ASSOCIATION FOR COMPUTING MACHINERY
ACM
FOUNDED 1947

SIGPLAN Notices

A Monthly Publication of the
SPECIAL INTEREST GROUP ON PROGRAMMING LANGUAGES

Contents: Volume 8, Number 6 1973 June

ABSTRACTS IN PROGRAMMING LANGUAGE-RELATED RESEARCH

Edited by
James B. Morris

University of California
Los Alamos Scientific Laboratory
Los Alamos, New Mexico 87544

THE DESIGN AND USE OF UNDERSTANDABLE, MINIMALLY ERROR-PRONE PROGRAMMING LANGUAGES

Project Name: The Design and Use of Understandable, Minimally Error-Prone Programming Languages

Name: E.J. Funk

Address: Department of Computer Science, Univ. of Toronto, Toronto M5S 1A7 CANADA

Introduction: The increasing sophistication of higher-level programming languages has relieved the programmer of the tedious bookkeeping which once was the source of most program errors, while simultaneously bringing to the fore a host of other difficulties formerly buried in the confusion. In particular, we note the following:

1. It is obvious that certain factors stemming from the way in which a programming language is used contribute to the "psychological complexity" of programs, and thereby to the ease with which they are comprehended and modified.
2. Certain language constructs, and certain ways in which a language is used due to its inherent characteristics, are more error-prone than others.
3. As the production of error-free software on the "first try" becomes an achievable goal, we wonder how much redundancy on the part of the programmer and the programming language will be necessary to make it a reality.

An integrated approach to the study of language and program complexity is the substance of this research.

Background: Blonski and Oldham of this Department have recently begun research centered around, respectively, the first and second problems mentioned above. The general "human factors" approach of Weinburg and others is, of course, intimately connected.

Blonski proposes twenty-two "program complexity factors" touching upon the following aspects of program construction: program flow; control flow; data flow; interaction between control flow and data flow. Blonski is planning a multi-factor experiment in which groups of students will be asked to explain, correct or evaluate programs possessing various combinations and levels of complexity factors. In this way the complexity measures will hopefully be validated, and concrete rules for writing understandable programs may be stated.

The primary difficulties which Blonski will encounter are twofold:

1. The combinatorial aspects of a multi-factor experiment, coupled with the expense (financial or educational) of conducting numerous trials on student groups, may preclude obtaining statistically valid results for even a very few complexity factors.
2. The general difficulty and unpredictable nature of psychological tests performed with student labor may further invalidate the results.

Oldham's proposed work is closely related. He intends to determine through examination of the program debugging process which language constructs are particularly error-prone and in what respects. He then plans to design a new language which elimates unnecessary and error-prone constructs, and incorporates grammatical redundancy at appropriate places in an effort to minimize the remaining errors.

Oldham's work will undoubtedly suffer from the second point mentioned above, and his problems will be compounded by the fact that impartial programmers are impossible to obtain; he will in all probability be unable to obtain an "objectively optimal" language from an error standpoint by observing student programmers at work.

Outline of Research: Briefly stated, the goal of this research is to integrate and extend the proposed work of Blonski and Oldham and at the same time to overcome the experimental limitations which, for all practical purposes, preclude the possibility that they will obtain meaningful results. In order to validate program complexity measures and to determine those program constructs in need of grammatical redundancy, large-scale psychological tests must be performed on a tremendous number of objective subjects. I plan to conduct my experiments on gerbils.

The numerous advantages of this approach should be, for the most part, immediately obvious:

1. Gerbils are inexpensive to obtain and maintain, and may be had in any desired quantity.
2. They are notoriously uniform and objective in their performance, eliminating the need to condition resluts on subject performance in other areas.
3. Techniques of psychological testing on gerbils are much more advanced than those for humans. No sense complicating the situation by experimenting on the latter.

Certainly, objections may be raised. I feel that I can adequately answer most of them, however:

1. Are we increasing the burden by requiring the gerbils to learn both English and a programming language?
   Emphatically not. The programming language may be taught independently of any natural language.
2. Can gerbils be taught to program, in the first place?
   Certainly. Brain-size studies have established the gerbil's cranium to be between 90 and 97 per cent as large as that of the average programmer-trainee.
3. What will the effect of these trained gerbils be on the Canadian labour market?
   The life span of gerbils is such that the impact, if significant, will at least be of short duration.

The basic plan of attack is as follows:

A. Determination of error-prone language constructs.

Proper programming practice will be taught by normal Pavlovian techniques. Initially, it may be too ambitious to expect to reward only correct programs. Instead, I may begin by rewarding mnemonic variable names, proper syntax in a single expression, good paragraphing, etc.

During the course of this training period, records will be scrupulously maintained which indicate those constructs which were most difficult for the gerbils to learn. The language will be appropriately modified and its error rate correspondingly reduced.

The use of several D to A converters will allow the compiler to notify the gerbil of errors detected by the grammatical redundancy incorporated into the language.

B. Measurement of complexity in completed programs.

Complexity factors will be measured in several ways. For the initial series of multi-factor experiments, ease of understanding will be measured by training the gerbils to reward the experimentor when they are presented with a comprehensible program.

Later on, error correction and ease of modification will be studied in timed trials, with a reward provided to the gerbil based upon the number of successful corrections and/or modifications which he(she(it)) is able to complete.

C. Determination of necessary redundancy.

It may be assumed that repeated coding of the same algorithm will lead, via "majority modification" of the resultant program, to a more correct implementation. The degree of reduncancy (or error checking) required on the part of the programmer is the subject of this investigation.

Pioneering work in this area has recently been undertaken by Steele, who writes "On Fault Tolerant Computing and the Number of Gerbils Necessary to Simulate a Turing Machine." In particular, Steele notes that five gerbils for each of the ten functions of his universal Turing machine simulator provides adequate reduncancy in most circumstances, since "one can be sleeping, one can be crapping and the other three can be polled to determine the correct result of the function."

Interesting work remains to be done in this area. How much noise can be tolerated while still maintaining the integrity of the computation? How long a computation may safely be undertaken by a Turing machine simulator with, say, a 99% accuracy rate? Finally, what is the MTBGF (Mean Time Between Gerbil Failure) of such a system? Once data have been collected, worthwhile comparisons could be made between gerbil and semiconductor reliability. Would ENIAC have been able to last more than 45 minutes if it contained 18,000 gerbils?

More to the point, I will have a number of gerbils code the same algorithm, and will attempt to determine the additional accuracy which can be obtained by redundancy factors ranging from two to, say, fifty. These results should be directly applicable to human programming endeavors.

Summary: At the conclusion of this research I will have determined those factors which make programs easily understandable to an objective group of programmers, and will have designed a lan-uage free of error-prone constructs. Finally, I will have established bounds on the expected improvements obtainable through redundant programming.