1974

Impurities Found in Algorithm Implementations

Necdet Bulnut

Maurice H. Halstead

Report Number:
74-111

CSD 2YR 111

technical contributions

IMPURITIES FOUND IN ALGORITHM IMPLEMENTATIONS

Necdet Bulut
Maurice H. Halstead

Department of Computer Science
Purdue University
West Lafayette, Indiana 47907

Recent studies in the field of Algorithm Dynamics or Software Physics have provided what appears to be a useful insight into both the use and design of programming languages, and into the programming process itself. (Halstead [3, 4, 5, 6, 7], Bayer [1], Bulut [2], Halstead and Bayer [8], Halstead and Zislis [9], Zislis [10], and Zweben [11]).

In this line of the research, a functional relation[1] between the length of an algorithm, N, and the number of unique operators, η_1 , and operands, η_2, it contains has been shown to agree with the experimental data for a range of programming languages. It was also shown that, for a group of algorithms published by respected authors, the product of the volume and the language level was independent of the language in which they were expressed, when volume[2] was calculated in bits, and level[3] was calculated as a simple function of operators and operands [2,3,4].

At the same time, however, it was noted that algorithms written by students in a first semester programming course exhibited large deviations from the relationship which held for published programs.

As a result, it has been possible to classify some six categories of "impurities" which, if present in an algorithm, produce deviations from the relationships.

---

[1] $N = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$

[2] $V = N \log_2 (\eta_1 + \eta_2)$

[3] $L = \frac{2}{\eta_1} \frac{\eta_2}{N_2}$ where N_2 is the total number of operand occurrences

Applying a "purification" process to detect and eliminate occurrences of these six impurities showed that the published algorithms originally studied contained comparatively few occurrences of them, but that their removal actually improved the previously good agreement between the relationships and the data.

While the six impurity classes have been obtained solely on the basis of conformity to the software physics relationships, the observation that they are virtually absent from a sample of published programs which should represent "good programming", but highly frequent in samples which represent "poor programming" suggests that they may serve a useful purpose. This view is further strengthened by two other points. First, since it appears that the removal of the six impurities yields virtual agreement between theory and data, the set of six must be reasonably complete. Second, but perhaps more significantly, each of the six classes has some intuitive appeal.

Simply stated, the six impurity classes are :

Class 1 - Self Cancelling Operations :
When an operator and later its inverse are specified for the same operand such that one cancels the other, the usages of these operators and the operand involved in these operations create this impurity.

Class 2 - Ambiguous Usage of an Operand :
This impurity occurs when the same operand is used to represent two or more variables in an algorithm.

Class 3 - Synonymous Usages of Operands :
This impurity occurs when two or more operands are specified to represent the same variable in an algorithm.

Class 4 - Common Subexpressions :
This is the type of impurity which occurs when the same subexpression is used more than once without being replaced by a temporary variable.

Class 5 - Unnecessary Replacements :
If a subexpression is assigned to a temporary variable which is then used only once, introducing and using this temporary variable produces an impurity.

Class 6 - Unfactored Expressions :
Repetitive usages of operands and operators due to unfactored terms in an expression cause this impurity.

It is of further interest to note that while each of the classes can, in general, be easily avoided, some languages do not permit a complete elimination.

While removal of all recognized impurities tends to leave a program in a form which conforms to the practice of "good programmers", it does not in itself assure that even a small program has been expressed in the "neatest" way possible. It still may be possible to express it at a higher level, even in the computer language selected.

If one calculates the level, L, from the approximate relationship given in the last footnote, then it follows that the more operators which are used, the lower the level will be. Since a "GOTO L1" must be counted as a distinctly different operator from a "GOTO L2", a reduction in GOTO's will usually produce a higher level program.

References :

[1] BAYER, R., A Theoretical Study of Halstead's Software Phenomenon, CSD TR 69, Purdue University Dept. of Computer Science, May 1972.

[2] BULUT, N., Invariant Properties of Algorithms, Ph. D. Thesis, Purdue University, Dept. of Computer Science, August 1973.

[3] HALSTEAD, M. H., A Thermodynamics of Algorithms? CSD TR 66, Purdue University, Dept. of Computer Science, February 1972.

[4] HALSTEAD, M. H., "Natural Laws Controlling Algorithm Structures?", ACM SIGPLAN Notices, 7, 2, (February 1972), 19-26.

[5] HALSTEAD, M. H., A Theoretical Relationship between Mental Work and Machine Language Programming, CSD TR 67, Purdue University, Dept. of Computer Science, February 1972.

[6] HALSTEAD, M. H., "An Experimental Determination of the 'Purity' of a Trivial Algorithm" ACM SIGME : Performance Evaluation Review, 2, 1, (March 1973), 10-15.

[7] HALSTEAD, M. H. "Language Level, a Missing Concept in Information Theory", ACM SIGME : Performance Evaluation Review, 2, 1, (March 1973), 7-9.

[8] HALSTEAD, M. H. and BAYER, R., "Algorithm Dynamics", Proc. of ACM Annual Conf. 1973, Atlanta.

[9] HALSTEAD, M. H. and ZISLIS, P. M. Experimental Verification of Two Theorems of Software Physics, CSD TR 97, Purdue University, Dept. of Computer Science, June 1973.

[10] ZISLIS, P. M., An Experiment in Algorithm Implementation, CSD TR 96, Purdue University, Dept. of Computer Science, June 1973.

[11] ZWEBEN, S. H., Software Physics : Resolution of an Ambiguity in the Counting Procedure, CSD TR 93, Purdue University, Dept. of Computer Science, April, 1973.