Does Program Verification Help? How Much?

Carl E. Landwehr

Computer Science and Systems Branch
Naval Research Laboratory
Washington. D.C. 20375-5000

1. Introduction

Verification techniques applied to programs, specifications, and hardware/software systems, are frequently touted as an effective way to assure that critical system properties such as security or safety are maintained. Empirical evidence for this assertion is minimal at best. This note sketches an experiment that could demonstrate effectiveness of verification techniques compared with more traditional methods. A topic for the Verkshop could be a more complete definition of such an experiment. This proposal is motivated in part by recent work to determine whether independent software implementations from a common specification in fact reduce the likelihood of common errors in the programs produced [Knight84].

2. Background

Assuring that a particular program, system of programs, or system of programs and hardware correctly implements its specification has been a fundamental motive for program verification. While more safety and security critical systems are being developed daily, developers of these systems have not exactly flocked to use program verification techniques. Reasons for this include:

familiarity with other approaches and lack of familiarity with program verification techniques,

uncertainty about what system properties need to be verified,

uncertainty of what system properties are feasible to verify,

uncertainty about the cost of performing feasible verifications, and

uncertainty about the effectiveness of program verification as opposed to more traditional methods of assuring system properties.

The proposed experiment focuses on the last two uncertainties.

Considerable resources are expended today assuring properties of systems involving use of nuclear weapons, aircraft and rocket control systems, and many others. Most of these resources are expended on testing of one kind or another. In some cases, parallel software developments from a common specification have been used as well. If program verification is in fact a more effective approach to gaining the needed assurance that these systems function properly, it ought to be possible to construct an experiment that demonstrates this fact.

3. An Experiment in Assurance

The basic structure of the experiment is as follows:

Define an informal but precise program specification.

Employ individuals to implement that specification using a variety of assurance techniques. There need to be many replications with each technique.

Keep track of the resources used in each implementation.

Collect the resulting programs and test them against each other and against a "correct" version.

Assess the "correctness" of programs produced using different techniques and also assess the resources used for each technique.

4. Issues

There are many pitfalls to be avoided in constructing an experiment such as the one proposed here. Among the issues that need to be considered are the ones listed below.

Task Definition

The task must be specified precisely enough to assure agreement among observers of the behavior of a correct implementation, but using a formal specification would bias results. In ordinary system development, the task of formalizing requirements is part of the burden imposed if formal verification techniques are to be employed. On the other hand, it would not seem unfair to choose a problem from an area in which some formal groundwork has been done (e.g., security). The scope of the task must not be so large as to prohibit large scale replications nor so small as to be a trivial exercise.

Choice of Approach

The individual building the implementation should be allowed at least some choice in the implementation and assurance techniques he chooses. Preferably, each individual should use the technique he thinks is best suited to his skills and the task at hand. On the other hand, the experiment needs to be limited to a fairly small and well-defined number of strategies, so some advance planning and guidance will be needed. The task should be sufficiently restricted so that resource constraints are not a limiting factor.

Use of Automated Tools

There are automated tools available for testing purposes as well as for verification. The experimental design may need to take into account the use of different tools by different individuals. Some restrictions on the set of possible tools may have to be considered to limit the scope of the experiment. An extreme case might be to restrict all use of automated test and verification tools and limit the program to a task sufficiently restricted that manual use of formal techniques would be practical.

Measuring Resource Consumption

To persuade developers to use a particular approach it is necessary not only to demonstrate that it can lead to a high level of assurance, but also that for a given expenditure, the approach can provide a higher level of assurance than the alternatives. This requires that the resources (human and computer time) expended by each individual be accounted for.

Responding to Queries

Different approaches may lead subjects to ask different kinds of questions about the specified task. In responding to such queries, should the experimenter broadcast results to all participants? Probably not, since one important characteristic of an approach may be the questions it leads one to ask during system development. Broadcasting questions and replies could erase the effect of this factor.

Measuring Assurance Achieved

An unambiguous way of assessing the assurance level achieved by a particular implementation is required. The least controversial measure might be obtained by testing each program exhaustively and counting the number of cases in which it fails. This approach is probably impractical, but a statistical sampling of substantial size could provide a convincing measure. In any case, some measure needs to be defined in advance.

5. Conclusion

If a satisfactory definition for such an experiment can be agreed on and the experiment can be carried out, there is potential for obtaining realistic cost data, convincing evidence that program verification is an effective way to assure that programs maintain crucial system properties, and widespread exposure of these techniques to potential users.

6. Reference

Knight84 — Knight, J. and N. Leveson. A large scale experiment in N-version programming. Univ. of Virginia, Charlottesville, 1984. Submitted for publication, 15th Fault Tolerant Computing Systems Conference.