Testing Proofs by Examples

Susumu Hayashi[1]* and Ryosuke Sumitomo[2]

[1] Department of Computer and Systems Engineering,
Faculty of Engineering, Kobe University,
shayashi@kobe-u.ac.jp
1-1 Rokko-dai, Nada, Kobe, Japan,
[2] Graduate School of Science and Technology
Kobe University, 1-1 Rokko-dai, Nada, Kobe, Japan,
sumitomo@pascal.seg.kobe-u.ac.jp

Abstract. We will present the project of *proof animation*, which started last April. The motivations, aims, and problems of the proof animation will be presented. We will also make a demo of ProofWorks, a small prototype tool for proof animation.

Proof animation means to execute formal proofs to *find* incorrectness in them. A methodology of executing formal proofs as programs is known as "proofs as programs" or "constructive programming." "Proofs as programs" is a means to *exclude* incorrectness from programs by the aid of formal proof checking. Although proof animation resembles proof as programs and in fact it is a contrapositive of proofs as programs, it seems to provide an entirely new area of research of proof construction.

In spite of wide suspicions and criticisms, formal proof developments are becoming a reality. We have already had some large formal proofs like Shanker's proof of Gödel's incompleteness theorem and proof libraries of mathematics and computer science are being built by some teams aided by advanced proof checkers such as Coq, HOL, Mizar, etc.

However, construction of big formal proofs is still very costly. The construction of formal proofs are achieved only through dedicated labors by human beings. Formal proof developments are much more time-consuming and so costly activities than program developments.

Why is it so?

A reason would be lack of means of *testing proofs*. Testing programs by examples is less reliable than verifying programs formally. It is practically impossible to exclude all bugs of complicated software only by testing. Verification is superior to testing for achieving "pure-water correctness," a correctness at the degree of purity of pure water.

However, testing is much easier and more efficient to find 80% or 90% of bugs in programs. Since the majority of softwares need correctness only at the degree

* Supported by No. 10480063, Monbusyo, Kaken-hi (the aid of Scientific Research, The Ministry of Education)

of purity of tap water, the most standard way of debugging is still testing rather than verification. Furthermore, the majority of people seem to find that testing programs is more enjoyable than verification.

For software developments, we have two options. However, we have only one option for formal proof developments. Obviously checking formal proofs by formal inference rules corresponds to verification. (In fact, the activity of verifications is a "subset" of formal proofs by formal proof checking.) Thus, we may set an equation

$$\frac{X}{\text{formal checking of proofs}} = \frac{\text{testing programs}}{\text{formal verification of programs}}$$

A solution X would be a means to find errors in formal proofs quickly and easily, although it cannot certify pure-water correctness.

By Curry-Howard isomorphism, a mathematical theory bridging functional programs and proofs, the solution of this equation is

$$X = \text{testing proofs by } \textit{execution of proofs},$$

Since it resembles and shares aims with "animation of formal specifications" in formal methods, we call it proof animation. We often call it "testing of proofs" as well.

A plausible reaction to proof animation may be as follows:

> How can bugs exist in formal proofs? Formally checked proofs must be correct by definition!

Bugs can exist in completely formalized proofs, since *correctness of formalization cannot not be formally certified.* This is an issue noticed in the researches of formal methods, e.g. Viper processor verification project [1] and, even earlier, by some logicians and philosophers, e.g., L. Wittgenstein and S. Kripke [3]!

We will discuss that this difficulty is a source of inefficiency of formal proof developments and proof animation may ease it. A proof animation tool ProofWorks, figure 1, is still under construction and case studies are yet to be performed. Nonetheless, the experiences with proof-program developments in PX projects [?] shows such a methodology can eliminate bugs of some kind of constructive proofs very quickly and easily.

In the talk, we will discuss the theoretical and technical problems to be solved to apply proof animation to actual proof development.

We will also give demos of ProofWorks, if facilities are available. The current version of ProofWorks is a JAVA applet proof checker. Figure 1 represents ProofWorks running on a Web browser. Formal proofs under development is represented in the left box by Mizar-like formal language. Clicking "Extract" button, it extracts a computational content of the proof, which appears in the right box. The computational content is pure functional programs in the current version of ProofWorks. ProofWorks associates the proof text with program text. Positioning a cursor in one of the boxes and clicking one of >> or << buttons,

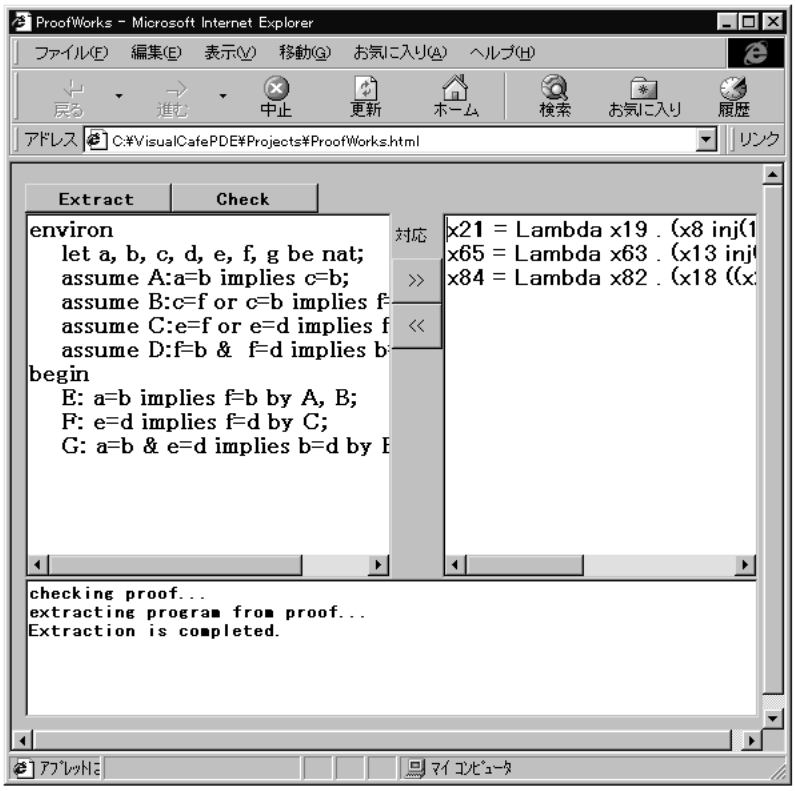

Fig. 1. ProofWorks

it shows the corresponding places in the other box. Thus, by finding a bug in a program in the right box, ProofWorks can show the corresponding points of in the proof in the left box, where a bug likely sits.

A full paper and other informations on proof animation will be available at

`http://pascal.seg.kobe-u.ac.jp/~hayashi`

References

1. Cohn, A.: The Notion of Proof in Hardware Verification, *Journal of Automated Reasoning*, Vol.5, 127-139, 1989.
2. Hayashi, S. and Nakano, H.: *PX: A Computational Logic*, The MIT Press, 1988
3. Kripke, S.: *Wittgenstein on Rules and Private Language*, Harvard University Press, Cambridge, Massachusetts, 1982
4. Sumitomo, R.: *ProofWorks: An Environment for Animation of Proofs*, Master thesis, Graduate School of Science and Technology, Kobe University, 1997.