Engineering Notes

ENGINEERING NOTES are short manuscripts describing new developments or important results of a preliminary nature. These Notes cannot exceed 6 manuscript pages and 3 figures; a page of text may be substituted for a figure and vice versa. After informal review by the editors, they may be published within a few months of the date of receipt. Style requirements are the same as for regular contributions (see inside back cover).

AIAA 79-1990R

Design of a Verifiable Subset for HAL/S

William D. Young,* Anand R. Tripathi,† Donald I. Good,‡ and James C. Browne§
The University of Texas at Austin, Austin, Tex.

Introduction

PROGRAM verification has been a topic of intense research interest in recent years. One result has been the design of programming languages such as Gypsy[1] and Euclid[2] for constructing formally verifiable software. These languages were designed to permit the application of modern verification techniques. The research described in this paper and in Ref. 3 was an attempt to measure the applicability of such techniques to features of an existing language, HAL/S,[4,5] and to isolate a "verifiable subset," HAL/S/V, of that language.

HAL/S is a PL/I-based language designed to fill the software needs of the NASA Space Shuttle project. The breadth of this work mandated a language broadly based enough to meet the requirements of a wide range of aerospace applications. Features of HAL/S include: floating point capabilities; error handling and recovery; multiple data types with implicit and explicit type conversions; concurrent and real time programming capabilities; independent compilation; interfaces with subroutines written in other languages; and macros for inline textual substitution. These features and others make HAL/S a powerful tool for the aerospace programmer.

However, programs in any language are only worthwhile if they correctly perform their intended function. This is doubly true in aerospace applications where software failure may endanger human lives and costly hardware. Ideally, only programs of demonstrable reliability should be used in critical applications. Many researchers believe that maximum reliability can be obtained by designing software in a structured fashion and formally verifying the resulting programs.

With respect to programs written in a complex language such as HAL/S, two major obstacles arise.

1) Proving the correctness of a program is often quite difficult. This difficulty is compounded for large and complex programs.

2) Many of the features of HAL/S which are desirable for writing aerospace software are difficult or impossible to verify.

The first point illustrates the need for structured programming to break a complex program into manageable subunits and for machine assisted program proving systems.[6]

The second point indicates the value of a careful analysis of language features for verifiability to alert the programmer to the fact that the use of certain language constructs will inhibit formal verification.

Criteria of Evaluation

Characteristics generally acknowledged as desirable in programming languages[7,8] are often crucial from the point of view of verification. Examples are simplicity of the language and good program and data structuring facilities. Simplicity of the language can be judged using the following metrics: size, generality and extensibility of constructs, coherence, and precision of the language description. A simple language eases the verification process since the programmer is well aware of what his program is doing and has confidence in it. Program and data structuring should be such as to allow modular decomposition of the program into manageable subunits for proof purposes. The more that global and intermodular analysis is necessary to understand a program, the more difficult verification becomes.

Other areas which relate more specifically to evaluating for verifiability are: aliasing, axiomatizability, simplicity of verification, and nondeterminacy. Aliasing (allowing variables to be referenced via several names) should be minimized in the language. This ensures that assertions, using one name for a variable, are not invalidated by changing the value of that variable using some different name.

Axiomatizability of a language feature refers to the ability to write a collection of rules which completely and unambiguously describes the results of applying that feature to any data item. Such descriptions are essential for automatic verification because a verification system manipulates symbolic quantities which represent arbitrary input values. The results must be comprehensible in terms of general principles which are independent of actual input values.

Simplicity of verification is violated for many language constructs which are theoretically possible to verify. Some constructs are so excessively tedious to verify that the inconvenience outweighs the benefits of retaining them in the language.

Finally, nondeterminacy, which results because the language definition does not specify adequately the results of certain operations, complicates the verification process by making it highly machine-dependent. Other sorts of nondeterminacy which result from performing concurrent operations on shared objects are more a problem of a particular style of concurrent programming than of the language used.

This list of criteria is not intended to be inclusive, but only to suggest the sorts of considerations relevant to evaluating language features for verifiability. The only true validation of the verifiability of a particular language feature is the design of a workable verification strategy or proof rule.

Verification and HAL/S

The primary goal of this research was to examine the various constructs of HAL/S from the point of view of verifiability and to define a subset of the language in which to write programs which can be verified using formal proof methods. What follows is a list of some of the major features which were omitted or restricted in defining the verifiable subset HAL/S/V along with the reasons for these changes.

1) One major feature of HAL/S omitted from the verifiable subset is the scalar (floating point) data type. No convenient axiomatization for floating point arithmetic is

Presented as Paper 79-1990 at the AIAA/IEEE/ACM/NASA 2nd Computers in Aerospace Conference, Los Angeles, Calif., Oct. 22-24, 1979; submitted Nov. 26, 1979; revision received Aug. 4, 1980. Copyright © American Institute of Aeronautics and Astronautics, Inc., 1980. All rights reserved.

*Research Scientist, Institute for Computing Science and Computer Applications.

†Research Scientist, Institute for Computing Science and Computer Applications.

‡Director, Institute for Computing Science and Computer Applications.

§Professor of Physics and of Computer Science.

currently available. Automatic verification of a program requires that the effects of every language construct should be symbolically representable in an exact way. Only in that way can meaningful assertions be constructed to describe the program state. Operations on scalar data items introduce error terms, the magnitude of which is highly data-dependent. The matrix and vector data types, being composed of scalar elements, are also omitted from HAL/S/V.

2) No nonlocal referencing is permitted in HAL/S/V. Nonlocal referencing inhibits program decomposition for proof purposes and complicates program analysis substantially. For parameterized program units, it also permits aliasing since global data may be referenced directly as well as through the parameter list.

3) Replace statements and replace macros, the textual substitution facility of HAL/S, have been omitted from the subset. Each instance of textual modification presents the verifier with two programs to verify—the original and the modified version. In the worst case, the difficulty of verification may be exponential in the number of replace statements appearing. Such a severe penalty seems to outweigh the benefits of the facility.

4) No implicit precision or type conversions are permitted in HAL/S/V. Implicit conversion masks from the programmer and from the verifier the fact that in the conversion process information may be lost or spurious information added. Requiring that the programmer perform all precision conversions explicitly forces him to recognize this possibility and compensate for it.

5) HAL/S/V functions are restricted to conform to the mathematical notion of "function" as much as possible. That is, the entire behavior of the unit should be representable by a functional (in the mathematical sense) input-output relation. The prohibition of nonlocal referencing is an important step in that direction. Additionally, functions may not call, directly or indirectly, time-dependent system routines such as RUNTIME, DATE, or PRIORITY.

6) Input parameters to a routine may not be altered during the execution of the routine. To ensure this, input (value) and assign (reference) parameter lists must be entirely disjoint; input parameters may not be passed as assign parameters to other routines.

7) The HAL/S name data type is not included in HAL/S/V. A restricted pointer facility is a valuable feature of many languages. Verifiable pointer facilities in languages such as Pascal allow pointers only to unnamed, dynamically created objects. However, treating pointers as merely alternate names for data items, as HAL/S does, invites aliasing of the worst kind.

8) Transient event data items are not permitted in the verifiable subset. The semantics of logical operations such as "not E," where E is a transient event, is not well-defined.

9) To reduce the complexity of proofs of concurrent programs, the following restrictions are made. Accesses to shared data should be carried out in mutual exclusion; therefore, all shared data items belong to lock groups. No process may terminate abnormally if it or any dependent accesses shared data. All scheduling should occur at the outermost program level.

Though this list contains the major features of HAL/S omitted from the verifiable subset, there were a large number of more minor deletions, e.g., array processing features and the subbit conversion facility. Some modifications were necessary to make HAL/S/V a consistent language, e.g., requiring that the arguments of schedule and wait statements, which are scalar valued expressions in HAL/S, be integer valued in HAL/S/V.

Conclusions

The attempt to isolate criteria of evaluation for verifiability and apply them to HAL/S has revealed the following:

1) The design of programming languages for aerospace applications must be a process of compromise between the goals of inclusiveness and expressiveness on the one hand and verifiability on the other.

2) A clearer understanding of the kinds of features which are verifiable will aid in future work in designing automatic verification systems for HAL/S and other languages. Such efforts will doubtlessly be applied to many languages as more effective program proving techniques become available.

3) Most users of a language are concerned with writing software which is correct, though not necessarily formally verifiable. For them, verifiability analysis identifies language features which are ill-suited to structured programming and which contribute to the writing of software which is difficult to read, modify, and understand.

Acknowledgments

The authors are grateful to Dr. J. Misra for discussions regarding HAL/S and language evaluation. This work was supported by NASA Grant NSG-1479.

References

[1] Good, D.I., Cohen, R.M., Hoch, C.G., Hunter, L.W., and Hare, D.F., "Report on the Language Gypsy, Version 2.0," The University of Texas at Austin, ICSCA-CMP-10, Sept. 1978.

[2] London, R.L., Guttag, J.V., Horning, J.J., Lampson, B.W., Mitchell, J.G., and Popek, G.J., "Proof Rules for the Programming Language Euclid," *Acta Informatica,* Vol. 10, Jan. 1978, pp. 1-26.

[3] Young, W.D., Tripathi, A.R., Good, D.I., and Browne, J.C., "Evaluation of Verifiability in HAL/S," *Proceedings of the AIAA Computers in Aerospace Conference II,* Oct. 1979, pp. 359-366.

[4] "HAL/S Programmer's Guide," Intermetrics, Inc., Cambridge, Mass., June 1976.

[5] "HAL/S Language Specification," Intermetrics, Inc., Cambridge, Mass., June 1976.

[6] Cheheyl, M.H., Huff, G.A., Gasser, M., and Millen, J.K., "Secure System Specification and Verification: Survey of Methodologies," MITRE Corp., Bedford, Mass., WP-22473, Sept. 1979.

[7] Wirth, N., "On the Design of Programming Languages," *Proceedings of IFIP Congress 1974,* North-Holland, Amsterdam, 1974, pp. 386-393.

[8] Hoare, C.A.R., "Hints on Programming Language Design," *Proceedings of the Symposium on Principles of Programming Languages,* SIGPLAN/SIGACT, 1973, pp. 1-30.

AIAA 80-0058R

Dumping Momentum Magnetically on GPS Satellites

Jack R. Ferguson Jr.*
University of Texas, Austin, Texas
and
George T. Kroncke†
Martin Marietta Corporation, Denver, Colo.

Introduction

GLOBAL Positioning System (GPS) is a planned worldwide navigation system which will provide users with a three-dimensional location accurate to ± 16 m. The space segment of the system will consist of 18 three-axis stable, Earth-pointing satellites in inclined circular orbits of 12-h periods ($r = 4.17\ R_e$). To date, six satellites have been launched and are being used in the concept validation phase of the program.

Presented as Paper 80-0058 at the AIAA 18th Aerospace Sciences Meeting, Pasadena, Calif., Jan. 14-16, 1980; submitted Jan. 17, 1980; revision received July 28, 1980. This paper is declared a work of the U.S. Government and therefore is in the public domain.

*Graduate Student, Aerospace Engineering. Member AIAA.

†Staff Engineer, Systems Engineering Branch, Aerospace Division.