1. Record Nr. UNISA996565869703316

Autore	Herber Paula
Titolo	iFM 2023 [[electronic resource]] : 18th International Conference, iFM 2023, Leiden, The Netherlands, November 13–15, 2023, Proceedings / / edited by Paula Herber, Anton Wijs
Pubbl/distr/stampa	Cham : , : Springer Nature Switzerland : , : Imprint : Springer, , 2024
ISBN	3-031-47705-7
Edizione	[1st ed. 2024.]
Descrizione fisica	1 online resource (405 pages)
Collana	Lecture Notes in Computer Science, , 1611-3349 ; ; 14300
Altri autori (Persone)	WijsAnton
Disciplina	005.1
Soggetti	Software engineering Software Engineering
Lingua di pubblicazione	Inglese
Formato	Materiale a stampa
Livello bibliografico	Monografia
Nota di contenuto	Intro -- Preface -- Organization -- Abstract of Invited Talks -- Formal Signoff Flows -- Industrial Experience with a Verification-Aware Programming Language -- Contents -- Invited Presentations -- SMT: Something You Must Try -- 1 Introduction -- 2 Satisfiability Checking -- 2.1 SAT Solving -- 2.2 SMT Solving -- 3 Applications -- 3.1 A Toy Encoding Example -- 3.2 Planning with Optimization Modulo Theories -- 3.3 Reachability Analysis for Hybrid Systems with HyPro -- 3.4 Parameter Synthesis for Probabilistic Systems -- 4 Hybrid Petri Nets and Rate Adaption -- 4.1 Hybrid Petri Nets -- 4.2 Rate Adaption -- 4.3 Formulation as an SMT Problem -- 5 Some Final Remarks -- References -- Analysis and Verification -- Automated Sensitivity Analysis for Probabilistic Loops -- 1 Introduction -- 2 Preliminaries -- 2.1 Syntax and Semantics of Probabilistic Loops -- 2.2 C-finite Recurrences -- 2.3 Higher Moment Analysis Using Recurrences -- 3 Sensitivity Analysis -- 3.1 Sensitivity Analysis for Admissible Loops -- 3.2 Sensitivity Analysis for Non-admissible Loops -- 4 Experiments and Evaluation -- 5 Related Work -- 6 Conclusion -- References -- diffDP: Using Data Dependencies and Properties in Difference Verification with Conditions -- 1 Introduction -- 2 Foundations of Difference Verification with Conditions -- 3 Dependency and Property Aware Difference Detection -- 4 Evaluation -- 4.1 Experimental Setup -- 4.2 Experimental Results -- 5 Related Work -- 6 Conclusion -- References -- CHC Model

Validation with Proof Guarantees -- 1 Introduction -- 2 Background -- 2.1 Constrained Horn Clauses -- 2.2 Related Witness Validation Approaches -- 3 Validation of CHC Models -- 4 Implementation -- 5 Evaluation -- 5.1 Benchmarks and Tools -- 5.2 Model Validation Results -- 5.3 Proof Checking Results -- 6 Conclusions -- References.
VerifyThis: Memcached-A Practical Long-Term Challenge for the Integration of Formal Methods -- 1 Introduction -- 2 System Description -- 3 Challenge Goals -- 4 Participation, Time Schedule, Conclusion -- References -- Deductive Verification -- Towards Formal Verification of a TPM Software Stack -- 1 Introduction -- 2 Frama-C Verification Platform -- 3 The TPM Software Stack and the tpm2-tss Library -- 4 Dynamic Memory Allocation -- 5 Memory Management -- 6 Lemmas -- 7 Verification Results -- 8 Related Work -- 9 Conclusion and Future Work -- References -- Reasoning About Exceptional Behavior at the Level of Java Bytecode -- 1 Introduction -- 2 Motivating Example -- 3 Specifying and Verifying Exceptional Behavior -- 3.1 Specifying Exceptional Behavior -- 3.2 The Vimp Intermediate Representation -- 3.3 Modeling Exceptional Control Flow -- 3.4 Transformation Order and Boogie Code Generation -- 3.5 Implementation Details -- 4 Experiments -- 4.1 Programs -- 4.2 Results -- 5 Related Work -- 6 Conclusions -- References -- Analysis and Formal Specification of OpenJDK's BitSet -- 1 Introduction -- 2 The BitSet Class -- 3 Formal Specification -- 3.1 Class Invariant -- 3.2 The wordsToSeq() Model Method -- 3.3 The get(int,int) Method -- 4 Issues in BitSet -- 4.1 A Bug in get(int,int) Caused by a Negative length() -- 4.2 Bugs Caused by valueOf(...) Corrupting length() -- 4.3 Solution Directions -- 5 Towards Formal Verification of the BitSet Class -- 5.1 Background -- 5.2 Proof Sketch of get(int,int) -- 5.3 Required Extensions to KeY -- 6 Conclusion -- References -- Joining Forces! Reusing Contracts for Deductive Verifiers Through Automatic Translation -- 1 Introduction -- 2 Design of the Specification Translator -- 3 Translating Annotations -- 3.1 OpenJML -- 3.2 Krakatoa -- 3.3 VerCors -- 4 Evaluation -- 4.1 Reuse of Specifications -- 4.2 Reuse of Tools.
5 Related Work -- 6 Conclusion -- References -- Hardware and Memory Verification -- Lifting the Reasoning Level in Generic Weak Memory Verification -- 1 Introduction -- 2 Program Syntax -- 3 Axiomatic Reasoning -- 3.1 Axioms -- 3.2 Reasoning Example on Axiom Level -- 4 Rules -- 5 Correctness Proof of WRC via Proof Rules -- 6 Related Work -- 7 Conclusion -- References -- Automatic Formal Verification of RISC-V Pipelined Microprocessors with Fault Tolerance by Spatial Redundancy at a High Level of Abstraction -- 1 Introduction -- 2 Background on Using Positive Equality to Formally Verify Pipelined Processors -- 3 Efficient Modeling of the RISC-V Architecture at a High Level of Abstraction -- 3.1 Abstraction of the Register File -- 3.2 Abstraction of the Decoding Logic -- 3.3 Abstraction of the CSR Memory Array -- 3.4 Non-Pipelined Specifications -- 3.5 Formal Verification of Liveness -- 3.6 Invariant Constraints Were Not Needed for the Correct Designs, but Were Proved for Debugging the 4-Stage Pipelined Processor -- 4 Modeling of Spatial Redundancy in Pipelined Processors at a High Level of Abstraction for Formal Verification of Safety -- 5 Results -- 6 Conclusion -- References -- Refinement and Separation: Modular Verification of Wandering Trees -- 1 Introduction -- 2 Wandering Trees -- 3 Structured Specifications of Algebraic Data Types -- 3.1 Algebraic Definitions -- 3.2 Modeling the Heap and Separation Logic -- 4 Modular Software Systems -- 5 Modularization -- 5.1 Specification of the Index -- 5.2 Index Refinement -- 5.3 Tree Refinement -- 5.4 Pointer Structures in the

Heap and on Flash -- 6 Verification -- 6.1 Correctness of the Tree Component -- 6.2 Correctness of Wandering Trees -- 6.3 Correctness of Flash Representation -- 7 Conclusion -- References -- Verification and Learning.
Performance Fuzzing with Reinforcement-Learning and Well-Defined Constraints for the B Method -- 1 Introduction -- 2 Background -- 2.1 ProB and the B Method -- 2.2 Performance Fuzzing -- 2.3 Thompson Sampling -- 3 The BanditFuzz Algorithm -- 4 Adapting BanditFuzz for ProB -- 4.1 Fuzzing for Classical B Instead of SMT-LIB -- 4.2 Targeted Fuzzing and Mutating -- 4.3 Well-Defined ASTs -- 5 Running the Performance Fuzzer -- 5.1 Fuzzing only -- 5.2 Performance Fuzzing Between ProB's Backends -- 5.3 Performance Fuzzing Between ProB's Settings -- 5.4 Performance Fuzzing Between the SMT-LIB Translations -- 6 Related Work -- 7 Discussion -- 8 Conclusions -- References -- Reinforcement Learning Under Partial Observability Guided by Learned Environment Models -- 1 Introduction -- 2 Preliminaries -- 2.1 Models -- 2.2 Learning MDPs -- 3 QA-Learning: RL Assisted by Automata Learning -- 3.1 Overview -- 3.2 Extended State Space -- 3.3 Partially Observable Q-Learning -- 3.4 Convergence -- 3.5 Generalization and Limitations -- 4 Evaluation -- 5 Related Work -- 6 Conclusion -- References -- Temporal Logics -- Mission-Time LTL (MLTL) Formula Validation via Regular Expressions -- 1 Introduction -- 2 Preliminaries: Mission-Time LTL and Bit String Computations -- 3 MLTL Regular Expressions -- 4 WEST Algorithm and Analysis -- 4.1 Proof of Correctness of WEST -- 4.2 Theoretical Complexity -- 4.3 Experimental Benchmarking -- 5 Correctness of WEST Tool Implementation -- 5.1 Intelligent Fuzzing -- 6 Regular Expression Simplification Theorem (REST) -- 6.1 Theoretical Analysis of REST -- 6.2 Experimental Benchmarking of REST -- 7 Using WEST: An Example -- 8 Closing Remarks -- 8.1 Future Work -- References -- Symbolic Model Checking of Relative Safety LTL Properties -- 1 Introduction -- 2 Motivating Examples -- 2.1 Bounded Response Example -- 2.2 Safety Contracts.
2.3 LTL G p vs. Invariant -- 3 Related Work -- 4 Notation and Preliminary Definitions -- 4.1 Notation for Sequences, Concatenation and Prefix -- 4.2 Safety and Relative Safety -- 4.3 LTL and Safety Fragments -- 4.4 Symbolic Transition System and Invariant Checking -- 4.5 Symbolic Techniques Reducing LTL Model Checking to Invariant Checking -- 5 Symbolic Compilation of Safety LTL -- 5.1 Symbolic Compilation of Full LTL -- 5.2 Symbolic Compilation of SafetyLTL -- 5.3 Symbolic Construction of Live Systems -- 6 Algorithm -- 6.1 Simple Algorithm Without Loops -- 6.2 CEGAR Loop Algorithm -- 6.3 Extending Algorithm with Lookahead -- 7 Experimental Evaluation -- 7.1 Benchmarks -- 7.2 Experimental Results and Analysis -- 8 Conclusions and Future Work -- References -- Extending PlusCal for Modeling Distributed Algorithms -- 1 Introduction -- 2 TLA+ and PlusCal -- 2.1 The Specification Language TLA+ -- 2.2 The Algorithmic Langage PlusCal -- 2.3 Translating PlusCal to TLA+ -- 3 Distributed PlusCal -- 3.1 Sub-processes -- 3.2 Communication Channels -- 4 Evaluation -- 4.1 Two Distributed Algorithms Expressed in Distributed PlusCal -- 4.2 Related Work -- 5 Conclusion -- References -- Autonomous Systems -- Formal Modelling and Analysis of a Self-Adaptive Robotic System -- 1 Introduction -- 2 Case Study: Pipeline Inspection by AUV -- 3 Modelling the AUV Case Study with ProFeat -- 3.1 The Feature Model -- 3.2 The Managed Subsystem -- 3.3 The Environment -- 3.4 The Managing Subsystem -- 4 Analysis -- 5 Related Work -- 6 Discussion and Future Work -- References -- CAN-verify: A Verification Tool For BDI Agents -- 1 Introduction -- 2 Can-

	Overview -- 3 CAN-verify Components and Features -- 4 Examples -- 5 Discussion and Conclusion -- References -- PhD Symposium Presentations -- Scalable and Precise Refinement Types for Imperative Languages -- 1 Introduction. 2 Combining Refinement Type Systems and Deductive Verification.
Sommario/riassunto	This volume LNCS 14300 constitutes the refereed proceedings of the 18th International Conference, IFM 2023, in November 2023, held in Leiden, The Netherlands. The 16 full papers presented together with 2 short papers were carefully reviewed and selected from 51 submissions. The conference focuses on all aspects of the design of integrated techniques, including language design, verification and validation, automated tool support, and the use of such techniques in software engineering practice.