

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Language Processors . . . . .	1
1.1.1	Exercises for Section 1.1 . . . . .	3
1.2	The Structure of a Compiler . . . . .	4
1.2.1	Lexical Analysis . . . . .	5
1.2.2	Syntax Analysis . . . . .	8
1.2.3	Semantic Analysis . . . . .	8
1.2.4	Intermediate Code Generation . . . . .	9
1.2.5	Code Optimization . . . . .	10
1.2.6	Code Generation . . . . .	10
1.2.7	Symbol-Table Management . . . . .	11
1.2.8	The Grouping of Phases into Passes . . . . .	11
1.2.9	Compiler-Construction Tools . . . . .	12
1.3	The Evolution of Programming Languages . . . . .	12
1.3.1	The Move to Higher-level Languages . . . . .	13
1.3.2	Impacts on Compilers . . . . .	14
1.3.3	Exercises for Section 1.3 . . . . .	14
1.4	The Science of Building a Compiler . . . . .	15
1.4.1	Modeling in Compiler Design and Implementation . . . . .	15
1.4.2	The Science of Code Optimization . . . . .	15
1.5	Applications of Compiler Technology . . . . .	17
1.5.1	Implementation of High-Level Programming Languages . . . . .	17
1.5.2	Optimizations for Computer Architectures . . . . .	19
1.5.3	Design of New Computer Architectures . . . . .	21
1.5.4	Program Translations . . . . .	22
1.5.5	Software Productivity Tools . . . . .	23
1.6	Programming Language Basics . . . . .	25
1.6.1	The Static/Dynamic Distinction . . . . .	25
1.6.2	Environments and States . . . . .	26
1.6.3	Static Scope and Block Structure . . . . .	28
1.6.4	Explicit Access Control . . . . .	31
1.6.5	Dynamic Scope . . . . .	31
1.6.6	Parameter Passing Mechanisms . . . . .	33

1.6.7	Aliasing . . . . .	35
1.6.8	Exercises for Section 1.6 . . . . .	35
1.7	Summary of Chapter 1 . . . . .	36
1.8	References for Chapter 1 . . . . .	38
<b>2</b>	<b>A Simple Syntax-Directed Translator</b>	<b>39</b>
2.1	Introduction . . . . .	40
2.2	Syntax Definition . . . . .	42
2.2.1	Definition of Grammars . . . . .	42
2.2.2	Derivations . . . . .	44
2.2.3	Parse Trees . . . . .	45
2.2.4	Ambiguity . . . . .	47
2.2.5	Associativity of Operators . . . . .	48
2.2.6	Precedence of Operators . . . . .	48
2.2.7	Exercises for Section 2.2 . . . . .	51
2.3	Syntax-Directed Translation . . . . .	52
2.3.1	Postfix Notation . . . . .	53
2.3.2	Synthesized Attributes . . . . .	54
2.3.3	Simple Syntax-Directed Definitions . . . . .	56
2.3.4	Tree Traversals . . . . .	56
2.3.5	Translation Schemes . . . . .	57
2.3.6	Exercises for Section 2.3 . . . . .	60
2.4	Parsing . . . . .	60
2.4.1	Top-Down Parsing . . . . .	61
2.4.2	Predictive Parsing . . . . .	64
2.4.3	When to Use $\epsilon$ -Productions . . . . .	65
2.4.4	Designing a Predictive Parser . . . . .	66
2.4.5	Left Recursion . . . . .	67
2.4.6	Exercises for Section 2.4 . . . . .	68
2.5	A Translator for Simple Expressions . . . . .	68
2.5.1	Abstract and Concrete Syntax . . . . .	69
2.5.2	Adapting the Translation Scheme . . . . .	70
2.5.3	Procedures for the Nonterminals . . . . .	72
2.5.4	Simplifying the Translator . . . . .	73
2.5.5	The Complete Program . . . . .	74
2.6	Lexical Analysis . . . . .	76
2.6.1	Removal of White Space and Comments . . . . .	77
2.6.2	Reading Ahead . . . . .	78
2.6.3	Constants . . . . .	78
2.6.4	Recognizing Keywords and Identifiers . . . . .	79
2.6.5	A Lexical Analyzer . . . . .	81
2.6.6	Exercises for Section 2.6 . . . . .	84
2.7	Symbol Tables . . . . .	85
2.7.1	Symbol Table Per Scope . . . . .	86
2.7.2	The Use of Symbol Tables . . . . .	89

2.8	Intermediate Code Generation . . . . .	91
2.8.1	Two Kinds of Intermediate Representations . . . . .	91
2.8.2	Construction of Syntax Trees . . . . .	92
2.8.3	Static Checking . . . . .	97
2.8.4	Three-Address Code . . . . .	99
2.8.5	Exercises for Section 2.8 . . . . .	105
2.9	Summary of Chapter 2 . . . . .	105
<b>3</b>	<b>Lexical Analysis</b>	<b>109</b>
3.1	The Role of the Lexical Analyzer . . . . .	109
3.1.1	Lexical Analysis Versus Parsing . . . . .	110
3.1.2	Tokens, Patterns, and Lexemes . . . . .	111
3.1.3	Attributes for Tokens . . . . .	112
3.1.4	Lexical Errors . . . . .	113
3.1.5	Exercises for Section 3.1 . . . . .	114
3.2	Input Buffering . . . . .	115
3.2.1	Buffer Pairs . . . . .	115
3.2.2	Sentinels . . . . .	116
3.3	Specification of Tokens . . . . .	116
3.3.1	Strings and Languages . . . . .	117
3.3.2	Operations on Languages . . . . .	119
3.3.3	Regular Expressions . . . . .	120
3.3.4	Regular Definitions . . . . .	123
3.3.5	Extensions of Regular Expressions . . . . .	124
3.3.6	Exercises for Section 3.3 . . . . .	125
3.4	Recognition of Tokens . . . . .	128
3.4.1	Transition Diagrams . . . . .	130
3.4.2	Recognition of Reserved Words and Identifiers . . . . .	132
3.4.3	Completion of the Running Example . . . . .	133
3.4.4	Architecture of a Transition-Diagram-Based Lexical Analyzer . . . . .	134
3.4.5	Exercises for Section 3.4 . . . . .	136
3.5	The Lexical-Analyzer Generator <i>Lex</i> . . . . .	140
3.5.1	Use of <i>Lex</i> . . . . .	140
3.5.2	Structure of <i>Lex</i> Programs . . . . .	141
3.5.3	Conflict Resolution in <i>Lex</i> . . . . .	144
3.5.4	The Lookahead Operator . . . . .	144
3.5.5	Exercises for Section 3.5 . . . . .	146
3.6	Finite Automata . . . . .	147
3.6.1	Nondeterministic Finite Automata . . . . .	147
3.6.2	Transition Tables . . . . .	148
3.6.3	Acceptance of Input Strings by Automata . . . . .	149
3.6.4	Deterministic Finite Automata . . . . .	149
3.6.5	Exercises for Section 3.6 . . . . .	151
3.7	From Regular Expressions to Automata . . . . .	152

3.7.1	Conversion of an NFA to a DFA . . . . .	152
3.7.2	Simulation of an NFA . . . . .	156
3.7.3	Efficiency of NFA Simulation . . . . .	157
3.7.4	Construction of an NFA from a Regular Expression . . . . .	159
3.7.5	Efficiency of String-Processing Algorithms . . . . .	163
3.7.6	Exercises for Section 3.7 . . . . .	166
3.8	Design of a Lexical-Analyzer Generator . . . . .	166
3.8.1	The Structure of the Generated Analyzer . . . . .	167
3.8.2	Pattern Matching Based on NFA's . . . . .	168
3.8.3	DFA's for Lexical Analyzers . . . . .	170
3.8.4	Implementing the Lookahead Operator . . . . .	171
3.8.5	Exercises for Section 3.8 . . . . .	172
3.9	Optimization of DFA-Based Pattern Matchers . . . . .	173
3.9.1	Important States of an NFA . . . . .	173
3.9.2	Functions Computed From the Syntax Tree . . . . .	175
3.9.3	Computing <i>nullable</i> , <i>firstpos</i> , and <i>lastpos</i> . . . . .	176
3.9.4	Computing <i>followpos</i> . . . . .	177
3.9.5	Converting a Regular Expression Directly to a DFA . . . . .	179
3.9.6	Minimizing the Number of States of a DFA . . . . .	180
3.9.7	State Minimization in Lexical Analyzers . . . . .	184
3.9.8	Trading Time for Space in DFA Simulation . . . . .	185
3.9.9	Exercises for Section 3.9 . . . . .	186
3.10	Summary of Chapter 3 . . . . .	187
3.11	References for Chapter 3 . . . . .	189
<b>4</b>	<b>Syntax Analysis</b> . . . . .	<b>191</b>
4.1	Introduction . . . . .	192
4.1.1	The Role of the Parser . . . . .	192
4.1.2	Representative Grammars . . . . .	193
4.1.3	Syntax Error Handling . . . . .	194
4.1.4	Error-Recovery Strategies . . . . .	195
4.2	Context-Free Grammars . . . . .	197
4.2.1	The Formal Definition of a Context-Free Grammar . . . . .	197
4.2.2	Notational Conventions . . . . .	198
4.2.3	Derivations . . . . .	199
4.2.4	Parse Trees and Derivations . . . . .	201
4.2.5	Ambiguity . . . . .	203
4.2.6	Verifying the Language Generated by a Grammar . . . . .	204
4.2.7	Context-Free Grammars Versus Regular Expressions . . . . .	205
4.2.8	Exercises for Section 4.2 . . . . .	206
4.3	Writing a Grammar . . . . .	209
4.3.1	Lexical Versus Syntactic Analysis . . . . .	209
4.3.2	Eliminating Ambiguity . . . . .	210
4.3.3	Elimination of Left Recursion . . . . .	212
4.3.4	Left Factoring . . . . .	214

4.3.5	Non-Context-Free Language Constructs . . . . .	215
4.3.6	Exercises for Section 4.3 . . . . .	216
4.4	Top-Down Parsing . . . . .	217
4.4.1	Recursive-Descent Parsing . . . . .	219
4.4.2	FIRST and FOLLOW . . . . .	220
4.4.3	LL(1) Grammars . . . . .	222
4.4.4	Nonrecursive Predictive Parsing . . . . .	226
4.4.5	Error Recovery in Predictive Parsing . . . . .	228
4.4.6	Exercises for Section 4.4 . . . . .	231
4.5	Bottom-Up Parsing . . . . .	233
4.5.1	Reductions . . . . .	234
4.5.2	Handle Pruning . . . . .	235
4.5.3	Shift-Reduce Parsing . . . . .	236
4.5.4	Conflicts During Shift-Reduce Parsing . . . . .	238
4.5.5	Exercises for Section 4.5 . . . . .	240
4.6	Introduction to LR Parsing: Simple LR . . . . .	241
4.6.1	Why LR Parsers? . . . . .	241
4.6.2	Items and the LR(0) Automaton . . . . .	242
4.6.3	The LR-Parsing Algorithm . . . . .	248
4.6.4	Constructing SLR-Parsing Tables . . . . .	252
4.6.5	Viable Prefixes . . . . .	256
4.6.6	Exercises for Section 4.6 . . . . .	257
4.7	More Powerful LR Parsers . . . . .	259
4.7.1	Canonical LR(1) Items . . . . .	260
4.7.2	Constructing LR(1) Sets of Items . . . . .	261
4.7.3	Canonical LR(1) Parsing Tables . . . . .	265
4.7.4	Constructing LALR Parsing Tables . . . . .	266
4.7.5	Efficient Construction of LALR Parsing Tables . . . . .	270
4.7.6	Compaction of LR Parsing Tables . . . . .	275
4.7.7	Exercises for Section 4.7 . . . . .	277
4.8	Using Ambiguous Grammars . . . . .	278
4.8.1	Precedence and Associativity to Resolve Conflicts . . . . .	279
4.8.2	The “Dangling-Else” Ambiguity . . . . .	281
4.8.3	Error Recovery in LR Parsing . . . . .	283
4.8.4	Exercises for Section 4.8 . . . . .	285
4.9	Parser Generators . . . . .	287
4.9.1	The Parser Generator Yacc . . . . .	287
4.9.2	Using Yacc with Ambiguous Grammars . . . . .	291
4.9.3	Creating Yacc Lexical Analyzers with Lex . . . . .	294
4.9.4	Error Recovery in Yacc . . . . .	295
4.9.5	Exercises for Section 4.9 . . . . .	297
4.10	Summary of Chapter 4 . . . . .	297
4.11	References for Chapter 4 . . . . .	300

<b>5</b>	<b>Syntax-Directed Translation</b>	<b>303</b>
5.1	Syntax-Directed Definitions . . . . .	304
5.1.1	Inherited and Synthesized Attributes . . . . .	304
5.1.2	Evaluating an SDD at the Nodes of a Parse Tree . . . . .	306
5.1.3	Exercises for Section 5.1 . . . . .	309
5.2	Evaluation Orders for SDD's . . . . .	310
5.2.1	Dependency Graphs . . . . .	310
5.2.2	Ordering the Evaluation of Attributes . . . . .	312
5.2.3	S-Attributed Definitions . . . . .	312
5.2.4	L-Attributed Definitions . . . . .	313
5.2.5	Semantic Rules with Controlled Side Effects . . . . .	314
5.2.6	Exercises for Section 5.2 . . . . .	317
5.3	Applications of Syntax-Directed Translation . . . . .	318
5.3.1	Construction of Syntax Trees . . . . .	318
5.3.2	The Structure of a Type . . . . .	321
5.3.3	Exercises for Section 5.3 . . . . .	323
5.4	Syntax-Directed Translation Schemes . . . . .	324
5.4.1	Postfix Translation Schemes . . . . .	324
5.4.2	Parser-Stack Implementation of Postfix SDT's . . . . .	325
5.4.3	SDT's With Actions Inside Productions . . . . .	327
5.4.4	Eliminating Left Recursion From SDT's . . . . .	328
5.4.5	SDT's for L-Attributed Definitions . . . . .	331
5.4.6	Exercises for Section 5.4 . . . . .	336
5.5	Implementing L-Attributed SDD's . . . . .	337
5.5.1	Translation During Recursive-Descent Parsing . . . . .	338
5.5.2	On-The-Fly Code Generation . . . . .	340
5.5.3	L-Attributed SDD's and LL Parsing . . . . .	343
5.5.4	Bottom-Up Parsing of L-Attributed SDD's . . . . .	348
5.5.5	Exercises for Section 5.5 . . . . .	352
5.6	Summary of Chapter 5 . . . . .	353
5.7	References for Chapter 5 . . . . .	354
<b>6</b>	<b>Intermediate-Code Generation</b>	<b>357</b>
6.1	Variants of Syntax Trees . . . . .	358
6.1.1	Directed Acyclic Graphs for Expressions . . . . .	359
6.1.2	The Value-Number Method for Constructing DAG's . . . . .	360
6.1.3	Exercises for Section 6.1 . . . . .	362
6.2	Three-Address Code . . . . .	363
6.2.1	Addresses and Instructions . . . . .	364
6.2.2	Quadruples . . . . .	366
6.2.3	Triples . . . . .	367
6.2.4	Static Single-Assignment Form . . . . .	369
6.2.5	Exercises for Section 6.2 . . . . .	370
6.3	Types and Declarations . . . . .	370
6.3.1	Type Expressions . . . . .	371

6.3.2	Type Equivalence . . . . .	372
6.3.3	Declarations . . . . .	373
6.3.4	Storage Layout for Local Names . . . . .	373
6.3.5	Sequences of Declarations . . . . .	376
6.3.6	Fields in Records and Classes . . . . .	376
6.3.7	Exercises for Section 6.3 . . . . .	378
6.4	Translation of Expressions . . . . .	378
6.4.1	Operations Within Expressions . . . . .	378
6.4.2	Incremental Translation . . . . .	380
6.4.3	Addressing Array Elements . . . . .	381
6.4.4	Translation of Array References . . . . .	383
6.4.5	Exercises for Section 6.4 . . . . .	384
6.5	Type Checking . . . . .	386
6.5.1	Rules for Type Checking . . . . .	387
6.5.2	Type Conversions . . . . .	388
6.5.3	Overloading of Functions and Operators . . . . .	390
6.5.4	Type Inference and Polymorphic Functions . . . . .	391
6.5.5	An Algorithm for Unification . . . . .	395
6.5.6	Exercises for Section 6.5 . . . . .	398
6.6	Control Flow . . . . .	399
6.6.1	Boolean Expressions . . . . .	399
6.6.2	Short-Circuit Code . . . . .	400
6.6.3	Flow-of-Control Statements . . . . .	401
6.6.4	Control-Flow Translation of Boolean Expressions . . . . .	403
6.6.5	Avoiding Redundant Gotos . . . . .	405
6.6.6	Boolean Values and Jumping Code . . . . .	408
6.6.7	Exercises for Section 6.6 . . . . .	408
6.7	Backpatching . . . . .	410
6.7.1	One-Pass Code Generation Using Backpatching . . . . .	410
6.7.2	Backpatching for Boolean Expressions . . . . .	411
6.7.3	Flow-of-Control Statements . . . . .	413
6.7.4	Break-, Continue-, and Goto-Statements . . . . .	416
6.7.5	Exercises for Section 6.7 . . . . .	417
6.8	Switch-Statements . . . . .	418
6.8.1	Translation of Switch-Statements . . . . .	419
6.8.2	Syntax-Directed Translation of Switch-Statements . . . . .	420
6.8.3	Exercises for Section 6.8 . . . . .	421
6.9	Intermediate Code for Procedures . . . . .	422
6.10	Summary of Chapter 6 . . . . .	424
6.11	References for Chapter 6 . . . . .	425

<b>7</b>	<b>Run-Time Environments</b>	<b>427</b>
7.1	Storage Organization . . . . .	427
7.1.1	Static Versus Dynamic Storage Allocation . . . . .	429
7.2	Stack Allocation of Space . . . . .	430
7.2.1	Activation Trees . . . . .	430
7.2.2	Activation Records . . . . .	433
7.2.3	Calling Sequences . . . . .	436
7.2.4	Variable-Length Data on the Stack . . . . .	438
7.2.5	Exercises for Section 7.2 . . . . .	440
7.3	Access to Nonlocal Data on the Stack . . . . .	441
7.3.1	Data Access Without Nested Procedures . . . . .	442
7.3.2	Issues With Nested Procedures . . . . .	442
7.3.3	A Language With Nested Procedure Declarations . . . . .	443
7.3.4	Nesting Depth . . . . .	443
7.3.5	Access Links . . . . .	445
7.3.6	Manipulating Access Links . . . . .	447
7.3.7	Access Links for Procedure Parameters . . . . .	448
7.3.8	Displays . . . . .	449
7.3.9	Exercises for Section 7.3 . . . . .	451
7.4	Heap Management . . . . .	452
7.4.1	The Memory Manager . . . . .	453
7.4.2	The Memory Hierarchy of a Computer . . . . .	454
7.4.3	Locality in Programs . . . . .	455
7.4.4	Reducing Fragmentation . . . . .	457
7.4.5	Manual Deallocation Requests . . . . .	460
7.4.6	Exercises for Section 7.4 . . . . .	463
7.5	Introduction to Garbage Collection . . . . .	463
7.5.1	Design Goals for Garbage Collectors . . . . .	464
7.5.2	Reachability . . . . .	466
7.5.3	Reference Counting Garbage Collectors . . . . .	468
7.5.4	Exercises for Section 7.5 . . . . .	470
7.6	Introduction to Trace-Based Collection . . . . .	470
7.6.1	A Basic Mark-and-Sweep Collector . . . . .	471
7.6.2	Basic Abstraction . . . . .	473
7.6.3	Optimizing Mark-and-Sweep . . . . .	475
7.6.4	Mark-and-Compact Garbage Collectors . . . . .	476
7.6.5	Copying collectors . . . . .	478
7.6.6	Comparing Costs . . . . .	482
7.6.7	Exercises for Section 7.6 . . . . .	482
7.7	Short-Pause Garbage Collection . . . . .	483
7.7.1	Incremental Garbage Collection . . . . .	483
7.7.2	Incremental Reachability Analysis . . . . .	485
7.7.3	Partial-Collection Basics . . . . .	487
7.7.4	Generational Garbage Collection . . . . .	488
7.7.5	The Train Algorithm . . . . .	490



7.7.6	Exercises for Section 7.7 . . . . .	493
7.8	Advanced Topics in Garbage Collection . . . . .	494
7.8.1	Parallel and Concurrent Garbage Collection . . . . .	495
7.8.2	Partial Object Relocation . . . . .	497
7.8.3	Conservative Collection for Unsafe Languages . . . . .	498
7.8.4	Weak References . . . . .	498
7.8.5	Exercises for Section 7.8 . . . . .	499
7.9	Summary of Chapter 7 . . . . .	500
7.10	References for Chapter 7 . . . . .	502
<b>8</b>	<b>Code Generation</b> . . . . .	<b>505</b>
8.1	Issues in the Design of a Code Generator . . . . .	506
8.1.1	Input to the Code Generator . . . . .	507
8.1.2	The Target Program . . . . .	507
8.1.3	Instruction Selection . . . . .	508
8.1.4	Register Allocation . . . . .	510
8.1.5	Evaluation Order . . . . .	511
8.2	The Target Language . . . . .	512
8.2.1	A Simple Target Machine Model . . . . .	512
8.2.2	Program and Instruction Costs . . . . .	515
8.2.3	Exercises for Section 8.2 . . . . .	516
8.3	Addresses in the Target Code . . . . .	518
8.3.1	Static Allocation . . . . .	518
8.3.2	Stack Allocation . . . . .	520
8.3.3	Run-Time Addresses for Names . . . . .	522
8.3.4	Exercises for Section 8.3 . . . . .	524
8.4	Basic Blocks and Flow Graphs . . . . .	525
8.4.1	Basic Blocks . . . . .	526
8.4.2	Next-Use Information . . . . .	528
8.4.3	Flow Graphs . . . . .	529
8.4.4	Representation of Flow Graphs . . . . .	530
8.4.5	Loops . . . . .	531
8.4.6	Exercises for Section 8.4 . . . . .	531
8.5	Optimization of Basic Blocks . . . . .	533
8.5.1	The DAG Representation of Basic Blocks . . . . .	533
8.5.2	Finding Local Common Subexpressions . . . . .	534
8.5.3	Dead Code Elimination . . . . .	535
8.5.4	The Use of Algebraic Identities . . . . .	536
8.5.5	Representation of Array References . . . . .	537
8.5.6	Pointer Assignments and Procedure Calls . . . . .	539
8.5.7	Reassembling Basic Blocks From DAG's . . . . .	539
8.5.8	Exercises for Section 8.5 . . . . .	541
8.6	A Simple Code Generator . . . . .	542
8.6.1	Register and Address Descriptors . . . . .	543
8.6.2	The Code-Generation Algorithm . . . . .	544

8.6.3	Design of the Function <i>getReg</i> . . . . .	547
8.6.4	Exercises for Section 8.6 . . . . .	548
8.7	Peephole Optimization . . . . .	549
8.7.1	Eliminating Redundant Loads and Stores . . . . .	550
8.7.2	Eliminating Unreachable Code . . . . .	550
8.7.3	Flow-of-Control Optimizations . . . . .	551
8.7.4	Algebraic Simplification and Reduction in Strength . . . . .	552
8.7.5	Use of Machine Idioms . . . . .	552
8.7.6	Exercises for Section 8.7 . . . . .	553
8.8	Register Allocation and Assignment . . . . .	553
8.8.1	Global Register Allocation . . . . .	553
8.8.2	Usage Counts . . . . .	554
8.8.3	Register Assignment for Outer Loops . . . . .	556
8.8.4	Register Allocation by Graph Coloring . . . . .	556
8.8.5	Exercises for Section 8.8 . . . . .	557
8.9	Instruction Selection by Tree Rewriting . . . . .	558
8.9.1	Tree-Translation Schemes . . . . .	558
8.9.2	Code Generation by Tiling an Input Tree . . . . .	560
8.9.3	Pattern Matching by Parsing . . . . .	563
8.9.4	Routines for Semantic Checking . . . . .	565
8.9.5	General Tree Matching . . . . .	565
8.9.6	Exercises for Section 8.9 . . . . .	567
8.10	Optimal Code Generation for Expressions . . . . .	567
8.10.1	Ershov Numbers . . . . .	567
8.10.2	Generating Code From Labeled Expression Trees . . . . .	568
8.10.3	Evaluating Expressions with an Insufficient Supply of Reg- isters . . . . .	570
8.10.4	Exercises for Section 8.10 . . . . .	572
8.11	Dynamic Programming Code-Generation . . . . .	573
8.11.1	Contiguous Evaluation . . . . .	574
8.11.2	The Dynamic Programming Algorithm . . . . .	575
8.11.3	Exercises for Section 8.11 . . . . .	577
8.12	Summary of Chapter 8 . . . . .	578
8.13	References for Chapter 8 . . . . .	579
<b>9</b>	<b>Machine-Independent Optimizations</b> . . . . .	<b>583</b>
9.1	The Principal Sources of Optimization . . . . .	584
9.1.1	Causes of Redundancy . . . . .	584
9.1.2	A Running Example: Quicksort . . . . .	585
9.1.3	Semantics-Preserving Transformations . . . . .	586
9.1.4	Global Common Subexpressions . . . . .	588
9.1.5	Copy Propagation . . . . .	590
9.1.6	Dead-Code Elimination . . . . .	591
9.1.7	Code Motion . . . . .	592
9.1.8	Induction Variables and Reduction in Strength . . . . .	592

9.1.9	Exercises for Section 9.1 . . . . .	596
9.2	Introduction to Data-Flow Analysis . . . . .	597
9.2.1	The Data-Flow Abstraction . . . . .	597
9.2.2	The Data-Flow Analysis Schema . . . . .	599
9.2.3	Data-Flow Schemas on Basic Blocks . . . . .	600
9.2.4	Reaching Definitions . . . . .	601
9.2.5	Live-Variable Analysis . . . . .	608
9.2.6	Available Expressions . . . . .	610
9.2.7	Summary . . . . .	614
9.2.8	Exercises for Section 9.2 . . . . .	615
9.3	Foundations of Data-Flow Analysis . . . . .	618
9.3.1	Semilattices . . . . .	618
9.3.2	Transfer Functions . . . . .	623
9.3.3	The Iterative Algorithm for General Frameworks . . . . .	626
9.3.4	Meaning of a Data-Flow Solution . . . . .	628
9.3.5	Exercises for Section 9.3 . . . . .	631
9.4	Constant Propagation . . . . .	632
9.4.1	Data-Flow Values for the Constant-Propagation Framework . . . . .	633
9.4.2	The Meet for the Constant-Propagation Framework . . . . .	633
9.4.3	Transfer Functions for the Constant-Propagation Framework . . . . .	634
9.4.4	Monotonicity of the Constant-Propagation Framework . . . . .	635
9.4.5	Nondistributivity of the Constant-Propagation Framework . . . . .	635
9.4.6	Interpretation of the Results . . . . .	637
9.4.7	Exercises for Section 9.4 . . . . .	637
9.5	Partial-Redundancy Elimination . . . . .	639
9.5.1	The Sources of Redundancy . . . . .	639
9.5.2	Can All Redundancy Be Eliminated? . . . . .	642
9.5.3	The Lazy-Code-Motion Problem . . . . .	644
9.5.4	Anticipation of Expressions . . . . .	645
9.5.5	The Lazy-Code-Motion Algorithm . . . . .	646
9.5.6	Exercises for Section 9.5 . . . . .	655
9.6	Loops in Flow Graphs . . . . .	655
9.6.1	Dominators . . . . .	656
9.6.2	Depth-First Ordering . . . . .	660
9.6.3	Edges in a Depth-First Spanning Tree . . . . .	661
9.6.4	Back Edges and Reducibility . . . . .	662
9.6.5	Depth of a Flow Graph . . . . .	665
9.6.6	Natural Loops . . . . .	665
9.6.7	Speed of Convergence of Iterative Data-Flow Algorithms . . . . .	667
9.6.8	Exercises for Section 9.6 . . . . .	669
9.7	Region-Based Analysis . . . . .	672
9.7.1	Regions . . . . .	672
9.7.2	Region Hierarchies for Reducible Flow Graphs . . . . .	673

9.7.3	Overview of a Region-Based Analysis . . . . .	676
9.7.4	Necessary Assumptions About Transfer Functions . . . . .	678
9.7.5	An Algorithm for Region-Based Analysis . . . . .	680
9.7.6	Handling Nonreducible Flow Graphs . . . . .	684
9.7.7	Exercises for Section 9.7 . . . . .	686
9.8	Symbolic Analysis . . . . .	686
9.8.1	Affine Expressions of Reference Variables . . . . .	687
9.8.2	Data-Flow Problem Formulation . . . . .	689
9.8.3	Region-Based Symbolic Analysis . . . . .	694
9.8.4	Exercises for Section 9.8 . . . . .	699
9.9	Summary of Chapter 9 . . . . .	700
9.10	References for Chapter 9 . . . . .	703
<b>10</b>	<b>Instruction-Level Parallelism</b> . . . . .	<b>707</b>
10.1	Processor Architectures . . . . .	708
10.1.1	Instruction Pipelines and Branch Delays . . . . .	708
10.1.2	Pipelined Execution . . . . .	709
10.1.3	Multiple Instruction Issue . . . . .	710
10.2	Code-Scheduling Constraints . . . . .	710
10.2.1	Data Dependence . . . . .	711
10.2.2	Finding Dependences Among Memory Accesses . . . . .	712
10.2.3	Tradeoff Between Register Usage and Parallelism . . . . .	713
10.2.4	Phase Ordering Between Register Allocation and Code Scheduling . . . . .	716
10.2.5	Control Dependence . . . . .	716
10.2.6	Speculative Execution Support . . . . .	717
10.2.7	A Basic Machine Model . . . . .	719
10.2.8	Exercises for Section 10.2 . . . . .	720
10.3	Basic-Block Scheduling . . . . .	721
10.3.1	Data-Dependence Graphs . . . . .	722
10.3.2	List Scheduling of Basic Blocks . . . . .	723
10.3.3	Prioritized Topological Orders . . . . .	725
10.3.4	Exercises for Section 10.3 . . . . .	726
10.4	Global Code Scheduling . . . . .	727
10.4.1	Primitive Code Motion . . . . .	728
10.4.2	Upward Code Motion . . . . .	730
10.4.3	Downward Code Motion . . . . .	731
10.4.4	Updating Data Dependences . . . . .	732
10.4.5	Global Scheduling Algorithms . . . . .	732
10.4.6	Advanced Code Motion Techniques . . . . .	736
10.4.7	Interaction with Dynamic Schedulers . . . . .	737
10.4.8	Exercises for Section 10.4 . . . . .	737
10.5	Software Pipelining . . . . .	738
10.5.1	Introduction . . . . .	738
10.5.2	Software Pipelining of Loops . . . . .	740

10.5.3	Register Allocation and Code Generation . . . . .	743
10.5.4	Do-Across Loops . . . . .	743
10.5.5	Goals and Constraints of Software Pipelining . . . . .	745
10.5.6	A Software-Pipelining Algorithm . . . . .	749
10.5.7	Scheduling Acyclic Data-Dependence Graphs . . . . .	749
10.5.8	Scheduling Cyclic Dependence Graphs . . . . .	751
10.5.9	Improvements to the Pipelining Algorithms . . . . .	758
10.5.10	Modular Variable Expansion . . . . .	758
10.5.11	Conditional Statements . . . . .	761
10.5.12	Hardware Support for Software Pipelining . . . . .	762
10.5.13	Exercises for Section 10.5 . . . . .	763
10.6	Summary of Chapter 10 . . . . .	765
10.7	References for Chapter 10 . . . . .	766
<b>11</b>	<b>Optimizing for Parallelism and Locality</b>	<b>769</b>
11.1	Basic Concepts . . . . .	771
11.1.1	Multiprocessors . . . . .	772
11.1.2	Parallelism in Applications . . . . .	773
11.1.3	Loop-Level Parallelism . . . . .	775
11.1.4	Data Locality . . . . .	777
11.1.5	Introduction to Affine Transform Theory . . . . .	778
11.2	Matrix Multiply: An In-Depth Example . . . . .	782
11.2.1	The Matrix-Multiplication Algorithm . . . . .	782
11.2.2	Optimizations . . . . .	785
11.2.3	Cache Interference . . . . .	788
11.2.4	Exercises for Section 11.2 . . . . .	788
11.3	Iteration Spaces . . . . .	788
11.3.1	Constructing Iteration Spaces from Loop Nests . . . . .	788
11.3.2	Execution Order for Loop Nests . . . . .	791
11.3.3	Matrix Formulation of Inequalities . . . . .	791
11.3.4	Incorporating Symbolic Constants . . . . .	793
11.3.5	Controlling the Order of Execution . . . . .	793
11.3.6	Changing Axes . . . . .	798
11.3.7	Exercises for Section 11.3 . . . . .	799
11.4	Affine Array Indexes . . . . .	801
11.4.1	Affine Accesses . . . . .	802
11.4.2	Affine and Nonaffine Accesses in Practice . . . . .	803
11.4.3	Exercises for Section 11.4 . . . . .	804
11.5	Data Reuse . . . . .	804
11.5.1	Types of Reuse . . . . .	805
11.5.2	Self Reuse . . . . .	806
11.5.3	Self-Spatial Reuse . . . . .	809
11.5.4	Group Reuse . . . . .	811
11.5.5	Exercises for Section 11.5 . . . . .	814
11.6	Array Data-Dependence Analysis . . . . .	815

11.6.1	Definition of Data Dependence of Array Accesses . . . . .	816
11.6.2	Integer Linear Programming . . . . .	817
11.6.3	The GCD Test . . . . .	818
11.6.4	Heuristics for Solving Integer Linear Programs . . . . .	820
11.6.5	Solving General Integer Linear Programs . . . . .	823
11.6.6	Summary . . . . .	825
11.6.7	Exercises for Section 11.6 . . . . .	826
11.7	Finding Synchronization-Free Parallelism . . . . .	828
11.7.1	An Introductory Example . . . . .	828
11.7.2	Affine Space Partitions . . . . .	830
11.7.3	Space-Partition Constraints . . . . .	831
11.7.4	Solving Space-Partition Constraints . . . . .	835
11.7.5	A Simple Code-Generation Algorithm . . . . .	838
11.7.6	Eliminating Empty Iterations . . . . .	841
11.7.7	Eliminating Tests from Innermost Loops . . . . .	844
11.7.8	Source-Code Transforms . . . . .	846
11.7.9	Exercises for Section 11.7 . . . . .	851
11.8	Synchronization Between Parallel Loops . . . . .	853
11.8.1	A Constant Number of Synchronizations . . . . .	853
11.8.2	Program-Dependence Graphs . . . . .	854
11.8.3	Hierarchical Time . . . . .	857
11.8.4	The Parallelization Algorithm . . . . .	859
11.8.5	Exercises for Section 11.8 . . . . .	860
11.9	Pipelining . . . . .	861
11.9.1	What is Pipelining? . . . . .	861
11.9.2	Successive Over-Relaxation (SOR): An Example . . . . .	863
11.9.3	Fully Permutable Loops . . . . .	864
11.9.4	Pipelining Fully Permutable Loops . . . . .	864
11.9.5	General Theory . . . . .	867
11.9.6	Time-Partition Constraints . . . . .	868
11.9.7	Solving Time-Partition Constraints by Farkas' Lemma . . . . .	872
11.9.8	Code Transformations . . . . .	875
11.9.9	Parallelism With Minimum Synchronization . . . . .	880
11.9.10	Exercises for Section 11.9 . . . . .	882
11.10	Locality Optimizations . . . . .	884
11.10.1	Temporal Locality of Computed Data . . . . .	885
11.10.2	Array Contraction . . . . .	885
11.10.3	Partition Interleaving . . . . .	887
11.10.4	Putting it All Together . . . . .	890
11.10.5	Exercises for Section 11.10 . . . . .	892
11.11	Other Uses of Affine Transforms . . . . .	893
11.11.1	Distributed memory machines . . . . .	894
11.11.2	Multi-Instruction-Issue Processors . . . . .	895
11.11.3	Vector and SIMD Instructions . . . . .	895
11.11.4	Prefetching . . . . .	896

11.12	Summary of Chapter 11 . . . . .	897
11.13	References for Chapter 11 . . . . .	899
<b>12</b>	<b>Interprocedural Analysis</b>	<b>903</b>
12.1	Basic Concepts . . . . .	904
12.1.1	Call Graphs . . . . .	904
12.1.2	Context Sensitivity . . . . .	906
12.1.3	Call Strings . . . . .	908
12.1.4	Cloning-Based Context-Sensitive Analysis . . . . .	910
12.1.5	Summary-Based Context-Sensitive Analysis . . . . .	911
12.1.6	Exercises for Section 12.1 . . . . .	914
12.2	Why Interprocedural Analysis? . . . . .	916
12.2.1	Virtual Method Invocation . . . . .	916
12.2.2	Pointer Alias Analysis . . . . .	917
12.2.3	Parallelization . . . . .	917
12.2.4	Detection of Software Errors and Vulnerabilities . . . . .	917
12.2.5	SQL Injection . . . . .	918
12.2.6	Buffer Overflow . . . . .	920
12.3	A Logical Representation of Data Flow . . . . .	921
12.3.1	Introduction to Datalog . . . . .	921
12.3.2	Datalog Rules . . . . .	922
12.3.3	Intensional and Extensional Predicates . . . . .	924
12.3.4	Execution of Datalog Programs . . . . .	927
12.3.5	Incremental Evaluation of Datalog Programs . . . . .	928
12.3.6	Problematic Datalog Rules . . . . .	930
12.3.7	Exercises for Section 12.3 . . . . .	932
12.4	A Simple Pointer-Analysis Algorithm . . . . .	933
12.4.1	Why is Pointer Analysis Difficult . . . . .	934
12.4.2	A Model for Pointers and References . . . . .	935
12.4.3	Flow Insensitivity . . . . .	936
12.4.4	The Formulation in Datalog . . . . .	937
12.4.5	Using Type Information . . . . .	938
12.4.6	Exercises for Section 12.4 . . . . .	939
12.5	Context-Insensitive Interprocedural Analysis . . . . .	941
12.5.1	Effects of a Method Invocation . . . . .	941
12.5.2	Call Graph Discovery in Datalog . . . . .	943
12.5.3	Dynamic Loading and Reflection . . . . .	944
12.5.4	Exercises for Section 12.5 . . . . .	945
12.6	Context-Sensitive Pointer Analysis . . . . .	945
12.6.1	Contexts and Call Strings . . . . .	946
12.6.2	Adding Context to Datalog Rules . . . . .	949
12.6.3	Additional Observations About Sensitivity . . . . .	949
12.6.4	Exercises for Section 12.6 . . . . .	950
12.7	Datalog Implementation by BDD's . . . . .	951
12.7.1	Binary Decision Diagrams . . . . .	951

12.7.2	Transformations on BDD's . . . . .	953
12.7.3	Representing Relations by BDD's . . . . .	954
12.7.4	Relational Operations as BDD Operations . . . . .	954
12.7.5	Using BDD's for Points-to Analysis . . . . .	957
12.7.6	Exercises for Section 12.7 . . . . .	958
12.8	Summary of Chapter 12 . . . . .	958
12.9	References for Chapter 12 . . . . .	961
<b>A</b>	<b>A Complete Front End</b>	<b>965</b>
A.1	The Source Language . . . . .	965
A.2	Main . . . . .	966
A.3	Lexical Analyzer . . . . .	967
A.4	Symbol Tables and Types . . . . .	970
A.5	Intermediate Code for Expressions . . . . .	971
A.6	Jumping Code for Boolean Expressions . . . . .	974
A.7	Intermediate Code for Statements . . . . .	978
A.8	Parser . . . . .	981
A.9	Creating the Front End . . . . .	986
<b>B</b>	<b>Finding Linearly Independent Solutions</b>	<b>989</b>
	<b>Index</b>	<b>993</b>