Compiler correctness

From Wikipedia, the free encyclopedia

In computing, **compiler correctness** is the branch of computer science that deals with trying to show that a compiler behaves according to its language specification. Techniques include developing the compiler using formal methods and using rigorous testing (often called compiler validation) on an existing compiler.

Contents

Formal methods

Compiler validation with formal methods involves a long chain of formal, deductive logic.[1] However, since the tool to find the proof (theorem prover) is implemented in software and is complex, there is a high probability it will contain errors. One approach has been to use a tool that verifies the proof (a proof checker) which because it is much simpler than a proof-finder is less likely to contain errors.

The most complete example of this approach is CompCert, which is a formally verified optimizing compiler of a large a subset of C99.[2][3][4]

Methods include model checking, formal verification, and provably correct semantics-directed compiler generation.[5]

Testing

Testing represents a significant portion of the effort in shipping a compiler, but receives comparatively little coverage in the standard literature. The 1986 edition of Aho, Sethi, & Ullman has a single-page section on compiler testing, with no named examples.[6] The 2006 edition omits the section on testing, but does emphasize its importance: “Optimizing compilers are so difficult to get right that we dare say that no optimizing compiler is completely error-free! Thus, the most important objective in writing a compiler is that it is correct.”[7] Fraser & Hanson 1995 has a brief section on regression testing; source code is available.[8] Bailey & Davidson 2003 cover testing of procedure calls[9] A number of articles confirm that many released compilers have significant code-correctness bugs.[10] Sheridan 2007 is probably the most recent journal article on general compiler

testing.[11] Commercial compiler compliance validation suites are available from Solid Sands,[12] Perennial,[13] and Plum-Hall.[14] For most purposes, the largest body of information available on compiler testing are the Fortran[15] and Cobol[16] validation suites.

Further common techniques when testing compilers are fuzzing[17] (which generates random programs to try to find bugs in a compiler) and test case reduction (which tries to minimize a found test case to make it easier to understand).[18]

See also

- Compiler
- Verification and validation (software)
- Correctness (computer science)
- CompCert C compiler—Formally verified C compiler
- Reflections on Trusting Trust

References

1. De Millo, R. A.; Lipton, R. J.; Perlis, A. J. (1979). "Social processes and proofs of theorems and programs". *Communications of the ACM*. **22** (5): 271–280. doi:10.1145/359104.359106 (https://doi.org/10.1145%2F359104.359106).
2. Leroy, X. (2006). "Formal Certification of a Compiler Back-End or: Programming a Compiler with a Proof Assistant". *ACM SIGPLAN Notices*. **41**: 42–54. doi:10.1145/1111320.1111042 (https://doi.org/10.1145%2F1111320.1111042).
3. Leroy, Xavier (2009-12-01). "A Formally Verified Compiler Back-end" (https://link.springer.com/article/10.1007/s10817-009-9155-4). *Journal of Automated Reasoning*. **43** (4): 363. doi:10.1007/s10817-009-9155-4 (https://doi.org/10.1007%2Fs10817-009-9155-4). ISSN 0168-7433 (https://www.worldcat.org/issn/0168-7433).
4. "CompCert - The CompCert C compiler" (http://compcert.inria.fr/compcert-C.html). *compcert.inria.fr*. Retrieved 2017-07-21.
5. Stephan Diehl, *Natural Semantics Directed Generation of Compilers and Abstract Machines*,Formal Aspects of Computing, Vol. 12 (2), Springer Verlag, 2000. doi:10.1007/PL00003929 (https://doi.org/10.1007%2FPL00003929)
6. *Compilers: Principles, Techniques and Tools*, *infra* 1986, p. 731.
7. *ibid*, 2006, p. 16.
8. Christopher Fraser; David Hanson (1995). *A Retargetable C compiler: Design and Implementation*. Benjamin/Cummings Publishing. ISBN 0-8053-1670-1., pp. 531–3.
9. Mark W. Bailey; Jack W. Davidson (2003). "Automatic Detection and Diagnosis of Faults in Generated Code for Procedure Calls" (http://big-oh.cs.hamilton.edu/~bailey/pubs/techreps/TR-2001-1.pdf) (PDF). *IEEE Transactions on Software Engineering*. **29** (11): 1031–1042. doi:10.1109/tse.2003.1245304 (https://doi.org/10.1109%2Ftse.2003.1245304). Retrieved 2009-03-24., p. 1040.
10. E.g., Christian Lindig (2005). "Random testing of C calling conventions" (http://quest-tester.googlecode.com/svn/trunk/doc/lindig-aadebug-2005.pdf) (PDF). *Proceedings of the Sixth International Workshop on Automated Debugging*. ACM. ISBN 1-59593-050-7. Retrieved 2009-03-24., and Eric Eide; John Regehr

(2008). "Volatiles are miscompiled, and what to do about it" (http://www.cs.utah.edu/~regehr/papers/emsoft08-preprint.pdf) (PDF). *Proceedings of the 7th ACM international conference on Embedded software*. ACM. ISBN 978-1-60558-468-3. Retrieved 2009-03-24.
11. Flash Sheridan (2007). "Practical Testing of a C99 Compiler Using Output Comparison" (http://pobox.com/~flash/Practical_Testing_of_C99.pdf) (PDF). *Software: Practice and Experience*. **37** (14): 1475–1488. doi:10.1002/spe.812 (https://doi.org/10.1002%2Fspe.812). Retrieved 2009-03-24. Bibliography at "http://pobox.com/~flash/compiler_testing_bibliography.html" (http://pobox.com/~flash/compiler_testing_bibliography.html). Retrieved 2009-03-13. External link in |title= (help).
12. "http://www.solidsands.nl" (http://www.solidsands.nl). Retrieved 2011-01-15. External link in |title= (help)
13. "http://peren.com/pages/products_set.htm" (http://peren.com/pages/products_set.htm). Retrieved 2009-03-13. External link in |title= (help)
14. "http://plumhall.com/suites.html" (http://plumhall.com/suites.html). Retrieved 2009-03-13. External link in |title= (help)
15. "Source of Fortran validation suite" (http://www.itl.nist.gov/div897/ctg/fortran_form.htm). Retrieved 2011-09-01.
16. "Source of Cobol validation suite" (http://www.itl.nist.gov/div897/ctg/cobol_form.htm). Retrieved 2011-09-01.
17. Chen, Yang; Groce, Alex; Zhang, Chaoqiang; Wong, Weng-Keen; Fern, Xiaoli; Eide, Eric; Regehr, John (2013). "Taming Compiler Fuzzers" (http://doi.acm.org/10.1145/2491956.2462173). *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '13. New York, NY, USA: ACM: 197–208. doi:10.1145/2491956.2462173 (https://doi.org/10.1145%2F2491956.2462173). ISBN 9781450320146.
18. Regehr, John; Chen, Yang; Cuoq, Pascal; Eide, Eric; Ellison, Chucky; Yang, Xuejun (2012). "Test-case Reduction for C Compiler Bugs" (http://doi.acm.org/10.1145/2254064.2254104). *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '12. New York, NY, USA: ACM: 335–346. doi:10.1145/2254064.2254104 (https://doi.org/10.1145%2F2254064.2254104). ISBN 9781450312059.

Retrieved from "https://en.wikipedia.org/w/index.php?title=Compiler_correctness&oldid=793711774"
