1-1-1966

Non-deterministic algorithms

Robert W. Floyd
Carnegie Mellon University

NON-DETERMINISTIC ALGORITHMS

BY

ROBERT W. FLOYD

Carnegie Institute of Technology
Pittsburgh, Pennsylvania
November, 1966

This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-146).

NON-DETERMINISTIC ALGORITHMS

Robert W. Floyd

Department of Computer Science
Carnegie Institute of Technology

Non-deterministic algorithms are conceptual devices to simplify the design of backtracking algorithms [3] by allowing considerations of program bookkeeping required for backtracking to be ignored. Typically, a backtracking program solves some problem by exhaustive enumeration of a set of possible solutions. If at any point in the algorithm the tentative and partially specified solution is found to be inconsistent with the stated problem, the program "backtracks"; that is, it restores the values of all variables at the most recent time that it added to its partial specification of the solution, and tries the next alternative at that level of specification. When all alternatives at one level of specification have been tried, another alternative must be tried at the previous level of specification.

Non-deterministic algorithms resemble conventional algorithms as represented by flowcharts, programming languages, machine language programs, etc., except that:

(1) One may use a multiple-valued function, choice (X), whose values are the positive integers less than or equal to X. (Other multiple-valued functions may also be introduced, but this one is adequate.)

(2) All points of termination are labeled as successes or failures.

In general, there may be many ways to execute a non-deterministic algorithm, carrying out all assignments, branches, etc., in the conventional way, and making an arbitrary selection from the set of possible values each time a multiple-valued function is encountered. Of these execution sequences, however, only those whose terminations are labeled as successes are considered to be computations of the algorithm.

For example, consider an algorithm to solve the problem of the eight queens: to place eight queens on the chessboard so that no two are on the same row, column, or diagonal ([1], pg.165). Rephrasing, the problem is to find a sequence of eight numbers $(r_1, r_2, \ldots r_8)$, where $1 \leq r_i \leq 8$ represents the row occupied by the queen in the ith column of the board, such that if $i \neq j$, then $r_i \neq r_j$, $r_i + i \neq r_j + j$, and $r_i - i \neq r_j - j$.

We shall construct a non-deterministic algorithm to solve this problem. To record which rows and diagonals are occupied we shall use a_j to represent the number (always 0 or 1) of queens in the jth row, b_j to represent the number of queens on that diagonal for which the row index plus the column index equals j, and c_j to record the number of queens on that diagonal for which the row index minus the column index equals j. We shall specify the solution by letting col range over the column indices from 1 to 8, and for each column choosing a row between 1 and 8, using the choice function. A failure is registered as soon as any row or diagonal is occupied by two queens; a success is registered if

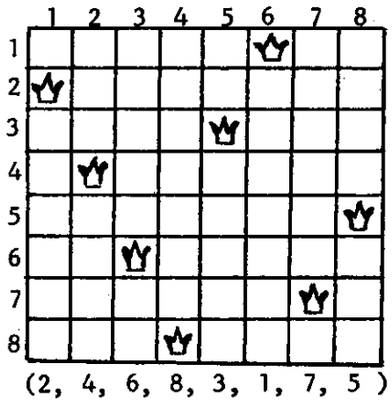


Figure 1

all eight choices are made without a failure. In flow chart form, the program is represented by Figure 2. The reader may verify that letting the value of choice(8) be successively 2, 4, 6, 8, 3, 1, 7, and 5 allows the algorithm to reach its successful termination. In fact, there is a one-for-one correspondence between computations of the algorithm and the ninety-two solutions of the problem.

The simplicity of specification of the algorithm is apparent. It corresponds relatively closely to a verbal algorithm such as "Pick one square in each column, being careful not to pick two on the same row or diagonal, and write down, for each column, the row you choose". There is, furthermore, a purely mechanical process of translation which may be used to convert a non-deterministic algorithm to a conventional deterministic one. Because this process is a local one, being applied to each command of the algorithm in turn, it is amenable to incorporation in compilers, macroassemblers, or simulators of machines. Each command is expanded into one or more commands, some of which carry out the effect of the original command in the non-deterministic algorithm, and additionally stack information required to reverse the effect of the command when backtracking is required, while others carry out the backtracking by undoing all the effects of the first set.

We add to the variables of the algorithm a new temporary variable T, and the three stacks M (memory), W (write), and R (read). Figure 3 shows for each command Σ of a non-deterministic algorithm the corresponding augmented command(s) Σ^+ of the deterministic algorithm,and the corresponding backtracking command Σ^- which undoes all the effects of Σ^+. In these commands, X is an arbitrary variable, f is an arbitrary expression, P is an arbitrary condition governing a conditional branch, and S is a subroutine, treated for simplicity as parameterless. All

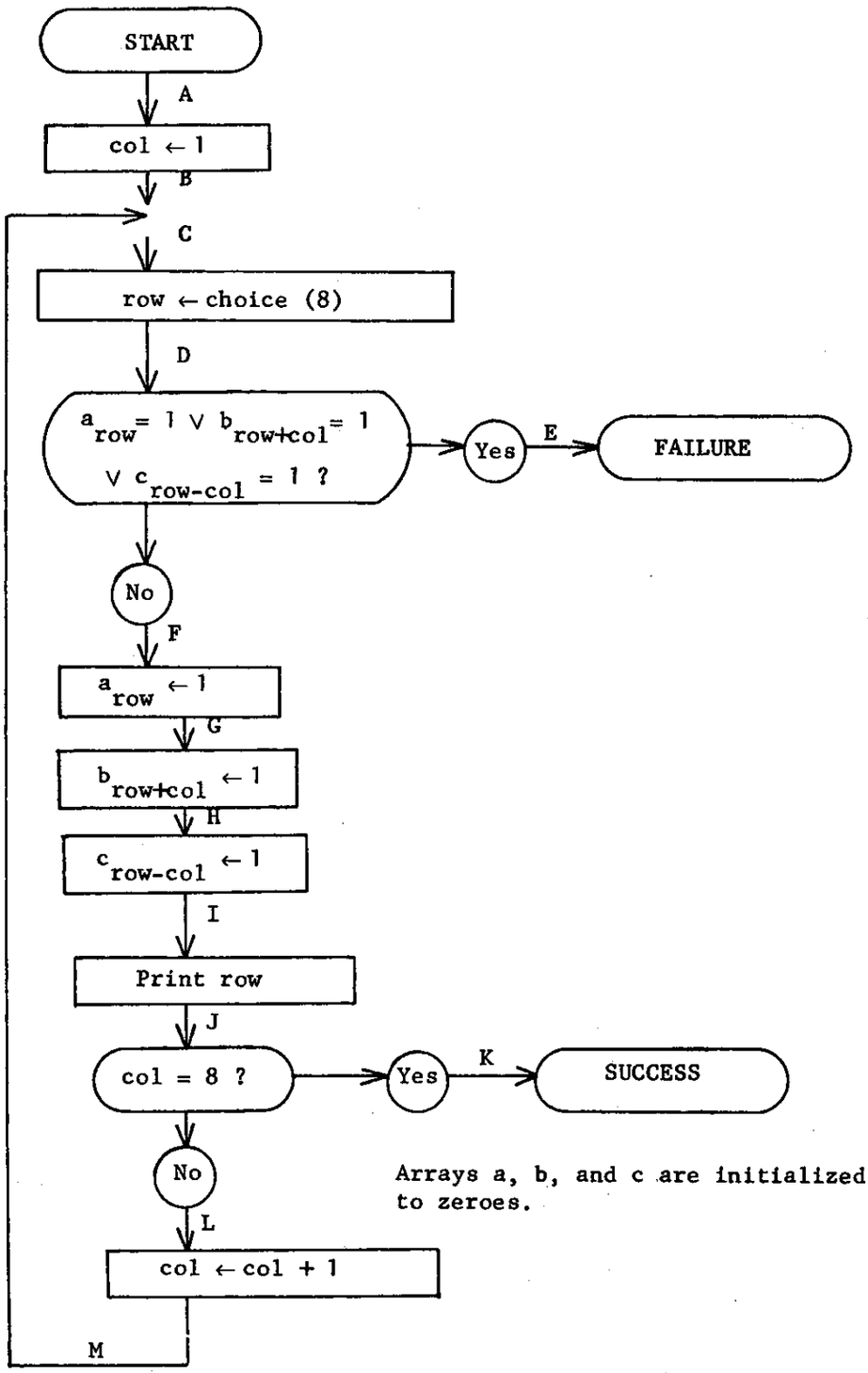


Figure 2

Non-deterministic Algorithm for the Eight Queens Problem

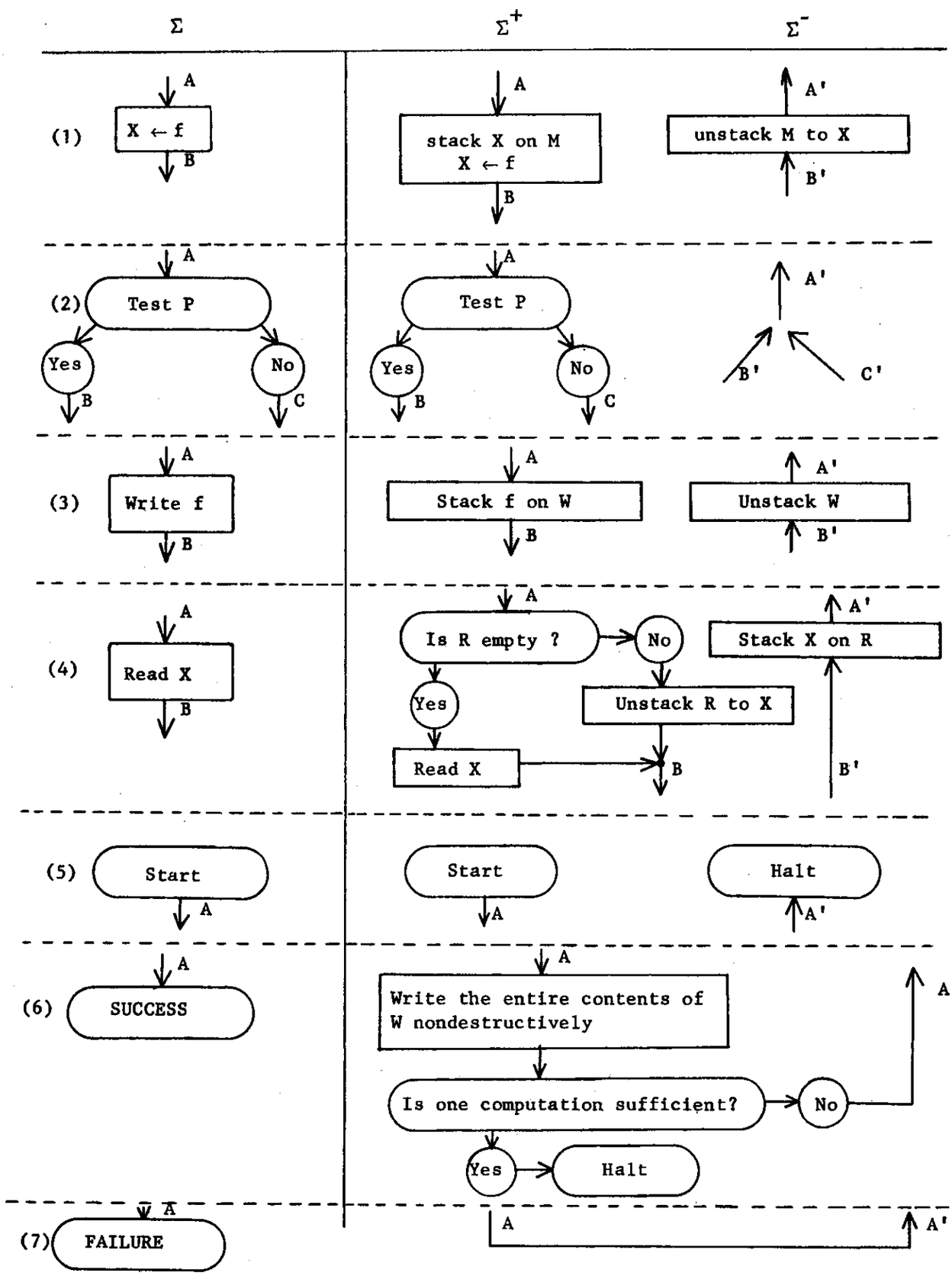


Figure 3 (Part 1)

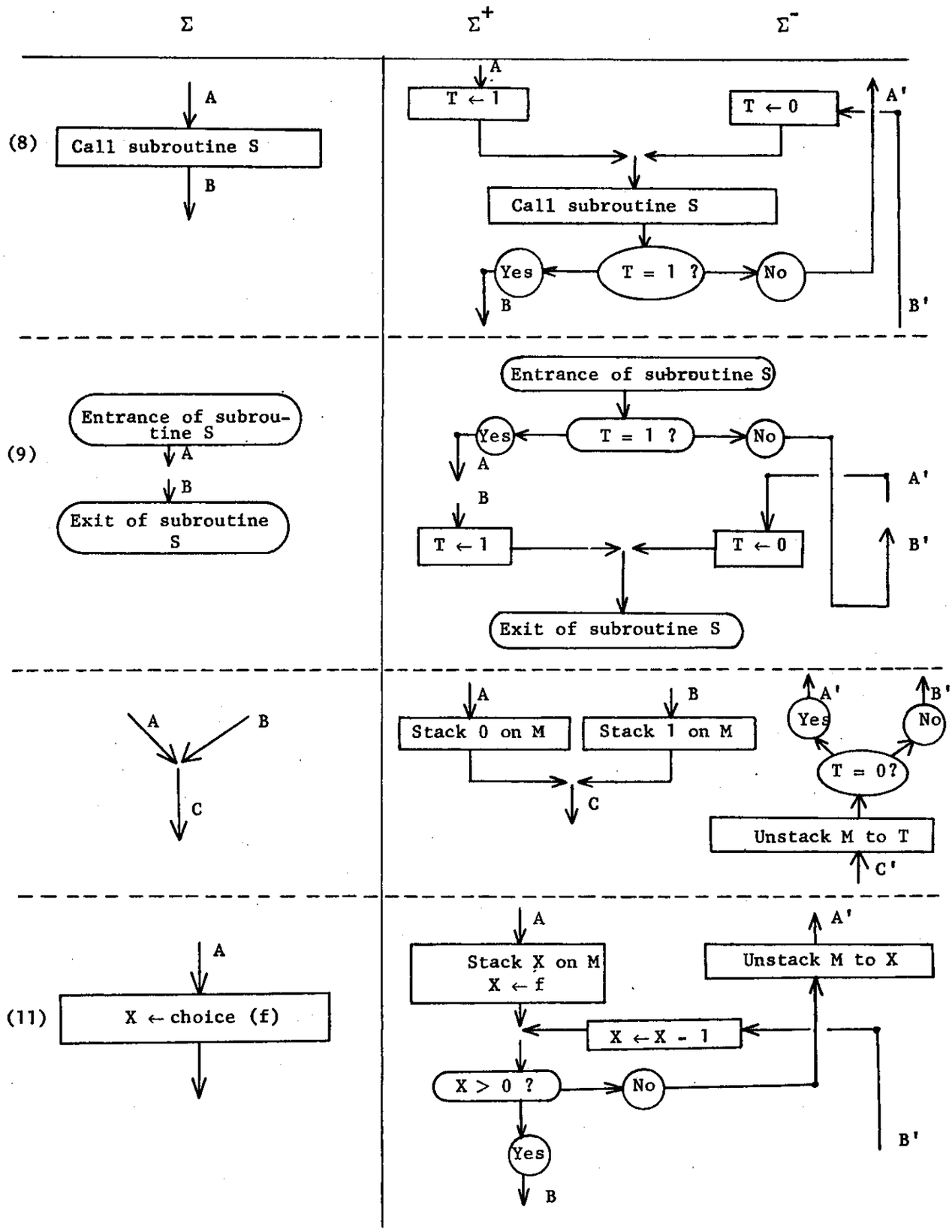


Figure 3 (Part 2)

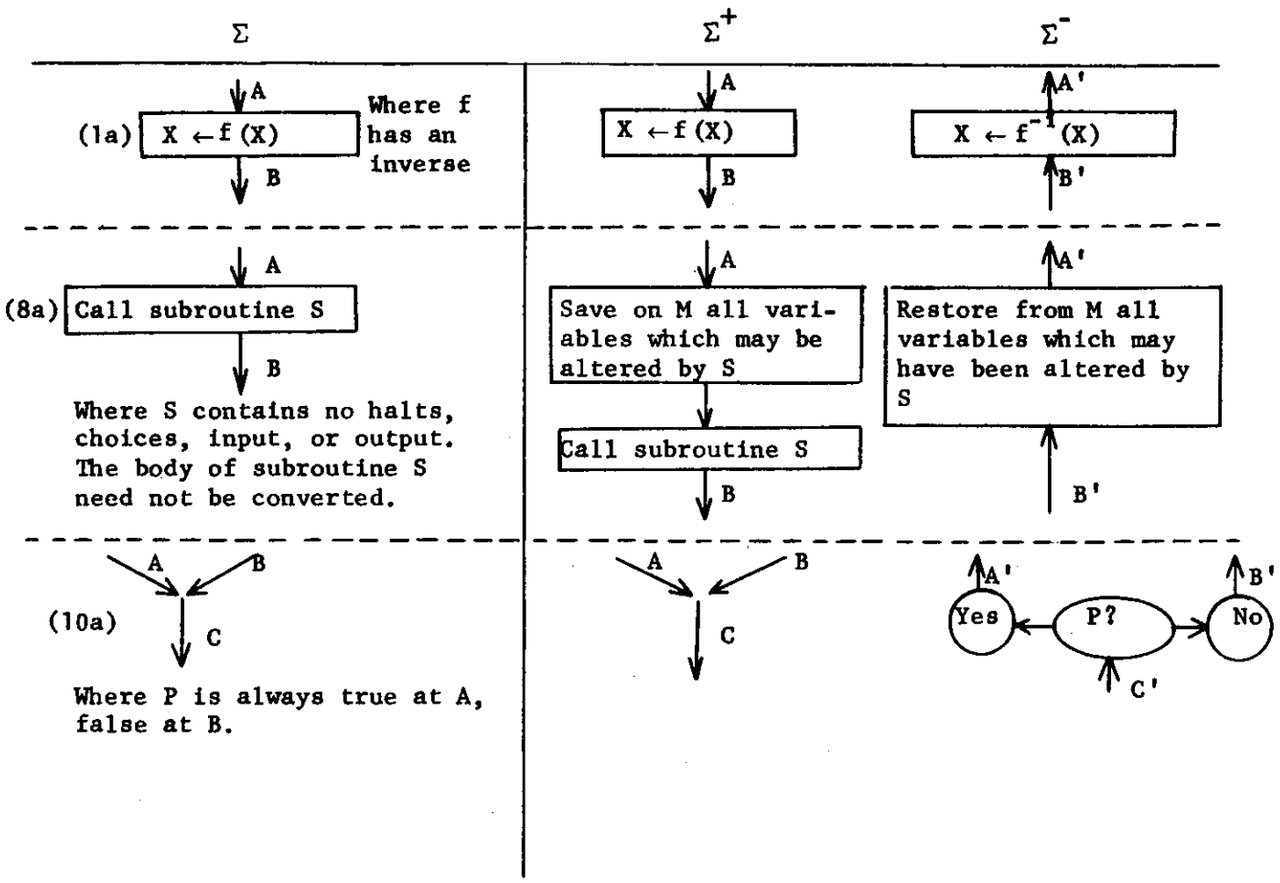


Figure 3 (Part 3)

flowchart connections are assumed to be labeled; this labeling is used as the basis for reconnection of the flowchart after conversion into deterministic form. Brief explanations follow:

(1) Before assigning a new value to X, the old value is stacked for restoration during backtracking.

(2) Because branching normally causes no loss of information, no special provisions for backtracking are required.

(3) All output is stacked, to be printed only if a successful termination is reached.

(4) Because most input devices are irreversible, a stack R, initially empty, is used to hold all input which has been backtracked over.

(5) Upon backtracking to the beginning of the non-deterministic algorithm, all possible solutions have been inspected, and the deterministic algorithm halts.

(6), (7) Upon reaching a success, all accumulated output is printed. If all solutions of the problem are desired, backtracking is initiated. A failure always initiates backtracking.

(8), (9) Subroutine calls and returns use a temporary storage cell, T, to indicate whether or not the program is in the backtracking state, thereby allowing free use of multiple-valued functions, and of points of termination, within subroutines. Recursive subroutines in non-deterministic algorithms are translated into recursive subroutines in deterministic algorithms.

(10) At a point where two paths of control join, one must preserve a record of which path was taken.

(11) One implementation of X← choice(f) saves the original value of X, and assigns f to X. After all possible computations with any particular value of choice (f) have been tried, the next smaller value is tried. When all values have been tried, the original value of X is restored and backtracking continues.

(1a) If an assignment command does not cause loss of information (e.g. X ← X + 1), no stacking is required; on backtracking the inverse command (e.g. X ←X-1) is executed. Stacking operations, and assignments which initialize previously undefined variables, may be treated similarly.

(8a) Conventional deterministic subroutines may be isolated from the conversion process, simply stacking the original values of any variables potentially altered by the subroutine.

(10a) More frequently than not, at the point where two paths of control join, the values of the program variables indicate which path was taken.

Applying these conversions to Figure 2, we construct Figure 4, a conventional deterministic algorithm for the eight queens problem. As is characteristic of such macro-expansion processes, there are minor inefficiencies, principally the duplication of the stack M by the stack W, but the algorithm appears to be reasonably satisfactory.

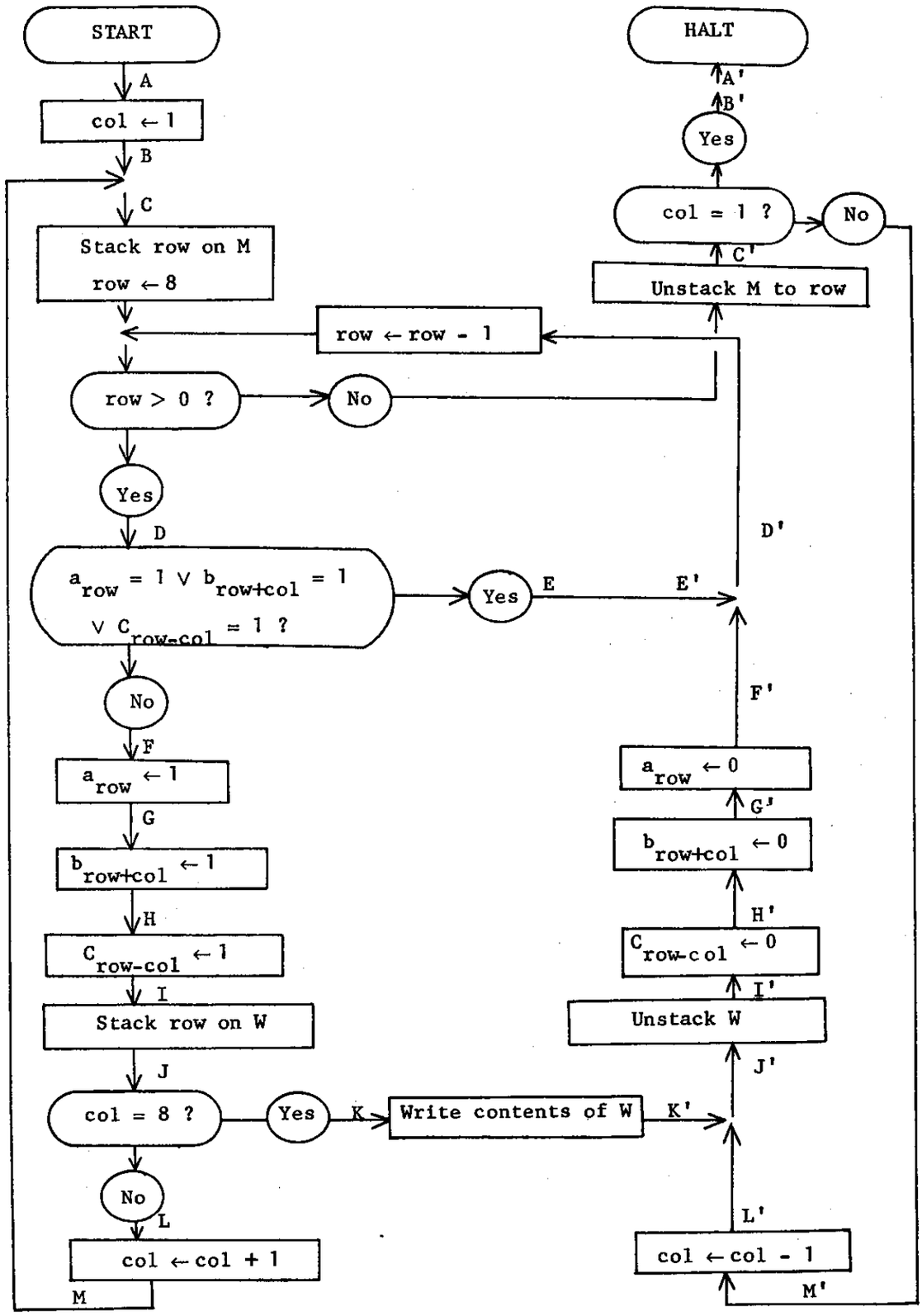


Figure 4

Deterministic Algorithm for the Eight Queens Problem

Another example of a non-deterministic algorithm enumerates all cycles in a network (loops in a flowchart, for example). Let us assume that the vertices of the network are named $X_1, X_2, \ldots, X_n$, as in Figure 5, and that step is an array of truth values such that $step_{i,j}$ is true if there is a direct connection from X_i to X_j in the network. A cycle is a sequence $(i_1, i_2, \ldots, i_k)$ of numbers between 1 and n such that $step_{i_j, i_{j+1}}$ is true for $1 \leq j < k$, $step_{i_k, i_1}$ is true, and if $\alpha \neq \beta$ then $i_\alpha \neq i_\beta$. We will obtain cycles in a canonical form such that i_1 is the largest number in the cycle. Figure 6 gives the cycles of Figure 5 in canonical form.

The algorithm of Figure 7 first selects i_1 (= initial), then repeatedly selects the value of i_{j+1} (= new) such that $step_{old,new}$ is true (i_j = old), and such that i_{j+1} is not equal to any of $i_2, i_3, \ldots, i_j$ ($used_{new}$ is false). Initially the vector $used_1, used_2, \ldots, used_n$ is assumed false. The process ends when $i_{j+1} = i_1$, at which time $i_1, i_2, \ldots, i_j$ have been printed. Figure 8 shows the array representation of Figure 5. Figure 7 corresponds to the verbal algorithm, "Pick an initial point for the cycle, then repeatedly pick a new one, of index no larger than the initial one, which has not been used before and which is directly accessible from the previous point. Continue until you return to the initial point. Write down all the points you pick, except the final repetition of the initial point."

One may frequently make a backtracking algorithm more selective in its search for solutions, with great gains in processing speed, by adding tests in the non-deterministic formulation of the algorithm, with one branch of each leading to a failure halt. For example, if we know for each X_i and X_j in a given network whether X_j can be reached from X_i (let the truth value of this be called $r_{i,j}$), we may

i \ j	1	2	3	4
1	F	T	T	F
2	T	F	F	T
3	F	T	F	F
4	T	F	T	F

Figure 8

Array $step_{i,j}$ Representing Figure 5.

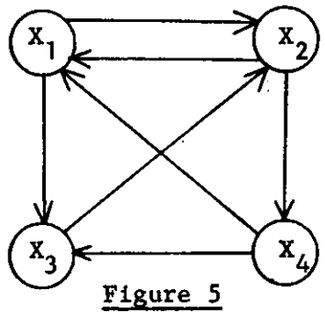


Figure 5

A Network.

(2,1)

(3,2,1)

(4,1,2)

(4,1,3,2)

(4,3,2)

Figure 6

Cycles of Figure 5.

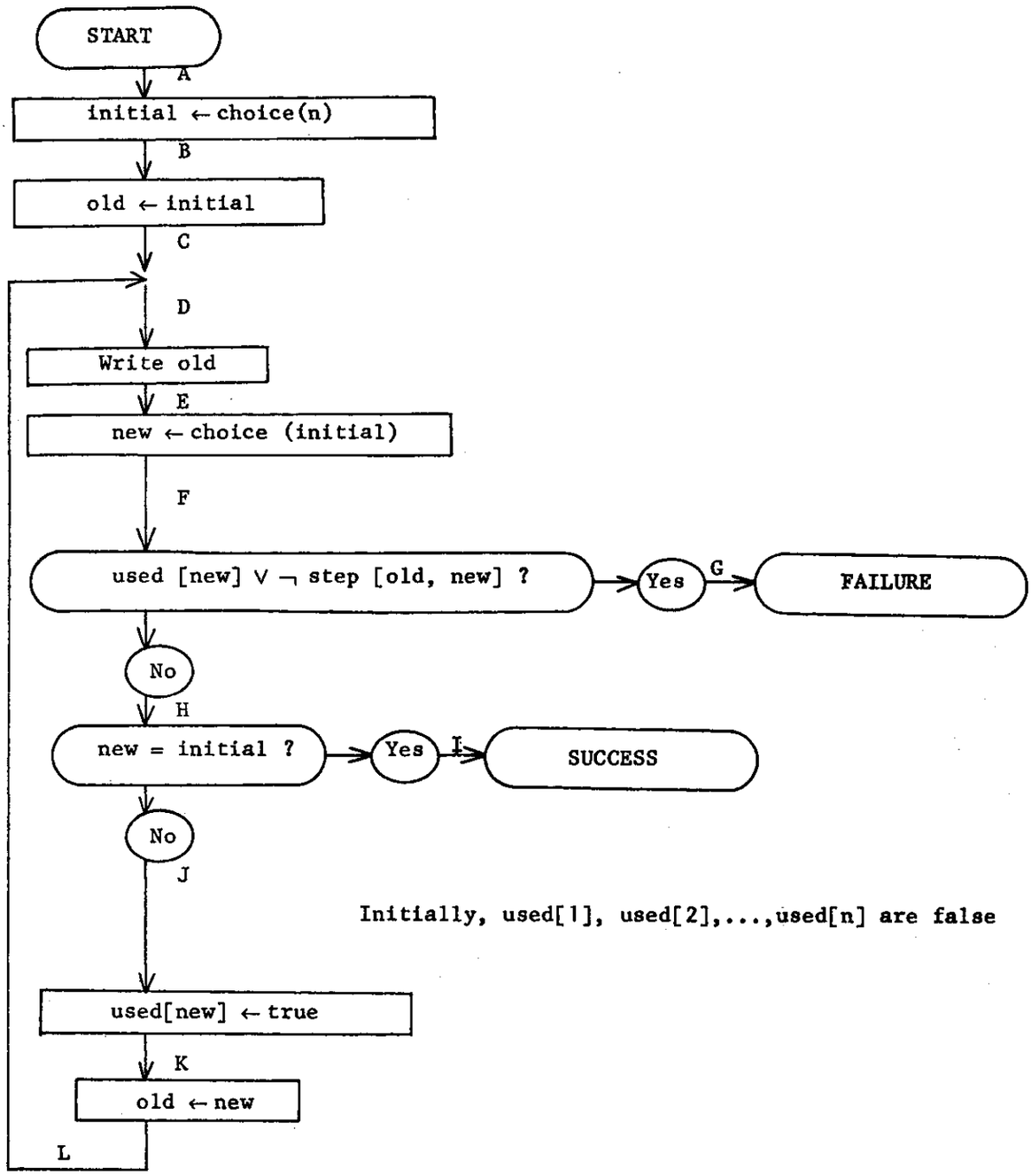
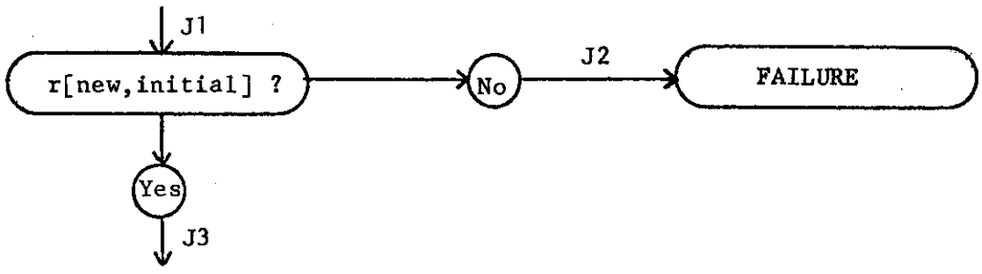


Figure 7

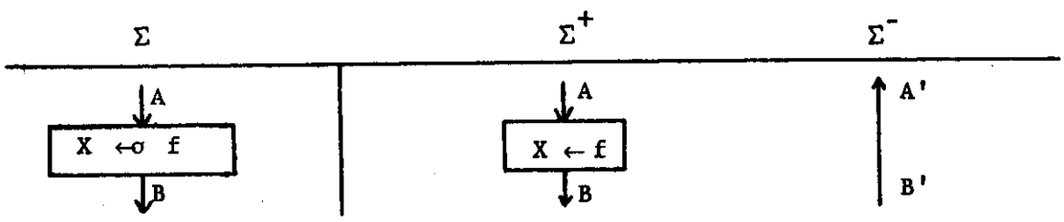
Non-deterministic Algorithm to Print any Cycle of a Network.

add at J in Figure 7 the following test:



In a network containing a rich collection of blind alleys and one-way streets, this test may greatly increase the efficiency of the generated algorithm. For other networks, of course, the test may simply consume time without eliminating any significant number of paths from consideration. One merit of the non-deterministic algorithm viewpoint is that such tests may be inserted and removed by local changes to the non-deterministic algorithm and its generated deterministic algorithm.

One might usefully extend the notion of the non-deterministic algorithm to allow some information to be carried over from one computation of the algorithm to another. This can be done by providing, for example, a form of assignment command whose effects are not reversed during backtracking. In programming a backtracking algorithm to find the solution of a given problem which achieves the minimum cost according to some measure, one may record by such an irreversible assignment the minimum cost of any solution yet found. As other solutions become partially specified, their partial costs are accumulated, and a failure may be programmed to occur when the implicit cost of a partially specified solution becomes greater than the minimum cost of the previous solutions. An irreversible assignment would be converted as follows:



Implementations of non-deterministic algorithm other than by the macro-expansion suggested by Figure 2 are possible. One may save the current values of all variables at each choice point, for example, unstacking the saved variables if a failure is reached and trying the next value for the choice. It is also possible to simulate a multi-processing machine which replicates itself at each choice point, pursuing all possibilities in parallel. In many typical applications, however, these approaches may be inefficient by comparison with macro-expansion.

Areas of application of backtracking are numerous. They include, but are certainly not limited to, syntactic analysis, [2, 4] economic resource allocation, cryptanalysis, design of efficient sorting procedures, and such applications in artificial intelligence as theorem proving and game playing. For use in such areas it is possible that programming languages capable of representing non-deterministic algorithms would be valuable, in the same way that simulation languages have proved valuable in certain areas of application. In both instances, a process with a very complicated control structure is represented by an algorithm with a simpler structure for an imaginary processor, and then converted to a more complicated algorithm for a conventional processor.

Because the word "non-deterministic" has a double meaning, it is perhaps desirable to make clear that non-deterministic algorithms are

not probablistic, random, or Monte Carlo algorithms. Rather, they are convenient representations of systematic search procedures. From one point of view, a non-deterministic algorithm represents a method of thinking of computer programs as being in part governed, not by efficient causes (causes which precede their effects) but by final causes (goals: causes for the sake of which their effects are carried out). Achievement of success and avoidance of failure is the goal of a non-deterministic algorithm, or, more precisely, of its imagined processor. One may say of the non-deterministic algorithm for the eight queens problem, for example, that when col = 1, row will never be chosen equal to 1 in any computation of the algorithm, because there are no solutions having a queen in the corner, and the goal of the processor is to find a solution. We may say that these algorithms are non-deterministic, not in the sense of being random, but in the sense of having free will.

References

[1] Ball, W. R., "Mathematical Recreations and Essays", Macmillan, New York, 12th edition, 1947.

[2] Floyd, R. W., "The Syntax of Programming Languages - A Survey". Institute of Electrical and Electronic Engineers Trans. on Electronic Computers EC-13, 4 (Aug.1964) 346-353.

[3] Golomb, S. W., and Baumert, L. D., "Backtrack Programming". Journal of the Association for Computing Machinery 12, 4 (Oct. 1965), 516-524.

[4] Irons, E. T., "A Syntax-Directed Compiler for ALGOL 60". Communications of the Association for Computing Machinery 4 (Jan. 1961) 51-55.