Lambda Calculus And Combinators An Introduction

Dictionary of Logic as Applied in the Study of LanguageLecture Notes on the Lambda CalculusLisp in Small PiecesLeveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering ChangeComputer Science LogicLambda-calculus, Combinators and Functional ProgrammingRandomness and ComplexityLambda Calculus with TypesCombinators, Lambda-Terms and Proof TheoryThe Parametric Lambda CalculusFoundations of Software Science and Computation StructuresConcepts in Programming LanguagesReduction and Type-assignment for Lambda-calculus and CombinatorsBasic Simple Type TheoryUnderstanding ComputationCentral European Functional Programming SchoolA++ The Smallest Programming Language in the WorldThe Calculi of Lambda-conversionTo Mock a MockingbirdIntroduction to Combinators and (lambda) CalculusLogic, Meaning and ComputationThe Lambda CalculusType Theory and Formal ProofAn Introduction to Functional Programming Through Lambda CalculusCombinatory LogicCombinatory LogicLectures on the Curry-Howard IsomorphismLambda-calculus, Types and ModelsThe Pi-CalculusProgramming Languages and SystemsTerm Rewriting SystemsIntroduction to Functional Programming Systems Using HaskellPattern CalculusCategorical Combinators, Sequential Algorithms, and Functional ProgrammingAdvances in

Artificial LifeLambda-Calculus and CombinatorsThe Implementation of Functional Programming LanguagesIntroduction to Combinators and the Lambda-CalculusAn Introduction to Lambda Calculi for Computer ScientistsIntroduction to Combinatory Logic

Dictionary of Logic as Applied in the Study of Language

Type theory is a fast-evolving field at the crossroads of logic, computer science and mathematics. This gentle step-by-step introduction is ideal for graduate students and researchers who need to understand the ins and outs of the mathematical machinery, the role of logical rules therein, the essential contribution of definitions and the decisive nature of well-structured proofs. The authors begin with untyped lambda calculus and proceed to several fundamental type systems, including the well-known and powerful Calculus of Constructions. The book also covers the essence of proof checking and proof development, and the use of dependent type theory to formalise mathematics. The only prerequisite is a basic knowledge of undergraduate mathematics. Carefully chosen examples illustrate the theory throughout. Each chapter ends with a summary of the content, some historical context, suggestions for further reading and a selection of exercises to help readers familiarise themselves with the material.

Lecture Notes on the Lambda Calculus

A++ has been developed in 2002 in the context of 'Programmierung pur' [Undiluted Programming] (ISBN 3-87820-108-7) with the purpose to serve as a learning instrument rather than as a programming language used to solve practical problems. A++ is supposed to be an efficient tool to become familiar with the core of programming and with programming patterns that can be applied in other languages needed to face the real world. This book does not only introduce A++ as a language, but also covers its implementation in Perl and C including an introduction to these languages using A++ itself. The book also contains an introduction to the Lambda-Calculus of Alonzo Church, which represents the theoretical foundation of A++.

Lisp in Small Pieces

This open access book constitutes the proceedings of the 23rd International Conference on Foundations of Software Science and Computational Structures, FOSSACS 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 31 regular papers presented in this volume were carefully reviewed and selected from 98 submissions. The papers cover topics such as categorical models and logics; language theory, automata, and games; modal, spatial, and temporal logics; type theory and proof theory; concurrency theory and process calculi; rewriting theory; semantics of programming languages; program analysis, correctness, transformation, and verification; logics of programming; software specification and refinement; models of concurrent, reactive, stochastic, distributed, hybrid, and mobile systems; emerging models of computation; logical aspects of computational complexity; models of software security; and logical foundations of data bases.

Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change

This introduction to lambda-calculus looks at aspects of the theory: combinatory logic, models, and type streams, showing how they interlink and underpin computer science.

Computer Science Logic

Well-respected text for computer science students provides an accessible introduction to functional programming. Cogent examples illuminate the central ideas, and numerous exercises offer reinforcement. Includes solutions. 1989 edition.

Lambda-calculus, Combinators and Functional Programming

Finally, you can learn computation theory and programming language design in an engaging, practical way. Understanding Computation explains theoretical computer science in a context you'll recognize, helping you appreciate why these ideas matter and how they can inform your day-to-day programming. Rather than use mathematical notation or an unfamiliar academic programming language like Haskell or Lisp, this book uses Ruby in a reductionist manner to present formal semantics, automata theory, and functional programming with the lambda calculus. It's ideal for programmers versed in modern languages, with little or no formal training in computer science. Understand fundamental computing concepts, such as Turing completeness in languages Discover how programs use dynamic semantics to communicate ideas to machines Explore what a computer can do when reduced to its bare essentials Learn how universal Turing machines led to today's general-purpose computers Perform complex calculations, using simple languages and cellular automata Determine which programming language features are essential for computation Examine how halting and self-referencing make some computing problems unsolvable Analyze programs by using abstract interpretation and type systems

Randomness and Complexity

Combinatory logic is one of the most versatile areas within logic that is tied to parts of philosophical, mathematical, and computational logic. Functioning as a comprehensive source for current developments of combinatory logic, this book is the only one of its kind to cover results of the last four decades. Using a reader-friendly style, the author presents the most up-to-date research studies. She includes an introduction to combinatory logic before progressing to its central theorems and proofs. The text makes intelligent and well-researched connections between combinatory logic and lambda calculi and presents models and applications to illustrate these connections.

Lambda Calculus with Types

Over time, basic research tends to lead to specialization – increasingly narrow t- ics are addressed by increasingly focussed communities, publishing in increasingly con ned workshops and conferences, discussing increasingly incremental contri- tions. Already the community of programming languages is split into various s- communities addressing different aspects and paradigms (functional, imperative, relational, and object-oriented). Only a few people manage to maintain a broader view, and even fewer step back in order to gain an understanding about the basic principles, their interrelation, and their impact in a larger context. The pattern calculus is the result of a profound re-examination of a 50-year - velopment. It attempts to provide a unifying approach, bridging the gaps between different programming styles and paradigms according to a new slogan – compu- tion is pattern matching. It is the contribution of this book to systematically and elegantly present and evaluate the power of pattern matching as the guiding paradigm of programming. Patterns are dynamically generated, discovered, passed, applied, and automatically adapted, based on pattern matching and rewriting technology, which allows one to elegantly relate things as disparate as functions and data structures. Of course, pattern matching is not new. It underlies term rewriting – it is, for example, inc- porated in, typically functional, programming languages, like Standard ML – but it has never been pursued as the basis of a unifying framework for programming.

Combinators, Lambda-Terms and Proof Theory

This volume presents eight carefully revised texts from selected lectures given by leading researchers at the Second Central European Functional Programming School, CEFP 2007, held in Cluj-Napoca, Romania, in June 2007. The eight revised full papers presented were carefully selected during two rounds of reviewing and improvement for inclusion in the book. The lectures cover a wide range of topics such as interactive workflows, lazy functional programs, lambda calculus, and object-oriented functional programming.

The Parametric Lambda Calculus

This volume began as a remembrance of Alonzo Church while he was still with us and is now finally complete. It contains papers by many well-known scholars, most of whom have been directly influenced by Church's own work. Often the emphasis is on foundational issues in logic, mathematics, computation, and philosophy - as was the case with Church's contributions, now universally recognized as having been of profound fundamental significance in those areas. The volume will be of interest to logicians, computer scientists, philosophers, and linguists. The contributions concern classical first-order logic, higher-order logic, non-classical theories of implication, set theories with universal sets, the logical and semantical paradoxes, the lambda-calculus, especially as it is used in computation, philosophical issues about meaning and ontology in the abstract sciences and in natural language, and much else. The material will be accessible to specialists in these areas and to advanced graduate students in the respective fields.

Foundations of Software Science and Computation Structures

Originally published in 1988, this book presents an introduction to lambda-calculus and combinators without getting lost in the details of mathematical aspects of their theory. Lambda-calculus is treated here as a functional language and its relevance to computer science is clearly demonstrated. The main purpose of the book is to provide computer science students and researchers with a firm background in lambda-calculus and combinators and show the applicabillity of these theories to functional programming. The presentation of the material is self-contained. It can be used as a primary text for a course on functional programming. It can also be used as a supplementary text for courses on the structure and implementation of programming languages, theory of computing, or semantics of programming languages.

Concepts in Programming Languages

The revised edition contains a new chapter which provides an elegant description of the semantics. The various classes of lambda calculus models are described in a uniform manner. Some didactical improvements have been made to this edition. An example of a simple model is given and then the general theory (of categorical models) is developed. Indications are given of those parts of the book which can be used to form a coherent course.

Reduction and Type-assignment for Lambda-calculus and

Combinators

The basic concepts of applicative programming are presented using the language HASKELL for examples. In addition to exploring the implications for parallelism, a discussion of lamda calculus and its relationship with SASL is included.

Basic Simple Type Theory

Combinatory logic and lambda-calculus, originally devised in the 1920's, have since developed into linguistic tools, especially useful in programming languages. The authors' previous book served as the main reference for introductory courses on lambda-calculus for over 20 years: this long-awaited new version is thoroughly revised and offers a fully up-to-date account of the subject, with the same authoritative exposition. The grammar and basic properties of both combinatory logic and lambda-calculus are discussed, followed by an introduction to type-theory. Typed and untyped versions of the systems, and their differences, are covered. Lambda-calculus models, which lie behind much of the semantics of programming languages, are also explained in depth. The treatment is as non-technical as possible, with the main ideas emphasized and illustrated by examples. Many exercises are included, from routine to advanced, with solutions to most at the end of the book.

Understanding Computation

The book contains a completely new presentation of classical results in the field of Lambda Calculus, together with new results. The text is unique in that it presents a new calculus (Parametric Lambda Calculus) which can be instantiated to obtain already known lambda-calculi. Some properties, which in the literature have been proved separately for different calculi, can be proved once for the Parametric one. The lambda calculi are presented from a Computer Science point of view, with a particular emphasis on their semantics, both operational and denotational.

Central European Functional Programming School

The lambda-calculus lies at the very foundations of computer science. Besides its historical role in computability theory it has had significant influence on programming language design and implementation, denotational semantics, and domain theory. The book emphasises the proof theory for the type-free lambda-calculus. The first six chapters concern this calculus and cover the basic theory, reduction, models, computability, and the relationship between the lambda-calculus and combinatory logic. Chapter 7 presents a variety of typed calculi; first the simply typed lambda-calculus, then Milner-style polymorphism and, finally, the polymorphic lambda-calculus. Chapter 8 concerns two variants of the type-free lambda-calculus that have appeared in the research literature: the lazy lambda-calculus, and the lambda sigma-calculus. The final chapter contains references and a guide to further reading. There are exercises throughout. In contrast to earlier books on these topics, which were written by logicians, this book is written from a computer science perspective and emphasises the practical relevance of many of the key theoretical ideas. The book is intended as a course text for final year undergraduates or first year graduate students in computer science. Research students should find it a useful introduction to more specialist literature.

A++ The Smallest Programming Language in the World

This is a set of lecture notes that developed out of courses on the lambda calculus that the author taught at the University of Ottawa in 2001 and at Dalhousie University in 2007 and 2013. Topics covered in these notes include the untyped lambda calculus, the Church-Rosser theorem, combinatory algebras, the simply-typed lambda calculus, the Curry-Howard isomorphism, weak and strong normalization, polymorphism, type inference, denotational semantics, complete partial orders, and the language PCF.

The Calculi of Lambda-conversion

A comprehensive undergraduate textbook covering both theory and practical design issues, with an emphasis on object-oriented languages.

To Mock a Mockingbird

Type theory is one of the most important tools in the design of higher-level programming languages, such as ML. This book introduces and teaches its techniques by focusing on one particularly neat system and studying it in detail. By concentrating on the principles that make the theory work in practice, the author covers all the key ideas without getting involved in the complications of more advanced systems. This book takes a type-assignment approach to type theory, and the system considered is the simplest polymorphic one. The author covers all the basic ideas, including the system's relation to propositional logic, and gives a careful treatment of the type-checking algorithm that lies at the heart of every such system. Also featured are two other interesting algorithms that until now have been buried in inaccessible technical literature. The mathematical presentation is rigorous but clear, making it the first book at this level that can be used as an introduction to type theory for computer scientists.

Introduction to Combinators and (lambda) Calculus

This book is a revised edition of the monograph which appeared under the same title in the series Research Notes in Theoretical Computer Science, Pit man, in 1986. In addition to a general effort to improve typography, English, and presentation, the main novelty of this second edition is the integration of some new material. Part of it is mine (mostly jointly with coauthors). Here is brief guide to these additions. I have augmented the account of categorical combinatory logic with a description of the confluence properties of rewriting systems of categor ical combinators (Hardin, Yokouchi), and of the newly developed cal culi of explicit substitutions (Abadi, Cardelli, Curien, Hardin, Levy, and Rios), which are similar in spirit to the categorical combinatory logic, but are closer to the syntax of A-calculus (Section 1.2). The study of the full abstraction problem for PCF and extensions of it has been enriched with a new full abstraction result: the model of sequential algorithms is fully abstract with respect to an extension of PCF with a control operator (Cartwright, Felleisen, Curien). An order extensional model of error-sensitive sequential algorithms is also fully abstract for a corresponding extension of PCF with a control operator and errors (Sections 2.6 and 4.1). I suggest that sequential algorithms lend themselves to a decomposition of the function spaces that leads to models of linear logic (Lamarche, Curien), and that connects sequentiality with games (Joyal, Blass, Abramsky) (Sections 2.1 and 2.6).

Logic, Meaning and Computation

Table of contents

The Lambda Calculus

In this entertaining and challenging collection of logic puzzles, Raymond Smullyan - author of Forever Undecided - continues to delight and astonish us with his gift for making available, in the thoroughly pleasurable form of puzzles, some of the most important mathematical thinking of our time. In the first part of the book, he transports us once again to that wonderful realm where knights, knaves, twin sisters, quadruplet brothers, gods, demons, and mortals either always tell thetruth or always lie, and where truth-seekers are set a variety of fascinating problems. The section culminates in an enchanting and profound metapuzzle in which Inspector Craig of Scotland Yard gets involved in a search for the Fountain of Youth on the Island of Knights and Knaves. In the second part of To Mock a Mockingbird, we accompany the Inspector on a summer-long adventure into the field of combinatory logic (a branch of logic that plays an important role in computer science and artificial intelligence). His adventure, which includes enchanted forests, talking birds, bird sociologists, and a classic quest, provides for us along the way the pleasure of solving puzzles of increasing complexity until we reach the Master Forest and - thanks to Godel's famous theorem - the final revelation.

Type Theory and Formal Proof

An Introduction to Functional Programming Through Lambda Calculus

Graduate text on the p-calculus, a mathematical model of mobile computing systems.

Combinatory Logic

Combinatory logic and lambda-conversion were originally devised in the 1920s for investigating the foundations of mathematics using the basic concept of 'operation' instead of 'set'. They have now developed into linguistic tools, useful in several branches of logic and computer science, especially in the study of programming languages. These notes form a simple introduction to the two topics, suitable for a reader who has no previous knowledge of combinatory logic, but has taken an undergraduate course in predicate calculus and recursive functions. The key ideas and basic results are presented, as well as a number of more specialised topics, and man), exercises are included to provide manipulative practice.

Combinatory Logic

This workshop on stochastic theory and adaptive control assembled many of the leading researchers on stochastic control and stochastic adaptive control to increase scientific exchange and cooperative research between these two subfields of stochastic analysis. The papers included in the proceedings include survey and research. They describe both theoretical results and applications of adaptive control. There are theoretical results in identification, filtering, control, adaptive control and various other related topics. Some applications to manufacturing systems, queues, networks, medicine and other topics are gien.

Lectures on the Curry-Howard Isomorphism

1. STRUCTURE AND REFERENCES 1.1. The main part of the dictionary consists of alphabetically arranged articles concerned with basic logical theories and some other selected topics. Within each article a set of concepts is defined in their mutual relations. This way of defining concepts in the context of a theory provides better understand ing of ideas than that provided by isolated short defmitions. A disadvantage of this method is that it takes more time to look something up inside an extensive article. To reduce this disadvantage the following measures have been adopted. Each article is divided into numbered sections, the numbers, in

boldface type, being addresses to which we refer. Those sections of larger articles which are divided at the first level, i.e. numbered with single numerals, have titles. Main sections are further subdivided, the subsections being numbered by numerals added to the main section number, e.g. I, 1.1, 1.2, , 1.1.1, 1.1.2, and so on. A comprehensive subject index is supplied together with a glossary. The aim of the latter is to provide, if possible, short defmitions which sometimes may prove sufficient. As to the use of the glossary, see the comment preceding it.

Lambda-calculus, Types and Models

The book is a collection of papers written by a selection of eminent authors from around the world in honour of Gregory Chaitin's 60th birthday. This is a unique volume including technical contributions, philosophical papers and essays.

The Pi-Calculus

Programming Languages and Systems

This handbook with exercises reveals in formalisms, hitherto mainly used for hardware and software design and verification, unexpected mathematical beauty.

The lambda calculus forms a prototype universal programming language, which in its untyped version is related to Lisp, and was treated in the first author's classic The Lambda Calculus (1984). The formalism has since been extended with types and used in functional programming (Haskell, Clean) and proof assistants (Coq, Isabelle, HOL), used in designing and verifying IT products and mathematical proofs. In this book, the authors focus on three classes of typing for lambda terms: simple types, recursive types and intersection types. It is in these three formalisms of terms and types that the unexpected mathematical beauty is revealed. The treatment is authoritative and comprehensive, complemented by an exhaustive bibliography, and numerous exercises are provided to deepen the readers' understanding and increase their confidence using types.

Term Rewriting Systems

The two-volume set LNCS 8802 and LNCS 8803 constitutes the refereed proceedings of the 6th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, ISoLA 2014, held in Imperial, Corfu, Greece, in October 2014. The total of 67 full papers was carefully reviewed and selected for inclusion in the proceedings. Featuring a track introduction to each section, the papers are organized in topical sections named: evolving critical systems; rigorous engineering of autonomic ensembles; automata learning; formal methods and analysis in software product line engineering; model-based code generators and compilers; engineering virtualized systems; statistical model checking; risk-based testing; medical cyber-physical systems; scientific workflows; evaluation and reproducibility of program analysis; processes and data integration in the networked healthcare; semantic heterogeneity in the formal development of complex systems. In addition, part I contains a tutorial on automata learning in practice; as well as the preliminary manifesto to the LNCS Transactions on the Foundations for Mastering Change with several position papers. Part II contains information on the industrial track and the doctoral symposium and poster session.

Introduction to Functional Programming Systems Using Haskell

The Curry-Howard isomorphism states an amazing correspondence between systems of formal logic as encountered in proof theory and computational calculi as found in type theory. For instance, minimal propositional logic corresponds to simply typed lambda-calculus, first-order logic corresponds to dependent types, second-order logic corresponds to polymorphic types, sequent calculus is related to explicit substitution, etc. The isomorphism has many aspects, even at the syntactic level: formulas correspond to types, proofs correspond to terms, provability corresponds to inhabitation, proof normalization corresponds to term reduction, etc. But there is more to the isomorphism than this. For instance, it is an old idea---due to Brouwer, Kolmogorov, and Heyting---that a constructive proof of an implication is a procedure that transforms proofs of the antecedent into proofs of the succedent; the Curry-Howard isomorphism gives syntactic representations of such procedures. The Curry-Howard isomorphism also provides theoretical foundations for many modern proof-assistant systems (e.g. Coq). This book give an introduction to parts of proof theory and related aspects of type theory relevant for the Curry-Howard isomorphism. It can serve as an introduction to any or both of typed lambda-calculus and intuitionistic logic. Key features - The Curry-Howard Isomorphism treated as common theme - Reader-friendly introduction to two complementary subjects: Lambda-calculus and constructive logics - Thorough study of the connection between calculi and logics - Elaborate study of classical logics and control operators - Account of dialogue games for classical and intuitionistic logic - Theoretical foundations of computer-assisted reasoning · The Curry-Howard Isomorphism treated as the common theme. · Reader-friendly introduction to two complementary subjects: lambda-calculus and constructive logics · Thorough study of the connection between calculi and logics. · Elaborate study of classical logics and control operators. · Account of dialogue games for classical and intuitionistic logic. · Theoretical foundations of computer-assisted reasoning

Pattern Calculus

The description for this book, The Calculi of Lambda Conversion. (AM-6), Volume 6, will be forthcoming.

Categorical Combinators, Sequential Algorithms, and Functional Programming

This book constitutes the proceedings of the 25th European Symposium on Programming, ESOP 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 29 papers presented in this volume were carefully reviewed and selected from 98 submissions. Being devoted to fundamental issues in the specification, design, analysis, and implementation of programming languages and systems, ESOP features contributions on all aspects of programming language research; theoretical and/or practical advances.

Advances in Artificial Life

Lambda-Calculus and Combinators

The Implementation of Functional Programming Languages

Introduction to Combinators and the Lambda-Calculus

This is a comprehensive account of the semantics and the implementation of the whole Lisp family of languages, namely Lisp, Scheme and related dialects. It describes 11 interpreters and 2 compilers, including very recent techniques of interpretation and compilation. The book is in two parts. The first starts from a simple evaluation function and enriches it with multiple name spaces, continuations and side-effects with commented variants, while at the same time the language used to define these features is reduced to a simple lambda-calculus. Denotational semantics is then naturally introduced. The second part focuses more on implementation techniques and discusses precompilation for fast interpretation: threaded code or bytecode; compilation towards C. Some extensions are also described such as dynamic evaluation, reflection, macros and objects. This will become the new standard reference for people wanting to know more about the Lisp family of languages: how they work, how they are implemented, what their variants are and why such variants exist. The full code is supplied (and also available over the Net). A large bibliography is given as well as a considerable number of exercises. Thus it may also be used by students to accompany second courses on Lisp or Scheme.

An Introduction to Lambda Calculi for Computer Scientists

The aim of this monograph is to present some of the basic ideas and results in pure combinatory logic and their applications to some topics in proof theory, and also to present some work of my own. Some of the material in chapter 1 and 3 has already appeared in my notes Introduction to Combinatory Logic. It appears here in revised form since the presen tation in my notes is inaccurate in several respects. I would like to express my gratitude to Stig Kanger for his invalu able advice and encouragement and also for his assistance in a wide variety of matters concerned with my study in Uppsala. I am also in debted to Per Martin-USf for many valuable and instructive conversa tions. As will be seen in chapter 4 and 5, I also owe much to the work of Dag Prawitz and W. W. Tait. My thanks also to Craig McKay who read the manuscript and made valuable suggestions. I want, however, to emphasize that the shortcomings that no doubt can be found, are my sole responsibility. Uppsala, February 1972.

Introduction to Combinatory Logic

This book constitutes the refereed proceedings of the 7th European Conference on Artificial Life, ECAL 2003, held in Dortmund, Germany in September 2003. The 96 revised full papers presented were carefully reviewed and selected from more than 140 submissions. The papers are organized in topical sections on artificial chemistries, self-organization, and self-replication; artificial societies; cellular and neural systems; evolution and development; evolutionary and adaptive dynamics;

languages and communication; methodologies and applications; and robotics and autonomous agents.