The KeY Approach: Integrating Object Oriented Design and Formal Verification

Wolfgang Ahrendt, Thomas Baar, Bernhard Beckert, Martin Giese, Elmar Habermalz, Reiner Hähnle, Wolfram Menzel, and Peter H. Schmitt

1 University of Karlsruhe, Institute for Logic, Complexity and Deduction Systems, D-76128 Karlsruhe, Germany, http://i12www.ira.uka.de/~key
2 Department of Computing Science, Chalmers University of Technology S-41296 Gothenburg, reiner@cs.chalmers.se

Abstract This paper reports on the ongoing KeY project aimed at bridging the gap between (a) object-oriented software engineering methods and tools and (b) deductive verification. A distinctive feature of our approach is the use of a commercial CASE tool enhanced with functionality for formal specification and deductive verification.

1 Introduction

1.1 Analysis of the Current Situation

While formal methods are by now well established in hardware and system design (the majority of producers of integrated circuits are routinely using BDD-based model checking packages for design and validation), usage of formal methods in software development is currently confined essentially to academic research projects. There are industrial applications of formal software development [8], but they are still exceptional [9].

The limits of applicability of formal methods in software design are not defined by the potential range and power of existing approaches. Several case studies clearly demonstrate that computer-aided specification and verification of realistic software is feasible [18]. The real problem lies in the excessive demand imposed by current tools on the skills of prospective users:

1. Tools for formal software specification and verification are not integrated into industrial software engineering processes.
2. User interfaces of verification tools are not ergonomic: they are complex, idiosyncratic, and are often without graphical support.
3. Users of verification tools are expected to know syntax and semantics of one or more complex formal languages. Typically, at least a tactical programming language and a logical language are involved. And even worse, to make serious use of many tools, intimate knowledge of employed logic calculi and proof search strategies is necessary.
Successful specification and verification of larger projects, therefore, is done separately from software development by academic specialists with several years of training in formal methods, in many cases by the tool developers themselves.

While this is viable for projects with high safety and low secrecy demands, it is unlikely that formal software specification and verification will become a routine task in industry under these circumstances.

The future challenge for formal software specification and verification is to make the considerable potential of existing methods and tools feasible to use in an industrial environment. This leads to the requirements:

1. Tools for formal software specification and verification must be integrated into industrial software engineering procedures.
2. User interfaces of these tools must comply with state-of-the-art software engineering tools.
3. The necessary amount of training in formal methods must be minimized. Moreover, techniques involving formal software specification and verification must be teachable in a structured manner. They should be integrated in courses on software engineering topics.

To be sure, the thought that full formal software verification might be possible without any background in formal methods is utopian. An industrial verification tool should, however, allow for gradual verification so that software engineers at any (including low) experience level with formal methods may benefit. In addition, an integrated tool with well-defined interfaces facilitates “outsourcing” those parts of the modeling process that require special skills.

Another important motivation to integrate design, development, and verification of software is provided by modern software development methodologies which are iterative and incremental. Post mortem verification would enforce the antiquated waterfall model. Even worse, in a linear model the extra effort needed for verification cannot be parallelized and thus compensated by greater workforce. Therefore, delivery time increases considerably and would make formally verified software decisively less competitive.

But not only must the extra time for formal software development be within reasonable bounds, the cost of formal specification and verification in an industrial context requires accountability:

4. It must be possible to give realistic estimations of the cost of each step in formal software specification and verification depending on the type of software and the degree of formalization.

This implies immediately that the mere existence of tools for formal software specification and verification is not sufficient, rather, formal specification and verification have to be fully integrated into the software development process.

1.2 The K¥ Project

Since November 1998 the authors work on a project addressing the goals outlined in the previous section; we call it the K¥ project (read “key”).