

## Table of Contents

<b>Chapter 1. Main Principles of Program Specialization</b> . . . . .	1
1.1. Specialized program . . . . .	2
1.1.1. Program specialization . . . . .	2
1.1.2. Context of specialization . . . . .	6
1.1.3. Specialization of a fragment of program . . . . .	8
1.1.4. Partial computations . . . . .	9
1.1.5. Range of specializations . . . . .	11
1.1.6. Equivalence between the specialized program and the generic program . . . . .	15
1.2. Specializing to improve performance . . . . .	16
1.2.1. Execution time . . . . .	17
1.2.2. Memory space . . . . .	17
1.2.3. Effect of the compiler . . . . .	18
1.2.4. Opacity of the code generated . . . . .	20
1.2.5. Effect of the memory cache . . . . .	21
1.3. Automatic specialization . . . . .	22
1.3.1. Specializer . . . . .	22
1.3.2. Operation of specialization . . . . .	24
1.3.3. Execution times . . . . .	25
1.3.4. Advantages and disadvantages to automatic specialization . . . . .	25
1.4. Main applications of specialization . . . . .	27
1.4.1. Application 1: compiling using an interpreter . . . . .	27
1.4.2. Application 2: transforming an interpreter into a compiler . . . . .	30
1.4.3. Application 3: creating a compiler generator . . . . .	31
1.5. Specialization times . . . . .	33
1.5.1. Compile-time specialization . . . . .	33
1.5.2. Runtime specialization . . . . .	34
1.5.3. Specialization server . . . . .	35
1.5.4. Specialized code cache . . . . .	36

1.6. Financial viability of specialization . . . . .	37
1.6.1. Specialization gain . . . . .	38
1.6.2. Specialization time . . . . .	39
1.6.3. Size of the specializer . . . . .	40
1.6.4. Specialization before execution . . . . .	40
1.6.5. Runtime specialization and break-even point . . . . .	42
<b>Chapter 2. Specialization Techniques . . . . .</b>	<b>43</b>
2.1. Transforming specialization programs . . . . .	44
2.1.1. Partial evaluation . . . . .	44
2.1.2. Specialization strategies . . . . .	44
2.1.3. Formulation of specialization using general transformations . . . . .	45
2.1.4. Formulation of specialization using ad hoc transformations . . . . .	48
2.1.5. Techniques for executing precomputations . . . . .	50
2.1.6. Speculative specialization . . . . .	51
2.1.7. Interprocedural specialization . . . . .	55
2.1.8. Polyvariant specialization . . . . .	56
2.2. Termination of specialization . . . . .	57
2.2.1. Online control . . . . .	58
2.2.2. Offline control . . . . .	59
2.3. Correctness of specialization . . . . .	60
2.3.1. Soundness, completeness and correctness . . . . .	60
2.3.2. Remedying laziness . . . . .	61
2.3.3. Execution error handling . . . . .	62
2.3.4. Portability . . . . .	63
2.3.5. Pre-processor . . . . .	64
2.4. Other forms of specialization . . . . .	65
2.4.1. Driving and supercompilation . . . . .	65
2.4.2. Generalized partial computation . . . . .	66
2.4.3. Configurable partial computation . . . . .	66
2.4.4. Program slicing . . . . .	67
2.4.5. Comparison with a compiler . . . . .	67
2.4.6. Comparison with a multilevel language . . . . .	68
<b>Chapter 3. Offline Specialization . . . . .</b>	<b>71</b>
3.1. Main principles of offline specialization . . . . .	72
3.1.1. Specification of input binding times . . . . .	72
3.1.2. Binding-time analysis . . . . .	74
3.1.3. Specialization by binding-time interpretation . . . . .	78
3.1.4. Action analysis . . . . .	79
3.1.5. Specialization by action interpretation . . . . .	81
3.1.6. Generating extension . . . . .	82
3.1.7. Compiler generator . . . . .	84

3.1.8. Generation of a specialized program . . . . .	85
3.1.9. Offline specializer . . . . .	88
3.1.10. Correction of offline specialization . . . . .	89
3.1.11. Specialization grammar . . . . .	89
3.1.12. Polyvariant offline specialization . . . . .	91
3.2. Compared advantages of offline specialization . . . . .	92
3.2.1. Evaluation <i>a priori</i> of the specialization degree . . . . .	92
3.2.2. Visualization of specialization information . . . . .	93
3.2.3. Declaration of expected binding times . . . . .	94
3.2.4. Specialization debugging . . . . .	95
3.2.5. Improvement of binding times . . . . .	95
3.2.6. Specialization speed . . . . .	96
3.2.7. Specialization time . . . . .	97
3.2.8. Task and expertise distribution . . . . .	97
3.2.9. Intellectual property . . . . .	98
3.2.10. Limits of offline specialization . . . . .	98
3.3. Main components of binding-time analysis . . . . .	99
3.3.1. Definition and use of memory locations . . . . .	100
3.3.2. Standard binding times and composition operations . . . . .	102
3.3.3. Static-and-dynamic binding time . . . . .	103
3.3.4. Undefined values and dead code . . . . .	104
3.3.5. Preliminary alias analysis requirement . . . . .	107
3.3.6. Formal definition and analysis implementation . . . . .	108
3.4. When static inputs become dynamic . . . . .	109
3.4.1. Initial and actual binding times . . . . .	109
3.4.2. Preservation of specialization interfaces . . . . .	110
3.4.3. Program specialization with revision of the static inputs . . . . .	113
<b>Chapter 4. A Specializer for C: Tempo . . . . .</b>	<b>117</b>
4.1. History . . . . .	118
4.1.1. Origins . . . . .	118
4.1.2. The “Tempo project” . . . . .	119
4.1.3. The tool Tempo . . . . .	120
4.2. Disruptive technologies . . . . .	121
4.2.1. Revision of specialization principles . . . . .	121
4.2.2. Revision of specialization analyses . . . . .	121
4.2.3. Revision of specialization transformations . . . . .	122
4.3. Architecture . . . . .	123
4.3.1. Preprocessing . . . . .	123
4.3.2. Processing . . . . .	128
4.3.3. Post-processing . . . . .	131
4.3.4. Interface . . . . .	131
4.4. Engineering economics . . . . .	132

4.4.1. Pragmatics of knowledge . . . . .	133
4.4.2. Language construction coverage . . . . .	133
4.4.3. The case of function pointers . . . . .	135
4.5. Beyond Tempo . . . . .	139
4.5.1. Certified runtime specialization . . . . .	140
4.5.2. Java program specialization . . . . .	140
4.5.3. C++ program specialization . . . . .	141
4.5.4. Specialization declaration and verification . . . . .	141
4.6. Other specializers for the C language . . . . .	142
4.6.1. C-Mix . . . . .	142
4.6.2. DyC . . . . .	143
<b>Chapter 5. Applications of Specialization . . . . .</b>	<b>145</b>
5.1. Applications in operating systems and networks . . . . .	146
5.1.1. Sun's RPC . . . . .	148
5.1.2. BSD Packet Filter . . . . .	155
5.1.3. Unix signals . . . . .	156
5.1.4. Chorus IPC . . . . .	157
5.1.5. Summary . . . . .	158
5.2. Applications to numerical computation . . . . .	159
5.3. Applications to compilation using an interpreter . . . . .	160
5.4. Applications to the optimization of software architectures . . . . .	164
5.4.1. Issue of flexibility . . . . .	165
5.4.2. Sources of inefficiency in the implementation of software architectures . . . . .	166
5.4.3. Improving efficiency while preserving flexibility . . . . .	169
5.4.4. Some case studies . . . . .	169
5.4.5. Framework of application of specialization . . . . .	174
5.4.6. Other approaches to optimizing software architectures . . . . .	176
5.4.7. Program specialization to optimize software architectures . . . . .	178
5.5. Specialization as a software engineering tool . . . . .	180
5.5.1. High-level optimizer . . . . .	180
5.5.2. Think generic . . . . .	181
5.5.3. Predefined adaptable components . . . . .	183
5.5.4. Other uses of specialization in software engineering . . . . .	183
<b>Chapter 6. Precision of Program Analysis . . . . .</b>	<b>185</b>
6.1. Choosing the precision of an analysis . . . . .	186
6.1.1. Degrees of freedom . . . . .	186
6.1.2. Too much of a good thing . . . . .	187
6.1.3. Targeting a program class . . . . .	188
6.1.4. Analysis combination . . . . .	188
6.1.5. Different analysis sensitivities . . . . .	189

6.2. Sensitivity to (control) flow . . . . .	189
6.2.1. Concrete specialization requirements . . . . .	192
6.3. Sensitivity to speculative evaluation . . . . .	193
6.3.1. Concrete specialization requirements . . . . .	193
6.4. Sensitivity to data structure components . . . . .	194
6.4.1. Concrete specialization requirements . . . . .	196
6.5. Sensitivity to data structure instances . . . . .	196
6.5.1. Concrete specialization requirements . . . . .	200
6.6. Sensitivity to use (of memory locations) . . . . .	201
6.6.1. Mapping of definitions and uses . . . . .	201
6.6.2. Static-and-dynamic binding time . . . . .	202
6.6.3. Sensitivity to the dynamic use of memory locations . . . . .	203
6.6.4. Case of non-reifiable values . . . . .	204
6.6.5. Sensitivity to the static use of memory locations . . . . .	206
6.6.6. Sensitivity to the use of memory locations . . . . .	207
6.7. Sensitivity to use of literal constants . . . . .	208
6.7.1. Concrete specialization requirements . . . . .	210
6.8. Intraprocedural versus interprocedural analysis . . . . .	211
6.8.1. Concrete specialization requirements . . . . .	212
6.9. Sensitivity to the context (of function call) . . . . .	213
6.9.1. Concrete specialization requirements . . . . .	214
6.10. Sensitivity to the return value . . . . .	214
6.10.1. Concrete specialization requirements . . . . .	215
6.11. Other precision forms . . . . .	216
6.11.1. Sensitivity to the execution context of code templates . . . . .	216
6.11.2. Sensitivity to continuations . . . . .	217
6.12. Precision of the existing C specializers . . . . .	217
6.12.1. Precision of Tempo . . . . .	218
6.12.2. Precision of C-Mix . . . . .	219
6.12.3. Precision of DyC . . . . .	220
<b>Chapter 7. Reification: From a Value to a Term . . . . .</b>	<b>221</b>
7.1. Different types of reification . . . . .	222
7.1.1. Direct literal reification . . . . .	222
7.1.2. Latent literal lifting . . . . .	223
7.1.3. Indirect literal lifting . . . . .	223
7.1.4. Computable lifting . . . . .	223
7.1.5. Complete and partial lifting . . . . .	224
7.1.6. Incremental lifting . . . . .	224
7.1.7. Memory zone of lifting . . . . .	225
7.1.8. Optimized lifting . . . . .	225
7.1.9. Instrumented lifting . . . . .	226
7.2. Constraints of lifting . . . . .	226

7.2.1. Reflexiveness constraints . . . . .	226
7.2.2. Syntactic constraints . . . . .	227
7.2.3. Semantic constraints . . . . .	228
7.2.4. Influence of the moment of specialization . . . . .	229
7.2.5. Efficiency constraints . . . . .	230
7.2.6. Decision to lift and processing of non-liftable values . . . . .	230
7.3. Lifting of immutable data . . . . .	231
7.3.1. Lifting of an elementary piece of data . . . . .	231
7.3.2. Lifting of an immutable composite piece of data . . . . .	232
7.4. Lifting of a non-shared mutable piece of data . . . . .	234
7.4.1. Lifting of a non-shared table . . . . .	234
7.4.2. Reification of a structure . . . . .	236
7.5. Reification of a shared mutable piece of data . . . . .	237
7.6. Reification of a reference . . . . .	238
7.6.1. Reference and memory address . . . . .	238
7.6.2. Symbolic entities, dynamic data, and visibility . . . . .	239
7.6.3. From a pointer to a symbol . . . . .	239
7.6.4. Type-based reification of a reference . . . . .	240
7.6.5. Offset-based reification of a pointer . . . . .	241
7.6.6. Profitability of pointer reification . . . . .	242
7.7. Physical data sharing between execution times . . . . .	243
7.7.1. Lifespan . . . . .	243
7.7.2. Controlled access . . . . .	244
7.8. Reification and binding time . . . . .	245
7.8.1. Dynamic treatment of non-reifiable values . . . . .	245
7.8.2. Superfluous copy elimination . . . . .	245
<b>Chapter 8. Specialization of Incomplete Programs . . . . .</b>	<b>249</b>
8.1. Constraints on the code to be specialized . . . . .	250
8.1.1. The “outside” of a program . . . . .	250
8.1.2. Availability of the code and resource sharing . . . . .	251
8.1.3. The specialization profitability in a development process . . . . .	252
8.1.4. Complex scaling of specialization techniques . . . . .	252
8.1.5. Software component specialization . . . . .	252
8.1.6. Modular and separated specialization . . . . .	253
8.2. Specialization module and language module . . . . .	254
8.2.1. Specialization module . . . . .	254
8.2.2. Modularity in C . . . . .	255
8.2.3. Modularity in Tempo . . . . .	255
8.3. Revision of the expression of specialization . . . . .	256
8.3.1. Componential semantics . . . . .	256
8.3.2. Interactions of an incomplete program . . . . .	258
8.3.3. Observability and equivalence of incomplete programs . . . . .	259

8.3.4. Observability and specialization . . . . .	260
8.3.5. Incomplete program specialization and binding times . . . . .	262
8.4. Calling context of a function to be specialized . . . . .	264
8.4.1. Calling context and specialization . . . . .	265
8.4.2. Modeling of a calling context . . . . .	266
8.5. Effect of external function calls . . . . .	266
8.5.1. External functions and specialization . . . . .	266
8.5.2. Modeling of an external function . . . . .	268
8.6. Abstract modeling languages . . . . .	269
8.6.1. Existing modeling languages . . . . .	269
8.6.2. Advantages and drawbacks . . . . .	271
8.7. Concrete modeling . . . . .	272
8.7.1. Concrete models . . . . .	272
8.7.2. Concrete calling contexts . . . . .	274
8.7.3. Concrete calling effects . . . . .	275
8.7.4. Advantages and drawbacks . . . . .	278
8.7.5. Experiment with concrete models . . . . .	280
<b>Chapter 9. Exploitation of Specialization . . . . .</b>	<b>283</b>
9.1. Means of exploiting specialization . . . . .	284
9.1.1. Specialization of programs versus specialization of subprograms . . . . .	284
9.1.2. Correctly exploiting specialization . . . . .	285
9.1.3. Knowing the input values . . . . .	286
9.2. Invariant execution context . . . . .	286
9.2.1. Fixed exploitation of a specialized program . . . . .	287
9.2.2. Fixed exploitation of a specialized function . . . . .	287
9.2.3. Fixed exploitation of runtime specialization . . . . .	287
9.3. Optimistic specialization . . . . .	288
9.3.1. Case-based specialization of a function call . . . . .	288
9.3.2. Moments of optimistic specialization . . . . .	289
9.3.3. Profitability of optimistic specialization . . . . .	290
9.3.4. Specialized function selection . . . . .	290
9.3.5. Explicit optimistic specialization . . . . .	291
9.3.6. Implicit optimistic specialization: The Trick . . . . .	292
9.3.7. Comparison of explicit and implicit optimistic specializations . . . . .	294
9.4. Selection by necessity of a specialized function . . . . .	294
9.4.1. Case-based specialization of a function . . . . .	295
9.4.2. Centralized selection by necessity . . . . .	295
9.4.3. Selection by necessity at the call site . . . . .	296
9.4.4. Inlining of selection-by-necessity functions . . . . .	297

9.5. Selection by anticipation of a specialized function . . . . .	298
9.5.1. General idea . . . . .	298
9.5.2. Specialized-call variable and indirect call . . . . .	300
9.5.3. Adaptation to different conformations of specialization . . . . .	301
9.5.4. Generic case by default . . . . .	301
9.5.5. Modification of a determined context . . . . .	303
9.5.6. Guards and complex execution contexts . . . . .	303
9.5.7. Placing of guards . . . . .	305
9.5.8. For or against selection by anticipation . . . . .	306
<b>Chapter 10. Incremental Runtime Specialization . . . . .</b>	<b>309</b>
10.1. Data availability staging . . . . .	310
10.1.1. Staging and program specialization . . . . .	311
10.1.2. Staging in program loops . . . . .	311
10.1.3. Advantages of incremental specialization . . . . .	312
10.2. Models for incremental specialization . . . . .	313
10.2.1. Mandatory or optional incrementality . . . . .	314
10.2.2. Multiple program specialization . . . . .	314
10.2.3. Multilevel generating extension . . . . .	316
10.2.4. Multilevel compiler generator . . . . .	317
10.2.5. Bi-level iterated specialization . . . . .	317
10.2.6. Understanding the nature of incremental specialization . . . . .	319
10.2.7. Multiple self-application . . . . .	320
10.2.8. Multistage specialization . . . . .	321
10.3. Binding-time analyses for incremental specialization . . . . .	322
10.3.1. Multilevel binding-time analysis . . . . .	322
10.3.2. Iterated bi-level binding-time analysis . . . . .	322
10.3.3. Comparison of the multilevel analysis with the iterated bi-level analysis . . . . .	323
10.4. Implementation . . . . .	323
10.4.1. (bi-level) runtime specialization . . . . .	324
10.4.2. Iterated runtime specialization . . . . .	328
10.4.3. Optimized iterated specialization . . . . .	332
10.4.4. Experimental results . . . . .	333
10.5. Compared advantages of iterated specialization . . . . .	335
10.5.1. Degree of specialization . . . . .	335
10.5.2. Engineering . . . . .	338
10.6. Related works . . . . .	339
10.7. Improving incremental runtime specialization . . . . .	341
<b>Chapter 11. Data Specialization . . . . .</b>	<b>343</b>
11.1. Program specialization and loop unrolling . . . . .	344
11.1.1. Principle of (offline) program specialization . . . . .	345



11.1.2. Staging in program loops . . . . .	345
11.1.3. Manual control of the loop unrolling . . . . .	348
11.2. General concept of data specialization . . . . .	350
11.2.1. Principle of data specialization . . . . .	351
11.2.2. Example of specialization encoded in the form of data . . . . .	353
11.2.3. Loading integrated at the first execution . . . . .	358
11.2.4. Data specialization times . . . . .	358
11.2.5. Exploitation of data specialization . . . . .	359
11.3. Caching and binding time . . . . .	360
11.3.1. Selection of the static expressions to be cached . . . . .	360
11.3.2. Speculative specialization . . . . .	363
11.3.3. Loader–reader analyses and separations . . . . .	364
11.3.4. Common inputs at the loader and at the reader . . . . .	365
11.4. Structuring the cache . . . . .	365
11.4.1. Cache structured by expressions to be stored . . . . .	365
11.4.2. Cache structured by iteration . . . . .	366
11.4.3. Cache structured in homogeneous data flow . . . . .	366
11.4.4. Cache structured in flow of heterogeneous data . . . . .	368
11.4.5. Cache structured as a flow of aligned heterogeneous data . . . . .	368
11.4.6. Cache structured as a flow of compact heterogeneous data . . . . .	369
11.4.7. Dynamic management of the cache size . . . . .	370
11.5. The question of control in data specialization . . . . .	371
11.5.1. Preserved control . . . . .	371
11.5.2. Cached control . . . . .	373
11.5.3. Rebuilt control . . . . .	374
11.6. Reconstructions of control . . . . .	375
11.6.1. Reconstruction of a simple loop . . . . .	375
11.6.2. Reconstruction of nested loops . . . . .	378
11.6.3. Reconstruction and interprocedural representation . . . . .	380
11.7. Program specialization versus data specialization . . . . .	382
11.7.1. Comparison of program and data specialization . . . . .	382
11.7.2. From statement locality to data locality . . . . .	383
11.7.3. Combination of two specialization encodings . . . . .	384
11.8. Experimental results . . . . .	387
11.8.1. Integration in Tempo . . . . .	387
11.8.2. Experiments on various program types . . . . .	387
<b>Chapter 12. Scientific Perspectives . . . . .</b>	<b>393</b>
12.1. Improving the specialized code . . . . .	394
12.1.1. Improving the analyses of specialization . . . . .	394
12.1.2. Choice of online specialization among alternative offline techniques . . . . .	396
12.1.3. Partial unfolding of specialization grammars . . . . .	397

12.1.4. Post-processing for runtime specialization . . . . .	398
12.1.5. Binding-time improvement . . . . .	400
12.1.6. Better integration of program specializations and data specializations . . . . .	401
12.1.7. Choice of differed marshaling . . . . .	402
12.2. Complexity of the process of specialization . . . . .	404
12.2.1. Optimizing using a specializer . . . . .	404
12.2.2. Optimizing using a compiler . . . . .	406
12.2.3. Fine optimization with a compiler versus with a specializer . . .	407
12.3. Simplifying the process of specialization . . . . .	408
12.3.1. Automation of tasks peripheral to the specialization . . . . .	408
12.3.2. Automatically seeking and exploiting specialization opportunities . . . . .	409
12.3.3. Integration into a compiler . . . . .	413
12.3.4. Monitoring and debugging of binding times . . . . .	414
12.3.5. Binding-time improvement . . . . .	417
12.4. Integration into a software engineering process . . . . .	418
12.4.1. Integration into the software's lifecycle . . . . .	418
12.4.2. Methodology for writing specializable programs . . . . .	419
12.4.3. A specialization-oriented programming environment . . . . .	420
<b>Chapter 13. Conclusion: From Prototype to Product . . . . .</b>	<b>421</b>
13.1. The race for performance . . . . .	422
13.1.1. Pareto's law . . . . .	422
13.1.2. Proebsting's law . . . . .	423
13.2. A different viewpoint . . . . .	423
13.2.1. Specializing for a better performance . . . . .	424
13.2.2. Specialization to better produce . . . . .	424
13.3. Difficulties for investing in software engineering . . . . .	425
13.3.1. Critical thinking . . . . .	425
13.3.2. Critical path . . . . .	427
13.3.3. Critical mass . . . . .	428
13.3.4. Critical moment . . . . .	429
13.3.5. Critical situation . . . . .	429
13.4. Niche uses . . . . .	429
13.4.1. Niche applications . . . . .	430
13.4.2. Niche functionalities . . . . .	431
13.5. Developing a specialization platform . . . . .	432
13.5.1. Magnitude of the task . . . . .	432
13.5.2. The economic model . . . . .	433
13.5.3. Specializing to study . . . . .	434

<b>Appendix. Basic Facts about Languages and Programs</b> . . . . .	435
A.1. Programming languages . . . . .	436
A.1.1. Code . . . . .	436
A.1.2. Data . . . . .	439
A.1.3. Programs and subprograms . . . . .	440
A.1.4. Input . . . . .	442
A.1.5. Output . . . . .	444
A.2. Semantics . . . . .	445
A.2.1. Semantic functions . . . . .	446
A.2.2. Semantic framework . . . . .	448
A.2.3. Multiple or undefined semantics . . . . .	449
A.2.4. Non-determinism . . . . .	451
A.2.5. Under-specification . . . . .	451
A.2.6. Undefined errors . . . . .	452
A.2.7. Defined errors . . . . .	453
A.2.8. Non-termination and infinite data . . . . .	454
A.2.9. Output of an abnormal execution . . . . .	455
A.2.10. Interactions of a program and an external code . . . . .	457
A.3. Program equivalence . . . . .	458
A.3.1. Domain of definition . . . . .	458
A.3.2. Strict or lazy equivalence . . . . .	458
A.3.3. Non-termination with partial output . . . . .	460
A.3.4. Equivalence of subprograms . . . . .	460
A.4. Execution . . . . .	462
A.4.1. Execution process . . . . .	462
A.4.2. Interpretation . . . . .	465
A.4.3. Compilation . . . . .	469
A.4.4. Observation, modification and code generation . . . . .	472
A.5. Program performance . . . . .	473
A.5.1. Execution time . . . . .	474
A.5.2. Memory size . . . . .	477
A.5.3. Performance optimization . . . . .	478
A.6. Program analysis . . . . .	479
A.6.1. Abstract executions . . . . .	479
A.6.2. Flow analysis . . . . .	480
A.6.3. Result of a program analysis . . . . .	481
A.7. Program transformation . . . . .	481
A.7.1. Program transformation . . . . .	481
A.7.2. Observation and equivalence . . . . .	482
A.7.3. Soundness, completeness and correctness . . . . .	483
A.7.4. Transformation of subprograms . . . . .	484
A.7.5. Transformation and termination algorithm . . . . .	485

xvi Program Specialization

<b>Bibliography</b> . . . . .	487
<b>Index</b> . . . . .	523