Formalizing Structured Control Flow Graphs

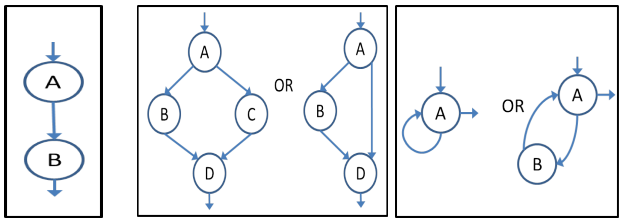
Amit Sabne, Putt Sakdhnagool, and Rudolf Eigenmann

Purdue University, West Lafayette IN 47907, USA
{asabne, psakdhna, eigenman}@purdue.edu

Abstract. Structured programs are easier to understand, and are compiler friendly [2][5][14]. However, compilers do not process the source programs directly; they instead work on control flow graphs (CFGs) of the programs. Despite a long history of contributions to structured programming, the existing formalizations of structured CFGs are insufficient. This paper aims to fill this gap.

1 Introduction

Structured programming is a paradigm where programs are written using just three base constructs [6][14], namely, i) sequence of statements, ii) **if**-then-**else** blocks, and iii) loops. Structured programming increases program readability [5][14]. It was believed that the generated CFGs from structured programs do not contain irreducibility [3], easing compiler analyses. However, this is only true if compiler front-ends performed one-to-one mappings of structured program constructs, which is not the case in practice. Unstructured CFGs are not supported on certain architectures, e.g., OpenCL disallows `gotos` because AMD GPUs do not support unstructured CFGs. Compiler optimizations, such as short-circuit optimization and inlining, can turn structured programs into unstructured CFGs [13]. Worse, some transformations, such as tail call elimination and jump threading, can lead to irreducibility [11]. It is therefore essential to have mechanisms that can ensure the structuredness of resulting CFGs.



(a) Sequence *(b) Selection* *(c) Loop*

Fig. 1: Pictorially Represented Base Structured Patterns

The prevalent notion [7][12] of structured CFGs considers three base patterns similar to the structured program constructs mentioned above. They comprise a sequence (Fig. 1a), a selection (Fig. 1b), and a loop (Fig. 1c). We argue that such pictorial depiction of base patterns alone is insufficient and leads to imprecision. The "definition" fails to clearly distinguish a structured CFG from an unstructured one. Consider the CFGs in Fig. 2, which do not show any obvious matching to the base patterns. Creating structured CFGs contrasts with structured programming, where just by looking for the presence of unstructuring-causing constructs, such as `goto` and `break` statements, unstructuredness can be easily detected. The difficulty in doing the same in CFGs arises because the pictorial representations of the base structured patterns do not show how to compose the patterns into larger CFGs or decompose large CFGs.

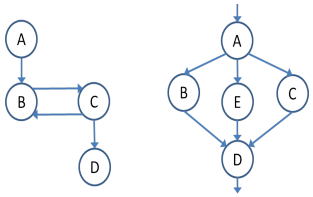


Fig. 2: Are these CFGs structured? *The base patterns in Fig. 1 fail to answer.*

The contribution of this paper is to formalize structured control flow graphs.

2 Preliminaries

Definition 1. Path : A path between nodes A and B is an ordered list of adjacent edges and vertices. The list begins with an out-edge of A, and ends with an in-edge of B.
Definition 2. Condition Node : Any node with two or more out-edges.
Definition 3. Region : The region between two nodes (edges) A and B contains all nodes and edges that are present on any path from A to B.
Definition 4. Single-entry-single-exit (SESE) region : The region between two nodes (edges) A and B is SESE if all of the following are true: i) A dominates B, ii) B post-dominates A, iii) every cycle containing A also contains B and vice versa.

A node (edge) is a SESE region by itself. Each base pattern in Fig. 1 represents a SESE region. For the selection pattern (Fig. 1b), node A is the entry and node D is the exit. For the loop pattern (Fig. 1c), node A is the entry as well as the exit. In Fig. 4a, region between R and F is not SESE, while the region between C and D is.
Definition 5. Loop condition node, loop path : A given condition node N is said to be a loop condition node, if there is a simple path (all nodes along the path have in/out-degree of one) that originates and ends at N. We refer to any such path as a loop path.

3 Previous Work on Defining Structuredness

There is ample ambiguity even in the terminology used while describing structured programming, e.g., structured programs can have `goto` statements in [8], while [9] considers programs with `case` statements to be structured too.

The first work, from Williams [12], intepreted structured CFGs to be composed of the base patterns in Fig. 1, and unstructured CFGs must comprise one of the five patterns of unstructuredness. Oulsnam [10] presented similar patterns. However, the lack of formalization in the definition required sophisticated pattern matching to assess the CFG structuredness, toward which no algorithms were provided. E.g., the CFG on the left in Fig. 2 neither matches base structured patterns in Fig. 1, nor the base unstructured patterns in [12].

The latest attempt of defining structured CFGs is from Anantpur et. al. [4]. However, their definition, although formal, does not truly capture the notion of structured CFGs. Their definition recognizes unstructuring as scenarios where i) there exists an incoming/outgoing edge from a loop, or ii) an edge exists between a condition node and a node with multiple in-edges where there is no dominator/post-dominator relationship between the two nodes. This definition departs from the base patterns. Consider the CFG in Fig. 3, where all nodes are strongly connected, with a common entry node, A. All nodes thus belong to the same loop [1], meaning that the first condition is satisfied. Node A is the only multiple in-edge node, but it dominates all other nodes. Hence the second condition is satisfied too. Thus, this CFG will be considered structured in [4], in spite of having no mapping to a high-level structured program.

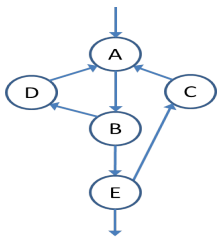


Fig. 3: This CFG is considered structured in [4], although it maps to no structured program!

To remove these ambiguities, it is essential to formalize the notion of structured CFGs in a way that enables an algorithmic mapping to the three high-level program constructs used in structured programming. Doing so will ease compiler analyses.

4 Formalizing Structured CFGs

We now formalize the notion of structured CFGs. We start by providing formal definitions for structured condition nodes and for the base structured patterns. Next we introduce the conceptual framework of *folding*, which decomposes CFGs into base patterns.

Restrictions on CFGs: We consider CFGs with single entry and exit nodes. There exists a path from each node in the CFG to the exit node. Symmetrically, each node in the CFG can be reached from the entry node. **Also, the maximum in-degree and out-degree for all nodes is two.** Generality is maintained since a CFG with any in-degree or out-degree can be converted into a CFG with a maximum in-degree and out-degree of two.

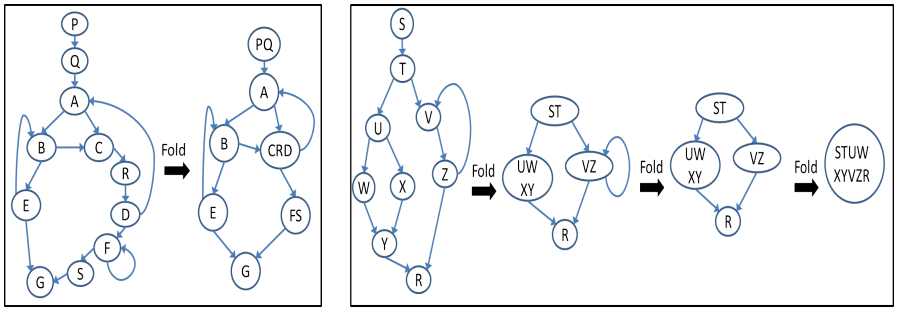
We begin by defining structured selection and loop condition nodes.

Definition 6. Structured selection condition node, selection body : A condition node N is a structured selection condition if for every path from N to its immediate post-dominator (IPDOM), the region between the first and last edges is SESE. Therefore, the region between the structured selection condition and its IPDOM is SESE as well, which is said to be its *selection body*.

Definition 7. Structured loop condition node, loop body : A structured loop condition node is a loop condition node having an SESE region between one of its out-edges and in-edges. This SESE region is called the *loop body*.

Definition 8. Unstructured condition node : If a condition node is neither a structured selection, nor a structured loop condition node, then it is an unstructured condition node.

Examples: Node A in the base selection pattern (Fig. 1b) is a structured selection condition. Node A in Fig. 1c is a structured loop condition, with edge A→A being both the entry and exit edge of the SESE region (left figure) or edge A→B being the entry edge and the edge B→A being the exit edge of the SESE region (right figure). For condition node A in Fig. 4a, G is the IPDOM. A→B →E→G is one path from A to G, on which the region between the first and last edges is not SESE. Hence, A is not a structured selection node. Similarly, for no out-edge and in-edge pair of A, there exists a SESE region. Therefore, A is not a structured loop condition node either. Hence, A is an unstructured condition node. On the other hand, in Fig. 4b, nodes U and T are structured selection condition nodes, while VZ is a structured loop condition node.



(a) Unstructured CFG ***(b) Structured CFG***

Fig. 4: Maximal Folding: Both CFGs are reducible

Now, we present formal definitions for the **base structured patterns**.

Definition 9. Sequence : Two nodes, A and B, and an edge A → B form a sequence if B is the sole successor of A, and A is the sole predecessor of B.

Definition 10. Selection : This pattern contains a structured selection condition node, its IPDOM, and the selection body. The selection body must contain at least one node, and any path from the selection condition node to the IPDOM can have at most one node.

Definition 11. Loop : This pattern 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.

To determine CFG structuredness, we introduce a new concept, called *folding*.

Definition 12. Folding : Folding is a process of replacing a base structured pattern with a single node in the CFG. During folding, any edge not belonging to the base pattern, but having its source (sink) node in the base pattern, is redirected so that the newly created single node is its source (sink).

Definition 13. Maximal Folding : Maximal folding repeatedly applies folding to a CFG until no more base structured patterns exist.

The above formalization of base structured patterns removes ambiguity and makes the process of folding straightforward. Implementing folding simply requires looking for base patterns formed on each node, which is a constant-time operation. Thus, maximal folding is $\mathcal{O}(n)$, where n is the number of nodes in the CFG. Fig. 4 shows examples of structured and unstructured CFGs and their maximally folded equivalents.

Definition 14. Completely Foldable CFG : If a maximally folded CFG contains a single node, then the CFG is called completely foldable.

Definition 15. Structured CFG : A CFG is said to be structured iff it is completely foldable. Otherwise, it is called an **unstructured CFG**.

Complete foldability, i.e., structuredness, implies that the CFG is composed of the base structured patterns. Structuredness gurantess CFG reducibility. While reducibility eases CFG analysis, it does not determine if the CFG is structured or not. E.g., the CFG in Fig. 4a is reducible, but is unstructured. Also, a structured CFG does not imply structuredness of all condition nodes. E.g. in Fig. 4b, Z is an unstructured condition node by itself. After the first folding step, the generated VZ node is a structured loop condition node.

5 Conclusion

This paper has identified that structured programs do not necessarily imply structured CFGs, and showed that previous definitions of structured CFGs led to ambiguities. The paper presented a formalization and a method for recognizing structured CFGs.

References

1. Aho, A.V., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1986)
2. Allen, F.E.: Control flow analysis. In: Proceedings of a Symposium on Compiler Optimization. pp. 1–19. ACM, New York, NY, USA (1970)
3. Ammarguellat, Z.: A control-flow normalization algorithm and its complexity. IEEE Trans. Softw. Eng. 18(3), 237–251 (Mar 1992)
4. Anantpur, J., R., G.: Taming control divergence in gpus through control flow linearization. In: Cohen, A. (ed.) Compiler Construction, Lecture Notes in Computer Science, vol. 8409, pp. 133–153. Springer Berlin Heidelberg (2014)
5. Baker, B.S.: An algorithm for structuring flowgraphs. J. ACM 24(1), 98–120 (1977)
6. Böhm, C., Jacopini, G.: Flow diagrams, turing machines and languages with only two formation rules. Commun. ACM 9(5), 366–371 (May 1966)
7. Chapin, N., Denniston, S.P.: Characteristics of a structured program. SIGPLAN Not. 13(5), 36–45 (May 1978), http://doi.acm.org/10.1145/953395.953398
8. Knuth, D.E.: Structured programming with go to statements. ACM Comput. Surv. 6(4), 261–301 (Dec 1974), http://doi.acm.org/10.1145/356635.356640
9. Moretti, E., Chanteperdrix, G., Osorio, A.: New algorithms for control-flow graph structuring. In: Fifth Conference on Software Maintenance and Reengineering, CSMR 2001, Lisbon, Portugal, March 14-16, 2001. pp. 184–187 (2001), http://dx.doi.org/10.1109/.2001.914984
10. Oulsnam, G.: Unravelling unstructured programs. Comput. J. 25(3), 379–387 (1982)
11. Stanier, J., Watson, D.: A study of irreducibility in c programs. Softw. Pract. Exper. 42(1), 117–130 (Jan 2012), http://dx.doi.org/10.1002/spe.1059
12. Williams, M.H.: Generating structured flow diagrams: The nature of unstructuredness. Comput. J. 20(1), 45–50 (1977), http://dx.doi.org/10.1093/comjnl/20.1.45
13. Wu, H., Diamos, G., Wang, J., Li, S., Yalamanchili, S.: Characterization and transformation of unstructured control flow in bulk synchronous gpu applications. Int. J. High Perform. Comput. Appl. 26(2), 170–185 (May 2012), http://dx.doi.org/10.1177/1094342011434814
14. Zhang, F., H. D'Hollander, E.: Using hammock graphs to structure programs. IEEE Trans. Softw. Eng. 30(4), 231–245 (Apr 2004)