1. Record Nr. UNISA996418211703316

Autore	Müller Peter
Titolo	Programming Languages and Systems [[electronic resource]] : 29th European Symposium on Programming, ESOP 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25–30, 2020, Proceedings / / edited by Peter Müller
Pubbl/distr/stampa	Cham : , : Springer International Publishing : , : Imprint : Springer, , 2020
ISBN	3-030-44914-9
Edizione	[1st ed. 2020.]
Descrizione fisica	1 online resource (XV, 772 p. 1 illus.)
Collana	Theoretical Computer Science and General Issues, , 2512-2029 ; ; 12075
Disciplina	005.13
Soggetti	Compilers (Computer programs) Computer engineering Computer networks Microprogramming Natural language processing (Computer science) Computer science Compilers and Interpreters Computer Engineering and Networks Control Structures and Microprogramming Computer Communication Networks Natural Language Processing (NLP) Theory of Computation
Lingua di pubblicazione	Inglese
Formato	Materiale a stampa
Livello bibliografico	Monografia
Nota di contenuto	Intro -- ETAPS Foreword -- Preface -- Organization -- Formal Methods for Evolving Database Applications (Abstract of Keynote Talk) -- Contents -- Trace-Relating Compiler Correctness and Secure Compilation -- Introduction -- Trace-Relating Compiler Correctness -- Property Mappings -- Trace Relations and Property Mappings -- Preservation of Subset-Closed Hyperproperties -- Instances of Trace-Relating Compiler Correctness -- Undefined Behavior -- Resource

Exhaustion -- Different Source and Target Values -- Abstraction Mismatches -- Trace-Relating Compilation and Noninterference Preservation -- Trace-Relating Secure Compilation -- Trace-Relating Secure Compilation: A Spectrum of Trinities -- Instance of Trace-Relating Robust Preservation of Trace Properties -- Instances of Trace-Relating Robust Preservation of Safety and Hypersafety -- Related Work -- Conclusion and Future Work -- Acknowledgements -- Bibliography -- Runners in action -- 1 Introduction -- 2 Algebraic effects, handlers, and runners -- 2.1 Algebraic effects and handlers -- 2.2 Runners -- 3 Programming with runners -- 3.1 The user and kernel monads -- 3.2 Runners as a programming construct -- 4 A calculus for programming with runners -- 4.1 Types -- 4.2 Values and computations -- 4.3 Type system -- 4.4 Equational theory -- 5 Denotational semantics -- 5.1 Semantics of types -- 5.2 Semantics of values and computations -- 5.3 Coherence, soundness, and finalisation theorems -- 6 Runners in action -- 7 Implementation -- 8 Related work -- 9 Conclusion and future work -- References -- On the Versatility of Open Logical Relations -- 1 Introduction -- 2 The Playground -- 3 A Fundamental Gap -- 4 Warming Up: A Containment Theorem -- 5 Automatic Differentiation -- 6 On Refinement Types and Local Continuity -- 6.1 A Refinement Type System Ensuring Local Continuity -- 6.2 Basic Typing Rules.
6.3 Typing Conditionals -- 6.4 Open-logical Predicates for Refinement Types -- 7 Related Work -- 8 Conclusion and Future Work -- References -- Constructive Game Logic -- 1 Introduction -- 2 Related Work -- 3 Syntax -- 3.1 Example Games -- 4 Semantics -- 4.1 Realizers -- 4.2 Formula and Game Semantics -- 4.3 Demonic Semantics -- 5 Proof Calculus -- 6 Theory: Soundness -- 7 Operational Semantics -- 8 Theory: Constructivity -- 9 Conclusion and Future Work -- References -- Optimal and Perfectly Parallel Algorithms for On-demand Data-flow Analysis -- 1 Introduction -- 2 Preliminaries -- 2.1 The IFDS Framework -- 2.2 Trees and Tree Decompositions -- 3 Problem definition -- 4 Treewidth-based Data-ow Analysis -- 4.1 Preprocessing -- 4.2 Word Tricks -- 4.3 Answering Queries -- 4.4 Parallelizability and Optimality -- 5 Experimental Results -- 6 Conclusion -- References -- Concise Read-Only Specifications for Better Synthesis of Programs with Pointers -- 1 Introduction -- 1.1 Correct Programs that Do Strange Things -- 1.2 Towards Simple Read-Only Specifications for Synthesis -- 1.3 Our Contributions -- 2 Program Synthesis with Read-Only Borrows -- 2.1 Basics of SSL-based Deductive Program Synthesis -- 2.2 Reducing Non-Determinism with Read-Only Annotations -- 2.3 Composing Read-Only Borrows -- 2.4 Borrow-Polymorphic Inductive Predicates -- 3 BoSSL: Borrowing Synthetic Separation Logic -- 3.1 BoSSL rules -- 3.2 Memory Model -- 3.3 Soundness -- 4 Implementation and Evaluation -- 4.1 Experimental Setup -- 4.2 Performance and Quality of the Borrowing-Aware Synthesis -- 4.3 Stronger Correctness Guarantees -- 4.4 Robustness under Synthesis Perturbations -- 5 Limitations and Discussion -- 6 Related Work -- 7 Conclusion -- References -- Soundness conditions for big-step semantics -- 1 Introduction -- 2 A meta-theory for big-step semantics -- 3 Extended semantics.
3.1 Traces -- 3.2 Wrong -- 4 Expressing and proving soundness -- 4.1 Expressing soundness -- 4.2 Conditions ensuring soundness-must -- 4.3 Conditions ensuring soundness-may -- 5 Examples -- 5.1 Simply-typed -calculus with recursive types -- 5.2 MiniFJ&amp -- - -- 5.3 Intersection and union types -- 5.4 MiniFJ&amp -- O -- 6 The partial evaluation construction -- 7 Related work -- 8 Conclusion and future work -- Acknowledgments -- References -- Liberate Abstract Garbage

Collection from the Stack by Decomposing the Heap -- 1 Introduction -- 1.1 Examples -- 1.2 Generalizing the Approach -- 2 A-Normal Form - Calculus -- 3 Background -- 3.1 Semantic Domains -- 3.2 Concrete Semantics -- 3.3 Abstracting Abstract Machines with Garbage Collection -- 3.4 Stack-Precise CFA with Garbage Collection -- 3.5 The k-CFA Context Abstraction -- 4 From Threaded to Compositional Stores -- 4.1 Threaded-Store Semantics -- 4.2 Threaded-Store Semantics with Effect Log -- 4.3 Compositional-Store Semantics -- 4.4 Compositional-Store Semantics with Garbage Collection -- 5 Abstract Compositional-Store Semantics with Garbage Collection -- 6 Discussion -- 6.1 The Effects of Treating the Store Compositionally -- 6.2 The Effect of Treating the Time Compositionally -- 7 Related Work -- 8 Conclusion and Future Work -- References -- SMT-Friendly Formalization of the Solidity Memory Model -- 1 Introduction -- 2 Background -- 2.1 Ethereum -- 2.2 Solidity -- 2.3 SMT-Based Programs -- 3 Formalization -- 3.1 Types -- 3.2 Local Storage Pointers -- 3.3 Contracts, State Variables, Functions -- 3.4 Statements -- 3.5 Assignments -- 3.6 Expressions -- 4 Evaluation -- 5 Related Work -- 6 Conclusion -- References -- Exploring Type-Level Bisimilarity towards More Expressive Multiparty Session Types -- 1 Introduction -- 2 Overview of our Approach -- 3 An MPST Theory with +, , and ll.
3.1 Types as Process Algebraic Terms -- 3.2 Global Types and Local Types -- 3.3 End-Point Projection: from Global Types to Local Types -- 3.4 Weak Bisimilarity of Global Types, Local Types, and Groups -- 3.5 Well-formedness of Global Types -- 3.6 Correctness of Projection under Well-Formedness -- 3.7 Decidability of Checking Well-Formedness -- 3.8 Discussion of Challenges -- 4 Practical Experience with the Theory -- 4.1 Implementation -- 4.2 Evaluation of the Approach -- 5 Related Work -- 6 Conclusion -- References -- Verifying Visibility-Based Weak Consistency -- 1 Introduction -- 2 Weak Consistency -- 2.1 Weak-Visibility Specifications -- 2.2 Consistency against Weak-Visibility Specifications -- 3 Establishing Consistency with Forward Simulation -- 3.1 Reducing Consistency to Safety Verification -- 3.2 Verifying Implementations -- 4 Proof Methodology -- 5 Implementation and Evaluation -- 6 Related Work -- 7 Conclusion and Future Work -- A Appendix: Proofs to Theorems and Lemmas -- References -- Local Reasoning for Global Graph Properties -- 1 Introduction -- 2 The Foundational Flow Framework -- 2.1 Preliminaries and Notation -- 2.2 Flows -- 2.3 Flow Graph Composition and Abstraction -- 3 Proof Technique -- 3.1 Encoding Flow-based Proofs in SL -- 3.2 Proof of the PIP -- 4 Advanced Flow Reasoning and the Harris List -- 4.1 The Harris List Algorithm -- 4.2 Product Flows for Reasoning about Overlays -- 4.3 Contextual Extensions and the Replacement Theorem -- 4.4 Existence and Uniqueness of Flows -- 4.5 Proof of the Harris List -- 5 Related Work -- 6 Conclusions and Future Work -- References -- Aneris: A Mechanised Logic for Modular Reasoning about Distributed Systems -- 1 Introduction -- 2 The Core Concepts of Aneris -- 2.1 Local and Thread-Local Reasoning -- 2.2 Node-Local Reasoning -- 2.3 Example: An Addition Service -- 2.4 Example: A Lock Server.
3 AnerisLang -- 4 The Aneris Logic -- 4.1 The Program Logic -- 4.2 Adequacy for Aneris -- 5 Case Study 1: A Load Balancer -- 6 Case Study 2: Two-Phase Commit -- 6.1 A Replicated Log -- 7 Related Work -- 8 Conclusion -- Acknowledgments -- Bibliography -- Continualization of Probabilistic Programs With Correction -- 1 Introduction -- 2 Example -- 2.1 Continualization -- 2.2 Parameter Synthesis -- 2.3 Improving Inference -- 3 Syntax and Semantics of Programs -- 3.1 Source Language Syntax -- 3.2 Semantics -- 4

Continualizing Probabilistic Programs -- 4.1 Overview of the Algorithm -- 4.2 Distribution and Expression Transformations -- 4.3 Inuence Analysis and Control-Flow Correction of Predicates -- 4.4 Bringing it all together: Full Program Transformations -- 5 Synthesis of Continuity Correction Parameters -- 5.1 Optimization Framework -- 5.2 Optimization Algorithm -- 6 Methodology -- 6.1 Benchmarks -- 6.2 Experimental Setup -- 7 Evaluation -- 7.1 RQ1: Benefits of Continualization -- 7.2 RQ2: Impact of Smoothing Factors -- 7.3 RQ3: Extending Results to Other Systems -- 8 Related Work -- 9 Conclusion -- References -- Semantic Foundations for Deterministic Dataflow and Stream Processing -- 1 Introduction -- 2 Monoids as Types for Streams -- 3 Stream Transductions -- 4 Model of Computation -- 5 Combinators for Deterministic Dataow -- 6 Algebraic Reasoning for Optimizing Transformations -- 7 Related Work -- 8 Conclusion -- References -- Connecting Higher-Order Separation Logic to a First-Order Outside World -- 1 Introduction -- 2 Background: Ghost State in Separation Logic -- 2.1 Ghost Algebras -- 3 External State as Ghost State -- 4 Verifying C Programs with I/O in VST -- 5 Soundness of External-State Reasoning -- 6 Connecting VST to CertiKOS -- 6.1 CertiKOS Specifications -- 6.2 Relating OS and User State -- 6.3 Soundness of VST + CertiKOS.
7 From syscall-level to hardware-level interactions.

Sommario/riassunto

This open access book constitutes the proceedings of the 29th European Symposium on Programming, ESOP 2020, which took place in Dublin, Ireland, in April 2020, and was held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020. The papers deal with fundamental issues in the specification, design, analysis, and implementation of programming languages and systems.
.