McGill University
School of Computer Science
ACAPS Laboratory

Advanced Compilers, Architectures and Parallel Systems

Identifying Loops Using DJ Graphs

Vugranam C. Sreedhar
Guang R. Gao
Yong-fong Lee†

ACAPS Technical Memo 98
September 10, 1995

† Intel Corp.
M/S RN6-18
2200 Mission College Blvd.
Santa Clara, CA 95052-8119

ACAPS • School of Computer Science • 3480 University St. • Montréal • Canada • H3A 2A7
FTP: ftp-acaps.cs.mcgill.ca • WWW: http://www-acaps.cs.mcgill.ca/

Abstract

Loop identification is a necessary step in loop transformations for high-performance architectures. The *Tarjan intervals* are single-entry, strongly connected subgraphs, so they closely reflect the loop structure of a program [Tar74]. They have been used for loop identification. In this paper we give a simple algorithm for identifying both reducible and irreducible loops using DJ graphs. Our method can be considered as a generalization of Tarjan's interval-finding algorithm, since we can identify nested intervals (or loops) even in the presence of irreducibility.

Contents

List of Figures

1 Introduction

Loop identification is a necessary step in loop transformations for high-performance architectures. Some compilers detect loop structures using syntactic constructs (e.g., `for`, `while`, etc.), while others detect loops using flowgraphs. The latter approach is more general in that it can detect loops in programs that even use `goto` statements or that are represented in low-level intermediate languages. We follow this approach in the paper. One classical technique for detecting loops is using Tarjan's interval algorithm [Tar74]. The *Tarjan intervals* are *single* entry, strongly connected subgraphs [Tar74]. However, Tarjan's interval finding algorithm does not directly handle flowgraphs containing loops with more than one entry, i.e., loops with *multiple entries*. Such loops will be called as *irreducible loops* in this paper, whereas loops with single entry will be called *reducible loops* [Hec77]. There are extensions to Tarjan's algorithms that are listed in Section 5.

In this paper we give a simple algorithm for identifying both reducible and irreducible loops using DJ graphs [SG95, SGL95b]. Our method can also be considered as a generalization of Tarjan's interval-finding algorithm, with the unique feature that we can identify nested intervals (or loops) even when they are nested within irreducible regions.[1] Furthermore, we use level information in the DJ graph to detect finer irreducible regions, thus confining the effect of irreducibility to small regions.

In the next section we introduce the necessary background material and define terminology that will be used in this paper. Then, in Section 3, we introduce the notion of reducible and irreducible loops. In Section 4 we present our loop-finding algorithm. Finally, in Section 5, we discuss related work and give our conclusion as well as post some open questions for future work.

2 Background and Notation

A flowgraph is a connected, directed graph $G = (N, E, \mathrm{START}, \mathrm{END})$, where N is the set of nodes, E is the set of edges, $\mathrm{START} \in N$ is a distinguished start node with no incoming edges, and $\mathrm{END} \in N$ is a distinguished end node with no outgoing edges. Figure 1(a) shows an example flowgraph. If $x \to y \in E$, then x and y are called the *source* and the *destination* node of the edge, respectively. A *path* of length n is a sequence of edges $(x_0 \to x_1 \to \ldots \to x_n)$, where each $x_i \to x_{i+1} \in E$. We use the notation $P : x_0 \xrightarrow{*} x_n$ to represent a path of length zero or more, and $P : x_0 \xrightarrow{+} x_n$ to represent a path of length one or more. In our definition of flowgraph, we will assume that for every node $x \in N$ there is a path from START to x and a path from x to END.[2]

[1] In [Wol92] Wolfe compares different algorithms for identifying loops, and discuss loop anomalies that arise in unstructured programs. Our intention here is only to show how to extend Tarjan's interval structure to handle irreducible loops.

[2] We will use the terms flowgraph and control flow graph interchangeably.

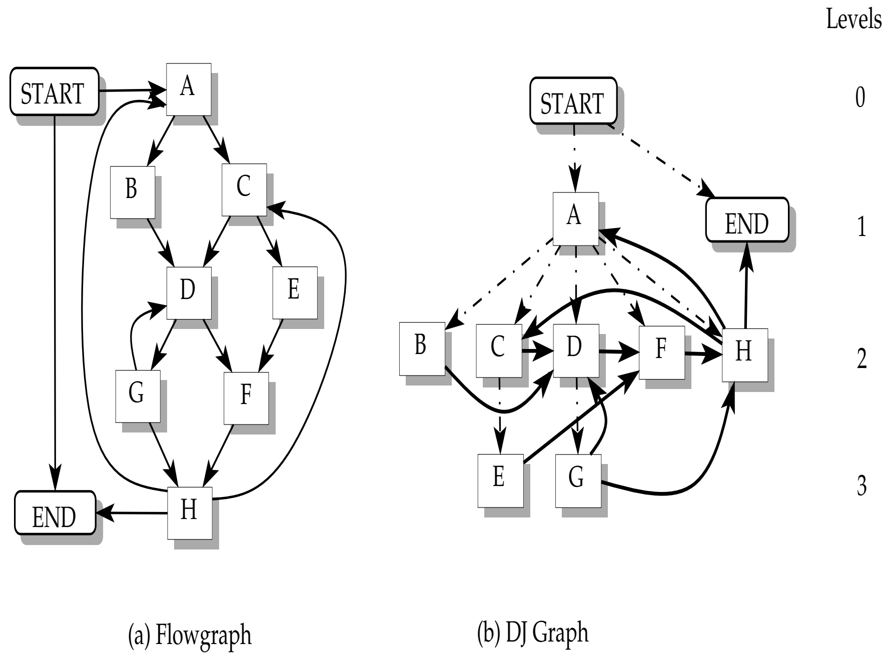


Figure 1: An example of a flowgraph and its DJ graph

If S is a set, we will use the notation $|S|$ to represent the number of elements in the set.

Node x **dominates** y if and only if all paths from START to y pass through x. We write x **dom** y to indicate that x dominates y. Node x **strictly dominates** y if and only if x **dom** y and $x \neq y$. We write x **stdom** y to indicate x strictly dominates y. A node y is said to immediately dominate another node x, denoted as $y = idom(x)$, if y **stdom** x and there is no other node z such that y **stdom** z **stdom** x. The dominance relation is reflexive and transitive, and can be represented by a tree, called the *dominator tree*. For each node in the dominator tree we associate a *level number* that is the depth of the node from the root of the tree. We write $x.level$ to indicate the level number of a node x.

The DJ graph of a flowgraph consists of the same set of nodes as in the flowgraph, and two types of edges called D edges and J edges. D edges are dominator tree edges. We define J edges as follows:

Definition 2.1 (J edge) *An edge $x \rightarrow y$ in a flowgraph is named a* **join edge** *(or* **J edge***) if $x \neq idom(y)$. Furthermore, y is named a* **join node***.*

Therefore, in order to construct the DJ graph of a flowgraph, we first construct the dominator tree of the flowgraph. Then, we insert the J edges into the dominator tree as follows:

For each **join node** y in the dominator tree, connect x to y (in the dominator tree) if and only if $x \to y$ is a **join edge** in the original flowgraph.

Figure 1(b) shows the DJ graph for our example flowgraph. To see how a J edge is inserted into the dominator tree, consider join node C in the example flowgraph. This node has two incoming edges, $A \to C$ and $H \to C$. Of these two incoming edges, node A strictly dominates join node C, so we do not insert a J edge from A to C in the corresponding dominator tree. However, node H does not dominate node C, and so we insert a J edge from node H to node C.

Given a DJ graph we distinguish between two types of J edges: Back J (BJ) edges and Cross J (CJ) edges. A J edge $x \to y$ is a BJ edge if y **dom** x; otherwise it is a CJ edge. Consider Figure 1: edge $H \to A$ is a BJ edge, whereas edge $H \to C$ is a CJ edge.

We can easily see that the time complexity for inserting all the J edges into the dominator tree is $O(|E|)$. Therefore the time complexity of constructing a DJ graph is linear with respect to the size of the flowgraph.[3]

Given a flowgraph (DJ graph) we can perform a depth-first search to derive a depth-first spanning tree. With respect to the depth-first spanning tree, we can classify an edge $x \to y$ in the flowgraph (DJ Graph) into one of four types: (1) **sp-back** edge, if $y = x$ or y is an ancestor of x on the spanning tree; (2) **sp-tree** edge, if x is the parent of y on the spanning tree; (3) **sp-forward** edge, if x is an ancestor but not the parent of y on the spanning tree; and (4) **sp-cross** edge, if x and y have no ancestor-descendant relationship.

3 Reducible and Irreducible Loops

Recall that reducible loops are single entry strongly connected components. Reducible loops are related to *reducibility* of flowgraphs. Hecht and Ullman gave the following definition for flowgraph reducibility [HU74, ASU86].

Definition 3.1 *A graph G is reducible if and only if we can partition the edges into two disjoint groups, called the* **forward** *edges and* **back** *edges, with the following two properties:*

1. *The forward edges form an acyclic graph in which every node can be reached from the initial node of G.*

2. *The back edges consists only of edges whose destination nodes dominate their source nodes.*

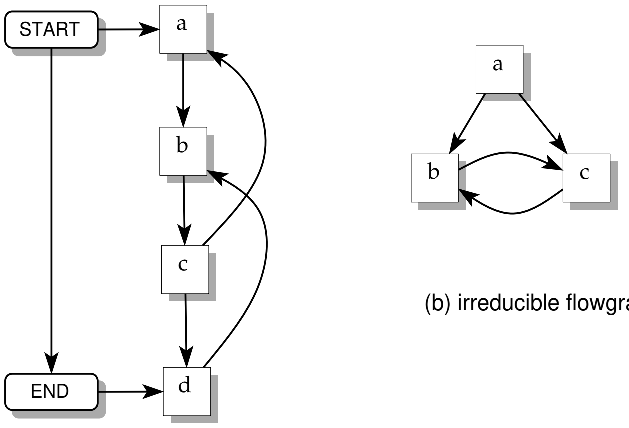
[3] Note that we can construct the dominator tree of a flowgraph in linear time [Har85].

The above definition of reducibility applies equally to both DJ graphs and flowgraphs. In other words, we can easily see that a DJ graph is reducible if and only if the corresponding flowgraph is also reducible.

Now to partition a flowgraph into forward and back edges we can perform a depth-first search and identify sp-back edges with respect to a depth-first spanning tree. For reducible flowgraph the destination node a sp-back edge will dominate the source node. Whereas in irreducible flowgraph the destination node of a sp-back edge will not dominate the source node.

In a reducible flowgraph the destination node h of a back edge $x \rightarrow h$ is called the *loop header* or *loop entry* node. Given this we can define a *natural loop* with loop entry h to be the set of nodes that can reach x without going through d. From this we can see that the loop entry node will dominate every node in the loop.

We can extend the above notion of loops to irreducible flowgraph. Recall that if the destination node of a sp-back edge does not dominate the source node then we have irreducibility. We can think of the destination nodes of such sp-back edges to be a loop entry node of an irreducible loop. But, unlike in the case of reducible loops, loop entry nodes in an irreducible loop are not unique. Different depth-first numbering will give rise to different sp-back edges, all of whose destination nodes will be potential entry nodes of loops. Therefore, in an irreducible loop there can be more than one entry node, and the entry nodes *collectively* dominate the set of nodes in the loop. We can think of an irreducible loop to be a multiple entry strong connected region such that the entry nodes collectively dominate the strongly connected region.



(b) irreducible flowgraph

(a) reducible flowgraph

Figure 2: Examples of reducible and irreducible flowgraphs

One classical approach to identifying loops is based on Tarjan's interval-finding algorithm. *Tarjan's intervals* are single-entry, strongly connected subgraphs, and they closely reflect the loop structures in programs [Tar74]. The basic idea behind Tarjan's method is to repeatedly *collapse* each loop into a single node inside-out until the whole graph reduces to one node. This idea will work if the flowgraph is reducible. Recall that a node h is a loop header if $x \to h$ is a back edge and h dominates x. Now as the reduction process proceeds in Tarjan's method, if we come across a back edge $y \to g$ such that g does not dominate y, further reduction of the graph cannot be continued. This is because g is not a unique loop header node. A number of actions can be take at this point: (1) split the header node and transform the graph to a reducible one [ASU86]; (2) abandon the reduction process and warn the programmer that the graph is not reducible; and (3) identify a single-entry region which encloses the irreducible portion, collapse the single-entry region into one node, and continue with the reduction [Bur90].

In this paper we take a different approach for reduction. Our approach uses DJ graphs. Translating the notion of reducibility on DJ graph, we can easily observe the following property.

Lemma 3.1 *A flowgraph is irreducible if and only if there exists a simple cycle in its DJ graph that does not contain a BJ edge (that is, the cycle is made of only D edges and CJ edges).*

Proof: Follows from the definition of DJ graphs and Definition 3.1. ∎

This lemma implies that if we perform a depth-first search on such a DJ graph, we can always find a potential back edge that is also a CJ edge. In other words, not every sp-back edge is a BJ edge. This gives us a simple way to detect irreducibility.

Example: Consider the flowgraph shown in Figure 1(a) whose corresponding DJ graph shown in Figure 1(b). The flowgraph is irreducible because of a multiple-entry loop that has two entry nodes C and D. The DJ graph contains a simple cycle $C \to D \to F \to H \to C$, in which there are no BJ edges.

Now that we can detect the existence of an irreducible loop, how are we to find its loop body? The following lemma states that all the "entry nodes" of an irreducible loop have the same immediate dominator.

Lemma 3.2 *All the entry nodes of an irreducible loop have the same immediate dominator.*

Proof: Let $x_1, x_2, \ldots, x_n$ be the set of loop entry nodes of an irreducible loop. By definition of loop, there exists a cycle C, such that $x_1, x_2, \ldots, x_n$ are in C. Let $y = idom(x_i)$ and let $z = idom(x_j)$, for some $i \neq j \in \{1, 2, \ldots n\}$. We want to show $y = z$. Suppose that $y \neq z$. Then there is a path from START $\xrightarrow{*} z \xrightarrow{+} x_j \xrightarrow{+} x_i$ that does not pass through y, contradicting $y = idom(x_i)$. Therefore, y must be the same as z. ∎

What the above lemma implies is that when we are looking for candidate nodes that belong to a loop, we can look only at nodes whose levels are equal to or greater than that of the

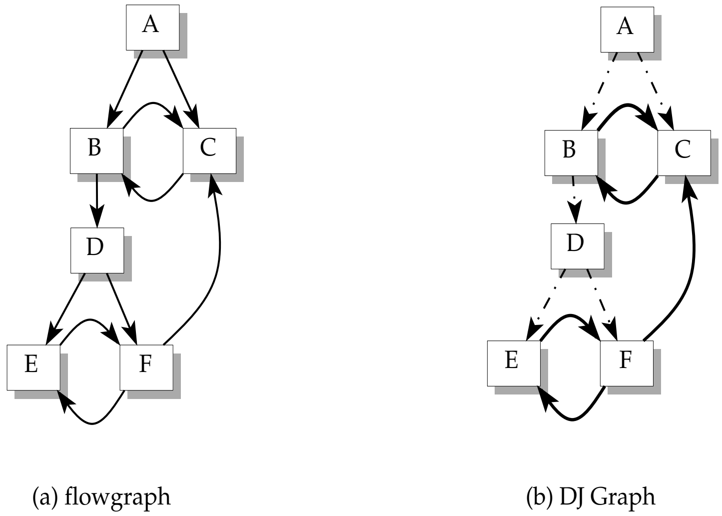


Figure 3: An irreducible flowgraph with two irreducible loops

loop entry nodes on the DJ graph. This is obvious for a reducible loop, since its unique entry node, the loop header, strictly dominates (or has a smaller level number than) any other nodes in the loop. For an irreducible loop all the entry nodes of the loop are at the same level (follows from Lemma 3.2). Therefore, we can identify an irreducible loop with entry nodes at a certain level by determining the Strongly Connected Component (SCC) while considering only nodes whose level number is equal to or greater than the current level. Once we find such a SCC, collapse the whole component into one node. It is also important to emphasize that using this technique we can identify reducible loops nested inside an irreducible loop, as well as irreducible loops nested within a reducible loops. For the flowgraph shown in Figure 1, using our approach we can say that the reducible loop consisting of nodes $\{D, G\}$ is nested within the irreducible loop consisting of nodes $\{C, L_D, G\}, E, F, H\}$, where L_D is the loop consisting of $\{D, G\}$.

Also, the level information allows us to detect finer irreducible regions. This is illustrated in Figure 3. For example, using our algorithm we can detect two irreducible loops instead of one for the flowgraph shown in Figure 3. The two loops are: $L1 = \{E, F\}$, and $L2 = \{B, C, D, L1\}$.

4 Our Algorithm

Our loop-finding algorithm performs a depth-first search on the DJ graph. With respect to the derived depth-first spanning tree, it identifies all **sp-back** edges. With the sp-back edges it can discover loops in a bottom-up fashion on the DJ graph. The complete algorithm is given below.

Algorithm 4.1 *The following algorithm identifies both reducible and irreducible loops.*

MainLoop()

{

1: Perform a depth-first search on the DJ graph and identify sp-back edges;
2: **for**($i = NumLevel - 1$ **downto** 0) /* visit nodes in a bottom-up fashion */
3: $Irreducible = False$;
4:   **foreach** node n with $n.level = i$ **do**
5:   **foreach** incoming edge $m \rightarrow n$ **do**
6:   **if** $m \rightarrow n$ is both a CJ edge **and** an sp-back edge **then**
7: $Irreducible = True$; `/* n is in an irreducible loop */`
8:   **endif**
9:   **if** $m \rightarrow n$ is a BJ edge **then**
10:   Find $ReachUnder(n)$ for all the BJ edges $m_1 \rightarrow n, \ldots, m_k \rightarrow n$;
11:   Collapse the loop consisting of nodes $\{n\} \cup ReachUnder(n)$;
12:   **endif**
13:   **endfor**
14:   **endfor**
15:   **if**($Irreducible$) `/* there exists an irreducible loop */`
16:   Identify SCCs for the subgraphs induced by nodes at level $\geq i$;
17:   Collapse each nontrivial SCC to a single node;
18:   **endif**
19: **endfor**

}

The algorithm visits all the nodes level by level in a bottom-up fashion. At every level, when there is a reducible loop whose header is at this level, we identify it by checking to see if that node is the destination node of any sp-back edge. If yes, we know a potential reducible loop is found. The algorithm goes to find the loop body with that node being the loop header, and collapses the loop – between step $\boxed{\mathbf{9}}$ and step $\boxed{\mathbf{12}}$.[4] The procedure $ReachUnder(n)$ will find all the nodes that can reach the source nodes of sp-back edges incident on the loop header n, without going through n. We collapse the nodes in $\{n\} \cup ReachUnder(n)$ using Tarjan's set-union data structure, and these nodes form the body of the loop.

Note that all the sp-back edges with the same destination node are sp-back edges for the same loop. The algorithm also checks to see if that node could be an entry node of an irreducible loop. If yes, turns on the *Irreducible* flag at step $\boxed{\mathbf{7}}$—indicating there are some irreducible loops that need to be handled later from step $\boxed{\mathbf{15}}$ to step $\boxed{\mathbf{18}}$. Once we are done with the above process for every node at the level, we check to see if there is any irreducible loop with its entry nodes at this level at step $\boxed{\mathbf{15}}$. Notice that the flag *Irreducible* is set at step $\boxed{\mathbf{7}}$ to *true* if irreducibility is detected. If yes, we use SCC's to identify the loop body for

[4] By "collapse" we mean that a loop body is condensed and becomes a single node. Any edge incoming into the loop from outside the loop will become an edge incoming into the representative node. Any edge outgoing from the loop to outside the loop will become an edge outgoing from the representative node.

an irreducible loop and collapse it. Also note that at this point we have collapsed any reducible loop whose header is at this level or beyond, so have any irreducible loop whose entry nodes are beyond this level.

For a loop nest consisting of L_1, L_2, L_3 and L_4, where L_1 and L_3 are reducible while L_2 and L_4 are irreducible, we can identify the loop body for all four loops. However, when L_1 and L_4 are reducible while L_2 and L_3 are irreducible, we will only identify three loops by merging L_2 and L_3 into one larger irreducible loop. That is, *immediately nested* irreducible loops will not be distinguished. In reality, it is hard to imagine the existence of such a loop nest. The advantage of our method is therefore to expose maximally the nesting structure of a loop nest.

Lemma 4.1 Algorithm 4.1 *correctly identifies both reducible and irreducible loops in a program.*

Proof: The proof is based on induction on levels of nodes in the DJ graph.

Base case: The level number is the maximum. It is trivial if sp-back edges are also BJ edges. But if the sp-back edge is also a CJ edge then *Irreducible* is true, and the nodes in a SCC will be collapsed at step $\boxed{\mathbf{17}}$.

Induction: Assume that the assertion is true for all levels $\geq k+1$. We will then show that the lemma is true for level k. At level k when we detect irreducibility (i.e., sp-back edge is the same as CJ edge), we mark *Irreducible* to be *True*. The procedure for $ReachUnder(n)$ will find all nodes that can reach the source node of BJ edges incident on node n, without going through node n. At step $\boxed{\mathbf{11}}$ we will collapse all such nodes. If *Irreducible* is true then Tarjan's SCC algorithm will be used to identify the loop body of irreducible loops. In both cases, loops with their headers at level k are collapsed. Hence the result. ∎

Finally we analyze the complexity of the algorithm. When a flowgraph is reducible, our algorithm has the same time complexity $O(|E| \times \alpha(|E|, |N|))$ as Tarjan's approach [Tar81]. The complexity of Tarjan's algorithm is dominated by the time taken to collapse the nodes of a loop. Tarjan uses balanced path compression to maintain the set of nodes in a collapsed loop, and this takes $O(|E| \times \alpha(|E|, |N|))$, where $\alpha()$ is the inverse Ackermann function.

For irreducible flowgraphs, the worst-case time complexity of the algorithm occurs when it needs to find irreducible loops at every level. Assume k is the number of levels in the DJ graph. Then the time complexity of the algorithm is $O(|E| \times \alpha(|E|, |N|) + k \times |E|)$, since finding strongly connected components needs $O(|E|)$ time. We anticipate in practice, k is a constant, so the algorithm is almost linear.

Example: We use Figure 4 to illustrate the application of our algorithm to the example DJ graph in Figure 1(b). Figure 4(a) gives the result of a depth-first search; the nodes are now labeled with respect to this search. In addition, three J edges $4 \rightarrow 3$, $5 \rightarrow 1$, and $7 \rightarrow 3$ are classified to be sp-back edges. In the figure, sp-back edges are shown with 'bubbles' at the source end of the edge.

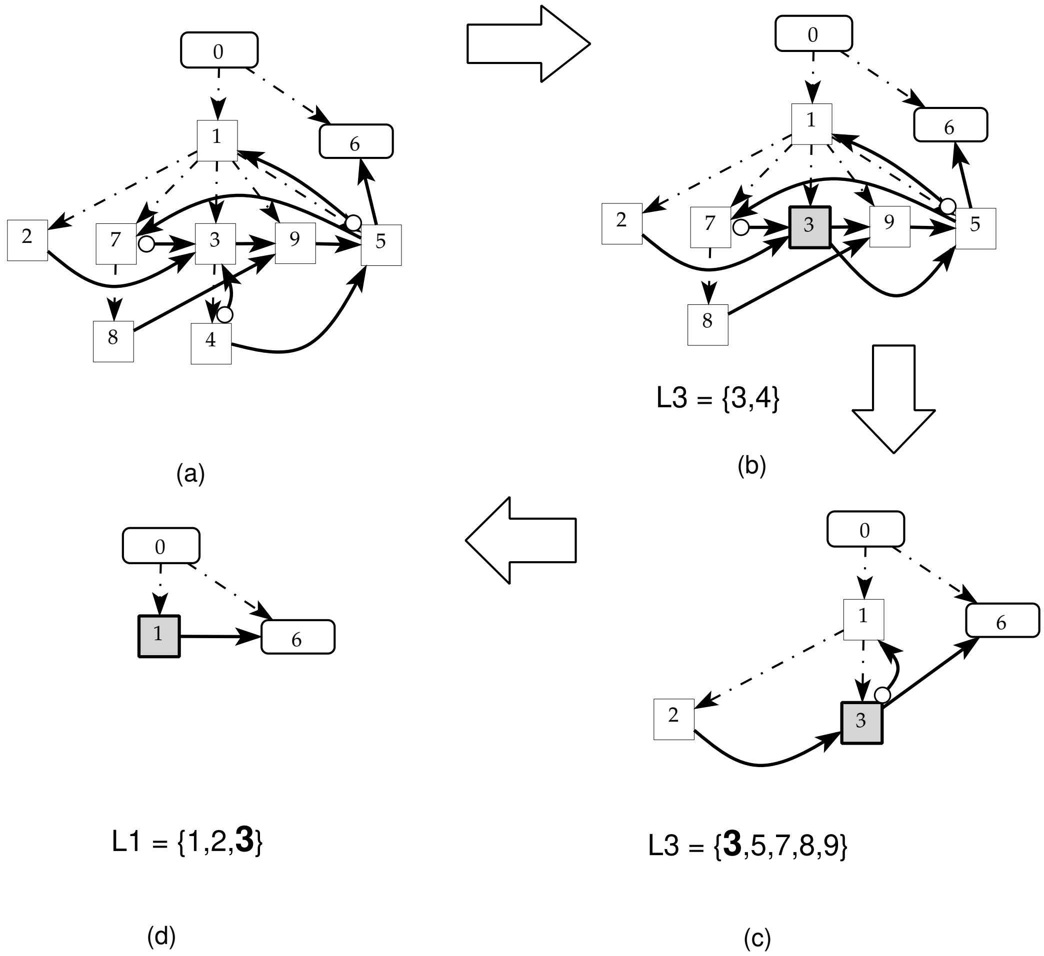


Figure 4: A trace of the loop identification algorithm. Nodes are numbered in a depth-first order. In the figures, edges with 'bubbles' are sp-back edges. Finally, high-lighted nodes are collapsed loops.

When the algorithm processes the nodes at level 3 (i.e., nodes 4 and 8), nothing interesting happens. It then moves on to process the nodes at level 2. When it visits node 3, it finds the incoming edge $4 \rightarrow 3$ to be a BJ edge. Consequently, the algorithm identifies and then collapses a reducible loop consisting of nodes 3 and 4. This is illustrated by the transition from (a) to (b). In Figure 4(b), the loop is represented by a node whose label is the same as the header's label.

Since another incoming edge $7 \rightarrow 3$ of node 3 is both a CJ edge and an sp-back edge, the $Irreducible$ flag is made $True$. After all the nodes at level 2 are processed, the algorithm performs Tarjan's SCC algorithm on nodes at levels 2 and 3 (i.e., nodes 2, 3, 5, 7, 8, and 9 in Figure 4(b)). A nontrivial SCC consisting of nodes 3, 5, 7, 8, and 9 is identified and then collapsed. This is illustrated by the transition from (b) to (c). Node 3 in Figure 4(c) represents this irreducible loop.[5]

When the algorithm visits nodes at level 1, a reducible loop with node 1 as its header is identified and collapsed. This is illustrated by the transition from (c) to (d). After that, nothing interesting happens. The algorithm terminates after having visited nodes 6 (at level 1) and 0 (at level 0).

As noted in the previous section, our algorithm identify two irreducible loops for the flowgraph shown in Figure 3(a), whose corresponding DJ graph is shown in Figure 3(b). The two loops identified are the inner irreducible loop $L1 = \{E, F\}$ and the outer irreducible loop $L2 = \{B, C, D, L1\}$.

5 Conclusion and Related Work

Identifying loops has been a classical exercise in control flow analysis. We have shown how to use DJ graphs for identifying both reducible and irreducible loops by extending Tarjan's approach [Tar74]. One feature of our algorithm for identifying irreducible loops is that it utilizes level information in DJ graphs to discover the body of irreducible loops. In [Sre95, SGL95a] we propose a new elimination-based data flow analysis that uses DJ graphs for graph reduction and variable elimination. Generally, elimination-based methods are applicable only to reducible flowgraphs. In [Sre95, SGL95a] we have used some of the key results presented in this paper for handling irreducible flowgraphs during the reduction and elimination process.

Many methods have been proposed in the context of elimination based data flow analysis for handling irreducibility such as node splitting [Hec77], identifying single-entry region that encloses the irreducible region [Bur90, SS79], etc. In a recent technical report, Steensgaard proposed a method for identifying nested loops, both reducible and irreducible [Ste93]. His method consists of first applying Tarjan's SCC algorithm to the whole graph and identifying each nontrivial SCC [Ste93]. He then identifies, for each non-trivial component, what he calls as *generalized entry nodes.* A node y in an SCC S is a generalized entry node of S if, in the original flowgraph, there is an edge from $x \to y$ such that $x \notin S$. Now if $z \to y$ is an edge in the original flowgraph such that $z \in S$, then he calls $z \to y$ as a *generalized back edge.* Once he identifies generalized back edges in an SCC, he eliminates them from the SCC and applies Tarjan's SCC algorithm once again on the SCC and identifies "inner" SCCs. This way he identifies loops in an outside-in fashion of loop nests. To illustrate Steengaard's algorithm, consider the flowgraph shown in Figure 3. Using his algorithm we will find only one loop, consisting of node $\{B, C, D, E, F\}$, whereas using our algorithm we will identify two loops one nested within another (see Section 4).

[5]We can use the label of any entry node of an irreducible to be the label of the "collapsed" node.

In our method, we apply Tarjan's SCC algorithm in an inside-out fashion of loop nests—our bottom-up reduction order will conform to this inside-out order of loop nests. Also, we will apply Tarjan's SCC algorithm only if we detect that there is an irreducible loop at a particular level. Therefore, the time complexity of our approach is better than Steensgaard's approach. The worst-case time of Steensgaard's algorithm can be quadratic in terms of the loop nesting depth.

We conclude our paper with the following open questions. One open question is: in the presence of irreducibility, can we define the notion of nested irreducible loop nests similar to nested reducible loopnests? If so, is it possible to construct an (efficient) algorithm for identifying such loop nests ? Answers to these questions remain to be our future work.

Acknowledgement

The first and second authors thank the National Sciences and Engineering Research Council (NSERC) and the Canadian Centers of Excellence (IRIS) for their continued support of this research. The last author would like to thank Vivek Sarkar for his management support. We also thank Bjarne Steensgaard for his in-depth comments on an earlier draft of the paper.

References

[ASU86] Aho, A. V., Sethi, R., and Ullman, J. D. *Compilers–Principles, Techniques, and Tools.* Addison-Wesley Publishing Co., 1986.

[Bur90] Burke, M. An interval-based approach to exhaustive and incremental interprocedural data-flow analysis. *ACM Transactions on Programming Languages and Systems*, 12(3):341–395, July 1990.

[Har85] Harel, D. A linear time algorithm for finding dominators in flow graphs and related problems. In *Symposium on Theory of Computing.* ACM, May 1985.

[Hec77] Hecht, M. S. *Flow Analysis of Computer Programs.* Elsevier North-Holland, Inc., 1977.

[HU74] Hecht, M. S., and Ullman, J. D. Characterizations of reducible flow graphs. *Journal of the ACM*, 21(3):367–375, July 1974.

[SG95] Sreedhar, V. C., and Gao, G. R. A linear time algorithm for placing ϕ-nodes. In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, January 1995. A longer version to appear in Journal of Programming Languages.

[SGL95a] SREEDHAR, V. C., GAO, G. R., AND LEE, Y. Efficient data flow analysis using DJ graphs: Elimination methods revisited. Technical Report ACAPS Memo 93, McGill University, June 1995.

[SGL95b] SREEDHAR, V. C., GAO, G. R., AND LEE, Y. Incremental computation of dominator trees. In *ACM SIGPLAN Workshop on Intermediate Representation (in conjunction with the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages)*, January 1995. Also appears in SIGPLAN Notices Volume 30, Number 4, 1995.

[Sre95] SREEDHAR, V. C. *Efficient Program Analyses Using DJ Graphs*. PhD thesis, School of Computer Science, McGill University, 1995.

[SS79] SCHWARTZ, J. T., AND SHARIR, M. A design for optimizations of the bitvectoring class. Technical report, Courant Institute of Mathematical Sciences, New York University, September 1979. Courant Computer Science Report No. 17.

[Ste93] STEENSGARD, B. Sequentializing program dependence graphs for irreducible programs. Technical Report MSR-TR-93-14, Microsoft Research, October 1993.

[Tar74] TARJAN, R. E. Testing flow graph reducibility. *Journal of Computer and System Sciences*, 9:355–365, 1974.

[Tar81] TARJAN, R. E. Fast algorithms for solving path problems. *Journal of the ACM*, 28(3):594–614, July 1981.

[Wol92] WOLFE, M. Flow graph anomalies: What's in a loop? Technical Report CS/E 92-012, Oregon Graduate Institute for Science and Technology, 1992.