

## The Little Book Of Semaphores

*This book constitutes contributions of the ISoLA 2021 associated events. Altogether, ISoLA 2021 comprises contributions from the proceedings originally foreseen for ISoLA 2020 collected in 4 volumes, LNCS 12476: Verification Principles, LNCS 12477: Engineering Principles, LNCS 12478: Applications, and LNCS 12479: Tools and Trends. The contributions included in this volume were organized in the following topical sections: 6th International School on Tool-Based Rigorous Engineering of Software Systems; Industrial Track; Programming: What is Next; Software Verification Tools; Rigorous Engineering of Collective Adaptive Systems.*

*If you're a student studying computer science or a software developer preparing for technical interviews, this practical book will help you learn and review some of the most important ideas in software engineering—data structures and algorithms—in a way that's clearer, more concise, and more engaging than other materials. By emphasizing practical knowledge and skills over theory, author Allen Downey shows you how to use data structures to implement efficient algorithms, and then analyze and measure their performance. You'll explore the important classes in the Java collections framework (JCF), how they're implemented, and how they're expected to perform. Each chapter presents hands-on exercises supported by test code online. Use data structures such as lists and maps, and understand how they work Build an application that reads Wikipedia pages, parses the contents, and navigates the resulting data tree Analyze code to predict how fast it will run and how much memory it will require Write classes that implement the Map interface, using a hash table and binary search tree Build a simple web search engine with a crawler, an indexer that stores web page contents, and a retriever that returns user query results Other books by Allen Downey include Think Java, Think Python, Think Stats, and Think Bayes.*

*This updated and expanded second edition of Book provides a user-friendly introduction to the subject, Taking a clear structural framework, it guides the reader through the subject's core elements. A flowing writing style combines with the use of illustrations and diagrams throughout the text to ensure the reader understands even the most complex of concepts. This succinct and enlightening overview is a required reading for all those interested in the subject . We hope you find this book useful in shaping your future career & Business.*

*The Little Book of SemaphoresThe Little Book of SemaphoresVersion 2.2.1*

*Synchronization of Concurrent Processes: Communication - Cooperation - Competition*

*OPERATING SYSTEM*

*Introduction to Parallel Computing*

*Learn all about building fast, scalable, and high performing applications with Delphi*

*Think Python*

*Nonsequential and Distributed Programming with Go*

*Teaching Formal Methods*

Get to grips with modern software demands by learning the effective uses of Rust's powerful memory safety. Key Features Learn and improve the sequential performance characteristics of your software Understand the use of operating system processes in a high-scale concurrent system Learn of the various coordination methods available in Rust Standard library Book Description Most programming languages can really complicate things, especially with regard to unsafe memory access. The burden on you, the programmer, lies across two domains: understanding the modern machine and your language's pain-points. This book will teach you to how to manage program performance on modern hardware and build fast, memory-safe, and concurrent software in Rust. It starts with the fundamentals of Rust and discusses machine architecture concepts. You will be taken through the Rust ecosystem, measure and improve the performance of Rust code systematically and how to write collections with confidence. You will learn about the Sync and Send traits applied to coordinate thread execution with locks, atomic primitives, data-parallelism, and more. The book will show you how to efficiently embed Rust in C++ code and explore the functionalities of various crates for multithreaded applications. It explores implementations in depth. You will know how a mutex works and build several yourself. You will explore radically different approaches that exist in the ecosystem for structuring and managing high-scale systems. By the end of the book, you will feel comfortable with designing consistent, parallel, and high-performance applications in Rust. What you will learn Probe your programs for performance and accuracy issues Create your own threading and processing environment in Rust Use coarse locks from Rust's Standard library Solve common synchronization problems or avoid synchronization using atomic programming lock-free/wait-free structures in Rust and understand their implementations in the crates ecosystem Leverage Rust's memory model and type system to build safety parallelism in your parallel programs Understand the new features of the Rust programming language to ease the writing of parallel programs Who this book is for This book is aimed at Rust engineers with a basic understanding of Rust who want to exploit the parallel and concurrent nature of modern computing environments, safely.

The pioneering organizers of the first UML workshop in Mulhouse, France in the summer of 1998 could hardly have anticipated that, in little over a decade, their initiative would blossom into today's highly successful MODELS conference series, the premier annual gathering of researchers and practitioners focusing on a very important technical discipline: model-based software and system engineering. This expansion is, of course, a direct consequence of the growing significance and success of model-based software engineering practice. The conferences have contributed greatly to the heightened interest in the field, attracting much young talent and leading to the gradual emergence of its corresponding scientific and engineering foundations. The proceedings from the MODELS conferences are one of the primary references for anyone interested in a more

study of the domain. The 12th conference took place in Denver in the USA, October 4–9, 2009 along with numerous satellite workshops and tutorials, as well as several scientific gatherings. The conference was exceptionally fortunate to have three eminent, invited keynote speakers from industry: Stephen Mellor, Larry Constantine, and ... This book constitutes the thoroughly refereed post-conference proceedings of the 12th International Conference on Membrane Computing, CMC 2011, held in Fontainebleau in August 2011. The 19 revised selected papers presented were carefully reviewed and selected from 27 papers and 5 posters presented at the conference. The book contains 19 papers or extended abstracts of the 5 invited presentations. The papers address all the main directions of research in membrane computing, ranging from theoretical computer science and mathematics to application issues.

The Little Book of Semaphores is a free (in both senses of the word) textbook that introduces the principles of synchronization for concurrent programming. In most computer science curricula, synchronization is a module in an Operating Systems class. OS textbooks present a standard set of problems with a standard set of solutions, but most students don't get a good understanding of the material or the ability to solve similar problems. The approach of this book is to identify patterns that are useful for a variety of synchronization problems and then show how they can be assembled into solutions. After each problem, the book offers a hint before showing a solution, giving students a better chance of discovering solutions on their own. The book covers the classical problems, including "Readers-writers," "Producer-consumer" and "Dining Philosophers." In addition, it collects a number of not-so-classical problems, some written by the author and some by other teachers and textbook writers. Readers are invited to create and submit new problems.

Hands-On Concurrency with Rust

Collectors Set

Think Data Structures

Concurrent Programming: Algorithms, Principles, and Foundations

Version 2.2.1

Historical Signals and Semaphores

Build fast Delphi applications using concurrency, parallel programming and memory management

Inspired by actual WWII signal training cards, a double deck of playing cards and companion booklet highlight this handsome gift set. The 48-page illustrated booklet recounts the fascinating historical background of the signal flag, semaphore, phonetic alphabet, and Morse code systems featured in the decks. Packaged in a durable and attractive case, the set includes a recreated Morse code flasher device, a signal training indicator wheel, a full-color poster, and other items of historical interest.

The Little Book of Semaphores is a textbook that introduces the principles of synchronization for concurrent programming. In most computer science curricula, synchronization is a module in an Operating Systems class. OS textbooks present a standard set of problems with a standard set of solutions, but most students don't get a good understanding of the material or the ability to solve similar problems. The approach of this book is to identify patterns that are useful for a variety of synchronization problems and then show how they can be assembled into solutions. After each problem, the book offers a hint before showing a solution, giving students a better chance of discovering solutions on their own. The book covers the classical problems, including "Readers-writers," "Producer-consumer," and "Dining Philosophers." In addition, it collects a number of not-so-classical problems. \*\*\* Published under the terms of the GNU Free Documentation License. Money raised from the sale of this book supports the development of free software and documentation.

Use structural, behavioral, and concurrent patterns in Delphi to skillfully develop applications Key Features Delve into the core patterns and components of Delphi to enhance your application's design Learn how to select the right patterns to improve your program's efficiency and productivity Discover how parallel programming and memory management can optimize your code Book Description Delphi is a cross-platform Integrated Development Environment (IDE) that supports rapid application development for most operating systems, including Microsoft Windows, iOS, and now Linux with RAD Studio 10.2. If you know how to use the features of Delphi, you can easily create scalable applications in no time. This Learning Path begins by explaining how to find performance bottlenecks and apply the correct algorithm to fix them. You'll brush up on tricks, techniques, and best practices to solve common design and architectural challenges. Then, you'll see how to leverage external libraries to write better-performing programs. You'll also learn about the eight most important patterns that'll enable you to develop and improve the interface between items and harmonize shared memories within threads. As you progress, you'll also delve into improving the performance of your code and mastering cross-platform RTL improvements. By the end of this Learning Path, you'll be able to address common design problems and feel confident while building scalable projects. This Learning Path includes content from the following Packt products: Delphi High Performance by Primož Gabrijelčič Hands-On Design Patterns with Delphi by Primož Gabrijelčič What you will learn Understand parallel programming and work with the various tools included with Delphi Explore memory managers and their implementation Leverage external libraries to write better-performing programs Keep up to date with the latest additions and design techniques in Delphi Get to grips with various modern multithreading approaches Break a design problem down into its component parts Who this book is for This Learning Path is for intermediate-level Delphi programmers who want to build robust applications using Delphi features. Prior knowledge of Delphi is assumed.

A comprehensive guide for students and practitioners to parallel computing models, processes, metrics, and implementation in MPI and OpenMP.

Operating Systems (Self Edition 1.1.Abridged)

Computational Modeling and Complexity Science

Dive into CUDA

How to Think Like a Computer Scientist

Bits and Bugs

A Scientific and Historical Review of Software Failures in Computational Science

Models, Algorithms, Logics and Tools

Some previous editions of this book were published from Pearson Education (ISBN 9788131730225). This book, designed for those who are taking introductory courses on operating systems, presents both theoretical and practical aspects of modern operating systems. Although the emphasis is on theory, while exposing you (the reader) the subject matter, this book maintains a balance between theory and practice. The theories and technologies that have fueled the evolution of operating systems are primarily geared towards two goals: user convenience in maneuvering computers and efficient utilization of hardware resources. This book also discusses many fundamental concepts that have been formulated over the past several decades and that continue to be used in many modern operating systems. In addition, this book also discusses those technologies that prevail in many modern operating systems such as UNIX, Solaris, Linux, and Windows. While the former two have been used to present many in-text examples, the latter two are dealt with as separate technological case studies. They highlight the various issues in the design and development of operating systems and help you correlate theories to technologies. This book also discusses Android exposing you a modern software platform for embedded devices. This book supersedes ISBN 9788131730225 and its other derivatives, from Pearson Education India. (They have been used as textbooks in many schools worldwide.) You will definitely love this self edition, and you can use this as a textbook in undergraduate-level operating systems courses.

Basic Analysis V: Functional Analysis and Topology introduces graduate students in science to concepts from topology and functional analysis, both linear and nonlinear. It is the fifth book in a series designed to train interested readers how to think properly using mathematical abstractions, and how to use the tools of mathematical analysis in applications. It is important to realize that the most difficult part of applying mathematical reasoning to a new problem domain is choosing the underlying mathematical framework to use on the problem. Once that choice is made, we have many tools we can use to solve the problem. However, a different choice would open up avenues of analysis from a different, perhaps more productive, perspective. In this volume, the nature of these critical choices is discussed using applications involving the immune system and cognition. Features Develops a proof of the Jordan Canonical form to show some basic ideas in algebraic topology Provides a thorough treatment of topological spaces, finishing with the Krein-Milman theorem Discusses topological degree theory (Brouwer, Leray-Schauder, and Coincidence) Carefully develops manifolds and functions on manifolds ending with Riemannian metrics Suitable for advanced students in mathematics and associated disciplines Can be used as a traditional textbook as well as for self-study Author James K. Peterson is an Emeritus Professor at the School of Mathematical and Statistical Sciences, Clemson University. He tries hard to build interesting models of complex phenomena using a blend of mathematics, computation, and science. To this end, he has written four books on how to teach such things to biologists and cognitive scientists. These books grew out of his Calculus for Biologists courses offered to the biology majors from 2007 to 2015. He has taught the analysis courses since he started teaching both at Clemson and at his previous post at Michigan Technological University. In between, he spent time as a senior engineer in various aerospace firms and even did a short stint in a software development company. The problems he was exposed to were very hard, and not amenable to solution using just one approach. Using tools from many branches of mathematics, from many types of computational languages, and from first-principles analysis of natural phenomena was absolutely essential to make progress. In both mathematical and applied areas, students often need to use advanced mathematics tools they have not learned properly. So, he has recently written a series of five books on mathematical analysis to help researchers with the problem of learning new things after they have earned their degrees and are practicing scientists. Along the way, he has also written papers in immunology, cognitive science, and neural network technology, in addition to having grants from the NSF, NASA, and the US Army. He also likes to paint, build furniture, and write stories.

"How to Think Like a Computer Scientist: Learning with Python" is an introduction to computer science using the Python programming language. It covers the basics of computer programming, including variables and values, functions, conditionals and control flow, program development and debugging. Later chapters cover basic algorithms and data structures. \*\*\* Published under the terms of the GNU Free Documentation License. Money raised from the sale of this book supports the development of free software and documentation.

By using this innovative text, students will obtain an understanding of how contemporary operating systems and middleware work, and why they work that way.

Fundamental Approaches to Software Engineering

Three Easy Pieces

Basic Analysis V

Confidently build memory-safe, parallel, and efficient software in Rust

Parameterized Verification of Synchronized Concurrent Programs

Operating Systems and Middleware

Model Driven Engineering Languages and Systems

Gray zeroes right in on the key techniques of processes and interprocess communication from primitive communications to the complexities of sockets. The book covers every aspect of UNIX/Linux interprocess communications in sufficient detail to allow experienced programmers to begin writing useful code immediately.

This book constitutes the refereed proceedings of the 21st Ada-Europe International Conference on Reliable Software Technologies, Ada-Europe 2016, held in Pisa, Italy, in June 2016. The revised 12 full papers presented together with one invited paper were carefully reviewed and selected from 28 submissions. They are organized in topical sections on concurrency and parallelism, testing and verification, program correctness and robustness, and real-time systems.

This book constitutes the refereed proceedings of the TFM 2009, held in Eindhoven, The Netherlands in November 2009. The 10 revised full papers presented together with an abstracts of invited talk were carefully reviewed and selected from 19 submissions. The papers presented explore the experiences of teaching FMs, both successful and unsuccessful, educational resources including the use of books, case studies and the internet, the education of weak and mathphobic students, the integration, or otherwise, of FMs into the curriculum, including, contributions to the definition of a Formal Methods Body of Knowledge (FMBOK), the advantages

of FM-trained graduates in the workplace, changing attitudes towards FMs in students, academic staff and practitioners and the necessary mathematical background.

A practice-oriented guide to using C# to design and program pricing and trading models In this step-by-step guide to software development for financial analysts, traders, developers and quants, the authors show both novice and experienced practitioners how to develop robust and accurate pricing models and employ them in real environments. Traders will learn how to design and implement applications for curve and surface modeling, fixed income products, hedging strategies, plain and exotic option modeling, interest rate options, structured bonds, unfunded structured products, and more. A unique mix of modern software technology and quantitative finance, this book is both timely and practical. The approach is thorough and comprehensive and the authors use a combination of C# language features, design patterns, mathematics and finance to produce efficient and maintainable software. Designed for quant developers, traders and MSc/MFE students, each chapter has numerous exercises and the book is accompanied by a dedicated companion website, <http://www.datasimfinancial.com/forum/viewforum.php?f=196&sid=f30022095850dee48c7db5ff62192b34>, providing all source code, alongside audio, support and discussion forums for readers to comment on the code and obtain new versions of the software.

Mastering Delphi Programming: A Complete Reference Guide

Leveraging Applications of Formal Methods, Verification and Validation

Think Complexity

Python - 2008 Edition

Complexity Science and Computational Modeling

Operating Systems

10th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2021, Rhodes, Greece, October 17 – 29, 2021, Proceedings

COMPUTATIONAL MODELING AND COMPLEXITY SCIENCE \* Published under the terms of the GNU Free Documentation License. Money raised from the sale of this book supports the development of free software and documentation.

"This book is organized around three concepts fundamental to OS construction: virtualization (of CPU and memory), concurrency (locks and condition variables), and persistence (disks, RAIDS, and file systems"--Back cover.

Enhances Python skills by working with data structures and algorithms and gives examples of complex systems using exercises, case studies, and simple explanations.

Pharo is a modern open-source development environment for the classic Smalltalk-80 programming language. This book, intended for both students and developers, will guide you gently through the language and tools by means of a series of examples and exercises. We are making this book available to you under the Creative Commons Attribution-ShareAlike 3.0 license. You can either download the PDF for free from [PharoByExample.org](http://PharoByExample.org), or you can buy a softcover copy from [lulu.com](http://lulu.com). (You can also pay for the PDF download from [lulu.com](http://lulu.com), if you would like to make a contribution to this effort.) Additional material is available from the book's web page at [PharoByExample.org](http://PharoByExample.org).

Directed Algebraic Topology and Concurrency

Algorithms and Information Retrieval in Java

Interprocess Communications in Linux

The Designer's Guide to the Cortex-M Processor Family

Java Programming

19th International Conference, FASE 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings

Concurrency Control and Recovery in Database Systems

*How to Think Like a Computer Scientist - JAVA Version 4.1. (2008 Edition) \* Published under the terms of the GNU Free Documentation License. Money raised from the sale of this book supports the development of free software and documentation.*

*This book constitutes the proceedings of the 19th International Conference on Fundamental Approaches to Software Engineering, FASE 2016, which took place in Eindhoven, The Netherlands, in April 2016, held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016. The 23 full papers presented in this volume were carefully reviewed and selected from 90 submissions. They were organized in topical sections named: concurrent and distributed systems; model-driven development; analysis and bug triaging; probabilistic and stochastic systems; proof and theorem proving; and verification.*

*This Festschrift is published in honor of Kim Guldstrand Larsen, one of the earliest precursors of computer science in Denmark, on the occasion of his 60th birthday. During the last three decades, Kim Guldstrand Larsen has given major contributions across a remarkably wide range of topics, including real-time, concurrent, and probabilistic models of computation, logic in computer science, and model checking. Since 1995, he has been one of the prime movers behind the model checking tool for real-time systems UPPAAL, for which he was a co-recipient of the CAV Award in 2013. The Festschrift contains 32 papers that feature the broad range of Kim Guldstrand Larsen's research topics, such as formal languages and automata theory; logic; verification, model checking*

*and testing; algorithmic game theory and mechanism design; semantics and reasoning; real-time and distributed systems; and modeling and simulation. Operating systems are an essential part of any computer system. Similarly, a course on operating systems is an essential part of any computer science education. This field is undergoing rapid change, as computers are now prevalent in virtually every arena of day-to-day life—from embedded devices in automobiles through the most sophisticated planning tools for governments and multinational firms. Yet the fundamental concepts remain fairly clear, and it is on these that we base this book. We wrote this book as a text for an introductory course in operating systems at the junior or senior undergraduate level or at the first-year graduate level. We hope that practitioners will also find it useful. It provides a clear description of the concepts that underlie operating systems. As prerequisites, we assume that the reader is familiar with basic data structures, computer organization, and a high-level language, such as C or Java. The hardware topics required for an understanding of operating systems are covered in Chapter 1. In that chapter, we also include an overview of the fundamental data structures that are prevalent in most operating systems. For code examples, we use predominantly C, with some Java, but the reader can still understand the algorithms without a thorough knowledge of these languages. Concepts are presented using intuitive descriptions. Important theoretical results are covered, but formal proofs are largely omitted. The bibliographical notes at the end of each chapter contain pointers to research papers in which results were first presented and proved, as well as references to recent material for further reading. In place of proofs, figures and examples are used to suggest why we should expect the result in question to be true. The fundamental concepts and algorithms covered in the book are often based on those used in both commercial and open-source operating systems. Our aim is to present these concepts and algorithms in a general setting that is not tied to one particular operating system. However, we present a large number of examples that pertain to the most popular and the most innovative operating systems, including Linux, Microsoft Windows, Apple Mac OS X, and Solaris. We also include examples of both Android and iOS, currently the two dominant mobile operating systems.*

*The Little Book of Semaphores*

*C# for Financial Markets*

*The Ins and Outs of Concurrency Control and Common Mistakes*

*An Introduction to Software Design - How to Think Like a Computer Scientist*

*Little Book of Semaphores*

*Reliable Software Technologies – Ada-Europe 2016*

*Verification, Model Checking, and Abstract Interpretation*

**This book is devoted to the most difficult part of concurrent programming, namely synchronization concepts, techniques and principles when the cooperating entities are asynchronous, communicate through a shared memory, and may experience failures. Synchronization is no longer a set of tricks but, due to research results in recent decades, it relies today on sane scientific foundations as explained in this book. In this book the author explains synchronization and the implementation of concurrent objects, presenting in a uniform and comprehensive way the major theoretical and practical results of the past 30 years. Among the key features of the book are a new look at lock-based synchronization (mutual exclusion, semaphores, monitors, path expressions); an introduction to the atomicity consistency criterion and its properties and a specific chapter on transactional memory; an introduction to mutex-freedom and associated progress conditions such as obstruction-freedom and wait-freedom; a presentation of Lamport's hierarchy of safe, regular and atomic registers and associated wait-free constructions; a description of numerous wait-free constructions of concurrent objects (queues, stacks, weak counters, snapshot objects, renaming objects, etc.); a presentation of the computability power of concurrent objects including the notions of universal construction, consensus number and the associated Herlihy's hierarchy; and a survey of failure detector-based constructions of consensus objects. The book is suitable for advanced undergraduate students and graduate students in computer science or computer engineering, graduate students in mathematics interested in the foundations of process synchronization, and practitioners and engineers who need to produce correct concurrent software. The reader should have a basic knowledge of algorithms and operating systems.**

**Der Band bietet eine kompakte Einführung in die Nichtsequentielle Programmierung als gemeinsamen Kern von Vorlesungen über Betriebssysteme, Verteilte Systeme, Parallele Algorithmen, Echtzeitprogrammierung und Datenbanktransaktionen. Basiskonzepte zur Synchronisation und Kommunikation nebenläufiger Prozesse werden systematisch dargestellt: Schlösser, Semaphore, Monitore, lokaler und netzweiter Botschaftenaustausch. Die Algorithmen sind in der Programmiersprache Google Go formuliert, mit der viele Synchronisationskonzepte ausgedrückt werden können.**

**There is currently an increasing demand for concurrent programs. Checking the correctness of concurrent programs is a complex task due to the**

interleavings of processes. Sometimes, violation of the correctness properties in such systems causes human or resource losses; therefore, it is crucial to check the correctness of such systems. Two main approaches to software analysis are testing and formal verification. Testing can help discover many bugs at a low cost. However, it cannot prove the correctness of a program. Formal verification, on the other hand, is the approach for proving program correctness. Model checking is a formal verification technique that is suitable for concurrent programs. It aims to automatically establish the correctness (expressed in terms of temporal properties) of a program through an exhaustive search of the behavior of the system. Model checking was initially introduced for the purpose of verifying finite-state concurrent programs, and extending it to infinite-state systems is an active research area. In this thesis, we focus on the formal verification of parameterized systems. That is, systems in which the number of executing processes is not bounded a priori. We provide fully-automatic and parameterized model checking techniques for establishing the correctness of safety properties for certain classes of concurrent programs. We provide an open-source prototype for every technique and present our experimental results on several benchmarks. First, we address the problem of automatically checking safety properties for bounded as well as parameterized phaser programs. Phaser programs are concurrent programs that make use of the complex synchronization construct of Habanero Java phasers. For the bounded case, we establish the decidability of checking the violation of program assertions and the undecidability of checking deadlock-freedom. For the parameterized case, we study different formulations of the verification problem and propose an exact procedure that is guaranteed to terminate for some reachability problems even in the presence of unbounded phases and arbitrarily many spawned processes. Second, we propose an approach for automatic verification of parameterized concurrent programs in which shared variables are manipulated by atomic transitions to count and synchronize the spawned processes. For this purpose, we introduce counting predicates that related counters that refer to the number of processes satisfying some given properties to the variables that are directly manipulated by the concurrent processes. We then combine existing works on the counter, predicate, and constrained monotonic abstraction and build a nested counterexample-based refinement scheme to establish correctness. Third, we introduce Lazy Constrained Monotonic Abstraction for more efficient exploration of well-structured abstractions of infinite-state non-monotonic systems. We propose several heuristics and assess the efficiency of the proposed technique by extensive experiments using our open-source prototype. Lastly, we propose a sound but (in general) incomplete procedure for automatic verification of safety properties for a class of fault-tolerant distributed protocols described in the Heard-Of (HO for short) model. The HO model is a popular model for describing distributed protocols. We propose a verification procedure that is guaranteed to terminate even for unbounded number of the processes that execute the distributed protocol.

This book constitutes the refereed proceedings of the 17th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI 2016, held in St. Petersburg, FL, USA, in January 2016. The 24 full papers together with 2 invited talks and 1 abstract presented were carefully reviewed and selected from 67 submissions. VMCAI provides topics including: program verification, model checking, abstractinterpretation and abstract domains, program synthesis, static analysis, type systems, deductive methods, program certification, debugging techniques, program transformation, optimization, hybrid and cyber-physical systems.

Second International Conference, TFM 2009, Eindhoven, The Netherlands, November 2-6, 2009, Proceedings

Learning with Python

Membrane Computing

Internals and Design Principles

Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday

(See other editions at <https://books.google.com/books/?id=zSbxCwAAQBAJ> and decide one)

12th International Conference, CMC 2011, Fontainebleau, France, August 23-26, 2011, Revised Selected Papers

*This monograph presents an application of concepts and methods from algebraic topology to models of concurrent processes in computer science and their analysis. Taking well-known discrete models for concurrent processes in resource management as a point of departure, the book goes on to refine combinatorial and topological models. In the process, it develops tools and invariants for the new discipline directed algebraic topology, which is driven by fundamental research interests as well as by applications, primarily in the static analysis of concurrent programs. The state space of a concurrent program is described as a higher-dimensional space, the topology of which encodes the essential properties of the system. In order to analyse all possible executions in the state space, more than "just" the topological properties have to be considered: Execution paths need to respect a partial order given by the time flow. As a result, tools and concepts from topology have to be extended to take privileged directions into account. The target audience for this book consists of graduate students, researchers and practitioners in the field, mathematicians and computer scientists alike.*

*Python for Software Design is a concise introduction to software design using the Python programming language. Intended for people with no programming experience, this book starts with the most basic concepts and gradually adds new material. Some of the ideas students find most challenging, like recursion and object-oriented programming, are*

*divided into a sequence of smaller steps and introduced over the course of several chapters. The focus is on the programming process, with special emphasis on debugging. The book includes a wide range of exercises, from short examples to substantial projects, so that students have ample opportunity to practice each new concept. Exercise solutions and code examples are available from thinkpython.com, along with Swampy, a suite of Python programs that is used in some of the exercises. \*\* Published under the terms of the GNU Free Documentation License. Money raised from the sale of this book supports the development of free software and documentation.*

*In scientific computing (also known as computational science), advanced computing capabilities are used to solve complex problems. This self-contained book describes and analyzes reported software failures related to the major topics within scientific computing: mathematical modeling of phenomena; numerical analysis (number representation, rounding, conditioning); mathematical aspects and complexity of algorithms, systems, or software; concurrent computing (parallelization, scheduling, synchronization); and numerical data (such as input of data and design of control logic). Readers will find lists of related, interesting bugs, MATLAB examples, and "excursions" that provide necessary background, as well as an in-depth analysis of various aspects of the selected bugs. Illustrative examples of numerical principles such as machine numbers, rounding errors, condition numbers, and complexity are also included.*

*For a one-semester undergraduate course in operating systems for computer science, computer engineering, and electrical engineering majors. Winner of the 2009 Textbook Excellence Award from the Text and Academic Authors Association (TAA)! Operating Systems: Internals and Design Principles is a comprehensive and unified introduction to operating systems. By using several innovative tools, Stallings makes it possible to understand critical core concepts that can be fundamentally challenging. The new edition includes the implementation of web based animations to aid visual learners. At key points in the book, students are directed to view an animation and then are provided with assignments to alter the animation input and analyze the results. The concepts are then enhanced and supported by end-of-chapter case studies of UNIX, Linux and Windows Vista. These provide students with a solid understanding of the key mechanisms of modern operating systems and the types of design tradeoffs and decisions involved in OS design. Because they are embedded into the text as end of chapter material, students are able to apply them right at the point of discussion. This approach is equally useful as a basic reference and as an up-to-date survey of the state of the art.*

*Functional Analysis and Topology*

*Supporting Controlled Interaction*

*Delphi High Performance*

*17th International Conference, VMCAI 2016, St. Petersburg, FL, USA, January 17-19, 2016. Proceedings*

*21st Ada-Europe International Conference on Reliable Software Technologies, Pisa, Italy, June 13-17, 2016, Proceedings*

*The Little Book of SEMAPHORES (2nd Edition)*

Build fast, scalable, and high performing applications with Delphi Key Features Build efficient and concurrent applications in Delphi with focused examples Identify performance bottlenecks and apply the correct algorithm to increase the performance of applications. Delve into parallel programming and memory management to optimize your code Book Description Delphi is a cross-platform Integrated Development Environment (IDE) that supports rapid application development for Microsoft Windows, Apple Mac OS X, Google Android, iOS, and now Linux with RAD Studio 10.2. This book will be your guide to build efficient high performance applications with Delphi. The book begins by explaining how to find performance bottlenecks and apply the correct algorithm to fix them. It will teach you how to improve your algorithms before taking you through parallel programming. You'll then explore various tools to build highly concurrent applications. After that, you'll delve into improving the performance of your code and master cross-platform RTL improvements. Finally, we'll go through memory management with Delphi and you'll see how to leverage several external libraries to write better performing programs. By the end of the book, you'll have the knowledge to create high performance applications with Delphi. What you will learn Find performance bottlenecks and easily mitigate them Discover different approaches to fix algorithms Understand parallel programming and work with various tools included with Delphi Master the RTL for code optimization Explore memory managers and their implementation Leverage external libraries to write better performing programs Who this book is for This book is for Delphi developers who would like to build high performance applications with Delphi. Prior knowledge of Delphi is assumed.

The Designer's Guide to the Cortex-M Microcontrollers gives you an easy-to-understand introduction to the concepts required to develop programs in C with a Cortex-M based microcontroller. The book begins with an overview of the Cortex-M family, giving architectural descriptions supported with practical examples, enabling you to easily develop basic C programs to run on the Cortex-

M0/M0+/M3 and M4 and M7. It then examines the more advanced features of the Cortex architecture such as memory protection, operating modes, and dual stack operation. Once a firm grounding in the Cortex-M processor has been established the book introduces the use of a small footprint RTOS and the CMSIS-DSP library. The book also examines techniques for software testing and code reuse specific to Cortex-M microcontrollers. With this book you will learn: the key differences between the Cortex-M0/M0+/M3 and M4 and M7; how to write C programs to run on Cortex-M based processors; how to make the best use of the CoreSight debug system; the Cortex-M operating modes and memory protection; advanced software techniques that can be used on Cortex-M microcontrollers; how to use a Real Time Operating System with Cortex-M devices; how to optimize DSP code for the Cortex-M4; and how to build real time DSP systems. Includes an update to the latest version (5) of MDK-ARM, which introduces the concept of using software device packs and software components Includes overviews of the new CMSIS specifications Covers developing software with CMSIS-RTOS showing how to use RTOS in a real world design Provides a new chapter on the Cortex-M7 architecture covering all the new features Includes a new chapter covering test driven development for Cortex-M microcontrollers Features a new chapter on creating software components with CMSIS-Pack and device abstraction with CMSIS-Driver Features a new chapter providing an overview of the ARMv8-M architecture including the TrustZone hardware security model

Pharo by Example

12th International Conference, MODELS 2009, Denver, CO, USA, October 4-9, 2009, Proceedings