan Books, Washington, D. C., 1965.

D3. Dijkstra, E. W. "Recursive programming", Programming Systems and Languages (Ed. Rosen, S.), McGraw-Hill, New York 1967.

D4. Dijkstra, E. W. "Go to statement considered harmful", Letter to the Editor, CACM 11 (March 1968).

D5. Dijkstra, E. W. "A constructive approach to the problem of program correctness", BIT 8 (1968).

D6. Dijkstra, E. W. "Notes on structured programming", EWD 249, Technical University, Eindhoven, Netherlands, 1969.

E1. Ershov, A. P. "Theory of program schemata", Proc. IFIP Congress 71, Ljubljana, Aug. 1971.

F1. Farber, D. J., Griswold, R. E. and Polonsky, I.P. "SNOBOL, a string manipulation language", JACM 11 (January 1964).

F2. Fisher, David A. "Control structures for programming languages", PhD. Thesis, Carnegie-Mellon Univ., Pittsburgh, Pa., May 1970.

F3. Floyd, R. W. "A descriptive language for symbol manipulation", JACM 8,4 (1961).

F4. Floyd, R. W. "Nondeterministic algorithms", JACM 14 (Oct. 1967).

F5. Floyd, R. W. "Assigning meanings to programs", Proc. Symp. Applied Math., AMS Vol. 19, 1967.

G1. Galler, B. A. and Fischer, M.J. "The iteration element", CACM 8 (June 1965).

G2. Galler, B. A. and Perlis, A. J. A View of Programming Languages, Addison-Wesley, Reading, Mass., 1970.

G3. Gilmore, P.C. "An abstract computer with LISP-like machine language without a label operator", Computer Programming and Formal Systems (Eds. Braffort & Hirschberg), North-Holland, Amsterdam, 1963.

G4. Goodstein, R. L. Recursive Analysis, North-Holland, Amsterdam, 1961.

G5. Griswold, R. E. Poage, J. F. and Polonsky, I. P. The SNOBOL4 Programming Language, Prentice-Hall, Englewood Cliffs, N.J. 1968.

G6. Guzman, Adolfo and McIntosh, H. "CONVERT", CACM 9 (Aug. 1966).

H1. Hopkins, Martin, "A case for the goto", Proceedings ACM '72, Boston, August 1972.

I1. Ianov, Y. I. "On the equivalence and transformation of program schemes", CACM 1 (1958), 8-12.

I2. Ianov, I. "The logical schemes of algorithms", Problems of Cybernetics I (English translation) Pergamon Press, Oxford 1960, 82-140.

J1. Johansen, Peter, "Non-deterministic programming", BIT 7 (1967), 289-304.

J2. Johnston, John B. "The contour model of block structured processes", Proc. Symposium on Data Structures in Programming Languages, SIGPLAN Notices 6,2 (Feb. 1971).

K1. Kleene, S. C. Introduction to Metamathematics, Van Nostrand, New York, 1952.

K2. Knuth, D. E. and Floyd, R. W. "Notes on avoiding 'goto' statements", Information Processing Letters 1, North-Holland, Amsterdam (1971), 23-31.

L1. Landin, P. J. "The mechanical evaluation of expressions", Computer Journal 6,4 (1964).

L2. Landin, P. J. "A correspondence between ALGOL 60 and Church's lambda-notation", CACM 8,2 and 3 (1965).

L3. Landin, P. J. "The next 700 programming languages", CACM 9 (March 1966).

L4. Leavenworth, B. M. "The definition of control structures in MCG360". Report RC2376 (Feb. 1969). IBM Research Division, Yorktown Heights, N.Y.

L5. Ledgard, H. F. "Ten mini-languages: A study of topical issues in programming languages", ACM Computing Surveys, 3,3 (Sept. 1971).

L6. Lucas, P. et al "Method and notation for the formal definition of programming languages", Tech. Report TR 25.087, IBM Laboratory, Vienna, 1968.

L7. Luckham, D. C., Park, D.M.R. and Paterson, M.S., "On formalized computer programs", Journal of Computer and System Sciences, June 1970.

M1. Markov, A.A. "The theory of algorithms" (Russian Translation), U.S. Dept. of Commerce, Office of Technical Services No. OTS 60-51085.

M2. McCarthy, J. et al, LISP 1.5 Programmers Manual, The M.I.T. Press, Cambridge, Mass. 1962.

M3. McCarthy, J. "Towards a mathematical science of computation", Proc. IFIP Congress, Munich 1962, North-Holland, Amsterdam.

M4. McCarthy, J. "Basis for a mathematical theory of computation", Computer Programming and Formal Systems (Eds. Braffort & Hirschberg), North-Holland, Amsterdam, 1963.

M5. Mills H. "Top down programming in large systems", Debugging Techniques in Large Systems (Ed. Rustin, Randall), Prentice-Hall, Englewood Cliffs, N.J. 1971.

M6. Minsky, M. L. Computation: Finite and Infinite Machines, Prentice-Hall, Englewood Cliffs, N.J. 1967.

M7. Mooers, C. N. and Deutsch, L.P. "TRAC: A text handling language", Proc. ACM 20th National Conf. Cleveland, Ohio (Aug. 1965).

N1. Naur, P. "Proof of algorithms by general snapshots", BIT 6, 1966.

N2. Naur, P. "Programming by action clusters", BIT 9, 1969.

P1. Paterson, M. S. "Program schemata", Machine Intelligence 3 (Ed. Michie, D.), Edinburgh Univ. Press, Edinburgh, 1968.

P2. Paterson, M.S. and Hewitt, C. E. "Comparative schematology", Proj. MAC Conference on Concurrent Systems and Parallel Computation (June 1970), ACM, New York, 1970.

P3. Perlis, A.J., Lecture Notes on Seminar on Extensible Languages. Carnegie-Mellon University, Fall, 1968.

P4. Peter, Rozsa. Recursive Functions, Academic Press, New York, 1967.

P5. Post, E. I. "Finite combinatory processes - formulation I", Journal of Symbolic Logic, Vol. 1, (1936).

R1. Reynolds, J.C. "GEDANKEN: A simple typeless language based on the principle of completeness and the reference concept", CACM 13 (May 1970).

R2. Rice, H. G. "Recursion and iteration", CACM 8 (Feb. 1965).

R3. Rice, J.R. "The goto statement reconsidered", Letter to the Editor, CACM 11 (1968) 538.

R4. Rutledge, J.D. "On Ianov's program schemata", JACM 11 (1964), 1-9.

S1. Schorre, D.V. "Improved organization for procedural languages", Technical Memo, August 1966, System Development Corp., Santa Monica, Calif.

S2. Shepherdson, J.C. and Sturgis, H.E. "Computability of recursive functions", JACM 10,2 (1963).

S3. Stark, R. "A language for algorithms", Computer Journal, Vol. 14, No. 1 (Feb. 1971).

S4. Strachey, C. "A general purpose macrogenerator", Computer Journal Vol. 8, (Oct. 1965).

S5. Strachey, C. "Fundamental concepts in programming languages", NATO Conf., Copenhagen 1967.

S6. Strong, H. R., Jr. "Translating recursion equations into flow charts", Journal of Computer and System Sciences, 5,3 (June 1971).

T1. Turing, A.M. "On computable numbers with an application to the Entscheidungsproblem", Proc. London Math. Soc., ser. 2, Vol. 42 (1936-1937).

V1. Van Wijngaarden, A. "Recursive definition of syntax and semantics", Formal Language Description Languages for Computer Programming, edited by T.B. Steel, Jr., North-Holland, Amsterdam, 1966.

W1. Wang, H. "A variant to Turing's theory of computing machines, JACM 4,1 (1957).

W2. Wegbreit, B. "Studies in extensible programming languages", ESD-TR-70-297, Directorate of Systems Design & Development, L. G. Hanscom Field, Bedford, Mass., May 1970.

W3. Wirth, Niklaus and Hoare, C.A.R. "A contribution to the development of ALGOL", CACM 9 (June 1966).

W4. Wirth, N. "On certain basic concepts of programming languages", Computer Science Technical Report No. CS65, Stanford University, 1967.

W5. Wirth, N. "Program development by stepwise refinement", CACM 14 (April 1971).

W6. Wozencraft, J. M. and Evans, A. Jr., "Notes on programming linguistics", Dept. of Electrical Engineering, MIT, Cambridge, Mass., Feb. 1971.

W7. Wulf, W. A. "Programming without the goto", Proc. IFIP Congress 71, Ljubljana, Aug. 1971.

W8. Wulf, W. A. Russell, D.B. and Habermann, A.N. "BLISS: a language for systems programming, CACM 14 (Dec. 1971).

W9. Wulf, W. A. "A case against the goto". Proceedings ACM '72, Boston, August 1972.

Y1. Yngve, V. H. Computer Programming with COMIT II, The M.I.T. Press, Cambridge, Mass. 1972.