Formalizing Structured Control Flow Graphs

Amit Sabne, Putt Sakdhangool, and Rudolf Eigenmann

School of Electrical and Computer Engineering, Purdue University

LCPC 2016

Structured Programming

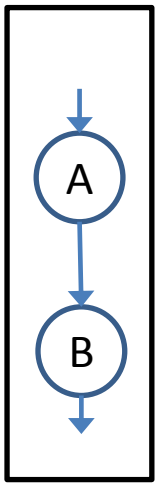
Programs written using few constructs [Zhang'04, Bohm'66, Williams'77]

- Sequence of statements
- **If then else** blocks
- **While/for** Loops

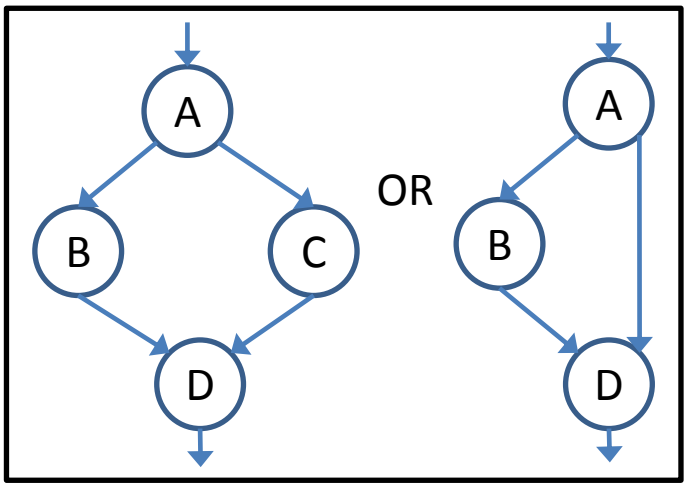
- **Case** statement (?) [Dijkstra'72, Moretti'01]

Why Structured Programming?

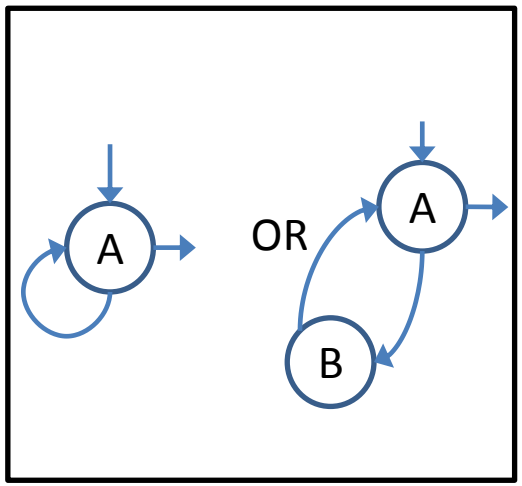
- Ease of program readability and maintenance
- "Structured" CFGs, which were assumed to form from structured programs, are easier to analyze
- Structured CFGs are "composed" of base patterns



Sequence



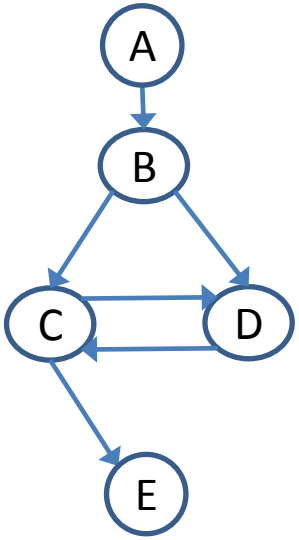
Selection



Loop

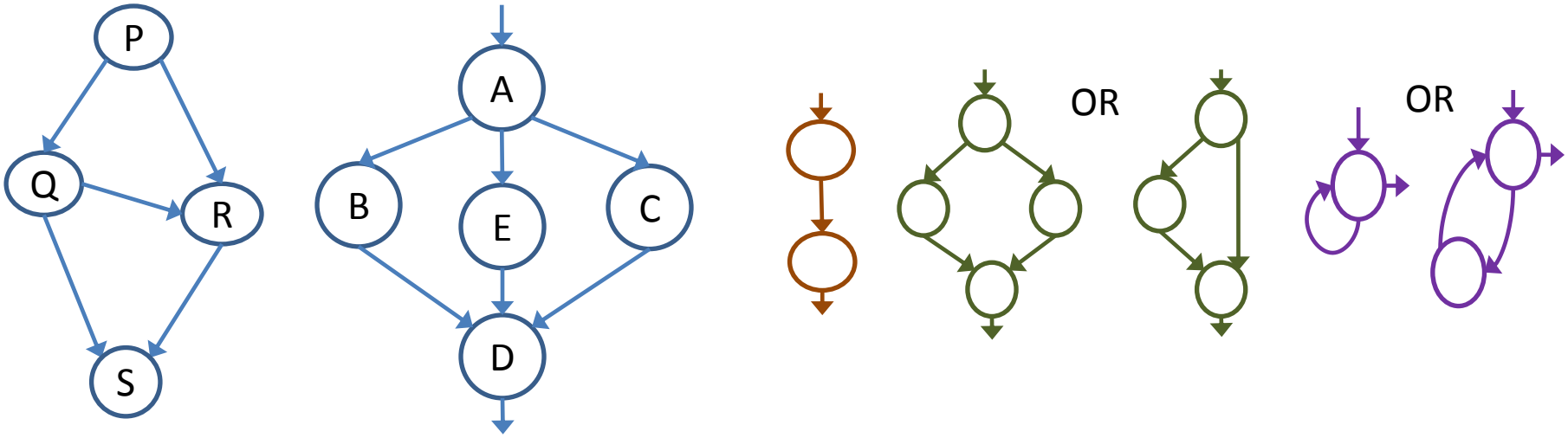
Why Structured CFGs?

- Are never irreducible
  - Compilers often don't optimize irreducible loops
- Analysis is easy (and fast) – all loops are canonical
- Lower the penalty of divergent execution on SIMD units
- Decompilation is easy and always possible
  - Java bytecode with irreducibility cannot be decompiled (Java does not support gotos)



Issues

- Structured programs can be easily identified
- Not the case for structured CFGs
  - Base patterns fail to “decompose” large CFGs



- Compiler front-ends can turn structured programs into “unstructured CFGs”
- Compiler optimizations cause unstructuring: e.g. jump threading, tail call elimination, short-circuit optimization etc.

Issues

- Abundant literature refers to structured CFGs, without defining them

gets of all previous branches. For local, structured branches, branch flags simplify out very

Kennedy et. al '83

gram, such as loops and conditionals [6, 8, 9]. A *structured control-flow graph* is a graph that can be decomposed into subgraphs that represent control structures of a high-level language, with a single entry point and a single exit point per subgraph. In most cases, the use of Goto statements

Moretti et. al '01

Tse'87

A module is said to be *unstructured* if and only if it contains multiple iteration exits and/or multiple entries.

definitions and a basic corollary are given. The methods for converting branches into block structured control statements are developed in Section 3. Here, program transfor-

Zhang et. al'04

statement. Given a source program with **go to**'s, can we produce an equivalent target program that renounces **go to**'s in favor of more structured control constructs? The first step in tackling this question is to settle the ground rules: What

Ramshaw '83

The linear scan algorithm does not operate on a structured control flow graph, but on a linear list of blocks. The block order has a

Wimmer et. al ‘10

tures. In this section, we show how loops can be recovered from unstructured graphs, and we define the macro *loop* which is used extensively in our specifi-

Kalvala et. al ‘09

Our concrete contributions are: The analysis can directly be performed on arbitrarily structured and immutable control-flow graphs. The computational model is non-recursive with

Kleinsorge et. al ‘13

10.9 DATA-FLOW ANALYSIS OF STRUCTURED FLOW GRAPHS

Gotoless programs have reducible flow graphs; so do programs encouraged by several programming methodologies. Several studies of large classes of programs have revealed that virtually all programs written by people have flow graphs that are reducible.[10] This observation is relevant for optimization purposes because we can find optimization algorithms that run significantly faster on reducible flow graphs. In this section we discuss a variety of flow-graph concepts, such as "interval analysis," that are primarily relevant to structured flow graphs. In essence, we shall apply the syntax-directed techniques developed in Section 10.5 to the more general setting where the syntax doesn't necessarily provide the structure, but the flow graph does.

Aho et. al ‘86

Compilers work on CFGs, and not source codes.
We need formalized way to detect CFG structuredness!

Single-entry-single-exit (SESE) Regions

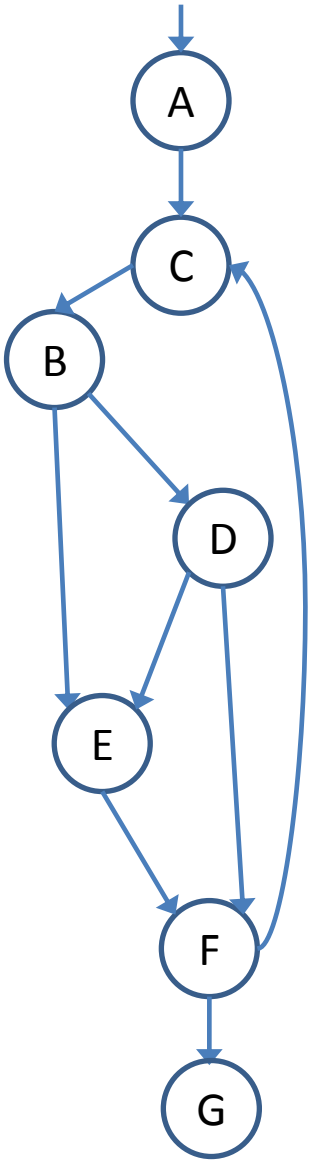
The region between two nodes (edges) A and B is said to be SESE if

- A dominates B, and
- B post-dominates A, and
- Every cycle containing A also contains B and vice versa.

A single node (edge) is always an SESE region.

Aren't SESE – regions between (B – E), (A – F)

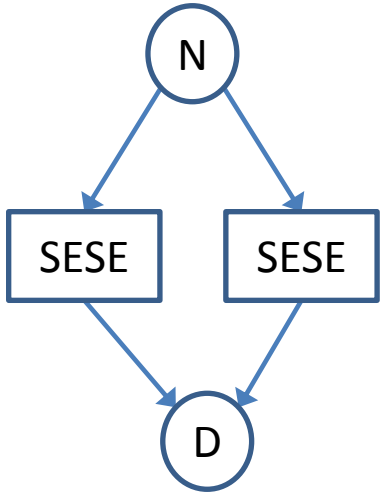
Are SESE – regions between (A – G), (C – F)



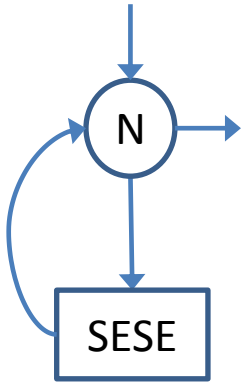
Formalizations

Maximum in/out degree is 2

Condition node: node with two out-edges



- **Structured selection condition node** : A condition node N where
  - For any path from N to its IPDOM, the region between the first and last edges is SESE.
  - the region between the N and its IPDOM is SESE and is called selection body.

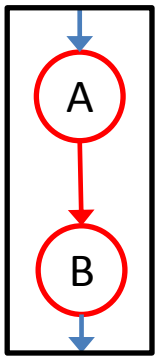


- **Structured loop condition node** : A condition node N where
  - there exists an SESE region between one of its out-edges and in-edges.
  - This SESE region is called the loop body.

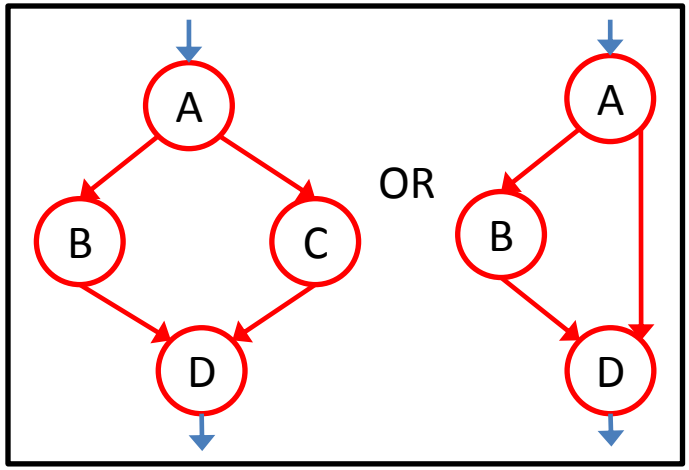
- **Unstructured condition node** : All other condition nodes

Formalizations – Base Patterns

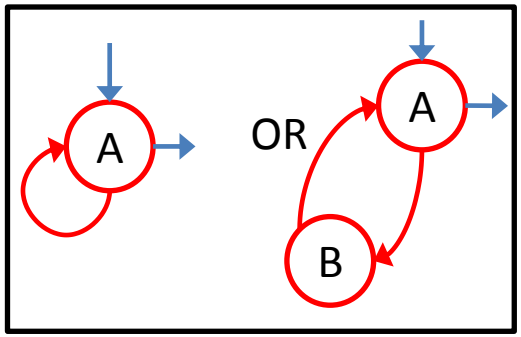
- **Sequence** : Two nodes, A and B, along with an edge A → B form a sequence if
  - B is the only successor of A, and
  - A is the only predecessor of B



- **Selection** : Contains a structured selection condition node, its IPDOM, and the selection body
  - The selection body must have at least one node, and
  - any path from the selection condition node to the IPDOM can have at most one node.



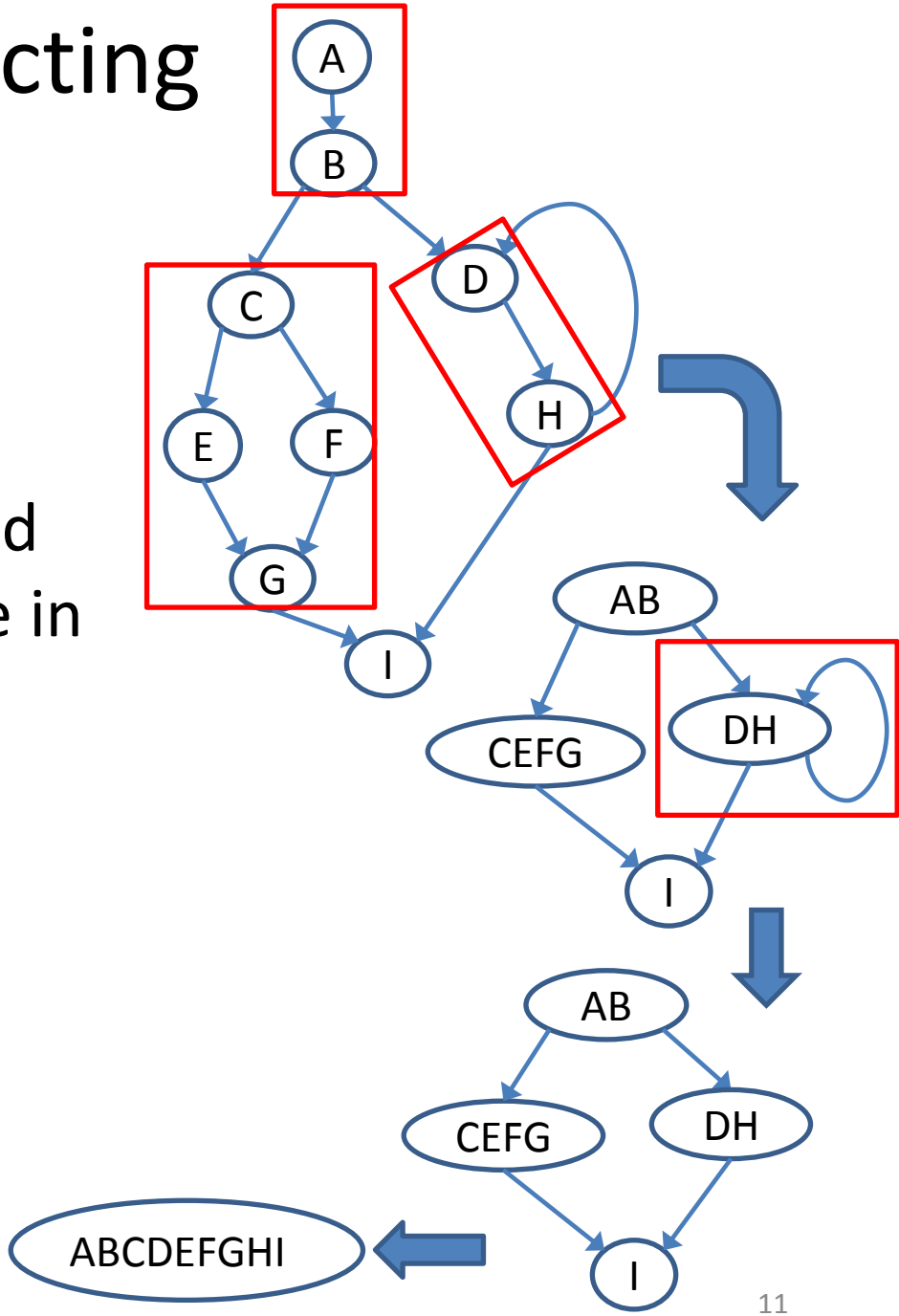
- **Loop** : Contains a structured loop condition node, the loop body, and the entry and exit edges of the loop body
  - The loop body can contain at most one node.



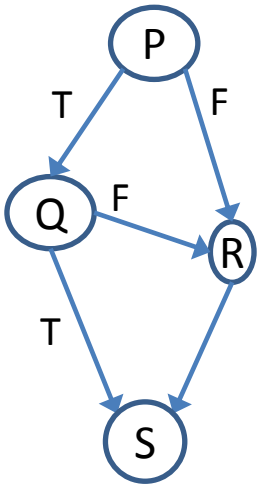
Formalization – Detecting Structuredness

- **Folding**
  - conceptual process of replacing a base structured pattern with a single node in the CFG.

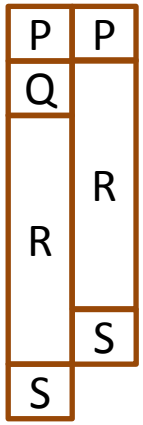
- If repeated folding yields single node → CFG is structured

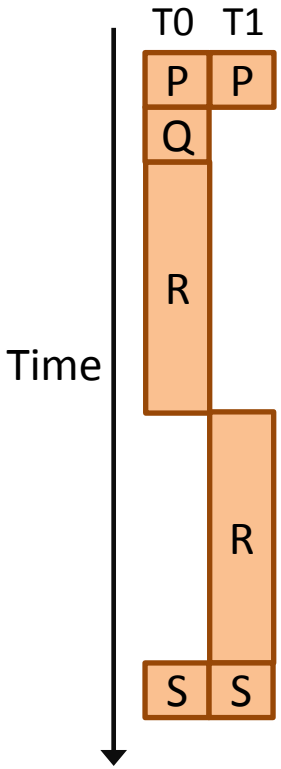


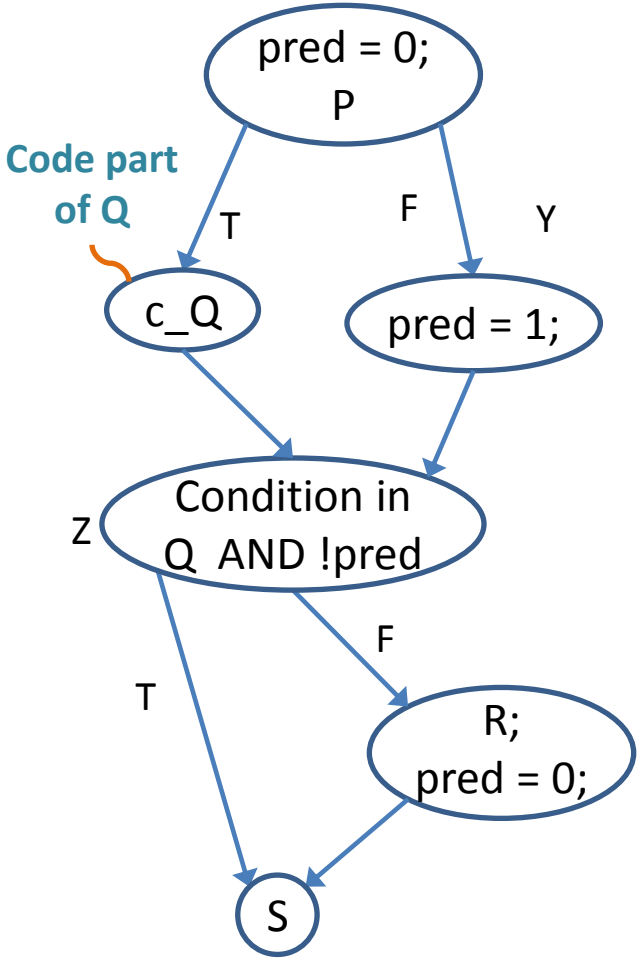
Better SIMD Execution

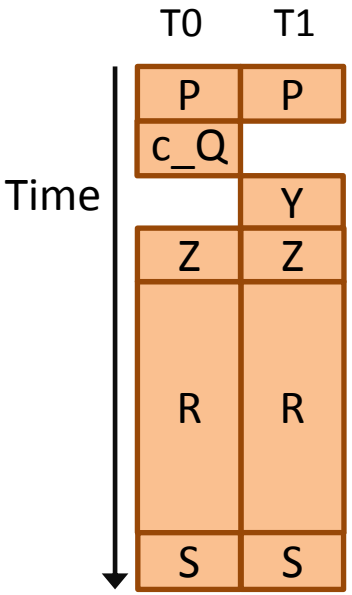


Execution pattern of each thread









Conclusion and Future Work

- We have formalized the notion of “structured CFGs” and have presented a mechanism to detect them

What’s next:

- Current unstructured-to-structured converters can lead to exponential code blowup
- Design a mechanism to avoid it

References

- C. Bohm and G. Jacopini, “Flow diagrams, turing machines and languages with only two formation rules,” Commun. ACM, vol. 9, pp. 366–371, May 1966.
- F. Zhang and E. H. D’Hollander, “Using hammock graphs to structure programs,” IEEE Trans. Softw. Eng., vol. 30, pp. 231–245, Apr. 2004.
- Z. Ammarguellat, “A control-flow normalization algorithm and its complexity,” IEEE Trans. Softw. Eng., vol. 18, pp. 237–251, Mar. 1992.
- M. H. Williams and H. L. Ossher, “Conversion of unstructured flow diagrams to structured form,” Comput. J., vol. 21, no. 2, pp. 161–167, 1978.
- G. Oulsnam, “Unravelling unstructured programs,” Comput. J., vol. 25, no. 3, 1982.
- O. J. Dahl, E. W. Dijkstra, C. A. R. Hoare,, Structured Programming. UK: Academic Press Ltd. 1972.
- H. Wu, G. Diamos, J. Wang, S. Li, and S. Yalamanchili, “Characterization and transformation of unstructured control flow in bulk synchronous gpu applications,” Int. J. High Perform. Comput. Appl., vol. 26, pp. 170–185, May 2012.
- J. Anantpur and G. R., “Taming control divergence in gpus through control flow linearization,” in Compiler Construction (A. Cohen,ed.), 2014.
- A. V. Aho, R. Sethi, and J. D. Ullman, Compilers: Principles, Techniques, and Tools. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1986.
- M. S. Hecht, Flow Analysis of Computer Programs. New York, NY, Elsevier Science Inc., 1977.

References

- L. Carter, J. Ferrante, and C. D. Thomborson, “Folklore confirmed: reducible flow graphs are exponentially larger,” in POPL 2003
- C. Collberg, C. Thomborson, and D. Low, “Manufacturing cheap, resilient, and stealthy opaque constructs,” POPL ’98
- B. S. Baker. 1977. An Algorithm for Structuring Flowgraphs. J. ACM 24, 1 (January 1977), 98-120
- N. Chapin and S. P. Denniston. “Characteristics of a structured program”. SIGPLAN Not. 13, 5 (May 1978), 36-45.
- E. W. Dijkstra. 1968. Letters to the editor: “Go to statement considered harmful”. Commun. ACM 11, 3 (March 1968), 147-148.
- A. M. Erosa and L. J. Hendren, "Taming control flow: a structured approach to eliminating goto statements," Computer Languages, Proceedings of the 1994 International Conference on, Toulouse, 1994, pp. 229-240.
- D. Knuth. 1979. Structured programming with go to statements. In Classics in software engineering, Edward Nash Yourdon (Ed.). Yourdon Press, Upper Saddle River, NJ, USA 257-321.
- E. Moretti, G. Chanteperdrix, and A. Osorio. 2001. New Algorithms for Control-Flow Graph Structuring. In Proceedings of the Fifth European Conference on Software Maintenance and Reengineering (CSMR '01)
- L. Ramshaw. 1988. Eliminating go to's while preserving program structure. J. ACM 35, 4 (October 1988), 893-920.