

The OSU RESOLVE Software Verification Tool Suite

Bruce Adcock, Jason Kirschenbaum, Derek Bronish

The Ohio State University, Columbus OH 43210, USA
{adcockb,kirschen,bronish}@cse.ohio-state.edu
<http://www.cse.ohio-state.edu/rsrg>

Abstract. To ensure true component reusability, it is essential that the client has confidence in the components employed. Our toolset takes specifications and code implementations written in the RESOLVE language, generates mathematical formulas from them in a provably sound manner, and proves or disproves the total correctness of the code relative to the specification using three automated provers.

1 Introduction

The OSU RESOLVE tool suite is a set of tools for the verification of software components written in RESOLVE [1–4]. Software component extensions, written using the RESOLVE programming language, are verified using one of three tools: Isabelle, SplitDecision, or Z3.

The tools accept programs written in RESOLVE, an imperative language designed with fostering reusable, component-based software as the foremost objective. The language includes a rich catalog of abstract data type interfaces, with corresponding model-based specifications, that client code can use; the language requires programming-by-contract. RESOLVE also includes syntactic slots for mathematical code annotations, which allow for a sound and relatively complete proof system [5, 6]. These annotations are exactly what is needed to ensure a sound and complete proof system; they include loop invariants, and termination metrics for loops and recursive operations. The language also allows arbitrary mathematical assertions to be added (thus requiring a proof of the assertion’s validity) to provide a “hint” to the back-end prover, but of course the goal is to minimize the use of this feature and make the verification truly automatic.

Given code purporting to implement a particular specification, the tool suite first generates verification conditions (VCs)—mathematical formulas that correspond to the correctness of the code. The tool suite then appeals to one or more back-end provers to establish the validity of these VCs. Finally, the result of the prover’s work is displayed to the user.

2 VC Generation

We generate VCs automatically in accordance with the RESOLVE program proof system defined formally in [6] and informally in [7]. This process can best be

characterized as filling out a symbolic tracing table: each line in the source code with a precondition is translated into a VC. Also, code annotations such as loop invariants and progress metrics, along with postconditions of implemented components are all translated into VCs. This is also known as a forward method of generating VCs.

VCs are generated by rote—creating an explosion of mathematical variables—and then simplified using simple theory-independent restructuring rules, e.g., using the assignment axiom of first order logic, to prune the number of variables. This process results in VCs that are simpler and easier to read. Also, case analysis that is required because of the code structure, i.e., if-then-else statements, is automatically performed. This process exploits information known by the VC generator to perform the “correct” case analysis rather than requiring the back-end automated theorem prover discover the case analysis.

3 Provers

3.1 Isabelle

Isabelle [8] is an automated proof assistant that is capable of checking a proof that a user directs. Moreover, this type of tool can perform many of the tedious steps needed to produce a proof. In simple cases, Isabelle can produce the proof outright, establishing the goal without human guidance.

More specifically, Isabelle has a simplifier that can be invoked to simplify assumptions and goals of a theorem. Isabelle also includes a classical reasoner that can perform many of the logical inference rules automatically. The proof structure of Isabelle is set up in a manner that mimics natural deduction. Assumptions and a goal are presented and simplifications can be applied to either. Rules for applying already-proved lemmas and theorems dictate how the assumptions and goals are modified. One can apply forward reasoning and modify the assumptions, apply backward reasoning and modify the goals, or apply both at the same time.

For convenience, many of the proof methods instantiated with common lemmas are performed in Isabelle via the `auto` and `force` commands, commonly referred to as “tactics.” These tactics use both the simplifier and the classical reasoner, the difference being that the `force` method will only succeed or fail, while the `auto` method will return a simplified goal (if possible).

New axiom systems, theories, and theorems can be entered into Isabelle using a built-in meta-level logic. We use this feature to import theories developed in a specialized manner for RESOLVE into Isabelle. RESOLVE’s string theory is one such example. Isabelle allows the proofs of the lemmas in string theory to be factored off from the usage of those lemmas for proving the various VCs. Of course, a useful theory must also have a witness (*i.e.*, a model that satisfies its axioms). Fortunately, such a model for string theory is readily available, namely the Isabelle `List` type. We have both imported the RESOLVE string theory and augmented it with extra lemmas and theorems to aid in automated verification.

Each VC is generated with a small proof script, that is designed to facilitate the automated proofs of correct VCs. The script applies simple techniques through the full-blown “auto” tactic in order to prove the VCs. This script is generated automatically.

We do not use most of theories available in `lsabelle` for the automatic proof of VCs, but rather use `lsabelle`’s proof engine along with theories already developed for use in RESOLVE specifications. This feature allows us to use other provers, e.g., `SplitDecision` and `Z3` without any modification to the VCs (other than syntactic translations).

3.2 `SplitDecision`

Proof assistants (such as `lsabelle`) have the default option of relying on human interaction. Often a proof assistant will halt its execution and present the user with alternatives for how to proceed, in an effort to reduce the number of “blind alleys” that it might follow in its proof. This dependence on outside intervention is not compatible with the desire to have VCs proven completely automatically. This is especially problematic, because they are not tuned to handle mechanically-generated formulas as what would be made for automated software verification. Proof scripts in `lsabelle` are one attack on this problem. Another is our custom-built tool designed specifically for the simplification of VCs, called `SplitDecision`.

`SplitDecision` leverages domain-specific mathematical knowledge along with general-purpose strategies for reducing logical formulas in order to simplify RESOLVE VCs, while maintaining logical equivalence. Each action it takes makes progress toward the ultimate goal of a valid or invalid conclusion, and never needs to backtrack. It continues to work unidirectionally until validity or invalidity is established, or until no more simplifications are applicable.

The simplifications `SplitDecision` is allowed to perform are defined by specialized decision procedures geared specifically to handle the mathematical theories defined in RESOLVE. The goal of our work is for `SplitDecision` to be able to either prove/disprove a RESOLVE VC, or at least reduce it to a form that a proof assistant will need no further guidance to prove or disprove a VC. This requires not only substantial software development, but also the theoretical work of developing and proving the validity of these decision procedures. Thus far `SplitDecision` performs simplifications that make use of simple arithmetic, and also a complete decision procedure for a substantial fragment of string theory.

3.3 `Z3` and others

Microsoft Research’s `Z3` [9] is one of many theorem provers designed with software verification in mind, and is now being used in many different Microsoft applications. We currently have limited support for `Z3`, by adding some of the decision procedures we have developed for string theory to it. This gives us the ability to analyze VCs with multiple tools concurrently, with the knowledge that if any one of them find it valid, it is so. The intent is to expand the theories

in Z3, and add additional theorem provers as time progresses. As more theories are added and more VCs are processed, a more robust comparison of our three tools can be undertaken.

4 Current and Future Work

Currently the VC generation supports only client usage of software components. In the future, we will add support for implementations of software component kernels. As we increase the sophistication of the tool support for programs, we will also increase the mathematical theories use in the specification of components; currently we have mathematical string theory along with a version of finite set theory. We will be adding other theories, including tree theory, to the back-end provers.

5 Acknowledgments

The authors are extremely grateful for the help of Bruce W. Weide, Paolo Bucci, Harvey M. Friedman, Wayne Heym, and Bill Ogden. This work was supported in part by the National Science Foundation under grants DMS-0701187 and DMS-0701260.

References

1. Ogden, W.F., Sitaraman, M., Weide, B.W., Zweben, S.H.: Part I: the RESOLVE framework and discipline: a research synopsis. *SIGSOFT Softw. Eng. Notes* **19**(4) (1994) 23–28
2. Edwards, S.H., Heym, W.D., Long, T.J., Sitaraman, M., Weide, B.W.: Part II: specifying components in RESOLVE. *SIGSOFT Softw. Eng. Notes* **19**(4) (1994) 29–39
3. Bucci, P., Hollingsworth, J.E., Krone, J., Weide, B.W.: Part III: implementing components in RESOLVE. *SIGSOFT Softw. Eng. Notes* **19**(4) (1994) 40–51
4. Hollingsworth, J.E., Sreerama, S., Weide, B.W., Zhupanov, S.: Part IV: RESOLVE components in Ada and C++. *SIGSOFT Softw. Eng. Notes* **19**(4) (1994) 52–63
5. Krone, J.: The Role of Verification in Software Reusability. PhD thesis, Department of Computer and Information Science, The Ohio State University, Columbus, OH (December 1988)
6. Heym, W.D.: Computer Program Verification: Improvements for Human Reasoning. PhD thesis, Department of Computer and Information Science, The Ohio State University, Columbus, OH (December 1995)
7. Sitaraman, M., Atkinson, S., Kulczycki, G., Weide, B.W., Long, T.J., Bucci, P., Heym, W.D., Pike, S.M., Hollingsworth, J.E.: Reasoning about software-component behavior. In: *ICSR-6: Proceedings of the 6th International Conference on Software Reuse*, Springer-Verlag (2000) 266–283
8. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL—A Proof Assistant for Higher-Order Logic. Volume 2283 of *LNCS*. Springer (2002)
9. Microsoft: Z3: SMT solver, <http://research.microsoft.com/projects/z3/>