Go To Statement Considered Harmful
Edsger W. Dijkstra

Go To ステートメントは有害と考えられる
エドガー・W・ダイクストラ

Reprinted from Communications of the ACM, Vol. 11, No. 3, March 1968, pp. 147-148. 

Communications of the ACM, Vol.11, No.3, March 1968, pp.147-148 からの転載です。

Key Words and Phrases:
go to statement, jump instruction, branch instruction, conditional clause, alternative clause, repetitive clause, program intelligibility, program sequencing
CR Categories:
4.22, 6.23, 5.24
Editor:

キーワードとフレーズ
go to statement , jump instruction , branch instruction, conditional clause, alternative clause, repetitive clause, program intelligibility, program sequencing
CR カテゴリー
4.22, 6.23, 5.24
編集者

For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of go to statements in the programs they produce. More recently I discovered why the use of the go to statement has such disastrous effects, and I became convinced that the go to statement should be abolished from all "higher level" programming languages (i.e. everything except, perhaps, plain machine code). At that time I did not attach too much importance to this discovery; I now submit my considerations for publication because in very recent discussions in which the subject turned up, I have been urged to do so.

私は何年も前から、プログラマーの質は、彼らが作るプログラムに含まれる go to 文の密度の減少関数であるという観察結果を知っていました。最近になって私は、なぜ go to 文を使うとこれほど悲惨な結果になるのかを発見し、すべての「高レベル」のプログラミング言語（おそらくプレーンなマシンコードを除くすべての言語）から go to 文を廃止すべきだと確信しました。当時、私はこの発見をあまり重要視していませんでしたが、最近の議論でこの話題が出てきたため、私の考察を出版するように求められたので、今回提出することにしました。

My first remark is that, although the programmer's activity ends when he has constructed a correct program, the process taking place under control of his program is the true subject matter of his activity, for it is this process that has to accomplish the desired effect; it is this process that in its dynamic behavior has to satisfy the desired specifications. Yet, once the program has been made, the "making' of the corresponding process is delegated to the machine.

最初に申し上げたいのは、プログラマーの活動は正しいプログラムを作成した時点で終了しますが、そのプログラムの制御下で行われるプロセスこそが彼の活動の真の主題であるということです。しかし、プログラムが作成されると、それに対応するプロセスの「作成」は機械に委ねられることになる。

My second remark is that our intellectual powers are rather geared to master static relations and that our powers to visualize processes evolving in time are relatively poorly developed. For that reason we should do (as wise programmers aware of our limitations) our utmost to shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.

第二に、私たちの知的能力は、どちらかというと静的な関係を習得することに向いており、時間的に変化するプロセスを視覚化する能力は相対的に劣っているということです。だからこそ、私たちは（自分の限界を知っている賢いプログラマーとして)、静的なプログラムと動的なプロセスとの間の概念的なギャップを縮めるために、(テキスト空間に広がる）プログラムと（時間的に広がる）プロセスとの間の対応関係をできるだけ些細なものにするために、最大限の努力をしなければならないのです。

Let us now consider how we can characterize the progress of a process. (You may think about this question in a very concrete manner: suppose that a process, considered as a time succession of actions, is stopped after an arbitrary action, what data do we have to fix in order that we can redo the process until the very same point?) If the program text is a pure concatenation of, say, assignment statements (for the purpose of this discussion regarded as the descriptions of single actions) it is sufficient to point in the program text to a point between two successive action descriptions. (In the absence of go to statements I can permit myself the syntactic ambiguity in the last three words of the previous sentence: if we parse them as "successive (action descriptions)" we mean successive in text space; if we parse as "(successive action) descriptions" we mean successive in time.) Let us call such a pointer to a suitable place in the text a "textual index."

ここで、プロセスの進行状況をどのように特徴づけるかを考えてみましょう。(具体的には，動作の時間的な連続と考えられるプロセスを，任意の動作の後に停止したとすると，そのプロセスを全く同じ時点までやり直すためには，どのようなデータを修正すればよいのでしょうか？） プログラムテキストが、例えば代入文（ここでは単一のアクションの記述とみなします）の純粋な連結であれば、プログラムテキストの中で、連続する 2 つのアクションの記述の間のポイントを示すだけで十分です。(go to 文がない場合、前の文の最後の 3 つの単語の構文上の曖昧さを自分で許容することができます。「連続した(アクション記述)」と解析すれば、テキスト空間での連続を意味し、「(連続したアクション)記述」と解析すれば、時間での連続を意味します)。このような、テキスト内の適切な場所へのポインタを "テキスト・インデックス "と呼ぶことにしましょう。

When we include conditional clauses (if B then A), alternative clauses (if B then A1 else A2), choice clauses as introduced by C. A. R. Hoare (case[i] of (A1, A2,⋯, An)),or conditional expressions as introduced by J. McCarthy (B1 -> E1, B2 -> E2, ⋯, Bn -> En), the fact remains that the progress of the process remains characterized by a single textual index.

条件節（if B then A)、代替節（if B then A1 else A2)、C. A. R. Hoare が紹介した選択節（case[i] of (A1, A2,---,An）)、J. McCarthy が紹介した条件式（B1 -> E1, B2 -> E2, ---, Bn -> En）などが含まれていても、処理の進行が単一のテキストインデックスで特徴づけられているという事実が残っているのである。

As soon as we include in our language procedures we must admit that a single textual index is no longer sufficient. In the case that a textual index points to the interior of a procedure body the dynamic progress is only characterized when we also give to which call of the procedure we refer. With the inclusion of procedures we can characterize the progress of the process via a sequence of textual indices, the length of this sequence being equal to the dynamic depth of procedure calling.

プロシージャを言語に含めるとなると、もはや単一のテキストインデックスでは不十分であることを認めざるを得ません。テキストインデックスが手続き本体の内部を指している場合、その手続きのどの呼び出しを参照しているかを示すときにのみ、動的な進行が特徴付けられます。手続きを含めると、一連のテキストインデックスによってプロセスの進行を特徴付けることができ、このシーケンスの長さは手続き呼び出しの動的深さと等しくなります。

Let us now consider repetition clauses (like, while B repeat A or repeat A until B). Logically speaking, such clauses are now superfluous, because we can express repetition with the aid of recursive procedures. For reasons of realism I don't wish to exclude them: on the one hand, repetition clauses can be implemented quite comfortably with present day finite equipment; on the other hand, the reasoning pattern known as "induction" makes us well equipped to retain our intellectual grasp on the processes generated by repetition clauses. With the inclusion of the repetition clauses textual indices are no longer sufficient to describe the dynamic progress of the process. With each entry into a repetition clause, however, we can associate a so-called "dynamic index," inexorably counting the ordinal number of the corresponding current repetition. As repetition clauses (just as procedure calls) may be applied nestedly, we find that now the progress of the process can always be uniquely characterized by a (mixed) sequence of textual and/or dynamic indices.

ここで、繰り返し節（while B repeat A とか、repeat A until B とか）について考えてみよう。論理的に言えば、再帰的手続きで繰り返しを表現できるのだから、このような句はもはや不要である。一方では、反復節は現在の有限な装置で非常に快適に実装でき、他方では、「帰納」と呼ばれる推論パターンが、反復節が生み出すプロセスを知的にも把握し続けることを可能にしてくれるからである。繰り返し節を含むと、動的なプロセスの進行を記述するのに、テキストによる指標ではもはや十分ではない。しかし、繰り返し節の各項目には、いわゆる「動的索引」を関連付けることができ、対応する現在の繰り返しの序数を不可避的に数えることができる。反復節は（手続き呼び出しと同様に）ネストして適用することができるので、現在、プロセスの進行は常にテキストおよび/または動的インデックスの（混合）シーケンスによって一意に特徴付けることができることがわかります。

The main point is that the values of these indices are outside programmer's control; they are generated (either by the write-up of his program or by the dynamic evolution of the process) whether he wishes or not. They provide independent coordinates in which to describe the progress of the process.

これらの指標は、プログラマが望むと望まざるとにかかわらず、（プログラムの記述やプロセスの動的進化によって）生成されるものであり、プログラマのコントロール外であることが要点である。これらの指標は、プロセスの進行を記述するための独立した座標を提供する。

Why do we need such independent coordinates? The reason is - and this seems to be inherent to sequential processes - that we can interpret the value of a variable only with respect to the progress of the process. If we wish to count the number, n say, of people in an initially empty room, we can achieve this by increasing n by one whenever we see someone entering the room. In the in-between moment that we have observed someone entering the room but have not yet performed the subsequent increase of n, its value equals the number of people in the room minus one!

なぜ、このような独立した座標が必要なのだろうか。その理由は--これは逐次処理に固有と思われるが--プロセスの進行に関してのみ変数の値を解釈することができるからである。例えば、最初は誰もいない部屋にいる人の数 n を数えたい場合、誰かが部屋に入ってくるのを見るたびに n を 1 つずつ増やしていけばよいのです。誰かが部屋に入るのを見たが、その後に n を増加させていない間の時間は、その値は部屋にいる人の数から 1 を引いたものになるのだ

The unbridled use of the go to statement has an immediate consequence that it becomes terribly hard to find a meaningful set of coordinates in which to describe the process progress. Usually, people take into account as well the values of some well chosen variables, but this is out of the question because it is relative to the progress that the meaning of these values is to be understood! With the go to statement one can, of course, still describe the progress uniquely by a counter counting the number of actions performed since program start (viz. a kind of normalized clock). The difficulty is that such a coordinate, although unique, is utterly unhelpful. In such a coordinate system it becomes an extremely complicated affair to define all those points of progress where, say, n equals the number of persons in the room minus one!

go to ステートメントを濫用すると、プロセスの進捗を記述するための有意義な座標系を見つけるのがひどく困難になるという直接的な結果が生じる。通常、よく選ばれた変数の値も考慮に入れますが、これは問題外です。なぜなら、これらの値の意味を理解するのは、進捗状況との関係だからです もちろん、go to 文を使っても、プログラム開始以来実行されたアクションの数をカウントするカウンタ（つまり一種の正規化された時計）により、進捗状況を一意に表現することができます。しかし、このような座標は一意ではあるが、全く役に立たない。このような座標系では、例えば n が部屋の中の人数から 1 を引いた数に等しくなるような進行のポイントをすべて定義するのは、非常に複雑な問題になります

The go to statement as it stands is just too primitive; it is too much an invitation to make a mess of one's program. One can regard and appreciate the clauses considered as bridling its use. I do not claim that the clauses mentioned are exhaustive in the sense that they will satisfy all needs, but whatever clauses are suggested (e.g. abortion clauses) they should satisfy the requirement that a programmer independent coordinate system can be maintained to describe the process in a helpful and manageable way.

Go to ステートメントは、そのままではあまりにも原始的で、自分のプログラムを混乱させることを誘いすぎています。Go to 文の使用を制限するために考えられた条項を考慮し、評価することができます。私は、言及した条項がすべてのニーズを満たすという意味で網羅的であるとは主張しないが、提案された条項（例えば中絶条項）が何であれ、それらは、有用かつ管理可能な方法でプロセスを記述するためにプログラマに依存しない座標系を維持することができるという要件を満たすはずである。

It is hard to end this with a fair acknowledgment. Am I to judge by whom my thinking has been influenced? It is fairly obvious that I am not uninfluenced by Peter Landin and Christopher Strachey. Finally I should like to record (as I remember it quite distinctly) how Heinz Zemanek at the pre-ALGOL meeting in early 1959 in Copenhagen quite explicitly expressed his doubts whether the go to statement should be treated on equal syntactic footing with the assignment statement. To a modest extent I blame myself for not having then drawn the consequences of his remark

これを正当に認めて終わらせるのは難しい。私の思考が誰によって影響を受けてきたかを判断するのは難しい。ピーター・ランディンやクリストファー・ストラチェイから影響を受けていないわけではないことは、かなり明白である。最後に、1959年初めにコペンハーゲンで開催されたALGOL以前の会議で、Heinz Zemanekがgo to文がassign statementと同等の構文で扱われるべきかどうか、かなり明確に疑問を表明したことを（かなりはっきりと覚えているので）記録しておきたいと思う。その時、彼の発言の帰結を導き出せなかった自分を、ささやかながら責めている。

The remark about the undesirability of the go to statement is far from new. I remember having read the explicit recommendation to restrict the use of the go to statement to alarm exits, but I have not been able to trace it; presumably, it has been made by C. A. R. Hoare. In [1, Sec. 3.2.1.] Wirth and Hoare together make a remark in the same direction in motivating the case construction: "Like the conditional, it mirrors the dynamic structure of a program more clearly than go to statements and switches, and it eliminates the need for introducing a large number of labels in the program."

go to文が好ましくないという指摘は、今に始まったことではありません。私は、go to文の使用をアラームの終了に限定することを明示的に推奨しているのを読んだ記憶があるが、それを追跡することはできなかった。[1, Sec. 3.2.1.]において、Wirth と Hoare は、格構造の動機付けにおいて、同じ方向性の発言をしている。"条件式と同様、go to文やスイッチよりもプログラムの動的構造をより明確に映し出し、プログラムに多数のラベルを導入する必要をなくすことができる。"

In [2] Guiseppe Jacopini seems to have proved the (logical) superfluousness of the go to statement. The exercise to translate an arbitrary flow diagram more or less mechanically into a jump-less one, however, is not to be recommended. Then the resulting flow diagram cannot be expected to be more transparent than the original one.

[2]では、Guiseppe Jacopiniがgo to文の（論理的な）余分さを証明したようである。しかし、任意のフロー図を多かれ少なかれ機械的にジャンプのない図に変換する作業は、あまりお勧めできない。そうすると、出来上がったフロー図が、元のフロー図よりも透明性が高くなることは期待できない。

References:

参考文献

Wirth, Niklaus, and Hoare C. A. R. A contribution to the development of ALGOL. Comm. ACM 9 (June 1966), 413-432.

BÖhm, Corrado, and Jacopini Guiseppe. Flow diagrams, Turing machines and languages with only two formation rules. Comm. ACM 9 (May 1966), 366-371.

Wirth, Niklaus, and Hoare C. A. R. A contribution to the development of ALGOL.（Wirth、Niklaus、Hoare C. A. R.はALGOLの開発への貢献）。Comm. ACM 9 (June 1966), 413-432.
BÖhm, Corrado, and Jacopini Guiseppe. フローダイアグラム、チューリングマシン、2つの形成規則のみを持つ言語。Comm. ACM 9 (1966年5月), 366-371.

Edsger W. Dijkstra
Technological University
Eindhoven, The Netherlands

エドガー・W・ダイクストラ
工科大学
オランダ・アイントホーフェン