History-Based Specification and Verification of Java Collections in KeY (Keynote)

Frank S. de Boer
CWI
Netherlands
F.S.de.Boer@cwi.nl

Hans-Dieter A. Hiep
CWI
Netherlands
Hans-Dieter.Hiep@cwi.nl

Abstract

Software libraries, such as the Java Collection Framework, are used by many applications: thus their correctness is of utmost importance. The state-of-the-art KeY system can be used to formally reason about program correctness of Java programs. Recently, KeY has been used to show major flaws in the Java Collection Framework. However, some methods are challenging for verification, namely those involving parameters of interface type. This lecture discussed a new history-based specification method for reasoning about the correctness of clients and arbitrary implementations of interfaces, and the `Collection` interface in particular.

CCS Concepts: • **Software and its engineering → Formal software verification**.

Keywords: Formal verification; Interface specification; History abstractions

ACM Reference Format:
Frank S. de Boer and Hans-Dieter A. Hiep. 2020. History-Based Specification and Verification of Java Collections in KeY (Keynote). In *Proceedings of the 22th ACM SIGPLAN International Workshop on Formal Techniques for Java-Like Programs (FTfJP '20), July 23, 2020, Virtual, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3427761.3432349

1 Introduction

Software libraries are the building blocks of many programs that run on the devices of many users every day. The functioning of a system relies for a large part on its used software libraries. A small error present in a heavily-used software library could lead to serious unwanted outcomes, such as system outages and other failures. To prevent failures from happening, program correctness is of the utmost importance.

FTfJP '20, July 23, 2020, Virtual, USA

ACM ISBN 978-1-4503-8186-4/20/07.
https://doi.org/10.1145/3427761.3432349

We take a formal approach to both specification and reasoning about Java programs, allowing us to increase the reliability of our reached conclusions to near certainty. In particular, the specifications we write are expressed in the Java Modeling Language (JML), and our reasoning is tool-supported and partially automated by KeY [1]. To the best of the presenters' knowledge, KeY is the only tool that supports enough features of the Java programming language for reasoning about real programs, of which its run-time behavior crucially depends on the presence of features such as: dynamic object creation, exception handling, integer arithmetic with overflow, nested classes, erased generics, etc.

In a recent paper [2], the discovery of a major flaw in the `LinkedList` —erratic behavior caused by an integer overflow—is described: a high-level overview on the specification and verification effort of the corrected `LinkedList` class as a whole is given for a more general audience. In another article [3], more technical details are given on how we use KeY to produce correctness proofs. However, some of the methods of the linked list implementation contain an interface type as parameter and were out of scope of these works. Thus arises the need for an approach to specify interfaces which allows us to verify its implementations and its clients.

The main takeaway of this lecture is a new specification method [4] which advocates the use of *histories* that record invocations of the methods provided by an interface, including their parameters and return value, to describe and reason about the correctness of interfaces, *abstracting and independent from any implementation.* This use of histories is motivated and semantically justified by the basic observation that any *encapsulated* state-based implementation of an interface is completely determined by the invocations of its methods and thus can be derived from the history. Application-specific, user-defined abstractions of histories further allow for high-level interface specifications which also support reasoning about both the correctness of client code and implementations.

We further discussed some of the challenges of proving client code correct with respect to arbitrary implementations, and a practical specification and verification effort of part of the `Collection` interface using KeY.

2 Biography

Frank S. de Boer is professor Program Correctness at Leiden University, and senior researcher at Centrum Wiskunde & Informatica (CWI), the national research institute for mathematics and computer science in the Netherlands. His research interests are semantics and proof theory of object-oriented programming languages featuring mechanisms like object creation, aliasing, method calls, multi-threading, inheritance and subtyping.

Hans-Dieter A. Hiep is a Ph.D. researcher at Leiden University and CWI under the supervision of Frank S. de Boer. His research interests are higher-order logic and type theory and their application to the proof theory of object-oriented programming languages.

References

[1] Wolfgang Ahrendt, Bernhard Beckert, Richard Bubel, Reiner Hähnle, Peter H. Schmitt, and Mattias Ulbrich, editors. *Deductive Software Verification – The KeY Book*, volume 10001 of *LNCS*. Springer, 2016.

[2] Hans-Dieter A. Hiep, Olaf Maathuis, Jinting Bian, Frank S. de Boer, Marko van Eekelen, and Stijn de Gouw. Verifying OpenJDK's LinkedList using KeY. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 12079 of *LNCS*, pages 217–234. Springer, 2020.

[3] Hans-Dieter A. Hiep, Jinting Bian, Frank S. de Boer, and Stijn de Gouw. A tutorial on verifying `LinkedList` using KeY. In Wolfgang Ahrendt, Richard Bubel, Bernhard Beckert, Reiner Hähnle, and Matthias Ulbrich, editors, *Deductive Verification: The State of the Future*, volume 12345 of *LNCS*. Springer, 2020. *(To appear)*.

[4] Hans-Dieter A. Hiep, Jinting Bian, Frank S. de Boer, and Stijn de Gouw. History-based specification and verification of Java collections in KeY. In *16th International Conference on integrated Formal Methods (iFM2020)*, volume 12546 of *LNCS*. Springer, 2020. *(To appear)*.