Exploiting Workstations and Displays in Verification Systems

S. Kamal Abdali and Ralph L. London

Computer Research Laboratory
Applied Research Laboratories
Tektronix, Inc.

February 15, 1985

Position Paper presented at the Verification Workshop III

We write as two researchers who have in the past worked in program verification and who note in the Call for Participation a concern for "the support provided by the interface" in ongoing verification systems. We wish to encourage the inclusion in verification systems of powerful human interfaces to allow an individual to exploit recent advances in display technology. We realize this is not a new or revolutionary idea; indeed, we know that authors of verification systems would like to include such facilities. For various reasons, including higher priority tasks, the difficulty of incorporating such advances into existing systems, a focus on algorithm development, and a desire to be able to accommodate many kinds of terminals, no existing verification system to our knowledge has advanced beyond simple extensions to "glass teletypes." Suppes supported our knowledge when he lamented, "As far as I know, there is no standard regular use of a theorem prover anywhere in the world that extensively and directly interacts with graphic displays related to that which is being proved."* We think the time is appropriate to incorporate modern interface facilities into verification systems.

During the period of about seven years since the peak of activity occurred in verification system development, the economics of computing have changed significantly. The advent of personal workstations has made the cost of central processor time a negligible factor relative to the cost of people's time. Any improvement in the programming environment which can enhance user productivity is well worth the overhead. The most dramatic improvements in the programming environment have been made possible by the graphics interface offered by modern workstations via high-resolution bitmap displays, pointing devices, windows, and icons. This type of interface, developed a decade ago at Xerox PARC for Smalltalk systems, and included recently in the Apple Macintosh and Lisa, is available on a number of workstations for a cost affordable by typical verification system developers. While we are not suggesting that a verification system be put on a Macintosh, we do think that verifiers would do well to study the Macintosh capabilities and style.

Address communications to either author, Computer Research Laboratory, Tektronix, Inc., P. O. Box 500, MS 50-662, Beaverton, OR 97077. Phone and net address are Abdali: (503) 627-5205; Abdali%Tektronix@CSNet-Relay and London: (503) 627-5210; London%Tektronix@CSNet-Relay

* Patrick Suppes, The Next Generation of Interactive Theorem Provers, *Proc. 7th International Conference on Automated Deduction*, R. E. Shostak (ed.), Springer-Verlag, 1984, p. 309.

In our laboratory (and ours is far from unique) such interfaces and facilities are standard for our workstations whether we work with Unix compilers and editors, with imported application packages, or with Smalltalk. We are able to maintain several windows containing various pieces of information even when editing a simple piece of text. For those systems which do not take full advantage of our interfaces, we notice it is often harder or more involved to do the tasks we want. Of course, there is no loss of functionality, only loss of convenience and efficiency.

But for a task as demanding as program verification we suggest the human user needs all the help that is available. We want natural actions to invoke simplification and proof steps on formulas as well as for obtaining status and relational information. Facilities to ease the managing of complex proof trees, of complex interconnections among a set of lemmas, and of complex formulas, all seem important additions to the arsenal of tools that comprise a modern verification system. It is the added convenience and efficiency that will make a practical difference if not a theoretical difference. We note (with some envy) that the more recent of even business and recreational software for personal computers are taking full advantage of the improved user interface.

As a specific example where we think a window-oriented user interface may well be an important improvement, we cite the rewrite rule systems Reve and RRL. Our understanding of these systems is gained from actual runs of Reve and from seeing papers displaying transcripts of RRL in action. Clearly their output capability is limited to the straightforward presentation of the state of computation by a sequential listing of each of their actions. In principle everything is available if only we look in the appropriate place, perhaps doing some bookkeeping ourselves either mentally or with a notepad. For an expert user this may well suffice. Still, it appears to us that a different method of presentation of information may be useful. For example, we immediately think that four separate windows ought to be considered: one for equations, one for rules, one for dialogue, and one for the workspace (e.g., for displaying critical pairs). Thus the current state is shown separately in these windows. We concede that the time sequence of events may not be as clear this way, but if that information is necessary, there may be ways to keep it from being lost. With more of the state immediately available, the possibility is present to point to various formulas instead of having to type in numbers or some other notation to invoke actions. This well used option tends to minimize needless retyping as well as to minimize mistakes. In fact it is interesting to see how little keyboarding is actually required in well designed and well engineered interfaces. Also, the advantages of menus become available to invoke actions rather than having to remember all the possible actions. Of course, by menus we mean not a printout of the choice list which clutters up the output, but the popup or pulldown variety which disappears as soon as an action has been selected. There is also the possibility of making the many switch settings in a verification system both visible and changeable in a separate area using both text and icons.

Based on our use of display workstations, we suggest the desirability of being able to carry out a set of operations, say a proof or derivation, by invoking operations from menus on previously selected parts of a formula, presumably as text but perhaps also as part of a tree. This would allow selective substitution of equals for equals, direct invocation of an induction hypothesis, or fine control of an algebraic simplification. By keeping a record of all "transactions," the system could later automatically reperform a proof with no (or little) human interaction. In addition, we feel that the graphic interface should be utilized to display and edit equations and formulas as two-dimensional entities rather than in linearized notation which was forced by Ascii terminal limitations. There

is no reason not to utilize multiple-fonts, special symbols, Greek letters, etc., to make the workstation screen as visually appealing as a printed mathematical page. This is not just for the sake of aesthetics; use of concise, familiar notation makes formulas comprehensible at one glance. Similarly, trees and graphs should be portrayed pictorially rather than with linear text. Traversal, scrolling, and zooming techniques are available which permit working with pictures much larger than a window or screen size. Individual nodes may be expanded in a window that overlaps part of the tree, and later this window can be removed to re-expose the overlapped part of the tree.

We also suggest the importance of separate and multilevel paths of problem exploration. Thus, subcomputations, digressions, and experimentations, all of which are important to problem solving, properly belong in separate windows, perhaps sometimes in a "notepad" or "scratchpad" window. Even proving a lemma ought, on request, to be treatable as a separate and separable activity. If any of these activities are physically interspersed with the mainline computations, they too clutter up the display and impede productive flow of thought.

Displays also allow for the presentation of information graphically rather than just as formulas or text. Discovery of proofs, lemmas, or algorithms may be aided by visual representation of appropriate icons, figures, or drawings. An example of such a facility is found in Chao's Geometry Theorem Prover which can display the diagram for a theorem of geometry from its algebraically encoded version. It may be useful to illustrate pictorially the content of other theorems and lemmas along with their text. To exploit graphical images London and colleagues have been successfully animating algorithms using the Smalltalk environment and in the process have discovered an optimization to an existing algorithm. For the open "Pancake Flipping" problem, they have constructed some graphical views of relevant structures and allowed simple mouse operations to move to new states; it remains to see whether new algorithms are discovered using this method over pencil and paper methods. Very effective use of animating algorithms has been made at Brown University for teaching computer science courses and for algorithm analysis research. The intuition gained by the animation has actually led to the discovery of new algorithms and new results in algorithm analysis.

For implementing a verification system we suggest a mixed-language strategy in which a Smalltalk-like or Macintosh-like environment provides the top-level interface while the computationally intensive activities are provided by other languages, for example, compiled Lisp. This strategy is currently being used successfully in our laboratory. Another reason for this strategy is that, along with Smalltalk's other virtues, it has proven to be good as a prototyping vehicle. Thus, ideas can be tested in implementations with realistic functionality far sooner than by using most conventional languages.

We are certain our suggestions above do not come close to exhausting the possibilities for interactive use of current display technology. Verification researchers and users should be able easily to propose other ideas to be explored and validated. For indeed, it will be necessary to refine our suggestions and those of others. Even though there may (properly) be more importance attached to theoretical breakthroughs and to algorithm development, researchers and funders should not overlook the importance of good system development tools and especially good system and programming environments. Workers who have them are most reluctant to surrender them.

This is neither the time nor the place for detailed comments from us about specific verification systems. Others know far better than we the kinds of efforts spent in the use of such systems to complete a verification and, therefore, the kinds of changes or additions that are appropriate. We do note, though, that the Affirm system does pay attention to important pragmatic aspects such as management of proof forests, browsing, and transaction recording for later reference. However, we observe that its human interface, of necessity, is not one of its strong points. That interface was at best added on rather than being an essential part of the design. It is probably also true that this was unavoidable. Now, however, we know more about human interfaces and the essential role they play in system productivity. It is time to include them as the essential system components they are.