The RESOLVE Compiler

Gregory Kulczycki
Virginia Tech
gregwk@vt.edu

John Hunt
Clemson University
hunt2@cs.clemson.edu

Murali Sitaraman
Clemson University
murali@cs.clemson.edu

1. ABSTRACT

Resolve is a unique specification-implementation language combination designed to promote a modular style of software development. The intent is to allow design and deployment of components that are correct and efficient, easy for clients to understand and reuse, and easy for implementation developers to adapt and maintain. Furthermore, alternative component implementations of the same specification of functionality must be available to provide component users performance trade-offs, and their performance specifications must be documented. To achieve these ambitious goals, Resolve design is not constrained by popular programming language practices or paradigms. For the promise of automated modular verification, Resolve requires contract specifications for components, and internal implementation assertions such as loop invariants, representation invariants, and abstraction relations. To eliminate routine aliasing complexity and permit direct reasoning of implementations, Resolve requires the use of a constant-time swap operator in component implementations instead of assignments. While new principles for specification and development have to be learned to use Resolve effectively, they should not be an impediment to competent programmers or students.

A current version of the Resolve compiler may be found at `www.cs.clemson.edu/~resolve`. The project vision is to have a verifier for functional and performance correctness of component-based systems.

2. REFERENCES

[1] D. E. Harms and B. W. Weide. Copying and swapping: Influences on the design of reusable software components. *IEEE Transactions on Software Engineering*, 17(5):424–435, May 1991.

[2] J. Krone, W. F. Ogden, and M. Sitaraman. Profiles: A compositional mechanism for performance specification. Technical report, Department of Computer Science, Clemson University, 2004. Available at `http://www.cs.clemson.edu/~resolve` as Technical Report RSRG-04-03.

[3] G. Kulczycki. *Direct Reasoning*. PhD thesis, Clemson University, 2004.

[4] M. Sitaraman, S. Atkinson, G. Kulczycki, B. W. Weide, T. J. Long, P. Bucci, W. Heym, S. Pike, and J. E. Hollingsworth. Reasoning about software-component behavior. In *Procs. Sixth Int. Conf. on Software Reuse*, pages 266–283. Springer-Verlag, 2000.

[5] M. Sitaraman, T. J. Long, B. W. Weide, E. J. Harner, and L. Wang. A formal approach to component-based software engineering: education and evaluation. In *Proceedings of the 23rd international conference on Software engineering*, pages 601–609. IEEE Computer Society, 2001.

[6] M. Sitaraman and B. W. Weide. Component-based software using RESOLVE. *ACM Software Engineering Notes*, 19(4):21–67, 1994.