# Automatable Verification of Specificationally Rich Software Components*

Nicodemus M. J. Mbwambo[1], Joan Krone[2], and Murali Sitaraman[1]

[1] Clemson University, Clemson SC 29634, USA
{nmbwamb, msitara}@clemson.edu
[2] Denison University, Granville, OH 43023, USA
krone@denison.edu

**Abstract**

To capture the behavior of complex software components, specifications must be exacting yet simultaneously comprehensible, simple, and scalable. To achieve this goal, we need rich specification languages that allow software developers to use arbitrary mathematical theories—including those which are undecidable, and to employ mathematical models and notations most suitable for describing the behavior of components. This flexibility in software development requires an extendable library of mathematical theories from which the automated prover can retrieve theorems appropriate to the verification condition under consideration.

While decision procedures such as SMT solvers have been shown to be fast and efficient for program verification within a fixed domain of decidable theories, they cannot easily scale beyond such theories. So automatable verification of software with adequately rich specifications remains a challenging problem. This paper outlines an example verification problem to motivate a discussion of what concepts beyond decision procedures might be necessary. The end goal is to make progress toward developing automatable provers that work with extendable mathematical libraries and employ effective strategies to achieve verification.

## 1 Introduction

Formal verification of software requires a formal description of the software component's behavior using a specification language rich enough to support comprehensible description and a prover that can establish its correctness. A programming language analogy is useful. Modern languages have advanced from earlier high-level languages, such as C, FORTRAN, and Pascal, to languages like C++ and Java that allow new objects to be defined and employed to construct intellectually complex software components. Similarly, specification languages must advance to accommodate adequately descriptive mathematical theories that support the writing of specifications that are comprehensible, simple, scalable, and amenable to automated verification. Such specification languages are **rich**, and they provide the intellectual leverage needed to scale up and describe the complex behavior of sophisticated object-based software components.

## 2 Requirements for Automatable Verification for Rich Languages

The notion of specification language richness concerns support for software engineers to employ sophisticated mathematical theories to describe accurately and completely a component's

*This research is supported in part by grants from the U.S. National Science Foundation

behavior, and having it can be viewed as providing an "insight optimization" capability that allows software components to be described in the best possible way for clients to comprehend. It is analogous to classical optimizations that focus on implementation efficiency. However, when components with rich specifications are verified, the resulting verification conditions (VCs) frequently lie outside the scope of decision procedures.

The need for an extendable mathematical library supporting rich languages in turn requires a verification system capable of working with more than a fixed set of theories and of employing theorems from arbitrary domains when proving VCs. The working hypothesis is that if the software is well-engineered, with suitable specifications and code annotations, the resulting VCs would be comparatively straightforward [1, 2]. So our focus is on an automatable system for verifying verification conditions (VCs) whose correctness, relative to supporting theories that have been appropriately developed, is "obvious": A VC is obvious if it can be established from the available theorems in relatively few proof steps. As seen from Figure 1, the prover may fail to verify a VC either because it is invalid or because it is not sufficiently obvious from the available theorems in the library. So an appropriately developed theory may need to include theorems that are somewhat redundant for a sophisticated mathematician, but that are proof step saving for an automated verifier.

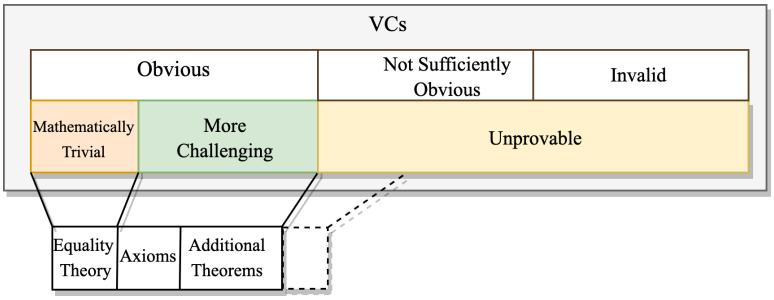


Figure 1: Verification condition provability when employing rich specifications

The use of the term "automatable" in this work means that after the system fails to verify a VC for not being sufficiently obvious, the responsibilities of software developers include making the correctness of their code obvious, such as by improving their specifications or their code annotations. It also means that mathematical specialists working in conjunction with software developers may need to formulate (and prove) additional theorems needed to make the correctness of all the VCs from their code obvious to the verification system.

Example decision procedures include SMT solvers such as Z3 [3], and Yices [4] that are designed to operate with a fixed set of mathematical first-order logic theories. Current verification systems using these decision procedures include Why3[5], and Dafny [6]. While these and other decision procedures have been successful in program verification because of their speed and efficiency, their scope is limited. Verifying new software components with varying complexities has a considerable development cost as for any assertions outside the decision procedure's domain, a new one has to be developed.

Our present work toward developing an automatable prover is set in the context of RESOLVE [7], an integrated language that enables rich specifications and object-based implementations. Key characteristics of the language are summarized in [8], including the notion of clean semantics. Recent automated verification work includes the role of specification engineering in proving [9] and generation of sequent-style verification conditions (Sun2021).

## 3 A Challenging Verification Example Outline

The specification presented in [11] conceptualizes a novel and generic tree data abstraction with a navigable position that makes it easy to explore and populate a tree, even while avoiding any explicit references in the specification. The comprehensibility of this concept is primarily attributed to its rich specifications. The specification language used makes it possible to extend the existing mathematical library to develop an undecidable, general tree theory with suitable definitions and predicates necessary to author succinct specifications of the data abstraction.

An implementation for a map data abstraction that uses the specified tree data abstraction is also described in [11]. VCs arising from proving the correctness of this implementation present a challenge to the current verification systems as it involves multiple theories, including the general tree theory for which there are no special-purpose solvers. The thesis here is that the VCs will be obvious for a verification system with well-engineered specifications and theories. Therefore, this workshop paper aims to raise a discussion of verification of such non-trivial implementations with specifications that use mathematics beyond fixed theories—a problem that will require advances in verification systems that go beyond the current capabilities of decision procedures.

## References

[1] Kirschenbaum J. et al. (2009) Verifying Component-Based Software: Deep Mathematics or Simple Bookkeeping?. In: Edwards S.H., Kulczycki G. (eds) Formal Foundations of Reuse and Domain Engineering. ICSR 2009. Lecture Notes in Computer Science, vol 5791. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-04211-9_4

[2] Cook, T. C., Harton, H., Smith, H., Sitaraman, M.: Specification engineering and modular verification using a web-integrated verifying compiler. In: 34th International Conference on Software Engineering (ICSE), pp. 1379-1382, 2012.

[3] de Moura L., Bjørner N. (2008) Z3: An Efficient SMT Solver. In: Ramakrishnan C.R., Rehof J. (eds) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2008. Lecture Notes in Computer Science, vol 4963. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-78800-3_24

[4] Dutertre B. (2014) Yices 2.2. In: Biere A., Bloem R. (eds) Computer Aided Verification. CAV 2014. LNCS, vol 8559. Springer, Cham. https://doi.org/10.1007/978-3-319-08867-9_49

[5] Filliâtre JC., Paskevich A. (2013) Why3 — Where Programs Meet Provers. In: Felleisen M., Gardner P. (eds) Programming Languages and Systems. ESOP 2013. Lecture Notes in Computer Science, vol 7792. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-37036-6_8

[6] Rustan, K., Leino, M. (2013) Developing verified programs with Dafny. In: 35th International Conference on Software Engineering (ICSE), pages 1488–1490, May 2013.

[7] Sitaraman, M., Adcock, B., Avigad, J. et al. Building a push-button RESOLVE verifier: Progress and challenges. Form Asp Comp 23, 607–626 (2011). https://doi.org/10.1007/s00165-010-0154-3

[8] Kulczycki G. et al. (2013) A Language for Building Verified Software Components. In: Favaro J., Morisio M. (eds) Safe and Secure Software Reuse. ICSR 2013. LNCS, vol 7925. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-38977-1_23

[9] Smith, H.: Engineering Specifications and Mathematics for Verified Software. PhD Thesis, Clemson University, 2013.

[10] Sun, Y., Welch, D., Sitaraman, M. (2021) F-IDEs with Features and VCs Designed to Assist Human Reasoning When Verification Fails. Applicable Formal Methods (aFM), Beijing, China.

[11] Mbwambo, N.: A Well-Designed, Tree-Based, Generic Map Component to Challenge the Progress towards Automated Verification. MS Thesis, Clemson University, 2017.