

Preface

Wer hat an der Uhr gedreht? Ist es wirklich schon so spät?—Every child growing up in the 1970s in Germany (like the author of this text) is familiar with these lines. They are from a theme song played during the closing credits of the German version of the *Pink Panther Show*. The ditty conveys the utter bafflement (“Who advanced the clock?”) of the listener about 30 minutes of entertainment having passed in what seemed to be mere seconds. And it is exactly this feeling that we editors have now: What? Ten years since the first book about KeY [Beckert et al., 2007] was published? Impossible! But of course it is possible, and there are good reasons for a new book about KeY, this time simply called *The KeY Book*.

What Is New in The KeY Book?

In short: almost everything! This is not merely an overhaul of the previous book, but a completely different volume: only eight of 15 chapters from the previous edition are still around with a roughly similar function, while there are 11 completely new chapters. But even most of the chapters retained were rewritten from scratch. Only three chapters more or less kept their old structure. What happened?

First of all, there were some major technical developments in the KeY system that required coverage as well as changes in the theoretical foundations:

- With KeY 2.x we moved from a memory model with an implicit heap to one with explicit heaps, i.e., heaps have a type in our logic, can be quantified over, etc. As a consequence, it was possible to:
- Implement better support for reasoning about programs with heaps [Weiß, 2011, Schmitt et al., 2010], essentially with a variant of dynamic frames [Kassios, 2011].
- We dropped support for specification with the Object Constraint Language (OCL) and drastically improved support for the Java Modeling Language (JML) [Leavens et al., 2013].
- Rules and automation heuristics for a number of logical theories were added, including finite sequences, strings [Bubel et al., 2011], and bitvectors.
- Abstract interpretation was tightly integrated with logic-based symbolic execution [Bubel et al., 2009].

In addition, the functionality of the KeY was considerably extended. This concerns not merely the kind of analyses that are possible, but also usability.

- Functional verification is now only one of many analyses that can be performed with the KeY system. In addition, there is support for debugging and visualization [Hentschel et al., 2014a,b], test case generation [Engel and Hähnle, 2007, Gladisch, 2008], information flow analysis [Darvas et al., 2005, Grahl, 2015, Do et al., 2016], program transformation [Ji et al., 2013], and compilation [Ji, 2014].

- There are IDEs for KeY, including an Eclipse extension, that make it easy to keep track of proof obligations in larger projects [Hentschel et al., 2014c].
- A stripped down version of KeY, specifically developed for classroom exercises with Hoare logic, was provided [Hähnle and Bubel, 2008].

Finally, the increased maturity and coverage of the KeY system permitted us to include much more substantial case studies than in the first edition.

Inevitably, some of the research strands documented in the first edition did not reach the maturity or importance we hoped for and, therefore, were dropped. This is the case for specification patterns, specification in natural language, induction, and proof reuse. There is also no longer a dedicated chapter about Java integers: The relevant material is now, in much condensed form, part of the ‘chapters on “First-Order Logic” and “Theories.”’

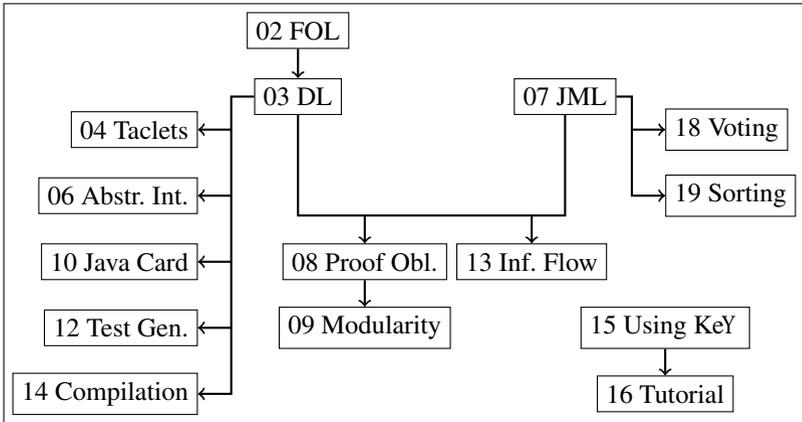
A number of topics that are actively researched have not yet reached sufficient maturity to be included: merging nodes in symbolic execution proof trees, KeY for Java bytecode, certification, runtime verification, regression verification, to name just a few. Also left out is a variant of the KeY system for the concurrent modeling language ABS called KeY-ABS, which allows one to prove complex properties about unbounded programs and data structures. We are excited about all these developments, but we feel that they have not yet reached the maturity to be documented in the KeY book. We refer the interested reader to the research articles available from the KeY website at www.key-project.org.

Also not covered in this book is KeYmaera, a formal verification tool for hybrid, cyber-physical systems developed in André Platzer’s research group at CMU and that has KeY as an ancestor. It deserves a book in its own right, which in fact has been written [Platzer, 2010].

The Concept Behind This Book

Most books on foundations of formal specification and verification orient their presentation along traditional lines in logic. This results in a gap between the foundations of verification and its application that is too wide for most readers. The main presentation principles of the KeY book is something that has *not* been changed between the first book about KeY and this volume:

- The material is presented on an advanced level suitable for graduate (MSc level) courses, and, of course, active researchers with an interest in verification.
- The dependency graph on the chapters in the book is not deep, such that the reader does not have to read many chapters before the one (s)he is most interested in. Moreover, the dependencies are not all too strong. More advanced readers may not have to strictly follow the graph, and less advanced readers may decide to follow up prerequisites on demand. The graph shows that the chapters on First-Order Logic, the Java Modeling Language, and Using the KeY Prover are entirely self-contained. The same holds for each chapter not appearing in the following graph.



- The underlying verification paradigm is deductive verification in an expressive program logic.
- As a rule, the proofs of theoretical results are not contained here, but we give pointers on where to find them.
- The logic used for reasoning about programs is not a minimalist version suitable for theoretical investigations, but an industrial-strength version. The first-order part is equipped with a type system for modeling of object hierarchies, with underspecification, and with various built-in theories. The program logic covers full Java Card and substantial parts of Java. The main omissions are: generics (a transformation tool is available), floating-point types, threads, lambda expressions.
- Much emphasis is on specification, including the widely used JML. The generation of proof obligations from annotated source code is discussed at length.
- Two substantial case studies are included and presented in detail.

Nevertheless, we cannot and do not claim to have fully covered formal reasoning about (object-oriented) software in this book. One reason is that the choice of topics is dependent on our research agenda. As a consequence, important topics in formal verification, such as specification refinement or model checking, are out of our scope.

Typographic Conventions

We use a number of typesetting conventions to give the text a clearer structure. Occasionally, we felt that a historical remark, a digression, or a reference to material outside the scope of this book is required. In order to not interrupt the text flow we use gray boxes, such as the one on page 40, whenever this is the case.

In this book a considerable number of specification and programming languages are referred to and used for illustration. To avoid confusion we usually typeset multiline expressions from concrete languages in a special environment that is set apart from the

project “Deductive Regression Verification for Evolving Object-Oriented Software” within Priority Programme 1593 “Design for Future: Managed Software Evolution,” as well as support by the German Federal Ministry of Education and Research (Bundesministerium für Bildung und Forschung, BMBF) for project “Formale Informationsflussspezifikation und -analyse in komponentenbasierten Systemen” as part of Software Campus. We also acknowledge support by the European Commission (EC) under FP7 Integrated Project “Highly Adaptable & Trustworthy Software using Formal Methods” and STREP “Engineering Virtualized Services.”

The authors have done their job. The success of the result of their toil will now be judged by the readers. We hope they will find the text easily accessible, illuminating, stimulating and, yes, fun to read. What can be more fun than gaining insight into what was nebulous before or solving a problem at hand that defied a solution so far? We will not fail to admit that besides all the labor, writing this book was fun. Putting in words often helped us reach a better understanding of what we were writing about. Thus, it is not easy to say who profits more, the authors in writing the book or the readers from reading it. In the end this may be irrelevant, authors and readers together constitute the scientific community and together they advance the state of the art. It would be the greatest reward for us to see this happening and perhaps after another ten years, or even earlier, a return of the pink panther.

August 2016

Wolfgang Ahrendt
Bernhard Beckert
Richard Bubel
Reiner Hähnle
Peter H. Schmitt
Mattias Ulbrich



<http://www.springer.com/978-3-319-49811-9>

Deductive Software Verification - The KeY Book

From Theory to Practice

Ahrendt, W.; Beckert, B.; Bubel, R.; Hähnle, R.; Schmitt,

P.H.; Ulbrich, M. (Eds.)

2016, XXXII, 702 p. 110 illus., Softcover

ISBN: 978-3-319-49811-9