Redundancy Elimination Algorithms & Their Limitations

Rahib B

Assistant Professor
Department of Computer Science
K.A.H.M. Unity Women's College, Manjeri
Malappuram, Kerala – 676122

***Abstract* - Global Value Numbering (GVN) is a compiler optimization technique for detecting equivalent expressions in a program. The GVN keeps the value numbering information which is useful for the optimization performed like redundant expressions, common sub expression elimination and also the redundant load and store. In this paper, an attempt has been made to discuss various redundancy elimination algorithms used for common subexpression elimination, and their limitations too.**

Keywords: **value graph, Global Value Numbering, redundancy detection.**

Published in IJIRMPS (E-ISSN: 2349-7300), Volume 2, Issue 3, May-June 2014

I. INTRODUCTION

Two expressions are said to be equivalent if we can statically determine that both the expressions will have the same value during execution. Global Value Numbering (GVN) is a program analysis that categorizes expressions in the program that compute the same static value. GVN works by assigning a value number to variables and expressions. The same value number is assigned to those variables and expressions which are provably equivalent. GVN partitions expressions and variables into classes (or assigns them a unique value number) all of which have the same static value.

An initial attempt on GVN, by Gary.A. Kildall, computes and detects all Herbrand equivalences in non-SSA form of program expressions using the powerful concept of structuring, but its time complexity is exponential [1]. This concept was introduced in 1973, and in about 15 years, there have been efforts in developing efficient redundancy elimination algorithms by using different programs and data structures.

II. VARIOUS COMMON SUBEXPRESSION ELIMINATION ALGORITHMS

A. A unified approach to global program optimization by Kildall

Kildall's algorithm [1] performs an abstract interpretation over the lattice of sets of Herbrand equivalences. It represents the set of Herbrand equivalences at each program point by means of a structured partition pool. The join operation of two structured partitions is their intersection. The algorithm detects all Herbrand equivalences in non-SSA form of program expressions, but the complexity of Kildall's algorithm is exponential. And the algorithm did not prove any upper bound on the number of iterations needed for

achieving a fixed point. Fig.1(a) shows a value graph with 4 nodes, and Fig.1(b) includes the structured partitions after applying the algorithm.

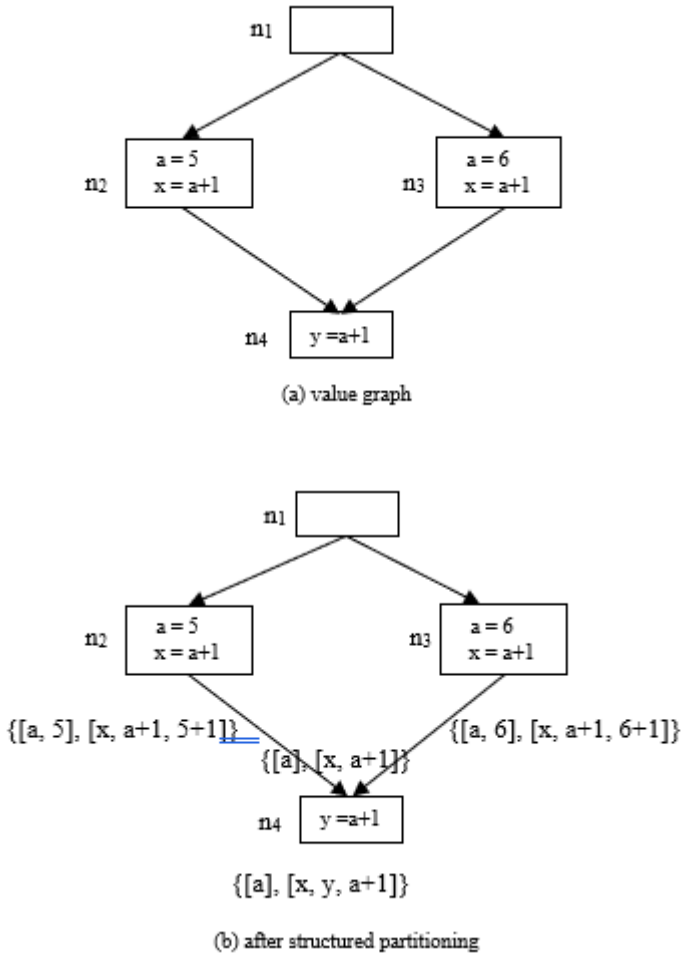
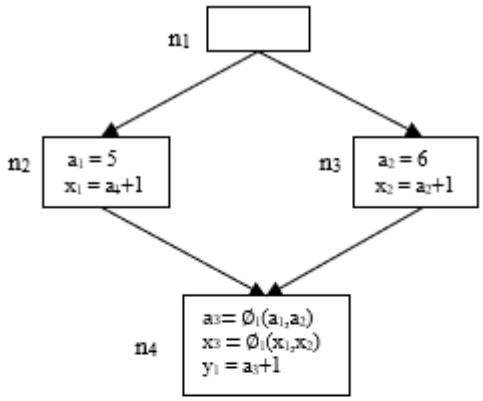


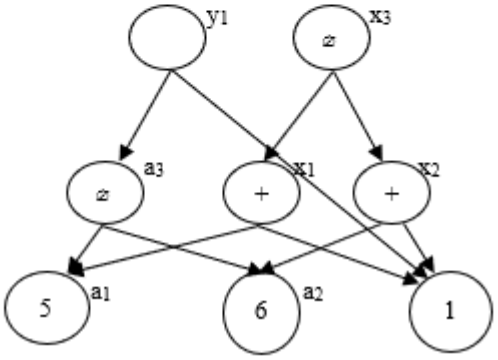
Fig.1. Kildall's algorithm

B. AWZ Algorithm by Alpern, Wegman and Zadeck

The AWZ algorithm works on value graphs in static single assignment (SSA) form of a program [2]. In an SSA representation, every variable of the original program are replaced by new versions such that every variable has a unique initialization point. A value graph can be represented by a collection of nodes of the form <V,t>, where V is a set of variables, and the type t is either ⊥, a constant c (indicating that the node has no successors), $F(\eta_1,\eta_2)$ or $\emptyset_i(\eta_1,\eta_2)$ (indicating that the node has two ordered successors η_1 and η_2). $\emptyset_i$ denotes the $\emptyset$ function associated with the i^{th} join point in the value graph. The core of the AWZ algorithm is to use congruence partitioning to merge some nodes of the value graph. Consider the following Fig.2. Fig.2(a) is the SSA form of Fig.1(a), and Fig.2(b) is the collapsed value graph after applying congruence partitioning.



(a) the program in SSA form



(b) the collapsed value graph after congruence partitioning.

Fig.2. AWZ algorithm,

From Fig.2(b), it s obvious that the equivalence of x and y at the exit of node 4 is not detected by the AWZ algorithm, because the algorithm treats the Ø-operators like ordinary operators.

Hence the AWZ algorithm cannot discover all equivalences among the program terms. The main weakness of the AWZ algorithm is a consequence of treating Ø-operators like ordinary operators.

C. KRS Algorithm by Knoop, Ruthing, and Steffen

The KRS algorithm is an extension of AWZ algorithm [3].

It tries to capture the semantics of Ø functions by applying the following two rules. For expressions e, e_1, e_2, e_3 and e_4, and the operator ω and the Ø function:

$$\forall e: \emptyset_i(e,e) = e \qquad (1)$$

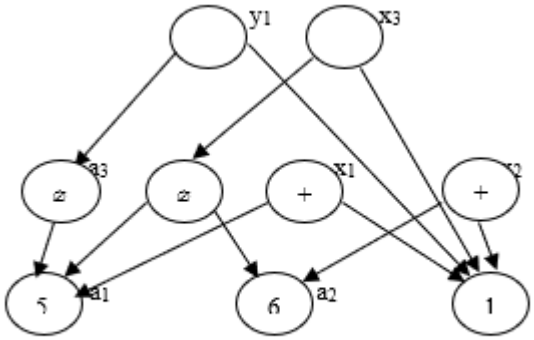
$$\forall e_1,e_2,e_3,e_4: \emptyset_i(e_1 \,\omega\, e_2 , e_3 \,\omega\, e_4) = \emptyset_i(e_1,e_3) \,\omega\, \emptyset_i(e_2,e_4) \qquad (2)$$

Rule (1) eliminates unnecessary Ø-operators which can either occur as the result of the partitioning process or of applications of Rule(2). Rule (1) is applicable whenever a node n with Ø-operator is present whose operands refer to the same node m. In this case any edge pointing to n is redirected to m, the variable annotations of n are added to m, and finally, node n is eliminated.

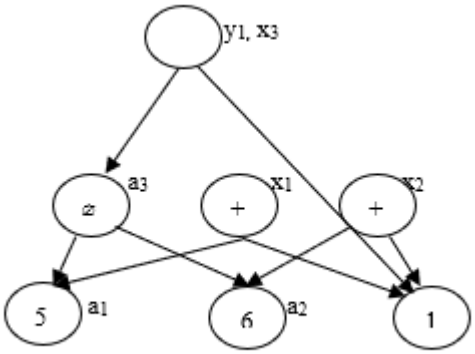
Rule (2) is applicable if there is a node n with a Ø-operator whose both operands have the same ordinary operator label, ω, at top-level. Moreover, n must not be strictly followed by an anonymous node. The pattern is then modified as displayed on the right-hand side of Rule (2). Two new nodes labeled with the Ø-operator

of n are introduced and connected with the successor nodes of n. Node n gets ω as its operator label. And the outgoing edges of n are redirected to the new nodes.

Starting with the collapsed value graph of Fig.2(b), application of Rule (2) followed by Rule (1) results in the value graph of Fig.3(a). A successive partitioning step leads to the value graph of Fig.3(b) in which x and y are equivalent.



(a) after the application of the rules.



(b) after a further partitioning step.

Fig.3. the collapsed value graph after RKS algorithm.

The RKS algorithm assumes that all assignments are of the form c = a ω b to make sure that for all original nodes n in the value graph, Vars(n) ≠ ∅. This precondition is necessary in arguing termination for this system of rewrite rules, and proving the polynomial complexity bound. The RKS algorithm alternately applies the AWZ algorithm and the two rewrite rules until the value graph reaches a fixed point. Thus, the RKS algorithm discovers more equivalences than the AWZ algorithm.

But the RKS algorithm cannot discover all equivalences even in acyclic programs too. This is because the precondition can prevent two equal expressions from reaching the same normal form.

D. A polynomial-time algorithm for GVN by Sumit Gulwani and G.C. Necula.

Sumit Gulwani presented an algorithm called “A polynomial-time algorithm for global value numbering “ for common subexpression elimination [4]. This algorithm is based on abstract interpretation. The algorithm states that the number of iterations required for reaching fixed-point is bounded by the number of variables live at any point in the program, and it avoids the problem of exponential sized representation for equivalences by using a data structure called Strong Equivalence Directed (SED) acyclic graph for representing the structured partitions of Kildall, and a more sophisticated join algorithm.

Gulwani claims that his algorithm detects all Herbrand equivalences that Kildall detects, and its time complexity is polynomial rather exponential though, Gulwani's algorithm fails to detect all equivalent expressions that Kildall detects. Consider Fig.4 which fails to detect the redundant expression x+y at the last node.

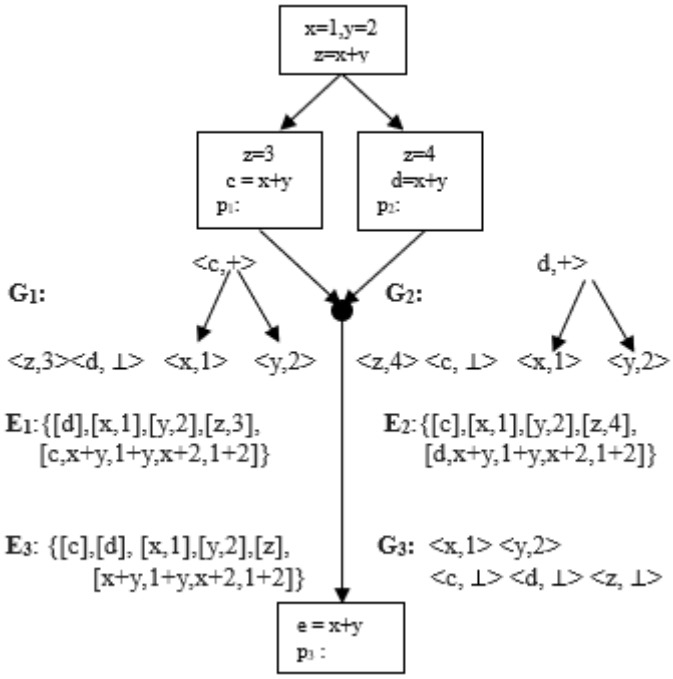


Fig.4 Join of SEDs for program point p_i, G_i is the SED that Gulwani computes and E_i is the optimizing pool that Kildall computes.

Fig.4. shows G_1 and G_2 are the SEDs and E_1 and E_2 are the structured partitions at program points p_1 and p_2 that Gulwani and Kildall compute. G_3 is the SED resulting after the join of the SEDs G_1 and G_2, and corresponding partition in Kildall's is E_3, which is the result of the confluence of E_1 and E_2. It is obvious that the expression x + y in the last node is redundant. As x + y is present in E_3, Kildall detects this redundancy. Gulwani does not detect the redundancy of x + y in the last node, because his Join algorithm computes intersection of only those SED nodes having at least one common variable. This reveals that the algorithm by Gulwani doesn't detect all Herbrand equivalences that Kildall detects, even though he claims so.

III. CONCLUSION

AWZ algorithm is an efficient algorithm for GVN, but is not as precise as Kildall's. The AWZ algorithm fails to detect the equivalences shown in Fig.2. RKS algorithm cannot discover all equivalences even in acyclic programs as the precondition prevents two equal expressions from reaching the same normal form. The algorithm given by Gulwani does intersection of only those classes having at least one common variable. But according to the Fig.4., the intersection of all pairs of classes is needed for detecting all sort of equivalences.

IV. ACKNOWLEDGMENT

I remain immensely obliged to Dr. Vineeth Kumar Paleri, Professor, Department of Computer Science, NIT Calicut, Kerala, India, for his sincere, unconditional and constant guidance for doing research, and I thank University Grants Commission too, for awarding Teacher Fellowship for doing M.Phil. in Wireless Sensor Networks under the Faculty Development Program of the UGC in 2012.

References

[1] Gary.A. Kildall. “A unified approach to global program optimization”. In 1st ACM Symposium on Principles of Programming Languages, pages 194–206, 1973.

[2] B. Alpern, M. N. Wegman, and F. K. Zadeck. “Detecting equality of variables in programs”. In 15th ACM Symposium on Principles of Programming Languages, pages 1–11, 1988.

[3] O. Ruthing, J. Knoop, and B. Steffen. “Detecting equality of variables: Combining efficiency with precision”. In 6th International Symposium on Static Analysis, pages 232–247, 1999.

[4] S. Gulwani , and G.C. Necula. “A polynomial time Algorithm for global value numbering”. Science of Computer Programming, 64(1):97–114, 2007.