Towards a Worldwide Verification Technology

Wolfgang Paul

April 2005

1 Introduction

Verisoft [1] is a large coordinated project funded by the German Federal Government. The mission of the project is i) to develop the technology which permits the pervasive formal verification of entire computer systems consisting of hardware, system software, communication systems and applications ii) to demonstrate in collaboration with industry this technology with several prototypes. During the fall and winter of 02/03 this project was planned by a task force headed by the author.

This task force had to face issues very closely related to what we want to discuss in Zurich and we have lived now with the decisions made early in the year 2003 for more than two years. Based on this—mostly positive—experience we make seven scientific, technological and administrative suggestions for the worldwide coordination of efforts in software verification.

2 Basic Research Versus Technology

Basic research identifies fundamental effects and laws. It also develops laboratory prototypes. Laboratory prototypes demonstrate, how newly discovered laws and effects may be applied: *something like* a laboratory prototype is expected to work in engineering. Turning this prototype into a component of a technology is left to the engineers. It requires the elaboration of details which are judged to be boring from a basic research point of view.

A component of a technology must work *as it is*. And it does not work in isolation. All components of a technology must work together as they are. The world of engineering technology is a binary world; time to work out details is (or at least should be) over: things work or they do not work. Imagine you are dying in an airplane crash due to bad software. That fixing a single line of code would have saved your life is no consolation whatsoever.

In the past, research in the field of verification has stressed the basic research aspect, very much at the expense of the engineering aspect. As a consequence, we now have CAV tools which are numerous, ingenious and often even powerful. But

1. the landscape of our tools is still very poorly integrated and
2. there is an ongoing discussion about the quality of the specifications that we are using.

3 Right Specifications and Stacks

A specification can be bad for two reasons:

1. It does not capture the user's intention. In general this cannot be discovered by mathematical methods alone.
2. In a computer system with layers, it does not permit to deduce desired properties of the next layer upwards or it cannot be proven using properties from the layer below.

Fortunately there are no principal difficulties to test, whether a specification works together with correctness theorems of other system layers. One simply tries it. That it can be done has been demonstrated as early as 1989 in the famous CLI stack [2]. Experiences since 2003 with far more complex stacks in the Verisoft project are also very encouraging.

We therefore judge it necessary, that the development of a technology for software verification has to be carried out in the context of the verification of entire stacks.

A case study: compilers in stacks We inspect three well known sources of definitions of programming language semantics: i) the classical Hoare/Wirth paper on Pascal semantics [3] ii) the textbook of Nielson and Nielson [4] iii) the textbook of Winskel [5].

In [4, 5] variables can range over the natural numbers. For a program running on a finite processor there is no way to prove this. In contrast, *int* is a finite data type in [3]!

Real programs run under operating systems and they perform I/O. Their computations are *interleaved* with the computations of other programs. Modelling this requires small steps semantics. Thus one cannot rely exclusively on the (big steps) definitions in [3].

Nevertheless the definitions of [3] are a component of engineering technology. Because theorems proven in Hoare logics hold in the corresponding small steps semantics (for the proofs see e.g. [4, 5]) they can be used exactly as they are to prove properties of terminating portions of programs. Experience from the Verisoft project suggests that this should by all means be done: productivity with Hoare logics is *much* higher than with small steps semantics alone.

If the verification of an operating system is also part of a project, then conventional small steps semantics alone does not suffice either: one needs to consider in line assembler code for the following reasons: i) arguing in high level language alone is impossible: an operating system written in high level language

alone could see in its own variables neither the processor registers nor the user processes. ii) arguing on the assembler language level alone (as was done in [2]) would not be productive enough.

4 Paper and Pencil Theory

Let us assume that we succeed to formally verify a complex stack. Then we can *necessarily* produce a human readable transcript of the formal correctness proof. This proof would be part of a big unified theory of computer science which would

1. be (at least!) as stringent as the classical mathematical theories,
2. include in a unified way the standard models of theoretical computer science, and
3. have real systems as examples.

We believe that progress will be faster if this theory is developed first with paper and pencil. In the language of G. Hotz these paper and pencil proofs then can serve as building plans for the formal proofs.

A case study: from gates to user processes i) Hardware is easily specified in the language of switching theory. ii) The random access machines of theoretical computer science are appropriate for specifying instruction sets. iii) Small steps semantics of high level languages is specified by abstract interpreters. iv) various models for distributed computation permit to treat communicating user programs.

Clearly in a verified stack one needs simulation theorems between different layers. Processor correctness is between models i) and ii). Compiler correctness is between models ii) and iii). Operating system correctness—with the scheduler abstracted away—is (because of the in line assembler code) between models ii) and iv). In the Verisoft project the paper and pencil proof for operating system correctness required the introduction of two parallel models of computation between models ii) and iv): one for operating system kernels and one for operating systems without abstracting away the scheduler.

5 Standardizing Language and Tools

Worldwide cooperative effort is impossible without establishing a common language. In software engineering there is a small number of standard programming languages, among them C and Java. The semantics are admittedly not too well defined, but compilers of a small number of large vendors establish a small number of de facto standards.

For the establishment of a worldwide verification technology we need as a counter part a small number of standard CAV systems. They should be

1. cheap mass products maintained by companies
2. universal: formalization of arbitrary mathematical statements and arguments should be reasonably straight forward. Presently interactive high order logic provers seem to be the best candidates.
3. easily extendible by automatic tools; interfaces to do this must be open. Without such interfaces we close the door to continuous increase in productivity.

Only with compatible standard tools can different groups of engineers exchange or trade (!) proofs.

6 Standardizing Definitions

Some crucial formal definitions should be standardized and maintained (in standard language) in downloadable form on certain web sites. Examples are

1. the semantics of some standard instruction sets (note that this includes the IEEE floating point standard).
2. small steps semantics and Hoare logics for C and Java
3. the semantics of certain standard real time operating systems (such standard systems exist for instance in the automotive industry).

7 Rewarding Engineering Work

Even the best technical decisions are useless if they are not supported by project members. Students will be reluctant to do work which will not lead to a thesis. Researchers at universities will be reluctant to do work which they cannot publish. We must therefore give proper rewards for the engineering aspect of our work. In particular we have to establish a forum for the following kinds of results:

1. The integration of known automatic methods in known interactive provers if that increases the productivity on large realistic benchmarks. This in turn requires the publication of such benchmarks *in a form which is easy enough to read*; putting existing large formal proofs on the web is not enough. In the Verisoft project we are working on such benchmarks together with the Southern Methodist University at Dallas.
2. The 'mechanization' of existing paper and pencil correctness proofs for major system components in a CAV system, if similar proofs were not mechanized before. In an engineering sense one can simply not trust paper and pencil proofs alone; in this respect we agree with [7]. In contrast a mechanized proof establishes not only trust in the verified component. It also shows, that the line of arguments that is used is *complete* and hence should work in future similar proofs.

Failing to reward the engineering aspects of our work as highly as the basic research aspects will slow down the development of technology.

8 Summary of Suggestions

In order to help establish a worldwide technology of software verification the author suggests

1. to clearly recognize the difference between basic research and engineering,
2. to study stacks in order to get specifications right,
3. to produce paper and pencil proofs first,
4. to recognize the need for a grand unified theory of computer science,
5. to agree on a small number of standard tools permitting both interactive and automatic work,
6. to standardize key definitions, and
7. to establish a proper reward structure for results with a strong engineering flavor.

9 The Suggestions and the Verisoft Project

1. The focus of Verisoft is on technology and applications. A major portion of the funding however goes into the integration and optimization of existing tools. The development of new tools is strictly driven by demand.
2. The goal is to verify three stacks: i) a 'public system'[1] including processor, compiler, operating system, TCP/IP, SMTP mail server and electronic signature. ii) In collaboration with BMW an automotive system including a FlexRay bus, the hardware of the electronic control units (ECUs), an OSEKTime real time kernel and a distributed application using four ECUs. iii) In collaboration with T-Systems a biometric access control system. Also in collaboration with Infineon we are attempting the complete formal verification of the future TriCore2 processor core.
3. We attribute the rapid progress in the Verisoft project so far to the extensive use of paper and pencil proofs early in the project.
4. At the time of this writing (April 2005) we have on paper a unified theory from the gates to the operating system kernel. We are in the process of writing down the theory until the operating system level. Formal processor verification is completed. Formal compiler verification is expected to be completed in 2005. Operating system kernel verification is expected to be completed in 2006 or 2007.

[1] All results can be published.

5. The standard interactive tool is Isabelle/HOL [6]. Automatic tools are NuSMV, SAT solvers, the SPASS and E-Setheo theorem provers as well as various new tools developed on demand (translation validation, array bound checking, termination)

6. As of today standard formal definitions in Verisoft are: i) Syntax and semantics of the DLX machine language. ii) small steps semantics for $C0_A$; in a nutshell $C0$ is Pascal with a C syntax. $C0_A$ is $C0$ with in line assembler code iii) Hoare logic for $C0$ iv) syntax and semantics for CVM (communicating virtual machines; a parallel formal model for operating system kernel and user programs)

7. progress with the integration of automatic tools was initially slower than expected. We attribute this to the unfavourable reward structure.

References

[1] The Verisoft Consortium. The Verisoft Project. `http://www.verisoft.de/`.

[2] William R. Bevier, Jr. Warren A. Hunt, J Strother Moore, and William D. Young. An approach to systems verification. *J. Autom. Reason.*, 5(4):411–428, 1989.

[3] C. A. R. Hoare and Niklaus Wirth. An axiomatic definition of the programming language PASCAL. *Acta Inf.*, 2:335–355, 1973.

[4] H. Riis Nielson and F. Nielson. *Semantics with Applications: A Formal Introduction.* Wiley, 1992, revised online version: 1999.

[5] G. Winskel. *The formal semantics of programming languages.* The MIT Press, 1993.

[6] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[7] Richard A. De Millo, Richard J. Lipton, and Alan J. Perlis. Social processes and proofs of theorems and programs. *Commun. ACM*, 22(5):271–280, 1979.