# Programming Languages And Operational Semantics: A Concise Overview

# (Undergraduate Topics In Computer Science)

The first comprehensive presentation of reduction

semantics in one volume, and the first tool set for such forms of semantics. This text is the first comprehensive presentation of reduction semantics in one volume; it also introduces the first

reliable and easy-to-use tool set for such forms of semantics. Software engineers have long known that automatic tool support is critical for rapid prototyping and modeling, and this book is addressed

to the working semantics engineer (graduate student or professional language designer). The book comes with a prototyping tool suite to develop, explore, test, debug, and publish semantic models of

programming languages. With PLT Redex, semanticists can formulate models as grammars and reduction models on their computers with the ease of paper and pencil. The text first presents a framework for the

formulation of language models, focusing on equational calculi and abstract machines, then introduces PLT Redex, a suite of software tools for expressing these models as PLT Redex models. Finally,

experts describe a range of models formulated in Redex. PLT Redex comes with the PLT Scheme implementation, available free at http://www.plt-scheme.org/. Readers can download the software and experiment with

Redex as they work their way through the book.

This book is based on material presented at the international summer school on Applied Semantics that took place in Caminha, Portugal, in September 2000.

We aim to present some recent developments in programming language research, both in semantic theory and in implementation, in a series of graduate-level lectures. The school was sponsored by

the ESPRIT Working Group 26142 on Applied Semantics(APPSEM),whichoperatedbetweenApril1998andMarch2002.The purpose of this working group was to bring together leading reseachers, both in semantic theory and in

implementation, with the speci?c aim of improving the communication between theoreticians and practitioners. Theactivities ofAPPSEMwerestructuredintoni neinterdisciplinarythemes: A: Semantics for object-

oriented programming B: Program structuring C: Integration of functional languages and proof assistants D: Veri?cation methods E: Automatic program transformation F: Games, sequentiality, and abstract

machines G: Types and type inference in programming H: Semantics-based optimization I: Domain theory and real number computation These themes were identi?ed as promising for pro?table interaction between semantic

theory and practice, and were chosen to contribute to the following general topics: – description of existing programming language features; – design of new programming language features; – implementation

and analysis of programming languages; – transformation and generation of programs; – veri?cation of programs. The chapters in this volume give examples of recent developments covering a broad range of topics of

interest to APPSEM. Structural operational semantics is a simple, yet powerful mathematical theory for describing the behaviour of programs in an implementation-independent manner. This book provides a

self-contained introduction to structural operational semantics, featuring semantic definitions using big-step and small-step semantics of many standard programming language constructs, including

control structures, structured declarations and objects, parameter mechanisms and procedural abstraction, concurrency, nondeterminism and the features of functional programming languages. Along

the way, the text introduces and applies the relevant proof techniques, including forms of induction and notions of semantic equivalence (including bisimilarity). Thoroughly class-tested, this book has

evolved from lecture notes used by the author over a 10-year period at Aalborg University to teach undergraduate and graduate students. The result is a thorough introduction that makes the subject clear to

students and computing professionals without sacrificing its rigour. No experience with any specific programming language is required.

Applied Semantics

Semantics of Programming

Languages
An Introduction
Transitions and Trees
Logic Programming
Essentials of Programming Languages, third edition

This book provides an introduction to the

essential concepts in programming languages, using operational semantics techniques. It presents alternative programming language paradigms and gives an in-

depth analysis of the most significant constructs in modern imperative, functional and logic programming languages. The book is designed to accompany lectures on

programming language design for undergraduate students. Each chapter includes exercises which provide the opportunity to apply the concepts and techniques presented.

First published in 1998, this textbook is a broad but rigourous survey of the theoretical basis for the design, definition and implementation of programming languages and

of systems for specifying and proving programme behaviour. Both imperative and functional programming are covered, as well as the ways of integrating these aspects into more

general languages. Recognising a unity of technique beneath the diversity of research in programming languages, the author presents an integrated treatment of

the basic principles of the subject. He identifies the relatively small number of concepts, such as compositional semantics, binding structure, domains,

transition systems and inference rules, that serve as the foundation of the field. Assuming only knowledge of elementary programming and mathematics, this text is

perfect for advanced undergraduate and beginning graduate courses in programming language theory and also will appeal to researchers and professionals in designing

or implementing computer languages.

Part I of this book is a practical introduction to working with the Isabelle proof assistant. It teaches you how to write

functional programs and inductive definitions and how to prove properties about them in Isabelle's structured proof language. Part II is an introduction to the semantics of

imperative languages with an emphasis on applications like compilers and program analysers. The distinguishing feature is that all the mathematics

has been formalised in Isabelle and much of it is executable. Part I focusses on the details of proofs in Isabelle; Part II can be read even without familiarity with

Isabelle's proof language, all proofs are described in detail but informally. The book teaches the reader the art of precise logical reasoning and the practical use of a proof

assistant as a surgical tool for formal proofs about computer science artefacts. In this sense it represents a formal approach to computer science, not just

semantics. The Isabelle formalisation, including the proofs and accompanying slides, are freely available online, and the book is suitable for graduate students,

advanced undergraduate students, and researchers in theoretical computer science and logic.

Concepts in Programming Languages

Formal Syntax and

Semantics of Programming Languages
Nondeterminism and Recursion
A Laboratory Based Approach
Practical Foundations for

Programming Languages Notes on Operational Semantics of Abstract Data Types and Programming Languages

"Programming languages embody the pragmatics of designing

software systems, and also the mathematical concepts which underlie them. Anyone who wants to know how, for example, object-oriented programming rests upon a firm foundation in logic should read this book. It guides one

surefootedly through the rich variety of basic programming concepts developed over the past forty years." -- Robin Milner, Professor of Computer Science, The Computer Laboratory, Cambridge University "Programming languages need not

be designed in an intellectual vacuum; John Mitchell's book provides an extensive analysis of the fundamental notions underlying programming constructs. A basic grasp of this material is essential for the understanding, comparative

analysis, and design of programming languages." -- Luca Cardelli, Digital Equipment Corporation Written for advanced undergraduate and beginning graduate students, "Foundations for Programming Languages" uses a

series of typed lambda calculi to study the axiomatic, operational, and denotational semantics of sequential programming languages. Later chapters are devoted to progressively more sophisticated type systems.

The Structure of Typed Programming Languages describes the fundamental syntactic and semantic features of modern programming languages, carefully spelling out their impacts on language design. Using classical

and recent research from lambda calculus and type theory, it presents a rational reconstruction of the Algol-like imperative languages such as Pascal, Ada, and Modula-3, and the higher-order functional languages such as

Scheme and ML. David Schmidt's text is based on the premise that although few programmers ever actually design a programming language, it is important for them to understand the structuring techniques. His use of these

techniques in a reconstruction of existing programming languages and in the design of new ones allows programmers and would-be programmers to see why existing languages are structured the way they are and how new languages

can be built using variations on standard themes. The text is unique in its tutorial presentation of higher-order lambda calculus and intuitionistic type theory. The latter in particular reveals that a programming language is a logic in

which its typing system defines the propositions of the logic and its well-typed programs constitute the proofs of the propositions. The Structure of Typed Programming Languages is designed for use in a first or second course on principles

of programming languages. It assumes a basic knowledge of programming languages and mathematics equivalent to a course based on books such as Friedman, Wand, and Haynes': Essentials of Programming Languages. As

Schmidt covers both the syntax and the semantics of programming languages, his text provides a perfect precursor to a more formal presentation of programming language semantics such as Gunter's Semantics of

Programming Languages. Semantics of Programming Languages exposes the basic motivations and philosophy underlying the applications of semantic techniques in computer science. It introduces the

mathematical theory of programming languages with an emphasis on higher-order functions and type systems. Designed as a text for upper-level and graduate-level students, the mathematically sophisticated approach will also

prove useful to professionals who want an easily referenced description of fundamental results and calculi. Basic connections between computational behavior, denotational semantics, and the equational logic of functional

programs are thoroughly and rigorously developed. Topics covered include models of types, operational semantics, category theory, domain theory, fixed point (denotational). semantics, full abstraction and other semantic

correspondence criteria, types and evaluation, type checking and inference, parametric polymorphism, and subtyping. All topics are treated clearly and in depth, with complete proofs for the major results and numerous

exercises.

Action Semantics

Denotational Semantics

An Operational Semantics of And-or-parallel Logic Programming Language, ANDOR-II

Concrete Semantics

Notes on operational semantics of abstract data types and programming languages

Structures and Techniques

***Programming Languages and Operational SemanticsA Concise OverviewSpringer***

***"First book-length exposition of the denotational (or `mathematical' or `functional') approach to the formal semantics of programming languages (in***

***contrast to `operational' and `axiomatic' approaches). Treats various kinds of languages, beginning with the pure-lambda-calculus and progressing through languages with states,***

***commands, jumps, and assignments. This somewhat discursive account is a valuable compilation of results not otherwise available in a single source." -- American Mathematical***

***Monthly***

***This proceedings volume of the 17th European Symposium on Programming examines fundamental issues in the specification, analysis and implementation***

***of programming languages and systems, including static analysis, security, concurrency and program verification.***

***Understanding Programming Languages***

***The Structure of Typed Programming Languages***

***A Descriptive-operational Semantics for Prescribing Programming Languages with "reflective" Capabilities***

***The Formal Semantics of***

***Programming Languages***
***Theories of Programming Languages***
***Reasoning about Programs***
***Using Operational Semantics and the Role of a Proof Support Tool***

A new edition of a textbook that provides students with a deep, working understanding of the essential concepts of programming languages, completely revised, with significant new material. This book provides students with a deep, working understanding of the essential concepts of programming languages. Most of these

essentials relate to the semantics, or meaning, of program elements, and the text uses interpreters (short programs that directly analyze an abstract representation of the program text) to express the semantics of many essential language elements in a way that is both clear and executable. The approach is both analytical and hands-on.

The book provides views of programming languages using widely varying levels of abstraction, maintaining a clear connection between the high-level and low-level views. Exercises are a vital part of the text and are scattered throughout; the text explains the key concepts, and the exercises explore alternative designs and other issues. The

complete Scheme code for all the interpreters and analyzers in the book can be found online through The MIT Press web site. For this new edition, each chapter has been revised and many new exercises have been added. Significant additions have been made to the text, including completely new chapters on modules and continuation-

passing style. Essentials of Programming Languages can be used for both graduate and undergraduate courses, and for continuing education courses for programmers.

A comprehensive introduction to type systems and programming languages. A type system is a syntactic method for

automatically checking the absence of certain erroneous behaviors by classifying program phrases according to the kinds of values they compute. The study of type systems—and of programming languages from a type-theoretic perspective—has important applications in software engineering, language design, high-

performance compilers, and security. This text provides a comprehensive introduction both to type systems in computer science and to the basic theory of programming languages. The approach is pragmatic and operational; each new concept is motivated by programming examples and the more theoretical sections are driven by the needs

of implementations. Each chapter is accompanied by numerous exercises and solutions, as well as a running implementation, available via the Web. Dependencies between chapters are explicitly identified, allowing readers to choose a variety of paths through the material. The core topics include the

untyped lambda-calculus, simple type systems, type reconstruction, universal and existential polymorphism, subtyping, bounded quantification, recursive types, kinds, and type operators. Extended case studies develop a variety of approaches to modeling the features of object-oriented languages.

Formal Syntax and Semantics of Programming Languages: A Laboratory Based Approach presents a panorama of techniques in formal syntax, operational semantics and formal semantics. Using a teaching/learning perspective rather than a research-oriented approach, an understanding of the meta-languages is

accessible to anyone with a basic grounding in discrete mathematics and programming language concepts. Throughout the book, valuable hands-on laboratory exercises provide the opportunity for practical application of difficult concepts. Various exercises and examples, implementing syntactic and semantic specifications on real

systems, give students hands-on practice. Supplemental software is available on disk or via file transfer protocol. This book is suitable for an advanced undergraduate or introductory graduate level course on the formal syntax and semantics of programming languages.

17th European Symposium on

Programming, ESOP 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008, Proceedings

Comparative Metric Semantics of Programming Languages

A Concise Overview

International Summer School, APPSEM 2000, Caminha, Portugal, September 9-15, 2000. Advanced Lectures

On the Relating Denotational and Operational Semantics for Programming Languages with Recursion and Concurrency

An Elementary Introduction Using Structural Operational Semantics

**The design and implementation of programming languages, from Fortran and Cobol to Caml and Java, has been one of the key developments in the management of ever**

**more complex computerized systems. Introduction to the Theory of Programming Languages gives the reader the means to discover the tools to think, design, and implement these languages.**

**It proposes a unified vision of the different formalisms that permit definition of a programming language: small steps operational semantics, big steps operational semantics, and**

**denotational semantics, emphasising that all seek to define a relation between three objects: a program, an input value, and an output value. These formalisms are illustrated by presenting the**

**semantics of some typical features of programming languages: functions, recursivity, assignments, records, objects, ... showing that the study of programming languages**

**does not consist of studying languages one after another, but is organized around the features that are present in these various languages. The study of these features leads to the development of**

**evaluators, interpreters and compilers, and also type inference algorithms, for small languages.**

**Semantics will play an important role in the future development of software**

**systems and domain-specific languages. This book provides a needed introductory presentation of the fundamental ideas behind these approaches, stresses their relationship by**

**formulating and proving the relevant theorems, and illustrates the applications of semantics in computer science. Historically important application areas are presented together with**

**some exciting potential applications. The text investigates the relationship between various methods and describes some of the main ideas used, illustrating these by means of**

**interesting applications. The book provides a rigorous introduction to the main approaches to formal semantics of programming languages.**

**Key ideas in programming**

**language design and implementation explained using a simple and concise framework; a comprehensive introduction suitable for use as a textbook or a reference for researchers. Hundreds of**

**programming languages are in use today—scripting languages for Internet commerce, user interface programming tools, spreadsheet macros, page format specification**

**languages, and many others. Designing a programming language is a metaprogramming activity that bears certain similarities to programming in a regular language, with**

**clarity and simplicity even more important than in ordinary programming. This comprehensive text uses a simple and concise framework to teach key ideas in programming**

**language design and implementation. The book's unique approach is based on a family of syntactically simple pedagogical languages that allow students to explore**

**programming language concepts systematically. It takes as premise and starting point the idea that when language behaviors become incredibly complex, the description of the**

**behaviors must be incredibly simple. The book presents a set of tools (a mathematical metalanguage, abstract syntax, operational and denotational semantics) and uses it to explore a**

**comprehensive set of programming language design dimensions, including dynamic semantics (naming, state, control, data), static semantics (types, type reconstruction,**

**polymporphism, effects), and pragmatics (compilation, garbage collection). The many examples and exercises offer students opportunities to apply the foundational ideas explained**

**in the text. Specialized topics and code that implements many of the algorithms and compilation methods in the book can be found on the book's Web site, along with such**

**additional material as a section on concurrency and proofs of the theorems in the text. The book is suitable as a text for an introductory graduate or advanced undergraduate programming**

**languages course; it can also serve as a reference for researchers and practitioners.**

**Operational Semantics and Proof Theory**

**The Semantics of**

**Programming Languages**
**Types and Programming Languages**
**An Introduction to Structural Operational Semantics**
**Semantics Engineering with**

# PLT Redex

# Programming Languages and Systems

During the last three decades several different styles of semantics for program ming languages have been developed. This book compares two of

them: the operational and the denotational approach. On the basis of several exam ples we show how to define operational and denotational semantic models for programming languages. Furthermore, we introduce a general technique for comparing

various semantic models for a given language. We focus on different degrees of nondeterminism in programming lan guages. Nondeterminism arises naturally in concurrent languages. It is also an important concept in specification

languages. In the examples discussed, the degree of non determinism ranges from a choice between two alternatives to a choice between a collection of alternatives indexed by a closed interval of the real numbers. The former arises in a language with

nondeterministic choices. A real time language with dense choices gives rise to the latter. We also consider the nondeterministic random assignment and parallel composition, both couched in a simple language. Besides non determinism our four example

languages contain some form of recursion, a key ingredient of programming languages.

Abstract: "An operational semantics of and- or-parallel logic programming language, ANDOR-II, is presented. ANDOR-II combines or-parallel

computation of Prolog and and-parallel computation of committed choice logic programming languages such as PARLOG, Concurrent Prolog and Guarded Horn Clauses. Starting from a naive semantics suitable for simulating in sequential machines, we

develop a new semantics with fine grain parallelism. The semantics is based on the coloring scheme which paints variable substitutions made in each or-parallel world by distinct colors."

This text develops a comprehensive

theory of programming languages based on type systems and structural operational semantics. Language concepts are precisely defined by their static and dynamic semantics, presenting the essential tools both intuitively and rigorously while relying

on only elementary mathematics. These tools are used to analyze and prove properties of languages and provide the framework for combining and comparing language features. The broad range of concepts includes fundamental data types such as sums

and products, polymorphic and abstract types, dynamic typing, dynamic dispatch, subtyping and refinement types, symbols and dynamic classification, parallelism and cost semantics, and concurrency and distribution. The methods are directly

applicable to language implementation, to the development of logics for reasoning about programs, and to the formal verification language properties such as type safety. This thoroughly revised second edition includes exercises at the end of nearly

every chapter and a new chapter on type refinements.

The Scott-Strachey Approach to Programming Language Theory

Programming Languages: Principles and Practices

Introduction to the Theory of

Programming Languages
Domains and Lambda-Calculi
Semantics with Applications: An Appetizer
Design Concepts in Programming Languages
*Stump's Programming*

*Language Foundations is a shortconcise text that covers semantics, equally weighting operationaland denotational semantics for several different*

*programmingparadigms: imperative, concurrent, and functional.*

*Programming Language Foundations provides: aneven coverage of denotational,*

*operational an axiomatic semantics;extensions to concurrent and non-deterministic versions;operational semantics for untyped lambda calculus;*

*functionalprogramming; type systems; and coverage of emerging topics andmodern research directions. A homogeneous treatment of the semantics of both*

*theoretical and practical logic programming languages. Action semantics is a novel approach to the formal description of programming languages.*

*Its abstractness is at an intermediate level, between that of denotational and operational semantics. Action semantics has considerable pragmatic*

*advantage over all previous approaches, especially regarding modularity of descriptions. In this volume, Dr. Peter Mosses gives a thorough*

*introduction to action semantics, and provides substantial illustrations of its use.*

*Towards an Operational Semantics for Concurrent*

*Logic Programming Languages With Isabelle/HOL*

*Programming Languages and Operational Semantics*

*Foundations for*

*Programming Languages Handbook of Process Algebra*

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

languages.

This book is about describing the meaning of programming languages. The author teaches the skill of writing semantic descriptions as an efficient way to understand the features of a

language. While a compiler or an interpreter offers a form of formal description of a language, it is not something that can be used as a basis for reasoning about that language nor can it serve as a definition of a programming

language itself since this must allow a range of implementations. By writing a formal semantics of a language a designer can yield a far shorter description and tease out, analyse and record design choices. Early in the book the author

introduces a simple notation, a meta-language, used to record descriptions of the semantics of languages. In a practical approach, he considers dozens of issues that arise in current programming languages and the key techniques

that must be mastered in order to write the required formal semantic descriptions. The book concludes with a discussion of the eight key challenges: delimiting a language (concrete representation), delimiting the abstract content of a

language, recording semantics (deterministic languages), operational semantics (non-determinism), context dependency, modelling sharing, modelling concurrency, and modelling exits. The content is class-tested and

suitable for final-year undergraduate and postgraduate courses. It is also suitable for any designer who wants to understand languages at a deep level. Most chapters offer projects, some of these quite advanced exercises that

ask for complete descriptions of languages, and the book is supported throughout with pointers to further reading and resources. As a prerequisite the reader should know at least one imperative high-level language and

have some knowledge of discrete mathematics notation for logic and set theory.

Process Algebra is a formal description technique for complex computer systems, especially those involving communicating,

concurrently executing components. It is a subject that concurrently touches many topic areas of computer science and discrete math, including system design notations, logic, concurrency theory, specification

and verification, operational semantics, algorithms, complexity theory, and, of course, algebra. This Handbook documents the fate of process algebra since its inception in the late 1970's to the present. It is intended to serve as a reference

source for researchers, students, and system designers and engineers interested in either the theory of process algebra or in learning what process algebra brings to the table as a formal system description and verification

technique. The Handbook is divided into six parts spanning a total of 19 self-contained Chapters. The organization is as follows. Part 1, consisting of four chapters, covers a broad swath of the basic theory of process algebra. Part 2

contains two chapters devoted to the sub-specialization of process algebra known as finite-state processes, while the three chapters of Part 3 look at infinite-state processes, value-passing processes and mobile processes in particular.

Part 4, also three chapters in length, explores several extensions to process algebra including real-time, probability and priority. The four chapters of Part 5 examine non-interleaving process algebras, while Part 6's three chapters

address process-algebra tools and applications.

Executable Operational Semantics of Programming Languages

Concepts Of Programming Languages

Programming Language

## Foundations

Graduate text on mathematical foundations of programming languages, and operational and denotational semantics.

The Formal Semantics of Programming Languages provides the basic mathematical

techniques necessary for those who are beginning a study of the semantics and logics of programming languages. These techniques will allow students to invent, formalize, and justify rules with which to reason about a variety of programming

languages. Although the treatment is elementary, several of the topics covered are drawn from recent research, including the vital area of concurency. The book contains many exercises ranging from simple to miniprojects.Starting with basic

set theory, structural operational semantics is introduced as a way to define the meaning of programming languages along with associated proof techniques. Denotational and axiomatic semantics are illustrated on a simple language of while-

programs, and fall proofs are given of the equivalence of the operational and denotational semantics and soundness and relative completeness of the axiomatic semantics. A proof of Godel's incompleteness theorem, which emphasizes the

impossibility of achieving a fully complete axiomatic semantics, is included. It is supported by an appendix providing an introduction to the theory of computability based on while-programs. Following a presentation of domain theory,

the semantics and methods of proof for several functional languages are treated. The simplest language is that of recursion equations with both call-by-value and call-by-name evaluation. This work is extended to lan guages with higher and

recursive types, including a treatment of the eager and lazy lambda-calculi. Throughout, the relationship between denotational and operational semantics is stressed, and the proofs of the correspondence between the operation and denotational

semantics are provided. The treatment of recursive types - one of the more advanced parts of the book - relies on the use of information systems to represent domains. The book concludes with a chapter on parallel programming languages,

accompanied by a discussion of methods for specifying and verifying nondeterministic and parallel programs.

Kenneth Louden and Kenneth Lambert's new edition of PROGRAMMING LANGUAGES: PRINCIPLES AND PRACTICE, 3E

gives advanced undergraduate students an overview of programming languages through general principles combined with details about many modern languages. Major languages used in this edition include C, C++, Smalltalk, Java, Ada, ML, Haskell,

Scheme, and Prolog; many other languages are discussed more briefly. The text also contains extensive coverage of implementation issues, the theoretical foundations of programming languages, and a large number of exercises,

making it the perfect bridge to compiler courses and to the theoretical study of programming languages. Important Notice: Media content referenced within the product description or the product text may not be available in the ebook version.