

C o n t e n t s

CHAPTER 1.: INTRODUCTION

W. M. McKeeman	COMPILER CONSTRUCTION	1
	1. <i>Definitions</i>	1
	1.1. <i>Source and Target Languages</i>	1
	1.2. <i>Implementation Languages</i>	2
	1.3. <i>Language Defining Languages</i>	3
	2. <i>Recursive Descent Compilation</i>	4
	2.1. <i>Introduction</i>	4
	2.2. <i>Writing a Recursive Descent Compiler</i>	7
	2.3. <i>Executing the Compiler</i>	12
	2.4. <i>Extending the Technique</i>	14
	3. <i>Modularization</i>	15
	3.1. <i>Modular Documentation</i>	15
	3.2. <i>Modular Programmer Assignment</i>	15
	3.3. <i>Modular Source Text</i>	16
	3.4. <i>Modular Target Text</i>	16
	4. <i>Intermodular Communication</i>	16
	4.1. <i>Specification</i>	16
	4.2. <i>"Need to Know"</i>	18
	4.3. <i>Test Environment</i>	19
	4.4. <i>Feedback-Free</i>	19
	5. <i>Vertical Fragmentation</i>	20
	5.1. <i>The Transformation $PT \rightarrow AST$</i>	22
	5.2. <i>The Transformation $AST \rightarrow ST$</i>	24
	5.3. <i>The Transformation $ST \rightarrow ACT$</i>	24
	5.4. <i>The Transformation $ACT \rightarrow ADT$</i>	26
	5.5. <i>The Transformation $ADT \rightarrow SET$</i>	27
	5.6. <i>The Transformation $SET \rightarrow SCT$</i>	27
	6. <i>Horizontal Fragmentation</i>	28
	7. <i>Equivalence Transformation</i>	29
	8. <i>Evaluation</i>	34
	9. <i>References</i>	36

CHAPTER 2.: ANALYSIS

F. L. DeRemer	REVIEW OF FORMALISMS AND NOTATION	37
	1. <i>Terminology and Definitions of Grammars</i>	37
	1.1. <i>Unrestricted Rewriting Systems</i>	37
	1.2. <i>The Chomsky Hierarchy</i>	38
	1.3. <i>Phrase Structure Implied by Context-Free Grammars</i>	39
	1.4. <i>Regular Grammars and Regular Expressions</i>	41
	2. <i>Parsing</i>	42
	2.1. <i>Syntactic Dominoes</i>	42
	2.2. <i>Parsing Strategies</i>	44
	2.3. <i>Ambiguity</i>	45
	2.4. <i>Debugging a Grammar</i>	45
	3. <i>Machines, Computational Complexity, Relation to Grammars</i>	46

VI

	4. <i>Transduction Grammars</i> 46 4.1. <i>String-To-String Grammars</i> 47 4.2. <i>String-To-Tree Grammars</i> 48 5. <i>Meta-Grammars</i> 50 5.1. <i>Self-Describing Grammars</i> 50 5.2. <i>Practical Applications</i> 53 <i>References</i> 56	
M. Griffiths	LL(1) GRAMMARS AND ANALYSERS	57
	1. <i>Introduction</i> 57 1.1. <i>Predictive Analysis</i> 58 1.2. <i>Efficiency</i> 60 1.3. <i>Semantics</i> 61 2. <i>LL(1) Conditions</i> 62 3. <i>Decision Algorithm</i> 64 4. <i>Production of Analyser</i> 67 5. <i>Grammar Transformation</i> 70 5.1. <i>Elimination of Left Recursion</i> 70 5.2. <i>Factorisation and Substitution</i> 73 5.2.1. <i>Ordering</i> 74 6. <i>Semantic Insertion</i> 75 6.1. <i>Generation of Postfixed Notation</i> 76 6.2. <i>Symbol Table Insertion</i> 78 6.3. <i>Interpretation of Postfixed Expressions</i> 79 7. <i>Generalisation and Conclusion</i> 80 7.1. <i>Vienna Notation</i> 80 7.2. <i>LL(k) Grammars</i> 81 8. <i>Practical Results</i> 82 <i>References</i> 83	
J. J. Horning	LR GRAMMARS AND ANALYSERS	85
	1. <i>Intuitive Description</i> 85 1.1. <i>Definition of LR(k)</i> 86 1.2. <i>Items</i> 86 1.3. <i>States</i> 87 2. <i>Interpreting LR Tables</i> 88 2.1. <i>Form of Entries</i> 88 2.2. <i>Example</i> 90 3. <i>Constructing LR Tables</i> 93 3.1. <i>The LR(0) Constructor Algorithm</i> 93 3.1.1. <i>Initialization</i> 94 3.1.2. <i>Closure</i> 94 3.1.3. <i>Successor States</i> 94 3.1.4. <i>Accessible States</i> 94 3.1.5. <i>Deriving the Parsing Action Table</i> 95 3.2. <i>Adding Lookahead</i> 96 3.2.1. <i>Using the Follower Matrix</i> 96 3.2.2. <i>Using the Shift Entries</i> 97 3.2.3. <i>Adding Context to Items</i> 97 4. <i>Representing LR Tables</i> 98 4.1. <i>Matrix Forms</i> 98 4.2. <i>List Form</i> 98 4.3. <i>Efficiency Transformations</i> 99 4.3.1. <i>LR(0) Reduce States</i> 99 4.3.2. <i>Column Regularities</i> 100 4.3.3. <i>Row Regularities</i> 100 4.3.4. <i>Don't Cares</i> 100 4.3.5. <i>Column Reduction</i> 101	

VII

4.3.6.	<i>Row Reduction</i>	101
4.3.7.	<i>List Overlapping</i>	101
4.3.8.	<i>Matrix Factoring</i>	101
4.3.9.	<i>Eliminating Single Productions</i>	101
5.	<i>Properties of LR Grammars and Analysers</i>	102
6.	<i>Modification to Obtain LR Grammars</i>	103
6.1.	<i>Multiple-Use Separators</i>	103
6.2.	<i>Compound Terminal Symbols</i>	103
7.	<i>Comparison with other Techniques</i>	104
7.1.	<i>Grammar Inclusions</i>	104
7.2.	<i>Language Inclusions</i>	105
7.3.	<i>Error Detection</i>	106
7.4.	<i>Efficiency</i>	106
8.	<i>Choice of a Syntactic Analysis Technique</i>	106
	<i>References</i>	107

F. L. DeRemer

LEXICAL ANALYSIS 109

1.	<i>Scanning, Then Screening</i>	109
2.	<i>Screening</i>	110
3.	<i>Scanning</i>	111
3.1.	<i>Lexical Grammars</i>	111
3.1.1.	<i>Tokens</i>	112
3.1.2.	<i>A Regularity Condition</i>	112
3.1.3.	<i>Converting Regular Grammars to Regular Expressions</i>	113
3.2.	<i>Generating Scanners Via LR Tech- niques</i>	113
3.2.1.	<i>Using a Simplified LR Parser as a Scanner</i>	113
3.3.	<i>Hand-Written Scanners</i>	117
3.4.	<i>Error Recovery</i>	119
4.	<i>On Not Including "Conversion Rou- tines" in Lexical Analysers</i>	119
	<i>References</i>	120

F. L. DeRemer

TRANSFORMATIONAL GRAMMARS 121

1.	<i>Language Processing as Tree Manipulation</i>	121
1.1.	<i>Lexical and Syntactical Processing</i>	123
1.2.	<i>Standardization</i>	123
1.3.	<i>Flattening</i>	124
2.	<i>Description of Subtree Transfor- mational Grammars</i>	125
3.	<i>Compiler Structure</i>	128
4.	<i>Further Examples of Transformations</i>	129
4.1.	<i>Local Effects</i>	129
4.2.	<i>Global Effects</i>	129
4.3.	<i>Iterative Transformations</i>	131
4.4.	<i>Extension to Regular Expressions</i>	133
5.	<i>Summary and Conclusions</i>	136
6.	<i>Appendix - Meta-Grammars and PAL Grammars</i>	137
	<i>References</i>	145

VIII

C. H. A. Koster	TWO-LEVEL GRAMMARS	146
	1. Context Sensitivity	146
	1.1. On the Borderline between Syntax and Semantics	146
	2. Van Wijngaarden Grammars	148
	2.1. One-Level Van Wijngaarden Grammars	148
	2.1.1. Definition: 1VWG	148
	2.1.2. Notation	148
	2.1.3. Terminology	148
	2.1.4. Properties	149
	2.1.5. Example	149
	2.2. Two-Level Van Wijngaarden Grammars	150
	2.2.1. Definition: 2VWG	150
	2.2.2. Notation	150
	2.2.3. Terminology	150
	2.2.4. Properties	151
	2.2.5. Example: Assignment Statement	152
	2.2.6. Example: Defining and Applied Occurrences	152
	3. Conclusion	156
	References	156

W. M. Waite	SEMANTIC ANALYSIS	157
	1. Tree Traversal	158
	2. Attribute Propagation	162
	3. Operator Identification and Coercion	165
	References	168

CHAPTER 3.: SYNTHESIS

W. M. Waite	RELATIONSHIP OF LANGUAGES TO MACHINES	170
	1. Data Objects	172
	1.1. Encodings	172
	1.2. Primitive Modes	176
	1.3. Mode Conversions	181
	2. Formation Rules	182
	2.1. Expressions	182
	2.2. Names	185
	2.3. Aggregates	187
	2.4. Procedures	189
	References	193

M. Griffiths	RUN-TIME STORAGE MANAGEMENT	195
	1. Introduction	197
	2. Static Allocation	197
	3. Dynamic Allocation	198
	3.1. Block Linkage	200
	3.2. Displays	203
	3.3. Compaction of the Stack	206
	3.4. Parameter Linkage	208
	3.5. Labels	209
	4. Aggregates	210
	4.1. Arrays	210
	4.1.1. Reference by Multiplication	211

IX

4.1.2.	<i>Reference by Code Words</i>	213
4.1.3.	<i>Range Testing</i>	214
4.2.	<i>Structures</i>	215
5.	<i>Lists</i>	215
5.1.	<i>Free Lists and Garbage</i>	216
5.2.	<i>Storage Collapse</i>	217
6.	<i>Parallel Processes</i>	218
7.	<i>Conclusion</i>	220
	<i>References</i>	221

U. Hill

SPECIAL RUN-TIME ORGANIZATION TECHNIQUES FOR ALGOL 68 222

1.	<i>Introduction</i>	222
1.1.	<i>Short Outline of Data Storage Principles</i>	224
1.1.1.	<i>Fixed and Variable Parts of Objects</i>	224
1.1.2.	<i>Modes and Objects in ALGOL 68</i>	224
1.1.3.	<i>Static and Dynamic Data Storage Areas</i>	228
1.1.4.	<i>The Heap</i>	231
1.2.	<i>Generative and Interpretative Handling</i>	232
2.	<i>Special Objects in ALGOL 68</i>	233
2.1.	<i>Flexible Arrays</i>	233
2.2.	<i>Objects Generated by Slicing</i>	233
2.3.	<i>Objects Generated by Rowing</i>	234
2.4.	<i>Objects of Union Modes</i>	235
3.	<i>Scopes of Values (Life-Time)</i>	235
3.1.	<i>Definition</i>	235
3.2.	<i>Checking the Scope Conditions</i>	236
4.	<i>Scopes and Data Storage Allocation</i>	237
4.1.	<i>Scope of Routines and Allocation of Base Registers</i>	237
4.2.	<i>Storage for Data</i>	239
4.2.1.	<i>Local Objects Generated in the Block which is their Scope</i>	239
4.2.2.	<i>Local Objects Generated in an "Inner" Block</i>	240
4.2.3.	<i>Local Objects with Flexible Length</i>	241
4.2.4.	<i>Global Objects</i>	242
5.	<i>Special Topics</i>	243
5.1.	<i>Results of Blocks and Procedure Calls</i>	243
5.2.	<i>General Actual Parameters</i>	244
5.3.	<i>Local Generators</i>	245
5.4.	<i>General Mode Declarations</i>	247
5.5.	<i>"Empty" Flexible Arrays</i>	247
6.	<i>Garbage Collection</i>	248
	<i>References</i>	252

W. M. McKeeman

SYMBOL TABLE ACCESS 253

1.	<i>Introduction</i>	253
2.	<i>Operations</i>	254
3.	<i>Method of Presentation</i>	258
4.	<i>Linear Symbol Table Access</i>	259
5.	<i>Sorted Symbol Table Access</i>	265
6.	<i>Tree Symbol Table Access</i>	273
7.	<i>Hash Symbol Table Access</i>	282

7.1. Hash Functions	291
7.2. Secondary Store	295
8. Evaluation of Access Methods	296

W. M. Waite	CODE GENERATION	302
-------------	-----------------	-----

1. A Model for Code Generation	304
1.1. The Transducer	305
1.2. The Simulator	306
1.3. Common Subexpressions	307
2. Sequencing and Control	309
3. Generator Data Structures	316
3.1. Value Descriptors	319
3.2. Register Descriptors	321
4. Instruction Generation	326
4.1. Primitive Operations	327
4.2. Interpretive Coding Language	329
References	332

W. M. Waite	ASSEMBLY AND LINKAGE	333
-------------	----------------------	-----

1. A Model for Assembly	335
1.1. Object and Statement Procedures	335
1.2. Cross Referencing	339
1.3. Assembly under a Storage Constraint	342
1.4. Operand Expressions	343
2. Two-Pass Assembly	347
2.1. Relative Symbol Definition	347
2.2. Multiple Location Counters	352
3. Partial Assembly and Linkage	353
References	355

CHAPTER 4.: COMPILER-COMPILER

M. Griffiths	INTRODUCTION TO COMPILER-COMPILERS	356
--------------	------------------------------------	-----

1. Motivation	356
2. Compiler-Compilers based on Context-Free Syntax Methods	357
2.1. Syntax	358
2.2. Languages for Compiler Writing	358
2.2.1. Production Language	359
2.2.2. Machine Oriented Languages	360
3. Current Research	361
3.1. Extensible Languages	362
3.2. Formal Semantics	363
4. Conclusion	363
References	364

C. H. A. Koster	USING THE CDL COMPILER-COMPILER	366
-----------------	---------------------------------	-----

0. Introduction	366
1. Affix Grammars and CDL	367
1.1. CF Grammar as a Programming Language	367
1.2. Extending CF Grammar	369
1.2.1. Affixes	369

1.2.2.	<i>Primitive Actions and Predicates</i>	370
1.2.3.	<i>Actions</i>	371
1.2.4.	<i>Repetition</i>	372
1.2.5.	<i>Grouping</i>	373
1.2.6.	<i>Data Types</i>	374
1.3.	<i>Affix Grammars</i>	375
1.4.	<i>From Language Definition to Compiler Description</i>	375
2.	<i>The CDL Compiler-Compiler</i>	376
2.1.	<i>Extensional Mechanisms</i>	376
2.2.	<i>Realizing the CDL Machine</i>	378
2.2.1.	<i>Some Instructions</i>	378
2.2.2.	<i>Parametrization</i>	380
2.2.2.1.	<i>Parametrisation of Rules</i>	380
2.2.2.2.	<i>Parametrization of System Macros</i>	381
2.2.2.3.	<i>Parametrisation of User Macros</i>	381
2.2.2.4.	<i>Suggested Ameliorations</i>	382
2.3.	<i>Data Structures and Data Access</i>	383
2.4.	<i>The CDL Compiler-Compiler as a Tool</i>	384
2.4.1.	<i>High-Level and Low-Level Version</i>	385
2.4.2.	<i>Syntactic Debugging Aids</i>	385
2.4.3.	<i>Efficiency of the Resulting Compiler</i>	387
2.4.4.	<i>Conclusion</i>	388
3.	<i>Example: A Simple Editor</i>	389
3.1.	<i>Specification of a Simple Editor</i>	390
3.2.	<i>Environment, and Arithmetic Macros</i>	391
3.3.	<i>Table Administration Strategy</i>	392
3.4.	<i>Input and Output Primitives</i>	394
3.5.	<i>The Works</i>	395
3.6.	<i>Packing more Characters into a Word</i>	398
3.7.	<i>Various Target Languages</i>	400
4.	<i>Example 2: Symbol Table Administration</i>	402
4.1.	<i>The Environment</i>	403
4.1.1.	<i>Interface with the Machine</i>	403
4.1.2.	<i>Character Encoding</i>	404
4.1.3.	<i>The Basic Macros</i>	404
4.2.	<i>Storing Representation</i>	405
4.2.1.	<i>The repr Table</i>	405
4.2.2.	<i>Building a repr Table Element</i>	406
4.2.3.	<i>Entering an Element into the repr Table</i>	407
4.3.	<i>Output</i>	408
4.4.	<i>Input</i>	409
4.4.1.	<i>A One Character Stock</i>	409
4.4.2.	<i>Reading a Single Character</i>	410
4.4.3.	<i>Recognizing Characters</i>	410
4.4.4.	<i>Recognizing a Symbol</i>	411
4.4.5.	<i>Reading Items</i>	412
4.5.	<i>The Work Table</i>	412
4.5.1.	<i>The Block Administration</i>	413
4.5.2.	<i>Symbol Table</i>	414
4.6.	<i>Conclusion</i>	415
5.	<i>Example 3: Treatment of Declarations</i>	416
5.1.	<i>The Language to be Recognized</i>	416
5.1.1.	<i>Closed Clauses</i>	416
5.1.2.	<i>Modes</i>	417
5.1.3.	<i>Declarers</i>	418
5.2.	<i>Recognizing Closed Clauses</i>	419
5.3.	<i>Storing Modes</i>	421
5.4.	<i>Recognizing Declarers</i>	422
5.5.	<i>Conclusion</i>	425
	<i>References</i>	426

CHAPTER 5.: ENGINEERING A COMPILER

P. C. Poole	PORTABLE AND ADAPTABLE COMPILERS	427
1.	<i>Basic Concepts</i>	427
1.1.	<i>Portability and Adaptability</i>	427
1.2.	<i>Problems with Current Compilers</i>	430
1.3.	<i>Portable and Adaptable Standards</i>	437
2.	<i>Survey of Techniques</i>	439
2.1.	<i>Portability through High Level Language Coding</i>	439
2.1.1.	<i>Bootstrapping</i>	440
2.1.2.	<i>Language-Machine Interface</i>	442
2.2.	<i>Portability through Abstract Machine Modelling</i>	444
2.2.1.	<i>A Standard Abstract Machine</i>	446
2.2.2.	<i>Janus Assembly Language Formats</i>	453
2.2.3.	<i>Some Janus Examples</i>	458
2.2.4.	<i>Realising the Abstract Machine by Interpretation</i>	464
2.3.	<i>Portability through Generation</i>	466
2.4.	<i>Adaptability</i>	467
3.	<i>Case Studies</i>	470
3.1.	<i>AED</i>	470
3.2.	<i>LSD</i>	471
3.3.	<i>BCPL</i>	473
3.4.	<i>Pascal</i>	478
3.4.1.	<i>Structure of Pascal</i>	478
3.4.2.	<i>The Abstract Machine for Pascal</i>	481
3.4.3.	<i>Incorporating the Abstract Machine into the LDT</i>	488
3.4.4.	<i>Measurements</i>	491
3.5.	<i>IBM S/360 FORTRAN (G) Compiler</i>	493
	<i>References</i>	497
J. J. Horning	STRUCTURING COMPILER DEVELOPMENT	498
1.	<i>Goals of Compiler Development</i>	498
1.1.	<i>Typical Compiler Goals</i>	498
1.1.1.	<i>Correctness</i>	498
1.1.2.	<i>Availability</i>	500
1.1.3.	<i>Generality and Adaptability</i>	501
1.1.4.	<i>Helpfulness</i>	501
1.1.5.	<i>Efficiency</i>	502
1.2.	<i>The Effects of Trade-Offs</i>	502
1.2.1.	<i>Compilation Efficiency VS. Execution Efficiency</i>	503
1.2.2.	<i>Compilation Efficiency VS. Helpfulness</i>	503
1.2.3.	<i>Generality VS. Efficiency</i>	503
1.2.4.	<i>Reliability VS. Complexity</i>	504
1.2.5.	<i>Development Speed VS. Everything Else</i>	504
2.	<i>Processes in Development</i>	504
2.1.	<i>Specification</i>	504
2.2.	<i>Design</i>	505
2.3.	<i>Implementation</i>	505
2.4.	<i>Validation</i>	506
2.5.	<i>Evaluation</i>	506
2.6.	<i>Maintenance</i>	506
3.	<i>Management Tools</i>	507

3.1.	<i>Project Organization</i>	507
3.2.	<i>Information Distribution and Validation</i>	508
3.3.	<i>Programmer Motivation</i>	509
4.	<i>Technical Tools</i>	509
4.1.	<i>Compiler Compilers</i>	509
4.2.	<i>Standard Designs</i>	510
4.3.	<i>Design Methodologies</i>	510
4.4.	<i>Off-The-Shelf Components and Techniques</i>	510
4.5.	<i>Structured Programming</i>	511
4.6.	<i>Structured Programs</i>	511
4.7.	<i>Appropriate Languages</i>	511
	<i>References</i>	512
W. M. McKeeman	PROGRAMMING LANGUAGE DESIGN	514
1.	<i>Who Should (Not?) Do It?</i>	514
2.	<i>Design Principles</i>	517
3.	<i>Models for Languages</i>	519
3.1.	<i>The ALGOL Family as Models</i>	520
3.2.	<i>Mathematics as a Model</i>	523
3.3.	<i>Street Language as a Model</i>	524
	<i>References</i>	524
J. J. Horning	WHAT THE COMPILER SHOULD TELL THE USER	525
0.	<i>Introduction</i>	525
1.	<i>Normal Output</i>	527
1.1.	<i>Headings</i>	527
1.2.	<i>Listings</i>	528
1.3.	<i>Summaries</i>	530
2.	<i>Reaction to Errors</i>	532
2.1.	<i>Styles of Reaction</i>	532
2.1.1.	<i>Crash or Loop</i>	532
2.1.2.	<i>Produce Invalid Output</i>	532
2.1.3.	<i>Quit</i>	532
2.1.4.	<i>Recover and Continue Checking</i>	532
2.1.5.	<i>Repair and Continue Compilation</i>	533
2.1.6.	<i>"Correct"</i>	533
2.2.	<i>Ensuring Chosen Reactions</i>	533
2.3.	<i>Error Sources, Detectors and Sinks</i>	534
3.	<i>Syntactic Errors</i>	535
3.1.	<i>Point of Detection</i>	535
3.2.	<i>Recovery Techniques</i>	535
3.3.	<i>Systematic Recovery Strategies</i>	536
3.4.	<i>Repair Techniques</i>	537
4.	<i>Other Errors</i>	539
4.1.	<i>Lexical Errors</i>	539
4.2.	<i>Static Semantic Errors</i>	539
4.3.	<i>Dynamically Detected Errors</i>	540
4.4.	<i>Limit Failures</i>	542
4.5.	<i>Compiler Self-Checking</i>	543
5.	<i>Error Diagnosis</i>	543
5.1.	<i>Locating the Problem</i>	544
5.2.	<i>Describing the Symptom</i>	544
5.3.	<i>Suggesting Corrections</i>	544
5.4.	<i>Localisation of Error Processing</i>	545
5.5.	<i>Synthesis and Placement of Messages</i>	545
5.6.	<i>Error Logging</i>	546
5.7.	<i>Run-Time Diagnosis</i>	547
	<i>References</i>	548

W. M. Waite	OPTIMIZATION	549
	1. <i>Classification of Techniques</i>	551
	1.1. <i>Transformations</i>	552
	1.2. <i>Regions</i>	555
	1.3. <i>Efficacy</i>	561
	2. <i>Local Optimization</i>	564
	2.1. <i>Rearrangement of an expression</i>	564
	2.2. <i>Redundant Code Elimination</i>	570
	2.3. <i>Basic Blocks</i>	579
	3. <i>Global Optimization</i>	585
	3.1. <i>Redundancy and Rearrangement</i>	587
	3.2. <i>Frequency and Strength Reduction</i>	590
	3.3. <i>Global Analysis</i>	596
	<i>References</i>	600
	 <u>CHAPTER 6: APPENDIX</u>	
F. L. Bauer	HISTORICAL REMARKS ON COMPILER CONSTRUCTION	603
	1. <i>Prehistorical "Gedanken-Compilers"</i>	605
	2. <i>The First Compilers</i>	606
	3. <i>Sequentially Working Compilers</i>	609
	4. <i>"Syntax Controlled Compilers"</i>	612
	5. <i>Compilers Based on Precedence</i>	612
	6. <i>Syntax Directed Compilers</i>	613
	7. <i>Concluding Remarks</i>	614
	<i>References</i>	615
A. P. Ershov	<i>Addendum</i>	622
	<i>References</i>	625
D. Gries	ERROR RECOVERY AND CORRECTION - An Introduction to the Literature	627
	1. <i>Introduction</i>	628
	2. <i>Recovery and/or Correction</i>	628
	3. <i>Minimum Distance Errors - Theoretical Results</i>	629
	4. <i>Parsers Defined Errors</i>	629
	5. <i>General Idea behind Error Recovery</i>	630
	6. <i>Annotated References</i>	631