A Note on “A polynomial-time algorithm for global value numbering”

Saleena Nabeezath and Vineeth Paleri

Department of Computer Science and Engineering
National Institute of Technology Calicut, India.
{saleena,vpaleri}@nitc.ac.in

Abstract

A Global Value Numbering (GVN) algorithm is considered to be complete (or precise), if it can detect all Herbrand equivalences among expressions in a program. A polynomial time algorithm for GVN is presented by Gulwani and Necula (2006). Here we present two problems with this algorithm that prevents detection of some of the Herbrand equivalences among program expressions. We suggest improvements that will make the algorithm more precise.

1 Introduction

Global Value Numbering (GVN) is a method for detecting equivalence among expressions in a program. A Global Value Numbering (GVN) algorithm is considered to be complete (or precise), if it can detect all Herbrand equivalences among program expressions. Two expressions are said to be Herbrand equivalent (or transparent equivalent), if they are computed by the same operator applied to equivalent operands [3, 5, 6].

Kildall’s GVN algorithm [4] is complete in detecting all Herbrand equivalences among program expressions. Gulwani and Necula [3] present a polynomial time algorithm for GVN. This uses a data structure called Strong Equivalence Dag (SED) for representing the structured partitions of Kildall [4]. We have observed two problems with this algorithm that prevents detection of some of the Herbrand equivalences (among program expressions) that Kildall detects. In the next section, we present two examples to demonstrate the problems. We suggest possible improvements that will make the algorithm more precise.
2 GVN algorithm by Gulwani and Necula\cite{3}

2.1 Problem 1: Join algorithm

Figure 1 shows four program nodes and a join point\cite{4}. $G_1$ and $G_2$ are the SEDs at program points $p_1$ and $p_2$ respectively. $E_1$ and $E_2$ are the structured partitions that Kildall\cite{4} computes at these points. $G_3$ is the SED resulting after the join of the SEDs $G_1$ and $G_2$. The corresponding partition in Kildall\cite{4} is $E_3$, which is the result of the meet of $E_1$ and $E_2$.

As per the definition for Herbrand equivalence of expressions given by Ruthing, Knoop, and Steffen\cite{6}(see the definition at the end of section 2), the expression $x + y$ in the topmost node is herbrand equivalent to the expression $x + y$ in the bottommost node. Since $x + y$ is present in $E_3$, using Kildall’s algorithm,\cite{4} we can deduce the information that whenever control reaches $p_3$, an expression equivalent to $x + y$ is already computed. But there is no way to deduce this information from the corresponding SED $G_3$. Hence the GVN algorithm by Gulwani and Necula\cite{3} fails in detecting the herbrand equivalence in this example.

\footnote{For convenience, we use $x + y$ instead of $F(x, y)$}
2.1.1 A solution

At a join point, the meet operation in Kildall does intersection of every pair of classes that have at least one common expression, whereas the Join algorithm in [3] computes intersection of only those SED nodes having at least one common variable (see line 3 of the Join algorithm: for each variable $x \in T$ \dots \textbf{Intersect}($\text{Node}_{C_1}(x)$, $\text{Node}_{C_2}(x)$)). Hence, a solution that will enable the algorithm to detect these kinds of equivalences is to modify the Join algorithm in such a way that, it computes the intersection of every pair of nodes in the two SEDs. In Figure 2 SED $G_3$ shows the result of computing \textit{Join} using the proposed method. The intersection of $\langle c, + \rangle$ in $G_1$ and $\langle d, + \rangle$ in $G_2$ results in the node $\langle \phi, + \rangle$ in $G_3$, which represents $x + y$ and its equivalent expressions. It may be noted that nodes like $\langle \phi, + \rangle$, having empty set of variables are considered unnecessary by Gulwani and Necula [3]. But in fact these are necessary (as will be shown in the next section) and hence the proposed method will retain such nodes.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{fig2}
\caption{Join of SEDs: pairwise intersection of nodes}
\end{figure}

2.2 Problem 2: Removal of SED nodes

Figure 3 shows a basic block in a program with the SEDs $G_1$ and $G_2$ at program points $p_1$ and $p_2$ respectively. Here the expressions $x + y$ and the two occurrences of $a + b$ are equivalent and this equivalence will be detected by Kildall’s algorithm (and also the local value numbering algorithm [2]). But it goes undetected in Gulwani and Necula [3] because of the following reasons.

In section 3.1 of Gulwani and Necula [3], it is stated that the transfer functions may yield SEDs with unnecessary nodes, and these unnecessary nodes may be removed (a node is considered unnecessary when all its ancestor nodes or all
From the above example, it is clear that the problem is due to the removal of some necessary nodes, which the algorithm considers as unnecessary. The simple solution is to retain all such nodes. In that case, for the above example, the SED reaching the input point of \( d := a + b \) will have a node representing the expression \( a + b \), indicating that an expression equivalent to it is already computed.

### 2.3 GVN for code optimization

In fact the GVN algorithm by Kildall was formulated with the aim of detecting common sub expressions. An optimization using this algorithm will subsume local value numbering also. The first example shown is an instance of the classical common sub expression elimination and the second is an instance of local value numbering. Hence the suggested modifications are necessary to make use of the
GVN algorithm by Gulwani and Necula [3] in compiler code optimization.

3 Conclusion

To the best of our knowledge, Kildall’s is the only GVN algorithm that is complete in detecting all herbrand equivalences among program expressions. It is already proved by Gulwani and Necula [3] that the GVN algorithm by Alpern, Wegman, and Zadeck [1] and that by Ruthing, Knoop and Steffen [6] are incomplete. It is stated in [6] that their algorithm is restricted to the equality problem of variables. We observe that the same is the case with Gulwani and Necula [3] (and also with Nie and Cheng [7]). The suggested modifications are required for the general problem of detecting equivalence among program expressions.

References

[1] B. Alpern, M. N. Wegman, and F. K. Zadeck. Detecting Equality of Variables in Programs, In Proceedings of the 15th ACM Symposium on Principles of Programming Languages, pages 1-11, January 1988.

[2] Appel A. W. Modern Compiler Implementation in Java, Cambridge University Press, 2000.

[3] Sumit Gulwani and George C Necula. A polynomial time Algorithm for Global Value Numbering, Science of Computer Programming, 64(1):97-114, January 2007.

[4] Gary A Kildall. A Unified Approach to Global Program Optimization, ACM Symposium on Principles of Programming Languages, 194-206, 1973.

[5] B. K. Rosen, M. N. Wegman, and F. K. Zadeck. Global Value Numbers and Redundant Computations, In Proceedings of the 15th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, , pages 12-27, January 1988.

[6] Oliver Ruthing, Jens Knoop, and Bernhard Steffen. Detecting Equality of Variables: Combining Efficiency with Precision, In Proceedings of the 6th International Symposium on Static Analysis, pages 232-247, September 1999.

[7] Jiu-Tao Nie and Xu Cheng. An efficient SSA-based algorithm for Complete Global Value Numbering, In Proceedings of APLAS 2007, LNCS 4807, pages 319-334, 2007.