Modern compiler design source code

p. 823 + xix, 2012; ISBN 978-1-4614-4698-9, Springer. Foreword and content can be found here. Description The book is designed for students who at least use a compiler and have given some thought to the concept of compilation. It's not an introductory course (though it explains almost everything from the ground up). The book consists conceptually of two parts. The first part concerns the general compilation process and contains three chapters based on the analytical-processing/synthesis paradigm: text analysis, context processing and code generation. The second part consists of four chapters that cover paradigm-specific imperative problems and object-oriented, functional, logical and parallel and distributed programs. These two parts are separated by a chapter on memory management/garbage collection. We, the authors, have tried to write a book in an intuitively appealing style, focusing on the reasoning and mechanics of algorithms, rather than emphasizing strict formulations and formal correctness evidence. Although the book covers most traditional techniques, it makes a number of powerful philosophical and perhaps controversial statements for which we think time has passed: It recognizes lexical analysis, LR analysis and BURS code generation as cases of bottom-up pattern matching and explains them uniformly by using dotting items, thereby unifying three important techniques in compiler design, and allowing students to expand them to suit their needs. A recurring theme is precomputation: first, a simple, understandable and, of course, the correct technique is proposed, then all calculations that can be performed at the time of generating the compiler are performed there. This naturally results from interpretive lexical analysis to FSA and allows us to view the generated code as an instantiation interpreter, thus establishing a connection with partial evaluation. It emphasizes closing algorithms wherever possible, unifying many seemingly different algorithms. It puts compiler construction in a wider file framework and data conversion, allowing the student to see applicability in other programming areas. The upper level structure of the book is: Introduction from the text program to abstract Syntax Tree Anotating Abstract Syntax Tree - Contextual Processing Intermediate Code Memory Management Imperative and Object-Oriented Programs Functional Programs Logical Programs Parallel and Distributed Programs Bibliography Responses to Exercise Special Features are: Techniques for Embedded Systems: Object Code Reduction Energy Reduction Memory Allocation Generalized (Non-Deterministic) LR Analysis, liberating compiler writer from the limitations of LALR Legacy Code Processing: Grammar Recovery Disassembly and/or Decompilation of Legacy Binary Code Optimization Techniques: Procedural Abstraction Binary Code optimal code generation through exhaustive tail recursion Search There is a first edition with translations in French, Spanish and Brazilian/portugese. Dick Grune (retired) taught the principles of programming languages and a construction compiler at Vrije Universiteit Amsterdam. In the 1970s he was involved in the construction of Algol 68 compilers and in the 1980s he was involved in the construction of algol 68 compilers. He co-authored two other books: Parsing Techniques - A Practical Guide - 2nd Ed. and Programming Language Essentials . Kees van Reeuwijk has been involved in programming language research and compilation design for more than 20 years, including hardware description projects, high-performance Fortrana and Java. Henri Bal is a professor at the Faculty of Sciences of the Vrije Universiteit Amsterdam. He is the author of more than 120 articles and (co-) author of two other books: Programming Language Essentials and Programming Distributed Systems. He holds the prestigious NWO Pioneer Prize (Dutch Organisation for Scientific Research) and many other grants. Ceriel Jacobs worked in several construction projects, among which amsterdam compiler Kit and Orca parallel programming system. He is also co-author of the book Parsing Techniques - Practical Guide - 2nd Ed. . Koen Langendoen is a full professor of computer science with Delft University of Technology. He has worked in compiler construction since 1985, and specializes in code generation and runtime support systems for various imperative, functional and parallel languages. No, it's okay. [Kees van Reeuwijk] No, it's okay. [Ceriel Jacobs] [Koen Langendoen] Modern Compiler Design – Second Edition/Dick Grune/dick@dickgrune.com curated a list of amazing resources, teaching materials, tools, frameworks, platforms, technology and source code projects in the field of compilers, interpreters and runtimes. This list has a bias towards education. Content Learning Books General Overview Introductory Basics Compiler Design – Provides a short processing of basic concepts. Beautiful Rocket - How to make your own programming language with a rocket. Build Your Own Lisp – Learn C and Create Your Own Lisp Programming Language in 1000 LoC. Compilers: Principles, Techniques and Tools - Dragons Book. Classic textbook about compiler Construction. Crafting Interpreters – an all-stop-shop for learning (almost) everything you need to learn to build an interpreted, full-featured, effective scripting language. GitHub Repo. Discussions: HN. Create your own Languauge programming – An example-driven approach to building your own programming language with video tutorials and source code projects. Engineering Compiler – Modern textbooks about construction compilers that cover SSA-Form and recent research on code generation. Basics Languages - Basic concepts of programming languages focusing on semantics, interpreting and CPS (Continuing Graduation Style). Language Implementation Patterns – Learn the patterns behind building programming languages and build an interpreter yourself, using an ANTLR. Modern Compiler Implementation in ML – Build a compiler using ML (MetaLanguage) with a textbook that has one of the best coverage on all compiler stages. The book comes with a reference compiler implementation guide to the software development process. Other releases: MCI v C, MCI in Java. Programming language Pragmatics - Integrated treatment of language design and implementation, examples of the function of famous architecture such as ARM and x86 64-bit. Programming Languages: Application and Interpreting - An excellent introduction to a subject that uses an incremental approach to building programs. Errors are also included to highlight key concepts. Programming languages: Theory and Practice - Collected lecture notes for a course in programming languages taught at Carnegie Mellon University, preferably as introductory text on the subject. Project Oberon - Design of the operating system and compiler. Other editions: 2013 edition. BEAM Book - Description ERTS (Erlang Runtime System) and BEAM Virtual Machine. Virtual Machines - A good book on how to build virtual machines specially tailored for the theme of building programming languages. Virtual machines: Universal platforms for systems and processes - A key textbook on virtual machines that explores virtual machine technologies within the disciplines that use them, e.g.: OS and programming languages. Write a compiler in Go – A well-known introduction to the Go programming language and its ecosystem through building a compiler project. Write an interpreter in Go - Successor book Write compiler in Go, but this one builds an interpreter project instead. Writing Compilers and Interpreters: Software Engineering Approach - How to Create Compilers Using Java, this book is tailor-made for Working Software Engineer. Other Releases: Using C++, Using C. Writing Interpreters and Compilers for Raspberry Pi Using Python – If you want to learn how to write interpreters and compilers, while learning how python, python bytecode, mounting language, and dynamic writing work, this is the book for you. Advanced Advanced Compiler Design and Implementation - In-depth treatment of advanced design themes such as: Intermediate Representation, SSA, Code Optimization and various processor architectures. Advanced design and implementation of virtual machines - Step by step hollistic introduction to the design of virtual machine architectures, themes and algorithms. It contains illustrated data and implementations of algorithms in the book. Advanced topics in types and programming - intensive study of type systems covering topics such as, but not limited to: accurate type analyses; Type systems for low-level languages and advanced techniques in ML-style type inference.. Retargetable C Compiler: Design and Implementation - Examines the design and implementation of THE ICC, a production-quality, retargetative compiler, designed in AT&amp;T Bell Labs for the ANSI C programming language. Building Optimization Compiler - fills a gap in code optimization. This book provides a high level of design for thorough optimization, code generator, scheduler and registry allocator for general modern RISC processor. Compilation with sequels - Introduction to CPS (Continuation-Passing Style) as an intermediate representation in the compiler for doing optimization and program transformation. Design concept in programming languages – systematic examination of techniques and ideas used in programming language design covers topics such as: Operational and denotic semantic techniques, dynamic semantic techniques and static semantic techniques. Briefing level parallelism - This book accurately formulates and simplifies the presentation of the technique of compiling the level of instruction (ILP). Linkers and Loaders - Definitive text to compile-time and runtime processes of linking and loading. Optimize compilers for modern architectures - Optimize programming based on recent gains and discoveries in modern high-performance CPU architectures. Techniques of analysis: Practical guide - Definitive guide to analysis of algorithms and techniques, also contains an introduction to formal grammar and analysis of theory. Garbage Collection Handbook: The Art of Automatic Memory Management - a live-classic and definitive text on the subject of Garbage Collection, also includes hardware-based optimization in light of modern advances in CPU architecture. Implementation of functional programming languages - Classic textbook on the implementation of functional languages, includes structured types, pattern corresponding semantics, Lambda Calculus transformation, polymorphic type control and many other topics. SSA Book - The only in-depth study of SSA-form (Static single assignment form) in book format. Types and programming languages - A comprehensive introduction to the theme Type systems and programming languages from a type-theoretical perspective. Warren's Abstract Machine: Prolog in Haskell – Introduction to WAM from Logical Programming in Prolog. Papers A Brief History of JIT Compilation, by J. Aycock. Flexible prologue interpreter in Python, C. Bolz &amp; M. Leuschel. Chart Higher-Order IR, R. Leißa, M. Koster &amp; S. Hack. Micro-Manual for LISP - Not the whole truth, J. McCarthy. Other releases: Clean and nice typeset. Prologue interpreter in Python, C. Bolz. Simple multi-processor computer based on Subleq, O. Mazonka, A. Kolodin. In 20 Copy: How the Loader Betrays You - About security issues related to dynamic loading. Incremental approach to compiler construction. Compiler Building System Usage, E. Hilsdale, J. Ashley, R. Dybvig &amp; D. Friedman. Compilation with sequel: Sequel, A. Kennedy. Definition interpreters for higher order programming languages, J. Reynolds. Drainage Swamp: Micro Virtual Machines like Solid Foundation for Languauage Development, K. Wang, Y. Lin, S. Blackburn, M. Norrish &amp; A. Hosking. Engineering officers, J. Midtgaard, N. Ramsey, B. Larsen. Garbage collection in a non-cooperating environment, H. Boehm, M. Weiser. Obfuscating machine code by reducing the instruction set and linearization cfg, C. Jonischkeit. The IOC is Turing-Complete, S. Dolan. Nanopass Framework for Commercial Compiler Development, A. Keep &amp; R. Dybvig. Nanopass Framework for Compiler Education, D. Sarkar, O. Waddell &amp; R. Dybvig. Chart Notes Algorithms used in optimizing compilers, C. Offner. Packrat Analysis work on PEG, B. Ford. Peg-based transformer provides front-end, mid-end and back-end phases in a simple compiler, I. Piumarta. Pycket: JIT tracking for functional language. PyPy's Approach to VM Construction, A. Rigo &amp; S. Pedroni. RABBIT: Compiler for SCHEME, G. Steele. Simple and efficient design of the SSA form. SSA-based Registry Allocation, S. Hack &amp; G. Goos. The Essence of Compiling with Continuations, C. Flanagan, A. Sabry, B. Oak &amp; M. Felleisen. Page-Mistakes Weird Machine: Lessons in Teaching-Less Calculation, by J. Bangert, S. Bratus, R. Shapiro, S. Smith. Trace-based JIT Compilation for Lazy Functional Languages, T. Schilling. Use datalog with binary decision diagrams for program analysis, J. Whaley, D. Avots, M. Carbine &amp; M. Lam. Researchers and institutes CLI specifications (ECMA-335) Specification. Mu Specification. the JVM SE8 specification. Courses Compilers Construction, Cambridge - Introduction to the compiler building course from the University of Cambridge. Compiler Construction for Undergrads, RICE University – Introduction to compiler construction and language translators course from RICE University. Compilers Theory, Stanford - YouTube, Stanford.edu, Class Notes-Introduction to Compilers Theory and Building Course from Stanford. Design and construction of compilers, University of Texas - Design and construction of compilers including lexical analysis, analysis, code generation techniques, error analysis and simple code optimizations. Lecture notes: PDF, HTML. DSL Design and Implementation Summer School – Summer School program on DSL Design themes and implementation hosted by EPFL University. Basics of programming languages - Concepts underlying the design, definition, implementation and use of modern programming languages from a formal point of view. Nand2Tetris: How to create a computer from Principles, Part 2 – This second part of the Nand2Tetris course includes basic language design and basic compiler building concepts among many other topics at the basic level. NPTEL Principles Compiler Design Course - Introductory Course from NPTEL to Compiler Design. NPTEL Compiler Design Course – A slightly more advanced course than their Principles Compiler Design course, includes the SSA Form to a good degree. Programming Languages: Part A by Grossman – Part 1 of the 3-part course series on the basic concepts of programming languages, with a strong emphasis on functional programming. Programming Languages: Part B by Grossman – Part 2 of the 3-part course series on basic concepts of programming languages, with a strong emphasis on functional programming. Programming Languages: Part C by Grossman – Part 3 of the 3-part course series on basic concepts of programming languages, with a strong emphasis on functional programming. Types, logic, semantics, and validation from Oregon University's Summer School – Summer School program, which consists of 80 minute lectures presented by internationally recognized leaders in programming languages and formal reasoning research. Virtual Machines and Controlled Runtimes, UCB CS294 - Introductory course on virtual machines and controlled runtimes from the University of Berkeley. Virtual Machines Summer School 2016 (VMSS 2016) - VMSS is a summer school program that aims to provide an overview of this field, focused on early career researchers. Interviews and Conference Channels Videos Articles Instructions Implementing Dependents written by Lambda Calculus. Beginner's Guide to Linkers – A guide to help C&amp;C++ programmers understand the basics of what a linker does. Algorithm W step by step. Building lisp from scratch with Swift. Compiler Optmization tutorial. Hindley-Damas-Milner Teaching – Extensively documented instructions for type checking basic functional language using the Hindley-Damas-Milner algorithm. How I wrote the programming language, and how can you too. Implementation of JIT compiled language with Haskell and LLVM. Kaleidoscope: Implementation of language with LLVM in the Target Caml. Let's build a simple interpreter. Lisperator - How to implement a programming language in JavaScript. Small Translator Lisp - An interpreter that supports invoking functions, lambdas, allows, ifs, numbers, strings, several library functions and lists within 120 lines of JavaScript. lis.py, v1: (How to Write (Lisp) Interpreter (in Python)) - Teaching Peter Norvig on Writing a Simple Lisp Interpreter. lis.py, v2: ((Even better) Lisp) Translator (in Python) - Follow-up tutorial by Peter Norvig on lis.py a little better. LLVM Tutorial: Implementation of kaleidoscope. Python version with LLVMPY. Metacompile course, Part 1. Project: Programming Language - Chapter 11 of the book JavaScript, 2. Email you Haskell. Writing language in Truffel - Interpreter development tutorial using Truffel, Cristian Esquivias. Community Discussion Tools and Frameworks Language Agnostic B3: Bare Bones Backend – WebKit is an optimization JIT compiler for procedures containing C-like code. Capstone - Lightweight multi-platform, multi-architecture dismantling framework with links to various famous programming languages. Keystone - Lightweight multi-platform, multi-architecture assembler framework with links to various famous programming languages. LLILCL - LLVM-based backend compiler for .NET Core. LLVM - LLVM compiler-backend framework. MicroVM - Mu Framewrok for the development of programming languages based on the MuVM specification. Movfuscator Compiler - M / o / Vfuscator compiles programs into mov instructions, and only mov instructions. QBE: Quick Backend - Pure C insert SSA-based backend compiler. Rubinius - Language Programming Development Platform. Summus - Basic, reusable, compiler-frontend implementation using LLVM as a backend. ZetaVM - Multi-language platform for dynamic programming languages. C/C++ AsmJIT - Complete x86/x64 JIT and remote assembler for C++. GCC - GNU compiler collection (C as backend). LCC - LCC retargetable ANSI C compiler (C as backend). libFirm - C-library that provides chart-based medium representation, optimization and mounting code generation suitable for use in compilers. libJIT - Library for general Just-In-Time compiler functions independent of a particular bytecode, language, or runtime. myJIT - Library for generating and running machine code at running time. OrangeC - Win32 C11/C++2014 compiler and toolkit. PCC - Portable C compiler (C as backend). Ragel - Ragel state machine compiler. TCC - Tiny C compiler (C as backend). CLR Cecil - Library for generating and checking programs and libraries in ECMA CIL. DotNetPELib format - Library to read and write .net reports in C++11 ILSpy - .NET Decompiler. Spotlight - .NET Decompiler. Reflexil - Mounting a code editor that can be used as a plugin with other .NET/CLR tools. D dunnart - LALR(1) Parser Generator. FancyPars-lite - Fast parser generator. libdparse - Library allows you to create lexers and parsers. Contains lexer and analyzer for D. llvm-d-D binding for LLVM itself. Suspended - design, test and generate parsers for PEG grammar. Go goyacc - YACC Implementation in Go. Standard LALR analyser generator [1]. LLVM Go Binding - Official Go LLVM Binding. Graal Graal - High-Performance Polyglot Runtime. Graal Core - compiler and truffle partial evaluator. Graal VM – Graal is a multi-language VM distribution. Haskell JavaScript IRHudra - A tool for displaying intermediates used by V8 and Dart VM optimizers. JISON - Contextually free grammatical parser for JavaScript. GerHobbelt / jison - an active fork jison with a bunch of improvements. Nearley - Simple, fast, powerful parser toolkit for JavaScript. Ohm - Library and language for building parsers, interpreters, compilers, etc. PEG.js - Simple parser generator for JavaScript. JVM ANTLR - Parser generator for reading, processing, performing or translating structured text or binaries. BYAAC/J - BYACC/Java is an extension of Berkeley's 1.8 YACC-compatible parser generator for Java. CGLIB - High-level API library for generating and transforming Java Byte code. FCP JVM - JVM Backend for generating Java Byte code that matches JDK v1.5+ specifications and Dalvik VM. JavaCC - Java Compiler Building and Parser Generator Toolkit. JavaCPP Preferences for LLVM - Library for easy interaction with LLVM APIs. JFlex - JFlex is a lexical analyzer generator for Java with full Unicode support. JLex – JLex is a lexical generator analyzer that can be used in combination with CUP. Kotlin Python AST - Python is a built-in abstract Syntax Tree package. Dis - Python is a built-in Disassembler package. Analysis - Pure-Python module that implements LR(1) parser generator as well as CFSM and GLR parser drivers. PLY - Implementation of lex and yacc parsing tools for Python. PyParsing – An alternative approach to creating and performing simple grammar, vs. traditional lex/yacc approach, or using regular expressions. RPLY - Port project PLY on RPython. RPython - RPython is a framework for the implementation of dynamic languages. Lists python parsing tools rust combine - Parser Combinator Library for Rust. IronLLVM - Secure LLVM ties for Rust. LALRPOP - LR (1) parser generator for Rust. Nom - Parser Combinator Framework. PEG - PEG Parser Generator. Pest - PEG Parser generator. RLS - Rust Language Server Implementation (a.ka RLS). Compilers and interpreters This section focuses on a list of code projects compilers, interpreters, translators, runtimes, virtual machines and so on. Serious projects educational and playing projects Akilang - compiler for simple language, built with Python and LLVM amacc - Small C Compiler generating ELF executable for Arm architecture. Black - Scheme interpreter for black reflective programming language, kenichi Asai's. Other sources: GitHub Mirror. C4 - C Lang in 4 functions. CarpVM - Experimental VM Implementation in C. Charly - Interpreter for dynamically written language written in Crystal. Dale - Lisp-flavored C: System programming language. EasyLang - Simple Programming Language / VM. Eschelle - Open source cross platform multi-paradigm language with VM &amp; JIT Gecho - Simple-stack language implementation in C. gocaml - Minimum functional programming language implementation in Go and LLVM. gone - Compiler for a small programming language called Gone, implemented using Python 3.6, SLY and Developed as part of the January 2018 Write Compiler course, overseen by David Beazly. Hython - Haskell-powered Python 3 interpreter. llgo - Go frontend for LLVM written in Go. MAL: Make Lisp – A Clojure-inspired Lisp interpreter performed in 64 languages. MetaScala - Metacirculatory JVM implementation in Scala. mini-js - Experimental self-hosted JavaScript compiler in 1K LoC. MunVM - Lua VM &amp; Compiler in C. MY-BASIC - insertable basic dialect interpreter in C with modern paradigms. oberonc - one pass, self-hosting compiler for Oberon-07 programming language. It focuses on the JVM. Poprc - Compiler for Language Popr. PyCOOLC - Compiler for COOL programming language written in Python 3. RabbitVM - RISC-based VM implementation in C. StackVM - Virtual machine with integrated VRAM display. stack_cpu - Stack-machine simulator. Super Tiny Compiler - Tiny Educational Compiler Project in JavaScript. tinyc.c - Tiny-C Language Compiler in C. tisp - Time is Space Programming Language Translator. Ultra Tiny Compiler - Another small compiler in less than 90 lines of code. Runtimes and VM Blogs Communities/r/Compilers – Subreddit communities on the theory and development of compilers. /r/ProgrammingLanguages - Subreddit community dedicated to discussing programming languages, programming language theory, design, their syntax and compilers. Vertical License Contributors To the extent possible under the law, Ahmad Alhour has waived all copyright and related or adjacent rights to this work. The logo was designed using TextCraft. TextCraft.