Available online at www.sciencedirect.com
SCIENCE DIRECT
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 113 (2005) 45–63
www.elsevier.com/locate/entcs

Deductive Runtime Certification

Konstantine Arkoudas[1] Martin Rinard[2]

Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, USA

Abstract

This paper introduces a notion of certified computation whereby an algorithm not only produces a result r for a given input x, but also proves that r is a correct result for x. This can greatly enhance the credibility of the result: if we trust the axioms and inference rules that are used in the proof, then we can be assured that r is correct. Typically, the reasoning used in a certified computation is much simpler than the computation itself. We present and analyze two examples of certifying algorithms.

We have developed denotational proof languages (DPLs) as a uniform platform for certified computation. DPLs integrate computation and deduction seamlessly, offer strong soundness guarantees, and provide versatile mechanisms for constructing proofs and proof-search methods. We have used DPLs to implement numerous well-known algorithms as certifiers, ranging from sorting algorithms to compiler optimizations, the Hindley-Milner $\mathcal{W}$ algorithm, Prolog engines, and more.

Keywords: Verification, certifying algorithms, program correctness, proofs, certificates, Athena, DPLs

1 Introduction

Complete deductive verification of software systems can be extremely onerous. It is a major challenge to prove mechanically that a complex piece of software will always produce the correct output for any given input. The difficulty is due partly to the fact that deductive technology has not yet reached a sufficiently advanced state of the art, and partly to the inherently high complexity of software. Nevertheless, formal proofs are a superb methodology for increasing reliability, and we would like to find a use for them even when it is not practical to prove a system completely correct.

[1] Email: arkoudas@lcs.mit.edu
[2] Email: rinard@lcs.mit.edu

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2004.01.035

This paper presents an alternative to complete static verification, namely partial dynamic certification. Instead of statically proving that an algorithm produces correct results for all inputs, we express the algorithm as a proof-search procedure that takes an input x and not only produces a result r but also *proves* that r is a correct result for x. Such *certifying algorithms* can be viewed as instrumented versions of conventional algorithms, modified to *justify* their results with deductive reasoning. The reasoning is performed at runtime, and applies only to the particular input x and result r. The completed proof can be regarded as a *certificate* for the result r; once this proof is validated, we may say that r has been "certified."

Runtime certification is less powerful than static verification. It guarantees no more and no less than this: if and when the algorithm produces a result, that result is correct—modulo the logic that specifies what counts as correct. That can still be very useful, since it prevents the program from silently generating a plausible but incorrect result.

The advantage of runtime certification is that usually it is much more practical than static verification. Consider, for instance, the unification example we give in Section 4. A complete verification of a unification algorithm was given by Paulson [28], where he states that the proof "relies on a substantial theory of substitutions, consisting of twenty-three propositions and corollaries... The project has grown too large to describe in a single paper... The proof is not entirely beautiful. A surprisingly diverse series of problems appeared." A more recent correctness proof for a Martelli-Montanari-style unification algorithm using the Boyer-Moore theorem prover runs to thousands of lines [31]. By contrast, expressed as a certifying algorithm, our Martelli-Montanari unification procedure was implemented in less than one page of Athena code. [3] This dramatic difference is an apt illustration of the main tradeoff: static verification gives us peace of mind for all inputs, but is difficult; runtime certification gives us more limited assurance, pertaining only to particular inputs and outputs, but is much more feasible. Another important difference is that static proofs—such as the aforementioned by Paulson—usually verify an abstract model of the software component, not the actual code; whereas in certified computation the theorem refers to the actual result obtained in real time.

We envision runtime certification as a methodology to be applied to selected parts of a software system, not to every part. In some cases it might not be viable to characterize output correctness with mathematical rigor. For other components, such as reactive systems, the important issue is not output correctness but behavioral safety, and in that case other methods and formalisms such as runtime monitoring [8] or I/O automata [18] will be ap-

[3] Athena is the DPL we have used for all our implementations; we describe it in Section 3.