Correctness of Compilers for Java-like Languages

James Baxter

Literature Review Seminar
9 December 2014

Outline

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] James Gosling et al. *The Java Language Specification*. Addison-Wesley, 2013.

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

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

[4] D. Locke et al. *Safety-Critical Java Technology Specification*. Draft. Version 0.94. The Open Group, June 25, 2013.

[5] Oracle Corporation. *Java Platform, Micro Edition (Java ME)*. 2014. URL: 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] John McCarthy and James Painter. "Correctness of a compiler for arithmetic expressions". In: *Mathematical aspects of computer science* 1 (1967).

[7] Rodney M Burstall and Peter J Landin. "Programs and their proofs: an algebraic approach". In: *Machine Intelligence 4*. Ed. by Bernard Meltzer and Donald Michie. 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] Robin Milner and Richard Weyhrauch. "Proving compiler correctness in a mechanized logic". In: *Machine Intelligence* 7 (1972), pp. 51–70.

Early Work

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

$$
\begin{tikzcd}
\text{source language} \arrow[r, "\text{compile}"] \arrow[d, "\text{source semantics}"'] & \text{target language} \arrow[d, "\text{target semantics}"] \\
\text{source meanings} \arrow[r, "\text{encode}"] & \text{target meanings}
\end{tikzcd}
$$

[9] F Lockwood 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] James W. Thatcher, Eric G. Wagner, and Jesse B. Wright. "More on Advice on Structuring Compilers and Proving Them Correct". In: *Proceedings of the 6th Colloquium, on Automata, Languages and Programming*. Ed. by Hermann A. Maurer. 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] CAR Hoare. "Refinement algebra proves correctness of compiling specifications". In: *3rd Refinement Workshop*. Ed. by Carroll Morgan and Jim Woodcock. 1991, pp. 33–48.

[12] Charles Antony Richard Hoare et al. "Laws of programming". In: *Communications of the ACM* 30.8 (1987), pp. 672–686.

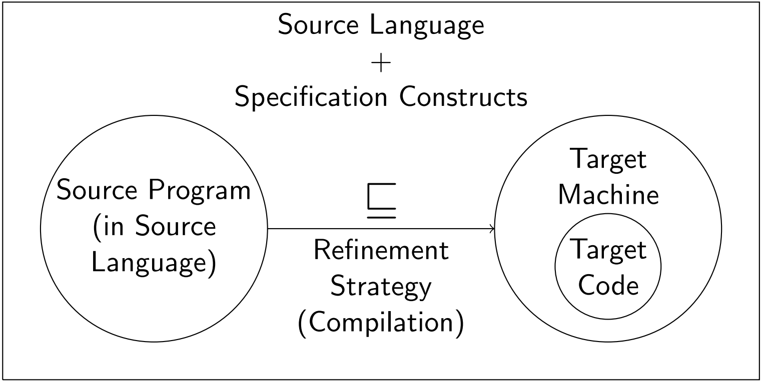
[13] RJR Back. "On correct refinement of programs". In: *Journal of Computer and System Sciences* 23.1 (1981), pp. 49–68.

[14] Joseph M Morris. "A theoretical basis for stepwise refinement and the programming calculus". In: *Science of Computer programming* 9.3 (1987), pp. 287–306.

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

Algebraic Approach

The following diagram illustrates the structure of the algebraic approach.



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] CAR Hoare, He Jifeng, and Augusto Sampaio. "Normal form approach to compiler design". In: *Acta informatica* 30.8 (1993), pp. 701–739.

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

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

[19] Joseph Goguen et al. "An introduction to OBJ 3". In: *Conditional Term Rewriting Systems*. Ed. by S. Kaplan and J. P. Jouannaud. Springer. 1988, pp. 258–263.

[20] Juan Ignacio Perna. "A verified compiler for Handel-C". PhD Thesis. University of York, 2010.

[21] Juan Perna et al. "Correct hardware synthesis. An algebraic approach". In: *Acta informatica* 48.7-8 (2011), pp. 363–396.

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] Xavier Leroy. *The CompCert C verified compiler*. 2012.

[23] Peng Wang, Santiago Cuellar, and Adam 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*. Ed. by Andrew Black and Todd Millstein. 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
- Verifying interpretation is different to verifying compilation

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] Robert Stärk, Joachim Schmid, and Egon 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] Ana Cavalcanti and David A Naumann. "A weakest precondition semantics for refinement of object-oriented programs". In: *IEEE Transactions on Software Engineering* 26.8 (2000), pp. 713–728.

[26] Adolfo Duran. "An Algebraic Approach to the Design of Compilers for Object-Oriented Languages". PhD Thesis. Universidade Federalde Pernambuco, 2005.

[27] Adolfo Duran, Ana Cavalcanti, and Augusto Sampaio. "An algebraic approach to the design of compilers for object-oriented languages". In: *Formal aspects of computing* 22.5 (2010), pp. 489–535.

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] Martin Strecker. "Formal verification of a Java compiler in Isabelle". In: *Automated DeductionCADE-18*. Ed. by Andrei Voronkov. Springer, 2002, pp. 63–77.

[29] Gerwin Klein and Tobias Nipkow. "A Machine-checked Model for a Java-like Language, Virtual Machine, and Compiler". In: *ACM Transactions on Programming Languages and Systems* 28.4 (2006), pp. 619–695.

[30] Andreas Lochbihler. "Verifying a compiler for Java threads". In: *Programming languages and systems*. Ed. by Andrew D. Gordon. 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] Ulrik Pagh Schultz et al. "Compiling java for low-end embedded systems". In: *ACM SIGPLAN Notices*. Ed. by Frank Mueller and Uli Kremer. Vol. 38. 7. ACM. 2003, pp. 42–50.

[32] Ankush Varma and Shuvra 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. 30161.

[33] Hans Søndergaard, Stephan E. Korsholm, and Anders 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*. Ed. by Martin Schoeberl and Andy Wellings. JTRES '12. ACM, 2012, pp. 44–53.

[34] Stephan E. Korsholm, Hans Søndergaard, and Anders P. Ravn. "A real-time Java tool chain for resource constrained platforms". In: *Concurrency and Computation: Practice and Experience* 26.14 (2014), pp. 2407–2431.

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?