

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
2 First-Order Logic	21
2.1 Introduction	21
2.2 Basic First-Order Logic	21
2.2.1 Syntax	21

2.2.2	Calculus	25
2.2.3	Semantics	29
2.3	Extended First-Order Logic	33
2.3.1	Variable Binders	34
2.3.2	Undefinedness	35
2.4	First-Order Logic for Java	35
2.4.1	Type Hierarchy and Signature	36
2.4.2	Axioms for Integers	37
2.4.3	Axioms for Heap	38
2.4.4	Axioms for Location Sets	41
2.4.5	Semantics	42
3	Dynamic Logic for Java	47
3.1	Introduction	47
3.2	Syntax of JavaDL	48
3.2.1	Type Hierarchies	49
3.2.2	Signatures	49
3.2.3	Syntax of JavaDL Program Fragments	50
3.2.4	Syntax of JavaDL Terms and Formulas	52
3.3	Semantics	53
3.3.1	Kripke Structures	53
3.3.2	Semantics of JavaDL Terms and Formulas	54
3.4	Describing Transitions between States: Updates	55
3.4.1	Syntax and Semantics of JavaDL Updates	55
3.4.2	Update Simplification Rules	57
3.5	The Calculus for JavaDL	58
3.5.1	JavaDL Rule Schemata and First-Order Rules	58
3.5.2	Nonprogram Rules for Modalities	61
3.5.3	Soundness and Completeness of the Calculus	61
3.5.4	Schema Variables for Program Constructs	64
3.5.5	The Active Statement in a Modality	65
3.5.6	The Essence of Symbolic Execution	65
3.5.7	Components of the Calculus	66
3.6	Rules for Symbolic Execution of Java Programs	67
3.6.1	The Basic Assignment Rule	67
3.6.2	Rules for Handling General Assignments	68
3.6.3	Rules for Conditionals	73
3.6.4	Unwinding Loops	74
3.6.5	Replacing Method Calls by their Implementation	75
3.6.6	Instance Creation and Initialization	85
3.6.7	Handling Abrupt Termination	91
3.7	Abstraction and Modularization Rules	95
3.7.1	Replacing Method Calls by Specifications: Contracts	96
3.7.2	Reasoning about Unbounded Loops: Loop Invariants	99

4	Proof Search with Taclets	105
4.1	Introduction	105
4.1.1	Purpose and Organization of this Chapter	106
4.2	A Taclet Tutorial	107
4.2.1	A Basic Theory of Lists	107
4.2.2	The List Theory in Concrete Syntax	109
4.2.3	Definitional Extension of the List Theory	112
4.2.4	Derivation of Lemmas	115
4.3	A Reference Manual of Taclets	119
4.3.1	The Taclet Language	119
4.3.2	Schema Variables	128
4.3.3	Application of Taclets	134
4.4	Reflection and Reasoning about Soundness of Taclets	136
4.4.1	Soundness in Sequent Calculi	137
4.4.2	Meaning Formulas of Sequent Taclets	138
4.4.3	Meaning Formulas for Rewriting Taclets	140
4.4.4	Elimination of Schema Variables	141
5	Theories	147
5.1	Introduction	147
5.2	Finite Sequences	147
5.3	Strings	154
5.3.1	Sequences of Characters	154
5.3.2	Regular Expressions for Sequences	156
5.3.3	Relating Java String Objects to Sequences of Characters	157
5.3.4	String Literals and the String Pool	158
5.3.5	Specification of the Java String API	159
5.4	Integers	159
5.4.1	Core Integer Theory	160
5.4.2	Variable Binding Integer Operations	161
5.4.3	Symbolic Execution of Integer Expressions in Programs	162
6	Abstract Interpretation	165
6.1	Introduction	165
6.2	Integrating Abstract Interpretation	166
6.2.1	Abstract Domains	166
6.2.2	Abstractions for Integers	168
6.2.3	Abstracting States	168
6.3	Loop Invariant Generation	169
6.4	Abstract Domains for Heaps	172
6.5	Abstract Domains for Objects	176
6.5.1	Null/Not-null Abstract Domain	177
6.5.2	Length Abstract Domain	177
6.6	Extensions	182
6.6.1	Abstractions for Arrays	182

6.6.2	Loop Invariant Rule with Value and Array Abstraction . . .	182
6.6.3	Computation of the Abstract Update and Invariants	184
6.6.4	Symbolic Pivots	185
6.7	Conclusions	187
7	Formal Specification with the Java Modeling Language	189
7.1	Introduction to Method Contracts	192
7.1.1	Clauses of a Contract	192
7.1.2	Defensive Versus Offensive Method Implementations . . .	195
7.1.3	Specifications and Implementations	196
7.2	Expressions	197
7.2.1	Quantified Boolean Expressions	197
7.2.2	Numerical Comprehensions	199
7.2.3	Evaluation in the Prestate	200
7.3	Method Contracts in Detail	201
7.3.1	Visibility of Specifications	201
7.3.2	Specification Cases	202
7.3.3	Semantics of Normal Behavior Specification Cases	204
7.3.4	Specifications for Constructors	205
7.3.5	Notions of Purity	205
7.4	Class Level Specifications	206
7.4.1	Invariants	207
7.4.2	Initially Clauses	212
7.4.3	History Constraints	213
7.4.4	Initially Clauses and History Constraints: Static vs. Instance	214
7.4.5	Inheritance of Specifications	214
7.5	Nonnull Versus Nullable Object References	216
7.6	Exceptional Behavior	218
7.7	Specification-Only Class Members	221
7.7.1	Model Fields and Model Methods	221
7.7.2	Ghost Variables	225
7.7.3	Ghost Variables Versus Model Fields	225
7.8	Integer Semantics	226
7.9	Auxiliary Specification for Verification	229
7.9.1	Framing	229
7.9.2	Loop Invariants	230
7.9.3	Assertions and Block Contracts	234
7.10	Conclusion	235
7.10.1	Tool Support for JML	235
7.10.2	Comparison to Other Program Annotation Languages . . .	236
8	From Specification to Proof Obligations	239
8.1	Formal Semantics of JML Expressions	240
8.1.1	Types in JML	241
8.1.2	Translating JML Expressions to JavaDL	242

8.1.3	Abstract Data Types in JML	248
8.1.4	Well-Definedness of Expressions	249
8.2	From JML Contract Annotations to JavaDL Contracts	250
8.2.1	Normalizing JML Contracts	251
8.2.2	Constructor Contracts	260
8.2.3	Model Methods and Model Fields	261
8.2.4	JavaDL Contracts	264
8.2.5	Loop Specifications	267
8.3	Proof Obligations for JavaDL Contracts	268
8.3.1	Proof Obligations for Functional Correctness	268
8.3.2	Dependency Proof Obligations	274
8.3.3	Well-Definedness Proof Obligations	275
9	Modular Specification and Verification	285
9.1	Modular Verification with Contracts	287
9.1.1	Historical and Conceptual Background	287
9.1.2	Example: Implementing a List	290
9.1.3	Modular Program Correctness	292
9.1.4	Verification of Recursive Methods	294
9.2	Abstract Specification	296
9.2.1	Model Fields	298
9.2.2	Model Methods	307
9.3	The Frame Problem	315
9.3.1	Motivation	316
9.3.2	Dynamic Frames	318
9.3.3	Proof Obligations for Dynamic Frames Specifications	320
9.3.4	Example Specification with Dynamic Frames	321
9.4	Calculus Rules for Modular Reasoning	324
9.4.1	Anonymizing Updates	325
9.4.2	An Improved Loop Invariant Rule	326
9.4.3	A Rule for Method Contracts	332
9.4.4	A Rule for Dependency Contracts	335
9.4.5	Rules for Class Invariants	337
9.5	Verifying the List Example	339
9.6	Related Methodologies for Modular Verification	343
9.7	Conclusion	347
10	Verifying Java Card Programs	349
10.1	Introduction	349
10.2	Java Card Technology	350
10.3	Java Card Transactions on Explicit Heaps	353
10.3.1	Basic Transaction Roll-Back	354
10.3.2	Transaction Marking and Balancing	355
10.3.3	Object Creation and Deletion	356
10.3.4	Persistent and Transient Arrays	357

10.3.5	Nonatomic Updates	359
10.4	Taclets for the New Rules	360
10.5	Modular Reasoning with Multiple Heaps	363
10.6	Java Card Verification Samples	364
10.6.1	Conditional Balance Updating	365
10.6.2	Reference Implementation of a Library Method	366
10.6.3	Transaction Resistant PIN Try Counter	370
10.7	Summary and Discussion	372
10.7.1	Related Work	373
10.7.2	On-going and Future Research	373
10.7.3	Multiple Heaps for Concurrent Reasoning	373
11	Debugging and Visualization	377
11.1	Introduction	377
11.2	Symbolic Execution	379
11.3	Symbolic Execution Debugger	384
11.3.1	Installation	384
11.3.2	Basic Usage	385
11.3.3	Debugging with Symbolic Execution Trees	389
11.3.4	Debugging with Memory Layouts	391
11.3.5	Help Program and Specification Understanding	392
11.3.6	Debugging Meets Verification	393
11.3.7	Architecture	395
11.4	A Symbolic Execution Engine based on KeY	397
11.4.1	Symbolic Execution Tree Generation	397
11.4.2	Branch and Path Conditions	401
11.4.3	Hiding the Execution of Query Methods	402
11.4.4	Symbolic Call Stack	403
11.4.5	Method Return Values	403
11.4.6	Current State	404
11.4.7	Controlled Execution	405
11.4.8	Memory Layouts	405
11.5	Conclusion And Future Work	406
12	Proof-based Test Case Generation	409
12.1	Introduction	409
12.2	A Quick Tutorial	410
12.2.1	Setup	411
12.2.2	Usage	411
12.2.3	Options and Settings	413
12.3	Test Cases, Test Suites, and Test Criteria	415
12.4	Application Scenarios and Variations of the Test Generator	420
12.4.1	KeYTestGen for Test Case Generation	420
12.4.2	KeYTestGen for Formal Verification	421
12.5	Architecture of KeYTestGen	422

12.6	Proof-based Constraint Construction for Test Input Data	424
12.6.1	Symbolic Execution for Test Constraint Generation	424
12.6.2	Implicit Case Distinctions	425
12.6.3	Infeasible Path Filtering	426
12.6.4	Using Loop Unwinding and Method Inlining	427
12.6.5	Using Loop Invariants and Method Contracts	428
12.7	From Constraints to Test Input Data	431
12.7.1	The Type System	433
12.7.2	Preserving the Semantics of Interpreted Functions	434
12.7.3	Preventing Integer Overflows	435
12.7.4	Model Extraction	436
12.8	Specification-based Test Oracle Generation	436
12.8.1	Generating a Test Oracle from the Postcondition	436
12.8.2	Using a Runtime Assertion Checker	439
12.9	Synthesizing Executable Test Cases	439
12.10	Perspectives and Related Work	442
12.11	Summary and Conclusion	444
13	Information Flow Analysis	447
13.1	Introduction	447
13.2	Specification and the Attacker Model	449
13.3	Formal Definition of Secure Information Flow	451
13.4	Specifying Information Flow in JML	452
13.5	Information Flow Verification with KeY	454
13.5.1	Efficient Double Symbolic Execution	455
13.5.2	Using Efficient Double Symbolic Execution in KeY	463
13.6	Summary and Conclusion	464
14	Program Transformation and Compilation	467
14.1	Interleaving Symbolic Execution and Partial Evaluation	468
14.1.1	General Idea	468
14.1.2	The Program Specialization Operator	472
14.1.3	Specific Specialization Actions	473
14.1.4	Example	475
14.2	Verified Correct Compilation	476
14.3	Implementation and Evaluation	485
14.4	Conclusion	486
15	Using the KeY Prover	487
15.1	Introduction	487
15.2	Exploring KeY Artifacts and Prover Simultaneously	490
15.2.1	Exploring Basic Notions And Usage	491
15.2.2	Exploring Terms, Quantification, and Instantiation	503
15.2.3	Exploring Programs in Formulas	507
15.3	Understanding Proof Situations	524

15.4	Further Features	529
15.5	What Next?	531
16	Formal Verification with KeY: A Tutorial	533
16.1	Introduction	533
16.2	A Program without Loops	535
16.3	A Brief Primer on Loop Invariants	538
16.3.1	Introduction	538
16.3.2	Why Are Loop Invariants Needed?	539
16.3.3	What Is A Loop Invariant?	539
16.3.4	Goal-Oriented Derivation of Loop Invariants	541
16.3.5	Generalization	542
16.3.6	Recovering the Context	543
16.3.7	Proving Termination	545
16.3.8	A More Complex Example	547
16.3.9	Invariants: Concluding Remarks	549
16.4	A Program with Loops	550
16.5	Data Type Properties of Programs	554
16.6	KeY and Eclipse	557
16.6.1	Installation	558
16.6.2	Proof Management with KeY Projects	558
16.6.3	Proof Results via Marker	559
16.6.4	The Overall Verification Status	560
17	KeY-Hoare	563
17.1	Introduction	563
17.2	The Programming Language	564
17.3	Background	565
17.3.1	First-Order Logic	565
17.3.2	Hoare Calculus	566
17.4	Hoare Logic with Updates	567
17.4.1	State Updates	568
17.4.2	Hoare Triples with Update	569
17.4.3	Hoare Style Calculus with Updates	569
17.4.4	Rules for Updates	571
17.5	Using KeY-Hoare	572
17.6	Variants of the Hoare Logic with Updates	574
17.6.1	Total Correctness	574
17.6.2	Worst-Case Execution Time	574
17.7	A Brief Reference Manual for KeY-Hoare	576
17.7.1	Installation	576
17.7.2	Formula Syntax	577
17.7.3	Input File Format	577
17.7.4	Loading and Saving Problems and Proofs	579
17.7.5	Proving	580

17.7.6	Automation	581
18	Verification of an Electronic Voting System	583
18.1	Electronic Voting	583
18.2	Overview	584
18.2.1	Verification of Cryptographic Software	585
18.2.2	Verification Approach	585
18.2.3	System Description	586
18.3	Specification and Verification	589
18.3.1	Specification	589
18.3.2	Verification	593
18.4	Discussion	595
18.4.1	A Hybrid Approach to Information Flow Analysis	595
18.4.2	Related Work	596
18.4.3	Conclusion	597
19	Verification of Counting Sort and Radix Sort	599
19.1	Counting Sort and Radix Sort Implementation	599
19.2	High-Level Correctness Proof	602
19.2.1	General Auxiliary Functions	603
19.2.2	Counting Sort Proof	604
19.2.3	Radix Sort Proof	605
19.3	Experience Report	607
20	Java Modeling Language Reference	609
20.1	JML Syntax	609
20.2	JML Expression Semantics	613
20.3	JML Expression Well-Definedness	616
21	KeY File Reference	619
21.1	Predefined Operators in JavaDL	619
21.1.1	Arithmetic Functions	619
21.1.2	Arithmetic Functions with Modulo Semantics	620
21.1.3	Predicates for Arithmetics and Equality	622
21.1.4	Integer Semantics Dependent Functions	622
21.1.5	Integer Semantics Dependent Predicate Symbols	624
21.1.6	Heap Related Function and Predicate Symbols	624
21.1.7	Location Sets Related Function and Predicate Symbols	625
21.1.8	Finite Sequence Related Function and Predicate Symbols	626
21.1.9	Map Related Function and Predicate Symbols	627
21.2	The KeY Syntax	628
21.2.1	Notation, Keywords, Identifiers, Numbers, Strings	629
21.2.2	Terms and Formulas	631
21.2.3	Rule Files	639
21.2.4	User Problem and Proof Files	647
21.2.5	Schematic Java Syntax	650

xxiv

References

Index

Contents

655

679