Goto Hell: the Need for Better Control Structures and Compilers

Craig Kelly

December 8, 2009

Table of contents

Thesis

Ongoing issues with goto should be viewed as a need to improve control structures and compiler optimization techniques

Background

- The unconditional branch instruction (called the "goto" statement) has been a source of controversy for more than 30 years.
- Although the most famous pro and con opinions were written by Dijkstra and Knuth respectively many more articles, papers, and discussion have considered the issue at length.
- In 1987, more than 20 years after the arguments began, F. Rubin claimed that the belief that goto should be eliminated lacked an adequate rational foundation

Justification

- Some of the most influential scholars in the computing field have published papers and articles concerned with nothing but the evaluation of a single small instruction common to most procedural languages.
- Dijkstra's "Go to statement considered harmful" was meant to be a short article in a journal, but published as a letter to the editor to speed publication.

Practically Everywhere

- The goto statement "does not appear in most formal systems of computability theory, but does appear in programming language extensions of those systems"
- The Turing machine itself has the concept of a goto statement!

Required

Rojas demonstrated there are only four basic instructions needed for a von Neumann computer:

1. load
2. store
3. increment
4. goto

Scope of Inquiry

- As mentioned above, goto is intrinsic to von Neumann computers and so much be viewed as necessary to the hardware instructions.
- Most declarative languages (such as SQL) and functional languages (such as Erlang) don't contain a concept related to a procedural branch and so are outside any scope of consideration.

Genesis

As a matter of history, Knuth claims that Peter Naur was the first to publish about "harmful go to's"

Peter Naur said:

If you look carefully you will find that surprisingly often a go to statement which looks back really is a concealed for statement. And you will be pleased to find how the clarity of the algorithm improves when you insert the for clause where it belongs...

Giants

The most influential publications in the historical goto debate are

- Dijkstra's "Go to statement considered harmful"
- Knuth's "Structured Programming with go to Statements"

Dijkstra was not a fan

Dijkstra's claimed that "[t]he goto statement as it stands is just too primitive; it is too much an invitation to make a mess of ones program" However, much of the article is dedicated to the conceptual handicap imposed by a program containing goto's

Dijkstra said

...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.

Halt!

- Dijkstra develops simple way of keeping enough data to restart a program
- Leavenworth pointed out that Dijkstra's argument can be considered in a stronger light; unrestricted goto's increase the difficulty of the halting problem. If goto's are removed, "there are only two ways a program may fail: either by infinite recursion, or by the repetition clause."

But Not Always

Knuth quotes personal correspondence he has with Dijkstra

Dijkstra said

Please don't fall into the trap of believing that I am terribly dogmatical about [the go to statement]. I have the uncomfortable feeling that others are making a religion out of it, as if the conceptual problems of programming could be solved by a single trick, by a simple form of coding discipline!

Best of Time, Worst of Times

- Knuth's argument is often seen as a counterpoint to Dijkstra – **pro goto**
- Knuth actually favors "the elimination of go to's in certain cases"
- Knuth does give multiple reasons when goto should be used. Leavenworth sums up:

Reasons to Use Goto

1. Synthesis of missing control structures
2. Efficiency
3. Abnormal Exits

Can Goto's Just be Removed?

- Two years before Dijkstra's article, Jacopini had shown that goto's could be removed via a technique involving the transformation of flow diagrams
- Dijkstra also pointed out that the resulting program was actually less clear than the original and so was actually worse than a program with a goto.
- Knuth also mentions Jacopini and reports even more work by Ashcroft and Kosaraju

Replacing Goto's Efficiently

- Goto's could be replaced with no extra computation
- **A Catch:** The language must allow exiting an arbitrary number of nested loops (labeled break)
- Accomplished with many languages via function return, but that requires *compiler inlining*

Cyclomatic Complexity

- Mainly thought of as a software engineering metric
- Also supplies a formal definition of a non-structured program...
- ...Which gives a working formal definition of what not to do to write a structured program

McCabe says

A structured program can be written by not branching out of or into a loop, or out of or into a decision

Goto and Bliss

Wulf said

Any rational set of restrictions is equivalent to eliminating the [goto] construct if an adequate set of other control primitives is provided

- Asks "is it practical"?
- Argues that it doesn't matter!
- The Bliss language worked out just fine

Let's Do Better

- Wulf explicitly states that we should be using better compilers if want more efficient programs
- Knuth theorized we might get better control structures but didn't think optimizing compiler were good enough (proposed "Program Manipulation Systems")

Beyond the While Loop

- Jonsson's Pancode – adds repeat to if's and has explicit short-circuiting
- Wulf's *leave*⟨*label*⟩ – Java's labeled break statements
- Bochman's multiple loop exits – Python's optional else clause for loops

Java and Python have no goto statements, but a cynic may respond that these languages still have efficiency problems since Python is interpreted and Java executes on a virtual machine. As a result, compiler technology must also be considered.

Need to Remove Goto's to be Efficient!

Much of the argument in favor of keeping goto in higher-level languages is based on efficiency, but many "modern" compiler techniques require removing goto's!

- Erosa – Work on eliminating goto's for an optimizing compiler (almost equivalent)
- Allen – Vectoring compiler
- Ammarguellat – Parallelizing compiler

Better?

$$T = \text{Time for program} \tag{1}$$

$$\alpha = \text{Goto Removal factor (increase)} \tag{2}$$

$$N = \text{Degree of parallelization} \tag{3}$$

$$\text{Execution Time} = \alpha T \frac{1}{N} = \frac{\alpha T}{N} \tag{4}$$

Yes! – as long as $N > \alpha$

Conclusion

- Those arguing for the use of goto do so almost universally on the basis of efficiency.
- So let's create alternatives to goto that are just as efficient

Which means

- Any situation that appears to require a goto should be considered a requirement for a control structure that can be used instead
- This creates an ongoing requirement to improve compiler optimizations so that these structures are at least as efficient as goto statements