A case study in class library verification: Java's vector class

(Abstract of [HJB99])

Marieke Huisman, Bart Jacobs, Joachim van den Berg

Computing Science Institute, University of Nijmegen
Toernooiveld 1, 6525 ED Nijmegen, The Netherlands
{marieke,bart,joachim}@cs.kun.nl

One of the reasons for the popularity of object-oriented programming is the possibility it offers for reuse of code. Usually, the distribution of an object-oriented programming language comes together with a collection of ready-to-use classes, in a class library. Typically, these classes contain general purpose code, which can be used in many applications. Before using such classes, a programmer usually wants to know how they behave and when their methods throw exceptions. One way to do this, is to study the actual code, but since this is time-consuming and requires understanding all particular ins and outs of the implementation, this is often not the most efficient way. Another approach is to study the documentation provided. As long as the documentation is clear and concise, this works well, but otherwise one still is forced to look at the actual code.

An alternative approach is to verify suitable properties of standard classes, and add these to the documentation. Examples of properties that can be verified are termination conditions (in which cases will a method terminate normally, in which cases will it throw an exception), pre-post-condition relations and class invariants. Once sufficiently many properties have been verified, one only has to understand these properties, and there is no need anymore to study the actual code, in order to be able to use the class safely.

The full paper describes a case study verification of one particular library class, namely `Vector`, which is in the standard distribution of the programming language Java [AG97,GJS96]. The `Vector` class basically consists of an array of objects, which is internally replaced by an array of different size, according to needs[1]. A verification of an invariant property for this `Vector` class is presented. The property says (essentially) that the actual size of a vector (*i.e.* the number of elements that it contains) is less than or equal to its capacity. This property is maintained by all methods of the `Vector` class, and it holds for all objects created by the constructors of the `Vector` class.

The choice for the `Vector` class is in fact rather arbitrary: it serves our purposes well because it involves a non-trivial amount of code (including the code from its surrounding classes from the library), and gives rise to an interesting invariant. However, other classes than `Vector` could have been verified. The

[1] Arrays in Java have a fixed size; vectors are thus useful if it is not known in advance how many storage positions are needed.

investment of formal verification for library classes can be justified, because these classes are used extremely often. The result of such a verification may be detection of errors, and also improvement of documentation. This particular verification effort did not bring forward errors in the `Vector` class—which would have been unlikely, given how often it is used. However, it pointed out several places where the documentation could be improved.

This verification project makes use of two tools: the PVS [ORR+96,ORSvH95] proof tool, developed at SRI, and the LOOP [JvdBH+98,HHJT98] translation tool. The latter is a compiler which translates Java classes into logical theories in the higher-order logic of PVS. Development of this tool is part of the so-called LOOP project, for Logic of Object-Oriented Programming, in which all three the authors are involved. Initially, this project aimed at reasoning about class specifications (see [HHJT98]), but a new branch of this project concentrated on reasoning about Java [JvdBH+98].

The LOOP tool translates Java classes into appropriate definitions in the language of PVS, by computing a semantical value $[\![s]\!]$, for each (legal) Java expression s. It also generates auxiliary definitions and results. Of particular importance are invariance definitions, which are generated for each class. Actual verification benefits from auxiliary results, which can be used for automatic rewriting. The series of logical theories that is generated when the compiler is applied to a series of Java classes, can be loaded into PVS. After type checking, the user can state the properties (s)he would like to prove about these Java classes, and subsequently (try to) prove them, using the full power of PVS.

The underlying Java semantics that is used in the automatic translation is based on so-called coalgebras [JR97,Rei95,Jac96]. These are special functions, which are useful for describing state-based dynamical systems. In the theory of coalgebras there are standard notions of invariance and bisimulation. Java classes are translated into coalgebras, acting on a single (global) memory (type), consisting of an infinite series of cells for storing objects. The language constructs of Java, like `if-else`, `while`, `try-catch-finally`, are represented in PVS, in what we call a semantic prelude. It is standardly loaded in the theories of translated Java classes. More information about the underlying semantics of Java can be obtained from [HJ99].

The contribution of the work presented in the paper is two fold. First of all, it shows the feasibility of tool-assisted verification of (standard library) classes in Java. The verification results could be used to improve the class documentation and make it exact, in contrast to informal explanations in ordinary language. Secondly, it is an illustration of the use (and capabilities) of the LOOP translation tool. Although the translation does not cover all of Java yet—threads are not incorporated at the moment—it already allows reasoning about real-life Java programs. This is the first time, such a large verification has been done within this project. An important point, worth making explicit, is that this verification is not about programs written is some clean, mathematically civilised, abstract programming language, but about actual Java programs with all their messy details. We consider it a challenge to be able to handle such details.

References

[AG97] K. Arnold and J. Gosling. *The Java Programming Language*. Addison-Wesley, 2^{nd} edition, 1997.

[GJS96] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. Addison-Wesley, 1996.

[HHJT98] U. Hensel, M. Huisman, B. Jacobs, and H. Tews. Reasoning about classes in object-oriented languages: Logical models and tools. In *Proceedings of European Symposium on Programming (ESOP)*, volume 1381 of *LNCS*, pages 105–121. Springer-Verlag, March 1998.

[HJ99] M. Huisman and B. Jacobs. Java program verification via a Hoare logic with abrupt termination. Manuscript, 1999.

[HJB99] M. Huisman, B. Jacobs, and J. van den Berg. A case study in class library verification: Java's Vector class (abstract). In B. Jacobs, G.T. Leavens, P. Müller, and A. Poetzsch-Heffter, editors, *Formal Techniques for Java Programs*, volume 251 - 5/1999 of *Informatik berichte FernUniversität Hagen*, 1999.

[Jac96] B. Jacobs. Objects and classes, co-algebraically. In B. Freitag, C.B. Jones, C. Lengauer, and H.-J. Schek, editors, *Object-Orientation with Parallelism and Persistence*, pages 83–103. Kluwer Acad. Publ., 1996.

[JR97] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62:222–259, 1997.

[JvdBH+98] B. Jacobs, J. van den Berg, M. Huisman, M. van Berkum, U. Hensel, and H. Tews. Reasoning about classes in Java (preliminary report). In *Object-Oriented Programming, Systems, Languages and Applications (OOPSLA)*, pages 329–340. ACM Press, 1998.

[ORR+96] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M.K. Srivas. PVS: Combining specification, proof checking, and model checking. In R. Alur and T.A. Henzinger, editors, *Computer-Aided Verification (CAV '96)*, volume 1102 of *LNCS*, pages 411–414, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[ORSvH95] S. Owre, J. Rushby, N. Shankar, and F. von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, February 1995.

[Rei95] H. Reichel. An approach to object semantics based on terminal co-algebras. *Math. Struct. in Comp. Sci.*, 5:129–152, 1995.