

Contents

List of Figures	XXV
List of Tables	XXIX
List of Listings	XXXI
1 Quo Vadis Formal Verification?	1
1.1 What KeY Is	1
1.2 Challenges To Formal Verification	2
1.3 Roles of Deductive Verification	4
1.3.1 Avoid the Need for Formal Specification	4
1.3.2 Restricted Target Language	5
1.3.3 Formal Verification in Teaching	6
1.3.4 Avoid the Need for Fully Specified Formal Semantics	8
1.3.5 Where Are We Now?	8
1.4 The Architecture of KeY	10
1.4.1 Prover Core	10
1.4.2 Reasoning About Programs	12
1.4.3 Proof Obligations	14
1.4.4 The Frontends	15
1.5 The Next Ten Years	16
1.5.1 Modular Architecture	16
1.5.2 Relational Properties Are Everywhere	17
1.5.3 Learning from Others	18
1.5.4 Code Reviews	18
1.5.5 Integration	18

Part I Foundations

- 2 First-Order Logic** 23
 - 2.1 Introduction 23
 - 2.2 Basic First-Order Logic 23
 - 2.2.1 Syntax 23
 - 2.2.2 Calculus 27
 - 2.2.3 Semantics 31
 - 2.3 Extended First-Order Logic 35
 - 2.3.1 Variable Binders 36
 - 2.3.2 Undefinedness 37
 - 2.4 First-Order Logic for Java 37
 - 2.4.1 Type Hierarchy and Signature 38
 - 2.4.2 Axioms for Integers 39
 - 2.4.3 Axioms for Heap 40
 - 2.4.4 Axioms for Location Sets 43
 - 2.4.5 Semantics 44

- 3 Dynamic Logic for Java** 49
 - 3.1 Introduction 49
 - 3.2 Syntax of JavaDL 50
 - 3.2.1 Type Hierarchies 51
 - 3.2.2 Signatures 51
 - 3.2.3 Syntax of JavaDL Program Fragments 52
 - 3.2.4 Syntax of JavaDL Terms and Formulas 54
 - 3.3 Semantics 55
 - 3.3.1 Kripke Structures 55
 - 3.3.2 Semantics of JavaDL Terms and Formulas 56
 - 3.4 Describing Transitions between States: Updates 57
 - 3.4.1 Syntax and Semantics of JavaDL Updates 57
 - 3.4.2 Update Simplification Rules 59
 - 3.5 The Calculus for JavaDL 60
 - 3.5.1 JavaDL Rule Schemata and First-Order Rules 60
 - 3.5.2 Nonprogram Rules for Modalities 63
 - 3.5.3 Soundness and Completeness of the Calculus 63
 - 3.5.4 Schema Variables for Program Constructs 66
 - 3.5.5 The Active Statement in a Modality 67
 - 3.5.6 The Essence of Symbolic Execution 67
 - 3.5.7 Components of the Calculus 68
 - 3.6 Rules for Symbolic Execution of Java Programs 69
 - 3.6.1 The Basic Assignment Rule 69
 - 3.6.2 Rules for Handling General Assignments 70
 - 3.6.3 Rules for Conditionals 75
 - 3.6.4 Unwinding Loops 76
 - 3.6.5 Replacing Method Calls by their Implementation 77
 - 3.6.6 Instance Creation and Initialization 87
 - 3.6.7 Handling Abrupt Termination 93

- 3.7 Abstraction and Modularization Rules 97
 - 3.7.1 Replacing Method Calls by Specifications: Contracts 98
 - 3.7.2 Reasoning about Unbounded Loops: Loop Invariants 101
- 4 Proof Search with Taclets 107**
 - 4.1 Introduction 107
 - 4.1.1 Purpose and Organization of this Chapter 108
 - 4.2 A Taclet Tutorial 109
 - 4.2.1 A Basic Theory of Lists 109
 - 4.2.2 The List Theory in Concrete Syntax 111
 - 4.2.3 Definitional Extension of the List Theory 114
 - 4.2.4 Derivation of Lemmas 117
 - 4.3 A Reference Manual of Taclets 121
 - 4.3.1 The Taclet Language 121
 - 4.3.2 Schema Variables 130
 - 4.3.3 Application of Taclets 136
 - 4.4 Reflection and Reasoning about Soundness of Taclets 138
 - 4.4.1 Soundness in Sequent Calculi 139
 - 4.4.2 Meaning Formulas of Sequent Taclets 140
 - 4.4.3 Meaning Formulas for Rewriting Taclets 142
 - 4.4.4 Elimination of Schema Variables 143
- 5 Theories 149**
 - 5.1 Introduction 149
 - 5.2 Finite Sequences 149
 - 5.3 Strings 156
 - 5.3.1 Sequences of Characters 156
 - 5.3.2 Regular Expressions for Sequences 158
 - 5.3.3 Relating Java String Objects to Sequences of Characters 159
 - 5.3.4 String Literals and the String Pool 160
 - 5.3.5 Specification of the Java String API 161
 - 5.4 Integers 161
 - 5.4.1 Core Integer Theory 162
 - 5.4.2 Variable Binding Integer Operations 163
 - 5.4.3 Symbolic Execution of Integer Expressions in Programs 164
- 6 Abstract Interpretation 167**
 - 6.1 Introduction 167
 - 6.2 Integrating Abstract Interpretation 168
 - 6.2.1 Abstract Domains 168
 - 6.2.2 Abstractions for Integers 170
 - 6.2.3 Abstracting States 170
 - 6.3 Loop Invariant Generation 171
 - 6.4 Abstract Domains for Heaps 174
 - 6.5 Abstract Domains for Objects 178
 - 6.5.1 Null/Not-null Abstract Domain 179

6.5.2	Length Abstract Domain	179
6.6	Extensions	184
6.6.1	Abstractions for Arrays	184
6.6.2	Loop Invariant Rule with Value and Array Abstraction	184
6.6.3	Computation of the Abstract Update and Invariants	186
6.6.4	Symbolic Pivots	187
6.7	Conclusions	189

Part II Specification and Verification

7	Formal Specification with the Java Modeling Language	193
7.1	Introduction to Method Contracts	196
7.1.1	Clauses of a Contract	196
7.1.2	Defensive Versus Offensive Method Implementations	199
7.1.3	Specifications and Implementations	200
7.2	Expressions	201
7.2.1	Quantified Boolean Expressions	201
7.2.2	Numerical Comprehensions	203
7.2.3	Evaluation in the Prestate	204
7.3	Method Contracts in Detail	205
7.3.1	Visibility of Specifications	205
7.3.2	Specification Cases	206
7.3.3	Semantics of Normal Behavior Specification Cases	208
7.3.4	Specifications for Constructors	209
7.3.5	Notions of Purity	209
7.4	Class Level Specifications	210
7.4.1	Invariants	211
7.4.2	Initially Clauses	216
7.4.3	History Constraints	217
7.4.4	Initially Clauses and History Constraints: Static vs. Instance	218
7.4.5	Inheritance of Specifications	218
7.5	Nonnull Versus Nullable Object References	220
7.6	Exceptional Behavior	222
7.7	Specification-Only Class Members	225
7.7.1	Model Fields and Model Methods	225
7.7.2	Ghost Variables	229
7.7.3	Ghost Variables Versus Model Fields	229
7.8	Integer Semantics	230
7.9	Auxiliary Specification for Verification	233
7.9.1	Framing	233
7.9.2	Loop Invariants	234
7.9.3	Assertions and Block Contracts	238

- 7.10 Conclusion 239
 - 7.10.1 Tool Support for JML 239
 - 7.10.2 Comparison to Other Program Annotation Languages 240
- 8 From Specification to Proof Obligations 243**
 - 8.1 Formal Semantics of JML Expressions 244
 - 8.1.1 Types in JML 245
 - 8.1.2 Translating JML Expressions to JavaDL 246
 - 8.1.3 Abstract Data Types in JML 252
 - 8.1.4 Well-Definedness of Expressions 253
 - 8.2 From JML Contract Annotations to JavaDL Contracts 254
 - 8.2.1 Normalizing JML Contracts 255
 - 8.2.2 Constructor Contracts 264
 - 8.2.3 Model Methods and Model Fields 265
 - 8.2.4 JavaDL Contracts 268
 - 8.2.5 Loop Specifications 271
 - 8.3 Proof Obligations for JavaDL Contracts 272
 - 8.3.1 Proof Obligations for Functional Correctness 272
 - 8.3.2 Dependency Proof Obligations 278
 - 8.3.3 Well-Definedness Proof Obligations 279
- 9 Modular Specification and Verification 289**
 - 9.1 Modular Verification with Contracts 291
 - 9.1.1 Historical and Conceptual Background 291
 - 9.1.2 Example: Implementing a List 294
 - 9.1.3 Modular Program Correctness 296
 - 9.1.4 Verification of Recursive Methods 298
 - 9.2 Abstract Specification 300
 - 9.2.1 Model Fields 302
 - 9.2.2 Model Methods 311
 - 9.3 The Frame Problem 319
 - 9.3.1 Motivation 320
 - 9.3.2 Dynamic Frames 322
 - 9.3.3 Proof Obligations for Dynamic Frames Specifications 324
 - 9.3.4 Example Specification with Dynamic Frames 325
 - 9.4 Calculus Rules for Modular Reasoning 328
 - 9.4.1 Anonymizing Updates 329
 - 9.4.2 An Improved Loop Invariant Rule 330
 - 9.4.3 A Rule for Method Contracts 336
 - 9.4.4 A Rule for Dependency Contracts 339
 - 9.4.5 Rules for Class Invariants 341
 - 9.5 Verifying the List Example 343
 - 9.6 Related Methodologies for Modular Verification 347
 - 9.7 Conclusion 351

10 Verifying Java Card Programs	353
10.1 Introduction	353
10.2 Java Card Technology	354
10.3 Java Card Transactions on Explicit Heaps	357
10.3.1 Basic Transaction Roll-Back	358
10.3.2 Transaction Marking and Balancing	359
10.3.3 Object Creation and Deletion	360
10.3.4 Persistent and Transient Arrays	361
10.3.5 Nonatomic Updates	363
10.4 Tactlets for the New Rules	364
10.5 Modular Reasoning with Multiple Heaps	367
10.6 Java Card Verification Samples	368
10.6.1 Conditional Balance Updating	369
10.6.2 Reference Implementation of a Library Method	370
10.6.3 Transaction Resistant PIN Try Counter	374
10.7 Summary and Discussion	376
10.7.1 Related Work	377
10.7.2 On-going and Future Research	377
10.7.3 Multiple Heaps for Concurrent Reasoning	377

Part III From Verification to Analysis

11 Debugging and Visualization	383
11.1 Introduction	383
11.2 Symbolic Execution	385
11.3 Symbolic Execution Debugger	390
11.3.1 Installation	390
11.3.2 Basic Usage	391
11.3.3 Debugging with Symbolic Execution Trees	395
11.3.4 Debugging with Memory Layouts	397
11.3.5 Help Program and Specification Understanding	398
11.3.6 Debugging Meets Verification	399
11.3.7 Architecture	401
11.4 A Symbolic Execution Engine based on KeY	403
11.4.1 Symbolic Execution Tree Generation	403
11.4.2 Branch and Path Conditions	407
11.4.3 Hiding the Execution of Query Methods	408
11.4.4 Symbolic Call Stack	409
11.4.5 Method Return Values	409
11.4.6 Current State	410
11.4.7 Controlled Execution	411
11.4.8 Memory Layouts	411
11.5 Conclusion And Future Work	412

- 12 Proof-based Test Case Generation 415**
- 12.1 Introduction 415
- 12.2 A Quick Tutorial 416
 - 12.2.1 Setup 417
 - 12.2.2 Usage 417
 - 12.2.3 Options and Settings 419
- 12.3 Test Cases, Test Suites, and Test Criteria 421
- 12.4 Application Scenarios and Variations of the Test Generator 426
 - 12.4.1 KeYTestGen for Test Case Generation 426
 - 12.4.2 KeYTestGen for Formal Verification 427
- 12.5 Architecture of KeYTestGen 428
- 12.6 Proof-based Constraint Construction for Test Input Data 430
 - 12.6.1 Symbolic Execution for Test Constraint Generation 430
 - 12.6.2 Implicit Case Distinctions 431
 - 12.6.3 Infeasible Path Filtering 432
 - 12.6.4 Using Loop Unwinding and Method Inlining 433
 - 12.6.5 Using Loop Invariants and Method Contracts 434
- 12.7 From Constraints to Test Input Data 437
 - 12.7.1 The Type System 439
 - 12.7.2 Preserving the Semantics of Interpreted Functions 440
 - 12.7.3 Preventing Integer Overflows 441
 - 12.7.4 Model Extraction 442
- 12.8 Specification-based Test Oracle Generation 442
 - 12.8.1 Generating a Test Oracle from the Postcondition 442
 - 12.8.2 Using a Runtime Assertion Checker 445
- 12.9 Synthesizing Executable Test Cases 445
- 12.10 Perspectives and Related Work 448
- 12.11 Summary and Conclusion 450
- 13 Information Flow Analysis 453**
- 13.1 Introduction 453
- 13.2 Specification and the Attacker Model 455
- 13.3 Formal Definition of Secure Information Flow 457
- 13.4 Specifying Information Flow in JML 458
- 13.5 Information Flow Verification with KeY 460
 - 13.5.1 Efficient Double Symbolic Execution 461
 - 13.5.2 Using Efficient Double Symbolic Execution in KeY 469
- 13.6 Summary and Conclusion 470
- 14 Program Transformation and Compilation 473**
- 14.1 Interleaving Symbolic Execution and Partial Evaluation 474
 - 14.1.1 General Idea 474
 - 14.1.2 The Program Specialization Operator 478
 - 14.1.3 Specific Specialization Actions 479
 - 14.1.4 Example 481
- 14.2 Verified Correct Compilation 482

- 14.3 Implementation and Evaluation 491
- 14.4 Conclusion 492

Part IV The KeY System in Action

- 15 Using the KeY Prover 495**
 - 15.1 Introduction 495
 - 15.2 Exploring KeY Artifacts and Prover Simultaneously 498
 - 15.2.1 Exploring Basic Notions And Usage 499
 - 15.2.2 Exploring Terms, Quantification, and Instantiation 511
 - 15.2.3 Exploring Programs in Formulas 515
 - 15.3 Understanding Proof Situations 532
 - 15.4 Further Features 537
 - 15.5 What Next? 539
- 16 Formal Verification with KeY: A Tutorial 541**
 - 16.1 Introduction 541
 - 16.2 A Program without Loops 543
 - 16.3 A Brief Primer on Loop Invariants 546
 - 16.3.1 Introduction 546
 - 16.3.2 Why Are Loop Invariants Needed? 547
 - 16.3.3 What Is A Loop Invariant? 547
 - 16.3.4 Goal-Oriented Derivation of Loop Invariants 549
 - 16.3.5 Generalization 550
 - 16.3.6 Recovering the Context 551
 - 16.3.7 Proving Termination 553
 - 16.3.8 A More Complex Example 555
 - 16.3.9 Invariants: Concluding Remarks 557
 - 16.4 A Program with Loops 558
 - 16.5 Data Type Properties of Programs 562
 - 16.6 KeY and Eclipse 565
 - 16.6.1 Installation 566
 - 16.6.2 Proof Management with KeY Projects 566
 - 16.6.3 Proof Results via Marker 567
 - 16.6.4 The Overall Verification Status 568
- 17 KeY-Hoare 571**
 - 17.1 Introduction 571
 - 17.2 The Programming Language 572
 - 17.3 Background 573
 - 17.3.1 First-Order Logic 573
 - 17.3.2 Hoare Calculus 574
 - 17.4 Hoare Logic with Updates 575
 - 17.4.1 State Updates 576
 - 17.4.2 Hoare Triples with Update 577

- 17.4.3 Hoare Style Calculus with Updates 577
- 17.4.4 Rules for Updates 579
- 17.5 Using KeY-Hoare 580
- 17.6 Variants of the Hoare Logic with Updates 582
 - 17.6.1 Total Correctness 582
 - 17.6.2 Worst-Case Execution Time 582
- 17.7 A Brief Reference Manual for KeY-Hoare 584
 - 17.7.1 Installation 584
 - 17.7.2 Formula Syntax 585
 - 17.7.3 Input File Format 585
 - 17.7.4 Loading and Saving Problems and Proofs 587
 - 17.7.5 Proving 588
 - 17.7.6 Automation 589

Part V Case Studies

- 18 Verification of an Electronic Voting System 593**
 - 18.1 Electronic Voting 593
 - 18.2 Overview 594
 - 18.2.1 Verification of Cryptographic Software 595
 - 18.2.2 Verification Approach 595
 - 18.2.3 System Description 596
 - 18.3 Specification and Verification 599
 - 18.3.1 Specification 599
 - 18.3.2 Verification 603
 - 18.4 Discussion 605
 - 18.4.1 A Hybrid Approach to Information Flow Analysis 605
 - 18.4.2 Related Work 606
 - 18.4.3 Conclusion 607
- 19 Verification of Counting Sort and Radix Sort 609**
 - 19.1 Counting Sort and Radix Sort Implementation 609
 - 19.2 High-Level Correctness Proof 612
 - 19.2.1 General Auxiliary Functions 613
 - 19.2.2 Counting Sort Proof 614
 - 19.2.3 Radix Sort Proof 615
 - 19.3 Experience Report 617

Part VI Appendices

- A Java Modeling Language Reference 621**
 - A.1 JML Syntax 621
 - A.2 JML Expression Semantics 625
 - A.3 JML Expression Well-Definedness 628

- B KeY File Reference** 631
- B.1 Predefined Operators in JavaDL 631
 - B.1.1 Arithmetic Functions 631
 - B.1.2 Arithmetic Functions with Modulo Semantics 632
 - B.1.3 Predicates for Arithmetics and Equality 634
 - B.1.4 Integer Semantics Dependent Functions 634
 - B.1.5 Integer Semantics Dependent Predicate Symbols 636
 - B.1.6 Heap Related Function and Predicate Symbols 636
 - B.1.7 Location Sets Related Function and Predicate Symbols 637
 - B.1.8 Finite Sequence Related Function and Predicate Symbols 638
 - B.1.9 Map Related Function and Predicate Symbols 639
- B.2 The KeY Syntax 640
 - B.2.1 Notation, Keywords, Identifiers, Numbers, Strings 641
 - B.2.2 Terms and Formulas 643
 - B.2.3 Rule Files 651
 - B.2.4 User Problem and Proof Files 659
 - B.2.5 Schematic Java Syntax 662
- References** 667
- Index** 691



<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