Correctness of Compilers for Java-like Languages

James Baxter

Literature Review Seminar
9 December 2014

Outline

Motivation

Compiler Correctness
- Early Work
- Algebraic Approach
- Recent Work

Compiler Correctness for Java-like Languages
- Compilation of Java
- Compiler Correctness for the whole of Java
- Compiler Correctness for subsets of Java

Conclusion

Motivation

- Popularity of Java[1]
- Variants of Java: JavaCard[2], RTSJ[3], SCJ[4], Java ME[5]
- Program correctness relies on compiler/interpreter correctness.
- Testing is usually not sufficient to ensure correctness.

---

[1] J. Gosling, B. Joy, G. L. Steele Jr, G. Bracha, and A. Buckley, *The Java Language Specification*. Addison-Wesley, 2013.

[2] Z. Chen, *Java card technology for smart cards: architecture and programmer's guide*. Addison-Wesley Professional, 2000.

[3] J. Gosling and G. Bollella, *The Real-Time Specification for Java*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000.

[4] D. Locke, B. S. Andersen, B. Brosgol, M. Fulton, T. Henties, J. J. Hunt, J. O. Nielsen, K. Nilsen, M. Schoeberl, J. Tokar, J. Vitek, A. Wellings, *et al.*, *Safety-critical java technology specification*, Draft, version 0.94, The Open Group, Jun. 25, 2013.

[5] Oracle Corporation. (2014), Java Platform, Micro Edition (Java ME), [Online]. Available: `http://www.oracle.com/technetwork/java/embedded/javame/index.html` (visited on 11/25/2014).

Compiler Correctness

Early Work

- McCarthy and Painter[6]
  - Source: simple single-operator expression language
  - Target: simple four-instruction single-register machine
  - Definition of Correctness: partial equality of machine states
- Burstall and Landin[7]
  - Source: expression language similar to McCarthy and Painter's but allowing for more operators
  - Target: several intermediate machines including a stack machine and a machine similar to McCarthy and Painter's
  - Definition of Correctness: construction of homomorphisms between algebras

---

[6] J. McCarthy and J. Painter, "Correctness of a compiler for arithmetic expressions," *Mathematical aspects of computer science*, vol. 1, 1967.

[7] R. M. Burstall and P. J. Landin, "Programs and their proofs: an algebraic approach," in *Machine Intelligence 4*, B. Meltzer and D. Michie, Eds., Edinburgh University Press, 1969, pp. 17–44.

Early Work

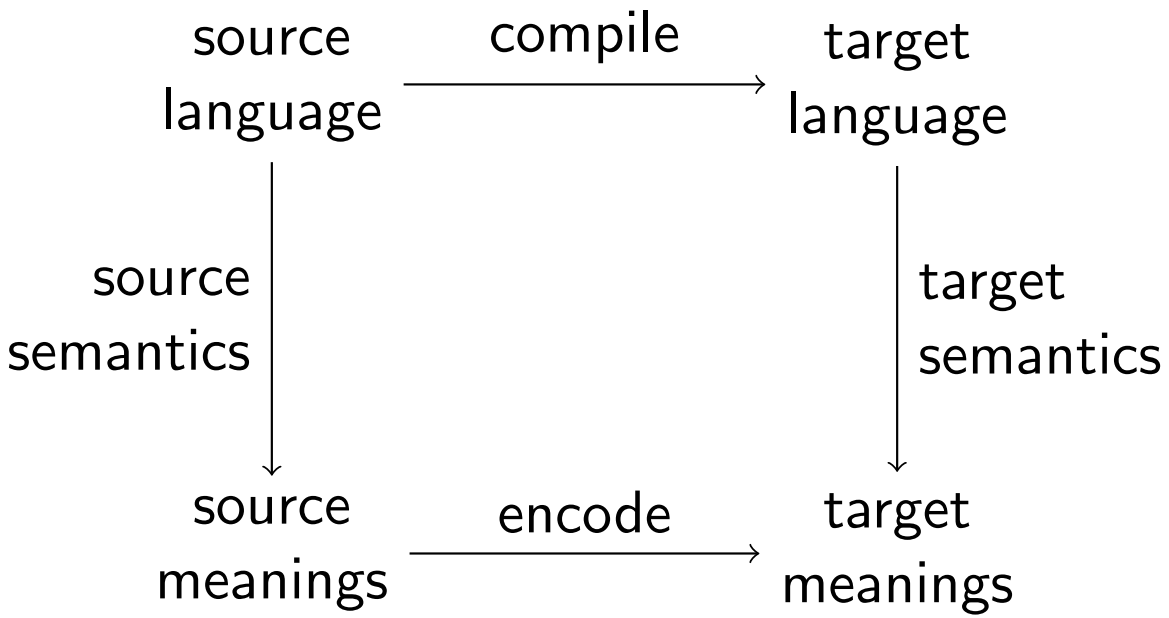
- Milner and Weyhrauch[8]
  - Source: simple ALGOL-like language
  - Target: stack machine that allows jumps
  - Definition of correctness: equality between the source semantics and the composition of the compilation function, the target semantics and a function to extract the source state of the machine
  - Partially mechanised proof using LCF

---

[8] R. Milner and R. Weyhrauch, "Proving compiler correctness in a mechanized logic," *Machine Intelligence*, vol. 7, pp. 51–70, 1972.

Early Work

In general, the traditional approach to compilation is based on showing that a diagram of the following form commutes[9], [10].



[9] F. L. Morris, "Advice on structuring compilers and proving them correct," in *Proceedings of the 1st annual ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, ACM, 1973, pp. 144–152.

[10] J. W. Thatcher, E. G. Wagner, and J. B. Wright, "More on advice on structuring compilers and proving them correct," in *Proceedings of the 6th Colloquium, on Automata, Languages and Programming*, H. A. Maurer, Ed., London, UK: Springer-Verlag, 1979, pp. 596–615.

Algebraic Approach

- Proposed by Hoare in 1991[11]
- Both the source and target languages are described using the *same specification language* with laws for reasoning about them [12].
- Uses the concept of a *refinement relation* between programs: $P \sqsubseteq Q$ means the program Q is at least as good as P [13]–[15].
- Reduces source program to a *normal form* that resembles an interpreter for the target machine.

[11] C. Hoare, "Refinement algebra proves correctness of compiling specifications," in *3rd Refinement Workshop*, C. Morgan and J. Woodcock, Eds., 1991, pp. 33–48.

[12] C. A. R. Hoare, I. J. Hayes, H. Jifeng, C. C. Morgan, A. W. Roscoe, J. W. Sanders, I. H. Sorensen, J. M. Spivey, and B. A. Sufrin, "Laws of programming," *Communications of the ACM*, vol. 30, no. 8, pp. 672–686, 1987.

[13] R. Back, "On correct refinement of programs," *Journal of Computer and System Sciences*, vol. 23, no. 1, pp. 49–68, 1981.

[14] J. M. Morris, "A theoretical basis for stepwise refinement and the programming calculus," *Science of Computer programming*, vol. 9, no. 3, pp. 287–306, 1987.

[15] C. Morgan, *Programming from specifications*. Prentice-Hall, Inc., 1990.

Algebraic Approach

- Sampaio[16]–[18]
  - Handles most imperative constructs, including procedures and recursion.
  - Performs compilation using rewrite rules proved from basic laws.
  - Mechanises compilation in the OBJ3 term rewriting system[19].
- Perna[20], [21] — Hardware compilation
  - Handles timed structures and parallelism with shared variables

[16] C. Hoare, H. Jifeng, and A. Sampaio, "Normal form approach to compiler design," *Acta informatica*, vol. 30, no. 8, pp. 701–739, 1993.

[17] A. Sampaio, "An algebraic approach to compiler design," PhD Thesis, Oxford University Computing Laboratory, 1993.

[18] ——, *An algebraic approach to compiler design*. World Scientific, 1997.

[19] J. Goguen, C. Kirchner, H. Kirchner, A. Mégrelis, J. Meseguer, and T. Winkler, "An introduction to OBJ 3," in *Conditional Term Rewriting Systems*, S. Kaplan and J. P. Jouannaud, Eds., Springer, 1988, pp. 258–263.

[20] J. I. Perna, "A verified compiler for handel-c," PhD Thesis, University of York, 2010.

[21] J. Perna, J. Woodcock, A. Sampaio, and J. Iyoda, "Correct hardware synthesis, An algebraic approach," *Acta informatica*, vol. 48, no. 7-8, pp. 363–396, 2011.

Recent Work

- CompCert[22]
  - Project to develop a realistic formally verified compiler
  - Produced a formally verified compiler for a large subset of C
  - Compiler developed and proved correct in the Coq proof assistant
- Wang, Cuellar and Chipala[23] — Verifying compilers that allow linking with other languages
  - Provides a combined algebraic and operational semantics of the source language
  - Allows for algebraic reasoning about calls to programs in other languages
  - Mechanised in Coq

[22] X. Leroy, *The compcert c verified compiler*, 2012.

[23] P. Wang, S. Cuellar, and A. Chlipala, "Compiler verification meets cross-language linking via data abstraction," in *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, A. Black and T. Millstein, Eds., ACM, 2014, pp. 675–690.

Compiler Correctness for Java-like Languages

Compilation of Java

- Java is usually compiled to Java bytecode, which is run on the JVM.
- The JVM may either interpret the bytecode or compile it to native code.
- Both the initial compilation and the JVM must be proved correct
-

ASM approach

Stärk, Schmid and Börger[24]

- Defines Java and the JVM in terms of abstract state machines (ASMs)
- Splits Java into sublanguages: imperative, procedural, object-oriented, exception-handling, concurrent
- Requires equivalences between the Java ASM and the JVM ASM to be satisfied for the compilation to be correct
- Proves correctness by induction over the structure of Java code

---

[24] R. Stärk, J. Schmid, and E. Börger, *Java and the Java Virtual Machine, Definition, Verification, Validation*. Springer-Verlag, 2001.

ROOL

ROOL (Refinement Object-Oriented Language)[25]

- Subset of Java with specification features
- Verified compiler using the algebraic approach[26], [27]
  - Adds class precompilation and redirection of method calls to Sampaios phases of compilation
  - Normal form representing a VM

---

[25] A. Cavalcanti and D. A. Naumann, "A weakest precondition semantics for refinement of object-oriented programs," *IEEE Transactions on Software Engineering*, vol. 26, no. 8, pp. 713–728, 2000.

[26] A. Duran, "An algebraic approach to the design of compilers for object-oriented languages," PhD Thesis, Universidade Federalde Pernambuco, 2005.

[27] A. Duran, A. Cavalcanti, and A. Sampaio, "An algebraic approach to the design of compilers for object-oriented languages," *Formal aspects of computing*, vol. 22, no. 5, pp. 489–535, 2010.

Java Compilers in Isabelle/HOL

- Strecker[28]
  - Develops a compiler for μJava, a subset of Java that contains many core features of Java but removes interfaces, arrays, access modifiers and concurrency
  - Correctness is shown via a "commuting diagram" argument
- Klein and Nipkow[29]
  - Proves correctness of a compiler and JVM for a slightly larger subset of Java called Jinja
- Lochbihler[30]
  - Adds support for concurrency to the verified compiler presented by Klein and Nipkow

[28] M. Strecker, "Formal verification of a java compiler in isabelle," in *Automated DeductionCADE-18*, A. Voronkov, Ed., Springer, 2002, pp. 63–77.

[29] G. Klein and T. Nipkow, "A machine-checked model for a Java-like language, virtual machine, and compiler," *ACM Transactions on Programming Languages and Systems*, vol. 28, no. 4, pp. 619–695, 2006.

[30] A. Lochbihler, "Verifying a compiler for java threads," in *Programming languages and systems*, A. D. Gordon, Ed., Springer, 2010, pp. 427–447.

Embedded Systems

- Schultz[31] — compiling Java to native code
- Varma and Bhattacharyya[32] — compiling Java to C
- Icecap Hardware Virtual Machine (HVM)[33], [34] — compiling SCJ to C
- No formal verification work done

---

[31] U. P. Schultz, K. Burgaard, F. G. Christensen, and J. L. Knudsen, "Compiling java for low-end embedded systems," in *ACM SIGPLAN Notices*, F. Mueller and U. Kremer, Eds., ACM, vol. 38, 2003, pp. 42–50.

[32] A. Varma and S. S. Bhattacharyya, "Java-through-c compilation: an enabling technology for java in embedded systems," in *Proceedings of the conference on Design, automation and test in Europe-Volume 3*, IEEE Computer Society, 2004, p. 30 161.

[33] H. Søndergaard, S. E. Korsholm, and A. P. Ravn, "Safety-critical Java for low-end embedded platforms," in *Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems*, M. Schoeberl and A. Wellings, Eds., ser. JTRES '12, ACM, 2012, pp. 44–53.

[34] S. E. Korsholm, H. Søndergaard, and A. P. Ravn, "A real-time Java tool chain for resource constrained platforms," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 14, pp. 2407–2431, 2014.

Conclusion

- There are two main approaches to compilation: the algebraic approach and the traditional approach based on commuting diagrams
  - The algebraic approach offers advantages over the more traditional approach
- Work has been done on verifying compilers for various languages, including Java
- There seems to be little work on verifying Java compilers for embedded systems

Any Questions?